# Value iteration and policy iteration algorithms for Markov decision problem

Elena Pashenkova Irina Rish
Rina Dechter
Department of Information and Computer Science,
University of California at Irvine
Irvine, CA 92717

April 18, 1996

**Abstract**

In this paper we consider computational aspects of decision-theoretic planning modeled by Markov decision processes (MDPs). Commonly used algorithms, such as *value iteration (VI)* [Bellman, 1957] and several versions of *modified policy iteration (MPI)* [Puterman, 1994] (a modification of the original Howard's *policy iteration (PI)* [Howard, 1960]), are compared on a class of problems from the motion planning domain. Policy iteration and its modifications are usually recommended as algorithms demonstrating a better performance than value iteration [Russel & Norvig, 1995], [Puterman, 1994]. However, our results show that their performance is not always superior and depends on the parameters of a problem and the parameters of the algorithms, such as number of iterations in the *value determination* procedure in MPI. Moreover, policy iteration applied to non-discounted models without special restrictions might not even converge to an optimal policy, as in case of the policy iteration algorithm introduced in [Russel & Norvig, 1995]. We also introduce a new stopping criterion into value iteration based on policy changes. The *combined value-policy iteration (CVPI)* algorithm proposed in the paper implements this criterion and generates an optimal policy faster then both policy and value iteration algorithms.

## 1 Introduction

We consider a commonly used approach to decision theoretic planning based on the theory of Markov decision processes (MDPs). Environment is described by a set of *states* $S$. A decision-making agent observes the current state of the environment and chooses an *action* from a given set $A$. We assume that the environment is *completely observable*, i.e. observations give correct information about the state of the world. Nondeterministic effects of actions are described by the set of *transition probabilities* $P_{ij}^{a}$, which are defined for every pair of states $s_i$ and $s_j$ and for every action $a$. $P_{ij}^{a}$ is the probability of reaching state $s_j$ from state $s_i$ after taking an action $a$. There is a *reward function* $R$ defined on $S$ which specifies an immediate reward of being in a given state $s \in S$. A *policy* is a mapping from the set of states to the set of actions $\pi : S \rightarrow A$ describing an action to be taken given the current state of the environment. Therefore, an MDP is a stochastic automaton. Decision making is based on *maximum expected utility (MEU)* approach. Given a policy, we compute an *expected utility* $U$ of each state $s_i$ according to the formula:

$$U(s_i) = R(s_i) + \sum_j P_{ij}^{pi(s_i)} U(s_j)$$

The goal is to find an *optimal policy* $\pi^*$ which maximizes the expected utility of each state, i.e. given any other policy $\pi$ and any $s$, $\pi^*(s) \geq \pi(s)$. There are several algorithms commonly used for determining an optimal policy for decision problems: *value iteration (VI)* [Bellman, 1957] and *policy iteration (PI)* [Howard, 1960], as well as a variation called *modified policy iteration (MPI)* [Puterman, 1994].
We compare the performance of those algorithms on a motion planning problem described in [Russel & Norvig, 1995].
The environment is described by a grid (see example in figure 1) of size $NXM$. Some squares in the grid represent obstacles such as a wall. The other squares correspond to the states of the world. There is a start state an agent is initially in, and a set of terminal states. An agent is moving from one square to another until a terminal state is reached. In each location, there are four available actions, *North, South, East* and *West*, that supposedly move an agent one square in the intended direction, but achieving the

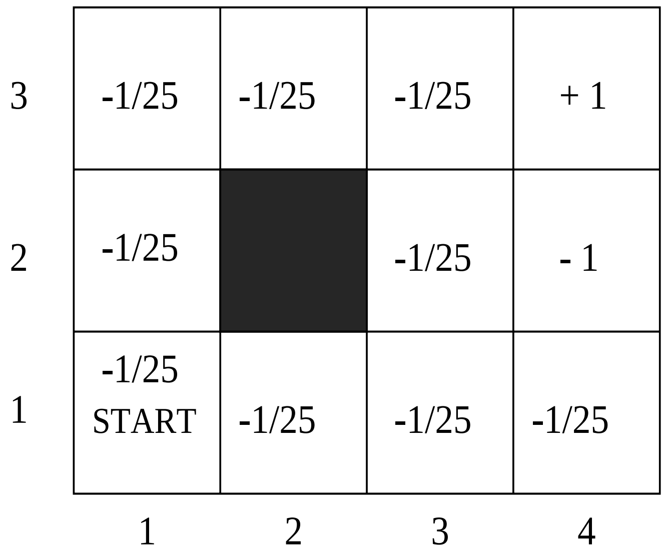

Figure 1: An example of an environment with reward function defined on states

desired effect only with some probability $p$. The rest of the time, the action moves the agent at right angles to the intended direction. For example, if the agent chooses the action *North*, it moves *North* with probability $p$, and with probability $(1-p)/2$ it moves *East* or *West*. If the agent cannot move in the intended direction because of an obstacle such as a wall, it stays in the current location. There is a reward function defined on all states, and the goal is to find an optimal policy.

VI and PI algorithms were implemented according to [Russel & Norvig, 1995]. VI does not differ from the original algorithm proposed in [Bellman, 1957]. PI was given in [Russel & Norvig, 1995] in two versions, as the original Howard's algorithm and as a variation on MPI described in [Puterman, 1994] (although the name of the last algorithm was not explicitly mentioned). We experimented with the second version of PI naming it MPI. Then we tried another modification of MPI proposed in [Puterman, 1994], which fixes the number of steps in value determination part of the algorithm. Finally, we propose a new algorithm called *combined value-policy iteration (CVPI)* which introduces a better stopping criterion for the VI algorithm.

In the next sections we give a brief description of the algorithms we have experimented with.

## 2 Value Iteration Algorithm

VI [Bellman, 1957] is an iterative procedure that calculates the expected utility of each state using the utilities of the neighboring states until the utilities calculated on two successive steps are close enough, i.e.

$$\max_{s_i} |U(s_i) - U'(s_i)| < \epsilon,$$

where $\epsilon$ is a predefined *threshold* value. The smaller the threshold is, the higher is the *precision* of the algorithm. Given certain conditions on the environment, the utility values are guaranteed to converge.
Given a utility matrix we can calculate a corresponding policy according to the *maximum expected utility* principle, i.e. choosing

$$\pi^*(s_i) = argmax_a \sum_j P_{ij}^a U(s_j)$$

as an optimal policy.
Here is a schematic description of the VI algorithm:

```
function $VALUE - ITERATION(P, R)$ returns a utility matrix
    inputs:    $P$, a transition-probability matrix
               $R$, a reward matrix
    local variables: $U$, utility matrix, initially identical to $R$
                     $U'$, utility matrix, initially identical to$R$
    repeat
        $U \leftarrow U'$
        for each state i do
            $U'(s_i) \leftarrow R(s_i) + \max_a \sum_j P_{ij}^a U(s_j)$
        end
    until $\max_{s_i} |U(s_i) - U'(s_i)| < \epsilon$
return $U$
```

## 3 Policy Iteration Algorithm

It was observed that the policy often becomes exactly optimal long before the utility estimates have converged to their correct values. This observation suggests another way of finding optimal policies, called policy iteration (PI). PI picks an initial policy, usually just by taking rewards on states as their utilities and computing a policy according to the maximum expected utility principle. Then it iteratively performs two steps: *value determination*, which calculates the utility of each state given the current policy, and *policy improvement*, which updates the current policy if any improvement is possible. The algorithm terminates when the policy stabilizes.
Here is a schematic description of the PI algorithm:

```
function  $POLICY-ITERATION(P,R)$ returns a policy
    inputs:      $P$, a transition-probability matrix
                 $R$, a reward matrix
    local variables: $U$, utility matrix, initially identical to $R$
                     $\pi$, a policy, initially optimal with respect to $U$
    repeat
        $U \leftarrow$ VALUE-DETERMINATION((P,U,R,$\pi$)
                     $changed \leftarrow false$
        for each state i do
            if $\max_a \sum_j P^a_{ij} U(s_j) > \sum_j P^{\pi(s_j)}_{ij} U(s_j)$ then
                 $\pi(s_i) \leftarrow arg \max_a \sum_j P^a_{ij} U(s_j)$
                 $changed \leftarrow true$
        end
    until $changed$ = false
return $U$
```

The most tricky part of PI is Value Determination. In [Howard, 1960] this step is done by solving a system of linear equations

$$U(s_i) = R(s_i) + \sum_j P^{\pi(s_i)}_{ij} U(s_j)$$

where the reward R and the transitional probabilities matrix $P^{\pi(s_i)}$ are given, and where $U(s_i)$ are unknown. This is often the most efficient approach for small state spaces. However, for larger state spaces solving a system of linear equations is not efficient.
Another way of implementing Value Determination is by iteratively calculating the utilities for the given policy as

$$U'(s_i) \leftarrow R[i] + \sum_j P_{ij}^{\pi(s_i)} U(s_j)$$

This implementation of Value Determination is similar to VI with the only difference that the utilities are computed for a *fixed policy* instead of computing the maximum of the utilities for all possible actions in each state. This version of PI was introduced by Puterman and Shin [Puterman & Shin, 1978] and is called modified policy iteration (MPI). It leaves open the number of approximation steps to be done in Value Determination. Puterman [Puterman, 1994] proposed the following options for this decision:

- fixed number of steps for all iterations, i.e. Value Determination always performs only $n$ approximation step where $n$ is fixed;
- choosing the number according to some prespecified pattern (for example, the number increases with each iteration of PI);
- selecting this number adaptively, for example, approximation steps are repeated until the utility values stabilize with a given precision as in VI.

The last option is the one mentioned in [Russel & Norvig, 1995] as an alternative to the original PI, although there was no reference to MPI in [Russel & Norvig, 1995]. We implemented that version of PI calling it MPI and compared it to VI. Subsequently, we compared both algorithm to MPI with fixed number of approximation steps in Value Determination. We use notions of *threshold* and *precision* both for VI and MPI.

## 4 Convergence

The motivation for our experiments was to compare VI against different versions of PI on the motion planning domain, and check if the policy iteration approach is indeed more efficient than VI, as it is often assumed in the literature. For example, Puterman [Puterman, 1994] gives an analysis of convergence of MPI for *discounted* MDPs which lets him say: " Therefore in practice, value iteration should never be used. Implementation of modified policy iteration requires little additional programming effort yet attains superior convergence".

In [Russel & Norvig, 1995] PI is also described as a preferred algorithm. However, we need to notice that the model used in [Russel & Norvig, 1995] is not discounted, and without making additional assumption VI and PI (MPI) are not guaranteed to converge. That is why our implementation of MPI was getting into an infinite loop on some instances. This happens when the system of linear equations in Value Determination does not have a solution. For example, for some policy $\pi$ the corresponding Markov chain might have an absorbing state, i.e. a state $i$ such that $P_{ii} = 1$. It is easy to see that $det(I - P^{\pi}) = 0$, where $I$ is identity matrix, therefore, the system of linear equations for $\pi$ cannot be solved, and the iterative version of Value Determination will not converge. Absorbing states should be treated differently. Although the original formulation of the grid problem in [Russel & Norvig, 1995] has absorbing states (terminal states), no specific treatment for them is described. [1]

For the discounted MDP model with the discount factor $0 < \lambda < 1$ VI, PI and MPI are guaranteed to converge [Puterman, 1994].

Our experiments showed that the performance of MPI depends on the way Value Determination is implemented. A small fixed number of successive approximation steps yields a much better performance than running Value Determination until the utility values converge with a given precision, especially when the precision is high (i.e. the threshold $\epsilon$ is small).

[1] Of course, we did not follow blindly the description of VI and PI given in [Russel & Norvig, 1995], and did not calculate the values of two terminal states according to the general formula given there. The authors of [Russel & Norvig, 1995] did probably same thing when experimented with the algorithms, because we got same results on the example given in the book. Nevertheless, formally speaking VI and PI presented in [Russel & Norvig, 1995] are incorrect in case of the given grid example.

## 5 Experimental Results

In our experiments we varied the size of domain (the number of states), the configuration (i.e. location of obstacles and goal states), and the threshold value. Some of the results are shown in tables 1, 2 and 3. We compared here VI and the precision-based version of MPI. The following input data were used: two goal states, one with reward 1, the other with reward -1 (penalty), and reward -0.01 for all non-goal states; probability $p = 0.7$ of successfully performed action.
In the tables below HD stands for the Hamming distance between the given policy and the optimal policy, e.g. the number of states on which the given policy differs from the policy obtained by running the algorithm with the same initial data and maximum possible precision. Each row in the tables below corresponds to one problem instance, only precision was varied.
Table 1 shows that VI finds an optimal policy only when the threshold is 0.00006 or smaller. The thresholds in Tables 2 and 3 when VI reaches an optimal policy are 0.000244 and 0.007812 respectively.
As to MPI we can see that its execution time increases dramatically when the threshold changes from 0.015625 to 0.007812. Interesting to note is that this "transition point" is the same for different environment size as long as other parameters are fixed (see figure 2).
Making a conclusion we point out that the execution time for both algorithms is sensitive to the threshold $\epsilon$. It is difficult to choose a threshold small enough to get an optimal policy before running experiments with MPI (for VI it can be computed in advance [Williams & Baird]). Also, the complexity of MPI might increase dramatically depending on the threshold value. Therefore, VI with zero threshold is preferable, because it is an order of magnitude faster than MPI with the same threshold, same order of magnitude as MPI with significantly larger thresholds ( for which MPI is not guaranteed to find an optimal policy), and is guaranteed to converge to an optimal policy.
However, VI still performs unnecessary iterations when an optimal policy is actually found but the utility values have not stabilized yet. We present here an algorithm called *combined value-policy iteration (CVPI)* which improves VI by introducing an alternative stopping criterion. At every step we not only calculate the utilities of all states, but also determine the policy using

Table 1: Size of the world 40X40

| Threshold | VI | | MPI | |
|---|---|---|---|---|
| | Time | HD | Time | HD |
| 1.000000 | 0.18 | 1339 | 7.23 | 2 |
| 0.500000 | 0.29 | 1210 | 6.62 | 2 |
| 0.250000 | 0.60 | 783 | 5.38 | 2 |
| 0.125000 | 1.03 | 469 | 6.91 | 0 |
| 0.062500 | 1.81 | 64 | 7.51 | 0 |
| 0.031250 | 2.33 | 24 | 8.46 | 0 |
| 0.015625 | 2.73 | 15 | 9.05 | 2 |
| 0.007812 | 3.02 | 18 | 89.29 | 2 |
| 0.003906 | 3.26 | 16 | 89.27 | 2 |
| 0.001953 | 3.45 | 22 | 89.43 | 2 |
| 0.000977 | 3.72 | 15 | 89.59 | 2 |
| 0.000488 | 4.15 | 13 | 90.20 | 2 |
| 0.000244 | 4.69 | 7 | 90.47 | 2 |
| 0.000122 | 5.13 | 1 | 91.21 | 2 |
| 0.000061 | 5.48 | 0 | 91.88 | 2 |
| 0.000031 | 5.96 | 0 | 92.39 | 2 |
| 0.000015 | 6.43 | 0 | 93.82 | 0 |
| 0.000008 | 6.95 | 0 | 94.46 | 2 |
| 0.000000 | 10.09 | | 103.14 | |

the utilities calculated at this step. Then we compare the policy found on the current step with the policy found on the previous step. If the policy remains the same it does not mean yet that this policy is optimal. In order to check if it is indeed the optimal policy, we use Value Determination as described above with zero threshold in order to calculate exact utility values for that fixed policy. Than we perform the policy evaluation step, i.e. check if this policy can be improved given just calculated utility values of states. If it cannot be improved, an optimal policy is found, otherwise we continue the value iteration starting from the most recently computed utility values.

Table 2: Size of the world 35X35

| Threshold | VI | | MPI | |
|---|---|---|---|---|
| | Time | HD | Time | HD |
| 1.000000 | 0.14 | 964 | 5.25 | 0 |
| 0.500000 | 0.23 | 835 | 4.55 | 0 |
| 0.250000 | 0.47 | 478 | 3.92 | 0 |
| 0.125000 | 0.79 | 277 | 4.30 | 0 |
| 0.062500 | 1.17 | 58 | 4.78 | 0 |
| 0.031250 | 1.69 | 16 | 5.28 | 0 |
| 0.015625 | 1.96 | 12 | 6.57 | 0 |
| 0.007812 | 2.10 | 7 | 54.81 | 0 |
| 0.003906 | 2.26 | 10 | 70.07 | 0 |
| 0.001953 | 2.50 | 8 | 70.42 | 0 |
| 0.000977 | 2.84 | 11 | 71.10 | 0 |
| 0.000488 | 3.14 | 7 | 71.55 | 0 |
| 0.000244 | 3.52 | 0 | 71.97 | 0 |
| 0.000122 | 3.93 | 0 | 71.23 | 0 |
| 0.000061 | 4.18 | 0 | 71.47 | 0 |
| 0.000000 | 7.73 | | 77.17 | |

## COMBINED VALUE-POLICY ITERATION

1. Perform Value Iteration + *compute policy at each step of VI*
2. **IF** no change in policy on two successive steps, fix the policy and perform one step of Policy Iteration:
    - Value Determination finding precise values for the fixed policy;
    - policy evaluation
    - **IF** no change in policy, return it as an optimal policy, **ELSE** go to 1.

Our experiments on various input problems show that CVPI algorithm

Table 3: Size of the world 20X20

| Threshold | VI | | MPI | |
|---|---|---|---|---|
| | Time | HD | Time | HD |
| 1.000000 | 0.05 | 171 | 0.76 | 1 |
| 0.500000 | 0.09 | 151 | 0.75 | 1 |
| 0.250000 | 0.15 | 103 | 0.74 | 1 |
| 0.125000 | 0.23 | 21 | 0.81 | 1 |
| 0.062500 | 0.31 | 3 | 0.89 | 1 |
| 0.031250 | 0.39 | 2 | 1.03 | 0 |
| 0.015625 | 0.47 | 2 | 1.25 | 0 |
| 0.007812 | 0.57 | 0 | 13.14 | 8 |
| 0.003906 | 0.69 | 0 | 17.23 | 9 |
| 0.001953 | 0.82 | 0 | 21.32 | 9 |
| 0.000977 | 0.92 | 0 | 22.42 | 9 |
| 0.000488 | 1.03 | 0 | 22.12 | 0 |
| 0.000000 | 2.50 | | 23.48 | |

works better then both MPI and VI (an example is shown in table 4, each row per one problem instance of the specified size).
After comparison VI, MPI and CVPI for different threshold values, we decided to vary the probability of success when fixing all other parameters. We ran VI and precision-based MPI with $\epsilon = 0$, and compared them to MPI with fixed number of steps and to CVPI. The grid size was varied from 5X5 to 50X50. The location of terminal states and an obstacle remained the same. We also varied the cost of step from -0.01 to -0.8 but did not see any significant difference in results, so that we present results just for the cost of step = -0.01.
We present the experimental results in figure 3 and, more detailed, in tables 5 and 6. MPI was tried with fixed number of steps (2,4,6) and with precision-based stopping criterion ( unlimited number of steps marked as $\infty$ in the tables). The numbers in the tables represent execution time in seconds. The pair of numbers in braces for MPI and CVPI has the following meaning: (the number of states where the final utility found by the algorithm differs

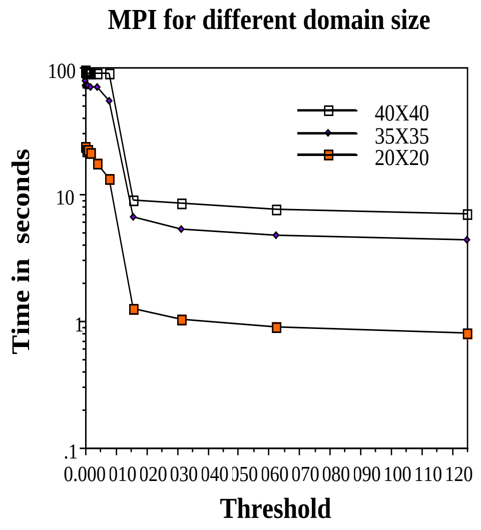

Figure 2: MPI performance as a function of precision in Value Determination

from one found by VI, the numbers of states where the policy found by the algorithm differs from one found by VI). Each point corresponds to one problem instance.

## 6 Conclusions

We have experimented with four different algorithms for computing an optimal policy in MDPs: VI, two versions of MPI, and CVPI, an algorithm proposed in this paper.
Our experiments show that CVPI is slightly more efficient than VI on the grid problems we experimented with. Both algorithms significantly (1-2 orders of magnitude) outperform MPI described in [Russel & Norvig, 1995] when using maximum possible precision (zero threshold) in value determination part of MPI. Also, the performance of this version of MPI highly depends on the precision required in value determination. MPI with fixed number of steps is more efficient than VI, CVPI, and precision-based MPI, but is not guaranteed to find an optimal policy. The behavior of all four

Table 4: Computing times for algorithms running with zero-threshold

| Size of the world | VI | MPI | CVPI |
|---:|---:|---:|---:|
| 10X10 | 0.61 | 5.51 | 0.33 |
| 20X20 | 2.50 | 23.48 | 1.33 |
| 25X25 | 3.89 | 37.35 | 1.51 |
| 30X30 | 5.66 | 55.55 | 3.74 |
| 35X35 | 7.73 | 77.17 | 5.02 |
| 40X40 | 10.09 | 103.14 | 7.06 |
| 50X50 | 31.20 | 191.44 | 11.86 |

algorithms depends on other parameters of the problem, particularly on the probability of success of actions.

## References

[Bellman, 1957] R. Bellman, Dynamic Programming. Princeton University Press, 1957.

[Howard, 1960] R.A. Howard, Dynamic Programming and Markov Processes. MIT Press, Cambridge, Massachusetts, 1960.

[Puterman, 1994] M.L. Puterman, Markov decision processes : discrete stochastic dynamic programming. New York : John Wiley & Sons, 1994.

[Puterman & Shin, 1978] M.L. Puterman and M.C. Shin, Modified policy iteration algorithms for discounted Markov decision problems. *Management Science*, 24:1127-1137, 1978.

[Russel & Norvig, 1995] S.J. Russel and P. Norvig, Artificial intelligence : a modern approach. Englewood Cliffs, N.J. : Prentice Hall, 1995.

[Williams & Baird] R.J. Williams and L.C.III Baird, Tight Performance Bounds on Greedy Policies Based on Imperfect Value Functions, Tech-

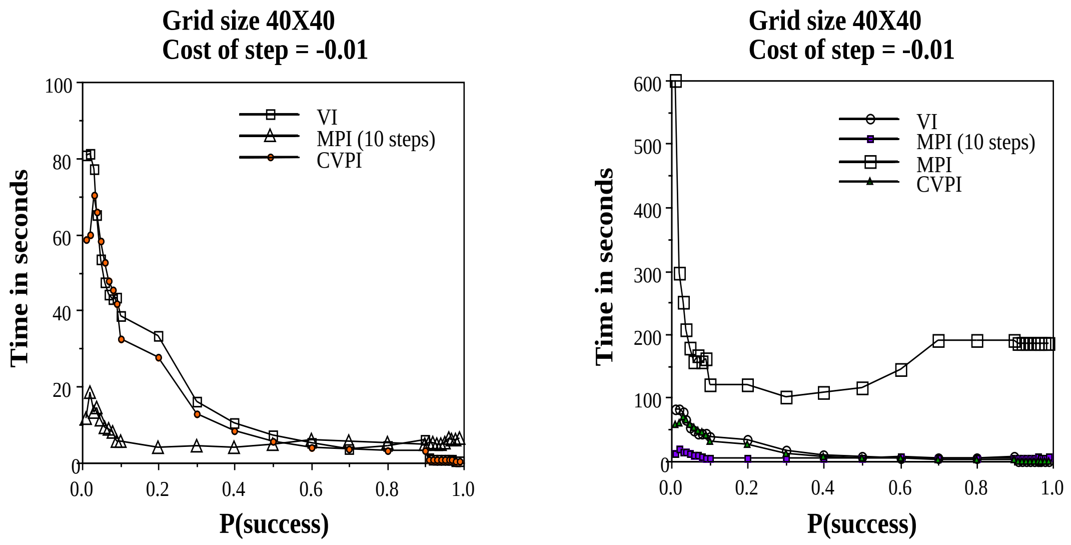

Figure 3: VI, MPI and CVPI for different probabilities of success

nicalReport NU-CCS-93-13, Northeastern University, College of Computer Science, Boston, MA, November 1993.

Table 5: VI, MPI and CVPI on grid size 30 X 30

| $\epsilon = 0.000000$ | | | | | | |
|---|---|---|---|---|---|---|
| Cost of step = -0.010000 | | | | | | |
| Pr of | VI | MPI with given # of steps in VD | | | | CVPI |
| success | | 4 | 7 | 10 | $\infty$ | |
| 0.010 | 25.73 | 2.31 (897, 19) | 2.60 (897, 14) | 3.45 (897, 7) | 132.70 (897, 1) | 15.82 (830, 1) |
| 0.020 | 25.88 | 3.90 (897, 5) | 3.52 (897, 5) | 3.32 (897, 5) | 150.68 (897, 0) | 18.53 (897, 0) |
| 0.030 | 26.61 | 3.19 (897, 86) | 5.26 (897, 4) | 5.02 (897, 4) | 89.47 (897, 0) | 20.82 (897, 0) |
| 0.040 | 27.65 | 3.12 (897, 37) | 3.93 (897, 12) | 3.85 (897, 12) | 84.12 (895, 0) | 18.78 (895, 0) |
| 0.050 | 23.27 | 3.77 (896, 20) | 3.82 (895, 10) | 3.61 (895, 10) | 71.54 (897, 1) | 21.64 (897, 1) |
| 0.060 | 20.12 | 3.78 (895, 11) | 3.31 (896, 11) | 3.18 (897, 10) | 65.51 (897, 0) | 19.44 (893, 0) |
| 0.070 | 18.55 | 3.19 (897, 16) | 3.31 (897, 9) | 3.05 (897, 9) | 59.48 (897, 1) | 18.26 (726, 1) |
| 0.080 | 17.18 | 2.96 (897, 14) | 2.59 (897, 14) | 2.40 (897, 14) | 60.86 (897, 1) | 16.22 (622, 0) |
| 0.090 | 16.93 | 2.30 (897, 16) | 2.18 (897, 13) | 2.67 (897, 4) | 55.24 (897, 0) | 16.85 (622, 0) |
| 0.100 | 16.67 | 2.22 (897, 13) | 2.40 (897, 3) | 2.53 (896, 1) | 58.85 (897, 0) | 15.28 (573, 0) |
| 0.110 | 16.84 | 1.80 (897, 8) | 1.81 (896, 4) | 1.62 (897, 2) | 46.63 (897, 0) | 14.22 (552, 0) |
| 0.210 | 8.16 | 2.00 (889, 3) | 1.87 (889, 3) | 1.86 (889, 3) | 41.81 (534, 0) | 6.93 |
| 0.310 | 5.55 | 1.64 (858, 1) | 1.66 (857, 0) | 1.88 (842, 0) | 45.02 ( 56, 0) | 4.15 |
| 0.410 | 3.67 | 1.41 (858, 0) | 1.87 (347, 0) | 2.41 (319, 0) | 44.24 | 2.91 |
| 0.510 | 2.64 | 1.49 ( 61, 1) | 1.77 ( 13, 1) | 2.11 ( 0, 1) | 52.29 ( 0, 1) | 1.99 ( 0, 1) |
| 0.610 | 2.07 | 1.32 ( 0, 3) | 1.56 ( 0, 3) | 2.01 ( 2, 3) | 72.22 ( 0, 3) | 1.83 ( 0, 3) |
| 0.710 | 2.56 | 1.57 ( 1, 0) | 1.88 | 2.42 | 103.87 | 1.72 |
| 0.810 | 3.47 | 1.20 ( 1, 0) | 1.67 ( 2, 0) | 2.07 | 103.65 | 1.77 |
| 0.910 | 0.52 | 1.26 ( 3, 0) | 1.81 | 2.25 | 101.96 | 0.47 |
| 0.920 | 0.47 | 1.35 | 1.82 | 2.20 | 102.07 | 0.44 |
| 0.930 | 0.50 | 1.25 | 1.79 | 2.14 | 101.95 | 0.41 |
| 0.940 | 0.44 | 1.25 | 1.89 | 2.15 | 101.82 | 0.39 |
| 0.950 | 0.45 | 1.33 | 1.88 | 2.23 | 101.68 | 0.41 |
| 0.960 | 0.39 | 1.31 | 1.88 | 2.15 | 101.72 | 0.33 |
| 0.970 | 0.37 | 1.77 | 2.35 | 2.50 | 101.41 | 0.31 |
| 0.980 | 0.32 | 1.89 | 2.33 | 2.44 | 101.44 | 0.27 |
| 0.990 | 0.29 | 1.99 | 2.31 | 2.52 | 101.17 | 0.23 |

Table 6: VI, MPI and CVPI on grid size 40 X 40

| $\epsilon = 0.000000$ | | | | | | |
|---|---|---|---|---|---|---|
| Cost of step = -0.010000 | | | | | | |
| Pr of | VI | MPI with given # of steps in VD | | | | CVPI |
| success | | 4 | 7 | 10 | $\infty$ | |
| 0.010 | 80.86 | 14.07 (1597, 8) | 13.61 (1597, 8) | 11.80 (1597, 8) | 599.93 (1597, 0) | 58.81 (1595, 0) |
| 0.020 | 81.19 | 14.06 (1597, 87) | 14.90 (1597, 43) | 18.61 (1597, 3) | 295.73 (1597, 0) | 59.72 (1597, 0) |
| 0.030 | 77.07 | 11.08 (1597, 90) | 11.21 (1597, 64) | 13.27 (1597, 40) | 250.23 (1597, 2) | 70.20 (1597, 2) |
| 0.040 | 64.97 | 10.65 (1332, 63) | 11.07 (1597, 46) | 14.59 (1597, 11) | 208.37 (1597, 0) | 65.89 (1597, 0) |
| 0.050 | 53.53 | 9.48 (1597, 53) | 9.03 (1597, 42) | 11.09 (1597, 12) | 178.03 (1597, 0) | 58.43 (1222, 0) |
| 0.060 | 47.77 | 9.47 (1596, 22) | 9.74 (1596, 12) | 9.21 (1597, 12) | 157.74 (1597, 0) | 52.54 (1127, 0) |
| 0.070 | 44.02 | 8.56 (1597, 26) | 7.94 (1597, 18) | 8.73 (1597, 12) | 166.18 (1597, 0) | 47.63 (958, 1) |
| 0.080 | 43.07 | 6.20 (1597, 39) | 8.48 (1596, 6) | 8.02 (1597, 6) | 155.45 (1597, 1) | 45.42 (897, 1) |
| 0.090 | 43.31 | 5.66 (1597, 33) | 6.07 (1597, 14) | 5.67 (1597, 14) | 160.64 (1597, 0) | 41.79 (813, 0) |
| 0.100 | 38.44 | 6.11 (1597, 11) | 5.74 (1596, 6) | 5.68 (1597, 5) | 120.44 (1597, 0) | 32.53 (781, 0) |
| 0.110 | 33.47 | 5.67 (1597, 10) | 5.73 (1597, 5) | 4.03 (1597, 11) | 121.41 (1597, 0) | 27.60 (781, 0) |
| 0.210 | 15.91 | 4.37 (1589, 4) | 4.24 (1589, 1) | 4.27 (1589, 1) | 100.11 (1092, 0) | 12.79 |
| 0.310 | 10.43 | 3.44 (1577, 2) | 3.33 (1577, 2) | 3.82 (1510, 0) | 108.46 (755, 0) | 8.41 |
| 0.410 | 7.11 | 2.80 (1308, 1) | 3.52 (630, 1) | 4.75 (486, 1) | 116.45 ( 80, 1) | 5.68 |
| 0.510 | 5.14 | 3.32 (147, 1) | 4.65 ( 27, 1) | 6.15 ( 2, 1) | 145.63 ( 0, 1) | 3.85 ( 0, 1) |
| 0.610 | 3.71 | 3.27 ( 0, 10) | 4.12 ( 0, 10) | 5.72 ( 0, 10) | 190.66 ( 0, 10) | 3.51 ( 0, 10) |
| 0.710 | 4.60 | 3.22 ( 0, 2) | 4.26 ( 0, 2) | 5.35 ( 0, 2) | 189.58 ( 0, 2) | 3.27 ( 0, 2) |
| 0.810 | 6.20 | 2.67 ( 1, 0) | 3.98 | 4.80 | 189.37 | 3.30 |
| 0.910 | 1.04 | 3.01 | 4.10 | 5.31 | 186.14 | 0.96 |
| 0.920 | 0.94 | 3.09 | 4.13 | 5.33 | 185.88 | 0.84 |
| 0.930 | 0.93 | 3.00 | 4.48 | 5.01 | 185.55 | 0.82 |
| 0.940 | 0.85 | 3.19 ( 6, 0) | 4.48 | 4.97 | 185.57 | 0.76 |
| 0.950 | 0.85 | 3.45 | 4.67 | 5.36 | 184.69 | 0.75 |
| 0.960 | 0.71 | 3.96 | 5.57 | 6.27 | 184.96 | 0.65 |
| 0.970 | 0.66 | 4.37 | 5.47 | 5.98 | 184.68 | 0.62 |
| 0.980 | 0.60 | 4.55 | 5.37 | 5.84 | 185.15 | 0.50 |
| 0.990 | 0.53 | 4.59 | 5.70 | 6.24 | 184.40 | 0.41 |