Markov Decision Processes

Son Ngo, Venecia Xu, Adela Yang

Abstract

Given a scenario involving a grid world with a starting position and and terminal states, different types of Markov Decision Processes (MDPs) are used to solve the problem. Specifically, Value and Policy Iteration are researched. The MDPs will be run with a different set of varying parameters and be compared for their efficiency. The number of iterations, policies and utilities of each state are recorded and analyzed.

1 Introduction

1.1 Problem Statement

In the assignment, we use MDPs to analyze a grid world. In this grid world, the agent always starts from the square marked Start.

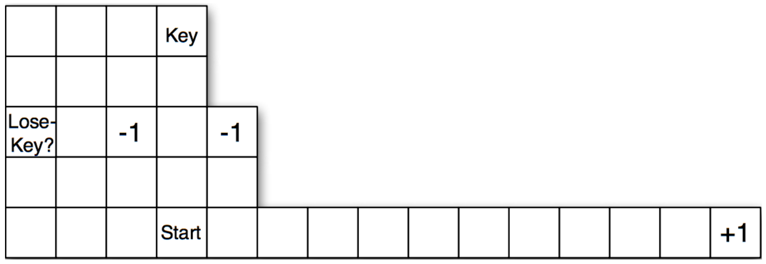


Figure 1: Grid World

There are four available actions:

1. `Go-North`
2. `Go-East`
3. `Go-South`
4. `Go-West`

With probability 0.8, the agent goes in the intended direction, with probability 0.1 it goes to the left instead, and with probability 0.1 it goes to the right. If it hits a wall, it remains in the square it started in. Once it enters the square marked `+1` or either of the two squares marked `-1`, no further movements are possible. This can be modeled by making all the transition probabilities from these squares 0.0. Rewards are specified as a function of the state the agent is in. There is a reward or step cost specified by the user associated with every state with the following exceptions. The reward associated with a state in which the agent occupies the square marked `+1` and has a `Key` is `+1`. The reward associated with a state in which the agent occupies the square marked `-1` is `-1`. To obtain a `Key`, the agent must visit the square marked `Key`. It can be assumed that there is a never-ending supply of `Keys` in this square so that the agent can go back and get another one if it loses the `Key` in its possession, which will happen with probability some specified probability if it enters the `Lose-Key?` square. The optimal policy will vary depending on the set of parameters being given to the problem, but in most cases, it will have the agent obtain a `Key` by visiting the Key space and obtain the positive reward by visiting the `+1` space with the `Key`. Notice that the possible paths to the Key square fall into three main categories:

1. The most direct path takes the agent between the two `-1` squares, both to and from the `Key` square, even though there is a risk of being trapped in one of the `-1` squares.
2. On the most indirect paths, the agent completely avoids the `-1` squares, but it must go through the `Lose-Key?` square, where it will lose any `Key` it has with a probability specified by the user.
3. There are in-between paths that pass through the square between the `Lose-Key?` square and the left-most `-1` square. Taking a path of this type lessens the agents risk of losing a `Key`, but exposes it to one of the `-1` squares.

Although this description of the grid world describes the terminal states' values as `+1` or `-1`, the values of the terminal states are actually determined by the user input.

2 Description of Algorithms

2.1 MDPs

Markov decision processes (MDP) plan by maximizing expected utilities, which involves mapping states to actions in order to find the optimal policy. For a given state and action, the expected utility is the sum of all probabilities of achieving another state from the current one, which should equal 1.0.

$$\sum_i T(s, a, s_i) = 1.0$$

The final solution(s) also involve a transition function, which maps state/action pairs to a probability distribution over states, and a reward function, which keeps track of the reward being granted at every time step. Given the true state utilities, we can prove that greedily choosing the next actions according to the transition function is optimal. There are several solutions to MDP, but a prevalent one learns by using a method for both the reward and transition functions. The true utility of a state can be found with the Bellman equation:

$$U_i(s) = R(s) + \gamma \sum_{s'} T(s, \pi(s), s') U_i(s')$$

2.2 Value Iteration

In synchronous, "in-place" value iteration, new utilities are calculated upon each iteration until such that the user defines as stable. The equals sign in the Bellman equation becomes an assignment operator, and the corresponding greedy policy is optimal, found by acheiving equilibrium after iterating through the modified Bellman equation. This is guaranteed to converge if $\gamma < 1.0$.

```
function VALUE-ITERATION ($mdp, \epsilon$) returns a utility function
  inputs: $mpd$, an MDP with states $S$, actions $A(s)$, transition model $P(s'|s,a)$, rewards $R(s)$, discount $\gamma$
          $\epsilon$, the maximum error allowed in the utility of any state
  local variables: $U, U'$, vectors of utilities for states in $S$, initially zero
                   $\delta$, the maximum change in the utility of any state in an iteration
  repeat
      $U \leftarrow U'; \delta \leftarrow 0$
      for each state $s$ in $S$ do
          $U'[s] \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s'|s,a)\, U'[s']$
          if $|U'[s] - U[s]| > \delta$ then $\delta \leftarrow |U'[s] - U[s]|$
  until $\delta < \epsilon(1-\gamma)/\gamma$
  return $U$
```

For as long as the error function is greater than δ, value iteration will continue to run. For every action in every state, we add up the sum of the utility of the connected states. If that sum is larger than our current saved sum, than the action defining the current policy is updated. δ and utility are updated in-place, and then the loop continues.

2.3 Policy Iteration

The idea behind Policy Iteration (PI) is to start with a random policy and then iteratively improve the current policy. It is more direct and requires solving systems of linear equations. There is a finite number of policies given that the system of linear equations can be solved for unique solution. Policy iteration involves two parts: policy evaluation, when a policy implies a utility function, and policy improvement, when a utility function implies a policy. The Bellman equation and current policy are used to set up a system of linear equations.

```
function POLICY-ITERATION ($mdp$) returns a policy
  inputs: $mpd$, an MDP with states $S$, actions $A(s)$, transition model $P(s'|s,a)$
  local variables: $U$, a vector of utilities for states in $S$, initially zero
                   $\pi$, a policy vector indexed by state, initially random
  repeat
     $U \leftarrow$ POLICY-EVALUATION $(\pi, U, mdp)$
     $unchanged? \leftarrow$ true
     for each state $s$ in $S$ do
        if $\max_{a \in A(s)} \sum_{s'} P(s'|s,a)\, U'[s'] > \sum_{s'} P(s'|s,\pi[s])\, U[s']$ then do
        $\pi[s] \leftarrow \operatorname{argmax}_{a \in A(s)} \sum_{s'} P(s'|s,a)\, U[s']$
        $unchanged? \leftarrow$ false
  until $unchanged?$
  return $\pi$
```

Each iteration, the policy is recomputed using matrices. For each state, all the states that can be reached from it contribute to the coefficients for the left side of the matrix. The right side is constructed for each individual state. The system is solved to find the updated utility given the policy. Once the policy evaluation is completed, we update the best action for a given state. In each state, we compare all the possible states each is connected to and choose the one with the highest utility. Once an iteration occurs where nothing has changed, policy iteration finishes.

3 Experimental Methodology

3.1 Experiment Description

Our experiments are divided into two main phases. In the first phase, we aim to compare two methods based on the number of iterations and the solution time of each method. In order to make sure that the comparison is consistent and comprehensive, we fix the parameters of the problem as the following:

- Discount factor: $\gamma = 0.999999$
- Maximum error: $\epsilon = 1e\text{-}6$
- Key loss probability: $P = 0.5$
- Positive terminal reward: 1.0
- Negative terminal reward: -1.0

There is one extra experiment in phase one to find the *actual* number of iterations each method takes to arrive at the optimal policy, regardless of the utility. The parameters used for this particular experiment will be the same as above. The total number of tests run in this phase is three.

In phase two, we carry out an intensive comparative statistics experiment. The purpose of this phase is to understand how sensitively the Value Iteration method behaves with respect to a change in one of the parameters while holding all the other parameters (listed above) constant. The maximum error will be held constant at $\epsilon = 1e\text{-}6$. We measure this "sensitivity" by first computing the change in the number of iterations the method takes. Then, we examine carefully the optimal policies resulting from the experiments, especially ones that indicate a change in key-acquisition and key-application strategy. In order to make sure the results from this phase are insightful, we need to capture all the possibilities by testing over a broad range of values for each parameter. Below are the values that we have used for this phase:

- Discount factor: $\gamma = 0.0, 0.5, 0.6, 0.7, 0.8, 0.95, 0.999999, 1.0$
- Step cost: $\epsilon = -100, -75, -50, -10, -5, -1, -0.5, -0.05, -0.04, 0.0, 0.001, 0.01$
- Key loss probability: $P = 0.0, 0.05, 0.15, 0.25, 0.5, 0.75, 0.95, 1.0$
- Positive terminal reward: $1, 5, 10, 15, 25, 50, 75, 100$
- Negative terminal reward: $-1, -5, -10, -15, -25, -50, -75, -100$

Each factor test runs with a specific array value, while keeping the other factors constant at their default value (see section 3.1). There will be 41 tests in total.

3.2 Test Cases

The tests are run on a Macintosh OS X Yosemite computer which has 16GB of memory and a 3.5 GHz Intel Core i7 processor. We have written a script that will carry out all the tests in one run and write out the results to two different files: `results.csv` and `results.txt`. The former stores the number of iterations for each different value of the parameters. The latter stores the associated optimal policy. We will use both of these files to analyze the comparative statics for phase two. For phase one, we simply run the `MDPSolver` with the specified arguments and make the comparison.

4 Results & Analysis

4.1 Phase one

The following are the results we obtain for running the tests for phase one.

Method	**# of iterations**	**Running time (ms)**	**# of iterations to reach optimal policy**
Value Iteration	64	5	31
Policy Iteration	12	16	12

Table 1: Phase one experiment results

We can see from Table 1 that the statistics for each solution method differ from each other, and they align with what we have studied in class. Value iteration (henceforth VI) method takes 64 iterations, which are a lot more than policy iteration (henceforth PI) method with only 12 iterations. However, the number of iterations alone does not fully reflect the effectiveness of the method. Therefore, we also look at the running time of each method, and find out that PI takes 16 milliseconds, 3 times slower than VI with only 5 milliseconds. Based on these two evaluations, it is hard to say which one is better than the other. VI would be much better for time-optimization, while PI is more preferable for iteration-optimization. The reason why we need both measurements is mainly due to the fact that one iteration of PI is not the same as one iteration of VI. Solving a system of linear equations requires a lot of effort, but at the same time it gets the method much closer to the solution after every iteration (order of convergence). So this is simply a trade-off between time and order of convergence.

Additionally, we see that VI converges to true optimal utility, while PI converges to true optimal policy. As a result, the number of iterations to reach optimal policy for PI is the same as the number of iterations of PI itself. However, for VI, it only takes 31 iterations, about half of its total number of iterations, to reach optimal policy. This is because VI does not use the current policy at any iteration, so it will not stop until the utility converges to within a certain amount, even if the policy is optimal. Therefore, it is not surprising that VI still takes more iterations to arrive at the optimal policy than 12 iterations of PI.

4.2 Phase two

The following are the results we obtain for running the tests for phase two.

DISCOUNT FACTOR TESTS

Value	0	0.5	0.6	0.7	0.8	0.95	0.999999
# of Iterations	1	19	22	26	30	45	64

KEY LOSS PROBABILITY TESTS

Value	0	0.05	0.15	0.25	0.5	0.75	0.95	1
# of Iterations	65	77	74	77	64	65	65	65

POSITIVE TERMINAL REWARD TESTS

Value	1	5	10	15	25	50	75	100
# of Iterations	64	132	311	318	325	334	338	341

NEGATIVE TERMINAL REWARD TESTS

Value	-100	-75	-50	-25	-15	-10	-5	-1
# of Iterations	71	71	71	71	71	71	113	64

STEP COST TESTS

Value	-100	-75	-50	-10	-5	-1	-0.5	-0.05	-0.04	0	0.001	0.01
# of Iterations	33	33	33	32	31	31	33	57	64	297	10898962	11354463

Table 2: Phase two experiment results

It is important to remember that for a specific factor test, all the other parameters are kept at their default values as specified in phase one. Therefore, the analysis of the results should be taken in comparison to the original problem's solution. We will see how the number of iterations and optimal policy change in each test, and determine the sensitivity of value iteration in the comparative statics.

4.2.1 Discount Factor Tests

In this section, we specifically look at the behavior of value iteration method as a response to a change in discount factor. To visualize the relationship between the discount factor and the number of iterations, we have constructed the following graph:

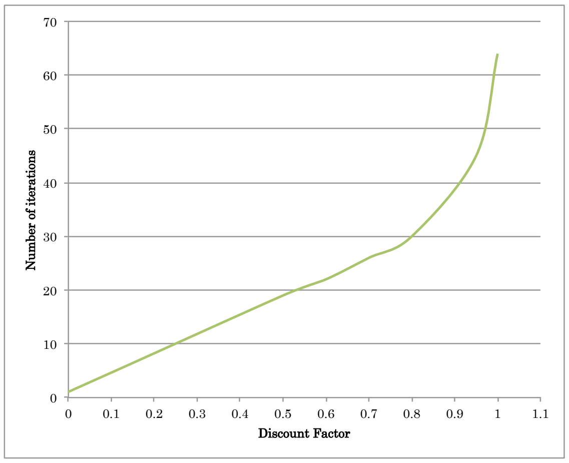


Figure 2: Relationship between discount factor and number of iterations

It can be seen from figure 2 that there exists an increasingly positive relationship between the discount factor and the number of iterations in VI method. This implies that if the discount factor is higher, the method will converge to the true utility with more iterations. A higher discount factor implies higher future value, so it may be more optimal to take more steps to obtain the `Key` then go for the terminal state. We can also see that the rate of change increases as well. From a discount factor of 0.0 to 0.75, the rate seems to be linear, but from 0.8 and above, the rate becomes approximately exponential. This behavior can be explained by the compounding effect of a positive factor that is less than one, so a discount factor of 0.7 would have a discounting effect only a little bit less than a discount factor of 0.5. We also notice that when the discount factor is 0, the method stops after 1 iteration. This makes sense since every other step will result in no extra utility except for the step cost. Therefore, the true utility of every state other than the terminal states is simply the step cost. Another special case is when the discount factor is 1 (i.e no discounting). This causes VI to run indefinitely. Reflecting this phenomenon on the graph, we see that the green curve asymptotically approaches a straight vertical line at discount factor of 1.

```
0.700000
(58) -0.13 (S),    (60) -0.13 (S),    (62) -0.13 (W),    (64) -0.13 (W),
(59) -0.13 (S),    (61) -0.13 (E),    (63) -0.13 (E),

(50) -0.13 (S),    (52) -0.13 (S),    (54) -0.13 (N),    (56) -0.13 (N),
(51) -0.13 (S),    (53) -0.13 (S),    (55) -0.13 (N),    (57) -0.13 (N),

(40) -0.13 (S),    (42) -0.13 (W),    (44) -1.00 (N),    (46) -0.25 (S),    (48) -1.00 (N),
(41) -0.13 (S),    (43) -0.13 (W),    (45) -1.00 (N),    (47) -0.25 (S),    (49) -1.00 (N),

(30) -0.13 (S),    (32) -0.13 (E),    (34) -0.13 (S),    (36) -0.13 (S),    (38) -0.12 (S),
(31) -0.13 (E),    (33) -0.13 (E),    (35) -0.13 (S),    (37) -0.13 (S),    (39) -0.13 (S),

( 0) -0.13 (E),    ( 2) -0.13 (E),    ( 4) -0.13 (E),    ( 6) -0.12 (E),    ( 8) -0.12 (E),    (10) -0.11 (E),    (12) -0.10 (E),    (14) -0.08 (E),
( 1) -0.13 (E),    ( 3) -0.13 (E),    ( 5) -0.13 (E),    ( 7) -0.13 (E),    ( 9) -0.13 (E),    (11) -0.13 (E),    (13) -0.13 (E),    (15) -0.13 (E),

    (16) -0.05 (E),    (18) -0.00 (E),    (20)  0.07 (E),    (22)  0.18 (E),    (24)  0.35 (E),    (26)  0.60 (E),    (28)  1.00 (N),
    (17) -0.13 (E),    (19) -0.12 (E),    (21) -0.12 (E),    (23) -0.11 (E),    (25) -0.09 (E),    (27) -0.07 (E),    (29) -0.04 (N),

0.950000
(58) -0.34 (S),    (60) -0.31 (S),    (62) -0.33 (E),    (64) -0.29 (S),
(59) -0.39 (E),    (61) -0.36 (E),    (63) -0.33 (E),

(50) -0.31 (E),    (52) -0.27 (S),    (54) -0.34 (N),    (56) -0.26 (S),
(51) -0.41 (N),    (53) -0.39 (N),    (55) -0.36 (N),    (57) -0.33 (N),

(40) -0.19 (S),    (42) -0.23 (S),    (44) -1.00 (N),    (46) -0.21 (S),    (48) -1.00 (N),
(41) -0.44 (N),    (43) -0.46 (W),    (45) -1.00 (N),    (47) -0.48 (N),    (49) -1.00 (N),

(30) -0.13 (S),    (32) -0.08 (S),    (34) -0.03 (S),    (36)  0.03 (S),    (38)  0.08 (S),
(31) -0.46 (N),    (33) -0.48 (N),    (35) -0.47 (S),    (37) -0.45 (S),    (39) -0.43 (S),

( 0) -0.09 (E),    ( 2) -0.04 (E),    ( 4)  0.02 (E),    ( 6)  0.08 (E),    ( 8)  0.14 (E),    (10)  0.21 (E),    (12)  0.28 (E),    (14)  0.35 (E),
( 1) -0.48 (N),    ( 3) -0.48 (E),    ( 5) -0.45 (E),    ( 7) -0.43 (E),    ( 9) -0.40 (E),    (11) -0.37 (E),    (13) -0.34 (E),    (15) -0.31 (E),

    (16)  0.43 (E),    (18)  0.51 (E),    (20)  0.60 (E),    (22)  0.69 (E),    (24)  0.78 (E),    (26)  0.89 (E),    (28)  1.00 (N),
    (17) -0.28 (E),    (19) -0.25 (E),    (21) -0.21 (E),    (23) -0.17 (E),    (25) -0.13 (E),    (27) -0.09 (E),    (29) -0.04 (N),
```

Figure 3: Optimal policies at $\gamma = 0.7$ and $\gamma = 0.9$

Given the starting point of the original problem, in both scenarios, there is no attempt to obtain the `Key`. However, the area of states in which obtaining the `Key` is optimal increases with the value of γ. When $\gamma = 0.7$, attempt to obtain the `Key` only appears in states that are very close to the `Key` in the first and second row and the negative terminal states. These states (squares) are: 61, 63, 55, 57, 45, 49. On the other hand, when $\gamma = 0.95$, there are a lot more attempt trying to obtain the `Key`, including every state on third row and above and the first two states of the forth row and the first state of fifth row. This trend aligns with our observation of the number of iterations above. When the discount factor is higher, obtaining the `Key` might be more beneficial since the future value is still high, allowing more optimal steps. Interestingly, the path the `Key` is not affected much by the discount factor, while the path from the `Key` avoids the path between the two negative terminal states when the discount factor is low.

4.2.2 Key Loss Probability Tests

In this section, we specifically look at the behavior of value iteration method as a response to a change in key lost probability. To visualize the relationship between key loss probability and number of iterations, we have constructed the following graph:

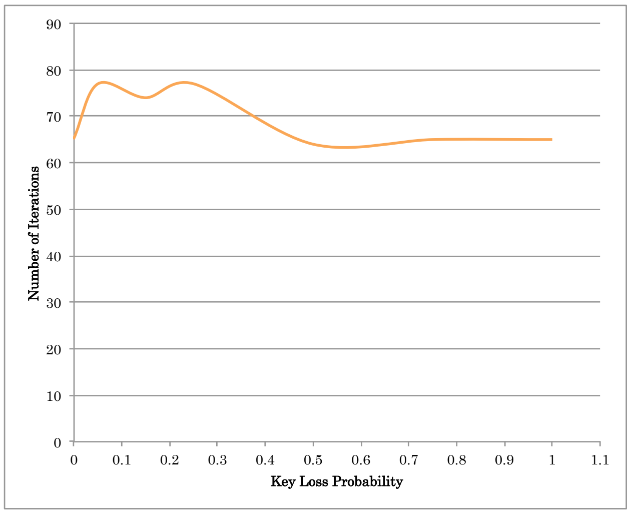


Figure 4: Relationship between key loss probability and number of iterations

From figure 4 we can see that key loss probability does not significantly affect the number of iterations of VI. A low probability key loss (less than 0.5) boosts the number of iterations to around 77, compared to 64 in the original problem. When the probability increases to 0.5 and above, the number of iterations drops back to 65 and seems to remain stable afterwards. This behavior is reasonable and intuitive because there are more steps needed to be taken when key loss probability is lower once the agent has obtained the `Key`. In the original problem, we weigh the negative terminal states and the key loss square equally. However, with low key loss probability, it is relatively easier to avoid the negative reward states than losing the `Key`, so the new optimal path from the `Key` is to `Go-West` and take the path in-between the `Lose-Key?` square and the left-most `-1` square to reach the positive terminal state. We can see this pattern more clearly by looking at the change in the optimal policy.

```
0.250000
(58) -0.06 (S),    (60) -0.04 (S),    (62) -0.10 (W),    (64) -0.05 (S),
(59) -0.22 (E),    (61) -0.16 (E),    (63) -0.11 (E),

(50) -0.01 (S),    (52)  0.01 (S),    (54) -0.12 (N),    (56)  0.00 (S),
(51) -0.26 (N),    (53) -0.21 (N),    (55) -0.16 (N),    (57) -0.11 (N),

(40)  0.15 (S),    (42)  0.08 (S),    (44) -1.00 (N),    (46)  0.07 (S),    (48) -1.00 (N),
(41) -0.32 (N),    (43) -0.34 (N),    (45) -1.00 (N),    (47) -0.33 (N),    (49) -1.00 (N),

(30)  0.22 (S),    (32)  0.27 (S),    (34)  0.33 (S),    (36)  0.39 (S),    (38)  0.44 (S),
(31) -0.37 (N),    (33) -0.40 (N),    (35) -0.51 (W),    (37) -0.40 (N),    (39) -0.51 (W),

( 0)  0.27 (E),    ( 2)  0.32 (E),    ( 4)  0.38 (E),    ( 6)  0.44 (E),    ( 8)  0.49 (E),    (10)  0.55 (E),    (12)  0.60 (E),    (14)  0.65 (E),
( 1) -0.42 (N),    ( 3) -0.45 (N),    ( 5) -0.51 (W),    ( 7) -0.47 (N),    ( 9) -0.52 (W),    (11) -0.49 (E),    (13) -0.44 (E),    (15) -0.39 (E),

    (16)  0.70 (E),    (18)  0.75 (E),    (20)  0.80 (E),    (22)  0.85 (E),    (24)  0.90 (E),    (26)  0.95 (E),    (28)  1.00 (N),
    (17) -0.34 (E),    (19) -0.29 (E),    (21) -0.24 (E),    (23) -0.19 (E),    (25) -0.14 (E),    (27) -0.09 (E),    (29) -0.04 (N),

0.500000
(58) -0.11 (S),    (60) -0.06 (S),    (62) -0.11 (E),    (64) -0.05 (S),
(59) -0.22 (E),    (61) -0.17 (E),    (63) -0.11 (E),

(50) -0.06 (E),    (52) -0.00 (S),    (54) -0.12 (N),    (56)  0.00 (S),
(51) -0.27 (N),    (53) -0.22 (N),    (55) -0.16 (N),    (57) -0.11 (N),

(40)  0.14 (S),    (42)  0.07 (S),    (44) -1.00 (N),    (46)  0.07 (S),    (48) -1.00 (N),
(41) -0.32 (N),    (43) -0.34 (N),    (45) -1.00 (N),    (47) -0.33 (N),    (49) -1.00 (N),

(30)  0.22 (S),    (32)  0.27 (S),    (34)  0.33 (S),    (36)  0.39 (S),    (38)  0.44 (S),
(31) -0.37 (N),    (33) -0.40 (N),    (35) -0.51 (W),    (37) -0.41 (N),    (39) -0.52 (W),

( 0)  0.27 (E),    ( 2)  0.32 (E),    ( 4)  0.38 (E),    ( 6)  0.44 (E),    ( 8)  0.49 (E),    (10)  0.55 (E),    (12)  0.60 (E),    (14)  0.65 (E),
( 1) -0.43 (N),    ( 3) -0.46 (N),    ( 5) -0.51 (W),    ( 7) -0.47 (N),    ( 9) -0.52 (W),    (11) -0.49 (E),    (13) -0.44 (E),    (15) -0.39 (E),

    (16)  0.70 (E),    (18)  0.75 (E),    (20)  0.80 (E),    (22)  0.85 (E),    (24)  0.90 (E),    (26)  0.95 (E),    (28)  1.00 (N),
    (17) -0.34 (E),    (19) -0.29 (E),    (21) -0.24 (E),    (23) -0.19 (E),    (25) -0.14 (E),    (27) -0.09 (E),    (29) -0.04 (N),
```

Figure 5: Optimal policies at key loss probability of 0.25 and 0.5

From figure 5 we can see that general optimal policy does not differ by much from the original optimal path, except for one small thing. At state 62, when the probability is 0.25 (i.e less than 0.5), the optimal action is to `Go-West` if the agent has the `Key` already, while in the original problem and problems with key loss probability greater than 0.5, the optimal action is to `Go-East`. This implies that once the agent has acquired the `Key`, the key loss probability is low enough that it is better to take more steps to avoid path between the two negative terminal states, which also results in more number of iterations as well. When the key loss probability is very high, then it is more optimal for the agent to take the path through the two negative terminal states to get to the positive terminal state.

4.2.3 Positive Terminal Reward Tests

In this section, we specifically look at the behavior of value iteration method as a response to a change in the positive terminal reward. To visualize the relationship between this reward and number of iterations, we have constructed the following graph:

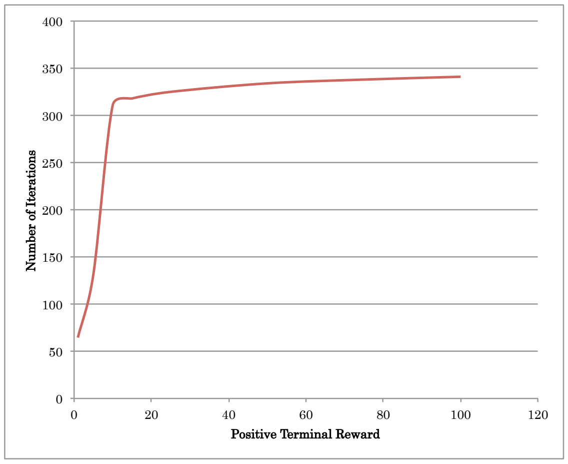


Figure 6: Relationship between positive terminal reward and number of iterations

In the figure above, we can see a very interesting relationship between the positive terminal reward and number of iterations. When the positive reward increases from 1 to 10, the number of iterations increases dramatically from 64 to around 300. Then, any increase in the reward only marginally increases the number of iterations. This phenomenon can be explained by the fact that obtaining the key is much more preferred now, as well as avoiding the negative terminal states. In order to see this more clearly, we look at the change in the optimal policies when the reward changes.

```
1.000000
(58) -0.11 (S),    (60) -0.06 (S),    (62) -0.11 (E),    (64) -0.05 (S),
(59) -0.22 (E),    (61) -0.17 (E),    (63) -0.11 (E),

(50) -0.06 (E),    (52) -0.00 (S),    (54) -0.12 (N),    (56)  0.00 (S),
(51) -0.27 (N),    (53) -0.22 (N),    (55) -0.16 (N),    (57) -0.11 (N),

(40)  0.14 (S),    (42)  0.07 (S),    (44) -1.00 (N),    (46)  0.07 (S),    (48) -1.00 (N),
(41) -0.32 (N),    (43) -0.34 (N),    (45) -1.00 (N),    (47) -0.33 (N),    (49) -1.00 (N),

(30)  0.22 (S),    (32)  0.27 (S),    (34)  0.33 (S),    (36)  0.39 (S),    (38)  0.44 (S),
(31) -0.37 (N),    (33) -0.40 (N),    (35) -0.51 (W),    (37) -0.41 (N),    (39) -0.52 (W),

( 0)  0.27 (E),    ( 2)  0.32 (E),    ( 4)  0.38 (E),    ( 6)  0.44 (E),    ( 8)  0.49 (E),    (10)  0.55 (E),    (12)  0.60 (E),    (14)  0.65 (E),
( 1) -0.43 (N),    ( 3) -0.46 (N),    ( 5) -0.51 (W),    ( 7) -0.47 (N),    ( 9) -0.52 (W),    (11) -0.49 (E),    (13) -0.44 (E),    (15) -0.39 (E),

    (16)  0.70 (E),    (18)  0.75 (E),    (20)  0.80 (E),    (22)  0.85 (E),    (24)  0.90 (E),    (26)  0.95 (E),    (28)  1.00 (N),
    (17) -0.34 (E),    (19) -0.29 (E),    (21) -0.24 (E),    (23) -0.19 (E),    (25) -0.14 (E),    (27) -0.09 (E),    (29) -0.04 (N),

10.000000
(58)  8.48 (S),    (60)  8.48 (S),    (62)  8.42 (W),    (64)  8.37 (W),
(59)  8.20 (E),    (61)  8.26 (E),    (63)  8.31 (E),

(50)  8.53 (S),    (52)  8.54 (S),    (54)  8.39 (N),    (56)  8.33 (N),
(51)  8.16 (N),    (53)  8.21 (N),    (55)  8.26 (N),    (57)  8.31 (N),

(40)  9.06 (S),    (42)  8.61 (W),    (44) -1.00 (N),    (46)  7.27 (S),    (48) -1.00 (N),
(41)  8.10 (N),    (43)  8.06 (W),    (45) -1.00 (N),    (47)  6.41 (N),    (49) -1.00 (N),

(30)  9.22 (S),    (32)  9.27 (S),    (34)  9.33 (S),    (36)  9.39 (S),    (38)  9.44 (S),
(31)  8.05 (N),    (33)  8.00 (N),    (35)  7.85 (S),    (37)  7.79 (S),    (39)  7.74 (S),

( 0)  9.27 (E),    ( 2)  9.32 (E),    ( 4)  9.38 (E),    ( 6)  9.44 (E),    ( 8)  9.49 (E),    (10)  9.55 (E),    (12)  9.60 (E),    (14)  9.65 (E),
( 1)  7.99 (N),    ( 3)  7.95 (N),    ( 5)  7.89 (W),    ( 7)  7.84 (W),    ( 9)  7.78 (W),    (11)  7.73 (W),    (13)  7.68 (W),    (15)  7.63 (W),

    (16)  9.70 (E),    (18)  9.75 (E),    (20)  9.80 (E),    (22)  9.85 (E),    (24)  9.90 (E),    (26)  9.95 (E),    (28) 10.00 (N),
    (17)  7.58 (W),    (19)  7.53 (W),    (21)  7.48 (W),    (23)  7.43 (W),    (25)  7.38 (W),    (27)  7.33 (W),    (29) -0.04 (N),
```

Figure 7: Optimal policies for positive reward of 1 and 10

In figure 7, we directly compare the change in optimal policy to that of the original problem. As mentioned before, an increase in the positive reward will encourage the agent to obtain the key, and to avoid the negative terminal state. When the positive reward is 1, the agent ceases trying to obtain the key when it enters the sixth square on the fifth row (i.e the tunnel) even if the agent does not have the key. However, if the positive reward becomes as high as 10, the agent will go all the way back to obtain the key even if it is next to positive terminal state already. In fact, the agent tries to obtain the `Key` everywhere on this map. It is also important to note that the path to the key avoids the negative terminal reward as much as possible. Instead of going through the square between the two negative terminal states, the agent will `Go-West` to take the “in-between” path between the key loss square and the left-most `-1` square. This behavior also confirms the negatively relative relationship between step cost and the positive reward. When the positive reward increases, even if step cost stays constant, its relative decreases compared to the positive reward. Therefore, it becomes more preferable to obtain the key when the relative step cost decreases. The effect of step cost will be discussed more in detail in section 4.2.5.

4.2.4 Negative Terminal Reward Tests

In this section, we specifically look at the behavior of value iteration method as a response to a change in the negative terminal reward. To visualize the relationship between this reward and number of iterations, we have constructed the following graph:

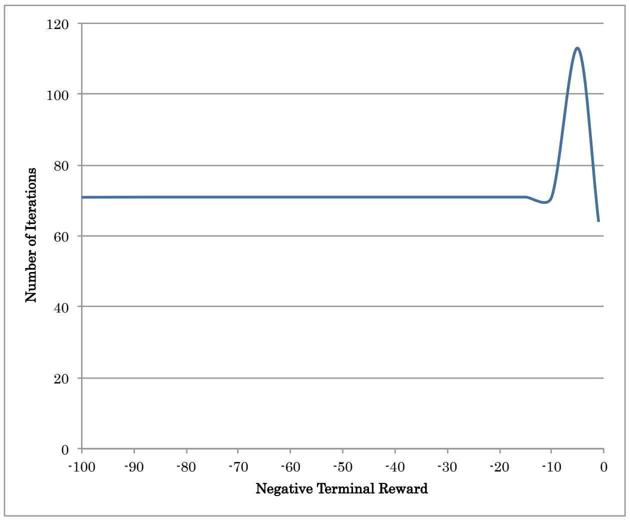


Figure 8: Relationship between negative9 terminal reward and number of iterations

In figure 8, we observe a peak in the number of iterations at 113 when the negative reward is -5, and then it stabilizes at 71 when the negative reward decreases to below -5. It might be the case that when the negative reward is close to -5, the utility’s values are quite similar around the negative terminal state, so it takes more iterations to have the utilities at those squares converge. But when the reward gets more negative, certain squares’ values dominate the others, so it eventually takes less iteration since the values around the negative terminal states are quite distinctive now.

```
-1.000000
(58) -0.11 (S),    (60) -0.06 (S),    (62) -0.11 (E),    (64) -0.05 (S),
(59) -0.22 (E),    (61) -0.17 (E),    (63) -0.11 (E),

(50) -0.06 (E),    (52) -0.00 (S),    (54) -0.12 (N),    (56)  0.00 (S),
(51) -0.27 (N),    (53) -0.22 (N),    (55) -0.16 (N),    (57) -0.11 (N),

(40)  0.14 (S),    (42)  0.07 (S),    (44) -1.00 (N),    (46)  0.07 (S),    (48) -1.00 (N),
(41) -0.32 (N),    (43) -0.34 (N),    (45) -1.00 (N),    (47) -0.33 (N),    (49) -1.00 (N),

(30)  0.22 (S),    (32)  0.27 (S),    (34)  0.33 (S),    (36)  0.39 (S),    (38)  0.44 (S),
(31) -0.37 (N),    (33) -0.40 (N),    (35) -0.51 (W),    (37) -0.41 (N),    (39) -0.52 (W),

( 0)  0.27 (E),    ( 2)  0.32 (E),    ( 4)  0.38 (E),    ( 6)  0.44 (E),    ( 8)  0.49 (E),    (10)  0.55 (E),    (12)  0.60 (E),    (14)  0.65 (E),
( 1) -0.43 (N),    ( 3) -0.46 (N),    ( 5) -0.51 (W),    ( 7) -0.47 (N),    ( 9) -0.52 (W),    (11) -0.49 (E),    (13) -0.44 (E),    (15) -0.39 (E),

    (16)  0.70 (E),    (18)  0.75 (E),    (20)  0.80 (E),    (22)  0.85 (E),    (24)  0.90 (E),    (26)  0.95 (E),    (28)  1.00 (N),
    (17) -0.34 (E),    (19) -0.29 (E),    (21) -0.24 (E),    (23) -0.19 (E),    (25) -0.14 (E),    (27) -0.09 (E),    (29) -0.04 (N),

-10.000000
(58) -0.50 (S),    (60) -0.51 (S),    (62) -0.56 (W),    (64) -0.62 (W),
(59) -0.78 (E),    (61) -0.73 (E),    (63) -0.67 (E),

(50) -0.45 (S),    (52) -0.45 (S),    (54) -0.60 (N),    (56) -0.66 (N),
(51) -0.83 (E),    (53) -0.78 (N),    (55) -0.72 (N),    (57) -0.67 (N),

(40)  0.06 (S),    (42) -0.38 (W),    (44) -10.00 (N),    (46) -1.73 (S),    (48) -10.00 (N),
(41) -0.87 (S),    (43) -0.89 (W),    (45) -10.00 (N),    (47) -2.56 (S),    (49) -10.00 (N),

(30)  0.22 (S),    (32)  0.27 (S),    (34)  0.33 (S),    (36)  0.39 (S),    (38)  0.44 (S),
(31) -0.82 (S),    (33) -0.77 (S),    (35) -0.71 (S),    (37) -0.65 (S),    (39) -0.60 (S),

( 0)  0.27 (E),    ( 2)  0.32 (E),    ( 4)  0.38 (E),    ( 6)  0.44 (E),    ( 8)  0.49 (E),    (10)  0.55 (E),    (12)  0.60 (E),    (14)  0.65 (E),
( 1) -0.77 (E),    ( 3) -0.72 (E),    ( 5) -0.66 (E),    ( 7) -0.60 (E),    ( 9) -0.55 (E),    (11) -0.49 (E),    (13) -0.44 (E),    (15) -0.39 (E),

    (16)  0.70 (E),    (18)  0.75 (E),    (20)  0.80 (E),    (22)  0.85 (E),    (24)  0.90 (E),    (26)  0.95 (E),    (28)  1.00 (N),
    (17) -0.34 (E),    (19) -0.29 (E),    (21) -0.24 (E),    (23) -0.19 (E),    (25) -0.14 (E),    (27) -0.09 (E),    (29) -0.04 (N),
```

Figure 9: Optimal policies for negative reward of -5 and -10

We can also see a change in the optimal policy as a response to the change in negative reward in figure 9. We can see a stronger tendency to avoid the negative terminal states when the negative reward increases (in terms of absolute value). When negative reward is `-1`, the path to obtain the `Key` still includes that path that lies between the two negative terminal states. However, when negative rewards becomes -10, the optimal path switches completely to the path between the key loss square and the left-most `-1` square. We can also see that the agent only tries to obtain the key if it starts at squares on row 1 and 2. Given the original starting point of the problem, the agent does not try to obtain the `Key`.

4.2.5 Step Cost Tests

In this section, we specifically look at the behavior of value iteration method as a response to a change in the step cost. To visualize the relationship between step cost and number of iterations, we have constructed the following graph:

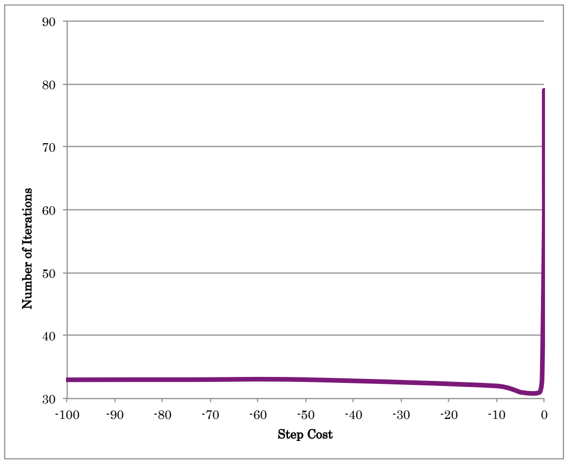


Figure 10: Relationship between step cost and number of iterations

From figure 10, we see a convergence towards 32 iterations when the step cost gets higher and higher (in terms of absolute value). This is because when the step cost is big enough, the agent will completely forgo obtaining the key and simply tries to reach terminal state as soon as possible. On the other hand, when the step cost gets closer to 0, the number of iterations increases drastically, and becomes extremely large - to the millions - when the step cost is positive. Intuitively, when the step cost is positive, the agent will try to take as long as possible to avoid the terminal state, even the positive one. The only thing that keeps the agent from going on forever is the discount factor, which will drives the marginal utility of taking one more step to 0. Both of these behaviors can be seen more clearly when we look at the optimal policies at different values of step cost.

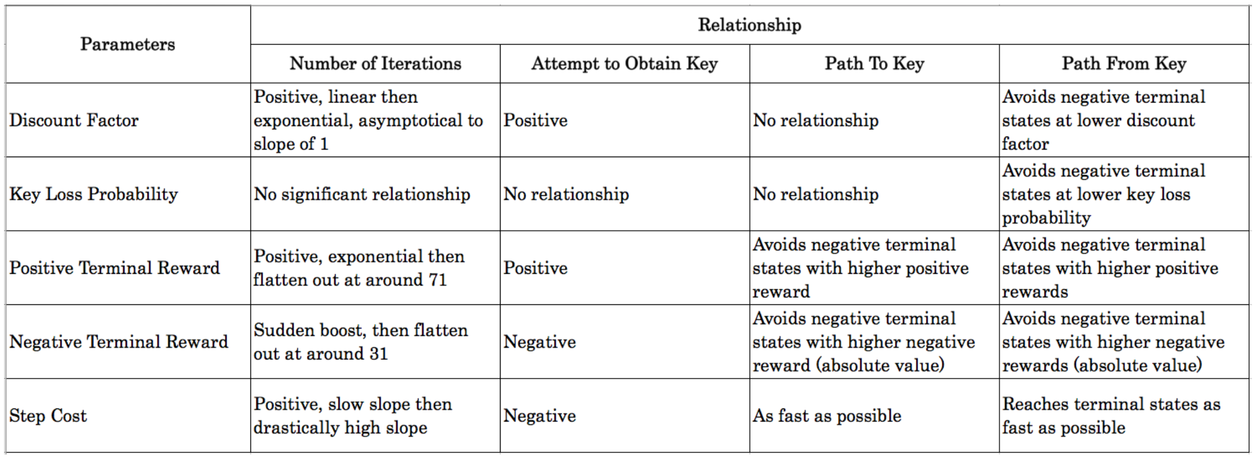
```
-0.500000
(58) -3.58 (E),    (60) -3.03 (S),    (62) -2.51 (S),    (64) -2.89 (S),
(59) -3.58 (E),    (61) -3.03 (S),    (63) -2.51 (S),

(50) -3.03 (E),    (52) -2.40 (E),    (54) -1.77 (S),    (56) -2.31 (S),
(51) -3.03 (E),    (53) -2.40 (E),    (55) -1.77 (S),    (57) -2.31 (S),

(40) -2.53 (E),    (42) -1.78 (E),    (44) -1.00 (N),    (46) -1.76 (E),    (48) -1.00 (N),
(41) -2.53 (E),    (43) -1.78 (E),    (45) -1.00 (N),    (47) -1.76 (E),    (49) -1.00 (N),

(30) -3.02 (E),    (32) -2.39 (E),    (34) -1.76 (N),    (36) -2.25 (N),    (38) -1.69 (N),
(31) -3.02 (E),    (33) -2.39 (E),    (35) -1.76 (N),    (37) -2.25 (N),    (39) -1.69 (N),

( 0) -3.58 (E),    ( 2) -3.02 (N),    ( 4) -2.49 (N),    ( 6) -2.79 (N),    ( 8) -2.44 (N),    (10) -3.07 (W),    (12) -3.69 (W),    (14) -3.37 (E),
( 1) -3.58 (E),    ( 3) -3.02 (N),    ( 5) -2.49 (N),    ( 7) -2.79 (N),    ( 9) -2.44 (N),    (11) -3.07 (W),    (13) -3.69 (W),    (15) -4.32 (W),

    (16) -2.75 (E),    (18) -2.12 (E),    (20) -1.50 (E),    (22) -0.88 (E),    (24) -0.25 (E),    (26)  0.37 (E),    (28)  1.00 (N),
    (17) -4.25 (E),    (19) -3.62 (E),    (21) -3.00 (E),    (23) -2.37 (E),    (25) -1.75 (E),    (27) -1.12 (E),    (29) -0.50 (N),

-0.050000
(58) -0.32 (E),    (60) -0.26 (S),    (62) -0.28 (E),    (64) -0.22 (S),
(59) -0.43 (E),    (61) -0.36 (E),    (63) -0.29 (E),

(50) -0.26 (E),    (52) -0.19 (S),    (54) -0.29 (E),    (56) -0.15 (S),
(51) -0.48 (N),    (53) -0.42 (N),    (55) -0.35 (N),    (57) -0.29 (N),

(40) -0.07 (S),    (42) -0.11 (S),    (44) -1.00 (N),    (46) -0.06 (S),    (48) -1.00 (N),
(41) -0.54 (N),    (43) -0.54 (N),    (45) -1.00 (N),    (47) -0.48 (N),    (49) -1.00 (N),

(30)  0.03 (S),    (32)  0.09 (S),    (34)  0.16 (S),    (36)  0.23 (S),    (38)  0.30 (S),
(31) -0.61 (N),    (33) -0.61 (N),    (35) -0.67 (E),    (37) -0.57 (N),    (39) -0.67 (W),

( 0)  0.09 (E),    ( 2)  0.16 (E),    ( 4)  0.23 (E),    ( 6)  0.30 (E),    ( 8)  0.37 (E),    (10)  0.44 (E),    (12)  0.50 (E),    (14)  0.56 (E),
( 1) -0.67 (N),    ( 3) -0.67 (N),    ( 5) -0.70 (E),    ( 7) -0.64 (N),    ( 9) -0.67 (E),    (11) -0.61 (E),    (13) -0.55 (E),    (15) -0.49 (E),

    (16)  0.62 (E),    (18)  0.69 (E),    (20)  0.75 (E),    (22)  0.81 (E),    (24)  0.87 (E),    (26)  0.94 (E),    (28)  1.00 (N),
    (17) -0.42 (E),    (19) -0.36 (E),    (21) -0.30 (E),    (23) -0.24 (E),    (25) -0.17 (E),    (27) -0.11 (E),    (29) -0.05 (N),
```

Figure 11: Optimal policies when step cost is -0.05 and -0.5

In figure 11, we can see that the optimal policy when step cost is -0.05 is quite similar to the original problem with step cost of -0.04, where the agent will try to obtain the `Key` and then go to the positive terminal state. However, only with a small increase to -0.5, the optimal policy changes significantly. The agent now tries to reach any terminal state as soon as possible, and obtaining the `Key` is irrelevant. Given the original starting point, the agent will simply try to `Go-North` to finish at one of the negative terminal states. This behavior makes sense because the step cost is now the dominating cost, relatively to the other parameters.

The positive step cost case is not being discussed here since the number of iterations becomes significantly large, and not applicable to real life problem. We still ran two tests with step cost of 0.001 and 0.01 and obtained 10898962 and 11354463 iterations respectively. When there is no step cost (i.e step cost is 0), the number of iterations is 297.

4.2.6 Analysis Summary



Parameters	Relationship			
	Number of Iterations	Attempt to Obtain Key	Path To Key	Path From Key
Discount Factor	Positive, linear then exponential, asymptotical to slope of 1	Positive	No relationship	Avoids negative terminal states at lower discount factor
Key Loss Probability	No significant relationship	No relationship	No relationship	Avoids negative terminal states at lower key loss probability
Positive Terminal Reward	Positive, exponential then flatten out at around 71	Positive	Avoids negative terminal states with higher positive reward	Avoids negative terminal states with higher positive rewards
Negative Terminal Reward	Sudden boost, then flatten out at around 31	Negative	Avoids negative terminal states with higher negative reward (absolute value)	Avoids negative terminal states with higher negative rewards (absolute value)
Step Cost	Positive, slow slope then drastically high slope	Negative	As fast as possible	Reaches terminal states as fast as possible

Figure 12: Summary of comparative statics

The experiments have given us a thorough comparative statics of value iteration method, where the endogenous variables are the number of iterations and the optimal policy. We can see in figure 12 that the relationship between the parameters and the behavior of VI aligns with our intuition. We also realize the path to the `Key` is not necessarily always the same as the path from the `Key`. This is because the `Lose-Key?` square is only effective when the agent has already acquired the `Key`. Based on the results of the experiment, we can conclude that the method behaves quite sensitively to a change in the parameters in general. In section 5, we will briefly discuss how we can expand this experiment to explore more properties of value iteration, and see how the method behaves to a more complex set of parameters.

5 Further Work

Although there was a wide range of values being tested for the discount factor, key loss probability, positive terminal reward, negative terminal reward and step costs, further testing with an even more diverse range of values can be tested. In the experiment, we are only looking at the effect of *one* parameter while holding all the other parameters constant. However, we can test different combinations of parameters' values and look at the change in the behavior of value iteration method instead of testing just one parameter. Additionally, we can apply the same experiments to policy iteration as well. Another MDP-method we can explore is Q-Learning. It is a form of model-free reinforcement learning, in which the transition function and reward functions are unknown. Its components consists of an adaptive heuristic critic method involving modified policy iteration that starts off with an arbitrary policy to start iterating on. Q-Learning also uses temporal difference learning and eligibility traces which replaces the policy evaluation phase. It would be interesting to see how this compares to VI and PI.

6 Conclusions

Value iteration and policy iteration are fairly balanced in that PI takes fewer iterations, but VI has a slightly faster runtime. When we took a closer look at how parameters could affect the number of iterations for VI, we found that iterations often stayed stable except for edge cases near zero or high numbers. Key loss probability had little significant change or patterns. Step cost is predictably stable, besides the sharp spike that occurs with very low step cost. Positive terminal reward actually only changes because of decisions revolving around the key. Negative terminal reward spikes around -5, possibly because that is similar to the utility values around the negative terminal state. Discount factor was the exception with an increasingly positive relationship with numbers of iteration. When the discount factor increases, it becomes more optimal to obtain the key.

Only discount factor and positive terminal reward affect whether the agent tries to obtain the key. Interestingly, however, discount factor had no impact on the path to the key while positive terminal reward did. The impact each parameter had on the path away from the key was fairly uniform. The path just avoids negative terminal states at lower parameter values, except for the reward parameters in which higher values were avoided.