Object Focused Q-learning for Autonomous Agents

Luis C. Cobo
College of Engineering
Georgia Tech
30332 Atlanta, GA, 30332
luisca@gatech.edu

Charles L. Isbell Jr.
College of Computing
Georgia Tech
30332 Atlanta, GA, 30332
isbell@cc.gatech.edu

Andrea L. Thomaz
College of Computing
Georgia Tech
30332 Atlanta, GA, 30332
athomaz@cc.gatech.edu

ABSTRACT

We present *Object Focused Q-learning* (OF-Q), a novel reinforcement learning algorithm that can offer exponential speed-ups over classic Q-learning on domains composed of independent objects. An OF-Q agent treats the state space as a collection of *objects* organized into different *object classes*. Our key contribution is a control policy that uses non-optimal Q-functions to estimate the risk of ignoring parts of the state space. We compare our algorithm to traditional Q-learning and previous arbitration algorithms in two domains, including a version of Space Invaders.

Categories and Subject Descriptors

I.2.6 [**Artificial Intelligence**]: Learning

Keywords

Reinforcement learning, task decomposition, state abstraction, modular RL.

1. INTRODUCTION

In our research, we aim to develop multi-purpose agents that can learn to perform a variety of real world tasks. One approach to this problem is to use reinforcement learning (RL); however, because of *the curse of dimensionality*, the time required for convergence in high-dimensional state spaces can make the use of RL impractical. There is no easy solution for this problem, yet humans are able to learn and perform new skills under the same circumstances by leveraging the fact that the dynamics of the world are not arbitrary.

Adult humans are consciously aware of only one stimulus out of each 3 hundred thousand received [22], and they can only hold a maximum of 3 to 5 meaningful items or chunks [1] in their short-term memory. This suggests that humans deal with high-dimensionality simply by paying attention to a very small number of features at once, a capability know as *selective atteintion*. Thus, the challenge to deal with complex domains is to decide which features to pay attention to, and shifting the focus of attention as needed.

With this inspiration, we have developed *Object Focused Q-learning* (OF-Q), a novel RL algorithm that efficiently solves a large class of high-dimensional real-world domains. Our algorithm assumes that the world is composed of a collection of independent objects that are organized into *object classes* defined by common behavior. We embrace the approach of paying attention to a small number of objects at any moment, so instead of a high-dimensional policy, we simultaneously learn a collection of low-dimensional policies along with when to apply each one, *i.e.,* where to focus at each moment.

Our algorithm is appropriate for many real-world scenarios, offers **exponential speed-ups over traditional RL** on domains composed of independent objects and is not constrained to a fixed-length feature vector. The setup of our algorithm includes aspects of both OO-MDPs [3] and modular reinforcement learning [13, 16]; however, we extend that body of work with **the key contribution of learning the Q-values of non-optimal policies to measure the consequences of ignoring parts of the state space**, and incorporating that knowledge in the control policy.

In the next section, we position our work among related approaches. We then briefly introduce our notation and in Section 4 explain OF-Q in detail. In Section 5 we highlight the differences between our arbitration scheme and previously proposed methods and in Section 6 we discuss the properties of our algorithm. In Section 7 we describe our test domains and experimental results, focusing on domains in which rewards obtained by interacting with different objects must be balanced with avoiding certain risks (as in the video game Space Invaders, depicted in Fig. 2(b)). Finally, we discuss our findings and present our conclusions.

2. RELATED WORK

The idea of using reinforcement learning on varying-length representations based on meaningful entities such as objects comes from the field of relational reinforcement learning [4, 19]. Typically, a human designer creates detailed task-specific representations of the state space in the form of high-level facts and relations that describe everything relevant for the domain. These approaches offer great flexibility, but encoding all this domain knowledge for complex environments is impractical and unsuitable for autonomous agents that cannot rely on an engineer to provide them with a tailored representation for any new task they may face. Object-oriented MDPs (OO-MDPs) [3] constitute a related but more practical approach that is closer to our own. Like OF-Q, OO-MDP solvers see the state space as a combination of objects of specific classes; however, OO-MDP solvers also need a designer to define a set of domain-specific *relations* that define how different objects interact with each other. While our approach is less expressive than OO-MDPs, it does not require specific domain knowledge. Additionally, OO-MDPs are a model-based approach while OF-Q is model-free.

RMDP [7] solvers make assumptions similar to ours, but require a full dynamic Bayesian network. Fern proposes a relational policy iteration algorithm [5], but it relies on a resettable simulator to perform Monte-Carlo rollouts and also needs either an initial policy that performs well or a good cost heuristic of the domain. Our approach does not have any of these requirements and relies solely on on-line exploration of the domain.

In OF-Q, each object produces its own reward signal and the algorithm learns an independent Q-function and policy for each object class. This makes our algorithm similar to modular reinforcement learning, even though we have different goals than modular RL. Russell & Zimdars [13] take into account the whole state space for the policy of each module, so they can obtain global optimality, at the expense of not addressing the dimensionality problems that we tackle. Sprague & York [16] use different abstractions of the state space for the different module policies, but because they use the SARSA algorithm to avoid the so-called *illusion of control*, they can no longer assure local convergence for the policy of each individual module. In OF-Q, as we explain in Section 4, we take a different approach to avoid this problem: for each object class, we learn the Q-values for optimal and non-optimal policies and use that information for the global control policy. Because we use Q-learning to learn class-specific policies, we can assure their convergence. In Section 7.3, we show that our control policy performs better than the basic command arbitration of modular RL algorithms.

Many other methods leverage different models of the world that are not based on objects. Some of these approaches rely on a hierarchical domain-specific structure that has to be specified by a system designer [2] or derived from heuristics that apply only to restricted classes of domains [8, 15, 17]. An alternative to hierarchies are local models [18], but these models require histories and tests of interest, a very domain specific knowledge that must also be specified by a system designer.

Other works have a set of candidate representations out of which one is chosen for a specific task [12, 14], but this is impractical in high-dimensional domains if there is no previous knowledge about which features are useful together. In addition, these approaches use a single representation for the whole state space instead of shifting the attention focus dynamically as our algorithm does. U-tree [9] can find its own representations, but it requires too many samples to scale well in realistic domains. There are also promising algorithms based on intrinsically motivated learning [20], but so far these are restricted to domains where the agent can control all the variables of the state space.

3. NOTATION

A Markov decision process (MDP) is a tuple

$$M = (S, A, P, R, \gamma)$$

with $S = F_1 \times \cdots \times F_n$ a finite state space with n features, A a finite set of available actions, $P = Pr(s'|s,a)$ the probability of transitioning to state s' when taking action a in state s, $R = r(s,a)$ the immediate reward when taking action a in state s, and $0 \leq \gamma < 1$ the discount factor.

A *policy* $\pi : S \rightarrow A$ defines which action an agent takes in a particular state s.

$$V^{\pi}(s) = r(s, \pi(s)) + \gamma \sum_{s' \in S} Pr(s'|s, \pi(s)) V^{\pi}(s')$$

is the state-value of s when following policy π, *i.e.,* the expected sum of discounted rewards that the agent obtains when following policy π from state s.

$$Q^{\pi}(s,a) = r(s,a) + \gamma \sum_{s' \in S} Pr(s'|s,a) Q^{\pi}(s', \pi(s'))$$

is the discounted reward received when choosing action a in state s and then following policy π. π^* is the optimal policy maximizing the value of each state. $V^*(s) = V^{\pi^*}(s)$ and $Q^*(s,a) = Q^{\pi^*}(s,a)$ are then the optimal state value and Q-values. Note that the state-values and Q-values are always defined with respect to a given policy that is not necessarily optimal.

4. OBJECT FOCUSED Q-LEARNING

OF-Q is designed to solve episodic MDPs with the following properties:

- The state space S is defined by a variable number of *independent objects*. These objects are organized into *classes* of objects that behave alike.
- The agent is seen as an object of a specific class, constrained to be instantiated exactly once in each state. Other classes can have none to many instances in any particular state.
- Each object provides its own reward signal and the global reward is the sum of all object rewards.
- Reward from objects can be positive or negative[1]. The objective is to maximize the discounted sum of reward, but we see negative rewards as a punishments that should be avoided, *e.g.,* being eaten by a ghost in a game of Pacman or shot by a bullet in Space Invaders. This construction can be seen as modeling safety constraints in the policy of the agent.

These are reasonable assumptions in problems that autonomous agents face, *e.g.,* a robot that must transport items between two locations while recharging its batteries when needed, keeping its physical integrity, and not harming humans in its surroundings.

For our experiments, we have chosen two domains in the mold of classic videogames, as we detail in Sec. 7.1. In these domains, an object representation can be easily constructed from the screen state using off-the-shelf vision algorithms. Also, because objects typically provide rewards when they come in contact with the agent, it is also possible to automatically learn object rewards by determining which object is responsible for a change in score.

[1]This convention is not necessary for the algorithm. We could have only positive rewards and interpret low values as either punishments or just poor rewards; however, to make the explanation of the algorithm more straightforward we will keep the convention of considering negative rewards as punishments to avoid.

4.1 Object Focused MDPs

We formalize an OF MDP:

$$M_{\text{OF}} = (S, A, \{P_c\}_{c\in C}, \{R_c\}_{c\in C}, \gamma)\,,$$

where C is the set of object classes in the domain. A state $s \in S$ is a variable-length collection of objects $s = \{o_a, o_1, \ldots, o_k\}$, $k \geq 1$ that always includes the agent object o_a. Each object o can appear and disappear at any time and has 3 properties:

- Object class identifier `o.class` $\in C$.
- Object identifier `o.id`, to track the state transitions of each object between time-steps.
- Object state `o.state` $= \{f_1, \cdots, f_n\}$, composed of a class-specific number of features.

There is a separate transition model P_c and reward model R_c for each object class c. Each of these models takes into account only the state of the agent object o_a and the state of a single object of the class c.

Our model differs greatly from OO-MDPs, despite working with similar concepts. In OO-MDPs, besides *objects* and *classes*, the designer must provide the learning algorithm with a series of domain-specific *relations*, which are Boolean functions over the combined features of two object classes that represent significant events in the environment, and *effect types*, which define how the features of a specific object can be modified, *e.g.,* increment feature by one, multiply feature by two or set feature to 0. Furthermore, there are several restrictions on these effects, for example, for each action and feature only effects of one specific type can take place. OF-Q does not require this additional information.

4.2 Algorithm overview

4.2.1 Q-value estimation

Q-learning is an off-policy learning algorithm, meaning that Q-values for a given policy can be learned while following a different policy. This allows our algorithm to follow any control policy and still use each object o present in the state to update the Q-values of its class `o.class`. For each object class $c \in C$, our algorithm learns Q^*_c, the Q-function for the **optimal policy** π^*, and Q^R_c, the Q-function for the **random policy** π^R. These Q-functions take as parameters the agent state $\texttt{o}_a\texttt{.state}$, the state of a single object o of class c `o.state` and an action a. For clarity, we will use s_o to refer the tuple $(\texttt{o}_a\texttt{.state, o.state})$ and will omit the class indicator c from Q-function notation when it can be deduced from context.

With a learning rate α, if the agent takes action a when an object o is in state s_o and observes reward r and next state s'_o, the Q-value estimate for the optimal policy of class *o.class* is updated with the standard Q-learning update

$$\hat{Q}^*(s_o, a) = (1-\alpha)\,\hat{Q}^*(s_o, a) + \alpha\left(r + \gamma \max_{a'\in A} \hat{Q}^*(s'_o, a')\right), \tag{1}$$

where the hat denotes that this is an estimate of the true Q^*.

Algorithm 1 Object Focused Q-learning algorithm.

```
for c ∈ C do
  Initialize Q̂*_c
  Initialize Q̂^R_c
  Initialize threshold τ_c
end for
T ← {τ_c}_{c∈C}
Q̂* ← {Q̂*_c}_{c∈C}
Q̂^R ← {Q̂^R_c}_{c∈C}
loop
  for c ∈ C do
    Q̂*_control ← Q̂*
    Q̂^R_control ← Q̂^R
    candidates ←GetCandidates(T, c)
    stats ← []
    for T' ∈ candidates do
      candidate_reward ← 0
      for i ← 1 to n_evaluations do
        episode_reward ← 0
        Observe initial episode state s
        repeat
          𝔸 ← GetSafeActions(s, Q̂^R_control, T')
          a ← ε-greedy(s, Q̂*_control, 𝔸)
          Take action a
          Observe new state s and reward r
          Update Q̂*, Q̂^R
          Update episode_reward
        until End of episode
        candidate_reward+ = episode_reward
      end for
      stats[T'] ← candidate_reward
    end for
    τ_c ← UpdateThresholds(stats, candidates)
  end for
end loop
```

Using the same sample, the Q-value estimate for the random policy of class *o.class* is updated with

$$\hat{Q}^R(s_o, a) = (1-\alpha)\,\hat{Q}^R(s_o, a) + \alpha\left(r + \gamma \frac{\sum_{a'\in A} \hat{Q}^R(s'_o, a')}{|A|}\right). \tag{2}$$

4.2.2 Control policy

The control policy that we use is simple. We first decide $\mathbb{A}$, the set of actions that are safe to take,

$$\mathbb{A} = \{a \in A | \forall o \in s, \hat{Q}^R\,(\texttt{o.state}, a) > \tau_{\texttt{o.class}}\}, \tag{3}$$

where $\tau_{\text{o.class}}$ is a per-class dynamic threshold obtained as described in Sec. 4.3. The set of all thresholds is $T = \{\tau_c\}_{c\in C}$. The control policy then picks the action $a \in \mathbb{A}$ ($a \in A$ if $\mathbb{A} = \emptyset$) that returns the highest Q-value over all objects,

$$\pi(s)_{\text{OF}} = \arg\max_{a\in\mathbb{A}} \max_{o\in s} \hat{Q}^*\,(\texttt{o.state}, a)\,.$$

During learning, we use an ϵ-greedy version of this control policy.

4.3 Risk threshold and complete algorithm

The structure of our algorithm is shown in Algorithm 1. We assume that the number of object classes is known in

advance, but the algorithm can be extended to handle new classes as they appear. The outer loop determines the safety threshold set candidates and runs `n_evaluations` episodes with each candidate to compare their performance to that of he current set of thresholds T. At each time-step of each episode, our algorithm makes an update to $\hat{Q}^*_{\texttt{o.class}}$ and $\hat{Q}^R_{\texttt{o.class}}$ per each object o in the state s using the update rules Eq. 1 and Eq. 2. The policies used for control are only refreshed when the thresholds are updated, so that the threshold performance estimation is not affected by changing Q-values. `GetSafeActions` is implemented using Eq. 3. In the next sections, we complete the details about threshold initialization and threshold updates.

4.3.1 Threshold initialization

To avoid poor actions that result in low rewards, we initialize the threshold for each class as a fraction of $Q_{\min}$, the worst possible Q-value in the domain[2]. For a domain with a minimum reward $r_{\min} < 0$ and discount factor γ, $Q_{\min} = \frac{r_{\min}}{(1-\gamma)}$. In our test domains, negative rewards are always generated by terminal states, so we assume $Q_{\min} = r_{\min}$.

4.3.2 Threshold updating

OF-Q thresholds are optimized with a hill climbing algorithm. Starting with a threshold set T, the algorithm iteratively picks each of the available classes $c \in C$ and evaluates two neighbors of T in which the threshold τ_c is slightly increased or decreased. These three candidates are the output of the function `GetCandidates` in Algorithm 1. We have empirically found that a variation factor of 10% from the current threshold works well across different domains. The algorithm runs `n_evaluations` episodes with the current threshold set T and each of the two neighbors to compute the expected reward with each candidate. Then, the threshold τ_c is updated with the value that performs the best.

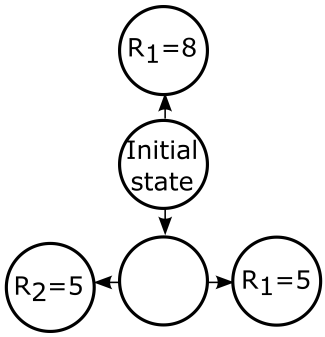
5. BENEFITS OF THE ARBITRATION

Our main contribution is a control policy that estimates the risk of ignoring dimensions of the state space using Q-values of non-optimal policies. In this section we explain the benefits of this arbitration.

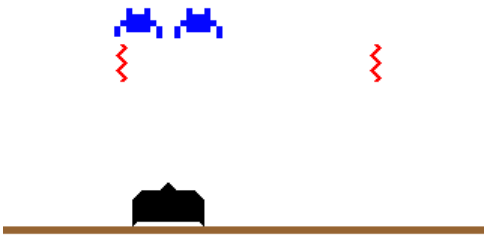
The concept of modules in modular RL literature [16] can be compared to object classes that are always instantiated once and only once in each state; modules never share a policy. This is due, in part, to modular RL aiming to solve a different type of problem than our work; however, modular RL arbitration could be adapted to OF-Q, so we use it as a baseline.

Previous modular RL approaches use a simple arbitration directly derived from the Q-values of the optimal policy of each module. The two usual options are *winner-takes-all* and *greatest-mass*. In *winner-takes-all*, the module that has the highest Q-value for some action in the current state decides the next action to take. In *greatest-mass*, the control policy chooses the action that has the highest sum of Q-values across all modules.

Winner-takes-all is equivalent to our OF-Q control policy with all actions being safe, $\mathbb{A} = A$. The problem with this approach is that it may take an action that is very positive for one object but fatal for the overall reward. In the Space Invaders domain, this control policy would be completely blind to the bombs that the enemies drop, because there will always be an enemy to kill that offers a positive Q-value, while bomb Q-values are always negative. *Greatest-mass* is



(a) With these two sources of reward, *greatest-mass* would choose the lower state, expecting a reward of 10. The optimal action is going to the upper state.



(b) For the pessimal Q-values, both bombs are just as dangerous, because they both can possibly hit the ship. Random policy Q-values will identify the closest bomb as a bigger threat.

Figure 1: Arbitration problems.

problematic due to the *illusion of control*, represented in Fig. 1(a). It does not make sense to sum Q-values from different policies, because Q-values from different modules are defined with respect to different policies, and in subsequent steps we will not be able to follow several policies at once.

In our algorithm, the control policy chooses the action that is acceptable for all the objects in the state and has the highest Q-value for one particular object. To estimate how inconvenient a certain action is with respect to each object, we learn the random policy Q-function Q^R_c for each object class c. Q^R_c is a measure of how dangerous it is to ignore a certain object. As an agent iterates on the risk thresholds, it learns when the risk is too high and a given object should not be ignored.

It would be impossible to measure risk if we were learning only the optimal policy Q-values Q^*. The optimal policy Q-values would not reflect any risk until the risk could not be avoided, because the optimal policy can often evade negative reward at the last moment; however, there are many objects in the state space and at that *last moment* a different object in the state may introduce a constraint that prevents the agent from taking the evasive action. Learning Q-values for

[2]In the case where only positive rewards are considered, the initial value for the thresholds would be a value in-between $Q_{\min}$ and $Q_{\max}$

the random policy allows us to establish adequate safety margins.

Another option we considered was to measure risk through the *pessimal policy*, *i.e.,* the policy that obtains the lowest possible sum of discounted rewards. This policy can be learned with the update rule

$$\hat{Q}^P(s,a) = (1-\alpha)\,\hat{Q}^P(s,a) + \alpha\left(r + \gamma \min_{a' \in A} \hat{Q}^P(s',a')\right).$$

The pessimal policy offers an upper bound on the risk that an object may pose, which can be useful in certain scenarios; however, this measure of risk is not appropriate for our algorithm. According to the pessimal policy the two bombs depicted in Fig. 1(b) are just as dangerous because both could possibly hit the agent; however, if we approximate the behavior of the agent while ignoring that bomb as a random walk, it is clear that the bomb to the left poses a higher risk. The Q-values of the random policy correctly convey this information.

In our algorithm, as well as in *winner-takes-all*, the *illusion of control* is not a problem. In Space Invaders, for example, the agent will target the enemy that it can kill fastest while staying safe, *i.e.,* not getting too close to any bomb and not letting any enemy get too close to the bottom of the screen. If the enemy that can be killed the fastest determines the next action, in the next time-step the same enemy will be the one that can be killed the fastest, and therefore the agent will keep focusing on that enemy until it is destroyed.

6. OF-Q PROPERTIES

6.1 Class-specific policies

THEOREM 1. *Let $(S, A, \{P_c\}_{c\in C}, \{R_c\}_{c\in C}, \gamma)$ be an OF MDP. $\forall c \in C$, OF-Q Q-function estimates $\hat{Q}_c^*$ $\hat{Q}_c^R$ converge to the true Q-functions Q_c^* Q_c^R.*

PROOF. Q-learning converges with probability 1 under the condition of bounded rewards and using, for each update t, a step size α_t so that $\sum_t \alpha_t = \infty$, $\sum_t \alpha_t^2 < \infty$ [21]. Given our independence assumption and that Q-learning is an off-policy algorithm, Q-values can be learned using any exploration policy. If we see the domain as a different MDP executed in parallel for each object class c, the same convergence guarantee applies. Several objects of the same class simultaneously present can be seen as independent episodes of the same MDP. The convergence proof applies too for Q-values of the random policies by changing the maximization by an average over actions. □

THEOREM 2. *Let $(S, A, \{P_c\}_{c\in C}, \{R_c\}_{c\in C}, \gamma)$ be an OF MDP and (S, A, P, R, γ) the equivalent traditional MDP describing the same domain with a single transition and reward function P and R. All OF MDP class-specific Q-functions will converge exponentially faster, in samples and computation, with respect to the number of objects in the domain, than the MDP Q-function.*

PROOF. Q-learning has a sample complexity $O(n \log n)$ with respect to the number of states in order to obtain a policy arbitrarily close to the optimal one with high probability [10]. Without loss of generality, assuming a domain with m objects of different classes, each with k possible states, the sample complexity of Q-learning on the global MDP would be $O(k^m \log k^m)$, while the sample complexity of each OF-Q class-specific policy would be only $O(k \log k)$.

Regarding computational complexity, assuming the same cost for each Q-update (even though updates on the whole MDP will be more expensive), OF-Q would take m times longer per sample, as it has to update m different policies for each sample. This makes OF-Q computational complexity linear in m, while Q-learning computational complexity is exponential in m, because the sample complexity is already exponential and all samples have at least a unit cost. □

Note that the speed-ups come from considering each object independently, and not from organizing the objects in classes that share a policy. The organization of objects in classes provides and additional speed-up: if there are l objects of each class on the environment, for example, each sample will provide l updates to each Q-function and the sample complexity would be additionally reduced by a factor of l.

6.2 Risk thresholds

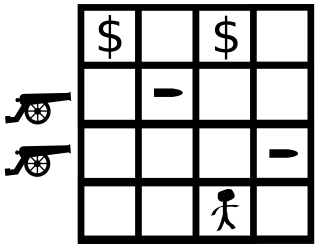
Besides deriving the appropriate class-specific Q-functions, OF-Q needs to find an appropriate set of class-specific risk thresholds. As discussed in Section 4.3.2, these thresholds are determined with a hill climbing algorithm using the expected reward as the objective function to optimize. The episodes used for computing the Q-functions are reused for these optimizations, and therefore, if the sample complexity for learning the Q-functions dominates the sample complexity for learning the thresholds, the total OF-Q sample and computational complexity (threshold updating costs are negligible) would be equal to the one for learning Q-functions. In this case, our algorithm would provide exponential speed-ups over traditional Q-learning. Experimentally we observe that in even moderately sized problems such as space-invaders, the time needed to learn the Q-functions dominates over the time needed to converge on thresholds.A definite answer to this question would require a complexity analysis of hill climbing, which is outside of the scope of this work.

Given the stochastic nature of our objective function, we can use a hill climbing variant such as PALO [6] to ensure the convergence of thresholds to a local optima. To find a global optima for the set of thresholds, simulated annealing [11] can be used instead. Unfortunately, PALO and simulated annealing are known to be slower than traditional hill climbing, which has also proved to work well in our experimental domains. Due to these factors, we have chosen a simple hill climbing algorithm for the first implementation of OF-Q.

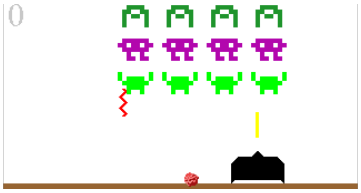
7. EXPERIMENTAL SETUP

7.1 Domains

We tested OF-Q in two different domains. The first one, which we call *Normandy*, is a 10x20 gridworld where an agent starts in a random cell in the bottom row and must collect two prizes randomly placed in the top row. At each time-step, the agent can stay in place or move a cell up, down, left or right. Additionally, there are cannons to the left of rows 3 and 4 which fire bombs that move one cell to the right every time-step. Each cannon fires with a prob-



(a) Normandy. The agent starts in a random cell in the bottom row and must collect the two rewards randomly placed at the top, avoiding the cannon fire. In our simulations the grid size is 10x20 with cannons on the third and fourth row.



(b) Space Invaders. In such a high-dimensional state space, traditional reinforcement learning fails to converge in any reasonable amount of time.

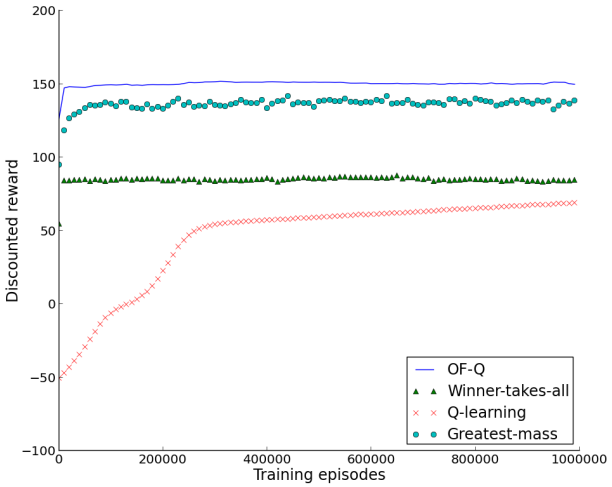
Figure 2: Domains.

ability 0.50 when there is no bomb in its row. The agent receives a reward $r = 100$ for collecting each prize and a negative reward $r = -100$ if it collides with a bomb, and the episodes end when all the rewards are collected or when a bomb hits the agent. Fig. 2(a) shows a representation of a reduced version of this domain.

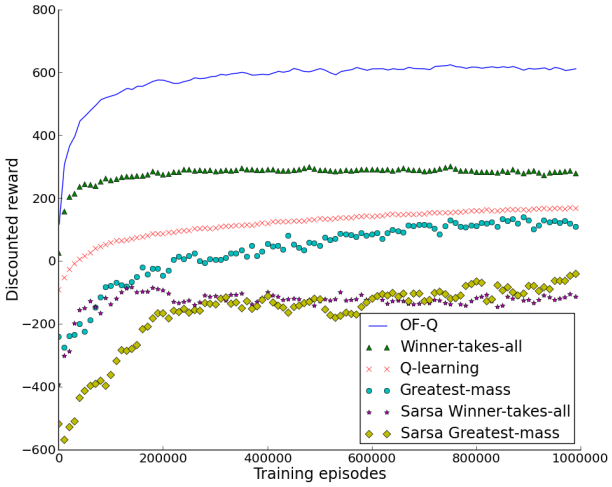
Our second domain is a full game of Space Invaders, shown in Fig. 2(b). The agent controls the ship which, at each time-step, may stay in place or move left or right. At the same time, the agent may fire or not, so there are six possible actions. Firing will only work if there is no bullet on the screen at the moment of taking the action. The agent object is thus defined by the x position of the ship and the x and y position of the bullet. The world has two object classes, namely, enemies and bombs, each one defined by their x and y positions and, in the case of enemies, direction of movement. Initially there are 12 enemies and each one may drop a bomb at each time-step with a probability 0.004. The agent receives a positive reward $r = 100$ for destroying an enemy by hitting it with a bullet, and a negative reward $r = -1000$ if a bomb hits the ship or an enemy reaches the bottom of the screen. The game ends when the agent is hit by a bomb, an enemy reaches the bottom or the agent destroys all enemies.

7.2 Baselines

We compare our algorithm with traditional Q-learning (no arbitration), and four other baselines. Two of them are variants of our algorithm using *winner-takes-all* or *greatest-mass* arbitration. These two variants still see the world as composed of objects of different classes and use Q-learning to find an optimal policy for each class, but due to the simpler nature of their arbitration methods, it is not necessary to learn the random-policy Q-values nor to find a set of risk thresholds. The other two baselines are variants of the algorithms proposed by Sprague & York [16]. These two variants



(a) Normandy domain.



(b) Space Invaders domain

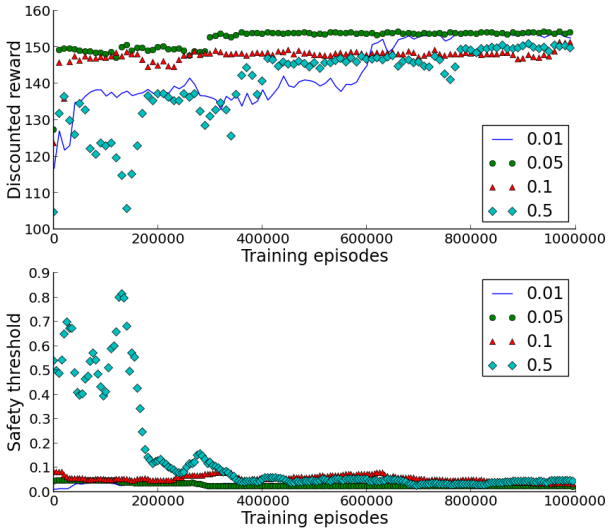
Figure 3: Performance vs. training episodes, averaged over 10 runs.

are identical to the previous two, but use SARSA instead of Q-learning update rule to avoid the illusion of control.

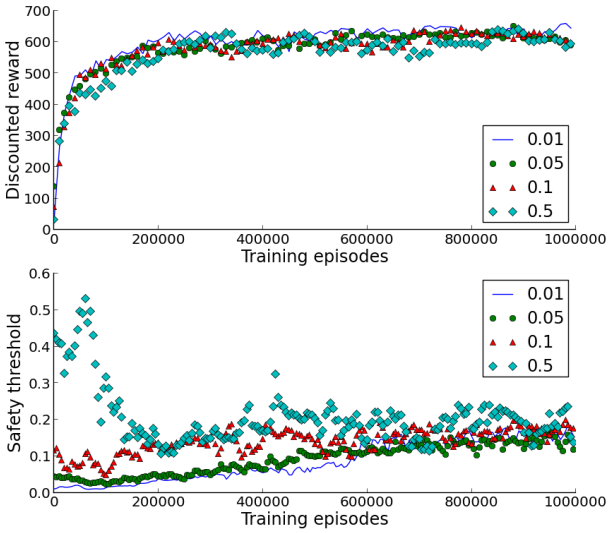
All algorithms and domains use a constant learning rate $\alpha = 0.25$, discount factor $\gamma = 0.99$, and an ϵ-greedy control policy with $\epsilon = 0.05$. For OF-Q, we use 100 evaluation episodes per threshold candidate and an initial threshold of $0.05 \cdot Q_{\min}$ for each class, unless otherwise stated.

7.3 Results

In Fig. 3 we compare the results of our algorithm and baselines on the Normandy and the Space Invaders domain. Our algorithm performs better in both domains, with a larger advantage in the more complex Space Invaders domain. Note that the performance of Q-learning on the Space Invaders domain keeps improving, but at a very slow rate. After 5 million episodes the average discounted reward was still



(a) Normandy domain.



(b) Space Invaders domain

Figure 4: Performance and threshold for bomb objects for single runs with different initial thresholds. The thresholds are expressed as fractions of the minimum Q-value of the domain $Q_{\min}$.

below 250. In the Normandy domain, the upward slope of Q-learning is more obvious, and in the 5x5 version of the domain it did converge within the first million training episodes.

We do not show the results for the SARSA algorithms on the Normandy domain because the algorithms would not make progress in days. The algorithm learned quickly how to avoid the cannon fire but not how to pick the rewards, and therefore the episodes were extremely long. In Space Invaders, these algorithms are the worst performers. These results are not surprising because SARSA is an on-policy algorithm and does not offer any convergence guarantees in these settings. If each object class policy was observing the whole state space, the SARSA policies would converge [13], but we would not get the scalability that OF-Q offers, since these come from considering a set of low-dimensional policies instead of a high-dimensional one.

The performance of *greatest-mass* varies a lot between each domain, being a close second best option in the Normandy domain and the worst option in Space Invaders. We believe this is due to the *illusion of control* that we discussed in Section 5. The Normandy domain is not affected by this problem, as there are only two sources of positive reward and, at each time-step, the agent will go towards the reward that is closest. However, the case for Space Invaders is more complicated. Initially, there are 12 enemies in the domain, all of them a source of reward. The agent can only fire one bullet at a time, meaning that after it fires it cannot fire again until the bullet hits an enemy or disappears at the top of the screen. At each time-step, firing is recommended only by one object, the closest enemy to kill; however, the best action for all the other enemies is to not fire yet. Because of the discount factor, the Q-value for the optimal action (fire) for the closest enemy is greater than each of the Q-values for the optimal actions of the rest of the enemies (do not fire yet), but there are more enemy objects whose Q-values recommend not to fire. This causes the ship to never fire, because firing at a specific enemy means losing the opportunity to fire on other enemies in the near future.

We ran our algorithm with different initial thresholds to test its robustness. We can see in Fig. 4 that the thresholds for the bombs in both domains converge to the same values, and so does performance, even though there are some oscillations at the beginning when Q-values have not yet converged. Starting with a large value of $0.5 \cdot Q_{\min}$ seems to be a bad option because even if the policy derived from the Q-values is already correct, the Q-values themselves may still be off by a scaling factor, leading to an ineffective threshold. Nonetheless, even this particularly bad initial value ends up converging and performing as well as the others.

8. DISCUSSION

OF-Q works better than previous arbitration algorithms because it is able to learn which actions are acceptable for all sources of reward, so that it can subsequently choose to be greedy safely. Imagine a domain where there is a big reward with a pit on the way to it, so that trying to get the reward will actually cause the agent to end the episode with a large negative reward. The random policy Q-values with respect to the pit will reflect this possible outcome, even if it takes a very specific set of actions to reach the pit. The closer in time-steps that the agent gets to the pit, the more likely it is that a random policy would lead to the pit and the more negative the Q-values will be. By learning a risk threshold, OF-Q will be able to keep a reasonable safety margin and learn that it is never worth falling into the pit even if there is a big reward ahead. *Winner-takes-all*, on the other hand, would be completely blind to the pit and even *greatest-mass* would still fall in the pit if the positive reward is large enough, even though it will never be reached.

Function approximation can improve RL performance, but in practice, it is often necessary to manually engineer a set of domain-specific features for function approximation to per-

form well. In many cases, such engineering is a non-trivial task that may essentially solve the abstraction problem for the agent. Still, we expect that using function approximation for the class-specific Q-functions can make OF-Q scale to larger problems in future work. In such work the goal will be to leverage function approximation while minimizing the need to hand-engineer domain-specifc features.

One limitation of the current formulation of OF-Q is that it cannot learn about interactions between two non-agent objects; however, we believe this is a reasonable trade-off for the benefit of not requiring a system designer to provide domain specific information (e.g., OO-MDPs). Moreover, the class of problems concerned only with agent-object interactions is quite large, for example, it covers all of the domains used by Diuk et al. [3] as well as many videogames and real world tasks; nonetheless, we plan to extend OF-Q in future work to account for interactions between different non-agent objects. To do that, we will look at pairs of objects to see if their behavior can be described more accurately when considered together. This way, the complexity of the learned policies will match the real complexity of the domain.

9. CONCLUSIONS

In this paper we introduced *Object Focused Q-learning* (OF-Q), a learning algorithm for autonomous agents that offers exponential speed-ups over traditional RL in domains where the state space is defined as a collection of independent objects. OF-Q requires less domain knowledge than earlier relational algorithms, being therefore better suited for use by multipurpose autonomous agents. We proposed a novel arbitration approach that is based on learning the Q-functions with respect to non-optimal policies to measure the risk that ignoring different dimensions of the state space poses and we explained why this arbitration performs better than earlier alternatives. Using two videogame domains, including a version of Space Invaders, we show that our algorithm indeed performs significantly better than previously proposed approaches. In future work, we plan to extend our algorithm to handle more complex domains where different non-agent objects may interact with each other and to study the interactions of OF-Q with function approximation.

10. ACKNOWLEDGMENTS

This work is supported by ONR Grant No. N000141210483.

11. REFERENCES

[1] Cowan, N. The magical mystery four: How is working memory capacity limited, and why? *Current Directions in Psychological Science 19* (Feb. 2010), 51–57.

[2] Dietterich, T. The MAXQ method for hierarchical reinforcement learning. In *Proc. Int. Conf. on Machine Learning* (1998), pp. 118–126.

[3] Diuk, C., Cohen, A., and Littman, M. An object-oriented representation for efficient reinforcement learning. In *Proc. Int. Conf. on Machine Learning* (2008), ACM, pp. 240–247.

[4] Dzeroski, S., Raedt, L. D., and Blockeel, H. Relational reinforcement learning. In *Proc. Int. Conf. on Machine Learning* (1998), ACM, pp. 136–143.

[5] Fern, A., Yoon, S., and Givan, R. Approximate policy iteration with a policy language bias. In *Advances in Neural Information Processing Systems* (2003), pp. 8–13.

[6] Greiner, R. PALO: A probabilistic hill-climbing algorithm. *Artificial Intelligence 84*, 1 (1996), 177–208.

[7] Guestrin, C., Koller, D., Gearhart, C., and Kanodia, N. Generalizing plans to new environments in relational MDPs. In *Proc. of the Int. Joint Conf. on Artificial Intelligence* (2003).

[8] Hengst, B. Discovering hierarchy in reinforcement learning with HEXQ. In *Proc. Int. Conf. on Machine Learning* (2002), pp. 234–250.

[9] Jonsson, A., and Barto, A. Automated state abstraction for options using the U-tree algorithm. *Advances in Neural Information Processing Systems* (2001), 1054–1060.

[10] Kearns, M., and Singh, S. Finite-sample convergence rates for q-learning and indirect algorithms. *Advances in Neural Information Processing Systems* (1999), 996–1002.

[11] Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. Optimization by simulated annealing. *Science 220*, 4598 (1983), 671–680.

[12] Konidaris, G., and Barto, A. Efficient skill learning using abstraction selection. In *Int. Joint Conf. on Artificial Intelligence* (2009), pp. 1107–1112.

[13] Russell, S., and Zimdars, A. Q-decomposition for reinforcement learning agents. In *Proc. Int. Conf. on Machine Learning* (2003).

[14] Seijen, H. V., Whiteson, S., and Kester, L. Switching between representations in reinforcement learning. In *Interactive Collaborative Information Systems*. 2010, pp. 65–84.

[15] Simsek, O., and Barto, A. G. Using relative novelty to identify useful temporal abstractions in reinforcement learning. *Proc. Int. Conf. on Machine Learning* (2004), 95.

[16] Sprague, N., and Ballard, D. Multiple-goal reinforcement learning with modular sarsa(0). In *Int. Joint Conf. on Artificial Intelligence* (2003).

[17] Stolle, M., and Precup, D. Learning Options in Reinforcement Learning. *Abstraction, Reformulation, and Approximation* (2002), 212–223.

[18] Talvitie, E., and Singh, S. Simple Local Models for Complex Dynamical Systems. In *Advances in Neural Information Processing Systems* (2008).

[19] Van Otterlo, M. A survey of reinforcement learning in relational domains. In *CTIT Technical Report Series, ISSN 1381-3625* (2005).

[20] Vigorito, C. M., and Barto, A. G. Intrinsically Motivated Hierarchical Skill Learning in Structured Environments. *IEEE Trans. on Autonomous Mental Development 2*, 2 (2010), 132–143.

[21] Watkins, C. J. C. H., and Dayan, P. Q-learning. *Machine Learning 8* (1992), 279–292.

[22] Wilson, T. *Strangers to ourselves: Discovering the adaptive unconscious*. Belknap Press, 2004.