Solving Grid World Markov Decision Processes with Reinforcement Learning

Michael Walton
Department of Computer Science, Georgia Institute of Technology

November 30, 2015

Abstract

In this work we explore the application of methods from Reinforcement Learning to solving Markov Decision Processes of varying complexity. We give an introduction to several key concepts in reinforcement learning before explicitly describing two 'grid-world' MDPs. We then explore three methods of finding optimal policies for our example problems using value iteration, policy iteration and Q-learning. Finally we discuss the algorithms in terms of wall-clock time, convergence and compare the qualitative structure of the learned policies.

1 Introduction

Broadly defined, Reinforcement learning is primary concerned with the control of autonomous agents in interaction with some environment and objective the agent is trying to optimally achieve. In this context, actions taken in the environment are commonly associated with reward values which reflect how the appropriateness of taking a particular action given some state. It is important to note, however, that this is not the value the agent would ultimately like to maximize; rather we are more concerned with learning optimal policies (state, action sequences) that achieve the greatest cumulative reward. Further complicating this property, a reward signal from the teacher component of a reinforcement learning problem may be delayed after a long series of actions or exclusively occur after the agent reaches some terminal state. This problem is commonly referred to as 'temporal credit assignment'.

In supervised learning, we are faced with a problem of learning from a fixed set of examples selected from some unknown and (typically) un-controllable distribution. This is not so in reinforcement learning, where actions selected by the autonomous agent determine the type of training examples available to the learner. This leads to a problem of exploration vs. exploitation: how should we variably adjust how much the agent explores the environment (to build a better understanding of its world) vs exploits what it knows to accrue more rewards.

Taking as an example a real autonomous agent acting in the real world we are also faced with the issue of non-determinism in both the robot's sensors and effectors. That is, measurements of the environment may be noisy or yield partial information giving an incomplete approximation of the agent's state. In this context we may expand on the notion of exploration where the optimal policy for the agent may be to select actions that improve it's approximation of the environment. In addition to partially observable state, non-deterministic state transitions force the agent to estimate conditional probability distributions over resulting states conditioned on the action and present state.

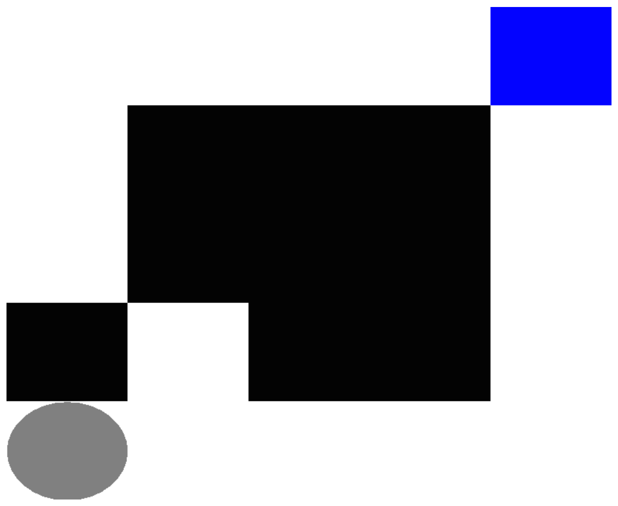
Our implementation is based on the Java BURLAP (Brown-UMBC Reinforcement Learning and Planning) library. BURLAP provides tools for defining Markov Decision Processes of arbitrary complexity and applying reinforcement learning algorithms to learn optimal policies from these models [1]. Classmate Juan Emeterio's aggregated several relevant BURLAP tutorials into what as been called in the forums 'open source assignment 4' [3]. The experiments we conducted were based on these tutorials with modifications where appropriate and interesting to our research interests. We also used various packages from the scientific python toolkit for post-processing and visualization of results [2].

2 Methods

In the sections that follow, we will provide a brief overview of the MDPs explored in the present study in terms of their possible states and a high level description of their structure and objectives. We will frame our solutions to these problems and discuss policy iteration, value iteration and Q-learning methods for solving MDPs.

2.1 Markov Decision Processes

The 'autonomous agent in iteration with an environment' scenario posed informally in the introduction may be formally described by a Markov Decision Process, or MDP. An MDP is a discrete time stochastic control process; that is, MDPs are a model of state action transitions where a controller chooses an action a available in state s and transitions to some next state $s' \in S$ where the the probability of transitioning to a particular s' is given by $T(s, a, s')$. In order to be defined as an MDP, the model must satisfy the Markov assumption; that is the MDP must be defined such that T


(a) Easy Grid World

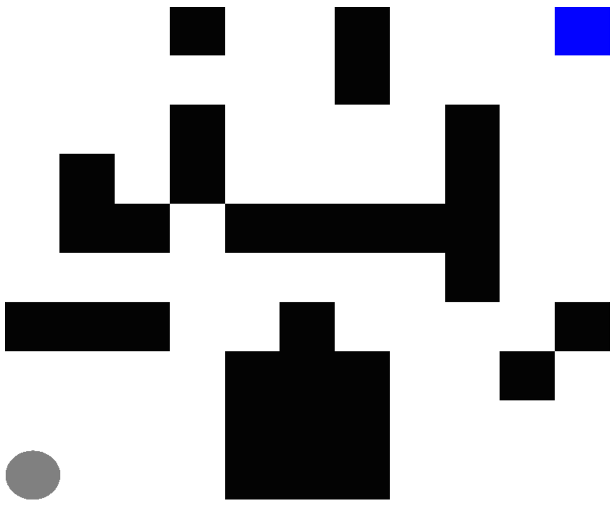

(b) Hard Grid World

Figure 1: Grid World Markov Decision Processes

is conditionally independent of the prior state action sequence. Also typically associated with MDPs is some reward function $R(s, a, s')$ which is the reward value assigned to taking a particular action in state s resulting in the observed next state s'. As previously described, however, the objective of our learner in this context is to maximize the cumulative reward over a sequence of state action transitions.

In our experiments we formulate a grid world MDP where the cells of an $n \times m$ grid are the possible states of the agent. Each state has at most four possible actions (transitions to the neighboring states) and is subject to the constraint of 'walls' (show in black in figure 1) which may restrict the space of available actions for a given state. The agent's initial position is denoted in the diagram as a gray ellipse and the goal state is indicated by a blue square.

2.2 Value Iteration

One relatively intuitive means of formulating a solution to this problem is by iteratively propagating the reward backward from the goal state to the initial state. In this way we estimate a function $V^{\pi}(s)$ which is the cumulative value of a state given a policy $\pi : S \rightarrow A$. We also define a discount parameter $\gamma \in [0, 1)$ which regulates the relative weight of anticipated future rewards vs. the immediate reward. From this notation we define cumulative reward V for a particular state s_t on iteration t to be

$$V^{\pi}(s_t) = \sum_{i=0}^{\infty} \gamma^i r_t + i \tag{1}$$

Here we observe that the weight assigned to each successive state will decrease exponentially in i by a factor of γ. $V(s_t)$ is therefore the cumulative discounted value of following a sequence of actions, and observing a sequence of states according to the policy π from some arbitrary state s_t. Thus the optimal policy π^* is the policy which maximizes V for all states.

$$\pi^* = \underset{\pi \in \Pi}{\operatorname{argmax}}[V^{\pi}(s), \forall s \in S] \tag{2}$$

With this notion of cumulative value we define the value iteration algorithm by looping over all possible states and estimating the value of the state following a particular action and following the current policy thereafter. Assuming some stopping criterion (some value threshold that must be minimized between adjacent states is often used) the algorithm will return a mapping that assigns values to states. From this mapping we may form an optimal policy by selecting the action for a particular state as the $a \in A$ which leads to a successive state s' with the maximal $V(s')$

Algorithm 1 Value Iteration Algorithm

```
1: randomly initialize $V(s)$ uniformly $\forall s \in S$
2: loop until convergence
3:     for $s \in S$ do
4:         $V(s) \leftarrow \max_a R(s,a) + \gamma \sum_{s' \in S} T(s,a,s')V(s')$
5:     end for
6: end loop
```

2.3 Policy Iteration

Policy iteration is similar in its approach to value iteration except in policy iteration the algorithm computes the cumulative discounted value of each next state s' under the current policy π. Policy iteration then compares the value each next state and updates the policy for the current state $\pi(s)$ if there is an action that would lead to a state with a greater cumulative value than the next state of the current policy.

In the first line of algorithm 2, the policy is arbitrarily initialized, this means that for every possible state, the initial policy will take some randomly selected action from the available set of possible actions for that state. In the main loop, the algorithm iterates until $\pi = \pi'$ this control state is reached when the algorithm cannot make any further adjustments to the policy (optimal actions have been found for each possible state) and the optimal policy π^* has been identified. On each iteration of the main loop, the value function of the current policy is derived by solving the system of linear equations defined by the equation on line 4 of algorithm 2. Finally, the current policy is improved by selecting the action $\pi'(s)$ which maximizes the cumulative discounted reward for the current state over the set of possible actions.

Algorithm 2 Policy Iteration Algorithm

```
1: randomly initialize $\pi'$
2: **loop** until $\pi = \pi'$
3:     $\pi \leftarrow \pi'$
4:     $V_\pi(s) = R(s, \pi(s)) + \gamma \sum_{s' \in S} T(s, \pi(s), s') V_\pi(s')$
5:     $\pi'(s) \leftarrow \underset{a}{\text{argmax}}(R(s, a) + \gamma \sum_{s' \in S} T(s, a, s') V_\pi(s'))$
6: **end loop**
```

2.4 Q-learning

In both value and policy iteration we assumed that the learner has access to the reward function $R(s, a, s')$ and the transition probability function $T(s, a, s')$. This scenario is however unrealistic in real world scenarios (ie a robot in some space). Roughly Q-learning eliminates the need for either of these values to be known by sampling actions with unknown resulting states s' and observing their corresponding rewards simultaneously. This is accomplished by defining $Q^*(s, a)$ to be the expected discounted cumulative value of taking action a in state s and then proceeding optimally thereafter. The fact that $V^*(s)$ denotes the case in which the optimal action is selected in state s as well, $Q^*(s, a)$ may be defined recursively as:

$$Q^*(s, a) = R(s, a) + \gamma \sum_{s' \in S} T(s, a, s') \max_{a'}[Q^*(s', a')] \tag{3}$$

In this context the optimal policy is simply the action in each state which maximizes Q. From this definition we can derive the simplified update rule in terms of Q, the explore vs. exploit weight parameter ϵ and a tuple containing an experience defined as a state s an action a an associated reward r and a resulting next state s':

$$Q(s, a) \leftarrow Q(s, a) + \epsilon(r + \gamma \max_{a'} Q^*(s', a') - Q(s, a)) \tag{4}$$

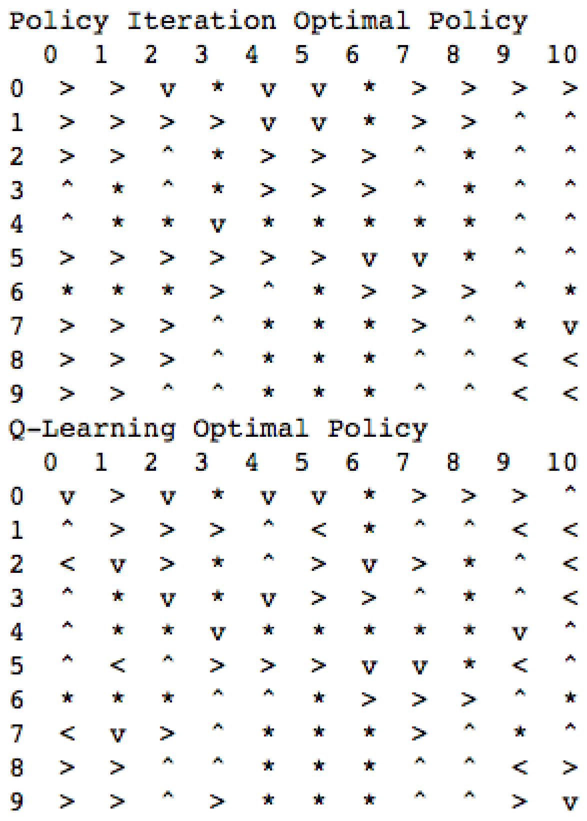
3 Results

In order to compare the behaviors and relative performance of the three aforementioned algorithms, two Java applications were developed which run the three implementations on the easy and hard grid world models. These applications output 2D char arrays representing the optimal policies learned by each algorithm, number of steps taken by each algorithm by number of iterations (episodes) and the runtime in milliseconds of each algorithm. Given our grid world implementation, we define the reward to be the inverse of the length of the shortest path to the goal induced by the current policy π_t on iteration t.

In the easy grid world, all three algorithms learned very similar policies and the agent was directed simply along the bottom edge of the grid and up the right margin toward the goal. Because the goal is an absorbing state of this model, the states on the upper and left boundaries of the grid were likely never visited by any of the algorithms and thus, these states have arbitrary associated actions.

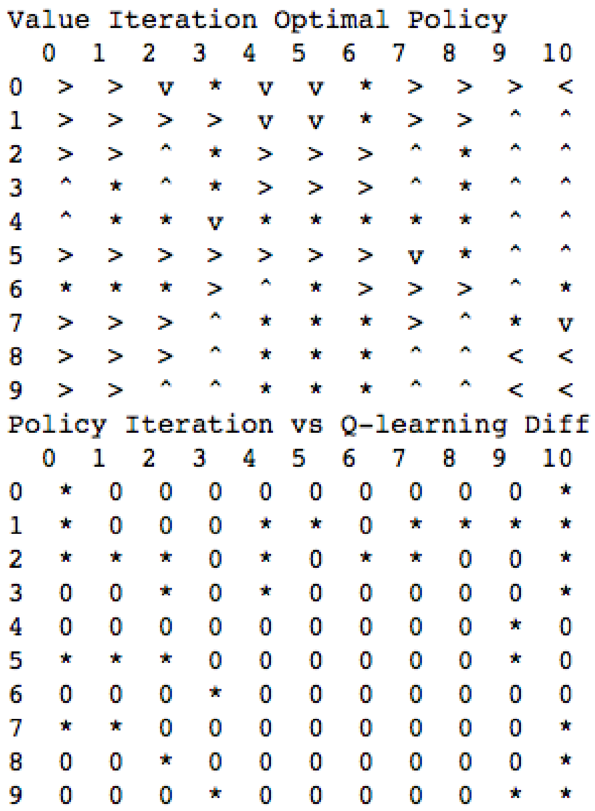
In the first three policy plots for the hard grid world shown in figure 2, the optimal action for each state is denoted with an arrow indicating the direction the agent should take in that particular state; asterisks indicate a 'wall' as previously described and represented as black squares. As expected, the optimal policies learned by value iteration and policy iteration are largely similar. This is because they both maximize the cumulative discounted reward (by minimizing the length of the path to the goal) for all states. Further, this yields a policy that is a solution to the problem posed as a dynamic programming problem wherein the optimal policy may be used as a lookup table where, for any arbitrary state in the grid, $\pi^*(s)$ will return the optimal next action.

The bottom right plot of figure 2 indicates the difference between the optimal policy returned by policy iteration vs Q-Learning. In this plot, zeros indicate states in which the policies agree and asterisks indicate positions in which they differ. It is interesting to note that in the easy grid world there is only one unique path from the initial state to the goal, however in the harder MDP there are in fact two unique possible paths. It is in this context that the differences between policies found by Q-Learning are most distinguished from the other two algorithms. Q-Learning's policy agrees with the other two along the true shortest path from the initial state but disagrees along the longer alternate path. There is agreement up to a certain point and after this state the path found by Q-learning appears sub-optimal. As previously described, Q-learning will explore each action and consider their relative cumulative value following from a scenarios where the algorithm behaves optimally following the next action. Keeping in mind that there is an inherent trade-off between exploration and exploitation; an intuitive explanation for this behavior is that Q-learning explored



```
Policy Iteration Optimal Policy
   0   1   2   3   4   5   6   7   8   9   10
0  >   >   v   *   v   v   *   >   >   >   >
1  >   >   >   >   v   v   *   >   >   ^   ^
2  >   >   ^   *   >   >   >   ^   *   ^   ^
3  ^   *   ^   *   >   >   >   ^   *   ^   ^
4  ^   *   *   v   *   *   *   *   *   ^   ^
5  >   >   >   >   >   >   v   v   *   ^   ^
6  *   *   *   >   ^   *   >   >   >   ^   *
7  >   >   >   ^   *   *   *   >   ^   *   v
8  >   >   >   ^   *   *   *   ^   ^   <   <
9  >   >   ^   ^   *   *   *   ^   ^   <   <
Q-Learning Optimal Policy
   0   1   2   3   4   5   6   7   8   9   10
0  v   >   v   *   v   v   *   >   >   >   ^
1  ^   >   >   >   ^   <   *   ^   ^   <   <
2  <   v   >   *   ^   >   v   >   *   ^   <
3  ^   *   v   *   v   >   >   ^   *   ^   <
4  ^   *   *   v   *   *   *   *   *   v   ^
5  ^   <   ^   >   >   >   v   v   *   <   ^
6  *   *   *   ^   ^   *   >   >   >   ^   *
7  <   v   >   ^   *   *   *   >   ^   *   ^
8  >   >   ^   ^   *   *   *   ^   ^   <   >
9  >   >   ^   >   *   *   *   ^   ^   >   v
```

(a) Policy Iteration and Q-Learning



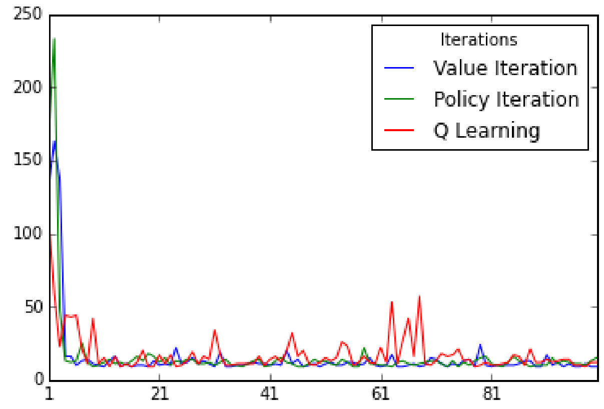
```
Value Iteration Optimal Policy
   0   1   2   3   4   5   6   7   8   9   10
0  >   >   v   *   v   v   *   >   >   >   <
1  >   >   >   >   v   v   *   >   >   ^   ^
2  >   >   ^   *   >   >   >   ^   *   ^   ^
3  ^   *   ^   *   >   >   >   ^   *   ^   ^
4  ^   *   *   v   *   *   *   *   *   ^   ^
5  >   >   >   >   >   >   >   v   *   ^   ^
6  *   *   *   >   ^   *   >   >   >   ^   *
7  >   >   >   ^   *   *   *   >   ^   *   v
8  >   >   >   ^   *   *   *   ^   ^   <   <
9  >   >   ^   ^   *   *   *   ^   ^   <   <
Policy Iteration vs Q-learning Diff
   0   1   2   3   4   5   6   7   8   9   10
0  *   0   0   0   0   0   0   0   0   0   *
1  *   0   0   0   *   *   0   *   *   *   *
2  *   *   *   0   *   0   *   *   0   0   *
3  0   0   *   0   *   0   0   0   0   0   *
4  0   0   0   0   0   0   0   0   0   *   0
5  *   *   *   0   0   0   0   0   0   *   0
6  0   0   0   *   0   0   0   0   0   0   0
7  *   *   0   0   0   0   0   0   0   0   *
8  0   0   *   0   0   0   0   0   0   0   *
9  0   0   0   *   0   0   0   0   0   *   *
```

(b) Value Iteration and Policy Itration vs QL

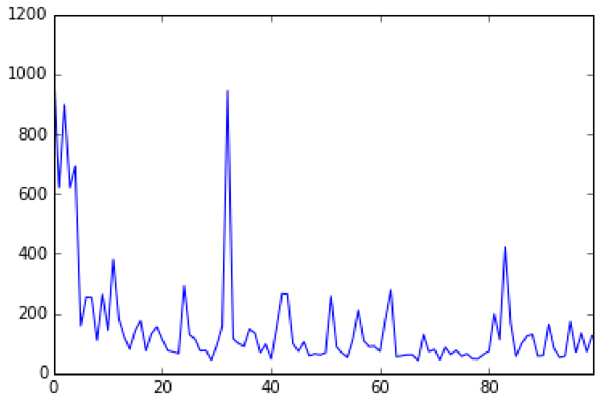
Figure 2: Optimal Policies by Algorithm

both paths while they both had similar cumulative value, however at a certain point Q-learning abandons further pursuit of the sub-optimal route and focuses exclusively on refining the primary path.

The algorithms were also assessed in terms of 'reward over time'; in the context of our previously described grid world MDPs we measure this in a somewhat different way by computing the shortest path found by policy $\pi_t(s)$ over t iterations. As indicated in figure 3, there was little difference in the path length over number of iterations in the case of the easy grid world; the behavior of the three algorithms in the hard grid world was similar. In order to force some discrepancy between the algorithms, a brief third set of experiments was briefly explored in which we further complicated the grid world maze. The 'hardest' maze consisted of multiple 'switchbacks' which was a construction of the freespace of the grid such that the agent would have to return in the direction it came to make progress along the maze. This version of the grid world also had only a single path and the walls of the maze were only spaced widely enough for a single forward action to be taken for most of its length. In this extreme version, both policy iteration and value iteration failed to converge in the



(a) Easy Grid World



(b) Hardest Grid World (Q-Learning)

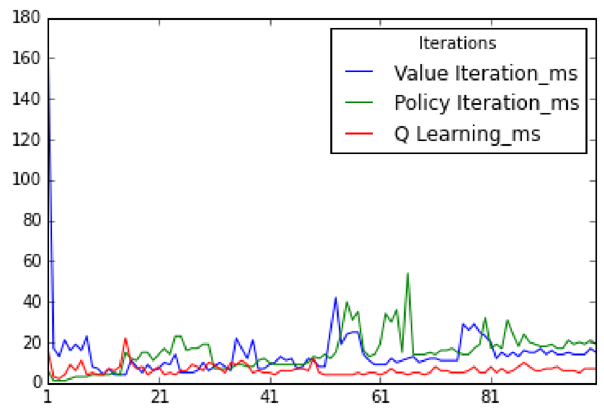
Figure 3: Path Length over Number of Iterations

time considered. Further, policy iteration exceeded the Java virtual machine's memory limitation. Q-learning on the other hand managed to find the optimal policy although it took more iterations than on the easy or hard grid world to converge.

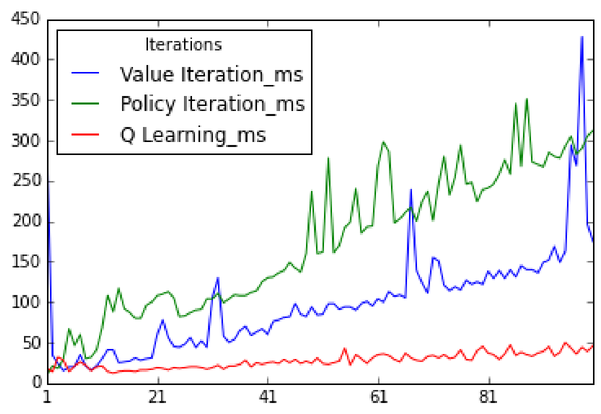
The most obvious difference between the algorithms was in terms of their wallclock runtimes, shown in figure 4. Again there is not too much difference in terms of time over number of iterations for finding policies for the easy grid world. However a significant difference was observed in the experiments for finding an optimal policy on the hard grid world. In the latter case, value iteration and policy iteration exhibited roughly linear time complexity, where the slope of the value iteration runtime curve over number of iterations was slightly steeper. As expected and evidenced by these results and the aforementioned memory overflow, we conclude that value iteration takes slightly longer per iteration however it is more space efficient with respect to policy iteration. Q-learning however, exhibited much faster wallclock times and appears to operate in roughly $O(1)$ time.

4 Conclusion

In the present study we have explored the application of three reinforcement learning algorithms to grid world Markov Decision processes. Though simple, grid world problems form the basis for more general problems in the navigation of autonomous agent and robots in more complex environments. Therefore, it is important to understand the behavior and capability of reinforcement learning in the context of simple problems in order to derive intuitions about the algorithms' behavior on more complex classes of problems. We described and compared value iteration and policy iteration and



(a) Easy Grid World



(b) Hard Grid World

Figure 4: Path Length over Number of Iterations

demonstrated the relationships between dynamic programming, value and policy iteration. Further we implemented and discussed Q-learning applied to grid worlds and demonstrated this method's robust capabilities in the context of more complex MDPs as well as its reduced time and space complexity when compared to value and policy iteration. Finally we discussed some of the trade offs inherent in this algorithm involving exploration vs exploitation and identified an example of this in the harder grid world experiments.

References

[1] Carlos Diuk, Andre Cohen, and Michael L. Littman. An object-oriented representation for efficient reinforcement learning. In *Proceedings of the 25th international conference on Machine learning - ICML*. Association for Computing Machinery (ACM), 2008.

[2] Eric Jones, Travis Oliphant, Pearu Peterson, et al. SciPy: Open source scientific tools for Python, 2001–. [Online; accessed 2015-09-15].

[3] Juan Jose. [burlap fork] open source assignment 4, 2015.