Application of TEXPLORE on Atari Games

Shun Zhang

***Abstract*—TEXPLORE is a sample-efficient reinforcement learning algorithm. In this report, I review work on TEXPLORE and its application to the domain of Fuel World. It shows significant improvement over traditional learning algorithms like R-MAX. I show changes made in TEXPLORE and features selected to make it work in a game domain, called Asterix. Then the literature is explored about feature subset selection and multiagent reinforcement learning.**

I. Introduction

Traditional Reinforcement Learning algorithms, like SARSA [17] and R-MAX [3], have difficulties when handling environment with a large state space. Fuel World (Figure 1) is an example of such domains. The agent, which is a car, starts from the left blue-colored area and tries to reach the goal, which is the red grid on the right. It consumes fuel in each step, and needs to visit the fuel stations if necessary, represented as green grids on the north and south boundary. For SARSA and R-MAX, it's possible to eventually learn the optimal policy. However, a learning algorithm needs to be sample-efficient to have the performance of efficiently learning the optimal policy.

TEXPLORE [6] is a more sample-efficient algorithm. Like R-MAX, it's a model-based reinforcement learning algorithm. It employs random forests to learn the transition model and the reward model. It uses UCT [9] for action selection. The Fuel World domain was used to show the advantage of this algorithm.

II. TEXPLORE

The features selected in Fuel World are quite straightforward - x coordinate of the car, y coordinate of the car, and the fuel level. For the transition model, as it uses feature extraction, we can define the mapping, $S \rightarrow F_1 \times F_2 \times \ldots \times F_n$, where S is the state space, and F_i is the space for the i-th feature. Rather than an equation, it's a mapping because it's possible that multiple states are represented as a same set of features. There would be n learning models, each of which learns $S \times A \rightarrow \Delta F_i$,[1] assuming independence between features [6]. Each model is a random forest, which contains several decision trees. The output of the random forest is the product of the prediction of each decision tree. The effects of control of moving in each direction can be easily learned in Fuel World. For example, if F_1 describes the x coordinate, then one decision tree in the transition model could be represented as Algorithm 1.

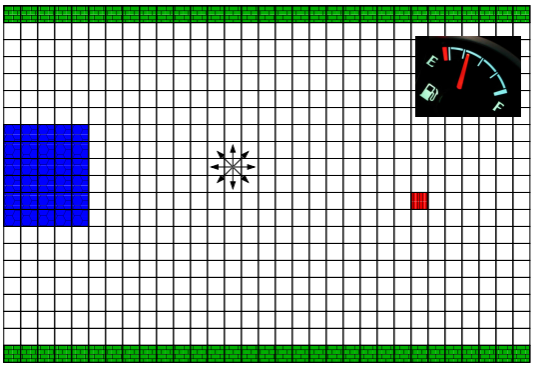

Fig. 1. The Fuel World domain [6].

Algorithm 1 A decision tree

```
if a = EAST then
    if f_1 = MAX_X then
        return 0
    else
        return 1
    end if
else
    if f_1 = 0 then
        return 0
    else
        return -1
    end if
end if
```

For the reward model, similarly, the goal is to learn $S \times A \rightarrow R$, where R is the reward space. Apart from the step cost, the cost of running out of fuel takes place when the fuel level is low. Thus it is also easy to learn.

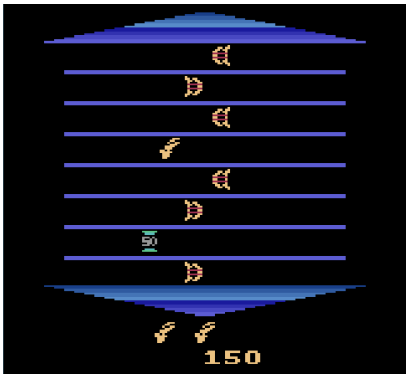

Fig. 2. Asterix Domain. The agent can move north, south, east or west. The lyres can move back and forth in east-west direction [1]. The agent should avoid the lyres, otherwise would be killed. There would be bonus objects, that the agent could get rewards if picking them up.

In this report, I show the application of TEXPLORE on Atari domain (Figure 2) and the experimental results. The ex-

[1]Strictly speaking, this should be $S_F \times A \rightarrow \Delta F_i$, where $S_F = F_1 \times F_2 \times \ldots \times F_n$. The learning algorithm is responding to the set of features, not the states themselves.

periments are designed to support the discussion in Section IV. Concretely, I show that TEXPLORE is sensitive to different sets of features.

III. Experimental Evaluation

I report on experiment results of TEXPLORE on Atari domains, and reproduce results on Fuel World as well. The action model and the sensor model employ random forests, each containing five homogeneous C4.5 [15] models. UCT planner is used for action selection, which includes exploration behavior. Also, I tried different sets of features to see how they perform differently.

A. Fuel World

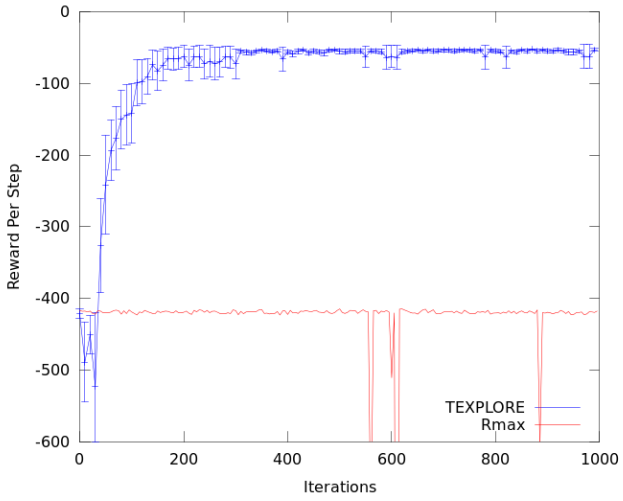

Fig. 3. Fuel World domain. Average of 50 runs. 1000 iterations in each run. Error bars represent 95% confidence interval. There is cost in each step, ranging from -1.4 to -1. There is a cost of -400 if running out of the fuel and not reaching the goal.

Figure 3 shows the results of application on Fuel World. This is the same as the experiment results in [6]. The number of iterations is increased from 300 to 1000. It reaches its convergence in around 200 iterations. R-MAX, on the other hand, didn't find the optimal path within 1000 iterations.

R-MAX even got lower reward occasionally after 500 iterations. It's possible that it figures out the position of some fuel stations, so it can live longer in the domain. But it still fails to find out the terminal state, so it perceives -400 reward in the end. Compared to other failures in which the agent fails to find a fuel station, this particular failure only makes it suffer more from step costs.

B. Asterix

Figure 4 and 5 show experiments of the application on a relaxed version of Asterix. There are only lyres wandering in the domain. They follow a deterministic trajectory - going backward and, when reaching the boundary, going forward. The agent needs to avoid them. Otherwise it'll be killed.

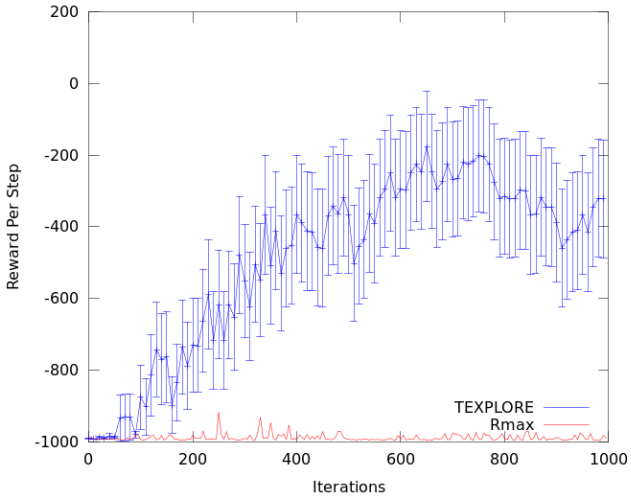

Fig. 4. Asterix Domain, with Feature Set 2. Average of 50 runs. 1000 iterations in each run. Error bars represent 95% confidence interval. The agent receives +1 reward in each step, if alive. There is a -1000 punishment if killed in 200 steps.

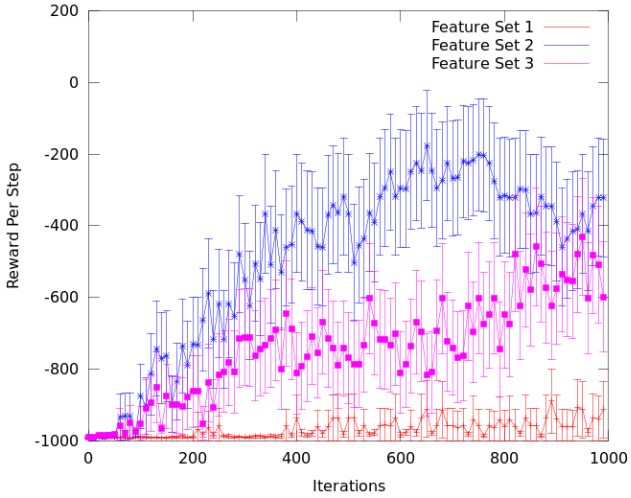

Fig. 5. Asterix Domain. Using EXPLORE with different set of features. Average of 50 runs. 1000 iterations in each run. Error bars represent 95% confidence interval. Other setup is same as Figure 4.

Figure 4 shows the comparison of TEXPLORE and R-MAX. TEXPLORE uses Feature Set 2, which is listed below. Figure 5 compares the performance of TEXPLORE on different set of features. These features are

Feature Set 1

- x-coordinate of the agent
- y-coordinate of the agent
- x-coordinate of the lyre in the first row.
- x-coordinate of the lyre in the second row.

  ...
- x-coordinate of the lyre in the n-th row.

Feature Set 2

- x-coordinate of the agent.
- difference between x-coordinates of the lyre in the row above and the agent.

- difference between x-coordinates of the lyre in the same row and the agent.
- difference between x-coordinates of the lyre in the row below and the agent.

Feature Set 3

- x-coordinate of the agent.
- the x-coordinates of the lyre in the same row.

Feature Set 1 contains the full information of the environment. It also includes redundant one, as an agent doesn't need information of the positions of lyres in more than row away from it. Feature Set 3 contains the least - it's actually the least information needed to determine the current state. It can be observed that its performance is even better than Feature Set 1, which has redundant features. In these feature sets, Feature Set 2 performs the best - it's neither too redundant, nor containing too little information, in which case it would be hard to learn.

Another observation is that the variance of the reward in the Asterix domain is much higher than that in the Fuel World domain. This is because that Asterix game is less tolerant to random actions than Fuel World is. The agent in Fuel World is allowed to make a non-optimal movement and still manages to arrives the goal. In Asterix, or generally in many games, there are many critical moments that the agent could get killed, if they doesn't choose an optimal movement. So, it's still possible that the agent can be killed after learning a perfect model.

IV. Discussion and Literature Review

A. Feature Extraction

Domain-specific features provide information that is helpful for the learning. They eliminate redundant observations. On the other hand, general features are more likely extracted from object recognition on the screen, possibly by a neural network [5]. It's possible to have more general, straightforward features, like tuples of (x, y) as inputs. However, it's unlikely that this can be accepted by a general model. In the experiments, all features are encoded as a vector of float numbers. If coordinates are unrolled, x would become a feature with a constant value, as lyres only move horizontally.

Our task is to select useful features out of general features. This is called the *Feature Subset Selection* problem [8]. It can be observed in Figure 5 that different choices of features can have significantly different learning performance. The domain-specific features are smaller in dimension and more relative to the learning task. Take Asterix as an example, after an expected successful feature selection, features of locations of lyres far away from the agent should be eliminated.

We expect features to be *strongly relevant* [2] to the samples. This can be defined as whether a difference in only one feature could result in a different label in the samples. However, there are noises in game domains, and the domains may not be perfectly Markov. The relevance of a feature can be identified according to its correlation with the labels.

There are two major models for features subset selection, the *filter model* and the *wrapper model* [8].

The Filter Model basically functions as a preprocesser for the learning algorithm. This assumes that feature selection can be done independently of the learning process. This is not practical in real time learning without knowledge of the domain. If we do have the knowledge of the domain, it becomes a trivial question - we perform as the filter and only select useful features.

The Wrapper Model changes the selection of features during the learning process. This is more likely to be useful in the game domains. In [2], the authors propose that the top k features with max correlation with the labels are selected and tested on a holdout set. This is generally used in text categorization tasks [10]. Similarly, in [8], the authors propose an idea using n-fold cross validation [13].

Another popular example of Wrapper Model is using genetic algorithms [18]. The features are represented as bits, where 1 means that the corresponding feature is selected, while 0 means not. Then the fitness function can combine the criteria of the accuracy and the cost.

From a different perspective, The task can be regarded as a searching problem - finding a subset of possible features [2]. Then it's possible that the learning agent starts with full set of features, and then eliminates features when the samples accumulate and the learning goes on. There are two issues here. First, same as common searching algorithms, it could get stuck at local optima. Second, searching is usually active. But in the game domains, the agent can not foresee what the set the features it will observe, so there can hardly be a heuristic way to do this.

Apart from Feature Subset Selection, another popular method in the literature, which also reduce the number of the features in favor of learning, is *Feature Compression*. The difference is that Feature Compression usually find a different set of features (usually done by picking up a projection matrix of size $D \times d$, which maps a vector from dimension of D down to dimension of d), which is smaller in cardinality, and can reconstruct the original state representation well [4]. This is a different goal from the one discussed in this report. We need to eliminate redundant information in the state representation. So there is no surprise that the subset of the features that we choose have less information, but complete and hopefully minimal for learning.

Feature Compression can also respond to the reward [12]. The criteria for the projection matrix talked above is not the ability to reconstruct the original representation. Instead, it responds to the reward gained. It's updated in a way in favor of the rewards.

B. Multi-agent Approach

TEXPLORE has been tested on many mostly static domains [6]. In this section, I analyze its performance on game domains.

In the experiments shown in Section III, we only build models for the agent and ascribe the change of the environment to the transition model. However, a common way is to use a multi-agent method [11]. Given all the features in the domain,

the agent needs to identify which are the adversary objects and which are neutral ones. After this is identified, and it has Markov property, we can frame a game as a *Markov Game*, in which our value is the maximum of the minimum of the values of the opponents [11].

We have seen that game domains are mostly dynamic. Different from static domains like Fuel World, the agent needs to learn the model of the dynamics of the environment. Generally, the agent can try to stay in place and observe the environment. However, this assumes that the agent knows which action means "staying". Even though the agent does know this, for the Asterix game in the experiment section, the agent would be killed in around 3 to 4 steps if it doesn't perform an avoiding action.

These are the challenges that make the agent fail to perform as well as in the Fuel World domain. In the experiments shown above, the action model of the agent is responsible for the transition model of the environment. For example, if a lyre moves left when the agent takes a right action, the agent will consider the effect of the lyre (actually, the corresponding feature) as an effect of the action, which is not expected.

One possible solution would be separating the transition model according to the roles. Consider separating the transition model in the following way, after identifying the agent and the opponents in the environment,

$$S_{op_i} \times S_a \rightarrow S'_{op_i}$$

$$S_a \times A_a \rightarrow S'_a$$

where op_i is the i-th opponent, S_{op_i} is the state of the i-th opponent. S_a is the state of the agent (the player) and A_a is the action of the agent. S' is generally the next state. Assume that the opponents behave independently (or no alliance). In this way, the transition model learns the behavior of the opponents and the control of the agents separately. The effect of an action of the agent becomes clear.

Unfortunately, this change doesn't make much difference. The problem is that there isn't a good way to separate the reward model in a similar way. The agent needs to learn that a negative reward comes when it's in a position same as any lyre. The problem of irrelevant features still exists.

V. Future Work

A. Feature Selection on TEXPLORE

This is essential for the work of TEXPLORE. The results have been shown that a much better performance can be achieved if subset selection works well. If TEXPLORE is not changed, it's useful to eliminate irrelevant features during the learning process. This could be done by either Feature Subset Selection or Feature Compression.

B. Hybrid Models

Empirically, decision trees have the best performance in many domains [7]. It's possible to test other models like Supper Vector Machine (SVM) and Artificial Neural Network (ANN) [16]. Decision trees and Artificial Neural Network do have nonlinear classification capacity, while SVM does not.

Presumably some domains can be better represented in ANN or other models. Therefore, it's possible to have a bag of different models, and their output can contribute to the overall model in the following way,

$$P(s,a) = \prod_i P_i(s,a) \cdot conf_i$$

where P_i is the prediction of the i-th model, and $conf_i$ is the confidence of the i-th model, or the probability that the i-th model is correct. $conf_i$ can be obtained from a test set in the supervised training. If we consider the most likely model is a hidden state in a Bayesian network, and can be better estimated over time, this approach can be regarded as a naive Bayesian RL method [14].

VI. Conclusion

This report shows that TEXPLORE can be applied to Atari game with acceptable learning performance. The performance is better if the features are carefully selected for this domain. There are also issues observed that TEXPLORE itself should accommodate. For example, it's expected to be able to select useful features itself. For a multi-agent domain, it should be aware of other agents. This accommodation is not designed for games exclusively, but for a more general domain.

Regarding the experiment results in Section III, ANN or other models may have equivalent or better performance, as the models in the random forests can be changed to arbitrary supervised learning models. It can also be tested on other game domains. TEXPLORE can be tested on selected features, if we use a wrapper model (so far, manually) to select useful features.

References

[1] Asterix (1983). http://www.emuparadise.me/Atari2600ROMs. Accessed: 2013-11-25.

[2] Avrim L Blum and Pat Langley. Selection of relevant features and examples in machine learning. *Artificial intelligence*, 97(1):245–271, 1997.

[3] Ronen I Brafman and Moshe Tennenholtz. R-max-a general polynomial time algorithm for near-optimal reinforcement learning. *The Journal of Machine Learning Research*, 3:213–231, 2003.

[4] Mahdi Milani Fard, Yuri Grinberg, Amir-massoud Farahmand, Joelle Pineau, and Doina Precup. Bellman error based feature generation using random projections on sparse spaces. *arXiv preprint arXiv:1207.5554*, 2012.

[5] Matthew Hausknecht, Piyush Khandelwal, Risto Miikkulainen, and Peter Stone. Hyperneat-ggp: A hyperneat-based atari general game player. In *Genetic and Evolutionary Computation Conference (GECCO)*, July 2012.

[6] Todd Hester. *TEXPLORE: Temporal Difference Reinforcement Learning for Robots and Time-Constrained Domains*. PhD thesis, University of Texas, 2012.

[7] Todd Hester and Peter Stone. Generalized model learning for reinforcement learning in factored domains. In *The*

Eighth International Conference on Autonomous Agents and Multiagent Systems (AAMAS), May 2009.

[8] George H John, Ron Kohavi, Karl Pfleger, et al. Irrelevant features and the subset selection problem. In *ICML*, volume 94, pages 121–129, 1994.

[9] Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *Machine Learning: ECML 2006*, pages 282–293. Springer, 2006.

[10] David D Lewis. Feature selection and feature extraction for text categorization. In *Proceedings of the workshop on Speech and Natural Language*, pages 212–217. Association for Computational Linguistics, 1992.

[11] Michael L. Littman. Markov games as a framework for multi-agent reinforcement learning. In *In Proceedings of the Eleventh International Conference on Machine Learning*, pages 157–163. Morgan Kaufmann, 1994.

[12] Poramate Manoonpong, Florentin Wörgötter, and Jun Morimoto. Extraction of reward-related feature space using correlation-based and reward-based learning methods. In *Neural Information Processing. Theory and Algorithms*, pages 414–421. Springer, 2010.

[13] L Breiman JH Friedman RA Olshen and Charles J Stone. Classification and regression trees. *Wadsworth International Group*, 1984.

[14] Pascal Poupart, Nikos Vlassis, Jesse Hoey, and Kevin Regan. An analytic solution to discrete bayesian reinforcement learning. In *Proceedings of the 23rd international conference on Machine learning*, pages 697–704. ACM, 2006.

[15] J. Ross Quinlan. Induction of decision trees. *Machine learning*, 1(1):81–106, 1986.

[16] Stuart Jonathan Russell, Peter Norvig, John F Canny, Jitendra M Malik, and Douglas D Edwards. *Artificial intelligence: a modern approach*, volume 74. Prentice hall Englewood Cliffs, 1995.

[17] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*, volume 1. Cambridge Univ Press, 1998.

[18] Jihoon Yang and Vasant Honavar. Feature subset selection using a genetic algorithm. In *Feature extraction, construction and selection*, pages 117–136. Springer, 1998.