Feature Selection for Reinforcement Learning by Learning Process Evaluation

Cleiton Alves da Silva
Escola de Artes, Ciências e Humanidades
Universidade de São Paulo (EACH USP)
cleitonalves@gmail.com

Valdinei Freire
Escola de Artes, Ciências e Humanidades
Universidade de São Paulo (EACH USP)
valdinei.freire@usp.br

ABSTRACT

Approaches to Transfer Learning in Reinforcement Learning aim to improve the learning process, whether in the same problem through generalization, by exploring similarity between similar states, or in different problems, by transferring acquired knowledge from a source problem to accelerate learning in a target problem. One way of obtaining generalization is through state abstraction through feature selection. This article presents FS-LPE algorithm to select a subset of features, where each subset of features is evaluated by designing an agent based on such features and observing the quality they present during the learning process. The algorithm is iterative, and in each round the FS-LPE algorithm evaluates a number of subset of features, gives a score for each features, and such score influence the next round of the algorithm. We propose three approaches to choose the subsets to evaluate at each round and compare them empirically in Discrete 2D Soccer Simulator.

Keywords

Reinforcement Learning, Transfer Learning and Feature Selection

1. INTRODUCTION

Problems involving sequential decision making can be solved by Reinforcement Learning (RL) techniques [13], which considers that the learning process is based on trial and error strategy, in which an agent does not need any information on the environment, but must learn to choose actions that allow the exploration and exploitation of the environment. After observing the result of the experience acquired with the execution of its actions, the agent that uses an RL algorithm such as Q-learning [21] or Sarsa [9] to estimate how well a particular action was performed on a particular state and stores the values that represent states and actions. The agent must be able to experience an infinite number of times the states for the algorithm to converge. In the tabular versions of the Q-learning and Sarsa algorithms, learning consists of learning a $Q : S \times A \rightarrow \mathbb{R}$ function involving $|S \times A|$ parameters. It is noted that this learning technique can become slow and impractical in environments where the size of state space grows exponentially according to the number of variables.

To deal with situations in which the state space is arbitrarily large, the agent must be able to generalize the knowledge gained from other experiences and share it among similar states [12], allowing to speed up learning. A common approach to generalization is to use a function approximation and to aggregate states that have similar characteristics, reducing the amount of learned parameters, making the value function approximate as a table. Also, to reduce the size of state space and speed up learning in problems where states are described by several features, we need to obtain a function approximation based on the methods of selection of features. In addition, another way to speed up learning is through Transfer Learning (TC) [16], considering that knowledge acquired in a source problem such as selected features can be transferred to a target problem and speed up learning.

The feature selection is a technique used to reduce dimensionality and consists of detecting, according to an evaluation criterion, features relevant to the original problem, which usually provides an improvement in learning, for example, by speed up the learning process. The main approaches to feature selection are: Filter, Wrapper and Embedded. The Filter approach estimates a relevance index for each feature after its evaluation and then ranks the features according to a statistical criterion. The Wrapper approach explores the feature space and generates multiple candidate subsets and, after evaluating each subset, selects the best performing subset. In the Embedded approach a learning algorithm performs the feature selection during the training process, adjusting the model and the feature selection simultaneously, such as the techniques used in classification trees.

This paper proposes a strategy for feature selection, integrating the Filter and Wrapper approaches, in order to discover subsets with the features that represent the original learning problem. Through the Filter approach each feature is individually rank according to its relevance, with the Wrapper approach the features subsets are evaluated based on sampling. Evaluations are performed during the learning process, selecting features that can speed up learning rather than just considering the power that a subset has to represent a quasi-optimal value function. This strategy allows the learning process to be tried over and over again. It is considered that the experience gained in solving the source problem, through the selected features subsets, can be transferred and used to improve learning in a more com-

Appears in: *Proceedings of the 1st Workshop on Transfer in Reinforcement Learning (TiRL) at the 16th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2017), A. Costa, D. Precup, M. Veloso, M. Taylor (chairs), May 8–9, 2017, São Paulo, Brazil.*

plex target problem.

The remainder of this paper is organized as follows: Section 2 presents background of Reinforcement Learning and Transfer Learning, Section 3 presents State Abstraction based on Learning-Process Evaluation and our propose. Section 4 presentes our experiments. Section 5 Feature Selection in Reinforcement Learning and Section 6 conclusion.

2. REINFORCEMENT LEARNING

The Reinforcement Learning problem considers an agent interacting with an environment, where the agent learns by trial and error, i.e., it has no beforehand information about the environment. At any time step t, the agent perceives the environment state s_t, chooses an action a_t and receives a reward r_t. The objective of the agent is to accumulate positive rewards, while avoiding negative ones.

2.1 SARSA Algorithm

While interacting with the environment, the agent accumulates experience $s_0, a_0, r_0, s_1, a_1, r_1, \ldots$. A RL algorithm must solve two problems: (i) how to obtain an optimal policy of action? and (ii) how to interact with the environment, i.e., to choose actions? A policy π maps each possible state $s \in \mathcal{S}$ into an action $a \in \mathcal{A}$, i.e., $\pi : \mathcal{S} \rightarrow \mathcal{A}$; here, an optimal policy is a policy that maximizes the expected discounted sum of rewards $\mathrm{E}[\sum_{t=0}^{\infty} \gamma^t r_t]$, where $\gamma \in (1, 0)$ is a discount factor. When acting in the environment to obtain experience, the agent must make a trade-off between exploitation, acting optimally by following the current learned policy to obtain rewards, and exploration, acting randomly to obtain experience to improve the current learned policy.

To obtain an optimal policy a common approach is to learn a value function $Q : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ to evaluate actions. Given a state s, $Q(s, a)$ evaluates the action a and a policy is obtained by $\pi(s) = \arg\max_{s \in \mathcal{S}} Q(s, a)$. A well-known algorithm in the literature is SARSA [9]; SARSA updates the value function Q at any time step t by using the tuple $(s_t, a_t, r_t, s_{t+1}, a_{t+1})$ with

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha_t \delta_t,$$

where $\delta_t = r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)$ is the temporal difference, and $\alpha_t \in (0, 1]$ is a learning rate. If an appropriated trade-off between exploitation and exploration and an appropriated learning rate are applied, independently of initial Q values, SARSA is guaranteed to converge to an optimal policy.

If the state-action pairs (s, a) are experienced by the agent an infinite number of times, the algorithm converges. However, it is necessary to specify an action selection strategy, when the agent must decide between exploration, i.e., learning more about actions that were not enough experienced in the environment, and exploitation, i.e., using current knowledge to guarantee rewards. The ϵ-greedy is a well-known strategy to the trade-off between exploration and exploitation; exploitation is applied with probability $1 - \epsilon_t$ and a uniform random choice is applied with probability ϵ_t.

2.2 Function Approximation

Although SARSA algorithms is proved to converge, convergence may take a long time. One technique to accelerate convergence considers some function approximation to value function Q [12, 10, 8]. In a tabular representation of Q values, it is necessary to learn $|\mathcal{S}| \times |\mathcal{A}|$ parameters; function approximation considers an analytical function with a vector of parameters $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_n)$ with $|\mathcal{S}| \times |\mathcal{A}| \gg n$, i.e., $Q(s, a) \approx q(s, a; \boldsymbol{\theta})$.

One way of obtaining a function approximation is by using abstraction, i.e., a function $M : \mathcal{S} \rightarrow \mathcal{X}$ that maps the original state space $\mathcal{S}$ into a smaller space $\mathcal{X}$ where $|\mathcal{S}| \gg |\mathcal{X}|$; then, tabular SARSA algorithm can be used directly in the mapped states.

The tile coding approach considers sets of abstractions $\{M_1, \ldots, M_{d_S}\}$ and approximates the Q values by the sum of each tabular function [13, 22, 11], i.e.,

$$q(s, a; \boldsymbol{\theta}) = \sum_{i=1}^{d_S} Q^i(M_i(s), a).$$

where each value $Q^i(M_i(s_t), a)$ is a parameter in the vector $\boldsymbol{\theta}$. SARSA algorithm can be used by updating each tabular function Q^i by

$$Q^i(M_i(s_t), a) \leftarrow Q^i(M_i(s_t), a) + \frac{\alpha}{d_S} \delta_t$$

where $\delta_t = r_t + \gamma q(s_{t+1}, a_{t+1}; \boldsymbol{\theta}) - q(s_t, a_t; \boldsymbol{\theta})$.

2.3 Transfer in RL

Another way to speed up learning in RL is to use Transfer Learning (TL) [16, 17], in which the agent must use experience gained in a source problem to speed up learning in a similar target problem. The TL problem is formalized when there is a source problem from which the knowledge is extracted and a target problem to which the knowledge is transferred. The hypothesis is that by reusing transferred knowledge will help solving the target problem faster.

The types of knowledge transferred between problems vary according to the different approaches that are commonly transferred: value function [15, 6]; Policy [2, 18, 19]; samples of interactions [7]; a state abstraction [20]. usually, two assumptions may be considered when the agent interacts with the source task: (i) the source task is simpler than the target one, and (ii) the agent interact with the environment for an arbitrary long time.

3. DESIGNING STATE ABSTRACTION BASED ON LEARNING-PROCESS EVALUATION

Although state abstraction may accelerate RL, designing an appropriated state abstraction automatically is not a trivial task. Under the assumption that there exists a set of state abstractions $\mathcal{F}$ and that the agent learns by using tile coding representation, the problem of designing a state abstraction can be pose as a feature selection problem. Consider a state s, than by using state abstractions $\mathcal{F}$ we can represent a state s by a vector of features $(F_1(s), F_2(s), \ldots, F_{|\mathcal{F}|}(s))$, where $F_i \in \mathcal{F}$ and $F_i \neq F_j$ for $i \neq j$. Then, designing an appropriated state abstraction is the same as selecting a subset of features. In this section we present a set of abstraction mappings and a feature selection method to evaluate state abstractions through the learning process.

3.1 Abstraction Mappings

In the experiments reported in the Section 4, the SARSA algorithm is combined to the Tile Coding function approximation and considers a set of tabular functions Q^i and approximates the Q function by the sum of these functions.

Each function Q^i considers a different abstraction M_i, enabling the generalization of knowledge differently in each function and allowing to approximate the Q function. In this work three types of abstractions are considered: Canonical, Delta and Joint, named as mappings.

Canonical mapping contains the direct variables of state space and represents the original features of the problem. Although in some problems the original features may help in constructing the Q function independently, this is not a general property. Thus, two forms of combinations between two canonical features were considered: Delta and Joint.

The two-to-two combinations of the existing features in the Canonical mapping define the amount of features of the Delta and Joint mappings. Let d be the quantity of Canonical features, the amount of Canons in Delta and Joint mapping is defined by $\frac{d(d-1)}{2}$. In the experiments carried out we consider the amount of 8 features existing in the Canonical mapping and 28 features for each of the Delta and Joint mappings.

Assume that a state s is defined by a d-tuple $(f_1(s), f_2(s), \ldots, f_n(s))$ and that each function $f_i(s) : S \to R_i$ has as its image a finite set R_i. Formally the mappings are defined as follows:

- Canonical mapping C is created from the use of the features that represent the original state of the problem, that is, $F_i^C(s) = f_i(s)$ with image R_i^C such that $|R_i^C| = |R_i|$;
- Delta mapping D is created from the existing combination between the difference between two Canonical features, that is, $F_{ij}^D(s) = f_i(s) - f_j(s)$ with image R_{ij}^D such that $|R_{ij}^D| = |R_i| + |R_j| - 1$; and
- Joint mappings J are created from the Cartesian product of two Canonical features, that is, $F_{ij}^J(f_1, ..., f_d) = f_j + |R_j|.f_i - 1$ with image R_{ij}^J such that $|R_{ij}^J| = |R_i|.|R_j|$.

Note that the size of the tables in each mapping type is different. A Delta mapping tends to be 2 times the Canonical mapping, while the Joint mapping tends to be the square of the Canonical mapping. Also, note that the number of Delta and Joint features grow in a quadratic fashion according to the number of Canonical features. While using only Canonical mappings may make learning impossible, using mappings can slow down learning.

3.2 Feature Selection through Learning-Process Evaluation

An essential part of feature selection problem is how to evaluate a set of features. In RL, usually it is considered how a set of features represents the value function; for example, the value function may be learned by considering the complete set of features, then, a small subset of features is selected that represents well that value function. Here, we propose to evaluate a set of features by the influence it has in the learning process; two main difficulties arise from this approach: (i) the learning process must be repeated over and over, that is why we consider such feature selection approach applied only for the knowledge transfer scenario; and (ii) there is not a unique way of evaluating the learning process.

Here we evaluate the learning process by simpling measuring the accumulated reward during a learning process for a fixed period, but different options may be considered. For example, the performance of the learned policy at the end of the learning process may be considered, or when its possible to evaluate, the amount of time to convergence. Somehow, accumulated reward combines performance of learned policy and time to convergence, and the importance that is given to each evaluation depends on the period for learning: if the period is long, importance is given to the performance of learned policy; if the period is short, importance is given to time to convergence. We set the fixed period empirically.

Another point regarding the evaluation of learning process is stochasticity; stochasticity comes from two places: (i) the environment itself is stochastic; and (ii) the trade-off between exploration and exploitation is stochastic. To evaluate a set of features properly averages of learning process trials should be done, increasing feature selection time. To couple with stochasticity we propose a hybrid method by using Filter and Wrapper methods: (i) sets of features are evaluated; (ii) the evaluation of each feature is averaged among the sets evaluation that it participates; and (iii) sets of features to be evaluated are chosen based on the evaluation of each feature.

The algorithm 1, `FS-LPE`, shows the solution to the problem of evaluating each of the various features and subsets. The algorithm considers four parameters: N_F, N_R, N_E, and T. The `FS-LPE` algorithm goes through N_R round; in each round, N_E subsets with N_F features are select (line 8); for each selected subset $\mathbf{S}(i)$, an agent with state abstraction based on the subset $\mathbf{S}(i)$ learns during T steps (line 9) and store the accumulated reward (line 10). The accumulated reward is used to evaluate each feature in that subset of features by the algorithm 2, `UpdateEval` (line 12).

For each feature in $\mathcal{F}$ (line 7–14), the `UpdateEval` algorithm accumulates rewards of experiments when the feature was considered (line 9) and when the feature was not considered (line 12); then the difference of the average of rewards when the feature is considered and the average of rewards when the feature is not considered is computed (line 20). If a feature never appears, the average is simply compute to be 0 (line 18); if a feature appears in every subset, the average is compute to be the best average among all the features (line 25). Finally, previous average is combine with the average in the current round (line 27).

FS-LPE algorithm must define the probability definition $Pr(F|\mathbf{e})$; here, we consider three different approaches: Probabilistic Selection, Hard Selection, and Uniform Selection. In the first approach, Probabilistic Selection, we choose a schedule $(p_1, p_2, \ldots, p_{N_R})$ where $p_j \geq \frac{N_F}{|\mathcal{F}|}$ defining a probability for each round; then for each round j

$$Pr(F_c|\mathbf{e}) = N_F \frac{\exp(\tau \mathbf{e}(c))}{\sum_{i=1}^{|\mathcal{F}|} \exp(\tau \mathbf{e}(i))},$$

where τ is chosen so that $\max_{c \in \{1,\ldots,|\mathcal{F}|\}} Pr(F_c|\mathbf{e}) = p_j$. We recommend choosing a schedule where $p_i > p_j$ if $i > j$ and $p_1 = \frac{N_F}{|\mathcal{F}|}$; then, in the first round all features have equal chances of being drawn and from the second round on the probability value increases, allowing more chances for features with better evaluation to be part of the subset.

In the second approach, Hard Selection, we choose a schedule $(n_1, n_2, \ldots, n_{N_R})$ where $n_j \in \{0, 1, \ldots, N_R - 1\}$ defining a number of fixed selected features for each round; then for

Algorithm 1 `FS-LPE` Algorithm.

```
1: Input: Set of mappings $\mathcal{F}$, number of features selected $N_F$, number of evaluations per round $N_E$, number of rounds $N_R$, time step $T$
2: Output: Subset with $N_F$ selected features
3: for $c = 1$ to $|\mathcal{F}|$
4:     $\mathbf{e}(c) \leftarrow 0$
5: endfor
6: for $j = 1$ to $N_R$ do
7:     for $i = 1$ to $N_E$ do
8:         Select a subset $\mathbf{S}(i) \subset \mathcal{F}$ with $N_F$ features by considering probability distribution $Pr(F|\mathbf{e})$
9:         Use a Reinforcement Learning agent that develops learning with the features of the subset $\mathbf{S}(i)$ for $T$ steps
10:        Store reward at $\mathbf{r}(i)$
11:    endfor
12:    $\mathbf{e} \leftarrow$ UpdateEval($\mathbf{e}, \mathbf{r}, \mathbf{S}$)
13: endfor
```

Algorithm 2 `UpdateEval` Algorithm.

```
1: Input: Set of mappings $\mathcal{F}$, number of evaluations per round $N_E$, vector of previous feature evaluations $\mathbf{e}$, vector of accumulated rewards $\mathbf{r}$, set of subset of features $\mathbf{S}$
2: Output: vector of updated features evaluations $\mathbf{e}$
3: for $c = 1$ to $|\mathcal{F}|$
4:     $Diff(c) \leftarrow 0, Soma(c) \leftarrow 0, N_{ocor}(c) \leftarrow 0$
5: endfor
6: for $i = 1$ to $N_E$ do
7:     for $c = 1$ to $|\mathcal{F}|$ do
8:         if $c \in \mathbf{S}(i)$
9:             $Soma(c) \leftarrow Soma(c) + \mathbf{r}(i)$
10:            $N_{ocor}(c) \leftarrow N_{ocor}(c) + 1$
11:        else
12:            $Diff(c) \leftarrow Diff(c) + \mathbf{r}(i)$
13:        endif
14:    endfor
15: endfor
16: for $c = 1$ to $|\mathcal{F}|$ do
17:    if $N_{ocor(c)} = 0$
18:        $\mathbf{m}(c) \leftarrow 0$
19:    else if $N_{ocor(c)} < N_E$
20:        $\mathbf{m}(c) \leftarrow \frac{Soma(c)}{N_{ocor}(c)} - \frac{Diff(c)}{N_E - N_{ocor}(c)}$
21:    endif
22: endfor
23: for $c = 1$ to $|\mathcal{F}|$ do
24:    if $N_{ocor(c)} = N_E$
25:        $\mathbf{m}(c) \leftarrow \max_{c \in \{1,\ldots,|\mathcal{F}|\}} \mathbf{m}(c)$
26:    endif
27:    $\mathbf{e}(c) \leftarrow \mathbf{e}(c) + \mathbf{m}(c)$
28: endfor
29: return the updated vector of evaluations $\mathbf{e}$
```

each round j

$$Pr(F_c|\mathbf{e}) = \begin{cases} 1 & \text{, if } c \text{ is } n_j\text{-best feature} \\ Pr(F_c|\mathbf{e}) = \frac{N_F - n_j}{|\mathcal{F}| - n_j} & \text{, otherwise} \end{cases},$$

where n_j-best feature means that c ranks equal or better than position n_j following evaluation $\mathbf{e}$. We recommend choosing a schedule where $n_i > n_j$ if $i > j$ and $n_1 = 0$; then, in the first round all features have equal chances of being drawn and from the second round on at least one new feature is hard selected, i.e., it will for sure be returned as a selected feature.

In the third approach, Uniform Selection, we constantly select for each round j:

$$Pr(F_c|\mathbf{e}) = \frac{N_F}{|\mathcal{F}|},$$

so that every feature has the same chance.

4. EXPERIMENTS

The experiments were performed in a 2D soccer simulator. The simulator can represent problems with different difficulties, depending on the chosen configuration. The experiments are performed with the configuration of two agents disputing a match, one of the agents implements the SARSA algorithm with function approximation and the other Planning agent. The SARSA algorithm in all experiments uses the following parameters: $\alpha = 0.1$, $\gamma = 0.99$ and $\epsilon = 0.1$. The other parameters defined in algorithm 1 are: number of features to be selected $N_F = 32$, number of evaluations per round $N_E = 10$, time step $T = 10^6$, number of rounds $N_R = 6$. The size of the field is 11×7. The M subset has 64 mappings (8 canonical features, 28 Delta mappings and 28 Joint mappings).

4.1 2D Soccer Simulator

We run our experiments in a Discrete 2D Soccer Simulator (D2DSS), implemented in MATLAB. The problem of soccer allows easily to set up multiagent environments; we can choose the amount of players on each team and the size of the field. Although a discrete simulator may be less realistic than a continuous one, the problems does not get that easier because of that, but allow less variable to be think of the RL algorithms, since discretization is given *a priori*.

In the D2DSS, the state is fully observable and is factored as follows: 4 factors for the ball, and 2 factors for each player. The ball state is represented by their vertical and horizontal coordinates, their velocity and their direction. Each player is represented only by their vertical and horizontal coordinates. For example, if we have one player in team A and one player in team B, then we have 8 factors in total.

In the D2DSS, the agent is suppose to control team A, while playing against team B. Each agent can choose among 13 actions: move to each of the four cardinal directions; kick the ball to each of the four cardinal directions and the four intercardinal directions; and try a tackle; although kick is only sounding when having a ball and tackle is only sounding when near a ball, these actions are allowed in not sounding situations and result no movement for the agent. In our experiments, the agents in team B implement a planning algorithm which considers a deterministic dynamic of the environment, heuristics and a small planning horizon; we call such agent Planning Agent. Because none learning happens in Planning Agents, we do not need to be concerned with game theory.

Finally, the D2DSS presents many configuration regarding stochasticity. It is possible to control: chance of player moving, chance of ball moving, chance of kicking and chance of tackle. It is also possible to control the size of the field and the number of players. Two other configurations are set automatically when the size of the field is chosen: the size of goal, and velocity of the ball after a successful kick.

In the example of Figure 1, the simulator settings define the use of two agents, and size of 11×7. The states are represented by Canonical mapping and the scenario of one player versus one player results in the following features:

- f_1 - position of the ball on the x-axis (11 values);
- f_2 - position of the ball on the y-axis (7 values);
- f_3 - direction of the ball (9 values);
- f_4 - ball velocity (3 values);
- f_5 - position of player A on the x-axis (11 values);
- f_6 - position of player A on y-axis (7 values);
- f_7 - position of player B on the x-axis (11 values);
- f_8 - position of player B on the y-axis (7 values).

In this configuration, the state space generated an amount of 12,326,391 states. To generate an environment with larger state space, to which the subset of features will be transferred, the setting that defines the size of the playing field has been changed to 33×21, resulting in a space of states with 8.9859e+09 states.

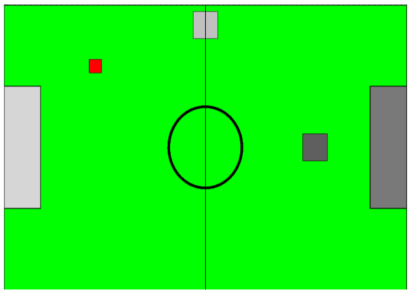

Figure 1: 2D Soccer Simulator. The figure shows two agents interacting with the environment and the ball. They are positioned in the field according to the coordinates on the Cartesian plane and player *A* (light grey) learns how to score more goals than team *B*.

4.2 Results

4.2.1 Feature Selection

Figures 2 and 3 shows our results for the FS-LPE algorithm when using Probabilistic Selection with schedule $(0.50, 0.59, 0.69, 0.79, 0.89, 0.99)$; results are averaged over 10 repetitions. Each curve shown in Figure 2 represents the mean of 100 learning curves (10 repetitions times $N_E = 10$ subsets) for each round; in addition to these curves, the average of 100 learning curves is shown using the 64 mappings features. Figure 3 shows the average of 100 learning curves (10 repetitions of the experiments times 10 repetitions of learning process with an agent with the same best features) of an agent designed with the 32 best feature in each round according evaluation vector $\mathbf{e}$; the average learning curve

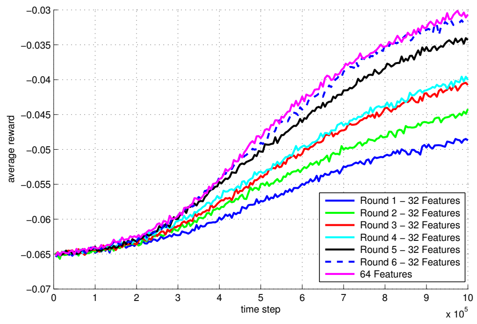

Figure 2: Average curves of learning obtained by Probabilistic Selection.

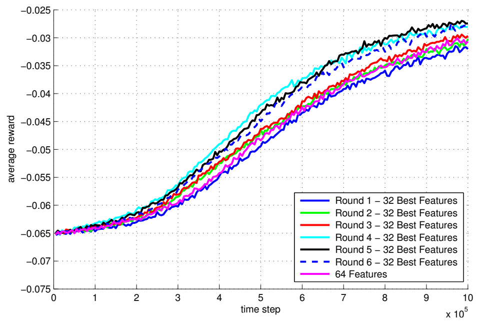

Figure 3: Average learning curves using the best selected features in each round by Probabilistic Selection.

using the 64 available mappings is also displayed for comparison.

Figures 4 and 5 shows our results for the FS-LPE algorithm when using Hard Selection with schedule $(0, 5, 11, 16, 21, 26)$; results are averaged over 10 repetitions. Figures have the same meanings that figures in Probabilistic Selection.

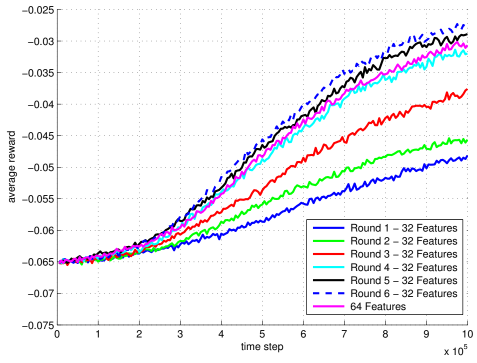

Figure 4: Average curves of learning obtained by Hard Selection.

Figures 6 and 7 shows our results for the FS-LPE algorithm when using Uniform Selection; results are averaged over 10 repetitions. Figures have the same meanings that figures in Probabilistic Selection.

Figure 8 shows the performance of the agent when accumulating rewards in each of the approaches: Probabilistic Selection, Hard Selection and Uniform Selection. There are two curves related to each approach, one represented

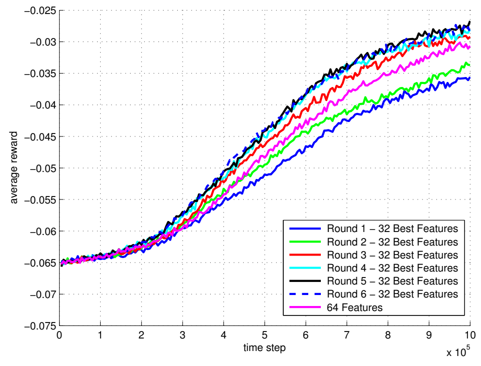

Figure 5: Average learning curves using the best selected features in each round by Hard Selection.

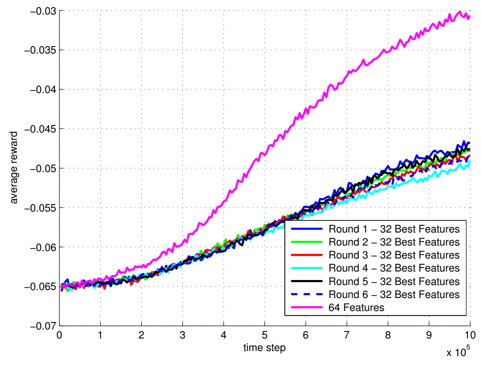

Figure 6: Average curves of learning obtained by Uniform Selection.

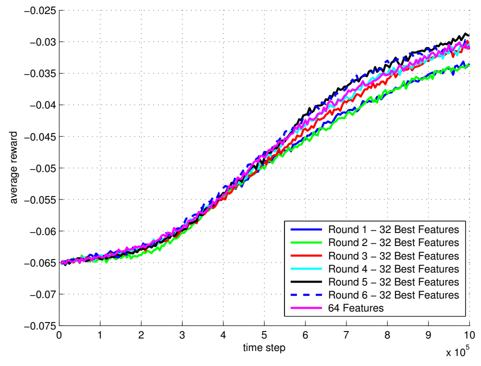

Figure 7: Average learning curves using the best selected features in each round by Uniform Selection.

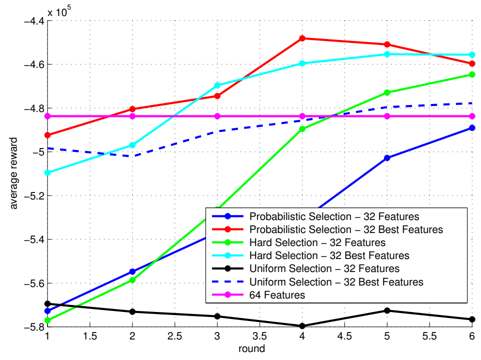

Figure 8: Accumulated rewards for each of the 3 approaches for each round.

by the subsets that were evaluated during the execution of the FS-LPE algorithm (32 Features), and another composed with the best features selected in each round (32 Best Features). In the end of 6 rounds, all of the approaches reached a better result than an agent designed withe the 64 features, but Probabilistic Selection and Hard Selection improves faster than Uniform Selection, showing that an appropriated schedule may result in a better result.

4.2.2 Transfer Learning

With the knowledge acquired through the FS-LPE algorithm, in this experiment, we tried to evaluate, through the transfer of knowledge, the performance of the agent when using subsets with better features of the source problem in a target problem. The settings for the parameters in this new problem are: field size 33×21 and each agent is evaluated during 10^7 steps. The subsets evaluated in this new problem are those obtained through the process of selection of attributes used in the three approaches.

The result presented in Figures 9, 10 and 11 are related to the use of the best subsets of features, according to the subset of features returned by the FS-LPE algorithm using Probabilistic Selection, Hard Selection and Uniform Selection, respectively.

Figure 12, shows the amount of rewards received in each round. In the target problem with field size 33×21, the agent uses the subsets of features that were selected and transferred from the source problem. The curve with the 64 available mappings is also presented for comparison.

We note in Figure 12 that Uniform Selection seems to be more coherent, even if only on the last round the agent with 32 features presented better result than the agent with 64 features, Uniform Selection shows an ascending behavior through rounds. Probabilistic Selection in every round presented a worse performance when compared to the agent with 64 features, but in the round 6 the agent with 32 features present a better result in the first half of steps (see Figure 9). Finally, Hard Selection presented the best performance but with no coherence: the best performance was on the round 2 and in the round 4 presented a performance worse than any other method.

Despite the negative transfer result at first sight, i.e, the agents with 32 features were not consistently better than the agent with 64 features; the different was not that large, but half of the memory and computation process is necessary to learn with an agent with half of the features. An agent may put up with a worse in performance if a gain in computational cost is obtained.

5. FEATURE SELECTION IN RL

Feature selection is considered important in an RL environment by reducing state space after identifying relevant features and substantially improving agent learning performance on the same problem. The methods related to feature selection are used for the purpose of selecting the relevant or irrelevant features from the original data, according to a certain evaluation criterion [3]. These features must adequately

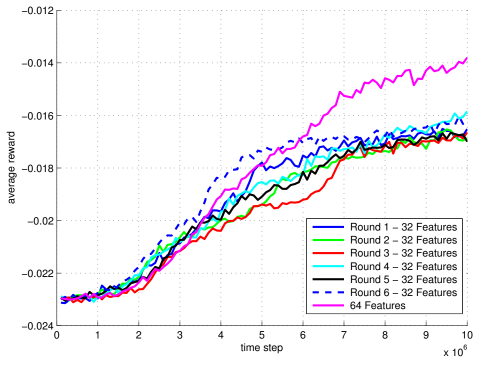

Figure 9: Average learning curves using the best selected features in each round by Probabilistic Selection in the environment with field size 33×21.

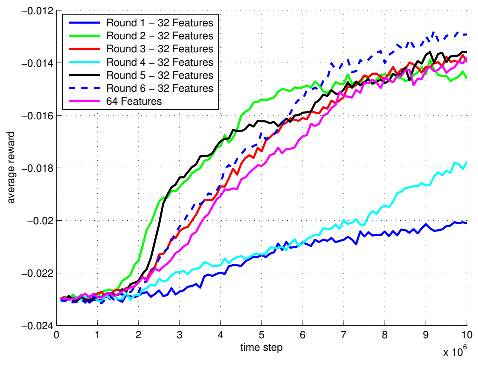

Figure 10: Average learning curves using the best selected features in each round by Hard Selection in the environment with field size 33×21.

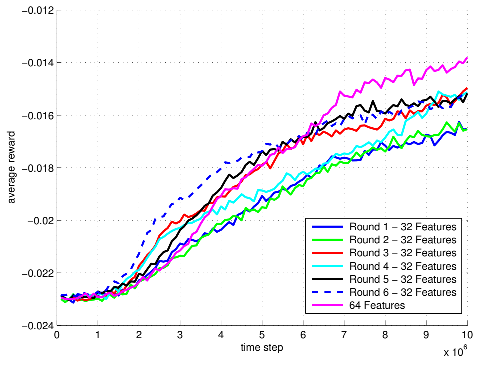

Figure 11: Average learning curves using the best selected features in each round by Uniform Selection in the environment with field size 33×21.

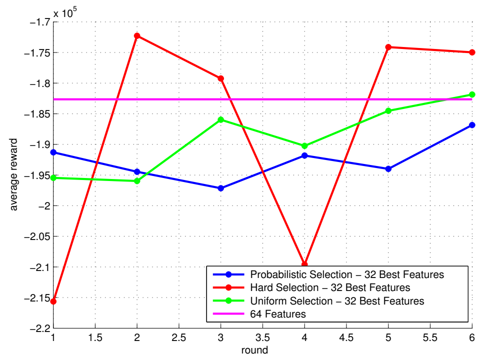

Figure 12: Accumulated rewards for each of the 3 approaches for each round in the environment with field size 33×21.

represent the learning problem and, in general, feature selection methods are classified into three approaches: Filter, Wrapper and Embedded [14]. The Filter approach applies a statistical measure by giving each feature a score that will rank it by relevance; the most commonly used algorithms are Relief [5] and Fisher score [1]. The Wrapper approach, rather than evaluating each feature individually, evaluates the subsets of Wrappers features; because it is put as a combinatorial optimization problem, some heuristics such as Hill climbing [4] and mixing forward and backward selection [23] are used. The Embedded approach describes the problem of selecting features as an optimization problem in which the use of features are penalized, and use constructive techniques such as classification trees, or techniques based on regularization.

6. CONCLUSION

The results presented in this paper demonstrate that the proposed feature selection method can find subsets of small size features that represent the original problem and that substantially improve agent performance and decrease computational costs in Transfer Learning. Considering the learning process as evaluation of subsets, it is possible to select features that not only adequately represent the value functions, but also allow to represent the value function during the learning process. If sufficient time for interaction with a source environment is available, the method proposed here allows selection of features that speed up learning.

Although the method proposed here achieves an speed up in learning, no comparison with literature methods was performed. A comparison with literature methods will allow to verify the quality of the selected features. Also, it is necessary to better understand the relationship between the selected features, the task used in the learning and the algorithm used for the learning.

In addition to investigating our approach in more detail, some immediate directions may be proposed. Initially how to select better features and then, as the features were evaluated individually, there is dependence between them. Since they present redundancy, then using a joint assessment of features can bring better results.

REFERENCES

[1] R. Duda, P. Hart, and D. Stork. *Pattern Classification.* Wiley, 2001.

[2] F. Fernández and M. Veloso. Probabilistic policy reuse in a reinforcement learning agent. In *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems*, AAMAS '06, pages 720–727, New York, NY, USA, 2006. ACM.

[3] R. Gaudel and M. Sebag. Feature selection as a one-player game. In J. Furnkranz and T. Joachims, editors, *ICML*, pages 359–366. Omnipress, 2010.

[4] M. A. Hall. Correlation-based feature selection for discrete and numeric class machine learning. In *Proceedings of the Seventeenth International Conference on Machine Learning*, ICML '00, pages 359–366, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

[5] K. Kira and L. A. Rendell. A practical approach to feature selection. In *Proceedings of the Ninth International Workshop on Machine Learning*, ML92, pages 249–256, San Francisco, CA, USA, 1992. Morgan Kaufmann Publishers Inc.

[6] G. Konidaris. A framework for transfer in reinforcement learning. In *In Proceedings of the ICML-06 Workshop on Structural Knowledge Transfer for Machine Learning*, 2006.

[7] A. Lazaric, M. Restelli, and A. Bonarini. Transfer of samples in batch reinforcement learning. In *ICML*, volume 307 of *ACM International Conference Proceeding Series*, pages 544–551. ACM, 2008.

[8] M. Ponsen, M. Taylor, and K. Tuyls. Abstraction and Generalization in Reinforcement Learning: A Summary and Framework. In M. Taylor and K. Tuyls, editors, *Adaptive and Learning Agents*, volume 5924 of *Lecture Notes in Computer Science*, pages 1–32. Springer Berlin Heidelberg, 2010.

[9] G. A. Rummery and M. Niranjan. On-line Q-learning using connectionist systems. Technical Report TR 166, Cambridge University Engineering Department, Cambridge, England, 1994.

[10] A. A. Sherstov and P. Stone. Function approximation via tile coding: Automating parameter choice. In J.-D. Zucker and I. Saitta, editors, *SARA 2005*, volume 3607 of *Lecture Notes in Artificial Intelligence*, pages 194–205. Springer Verlag, Berlin, 2005.

[11] P. Stone, R. S. Sutton, and G. Kuhlmann. Reinforcement learning for robocup-soccer keepaway. *Adaptive Behavior*, 13(3):165–188, 2005.

[12] R. S. Sutton. Generalization in reinforcement learning: Successful examples using sparse coarse coding. In *Advances in Neural Information Processing Systems 8*, pages 1038–1044. MIT Press, 1996.

[13] R. S. Sutton and A. G. Barto. *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition, 1998.

[14] J. Tang, S. Alelyani, and H. Liu. Feature selection for classification: A review, 2013.

[15] M. E. Taylor and P. Stone. Behavior transfer for value-function-based reinforcement learning. In F. Dignum, V. Dignum, S. Koenig, S. Kraus, M. P. Singh, and M. Wooldridge, editors, *The Fourth International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 53–59, New York, NY, July 2005. ACM Press.

[16] M. E. Taylor and P. Stone. Transfer learning for reinforcement learning domains: A survey. *The Journal of Machine Learning Research*, 10:1633–1685, 2009.

[17] M. E. Taylor, P. Stone, and Y. Liu. Transfer learning via inter-task mappings for temporal difference learning. *Journal of Machine Learning Research*, 8(1):2125–2167, 2007.

[18] M. E. Taylor, S. Whiteson, and P. Stone. Transfer learning for policy search methods. In *In ICML Workshop on Structural Knowledge Transfer for Machine Learning*, 2006.

[19] L. Torrey, J. W. Shavlik, T. Walker, and R. Maclin. Skill acquisition via transfer learning and advice taking. In *ECML*, volume 4212 of *Lecture Notes in Computer Science*, pages 425–436. Springer, 2006.

[20] T. J. Walsh, L. Li, and M. L. Littman. Transferring state abstractions between mdps. In *In ICML Workshop on Structural Knowledge Transfer for Machine Learning*, 2006.

[21] C. Watkins and P. Dayan. Technical Note. In R. Sutton, editor, *Reinforcement Learning*, volume 173 of *The Springer International Series in Engineering and Computer Science*, pages 55–68. Springer US, 1992.

[22] S. Whiteson, M. E. Taylor, and P. Stone. Adaptive tile coding for value function approximation. Technical Report AI-TR-07-339, University of Texas at Austin, 2007.

[23] T. Zhang. Adaptive forward-backward greedy algorithm for sparse learning with linear models. NIPS, 2008.