ECAI 2016
G.A. Kaminka et al. (Eds.)
© 2016 The Authors and IOS Press.
This article is published online with Open Access by IOS Press and distributed under the terms of the Creative Commons Attribution Non-Commercial License 4.0 (CC BY-NC 4.0).
doi:10.3233/978-1-61499-672-9-46

Dynamic Choice of State Abstraction in Q-Learning

Marco Tamassia and **Fabio Zambetta** and **William L. Raffe** and **Florian 'Floyd' Mueller** and **Xiaodong Li**[1]

Abstract. Q-learning associates states and actions of a Markov Decision Process to expected future reward through online learning. In practice, however, when the state space is large and experience is still limited, the algorithm will not find a match between current state and experience unless some details describing states are ignored. On the other hand, reducing state information affects long term performance because decisions will need to be made on less informative inputs. We propose a variation of Q-learning that gradually enriches state descriptions, after enough experience is accumulated. This is coupled with an ad-hoc exploration strategy that aims at collecting key information that allows the algorithm to enrich state descriptions earlier. Experimental results obtained by applying our algorithm to the arcade game Pac-Man show that our approach significantly outperforms Q-learning during the learning process while not penalizing long-term performance.

1 Introduction

Planning and learning under uncertainty are fundamental problems in artificial intelligence. A framework to address such problems is the Markov Decision Process (MDP) [1]. MDPs are based on the Markov assumption which states that it is sufficient to know the current state of the environment to make predictions about the outcome of actions. One way for an agent to learn useful information about the environment dynamics is by interacting with it, in a sequence of observations of state and action. Based on the Markov assumption, Temporal Difference (TD) algorithms [2] encode useful information about the environment in the form of associations of states to utilities. For example, Q-learning, one of the most popular TD algorithms, associates state-action pairs to future rewards [3]. When a TD agent needs to make a decision, it will choose the action that is likely to yield the highest long-term utility value according to previous experience.

If states are rich in information, in the early stages of the learning process, the agent knowledge of the environment is sparse. If the agent considers every feature making up the state, it will take a considerable amount of time to learn associations for all of the many possible states, especially if outcomes are stochastic. To quote Andre and Russell, "Without state abstraction, every trip from A to B is a new trip" [4]. During this learning period, an agent may make blind decisions due to "details" in the state preventing an exact match with past experience. TD agents lack the ability to use knowledge of states similar to the current one. This research problem falls under the umbrella of Transfer learning [5].

The most common approach to address this issue is to use linear approximation [6], [7]. In this instance, an agent only has to learn the weights of the linear transformation mapping state features to utility.

However, a linear approximation may not be sufficient if non-linear dynamics exist in the environment. In this case, sparsity issues are addressed by stripping states of "superfluous" details. This process is called *state abstraction*, and it consists in aggregating states. By only considering the most important information and ignoring details, two states that are effectively different will appear the same, inducing a partitioning of the state space. The drawback of this, however, is that if the information used to encode states is not rich enough, the agent will not be able to make informed decisions. Examples of this approach are coarse coding and tile coding [1].

State abstraction has attracted attention in the reinforcement learning community in the past two decades. Most of the literature on the subject focuses on choosing an abstraction prior to the actual learning [8]–[11]. McCallum's work, however, explored online state abstraction, which is also the focus of our work. [12]–[14].

In this paper we propose an algorithm that shifts from coarse partitionings to more fine-grained ones through time. The choice of which partitioning to use is done at every step and can be different from state to state, allowing for more flexible learning. The criteria used by our algorithm to decide when to enrich state information is to compare the confidence interval of utility estimates. The idea is that, at the beginning of the process, most decisions are made using coarse partitionings while, in the long run, more choices are made with more informative partitionings. To our knowledge, this is the first attempt to combine both coarse and fine-grained partitionings online.

We evaluate our algorithm by comparing it with Q-learning in the context of the video game Pac-Man. Our experiments show that the proposed algorithm produces better performance than fixed state-size Q-learning during the learning phase. We also propose a strategy to direct exploration in a way that allows the algorithm to switch to fine-grained abstractions earlier. Experiments show that this strategy produces better performance than the standard ϵ-Greedy.

The paper is structured as follows: in Section 2 we introduce Markov Decision Processes and Q-learning and provide an overview of the relevant literature; in Section 3 we introduce our algorithm and our ad-hoc exploration strategy; in Section 4 we detail the setup of our experiments and in Section 5 we report the results of our experiments.

2 Background and Notation

A Markov Decision Process is formally defined by a tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, T, R, \gamma)$, where $\mathcal{S}$ is the set of all possible *states* of the environment; $\mathcal{A}$ is the set of *actions* the agent can perform; $T : \mathcal{S} \times \mathcal{A} \to \mathbb{P}(\mathcal{S})$ associates a state-action pair to a *next state probability distribution*; $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ associates a state-action pair to a *reward*; $\gamma < 1$ is a *discount factor* used to decrease the value of future rewards.

An agent interacting with the environment, tries to maximize the cumulative reward collected over time. While interacting with the environment, the agent collects information: at time t, after observing state s_t, it decides which action a_t to perform and observes the reward r_t it receives and the next state s_{t+1} to which the environment tran-

[1] RMIT University, Melbourne, Australia, email: first.last@rmit.edu.au

sitions. Collected information can be used to make informed future decisions.

An effective way to achieve optimal decision making is to compute, for all $s \in \mathcal{S}$ and $a \in \mathcal{A}$, an estimate $Q^*(s,a)$ of the expected cumulative reward to be expected by taking action a while in state s and behaving optimally afterwards. Then, the optimal policy π^* is defined as: $\pi^*(s) = \arg\max_{a\in\mathcal{A}} Q^*(s,a)$. The Q-learning algorithm [3] updates of the estimates after each step of the agent according to the following rule:

$$Q(s_t,a_t) \overset{\alpha}{\leftarrow} r_{t+1} + \gamma \max_{a\in\mathcal{A}} Q(s_{t+1},a) - Q(s_t,a_t),$$

where $x \overset{\alpha}{\leftarrow} y$ is short for $x \leftarrow x + \alpha y$ and $0 < \alpha \leq 1$ is a learning factor.

Since the agent starts the process with no information about the environment, it needs to balance exploration of the environment with exploitation of the current knowledge. This is to avoid focusing on what the agent believes to be the best action and missing the actual best action due to incomplete knowledge. A common exploration strategy, which we also use in this work, is called ϵ-greedy and is defined as follows:

$$\pi_\epsilon(s) = \begin{cases} \arg\max_{a\in\mathcal{A}} Q(s,a) & \text{with prob. } 1-\epsilon \\ \text{random}(\mathcal{A}) & \text{with prob. } \epsilon \end{cases},$$

where $0 \leq \epsilon \leq 1$ balances exploration and exploitation.

Q-learning is said to learn "off-policy"; this means that, in the long run, its estimates $Q(\cdot,\cdot)$ approximate the exact values $Q^*(\cdot,\cdot)$ regardless of what policy the agent is following[2]. This allows an agent to learn the optimal policy while using a sub-optimal exploration-oriented policy, such as ϵ-greedy.

Related work

Significant research effort has been directed in state abstraction. However, most of the work has been focused on choosing an abstraction prior to the learning phase. Many papers fall in this area: [8] use hypothesis testing to discover useful abstractions using Q-values learned in multiple runs; [9] select the features that are useful to reproduce the behavior of a given set of demonstrations; [10] use time-series data to select among the models provided by a human expert; [11] provide theoretical guarantees on the used abstraction, but their approach requires the use of value iteration, an expensive, model-based algorithm [1].

In the recent years there has been work at the intersection of Deep Learning and Reinforcement Learning. In the work by Mnih *et al.*, neural networks are able to learn features of the game state from very unstructured data (such as pixels) [15]. We propose a different approach: rather than learning features over time from raw data, we suggest that, given a set of meaningful features, an algorithm can use more and more of them over time to make its decisions, and that this can help achieve better performance at the early stages of the learning process, when knowledge of the environment is still scarce.

Literature covers two different, principled approaches at online learning of state abstractions:

- adaptively expanding memories to store past information, which is helpful when past information is relevant to make good decisions [12]–[14];
- adaptively split tiles in tile coding [1], [16]–[18], which works well when state features range widely in ordinal values.

[2] as long as there is a non-zero probability of visiting each state

The first of these approaches are all works of McCallum. In [12], he expands temporal memory to distinguish variations in rewards, and does so via hypothesis testing; this approach, however, is slow because memory is expanded one step at a time. In [13] he proposes to store raw history, so when memory is expanded, history can be re-analysed to properly compute values; this approach, reportedly, does not handle noise very well. In [14], he proposes to use, along with stored history, a tree to know how deep (how far back in history) one needs to look to distinguish situations: branches are added when a statistical test says that samples come from two different distributions[3].

The second approach focuses on tile coding. Tile coding (TC) [1], [16] is a technique to extract useful features from large state spaces including widely varying ordinal features. The idea is to produce multiple discretizations (tilings) of the state features with different offsets: for each discretization, every tile becomes a state feature all of which but one are set to zero. To expand on this, Whiteson *et al.* propose to adaptively split tiles so as to maximize changes in the value function or in the policy during learning [17]. More recently, a paper by Scopes and Kudenko proposes to split tiles that are closer to the optimal transition path and suggests that perpendicular tiles to the optimal path can achieve better performance that square tiles.

The approach we propose complements the above mentioned, because we focus on augmenting the set of features used to describe the state, as opposed to adding information from the past or augmenting the granularity within the features. the video game Pacman is one example of scenario where the mentioned approaches would not work well. To an agent that already has information about food and ghosts, it is more useful to know where the closest power capsule is than it is to know where food and ghosts were in the past. Tile coding (and derivatives) are also ill-suited to Pacman because the features have few possible values each, so they would not allow for many tiles; furthermore,half of the features are not ordinals, making TC inapplicable on them. TC could possibly be applied to the original features, but this would mean counting pills per tile, and this, to the best of our knowledge, has not been tried before and deserves a deep analysis per se.

3 Dynamic Abstraction Choice

In reality, because environments are often stochastic, a number of trials are necessary for each state and action pair to evaluate reasonable estimates. In particular, in the early stages of an agent's life, its knowledge is rather sparse, often leading to blind decisions. To deal with this, a common approach is not to model the entire MDP, but a simplified one obtained by reducing the state space size via state aggregation [5], [20]. By means of carefully engineered state aggregations, Q-learning generalizes well over the little information it has.

However, it is desirable that in the long-run, the agent makes its decisions considering all the nuances of each state, rather than based on coarse aggregations. More information on the state allows the agent to make more informed decisions.

We propose a novel algorithm to achieve the advantages of both situations, at the cost of a slight increase in processing time. In the following, we refer to abstractions as functions mapping a state to an aggregation of states. Each abstraction induces an abstrated state space of smaller size than the original one and, consequently, a smaller Q-table. However, such Q-tables do not need to be memorized since they can be inferred by appropriately aggregating entries of the original Q-table.

[3] The test used is Kolmogorov-Smirnov [19]

Algorithm 1: Multi-Abstraction Q-learning algorithm for abstraction shifting. $\biguplus$ indicates a multiset sum; a multiset is a set where information about the number of occurrences of each element is preserved. $\overline{X}$ indicates the sample average. Procedure CI computes the confidence interval of the mean of the given sample.

```
   Input  : Learning rate $\alpha$
   Input  : Exploration parameter $\epsilon$
   Input  : Abstractions $\beta_1 > \beta_2 > \ldots > \beta_m$
   Input  : Default Q-value, initQ
   Input  : Significance level for t-tests, $\sigma$
1  for $s, a \in \mathcal{S} \times \mathcal{A}$ do
2      $Q(s,a) \leftarrow$ initQ
3      $H(s,a) \leftarrow$ empty list    // history of $Q(s,a)$
4  end
5  $s \leftarrow$ observe state
6  repeat
7      $j^* \leftarrow m$
8      found $\leftarrow$ false
9      for $j \leftarrow 1 \ldots m$ do
10         $\xi \leftarrow \{ s' \in \mathcal{S} \mid \beta_j(s') = \beta_j(s) \}$    // siblings
11         for $a \in \mathcal{A}$ do
12             $X_a^j \leftarrow \biguplus_{s' \in \xi} H(s',a)$    // samples of siblings
13             $\perp_a^j, \top_a^j \leftarrow \mathrm{CI}(\sigma, X_a^j)$    // lower and upper bounds
14         end
15         $a^* \leftarrow \arg\max_{a \in \mathcal{A}} \overline{X_a^j}$
16         if $\perp_{a^*}^j > \top_a^j$ for all $a \in \mathcal{A}, a \neq a^*$ then
17             $j^* \leftarrow j - 1$
18             found $\leftarrow$ true
19             go to line 22
20         end
21     end
22     $a* \leftarrow \begin{cases} \arg\max_{a \in \mathcal{A}} \overline{X_a^{j^*}} & \text{with prob. } 1-\epsilon \\ \mathrm{random}(\mathcal{A}) & \text{with prob. } \epsilon \end{cases}$
23     perform action $a^*$
24     $s' \leftarrow$ observe state
25     $r \leftarrow$ receive reward
26     $\hat{q} \leftarrow r + \gamma \max_{a' \in \mathcal{A}} Q(s',a')$
27     $Q(s,a^*) \overset{\alpha}{\leftarrow} \hat{q} - Q(s,a^*)$
28     append $\hat{q}$ to $H(s,a^*)$
29     $s \leftarrow s'$
30 until apocalypse
```

The algorithm we propose, “Multi-Abstraction Q-learning”, is presented in pseudocode in Algorithm 1. Multi-Abstraction Q-learning is given a list of abstractions of decreasing granularity, and maintains the Q-table associated with the original state representation. At decision time, the algorithm chooses the most granular abstraction whose Q-values are precise with sufficient confidence.

Formally, an abstraction is defined as $\beta_i : \mathcal{S} \to \mathcal{S}_i$. We also introduce an ordering for abstractions, based on their granularity. Formally, an abstraction β is more granular than a second abstraction β' (denoted $\beta > \beta'$) if both the following conditions hold:

- Any two states $s, s' \in \mathcal{S}$ mapped to the same abstracted state by (the more granular) abstraction β are also mapped to the same abstracted state by (the more coarse) abstraction β'. Formally:

$$\forall s, s' \in \mathcal{S} \,.\, \beta(s) = \beta(s') \Rightarrow \beta'(s) = \beta'(s')$$

- There is at least a pair of states $s, s' \in \mathcal{S}$ that are mapped to different abstracted states by (the more granular) abstraction β but that are mapped to the same abstracted state by (the more coarse) abstraction β'. Formally:

$$\exists s, s' \in \mathcal{S} \,.\, \beta(s) \neq \beta(s') \wedge \beta'(s) = \beta'(s')$$

Algorithm 1 is given a list of abstractions of decreasing granularity. The algorithm decides which action to take by choosing the most suitable abstraction at every time step. The chosen abstraction is the most granular one that provides a high confidence that the action with the highest estimated Q-value is actually the best one. Confidence of a state-action pair (s,a) is computed by running a t-test over the history $H(s,a)$ of values that the Q-table entry $Q(s,a)$ has assumed. The steps used to test the confidence of an abstraction are as follows:

1. the action a^* with the highest estimated Q-value is found;
2. the boundaries $\perp_a, \top_a$ of the confidence intervals of all actions a are calculated;
3. for all $a \neq a^*$, test whether $\perp_{a^*} > \top_a$: each test is passed with a $1 - \sigma$ confidence level;
4. if all the tests are passed, it is reasonable to assume that the true Q-value of a^* is actually the highest.

Optimization

The procedure described in Algorithm 1 has some significant inefficiencies. However, notice that they can be overcome and have been introduced in the listing for the purpose of clarity. In the follwing paragraphs, we briefly explain how to address these issues.

There are two main bottlenecks. The first is at line 12, where the union operation iterates over all the states to find the siblings. This adds a significant amount of computational time to compute the same information repeatedly. In fact, caching the state space partitioning (i.e. the sets of “siblings”) for each abstraction is a better solution. By doing so, the time complexity of retrieving such information is constant at the cost of a linear (in the number of states and abstractions) increase in memory complexity.

The second bottleneck is the procedure at line 13 which computes the confidence intervals. The procedure iterates over the whole set X_a^j of samples at every invocation to compute their mean and variance. An alternative, more efficient solution is to store mean and variance of the Q-value for every state-action pair and update them online [21]. It is, then, possible to efficiently compute mean and variance of a virtual union set by aggregating the stored statistics [22]. This modification makes storing the history of Q-values unnecessary, significantly reducing the space requirements of the algorithm.

Confidence driven exploration

While ϵ-greedy is expected to shrink the confidence intervals in the long run through random exploration, it has no awareness of their existence. It is reasonable to suppose that using a different exploration strategy making use of this knowledge would produce better results. Such a strategy would bias the exploration so as to perform actions whose confidence intervals are preventing the use of the next abstraction. We propose a variation on the traditional ϵ-greedy strategy that integrates such bias. The close-form definition is slightly cumbersome; so, with clarity in mind, we provide the pseudocode instead. The pseudocode in Algorithm 2 describes such procedure and is meant to replace line 22 of Algorithm 1.

Algorithm 2: C.I. driven exploration.

```
 1  { return $a^*$              with prob. $1 - \epsilon_{CI} - \epsilon_R$
    { return random($\mathcal{A}$)   with prob. $\epsilon_R$
    { go to line 11             with prob. $\epsilon_{CI}$
 2  if $j^* = 1$   // most granular abstraction already in use then
 3  |   return $a^*$   // no exploration needed
 4  end
 5  if $found$   // acceptable abstraction found then
 6  |   $\hat{j} \leftarrow j^* - 1$   // use it
 7  else
 8  |   $\hat{j} \leftarrow m$   // use the most coarse one
 9  end
10  $a^* \leftarrow \arg\max_{a \in \mathcal{A}} \overline{X_a^{\hat{j}}}$
11  $K_1 = \left\{ a \in \mathcal{A} \;\middle|\; \top_a^{\hat{j}} > \overline{X_{a^*}^{\hat{j}}} \right\}$
12  $K_2 = \left\{ a \in \mathcal{A} \;\middle|\; \overline{X_a^{\hat{j}}} > \bot_{a^*}^{\hat{j}} \right\}$
13  if $K_1 \neq \emptyset$ and $K_2 \neq \emptyset$ then
14  |   return random($K_1 \cup \{ a^* \}$)
15  else if $K_1 \neq \emptyset$ then
16  |   return random($K_1$)
17  else
18  |   return $a^*$
19  end
```

This procedure requires two parameters ϵ_{CI} and ϵ_R, which set the probability of exploring versus exploiting, similarly to ϵ-greedy. Unlike ϵ-greedy, however, this procedure performs a second type of exploration. That is, with probability ϵ_{CI}, it selects an action whose confidence interval needs to be reduced in order for the next abstraction to be usable.

Figure 1 shows a qualitative representation of the reasoning behind Algorithm 2. Depicted are the different possible situations in which confidence intervals of actions a^* (action with the highest mean) and a overlap. On top of each subfigure, the behavior of the algorithm in lines 11–12 is reported. Notice that in many cases there will be multiple actions matching the criteria of a: in such cases the choice is random among them. For the sake of concisenes, in the remainder of this section as well as in Algorithm 2 we will refer to the confidence interval of the estimate of the average Q-value of action a simpy as the confidence interval of a.

Line 11 of Algorithm 2 captures cases shown in figures 1a, 1b, 1e, 1f, 1g, where the upper bound of the confidence interval of some action a is higher than the average Q-value of the best action a^*. Such actions are stored in set K_1. Line 12 of Algorithm 2 captures cases shown in figures 1c, 1d, 1e, 1f, 1g, where the average Q-value of some action a is higher than the lower bound of the confidence interval of the best action a^*. Such actions are stored in set K_2. Algorithm 2 makes the simplifying assumption that further samples will shrink the confidence intervals without moving the average value. Notice that this does not introduce bias since the average value is an unbiased estimate of the mean value. With this assumption in mind, the procedure selects:

- a random action from K_1 if $K_1 \neq \emptyset$ and $K_2 = \emptyset$ because the only way to remove the overlaps of the confidence intervals is to shrink those of the actions in K_1;

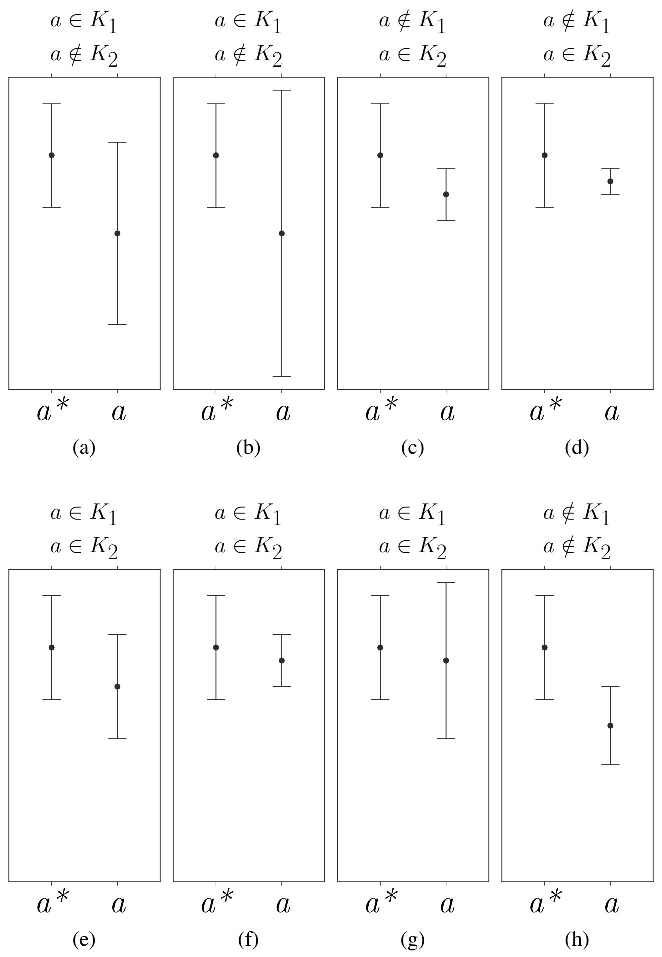


Figure 1: A visual qualitative representation of the possible situations in which confidence intervals overlap. Here, a^* is the action with the highest mean and a is another action. When confidence intervals overlap, the confidence level cannot be guaranteed. On top of each subfigure, the decision made by Algorithm 2 at lines 11–12 is indicated.

- a^* if $K_1 = \emptyset$ and $K_2 \neq \emptyset$ because, while choosing actions from K_2 would also be an effective way to remove the overlaps, choosing a^* is the choice with the highest expected future reward;
- a random action from $K_1 \cup \{a^*\}$ if $K_1 \neq \emptyset$ and $K_2 \neq \emptyset$ because of the same reasons explained above.

Notice that the case where $K_1 = \emptyset$ and $K_2 = \emptyset$ is only possible if the all the abstractions are usable, but this case is captured in lines 2–4. Following this strategy, confidence intervals that are overlapping will shink until they do not overlap anymore, therefore allowing the usage of the next abstraction, until the most fine-grained abstraction is usable.

4 Experiments

We evaluate the effectiveness of our algorithm using Pac-Man, a real-time arcade retro game[4]. Games of this type are of interest to the

[4] Additional information can be found at `http://www.gamasutra.com/view/feature/3938/the_pacman_dossier.php?print=1` (checked on March 3rd, 2016).

scientific Artificial Intelligence community due to the challenges of open-endedness and tight time-constraints they pose [23]. Pac-Man has been used as test-bed in a sizable amount of literature, including [24]–[26]. We adopted the implementation currently used in UC Berkeley to teach AI, originally developed by DeNero and Klein [27][5]. Our algorithms were implemented in Python using the Numpy library[6] [28] and parallelized using GNU Parallel[7] [29].

Pac-Man

In the game, the player controls Pac-Man, an agent moving in a two-dimensional environment whose purpose is to maximise a score. The score increases by collecting food pills while steering clear of ghosts, which will kill Pac-Man when colliding with it. Special capsules in the game area can be picked up by the player that make all the ghosts edible for a limited period of time. If Pac-Man collides with an edible ghost, the ghost dies and reappears at the center of the game area in a threatening (non-edible) state. The level topology used in the experiments is depicted in figure 2. This is not a standard Pac-Man level, but a simpler one provided with the codebase. We chose this because experiments require less computational time.

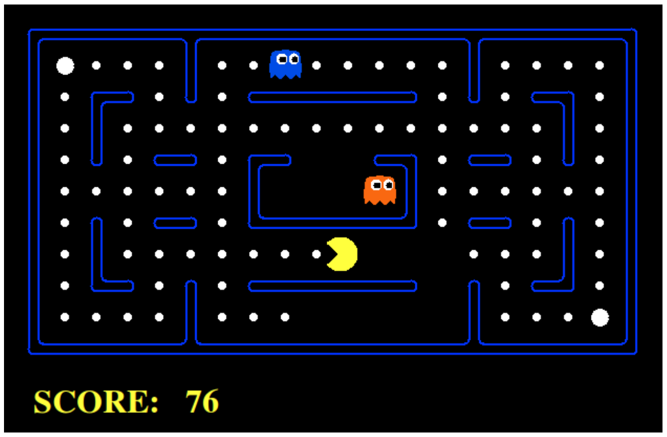


Figure 2: A screenshot of the video game used in the experiments, Pac-Man.

Players receive a score based on their performance. In the implementation we used, Pac-Man receives 10 points for each eaten pill, and it loses 1 point at each time step, while receiving 200 points every time a ghost is killed. Furthermore, 500 points are earned upon victory (i.e. when all the pills have been eaten), while 500 points are lost upon death. These scores have been chosen by the developers of the framework, and we adopt them without changes in this work.

Tests

The first question we wanted to answer is what significance setting σ yields the highest performance in Multi-Abstraction Q-learning (Algorithm 1) with ϵ_{CI} strategy. To answer this question, we tested the algorithm with $\sigma \in \{0.1, 0.2, 0.5, 0.9\}$. All the configurations used food and threatening ghosts information to describe states, successively adding edible ghosts information and, lastly, capsules information; these represents the abstractions β_i in Algorithm 1. In these tests, we set $\epsilon_{\mathrm{CI}} = 0.05$ and $\epsilon_{\mathrm{R}} = 0.05$.

Secondly, we wanted to evaluate whether shifting abstractions - from coarse to fine-grained - improves agents performance. To test this, we ran tests on Pac-Man using different agent algorithms:

- Q-learning where states included food and threatening ghosts information;
- Q-learning where states included food, threatening ghosts and edible ghosts information;
- Q-learning where states included food, threatening ghosts, edible ghosts and capsules information.

We compared the performance of these algorithms with those of the best configuration from the previous test, that with $\sigma = 0.9$. The exploration strategy used in these three configurations was ϵ-Greedy with $\epsilon = 0.1$.

We ran tests using each of these algorithms for 30000 consecutive episodes and we measured the reward collected during each of them. We repeated this 50 times and averaged the results. Agent performance is expected to improve over time, as they gather information on the environment: however, improvement rate and final performance depend upon the agent algorithm and its state representation.

The final question we wanted to answer is to what degree the performance of the other experiments are due to Multi-Abstraction Q-learning versus to the ϵ_{CI}-Greedy strategy. To test this, we ran experiments using the following algorithms:

- Multi-Abstraction Q-learning with ϵ-Greedy with $\epsilon = 0.1$;
- Q-learning with ϵ_{CI} with $\epsilon_{\mathrm{CI}} = 0.05$ and $\epsilon_{\mathrm{R}} = 0.05$.

We compared the performance of these algorithms with the performance of the best configuration in the first experiment, that with $\sigma = 0.9$. The features used in these experiments are the same used in the first set of experiments; that is, all of them: food, threatening/edible ghosts and capsules.

State space

Performance of MDPs are heavily influenced by the shape of the state space. In our experiments, the state space is the cartesian product of 8 features. Each of the features is related to objects in game; i.e., threatening/edible ghosts, pills and capsules. Features are either distance or direction information to such objects.

The "direction" feature specifies the direction that Pac-Man should follow to reach the closest object of the category. The direction information can assume five different values: one for each of the cardinal directions, plus an additional value used when there are no instances of the objects; e.g., if all the ghosts are edible, there is no threatening ghost. In the case of distance, the feature specifies $\lfloor \log_2 d \rfloor$, where d is the length of the shortest path to the closest object of the category. Shortest paths are computed by the Dijkstra algorithm for shortest path on graphs (see [30] for more information). Distance information can be null as well.

Annealing exploration

We compared different exploration strategies; i.e. ϵ-Greedy and our proposal, ϵ_{CI}-Greedy, showed in Algorithm 2. Even though we presented the naïve versions of the two strategies, in our experiments we used the simulated annealing version. This technique slowly decreases the amount of exploration as time progresses, so to gradually shift

[5] Currently available for download at `http://ai.berkeley.edu/project_overview.html`
[6] `http://www.numpy.org/`
[7] `http://www.gnu.org/software/parallel/`

from an exploration policy to the greedy policy over time. In ϵ-Greedy policies this is done by decreasing the value of ϵ. In ϵ_{CI}-Greedy, we similarly decrease both ϵ_{R} and ϵ_{CI}. The annealing schedule we chose is based on the sigmoid function, $s(x) = \frac{1}{1+e^x}$. Our schedule is defined as follows:

$$\epsilon(t) = \hat{\epsilon} \cdot s\left(u \cdot (m - t)\right),$$

where $\hat{\epsilon}$ is the maximum value for the exploration parameter, t is the current episode number, m is the desired center for the schedule and u controls the width of the function.

5 Results

In this section we discuss the results of the experiments we performed. All the figures in this section are smoothed using a moving average weighted by a Hanning function. The Hanning function is bell-shaped and smoothly zeroes at the edges. Using it to weight contributions in a moving window gives greater importance to central elements while still taking the surrounding element in account.

In the first experiment, different σ-values are compared in Multi-Abstraction Q-learning (Algorithm 1) using ϵ_{CI}-Greedy. The scores achieved by the different configurations are shown in Figure 3a. It is surprising that the lines dominating the chart are those using 0.5 and 0.9 as σ-values.

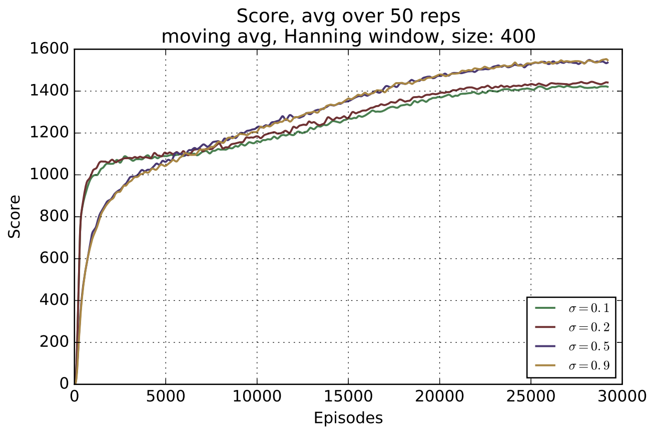
The most likely explanation for this is that 0.1 and 0.2 are too conservative values. While in normal t-tests values of 0.1 are unacceptably high, the trend here is heavily shifted. In fact, orthodox t-tests assume that the distributions are static over time. Here, however, (expected) Q-values veer from the common initial value towards their true values. For this reason, seemingly "premature" Q-values, which have a "high variance" from a t-test perspective, reliably estimate the best action.

Figures 4a and 4b show the percentage of decisions that have been made with each abstraction in successive episodes for the two configurations $\sigma = 0.2$ and $\sigma = 0.5$. There is a remarkable difference in that the former keeps using coarse abstractions throughout the learning process, while the latter barely uses any, except at the very early stages.

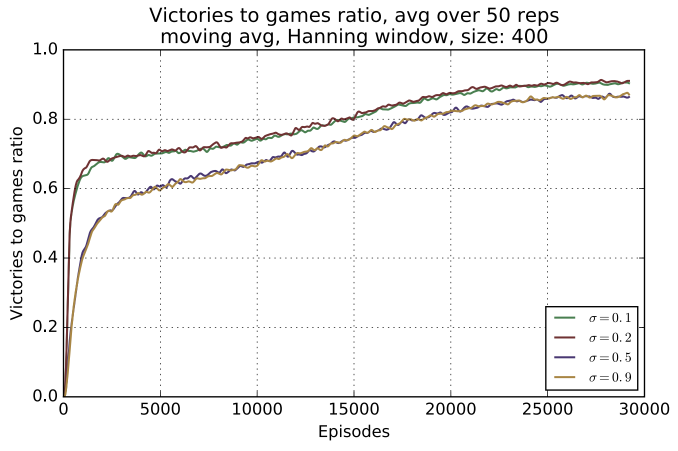
Figure 3b shows the percentage of victories for the configurations in the first experiment. The configurations winning the most often are $\sigma = 0.1$ and $\sigma = 0.2$. Considering the scores shown in Figure 3a this seems counterintuitive, because one would expect that the configurations with the highest scores are also those winning the most often. However, these numbers make sense when the structure of the game is considered: to maximize the score, Pac-Man needs to eat ghosts, but that poses an added risk in terms of winning/losing (i.e. if the ghost suddenly turns back to a threatening status). Figure 3c shows the average number of ghosts eaten in each successive episode: it can be observed that there is a significant difference between the configurations $\sigma = 0.1$ and $\sigma = 0.2$ and the configurations $\sigma = 0.5$ and $\sigma = 0.9$. The similarity of the trends showed in Figures 3c and 3a, where dominant configurations are $\sigma = 0.5$ and $\sigma = 0.9$ in both cases, supports this theory.

In the second experiment, our technique is compared with three configurations of Q-learning, each using an increasing amount of features. Figure 5 compares them to the best configuration of the first experiment, Multi-Abstraction Q-learning with $\sigma = 0.9$.

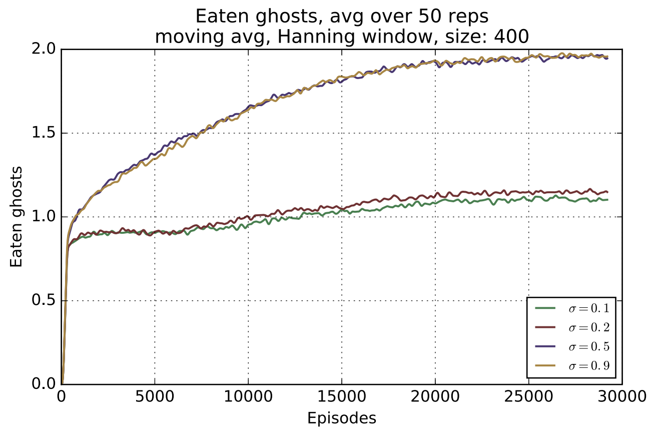
The curves show that the Q-learning configuration with the least features, at first, performs the best, showing that using more features at the beginning worsens the performance. However, this configuration is later surpassed by the Q-learning configuration using the intermediate amount of features, showing that having more features pays off when sparsity fades out. Finally, the Q-learning configuration using the most features surpasses both the other two Q-learning configurations. These trends show how, in normal Q-learning, more features produce better performance at later stages at the cost of performance in the early stages.



(a) Scores of the games

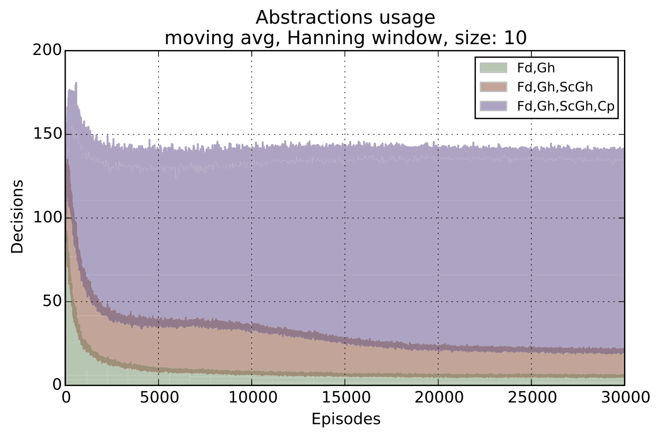


(b) Victories to total games ratio

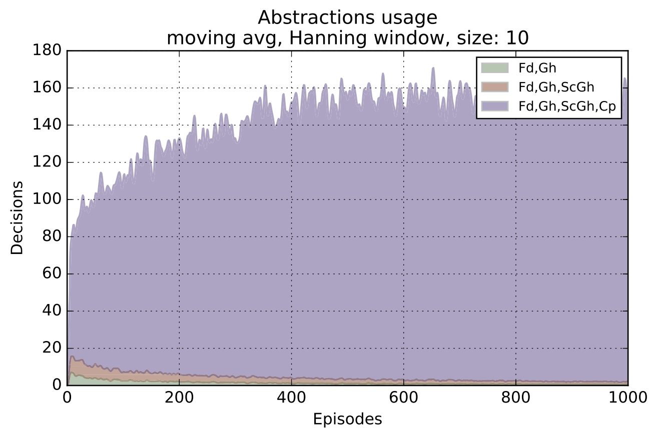


(c) Eaten ghosts

Figure 3: Data (y axis) per succesive episode (x axis) using Multi-Abstraction Q-learning with ϵ_{CI}-Greedy, varying significance parameter.



(a) $\sigma = 0.2$



(b) $\sigma = 0.5$

Figure 4: Decisions made with each abstraction (y axis) for each successive episode (x axis), using different values of significance in Multi-Abstraction Q-learning with ϵ_{CI}-Greedy. The second plot shows fewer episodes (x axis) because the more coarse abstractions quickly become almost unused.

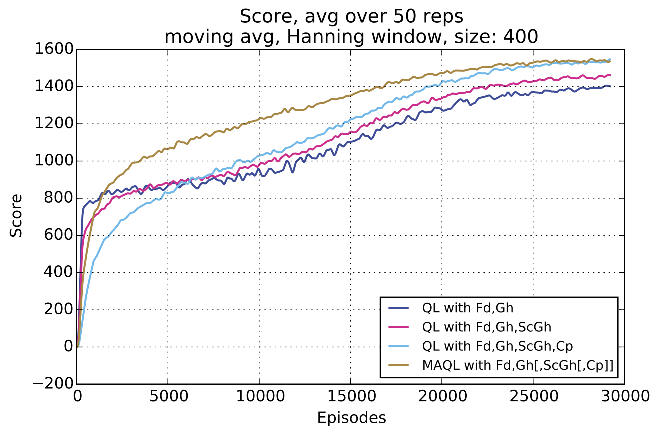


Figure 5: Score (y axis) per succesive episode (x axis) using Multi-Abstraction Q-learning with ϵ_{CI}-Greedy versus standard Q-learning with different sets of features.

Except at the very beginning of the process, Multi-Abstraction Q-learning produces significantly better results than the other configurations. Importantly, it also converges to the same values as the

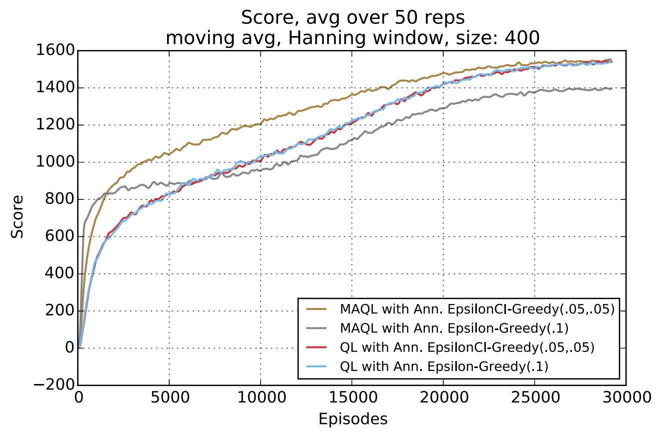


Figure 6: Score (y axis) per succesive episode (x axis) using Multi-Abstraction Q-learning with ϵ-Greedy versus ϵ_{CI}-Greedy.

Q-learning configuration using all of the features: this shows that the early improvement in performance does not come at the cost of later performance.

It could be argued that our approach can be replaced by predetermined rules. In fact, the intersection points of Q-Learning performance curves in Figure 5 provide a clear indication of when it is convenient to switch abstraction. This would produce better performance than any of the three Q-Learning agents. However, because our approach allows each state to be used at a different abstraction, it is more adaptive and produces far better performance during the learning phase, as shown in Figure 5.

It could also be argued that, since the final performance of Multi-Abstraction Q-learning is the same as that of standard Q-Learning, an agent might as well just use standard Q-Learning. While this is true, the advantage of Multi-Abstraction Q-learning is an improvement in performance during the learning phase as opposed to an improvement in final performance.

Finally, notice that Algorithm 1 has the same convergence guarantees of Q-learning [31]. This is because the set of features in use at each state is expanded to the full features set within a finite amount of time. Notice that Algorithm 2 guarantees at least the same amount of exploration of ϵ-greedy.

	MQL 🟫	QL w/4 🟦	QL w/3 🟪	QL w/2 🟦
Mean	1281.07	1152.11	1133.77	1099.25
Std. err	23.71	20.51	9.46	7.62

Table 1: Mean and standard error of the average reward per episode across the 50 runs of the experiments in Figure 5. Columns report values, respectively, for Multi-Abstraction Q-learning and for Q-Learning with features Fd,Gh,ScGh,Cp, Fd,Gh,ScGh and Fd,Gh.

The results of the third experiment are shown in Figure 6. The experiment shows that Multi-Abstraction Q-learning and ϵ_{CI}-Greedy do create a synergy in performance. On one side, ϵ_{CI}-Greedy does not seem to affect the performance of Q-learning; on the other side, Multi-Abstraction Q-learning without ϵ_{CI}-Greedy performs worse than Q-learning. However, when Multi-Abstraction Q-learning is used in concert with ϵ_{CI}-Greedy, they produce better performance than all other combinations.

Statistical analysis of Q-learning performance

The experiments showed that Multi-Abstraction Q-learning produces better performance than Q-Learning with any set of features. To quantitatively measure the performance of the different configurations, we computed the average reward per episode for each of the 50 runs of each of the 3 agents. This operation induces a distributions for each agent, all of which appear to be approximately normally distributed, as shown in Figure 7. Table 1 shows the means and standard errors of the data. Three two-samples t-tests (with unequal variance) have been executed to pairwise compare the agents sorted by average per-episode reward. All t-tests determined that the distributions mean are different with p-value < 0.001, therefore confirming that both agents using learned options perform significantly better than the agent that does not use options.

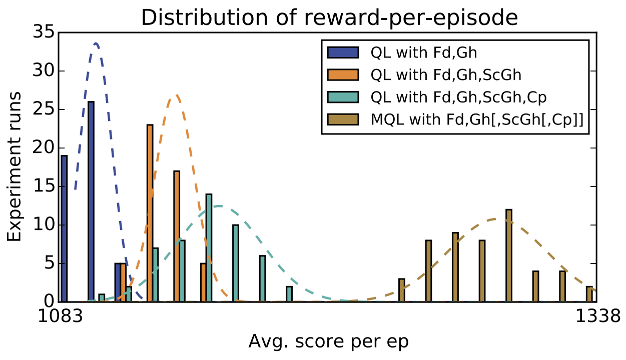


Figure 7: Histograms of the average reward per episode.

6 Conclusions

In this paper we presented a novel variation of Q-learning, which we name "Multi-Abstraction Q-learning". The algorithm we propose uses different state-abstractions for each state, increasing the level of detail over time. This allows the agent to overcome the initial sparsity in its utility estimates, typical of richer state representations. The agent can still make full use of the maximum level of detail later on in the learning process. Our experiments show that this algorithm produces better performance than standard Q-learning.

We also proposed a novel exploration strategy, ϵ_{CI}-Greedy. This strategy directs exploration to reduce sparsity of information, thereby allowing the agent to switch to more detailed abstractions earlier. Our experiments show that this strategy produces better results than standard ϵ-Greedy.

The proposed algorithm takes advantage of similarity in Q-values of similar states. In particular, factored state spaces where states in (hyper-)rectangular regions share similar values are a good fit for the algorithm. However, the algorithm just works with increasingly granular abstractions: it cannot take advantage of redundancy in regions defined by different sets of features. For example, in the case of Pac-Man, in states where a threatening ghost is nearby, food information has a low impact on Q-values, while in states where all threatening ghosts are distant, the direction from which they are likely to come has a low impact on Q-values.

To optimize the use of redundancy in these cases, the algorithm would need to consider multiple, separate sets of features. This means that the abstractions provided in input should be allowed to form a lattice, as opposed to just a list. In fact, while a list allows for only one "line of specialization", a lattice allows for more possibilities. In other words, the proposed algorithm does not have a choice in *what* information to add over time: it can solely choose *when* to add it. Using a lattice, different sets of features could be selected for different states and they would still all converge to the abstraction with all features in the end. This is currently the strongest limitation of the algorithm and is an important direction for future research.

One drawback of our approach is its reliance on meaningful features and, hence, domain knowledge. An interesting avenue of research is to investigate a combination of our approach and a general-purpose function approximation architecture, similar in spirit to [32].

It would also be interesting to combine temporal approaches [12]–[14] and/or adaptive tile coding approaches [17], [18] with the proposed approach in more complex domains.

Acknowledgements

The authors acknowledge support from the ARC Grant LP130100743. We would like to thank the VxLab RMIT for their support in this work.

References

[1] R. S. Sutton and A. G. Barto, *Introduction to Reinforcement Learning*, 1st. Cambridge, MA, USA: MIT Press, 1998.
[2] R. S. Sutton, "Learning to predict by the methods of temporal differences", *Machine learning*, vol. 3, no. 1, pp. 9–44, 1988.
[3] C. J. C. H. Watkins, "Learning from delayed rewards.", PhD thesis, University of Cambridge, 1989.
[4] D. Andre and S. J. Russell, "State abstraction for programmable reinforcement learning agents", in *AAAI/IAAI*, 2002, pp. 119–125.
[5] M. E. Taylor and P. Stone, "Transfer learning for reinforcement learning domains: A survey", *The Journal of Machine Learning Research*, vol. 10, pp. 1633–1685, 2009.
[6] J. N. Tsitsiklis and B. Van Roy, "An analysis of temporal-difference learning with function approximation", *Automatic Control, IEEE Transactions on*, vol. 42, no. 5, pp. 674–690, 1997.
[7] V. Tadi, "On the convergence of temporal-difference learning with linear function approximation", *Machine learning*, vol. 42, no. 3, pp. 241–267, 2001.
[8] N. K. Jong and P. Stone, "State abstraction discovery from irrelevant state variables.", in *IJCAI*, Citeseer, 2005, pp. 752–757.
[9] L. C. Cobo, P. Zang, C. L. Isbell Jr, and A. L. Thomaz, "Automatic state abstraction from demonstration", in *IJCAI Proceedings-International Joint Conference on Artificial Intelligence*, Citeseer, vol. 22, 2011, p. 1243.
[10] A. Hallak, D. Di-Castro, and S. Mannor, "Model selection in markovian processes", in *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, ACM, 2013, pp. 374–382.
[11] N. Jiang, A. Kulesza, and S. Singh, "Abstraction selection in model-based reinforcement learning", in *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, 2015, pp. 179–188.
[12] R. A. McCallum, "Overcoming incomplete perception with utile distinction memory", in *Proceedings of the Tenth International Conference on Machine Learning*, 1993, pp. 190–196.

[13] ——, “Instance-based utile distinctions for reinforcement learning with hidden state”, in *Proceedings of the Twelfth International Conference on Machine Learning*, Citeseer, 1995, pp. 387–395.

[14] R. A. McCallum, G. Tesauro, D. Touretzky, and T. Leen, “Instance-based state identification for reinforcement learning”, *Advances in Neural Information Processing Systems*, pp. 377–384, 1995.

[15] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, “Human-level control through deep reinforcement learning”, *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[16] J. S. Albus, *Brains, behavior, and robotics*. 1981.

[17] S. Whiteson, M. E. Taylor, P. Stone, *et al.*, *Adaptive tile coding for value function approximation*. Computer Science Department, University of Texas at Austin, 2007.

[18] P. Scopes and D. Kudenko, “Automated mixed resolution tiling”,

[19] G. W. Corder and D. I. Foreman, *Nonparametric statistics: A step-by-step approach*. John Wiley & Sons, 2014.

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

[21] B. Welford, “Note on a method for calculating corrected sums of squares and products”, *Technometrics*, vol. 4, no. 3, pp. 419–420, 1962.

[22] D. West, “Updating mean and variance estimates: An improved method”, *Communications of the ACM*, vol. 22, no. 9, pp. 532–535, 1979.

[23] P. Rohlfshagen and S. Lucas, “Ms pac-man versus ghost team cec 2011 competition”, in *Evolutionary Computation (CEC), 2011 IEEE Congress on*, Jun. 2011, pp. 70–77.

[24] M. Gallagher and A. Ryan, “Learning to play pac-man: An evolutionary, rule-based approach”, in *Evolutionary Computation, 2003. CEC ’03. The 2003 Congress on*, vol. 4, Dec. 2003, 2462–2469 Vol.4.

[25] D. Robles and S. M. Lucas, “A simple tree search method for playing ms. pac-man”, in *Computational Intelligence and Games, 2009. CIG 2009. IEEE Symposium on*, IEEE, 2009, pp. 249–255.

[26] S. Samothrakis, D. Robles, and S. Lucas, “Fast approximate max-n monte carlo tree search for ms pac-man”, *Computational Intelligence and AI in Games, IEEE Transactions on*, vol. 3, no. 2, pp. 142–154, 2011.

[27] J. DeNero and D. Klein, “Teaching introductory artificial intelligence with pac-man”, in *Proceedings of the Symposium on Educational Advances in Artificial Intelligence*, 2010.

[28] S. v. d. Walt, S. C. Colbert, and G. Varoquaux, “The numpy array: A structure for efficient numerical computation”, *Computing in Science & Engineering*, vol. 13, no. 2, pp. 22–30, 2011.

[29] O. Tange, “Gnu parallel - the command-line power tool”, *;login: The USENIX Magazine*, vol. 36, no. 1, pp. 42–47, Feb. 2011.

[30] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 2nd ed. The MIT Press, 2001.

[31] C. J. Watkins and P. Dayan, “Q-learning”, *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[32] C.-S. Chow and J. N. Tsitsiklis, “An optimal one-way multigrid algorithm for discrete-time stochastic control”, *IEEE transactions on automatic control*, vol. 36, no. 8, pp. 898–914, 1991.