Reduction of Learning Time for Robots Using Automatic State Abstraction

Masoud Asadpour[1] Majid Nili Ahmadabadi [2] Roland Siegwart[1]

[1] Autonomous Systems Lab, Ecole Polytechnique Fédérale de Lausanne (EPFL), CH-1015 Lausanne, Switzerland {masoud.asadpour,roland.siegwart}@epfl.ch
[2] Control and Intelligent Processing Center of Excellence, ECE Dept., University of Tehran, Iran mnili@ut.ac.ir

Summary. The required learning time and curse of dimensionality restrict the applicability of Reinforcement Learning(RL) on real robots. Difficulty in inclusion of initial knowledge and understanding the learned rules must be added to the mentioned problems. In this paper we address automatic state abstraction and creation of hierarchies in RL agent's mind, as two major approaches for reducing the number of learning trials, simplifying inclusion of prior knowledge, and making the learned rules more abstract and understandable. We formalize automatic state abstraction and hierarchy creation as an optimization problem and derive a new algorithm that adapts decision tree learning techniques to state abstraction. The proof of performance is supported by strong evidences from simulation results in nondeterministic environments. Simulation results show encouraging enhancements in the required number of learning trials, agent's performance, size of the learned trees, and computation time of the algorithm.

Key words: State Abstraction, Hierarchical Reinforcement Learning

1 Introduction

It is well accepted that utilization of proper learning methods reduces robots' after-design problems caused naturally due to use of inaccurate and incomplete models and inadequate performance measures at the design time in addition to confronting unseen situations and environment and robot changes. Existing learning methods that are capable of handling dynamics and complicated situations are slow and produce rules – mostly numerical – that are not abstract, modular, and comprehensible by human. Therefore, the designer's intuition cannot be simply incorporated in the learning method and in the learned rules. In addition, the existing learning methods suffer from the curse of dimensionality, requiring a large number of learning trials and lack of abstraction capability. RL [1] despite its strength in handling dynamic and nondeterministic environments is an example of such learning methods. In this

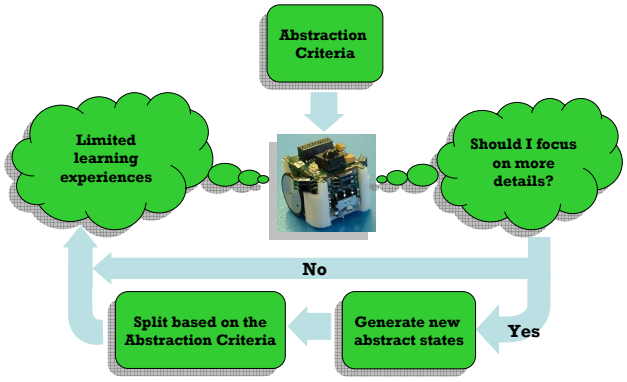

Fig. 1. state abstraction concept

research we try to solve some of the mentioned problems in RL domain because of its simplicity and variety of its applications in robotics field, in addition to its mentioned advantages.

It is believed that some of the discussed drawbacks can be lessen to a great extend by implementation of state abstraction methods and hierarchical architectures. Having abstract states and hierarchies in the agent's mind, the learned rules are partitioned, definition of performance measures for future learning is simplified, and the number of learning trials can be reduced by a careful design of learning method. In addition, incremental improvement of agent's performance becomes much simpler.

Different approaches have been proposed so far for state abstraction and creation of hierarchies and implementation of RL methods in those architectures. One group of these methods assumes a known structure and use RL methods to learn both rules in each layer and the hierarchy. Another group of researches proposes hand-design of subtasks and learning them in a defined hierarchy. These two approaches require much design effort for explicit formulation of hierarchy and state abstraction.Moreover, the designer should to some extent know how to solve the problem before s/he designs the hierarchy and subtasks. The third approach uses decision-tree learning methods to automatically abstract states, detect subtasks and speed up learning (Fig. 1). The devised approaches however do not take the exploratory nature of RL into account which results in non-optimal solution. In this paper, we take a mathematical approach to develop new criteria for utilization of decision trees in state abstraction. Our method outperforms the existing solutions in performance, number of learning trials, and size of trees.

Related works are reviewed in the next section. U-Tree algorithm and its drawbacks are briefly described in the third section. In the fourth section we formalize state abstraction as an optimization problem and derive new split criteria. The fifth section describes the simulation results. Conclusions and future works are discussed finally.

2 Related Works

The simplest idea to gain abstract knowledge is detection of subtasks, specially repeating ones [2], and discovering the hierarchy among them. Once the agent learned a subtask, which is faster and more efficient due to confronting with smaller state space, it would be able to use it afterwards. Subtasks in Hierarchical RL(HRL) are closed-loop policies that are generally defined for a subset of the state set [3]. These partial policies are sometimes called *temporally-extended-actions*, *options* [4], *skills* [5], *behaviors* [6], etc. They must have well-defined termination conditions.

Although these methods give more understanding about the underlying aspects of Hierarchical RL however; since particular features are designed manually based on the task domain, it is hard to apply them to complex tasks. The general problem that should be solved first, either when designing manually or extracting the structures automatically is the *state abstraction.* [3] In many situations significant portions of a large state space may be irrelevant to a specific goal and can be aggregated into a few, relevant, states. As a consequence, the learning agent can learn the subtask reasonably fast. Our work addresses this problem.

Techniques for non-uniform state-discretization are already known e.g. Parti-game [8], G algorithm [9], and U-Tree [10]. They start with the world as a single state and recursively split it when necessary. Continuous U-Tree [11] extends U-Tree to work with continuous state variables. TTree [12] applies Continuous U-Tree to SMDPs. Jansson and Barto [13] applied U-Tree to options but they use one tree per option and build the hierarchy of options manually. We adopt U-Tree as our basic method because the reported results illustrate it as a promising approach to automatic state abstraction.

In this paper we show that the employed U-Tree methods ignore RL's explorative nature. This imposes a bias in the distribution of samples that are saved for introducing new splits in U-Tree. As a result, finding a good split becomes more and more difficult and the introduced splits can be far from optimal. Moreover, U-Tree-based techniques have been excerpted in essence from decision tree learning methods. Therefore the used split criteria are very general and can work with any sort of data. Here we devise some specialized split criteria for state abstraction in RL and show that they are more efficient than the general ones both in learning performance and computation time.

Dean and Robert [14] have developed a minimization model, known as ϵ-reduction, to construct a partition space that has fewer number of states than the original MDP, while ensuring that the utility of the policy learned in the reduced state space is within a fixed bound of the optimal policy. Our work is different because ϵ-reduction does not extract any hierarchy among state

[3] The inverse of state abstraction, *State Aggregation* [15] clusters "similar" states and assigns them same values. It effectively reduces the state space size, but requires large number of trials and huge initial memory.

partitions, while building a hierarchy can reduce search time in partition lists from $O(n)$ to $O(\log n)$, n being the number of partitions. Also, their theory is developed based on immediate return of actions instead of long-term return. We can argue that ϵ-reduction is a special case of our method.

3 Formalism

We model the world as a MDP which is a 6-tuple (S, A, T, γ, D, R) , where S is a set of *states* (here can be infinite), $A = \{a_1, \ldots, a_n\}$ is a set of *actions*, $T = \{P^a_{s\acute{s}}\}$ is a *transition model* that maps $S \times A \times S$ into probabilities in $[0, 1]$, $\gamma \in [0, 1)$ is a *discount factor*, D is the initial-state distribution from which the start state is drawn (shown by $s_0 \sim D$), and R is a *reward function* that maps $S \times A \times S$ into real-valued rewards. A policy π maps from S to A, a real-valued *value function* V on states, or a real-valued *action-value function* (also called *Q-function*) on state-action pairs. The aim is to find an optimal policy π^* (or, equivalently, V^* or Q^*) that maximizes the expected discounted total rewards of the agent. We assume that each state s is a *sensory-input* vector $(x_1, \ldots, x_n)$, where x_i is a *state-variable* (called also *feature*, or *attribute*).

3.1 U-Tree

A U-Tree [10] [11] abstracts the state space incrementally. Each leaf L_j of the tree corresponds to an abstract state $\bar{S}_j$. Each leaf keeps action-values $Q(\bar{S}_j, a)$ for all available actions. The tree is initialized with a single leaf, assuming the whole world as one abstract state. New abstract states are added if necessary. Sub-trees of the tree represent subtasks of the whole task. Each sub-tree can have other sub-sub-trees that correspond to its sub-sub-tasks. The hierarchy breaks down to leaves that specify primitive sub-tasks.

Continuous U-Tree [11] loops through a two phase process: sampling and processing phases. In sampling phase, a history of the transition steps $T_i = (s_i, a_i, r_i, \acute{s}_i)$ composed of the current state, the selected action, the received immediate reward, and the next state is saved. The sample is assigned to a unique leaf of the tree, based on the value of the current state-variables. During the sampling phase the algorithm behaves as a standard RL algorithm, with the added step of using the tree to translate sensory input to an abstract state. After some number of learning episodes the processing phase starts. In processing phase a value is assigned to each sample:

$$V(T_t) = r_t + \gamma V(\bar{s}_{t+1}), \quad V(\bar{s}_{t+1}) = \max_a Q(\bar{s}_{t+1}, a) \tag{1}$$

where $\bar{s}_{t+1}$ is the abstract state that s_{t+1} belongs to. Then if a significant difference among the distributions of sample-values within a leaf is found that leaf is split in two leaves.

To find the best split point for each leaf, the algorithm loops over state-variables. The samples within a leaf are sorted according to a state-variable and a trial split is virtually added between each consecutive pair of state-variable values. This split divides the abstract state in two sets. A *splitting criterion*, like Kolmogorov-Smirnov test (KS), compares the two sets and returns a number indicating the different between the two distributions. If the largest difference among all state-variables is bigger than a confidence threshold, then a split is introduced. This procedure is repeated for all leaves.

Although the original algorithm for U-Tree [10] can function in *partially observable* domains, for the sake of simplification, we make the standard assumption that the agent has access to a complete description as well. Also, since the structure of the tree is not revised once a split is introduced; we assume the environment is *stationary*.

If we assume processing phase takes place after each learning episode, number of leaves [4] at episode t is $L(t)$, and sample size in each leaf is M, the best sort algorithms, i.e. Quicksort, would sort the samples based on a feature in $O(M\ \log M)$. Maximum number of split points could be $(M-1)$. KS and IGR tests need the samples in the either sides of the virtual splits to be sorted based on their values. This needs at least $O(2\ \frac{M}{2}\ \log \frac{M}{2}))$. Then a pass through all sorted samples needs $O(M)$. Totally the order of the algorithm, assuming n being dimension of the state vectors, is:

$$O\left(L(t)\ n\ M^2\ \log M\right) \tag{2}$$

3.2 Splitting Criteria

In literature two types of split criteria have been used for state abstraction. Rank-based tests e.g. non-parametric KS-test [10] [11], and tests inspired from Information Theory e.g. Information Gain Ratio(IGR) [17]. Non-parametric KS-test looks for violation of Markov Property by computing statistical difference between the distributions of samples on either side of a split using the difference in cumulative distributions of the two datasets. IGR-test measures the amount of reduction in uncertainty of sample values and is computed as follows; given a set of samples $T = \{T_i = (s_i, a_i, r_i, \acute{s}_i)\}, i = 1, \ldots, M$ from abstract state $\bar{S}$, labeled with $V(T_i)$ – computed according to (1) – the *Entropy* of sample-values is:

$$Entropy_{\bar{S}}(V) = -\sum_{v} P(V(T_i) = v|\bar{S})\ \log_2 P(v(T_i) = v|\bar{S}) \tag{3}$$

where $P(V(T_i) = v|\bar{S})$ is the probability that samples in T have value v. *Information Gain* of splitting the abstract state $\bar{S}$ to $\bar{S}_1$ and $\bar{S}_2$ is defined as the amount of reduction in the entropy of sample-values i.e.:

[4] Total number of nodes is $2\ L(t) - 1$.

$$InfoGain(\bar{S}, \bar{S}_1, \bar{S}_2) = Entropy_{\bar{S}}(V) - \sum_{i=1}^{2} \frac{M_i}{M} \; Entropy_{\bar{S}_i}(V) \qquad (4)$$

where M_i is the number of samples from T that fall in $\bar{S}_1$. IGR is defined as:

$$IGR(\bar{S}, \bar{S}_1, \bar{S}_2) = \frac{InfoGain(\bar{S}, \bar{S}_1, \bar{S}_2)}{-\sum_{i=1}^{2} \frac{M_i}{M} \log_2 \frac{M_i}{M}} \qquad (5)$$

3.3 Biased Sampling Problem

Each leaf of U-Tree in implementation can hold a limited number of samples. If the number of samples exceeds an upper limit, an old sample is selected randomly and is replaced by the new sample. However, selection probability of an action in a state depends on its action-value function. This fact affects the distribution of samples in a leaf. We call this a *biased sampling*.

The biased sampling problem has a negative effect; RL algorithms escape from local optimums by making a trade off between exploration and exploitation. If size of a sample list is not big enough, then number of samples with explorative character would not be statistically significant to introduce a split, until the confidence threshold for split criterion is reduced. Reducing the threshold increases the occurrence of non-optimal splits. One expect that size of the generated tree might be large with low performance. This problem can be solved by having in each leaf, instead of one big list with capacity k, $|A|$ smaller lists with capacity $\frac{k}{|A|}$ per action. Then the algorithm is modified so that when new samples arrive and the sample list is full, an old sample from the list that corresponds to the action of the new sample is deleted randomly. Now sorting the new sample list would be faster:

$$|A| \; O(\frac{k}{|A|} \; \log \frac{k}{|A|}) = O(k \; \log \frac{k}{|A|}) < O(k \; \log k) \qquad (6)$$

4 SANDS

In this section we formalize the state abstraction problem and provide mathematical description of a new algorithm that is specially derived for HRL. We call it SANDS which stands for State Abstraction in Nondeterministic Systems. Also we show how we can combine SANDS with other heuristics.

4.1 State Abstraction

An *abstract state* $\bar{S}$ is a subset of state space S, so that all states within it have "close" values (see bellow). *Value* of abstract state $\bar{S}$ is defined as the expected discounted reward return if an agent starts from a state in $\bar{S}$ and follows the policy π afterwards i.e.:

$$V^{\pi}(\bar{S}) = \sum_{s \in \bar{S}} \left\{ P(s|\bar{S}) \sum_{s \in A} \left[\pi(s,a) \sum_{\acute{s}} P^{a}_{s\acute{s}} (R^{a}_{s\acute{s}} + \gamma V^{\pi}(\acute{s})) \right] \right\} \tag{7}$$

where $\pi(s,a)$ is the selection probability of action a in state s under policy π, $P^{a}_{s\acute{s}}$ is the probability that environment goes to state $\acute{s}$ after doing action a in state s, and $R^{a}_{s\acute{s}}$ is the expected immediate reward after doing action a in state s and going to state $\acute{s}$. Similarly, $Q^{\pi}(\bar{S},a)$ is defined as:

$$Q^{\pi}(\bar{S},a) = \sum_{s \in \bar{S}} P(s|\bar{S}) \sum_{\acute{s}} P^{a}_{s\acute{s}} (R^{a}_{s\acute{s}} + \gamma V^{\pi}(\acute{s})) \tag{8}$$

From the set theory, a *set partition* of a set S, is defined as a collection of disjoint subsets of S whose union is S. So $U = \{\bar{S}_1, \ldots, \bar{S}_u\}$ is a set partition of S if and only if:

$$\begin{gathered} \bar{S}_i \subseteq S, \forall i \in \{1, \ldots, u\} \\ S = \bigcup_{i=1}^{u} \bar{S}_i \\ \bar{S}_i \bigcap \bar{S}_j = \emptyset, \forall i,j \in \{1, \ldots, u\}, i \neq j \end{gathered} \tag{9}$$

State abstraction could be formalized as an optimization problem, which aims at finding a set partition U and a policy π on the state space S, where u the number of partitions is minimum, and the value of the learned policy π is within a fixed bound ϵ of the optimal policy π^* i.e.:

$$|V^{\pi^*}(s) - V^{\pi}(\bar{S}_i)| < \epsilon, \forall s \in \bar{S}_i, \forall \bar{S}_i \in U \tag{10}$$

4.2 SANDS Splitting Criterion

It is easy to prove that leaves of U-Tree generate a set-partition of state space. It is clear from definition of state abstraction in (10) that we are minimizing the difference between the optimal policy and the policy that U-Tree represents. Here we, instead of walking toward the optimal policy which is unknown, try to walk away from the current policy toward the optimal policy. Now, assume that we have a U-Tree that defines a policy π. We would like to enhance it to a policy $\tilde{\pi}$ by splitting the partition $\bar{S}$ (one of the leaves in the tree) to unknown partitions $\bar{S}_1$ and $\bar{S}_2$, so that the expected return of the resulting tree is maximized i.e. we maximize:

$$G = \sum_{s \sim D} P(s)[V^{\tilde{\pi}}(s) - V^{\pi}(s)] \tag{11}$$

where D is initial states distribution. Using Bellman equations we can write:

$$G = \sum_{s \sim D} P(s) \begin{pmatrix} \sum_a [\tilde{\pi}(s,a) \sum_{\acute{s}} P^{a}_{s\acute{s}} (R^{a}_{s\acute{s}} + \gamma V^{\tilde{\pi}}(\acute{s}))] \\ - \sum_a [\pi(s,a) \sum_{\acute{s}} P^{a}_{s\acute{s}} (R^{a}_{s\acute{s}} + \gamma V^{\pi}(\acute{s}))] \end{pmatrix} \tag{12}$$

$$G = \sum_{s\sim D} \sum_{a} \sum_{\acute{s}} P(s) P^{a}_{s\acute{s}} \begin{pmatrix} R^{a}_{s\acute{s}}(\tilde{\pi}(s,a) - \pi(s,a)) \\ +\gamma(\tilde{\pi}(s,a) V^{\tilde{\pi}}(\acute{s}) - \pi(s,a) V^{\pi}(\acute{s})) \end{pmatrix} \quad (13)$$

$V^{\tilde{\pi}}(s')$ is unknown, we can use $V^{\pi}(s')$ as an estimation. For the states that fall outside of $\bar{S}$ policy remains unchanged after split i.e $\forall s \notin \bar{S}, \tilde{\pi}(s,a) = \pi(s,a)$. For the states within $\bar{S}$ or within the new abstract states $\bar{S}_1$ and $\bar{S}_2$ we define $\pi(a) = \pi(s,a), s \in \bar{S}$ and $\tilde{\pi}_i(a) = \tilde{\pi}(s,a), s \in \bar{S}_i, i = 1,2$. Hence:

$$G = \sum_{a} \sum_{i=1}^{2} \left((\tilde{\pi}_i(a) - \pi(a)) \sum_{s\in\bar{S}_i} \sum_{\acute{s}} P(s) P^{a}_{s\acute{s}} (R^{a}_{s\acute{s}} + \gamma V^{\pi}(\acute{s})) \right) \quad (14)$$

From the properties of conditional probabilities $P(s) = P(s|\bar{S}_i) \times P(\bar{S}_i|\bar{S}) \times P(\bar{S}), \forall s \in \bar{S}_i, i = 1,2$. Considering the definition in (8), we can write:

$$G = P(\bar{S}) \sum_{a} \sum_{i=1}^{2} \left[(\tilde{\pi}_i(a) - \pi(a)) Q^{\pi}(\bar{S}_i, a) P(\bar{S}_i|\bar{S}) \right] \quad (15)$$

This way the learner finds the split that maximizes the performance of the tree over the "whole action set", provided that actions are selected according to Softmax distribution. But an important point here is that the learner can not revise the tree after initiating the split. This means that it is fixing somehow a part of the policy in this level. Therefore, it is more valuable to judge the splits based on the performance of the greedy action instead of Softmax. The optimization term then should change from summation to maximum. Moreover, $P(\bar{S})$ is same for all states in abstract state $\bar{S}$. So, the final formulation for SANDS criterion is simplified to:

$$SANDS(\bar{S}, \bar{S}_1, \bar{S}_2) = \max_{a} \sum_{i=1}^{2} \left[(\tilde{\pi}_i(a) - \pi(a)) Q^{\pi}(\bar{S}_i, a) P(\bar{S}_i|\bar{S}) \right] \quad (16)$$

U-Tree generates a set-partition of state space. But, optimality of the number of partitions is not guaranteed although the algorithm starts with minimum partitions and adds more partitions if required. In fact the splits here are Univariate [16] that shrink the state space parallel to one of its axes. The necessary (but not sufficient) condition to have minimum size tree is to have Multivariate [16] splits, which can split the space in any direction. Multivariate splits are not covered in the current version of the algorithm however; we discuss the size of the trees in result section.

If we assume the processing phase starts after one learning episode, number of leaves in episode t is $L(t)$, sample size of the $|A|$ sample-lists in the leaves is $m = \frac{M}{|A|}$, and space dimension is n, time order of the new algorithm would be: $O(L(t)\ n\ \frac{M^2}{|A|})$ which is faster than (2) by $|A|\ \log M$ factor.

4.3 Using Mont Carlo Method to Estimate SANDS

We now show how to compute each term by Mont-Carlo method using the samples recorded during learning episodes. Each logged sample is a 4-tuple

$(s_i, a_i, r_i, \acute{s}_i)$, corresponding to current state, selected action, next state, and received reward. If among m^a samples for action a in $\bar{S}$, m_1^a and $m_2^a = m^a - m_1^a$ samples correspond to $\bar{S}_1$ and $\bar{S}_2$ respectively, we can compute:

$$\begin{aligned}
&\hat{\rho}_i^a = \tfrac{m_i^a}{m^a}, i = 1,2 && \hat{\rho}_i = \tfrac{\sum_a m_i^a}{\sum_a m^a}, i = 1,2 \\
&\hat{\mu}_i^a = \tfrac{1}{m_i^a} \textstyle\sum_{j=1}^{m_i^a} (r_j + \gamma \max_b Q(\acute{s}_j, b)), i = 1,2 && \hat{\mu}_a = \hat{\rho}_1^a \hat{\mu}_1^a + \hat{\rho}_2^a \hat{\mu}_2^a \\
&\hat{\pi}_i^a = \tfrac{\exp(\hat{\mu}_i^a/\tau)}{\sum_b \exp(\hat{\mu}_i^b/\tau)}, i = 1,2 && \hat{\pi}(a) = \tfrac{\exp(\hat{\mu}^a/\tau)}{\sum_b \exp(\hat{\mu}^b/\tau)} \\
&\multicolumn{2}{c}{SANDS(\bar{S}, \bar{S}_1, \bar{S}_2) = \max_a \textstyle\sum_{i=1}^2 (\hat{\pi}_i(a) - \hat{\pi}(a)) \hat{\mu}_i^a \hat{\rho}_i^a}
\end{aligned} \tag{17}$$

4.4 SoftMax Gain Ratio (SMGR)

New criteria can be derived by combining SANDS with different heuristics. One of them is what we call SoftMax Gain Ratio:

$$SMGR(\bar{S}, \bar{S}_1, \bar{S}_2) = \max_a \frac{-\hat{\pi}(a) \log \hat{\pi}(a) + \sum_{i=1}^2 \hat{\rho}_i^a \hat{\pi}_i(a) \log \hat{\pi}_i(a)}{-\sum_{i=1}^2 \hat{\rho}_i^a \log \hat{\rho}_i^a} \tag{18}$$

In this criterion, instead of defining entropy function on sample-values, we define it on action-probabilities and try to find the splits that maximize the reduction of this entropy i.e. the split that results in more certainty on selection probability of actions. We can prove that SMGR scales and combines SANDS with a penalty term as following:

$$SMGR(\bar{S}, \bar{S}_1, \bar{S}_2) = \frac{\frac{1}{\tau} SANDS(\bar{S}, \bar{S}_1, \bar{S}_2) + \log \sum_a \prod_{i=1}^2 \pi_i^{\hat{\rho}_i}(a)}{-\sum_{i=1}^2 \hat{\rho}_i \log \hat{\rho}_i} \tag{19}$$

We can show the penalty term punishes the splits that result in: (1) giving high probability to a specific action without support of enough evidence (samples), and (2) contradicting estimations on selection probabilities of a specific action in either side of the virtual split, in the sense that an action is the best action in one split but the same action is the worst action in the other one.

5 Simulation results

We have tested our algorithms on a simplified football task. It is very similar to the classic *Taxi* problem with the possibility of having moving players with different patterns to create environments with different levels of nondeterministicity. The playing field is an 8×6 grid (Fig. 2). The x axis is horizontal and y axis is vertical. There are two goals located at $(0, 3)$ and $(8, 3)$. A learning agent plays against two "passive" opponents and learns to deliver the ball to the left goal. By passive opponents we mean they move in the field only to restrict movements of the learner and do not perform any action on the

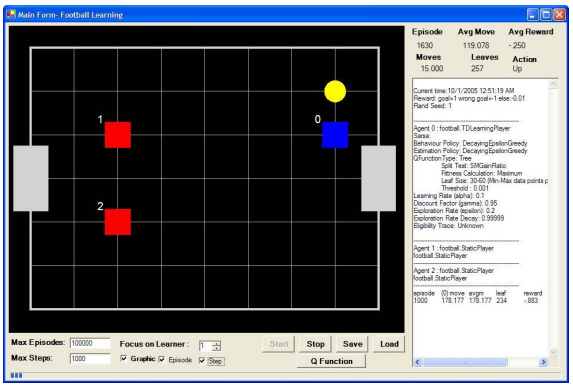

Fig. 2. A screen-shot from the simulator

ball. The learner can choose among 6 actions: *Left, Right, Up, Down, Pick, Put*. If any player selects an action that results in touching the boundaries or going to a pre-occupied position by another player, that action is ignored. In a special case when the learner has picked the ball, if it hits the other players or a boundary then the ball would fall down.

We test the learning simulations on 3 types of opponents: *Static, Offender,* and *Midfielder*. Static-players do not move at all. Offenders move randomly in x direction (i.e. they select randomly between Left and Right actions). Midfielders move randomly in all directions. States of the learner consists of absolute x and y position of ball and players, and status of ball from {*Free, Picked*} states, plus two state for left or right goal. This sums up to $3,001,252 (= 2 \times (7 \times 5)^4 + 2)$ states. Players act in parallel; so the next state of the learner when playing against offenders or midfielders would be a stochastically selected state among 4 or 16 different states, respectively. Each experiment consists of 100,000 episodes. Each episode starts by positioning the learner and ball in a random (and unoccupied) place. Each episode consists of maximum 1000 steps. Each step includes one movement by each player. An episode is finished if ball enters any goal or maximum number of steps passed. Upon delivering ball to the left goal the learner receives +1 reward. In case of wrong goal it receives -1 punishment. Otherwise it is punished with -0.01.

All methods use SARSA with decaying ϵ-greedy policy. Learning rate is 0.1 and discount factor γ is 0.95. Exploration rate ϵ is initialized with 0.2 and decays by multiplying to decay factor 0.99999 after each episode. Leaf sample size is 360 (60 samples per action). For each state-abstraction method a range of confidence thresholds are tested to find the best settings; the one that minimizes the number of actions each learner executes from a random initial position to the goal. The number of actions is averaged over the last 10,000 episodes. This average shows how effective has the agent learned to make goal in fewer moves. The results are averaged over 30 runs.

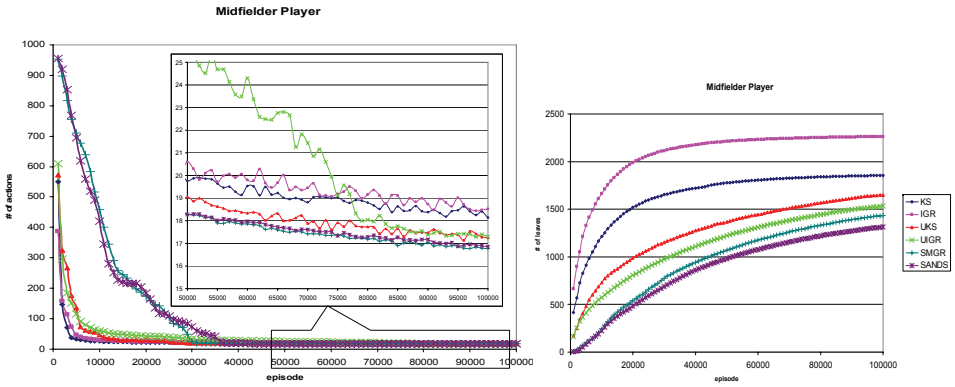

Fig. 3. Number of actions (left) and leaves of the tree (right) during learning episodes in midfielder case, averaged per every 1000 episodes (non-accumulated)

Table 1. Number of moves in each episode averaged over the last 10,000 episodes

Player	SARSA	KS	UKS	IGR	UIGR	SMGR	SANDS
Static	10.85±.01	11.23±.2	11.35±.22	11.21±.05	11.23±.04	11.12±.03	11.1±.04
Offender	19.63 ± .06*	14.97±.57	14.91±.14	14.55±.15	14.58±.5	14.28±.07	14.39±.09
Midfielder	668.36**	18.41±1.2	17.39±.21	18.65±1.73	17.39±.24	16.84±.06	16.95±.2

* 14.72 after 1 million episodes ** 18.92 after 15 million episodes

Table 2. Number of leaves in the learned trees

	SARSA	KS	UKS	GR	UIGR	SMGR	SANDS
Static	2452	692±119	596±92	703±35	666±28	516±10	491±7
Offender	120052	1476±404	1331±78	1536±302	1242±23	1259±26	1282±22
Midfielder	3001252	1857±253	1651±141	2263±504	1533±48	1436±78	1313±77

Table 3. Computation time of the algorithms in minutes

	SARSA	KS	IGR	SMGR	SANDS
Static	1	14	15	13	10
Offender	11*	25	27	18	12
Midfielder	50**	35	40	27	15

* 16 min for 1 million episodes ** 370 min for 15 million episodes

Table 1 shows number of actions in each episode averaged over the last 10,000 episodes. UKS and UIGR are new versions of KS and IGR with unbiased sampling. Table 2 shows number of leaves in the learned trees(For SARSA it is the number of states). Confidence intervals are computed with significance level 0.05. Table 3 shows computation time in minutes [5]. Fig. 3 shows number of actions and leaves recorded during learning episodes in midfielder case.

[5] Tests were done on a Dell® Inspiron™5150 laptop with CPU@3.06 MHz.

5.1 Hierarchical vs. Flat RL

From Table 1, it is clear that although SARSA shows better performance in static-player case but it has quite poor performance in other cases. Even we let the agent continue its learning episodes to 10 and 150 times more (1 and 15 million) for offender and midfielder cases, respectively. The performance is still poor compared to hierarchical methods.

By looking at time order of the HRL algorithms and flat RL we observe that SARSA takes only one minute to learn the static-player case, with higher efficiency than all hierarchical algorithms. But when environment becomes more nondeterministic the required computation time increases exponentially and efficiency of the results is far from that of hierarchical approaches. An interesting result for HRL algorithms is that the required time increases in logarithmic order while the number of states increases exponentially. These results show that HRL have a great potential in decreasing learning time and number of trials in online robot learning tasks.

5.2 Unbiased Sampling

Table 1 shows as environment becomes more and more nondeterministic, positive effect of unbiased sampling becomes more clear. Although the average in UKS and UIGR is a little bit bigger than KS and IGR for static players, it is much smaller for midfielder case. Also number of leaves in the unbiased versions is always fewer than the regular versions; e.g. difference between IGR and UIGR in midfielder case is around 700 partitions. Confidence intervals are always less for unbiased versions and this means that number of leaves in their generated trees does not vary a lot.

5.3 Efficiency of SANDS and SMGR

Table 1 and Table 2 show in all cases SANDS and SMGR create trees with better performance than all other methods (even unbiased versions). The difference becomes more clear as the environment becomes more nondeterministic. In midfielder player case the difference between the new criteria and the old ones is about 2 actions per episode. The same is true for size of the generated trees. In almost all cases (except in UIGR, offender player with negligible difference) SANDS and SMGR generate smaller trees.

Figure 3 shows that SANDS and SMGR converge slower than the others. Although this can be problematic when state space is very big, in the sense that their performance does not increase very much for a long time, but it is expected that after a time, their performance anticipate the other criteria. Table 1 and Fig. 3 (Left) confirm that splits in SANDS and SMGR are selected more carefully because not only their tree is the smallest, but also their performance is the best.

SMGR shows a small enhancement in offender and midfielder case compared to SANDS. It is due to a penalty term that SMGR adds to the splitting criterion which rejects some inefficient splits. In static and offender case it is not evident that which one generates fewer partitions. But in midfielder case SANDS generates trees with around 100 leaves less.

5.4 Computation time

As explained in Sect. 4.2, time order of SANDS and SMGR is less than KS and IGR. In static-player case the difference is not clear but in other cases SANDS decreases computation time to half compared to KS and even more compared to IGR. A part of the reduction in computation time is due to reduction in the order of sort algorithm in sample lists. Another part is due to creating smaller trees. This is important because it also affects the speed of knowledge retrieval and the required memory of the trees. Another part is due to reduction in the number of actions in episodes.

Conclusion

In this paper we formalized state abstraction as an optimization problem and by mathematical formulation we derived two new criteria for split selection. The new criteria adapt decision tree learning techniques to state abstraction. We showed in simulation that the new criteria outperform the existing criteria not only in efficiency and size of the learned trees but also in computation time. We believe that criteria, like KS and IGR, judge splits based only on their rank or probability of sample-values without taking their magnitude into account while; they are very informative for state abstraction. SANDS and SMGR mix probability and q-value of actions.

We have tested other criteria like Information-Gain [19], Gini-Index [18] and Students T-test [10] but the results were with limited success, as reported in [10]. Students T-test has poor performance when variance over the splits gets close to zero because of a variance term in its divisor. Information-Gain and Gini-Index have poor performance in nondeterministic environments.

Inability to revise the tree in non-stationary environments is a drawback of Univariate methods. Linear Multivariate [16] splits are perhaps more suitable candidates as they use a weighted sum of state-variables in split point. Adjusting these weights might be easier than reordering the nodes of the tree.

Acknowledgment

This research is funded by the Future and Emerging Technologies programme (IST-FET) of the European Community, under grant IST-2001-35506. The information provided is the sole responsibility of the authors and does not reflect the Community's opinion. The Community is not responsible for any use that might be made of data appearing in this publication. The Swiss participants to the project are supported under grant 01.0573 by the Swiss Government.

References

1. Sutton R.S. and Barto A.: Reinforcement Learning: An Introduction, MIT Press, Cambridge, MA (1996)
2. Uther W. and Veloso M.: The Lumberjack Algorithm for Learning Linked Decision Forests, In Proc. of PRICAI-2000 (2000)
3. Barto A.G. and Mahadevan S.: Recent Advances in Hierarchical Reinforcement Learning, Discrete Event Dynamic Systems, 13(4):41–77 (2003)
4. Sutton R.S., Precup D., and Singh S.: Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning, Artifcial Intelligence, 112:181–211 (1999)
5. Thrun S.B., and Schwartz A.: Finding structure in reinforcement learning, In Tesauro G., Touretzky D.S., and Leen T.(eds), Advances in Neural Information Processing Systems, pp.385–392, MIT Press, Cambridge, MA(1995)
6. Brooks R.A.: Achieving artificial intelligence through building robots, Technical Report A.I. Memo 899, AI Lab., MIT, Cambridge, MA (1986)
7. Dayan P. and Hinton G.E.: Feudal reinforcement learning, In Hanson S.J., Cowan J.D., and Giles C.L.(eds), Neural Information Processing Systems 5, San Mateo, CA, Morgan Kaufmann (1993)
8. Moore A.W.: The parti-game algorithm for variable resolution reinforcement learning in multidimensional state-spaces, In Cowan J.D., Tesauro G., and Alspector J.(eds), Advances in Neural Information Processing Systems, 6:711-718, Morgan Kaufmann (1994)
9. Chapman D., and Kaelbling L.P.: Input generalization in delayed reinforcement learning: An algorithm and performance comparisons, In Proc. of the 12th International Joint Conf. on Artificial Intelligence (IJCAI-91), pp.726-731 (1991)
10. McCallum A.: Reinforcement Learning with Selective Perception and Hidden State, PhD thesis, Computer Science Dept., Univ. of Rochester (1995)
11. Uther W.T.B. and Veloso M.M.: Tree based discretization for continuous state space reinforcement learning, In Proc. of the 15th National Conf. on Artificial Intelligence, AAAI-Press/MIT-Press, pp.769-774 (1998)
12. Uther W.T.B.: Tree Based Hierarchical Reinforcement Learning. PhD thesis, Dept. of Computer Science, Carnegie Mellon Univ., Pittsburgh, PA, USA (2002)
13. Jonsson A. and Barto A.G.: Automated state abstraction for options using the U-tree algorithm, In Advances in Neural Information Processing Systems: Proc. of the 2000 Conf., pp.1054–1060, MIT Press, Cambridge, MA (2001)
14. Dean T.,and Robert G.:Model minimization in markov decision processes, In Proc. AAAI-97, p.76 (1997)
15. Singh S.P., Jaakola T., and Jordan M.I.: Reinforcement learning with soft state aggregation, In Tesauro G., Touretzky D.S., and Leen T.K.(eds), Neural Information Processing Systems 7, MIT Press, Cambridge, MA (1995)
16. Yildiz O.T., Alpaydin E.: Omnivariate Decision Trees, IEEE Trans. on Neural Networks, 12(6):1539–1546 (2001)
17. Au M., Maire F.: Automatic State Construction using Decision Tree for Reinforcement Learning Agents, International Conf. on Intelligent Agents, Web Technologies and Internet Commerce (CIMCA), Gold Coast, Australia (2004)
18. Breiman L., Friedman J., Olshen R., Stone C.: Classification and Regression Trees, Wadsworth, Pacific Grove, CA (1984)
19. Quinlan R.: Induction of decision trees, Machine Learning, 1:81–106 (1986)