Iteration and Parameterization Behavior of Two MDP Algorithms

Liam Taylor and Henry Daniels-Koch
Optimization and Uncertainty
May 21, 2016

Abstract

Markov Decision Processes (MDPs) are an efficient formulation of probabilistic environments out of which utility-oriented plans can be extracted. We implement and discuss two model-based, fully observable MDP algorithms, Value Iteration and Policy Iteration, each of which operate by discovering utilities of a given test MDP environment and translating those utilities into an optimal plan. We compare the speed and iteration time of these algorithms. Additionally, we analyze the effects specific MDP and algorithm parameters have on the required number of iterations and the actual policies. We find that Value Iteration is the significantly faster choice, although Policy Iteration is more efficient in terms of number of iterations. We also find that changing environmental and algorithmic parameters have a drastic effect on the number of required iterations and the potential behavior of an agent in the test environment. Further work should focus on increasing the efficiency of Policy Iteration to capitalize on its unique strengths as well as more closely characterizing the interaction between algorithm choice and the formulated environment.

1. Introduction

A major goal of artificial intelligence (AI) research is determining the most effective and efficient modes of planning and problem-solving in probabilistic environments. Effective plans allow agents, such as automated robots, to face complex environments and respond correctly to complete their assigned task or move towards their overall goal. These plans have two critical steps. The first is formulating the environment in such a way that appropriate information can be extracted, while the second is actually extracting an optimal or near-optimal plan. Current research has led to a variety of software strategies through which agents can develop plans in constructed environments.

One such strategy is the modeling of environments and subsequent plans as Markov Decision Processes (MDPs). In MDPs, agents are placed in a world represented by a set of states. Actions allow the agent to probabilistically move from state to state given a set of "transition probabilities." States have pre-assigned rewards or punishments to appropriately represent the environment. Agents can develop an optimal plan towards the goal by finding the plan with the highest utility (the maximum overall reward, or the least overall punishment). This process is rendered highly efficient due to the Markov property, through which an agent only needs to understand its current state to make decisions about future ones. MDPs can not only incorporate action uncertainty, they can also restrict or remove knowledge about the transitions and rewards of states (as in "model-free MDPs," e.g., Lagoudakis and Parr 2001) or render the current state

only partially observable and, therefore, reject the Markov property (as in POMDPs, e.g., Smith and Simmons 2004).

Given an MDP formulation, there are multiple algorithms with which the utilities of various states can be parsed and translated into an actual plan. Two such algorithms, Value Iteration (VI) and Policy Iteration (PI) rely on different iterative calculations to not only discover the true utilities of states given the rewards and transition probabilities, but also translate those utilities in a final, optimal action plan. Both algorithms require parameters that dictate their exit criteria and their planning behavior along the way. In addition to those algorithm parameters, more fundamental parameters dictate the structure of the MDP environment, especially if that environment is complex.

In this paper, we seek to understand the relative efficiency of VI and PI MDP algorithms. We compare their speed both in regards to number of iterations and actual runtime using a non-trivial MDP test environment and a constant set of parameters. We also explore the effects of individual parameters on our VI algorithm in terms of both the number of iterations required to determine an optimal policy and the effects these parameters have on the policies themselves.

In Section 2, we describe MDPs, VI, and PI in greater detail. In Section 3, we explain our specific MDP test problem and our comparison and parameterization experiments. In Section 4, we present and discuss the results of those experiments. In Section 5, we conclude with broad patterns and suggest relevant future work.

2. Description of Algorithms

MDP Problems

MDP problems are centered around states, observations, and actions in a static environment. A state is an agent's placement within the broader context of the problem, and an observation is knowledge that the agent has about current or past states. Actions allow agents to transition from state to state. Given action uncertainty, a certain action may have only probabilistic effects. To represent action uncertainty in an MDP environment, a transition probability function governs the likelihood that an agent in a starting state s, will reach a particular state s'. It then follows that the sum of the transitional probability for reaching all possible states s' is 1.

An MDP problem is fundamentally guided by utilities. A reward function links states to their utilities, which are designed to appropriately represent the environment and push an agent towards its goal. Rewards can be both positive and negative. In some MDP algorithms, like the Value Iteration and Policy Iteration approaches applied here, an agent has the full knowledge of the "model" (the full transition probabilities and rewards of each state). Other, "model-free" problems are not constrained by this assumption, and require more complex methods to solve.

With full knowledge of transition probabilities and rewards, an agent can develop an optimal action plan for each state (a "policy") by determining the action that maximizes expected utility. With full observability, agents can evaluate the maximum utility by simply picking the

highest weighted average utility of the state resulting from a given action as adjusted by the transition probability from the current state to that target state. Borrowing from economic theory, the algorithms applied here also use discounting, in which future rewards are less valuable than current ones. This discounting adds an implicit sense of time to MDP solutions, in which every action essentially occurs at a single timestep. In summary, the optimal policy (π^*) over an infinite discounting horizon can be described as:

$$\pi^* = argmax_\pi E(\sum_{t=0}^{inf} \gamma^t r_t | \pi)$$

Where E is the expected utility function, t is the current time step, γ is the discount factor, r is the reward, and π is an individual policy.

A required assumption for this process is the Markov property, wherein the current state contains all the necessary information for choosing the optimal next action. Under the Markov property, an agent only needs to know where it is, not where it has been. To fulfill this property and to develop the optimal policy as discussed above, an agent must seek the true utility of each state. This utility must emerge not only from the rewards of the state itself, but from the proximity and utilities of other states that may be linked through future actions. Value Iteration and Policy Iteration are two model-based solutions to extracting these utilities.

Value Iteration

The first MDP algorithm we present is Value Iteration (VI). In VI, utilities are extracted iteratively from the initial rewards and transition probabilities of each state. In order to calculate the utility of each state, we utilize a Bellman equation:

$$U(s) = R(s) + \gamma \, max_a \sum_{s'} [T(s, a, s') U(s')]$$

Where, given current state *s*, action *a*, and subsequent state *s'*, *U* is the utility function, *R* is the reward function, γ is the discount factor, and *T* is the transition probability function.

Using this equation, VI begins to estimate more precise utilities as it iterates across every state *s* and, for each *s*, every possible *s'*. This process essentially defuses the fundamental rewards and punishments of each state across transition boundaries. Our VI implementation uses "in-place" updating, where the new utilities for each state are updated as information for all subsequent states as soon as they are calculated within a given iteration.

VI iterations continue until they reach a stop criterion governed by the equation:

$$\|U_i - U_{i-1}\| \leq \varepsilon \frac{1-\gamma}{\gamma}$$

Given U_i-U_{i-1} as the maximum utility change of any state from iteration i-1 to iteration i, the discount factor γ, and an error threshold ε.

The error threshold simply governs the required precision of the utilities estimated by VI as compared to the true utilities of each state. As we will discuss further, this stopping criterion is focused on utility changes as they converge to the true utilities, and does not factor in changes or lack of changes to the policy itself.

Policy Iteration

Policy Iteration (PI) is similar to VI in that it is a model-based solution that fully utilizes transition and reward functions. The general idea behind PI is to start with a random policy, iteratively improve it by evaluating its utilities, and eventually terminate the function when the policy stops changing. In this algorithm, each state at each iteration is associated with a Bellman equation, including terms for the utilities of every possible subsequent state. Thus, we can consider these states as a set of *s* equations (one for each state) with *s* unknowns (one for each state utility). Because each iteration starts with a predetermined policy, the equations do not include complex operators like *argmax* and can be presented and solved as a matrix equation AX=B, where X is a single column *s*-length vector of utilities of each state, B is the reward of each state, and A is the *s***s* matrix of subsequent state utilities discounted by γ and weighted by transition probabilities. We solve this matrix equation for X by performing lower-upper decomposition to factor the matrix as a product of a lower matrix (zeros above the diagonal) and an upper matrix (zeroes below the diagonal). Through X, we receive a utility estimate for each state. We can then iterate through each state and decide the new policy for the next iteration that appropriately chooses actions based on probability weighted average utility. With a finite number of policies, PI is guaranteed to converge on the optimal policy as soon as the policy does not change from one iteration to the next.

While this algorithm is similar to VI in its general utility estimate using Bellman equations, it is different in two key ways. First, it relies on matrix computation to develop utility estimates. While these equations are tractable for the number of states used in our test problem, the solution strategy represents a significant computational investment. Second, PI exits as soon as the policy stops changing, unlike VI, which exits when based on changes to utilities themselves.

3. Experimental Methodology

Test Problem

To test the behavior of our different algorithms and the effects of changing parameters for those algorithms, we implemented an MDP test problem and a series of experiments supplied by S. Majercik. Our test problem consisted of 33 tiles, with a positive terminal tiles, two negative terminal tiles, a Key (gain) tile, and a Lose-Key? tile (Fig. 1). At each tile, the agent has four actions (north, east, south, or west). Given a chosen action, the agent has an 80% probability of moving in that direction and a 10% probability of accidentally moving either 90° clockwise or 90° counterclockwise. If an agent hits a wall, it remains in its current tile.

The agent begins on a "start" tile towards the bottom of the board. Arriving on any of the three terminal spots ends the game with the given reward on that terminal tile, but the agent only gets the positive terminal reward if it holds a key. The key can only be obtained on the Key tile, but landing on the Lose-Key? tile may remove the key with a certain probability. To effectively represent the key in an MDP framework, the key operates as a state variable, where each tile (except for the Key tile itself) is represented by a key state and a non-key state. In total, then, there are 65 states. The structure of this test problem adds three additional parameters (key loss probability, negative terminal reward, and positive terminal reward) to the more general VI and PI parameters (Table 1).

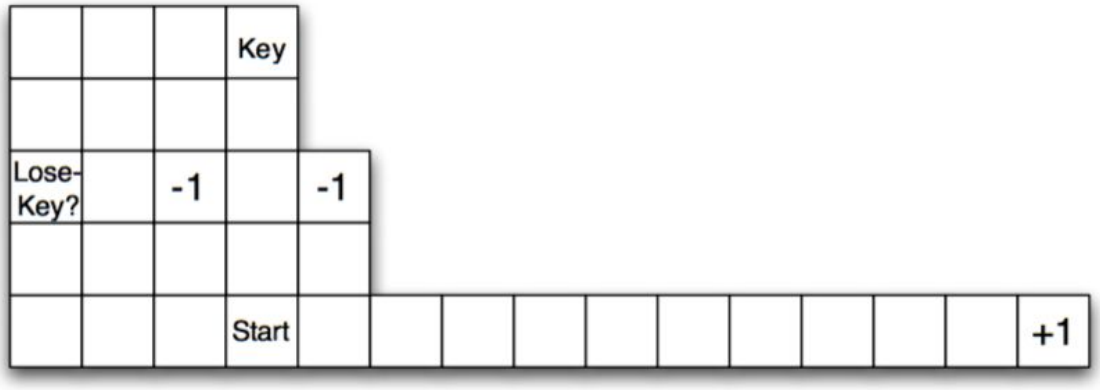

Figure 1. The structure of the MDP test problem used in this study. Negative and positive values represent terminal rewards. An agent requires a key for the positive terminal reward, and can gain that key by landing on the Key tile. An agent loses a key with some probability on the Lose-Key? tile. North is up, south is down, east is right, and west is left.

Table 1. The experimental parameters applied to the MDP test problem (Fig. 1), or to the value and policy iteration algorithms. Constant values were used in all PI/VI comparisons. Individual parameters were examined across their given range, with all other parameters for those tests held to their constant value.

Parameter	Constant Value	Experimental Range		
		Min	Max	Iteration
Error threshold (ε)	1e-6	N/A	N/A	N/A
Discount rate (γ)	0.999999	0.16	1.0	0.001
Step cost	-0.04	-1.00	-0.001	0.001
Negative terminal reward(s)	-1	-10	-0.01	0.01
Positive terminal reward	1	0.01	10	0.01
Key loss probability	0.5	0.01	0.99	0.01

Experiments

We sought to analyze our MDP algorithms from two different perspectives. First, we aimed to understand the relative efficiency of our two different algorithms, VI and PI. We measured efficiency in three ways: (1) The number of iterations the algorithms went through before exiting with an optimum policy; (2) The number of iterations the algorithm went through before actually discovering the final, optimal policy and; (3) The time (in seconds) each algorithm takes to run through their total iterations. To test these values, we ran our VI and PI algorithms on our test MDP problem using the constant parameter values in Table 1. Number of iterations (1) and number of effective iterations (2) were deterministic given the problem and constant parameters, so only one trial was needed for those two measurements. Alternatively, we ran 10 trials to test for runtime to account for slight variations in processor speed.

Second, we sought to understand the effects changing parameters has on the behavior of the algorithm in two ways: (A) The number of iterations it takes to find an optimal solution and (B) The changes to the actual policy given parameter changes. We tested for these changes using a range of values particular to the behavior and magnitude of each parameter (Table 1). The error factor (ε) was not tested, as it simply controls the precision of the utilities used to decide the optimal policy. To control for potential changes as being tested in our first set of experiments, we used only one algorithm (VI). Holding all other parameters constant, we ran a single trial for each target parameter value, recording both the required number of VI iterations and the actual policy. We analyzed the VI/PI comparison data and the iteration number results using the R programming language (R Core Team 2015). We considered policy changes for each parameter manually.

4. Results and Discussion

Algorithm Comparisons

Our policy and value iteration algorithms varied significantly in both number of iteration and runtime. Using constant parameters (Table 1), we found that VI needed 64 iterations to solve this particular MDP problem while PI needed only 8 iterations. Because PI is designed such that it exits when the policy itself stops changing, the final of its 8 iterations represents the iteration at which it found the actual policy. On the other hand, a sizeable number (33) of the iterations in VI happened after the optimal policy was found. In other words, after iteration 31/64 using VI, the utilities change enough for the given error threshold and discount factor but the policies themselves do not change. Because the goal of both of these algorithms is to find the policy, not the exact utility of each state, the final 33 iterations are essentially wasted time. Based on iteration number alone, PI seems to significantly outstrip VI.

However, the processes of the operations themselves are different in each algorithm, meaning we must also look to the actual amount of time each algorithm takes to solve the problem. As shown in Figure 2, we found that VI was significantly faster than PI, taking an average of 0.0065s (1e-4 seconds/iteration) while PI took an average of 0.0088s (0.0011 seconds/iteration). Even for this relatively small problem, therefore, the per iteration speed of PI is not worth the reduced number of iterations. We expect that this decrease in speed is due primarily to the time it takes to solve the matrix in PI, as this is the major computational cost of the algorithm. Based on these time data, we can conclude that VI is preferably to PI for problems of similar size and behavior to our test problem due to its increased speed, although we cannot speak to the behavior of our algorithms given much larger MDP problems (see **Further Work**). Additionally, we expect that it is more important to carefully manage parameters with PI than VI, as each PI iteration is significantly slower.

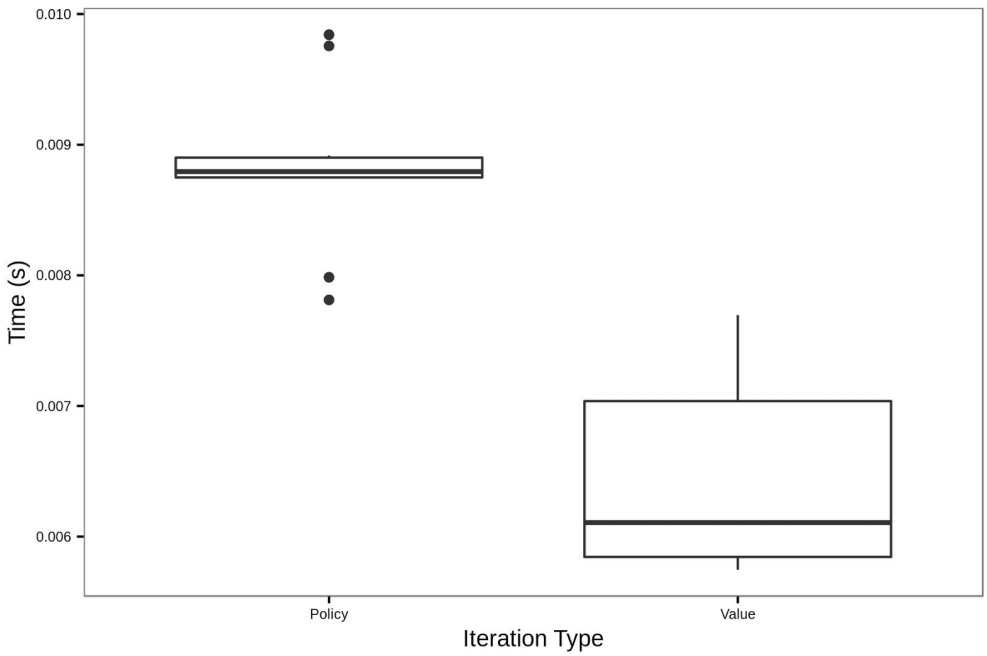

Figure 2. A runtime comparison between policy and value iteration algorithms when solving an MDP test problem with constant parameters (Fig 1, Table 1). Each algorithm's performance was tested over 10 trials. Policy iteration was significantly slower than value iteration at exiting with the optimal policy (Unpaired T-test, d.f. = 17.52, $p < 0.0001$). Boxplots show median value bounded by upper 75th and lower 25th quartiles, with whiskers extended to furthest non-outlier.

Parameters and Iterations

Given the importance of iteration number to the efficiency of the overall algorithm, we looked closely at the interactions between specific parameters and number of iterations. We found that each parameter affects the number of VI iterations differently (Fig. 3). The effects of changing the discount factor are perhaps the most straightforward. Increasing the discount factor (i.e., decreasing the discounting effect) results in an increased number of iterations (Fig. 3A). This response seems to follow a somewhat exponential trend. This change occurs as a direct result of the optimization requirements of our VI algorithm. VI only exits with its current policy if the maximum change to any state's utility is greater than $\epsilon * (1-\gamma)/\gamma$. In other words, the greater the discount, the further the algorithm needs to analyze into the future. If

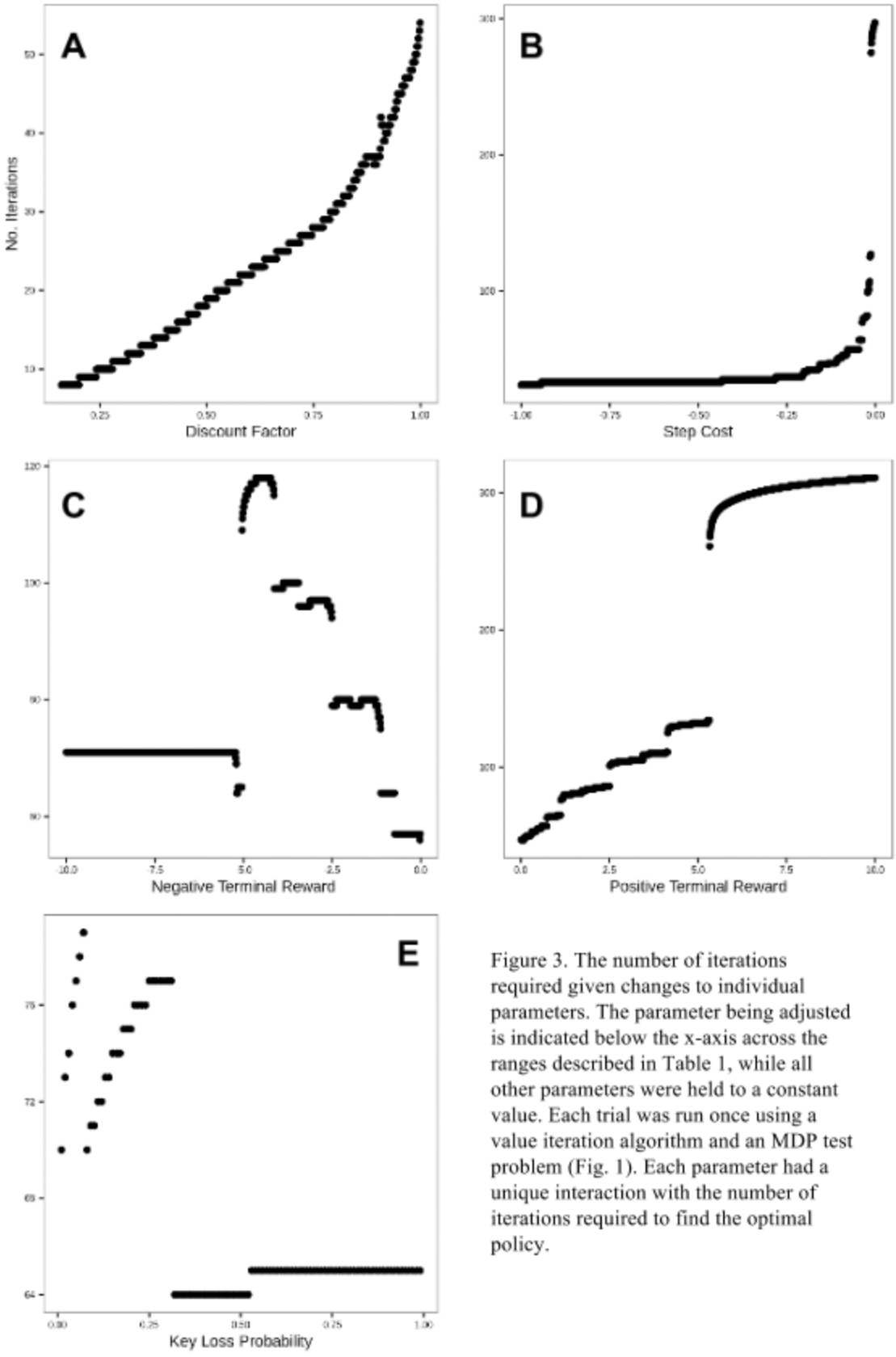

Figure 3. The number of iterations required given changes to individual parameters. The parameter being adjusted is indicated below the x-axis across the ranges described in Table 1, while all other parameters were held to a constant value. Each trial was run once using a value iteration algorithm and an MDP test problem (Fig. 1). Each parameter had a unique interaction with the number of iterations required to find the optimal policy.

discounting is very small force, an understanding of early utilities encapsulate much of the final expected utility of the state. Because we held ϵ constant while testing changes to the discount factor, an increase in the discount factor led to a direct and consistent increase in the number of iterations.

Decreasing the step cost (increasing the negative value towards 0) also increased the number of iterations required by our VI algorithm (Fig. 3B). While also resembling an exponential curve somewhat, this effect was not as consistent as the effects of discounting. Increasing a very low step cost resulted in a drastic decrease in the number of iterations, while increasing already high step costs only minimally decrease the number of iterations. This result is best explained by again looking at the VI exit criteria. Step cost is factored into the rewards that influence new utility assignments at each iteration. If the step cost is very high, the majority of utility at each iteration dominates the overall utility of each state, especially if the cost is near the value of the negative terminal states (as it does at the high end of these tests). Thus, each iteration only changes utilities with a relatively small magnitude, quickly resulting in a convergence of the utilities at each state sufficient to exit from the value iteration algorithm. At very low step costs, the positive and negative terminal rewards matter more, and so the spreading of those values across the states through every iteration is important enough to cause the algorithm to continue.

Negative and positive terminal rewards have more complex interactions with the number of iterations required to find the correct policy. With low or moderate negative terminal rewards (near 0), increasing the magnitude of the negative reward requires more iterations (Fig. 3C). Once the magnitude of the negative reward is sufficiently high (around -5.0), the required number of iterations quickly drops. Increasing the magnitude of this reward past ~ -5.2 does not affect the required number of iterations which stabilizes at ~70. Again, we can understand this result by considering the effect of high and low negative rewards on the updating utilities of each VI iteration. With a very low negative reward, especially with a reward of lesser magnitude than the step cost, the negative terminal states only barely factor into the problem as a whole. Essentially, very small negative rewards remove the constraint of having negative rewards in the first place and simplify the problem such that it can be solved in fewer iterations. As the negative terminal rewards increase somewhat, the negative states constrain the optimal solution more and more, required more extensive diffusion of the utility information onto other states. This increased diffusion effect increases the magnitude of utility changes of later iterations, forcing the VI algorithm to continue further until the utilities stabilize. With a very high negative terminal reward, however, the value diffuses and dominates the other states quickly, resulting in a relatively quick utility convergence and, therefore, relatively few iterations.

As with small negative terminal rewards, positive terminal rewards near zero require relatively few iterations (Fig. 3D). Furthermore, increasing the magnitude of the positive terminal reward also increases the number of required iterations. At this point, we can explain this pattern as we did with negative rewards. A very low reward simplifies the problem, resulting

in fewer required iterations. As the constraint of the positive reward grows with its magnitude, the required number of iterations increases. Unlike the pattern of negative rewards, however, very high positive rewards (>5) result in a high and increasing number of iterations (>250-300). We hypothesize that the root of this difference rests in the structure of the test problem itself. As we discuss more fully in the next section, a large negative terminal reward increases the value of reaching the positive terminal state regardless of where the agent is or whether or not that agent has the key. In the case of the positive reward, however, high rewards increasingly force the policy to dictate better ways to navigate the more complex Key/Lose-Key?/negative terminal interactions at the top of the board. In other words, increased positive rewards given the structure of our test problem force the algorithm to think harder (i.e., run through more iterations) about how to reach that reward with more and more certainty. High negative rewards simply drive the policy towards a more straightforward, non-negative option.

The final parameter we tested, the key loss probability, also had a unique interaction with iteration number (Fig. 3E). Importantly, this parameter had the smallest effect on the required number of iterations, with iteration numbers all falling between 64 and 79. There were two major results dictated by either high (>=0.32) or low (<0.32) key loss probabilities. In the case of low key loss probabilities, the number of required iterations increases as the key loss probability increases because many states become newly important given increasing an increasing threat of key loss. The northernmost states increasingly become factored into decisions about the optimal policy is move east through the dangerous region between the negative terminal states or move west and risk losing a key. The states near the lose key state also slowly move towards avoiding the Lose-Key? state. On the other hand, there are less iterations when the key loss probability is high because only one state changes its optimal policy in both moderate and high probabilities. When states do not change their policies, it is an indication that policy decisions are less difficult and require fewer iterations.

Parameters and Policies

Discount Factor

From the above results, it is clear that parameters closely control the solution behavior of the VI algorithm. As our discussion of those results has implied, these changes to iteration number are closely tied to the actual policy solutions being extracted given each parameter set. Thus, we looked closely at the policy patterns emerging from changes to each parameter, beginning with discount factor. With a discount factor less than 0.99, the optimal policy is to move east towards the positive reward as fast as possible, although without the key, the positive reward state simply represents a non-negative terminal state. With high discounting (i.e., low discount factor), future rewards are discounted so much that it is not worth getting the key before moving to the terminal state. However, the neutral terminal state (the positive terminal state without a key) is still preferable to the slightly closer negative terminal states given the applied parameters for negative terminal reward and step cost. With very low discounting (>= 0.99), the

optimal policy is to move north along the path between the negative rewards (henceforth the "danger zone"), get the key, return south to the start, and then head east. If the agent ends up west of the start with these discount values, the optimal policy is to move north, to the west of the negative terminal, east to get the key, and then south towards the start and east towards the positive terminal state. This is because low discounting allows for more moves to be taken and makes getting the key more valuable as a future reward. This path risks the negative terminal states but avoids the step cost of a long path, especially because there is still some discounting and the key could be lost multiple times on the longer path to the west. Even with relatively moderate discounting (between 0.97 and 0.99), agents finding themselves north of the starting position should choose to follow the path towards the key before moving towards the positive reward. Presumably, this is because the additional reward for getting the key and ending in the positive state is not as highly discounted when you are already closer to being in that state with the key. In general, very high discounting seems to cause agents to react to current step costs rather than more distant terminal rewards, while low discounting allows the agent to focus on the path that maximizes terminal rewards even if they are far in the future.

Step Cost

Given small step costs (between -0.03 and 0), the optimal policy is to move northwest initially, to avoid the danger zone, move east and get the key, and return along the same path. Even if a keyless agent is right next to the positive reward, the optimal policy is still to go and retrieve the key. As small step costs decrease, the state at which it is optimal to move away from the positive terminal tile without the key propagates west. With such a low step cost, the agent is able to spend many moves trying to key the key along the most cautious path before arriving at the positive terminal tile with the key.

With more moderate step costs (between -0.1 and -0.03), the optimal policy is to move directly north from the start into the danger zone, retrieve the key, return south along the same path to the start and east to the positive terminal. At these moderate step costs it is no longer optimal to avoid the risk of the danger zone because the extra steps of going around the danger zone are too high, especially given the potential for key loss. With somewhat larger step costs (between -0.3 and -0.1), the optimal policy without a key is to minimize the inevitable negative utility from each step and terminate the problem at the closest terminal tile as fast as possible, which forces the agent to move towards the negative terminal state. If the agent manages to be in the south with the key, the optimal policy is still to move east to the positive terminal.

With large step costs (<-0.3), the optimal strategy with or without the key is move directly north to the nearest terminal state and terminate the problem to limit the accumulation of negative utility. A high step cost means that the cost of traveling to the positive terminal tile, even with the key, is not worth the damage that the agent would incur for its steps. However, if the agent finds itself on the path in between the start and positive reward, increasingly large step costs suggest policies of moving towards the negative reward starting further to the east until the

the spot directly between the negative and positive reward is reached. As we can see from these various policies, step costs directly tune the trade-offs agents should make for terminal rewards or the more complex interactions around the key.

Negative Terminal Reward

With small negative terminal rewards (between -0.04 and 0), the optimal policy without the key is to move directly towards the key tile from any non-key state, even if that means going through the danger zone. The one exception to this rule is when the actor is in the positive terminal tunnel, in which case the standard step cost generally dissuades the agent from getting the key. At this parameter range, the agent does not have to worry about avoiding the negative terminal because the negative terminal rewards are so insignificant.

Given any larger negative terminal reward (<-0.04), it becomes optimal to avoid the danger zone. The punishment of the negative terminal rewards now outweighs the extra step costs from having to obtain the key one or more times on this longer path. As the negative reward increases in punishment (< -0.05), it becomes optimal to avoid more north-western negative tiles for the same reason.

With moderate negative rewards (between -4 and -1.5), the optimal policy directs key-less agents in the south to move directly east to the positive terminal, avoiding the negative rewards at the expense of losing the actual positive reward from the terminal state. However, if the agent is already to the west of the negative terminal tiles, it is still optimal to get the key because it is less risky than going through the danger zone.

With high terminal values (<-4), the optimal strategy increasingly becomes avoiding the negative terminal states at all costs, even if that means not getting the key unless on the northernmost row. Instead, agents should head for the positive terminal state as safely and quickly as possible. If the agent is above the negative terminal states, the agent should avoid the danger zone and move west as far from the negative terminal as possible, move south along the perimeter, and then east to the positive terminal. This strategy becomes optimal because the negative terminal punishment is so large that it outweighs all other decisions including getting the key, losing the key, or limiting step costs.

Positive Terminal Reward

With low positive terminal rewards, (between 0 and 0.7) it is optimal for key-less agents to move towards the positive terminal state, even if it means not getting the key. The agent's optimal path should be to move immediately towards the positive reward, not to attain the reward but to avoid an accumulation of step costs or the possibility of ending up in the -1 negative terminal state. Additionally, with low positive terminal values, higher step costs cause the agent to pursue a strategy that prioritizes taking the fewest steps possible before reaching a positive terminal state. This strategy is similar to the policy responses to high negative terminal policy, as

a negative terminal reward of about -1 negative terminal policy is very high when compared to a low positive terminal reward such as 0.01.

With moderate positive terminal values (between 0.7 and 2), the optimal policy for actors without keys is to move directly north into the danger zone, get the key, and return south to the start, and then move east to the positive terminal. In this situation, the positive reward is not high enough to cause the actor to avoid the negative terminal at all costs, but the positive reward is not low enough to necessitate the agent moving directly to the positive reward to limit the accumulation of step cost punishments.

Given high positive terminal rewards (>2), the optimal strategy for key-less agents is to get the key and avoid the danger zone at all costs. With such a high positive reward, the agent should not reduce its steps by going through the danger zone because even a small chance of not attaining the positive reward is not worth the risk. This policy is similar to small negative terminal reward policies. In general, the positive terminal reward's relationship to step cost and the negative terminal reward dictate whether a key-less actor should get the key and whether they should take the longer path in order to avoid the danger zone.

Key Loss Probability

When the key loss probability is low (between 0 and 0.4), it is optimal for the agent to not consider the Lose-Key? tile as a risk when making decisions. Even when an agent with a key is directly above the key loss state, the agent's optimal strategy is to move directly across it in order to minimize step costs. A key-less agent should pursue the path to the key that requires the fewest steps, similar to the policy response to moderate positive terminal values. The probability of losing the key is low in comparison to the highly probable gains of reaching the positive terminal state.

With a moderate key loss probability (between 0.4 and 0.6), it is optimal to avoid the key loss state when the agent has the key directly to the north of the Lose-Key? tile. Additionally, in the northernmost row, it becomes optimal for agents east of the key tile to move towards the positive terminal tile through the danger zone instead of back towards the key loss state. The step costs and increased risk from this long path outweigh the risks of losing the key again. Thus the agent now focuses on not losing the key.

When the key loss probability is high (>=0.6), it is optimal for agents with a key to move to the east, even if they are in the north west corner of the grid. With such a high probability of the agent losing its key, the optimal strategy is to take the risk of going through the danger zone instead of venturing near the Lose-Key? tile. Generally, the key loss probability influences the extent to which agents are willing to risk losing the key or whether they will move through the danger zone and away from the Lose-Key? tile.

5. Further Work and Conclusions

We found clear patterns when comparing our VI and PI algorithms. While PI took far fewer iterations to find the optimal policy and exited immediately when that policy was found, each iteration in VI was significantly faster, resulting in an overall faster discovery of the optimal policy. The main drawback of our VI algorithm was that many of the last iterations of the algorithm had no effect on the actual policy. We therefore conclude that while the faster VI algorithm is clearly better for the test problem used here, and likely similar MDP problems, PI may be better for larger problems in which each iteration is more computationally valuable.

Tuning each parameter had significant effects on both the number of iterations required to solve the problem as well as the actual policies that are extracted from the algorithm. Decreasing discount factor (increasing the discount effect) devalues future rewards, and in doing so allows the VI algorithm to converge more quickly on an optimal policy that relaxes the drive towards the key and positive terminal states. This parameter thus clearly controls the agent's decisions around the trade-off between current and future rewards given its current state. Larger step costs also result in fewer iterations, as the plan is primarily dictated by reducing the punishment from each step. Intuitively, large step costs policies are focused on moving quickly towards any terminal state, while small step costs mean agents can focus on other factors in the environment like the actual values of those terminal states. Negative and positive terminal rewards both change the number of iterations and the resulting policy in environment-specific ways. Because the negative rewards stand between the agent's starting spot and the key to the positive reward, high negative rewards strongly drive agents away from that key and towards a simple, somewhat neutral path to the positive terminal tile. Strong positive rewards, on the other hand, prioritize complex paths than minimize risk while forcing the agent to jump through the hoops required to get the key and bring it to the south-east corner. Key loss probability changes the number of iterations and the final policy only slightly, causing the agent to value paths near the Lose-Key? tile less as the probability of losing the hard work of the key increases. Changes to both step costs, terminal rewards, and other specifics like key loss probability highlight how distinct environmental factors represent the goals of the agent as well as the paths they ultimately choose to try and reach those goals. The significant iteration number changes that result from some minor parameter changes (particularly for low step costs and high positive rewards) highlight how proper parameterization is an important part of maintaining algorithm efficiency. In more complex problems with many more states and terminal state interactions, exact environmental accuracy may have to be sacrificed in order to keep the algorithm within a reasonable number of iterations. While we only tested these parameter changes on our VI algorithm, we predict that parameterization affects the iteration behavior of PI in some similar ways. If PI is being used, parameterization is even more important, as each iteration takes significantly longer.

Despite our rigorous testing of each relevant parameter, there is much further work to be done to elaborate on the VI and PI algorithms presented here. Given the fact that PI was

significantly slower than VI, we suggest that some work go into implementing more elaborate policy iteration techniques. Particularly important are strategies that would mitigate the major computational cost of PI: the matrix solution step. Removing or minimizing the need to solve large matrices each iteration would significantly speed up the algorithm while still maintaining advantages like algorithm's ability to exit as soon as the optimal policy is found. One way to implement such an algorithm would be to combine PI and VI, allowing a minimal version of VI to estimate the utilities within each iteration of PI as a substitution for the matrix calculations. A second critical path of further work is more closely characterizing the ties between VI, PI, and the structure of the actual MDP environment used to test them. In our tests, we used only one relatively simple MDP problem. Different environments may play to the strengths of our algorithms differently. For example, larger problems with many more state variables may make PI completely intractable by drastically inflating the size of the necessary matrices, although overcoming those obstacles may reveal PI's fundamental strength in minimizing iteration number. Furthermore, changes to the relative number and distribution of terminal states may result in more computationally complicated problems. Characterizing the ways in which terminal states interact with one another in an MDP environment will add to our general understanding of how utility information diffuses across iterations in each algorithm. Such an understanding is critical if we are to design and implement algorithms that can handle the extremely complex and probabilistic environments of the real world.

Literature Cited

Lagoudakis, MG and R Parr. 2001. Model free least-squares policy iteration. NIPS 14: 345.

R Core Team. 2015. R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. URL https://www.R-project.org/.

Smith, T and R Simmons. 2004. Heuristic search value iteration for POMDPs. Proceedings of the 20th conference on Uncertainty in artificial intelligence. AUAI Press.