香港中文大學（深圳）
The Chinese University of Hong Kong, Shenzhen

DDA4300 - Optimization in Data Science and Machine Learning

Chinese University of Hong Kong, Shenzhen

Value-Iteration Method for MDP

Tian Lang (ID: 120090793)
Feng Chuqiao (ID: 120090272)
Wu Manning (ID: 120090753)
Zhou Linli (ID: 120090555)
Wan Zhenglin (ID: 121090525)

Date: April 26, 2023

1 Problem

In MDP, we consider the following minimization problem

$$\min_{\pi \in A^S} V_\pi(i) := \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t c_{a_t}(i_t) | i_0 = i \right] \tag{1}$$

where $\{i_0, a_0, i_1, a_1, \cdots, i_t, a_t, \cdots\}$ are state-action transitions generated by the MDP under the fixed policy π, i.e. $a_t = \pi_{i_t}$, and the expectation $\mathbb{E}_\pi[\cdot]$ is over the set of (i_t, a_t) trajectories.

2 Preliminaries

We describe a MDP by the tuple (S, A, P, c, γ), where S is finite state space, and the number of the total states is $|S|$; A is finite state action space; $\gamma \in [0, 1)$ is the discounted factor; P is the collection of state-action-state transition probabilities, with $P(i'|i, a)$ represents the probability of going to state i' from state i when taking action a; c is the collection of costs at different state-action pairs, i.e. we cost $c_a(i)$ if we are currently in state i and take action a.
For the *proof* of Lemma 1, Lemma 2, Lemma 3, please see A.1

Definition 1 (value operator)
For a given MDP, the value operator $T : \mathbb{R}^{|S|} \to \mathbb{R}^{|S|}$ is defined for all $U \in \mathbb{R}^{|S|}$ and $i \in S$ by

$$T(U)_i = \min_{a \in A_i} \left(c_a(i) + \gamma \sum_{i' \in S} P(i'|i, a) U(i') \right) \tag{2}$$

Definition 2 (ϵ – optimality)
We say values $u \in \mathbb{R}^S$ are $\epsilon-optimal$ if $\|v^* - u\|_\infty \leq \epsilon$ and we say a policy $\pi \in A^S$ is $\epsilon-optimal$ if $\|v^* - v_\pi\|_\infty \leq \epsilon$.

2.1 Linear systems

Noticing that the original problem(1) can be converted into a linear problem as follows:

$$\begin{aligned} \min \quad & \sum_{\forall i \in S} V(i) \\ \text{s.t.} \quad & V(i) \leq c_a(i) + \gamma \textstyle\sum_{i' \in S} P(i'|i, a) V(i') \quad \forall a \in A, i \in S \end{aligned} \tag{3}$$

and its dual problem is:

$$\begin{aligned} \max \quad & \sum_{\forall i \in S} \sum_{\forall a \in A_i} c_a(i) x_a(i) \\ \text{s.t.} \quad & \sum_{\forall a \in A_i} x_a(i) - \gamma \sum_{\forall i' \in S} \sum_{\forall a \in A_i} P(i'|i, a) x_a(i') = 1, \quad \forall i \in S \end{aligned} \tag{4}$$

Lemma 1
In (4), every basic feasible solution represents a policy. Furthermore, each basic variable value is no less than 1, and the sum of all basic variable values is $\frac{m}{1-\gamma}$.

2.2 Value Iteration Properties

Lemma 2 (Contraction Mapping)
For all values $U, V \in \mathbb{R}^{|S|}$ we have that $\|T(U) - T(V)\|_\infty \leq \gamma \|U - V\|_\infty$ and consequently $\|T(U) - V^*\|_\infty \leq \gamma \|U - V^*\|_\infty$, where V^* is the optimal value vector.

Lemma 3 (Entry-wise Monotone Property)
if values $U, V \in \mathbb{R}^{|S|}$ satisfy $U \leq V$ entry-wise, then $T(U) \leq T(V)$ entry-wise.

3 Value Iterations

When the state space $|S|$ is large, updating all state values in each iteration can be computationally expensive. To address this issue, we modified the value iteration method by introducing four new approaches: random value iteration (randomVI), influence-tree-based random value iteration (influence-tree-based randomVI), cyclic value iteration (cyclicVI), and randomly permuted value iteration (rpVI).

3.1 Random Value Iteration and Influence-tree-based Random Value Iteration

RandomVI randomly selects a subset B^k to update in kth iteration, rather than updating all state values. The value of state i in B^k is updated by

$$V^k(i) = \min_{a \in \mathcal{A}_i} \{c_a + \gamma \sum_{i'} P(i'|i,a) V^{k-1}(i')\}. \tag{5}$$

To further reduce the number of states to update, we built an "influence tree" that selects a subset of reachable states based on B^{k-1}. Influence-tree-based randomVI ensures many unimportant or irrelevant states are avoided in the next iteration.

3.2 Cyclic Value Iteration and Randomly Permuted Value Iteration

Different from the classicalVI, cyclicVI is asynchronous. In the kth iteration, the value of state i is updated by the following equation in an ascending order.

$$\tilde{V}^k(i) = \min_{a \in \mathcal{A}_i} \{c_a(i) + \gamma \sum_{i'} P(i'|i,a) \tilde{V}^k(i')\} \tag{6}$$

However, this ascending order makes cyclicVI only consider the linkage between adjacent states, while relationships between non-adjacent states may also exist. To address this limitation,we modified the order of updates to a random permutation order. Randomly permuted VI ensures all cases are taken into account.

3.3 Numerical Experiments

In order to compare all the modifications and classicalVI, we perform our numerical experiments on two kinds of maze games: 2-dimensional standard maze and a 3-dimensional terrain maze with an additional height. We tested four different maze sizes (16, 64, 100, and 200) and varied the probability of sample inclusion p in randomVI and influence-tree-based randomVI over eight values: 0.125, 0.25, 0.375, 0.5, 0.625, 0.75, 0.875, and 1.

3.3.1 Results and Conclusion

First, we compared the convergence rates among different p and found that both randomVI and influence-tree-based randomVI performed best when the $p = 1$. (Details are shown in the Appendix) Thus, we choose $p = 1$ for the comparison of convergence rates. Results are shown in Figure 1 and Figure 2.
Figures indicate that for the standard maze, classicalVI and cyclicVI exhibit the best convergence rates, while rpVI performs best in the terrain maze. It may because that the terrain maze is more like a real-life maze, where the correlation between non-adjacent states is stronger, making the rpVI approach more effective.
Furthermore, we analyzed the number of iterations required for convergence across different maze sizes and methods. (Details are shown in the Appendix) The result shows that the size of maze has little impact on the number of iterations required for convergence, except when using influence-tree-based randomVI

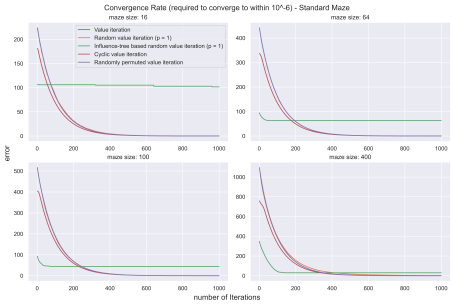


Figure 1: Standard Maze ($\gamma = 0.99$, 20000 runs)

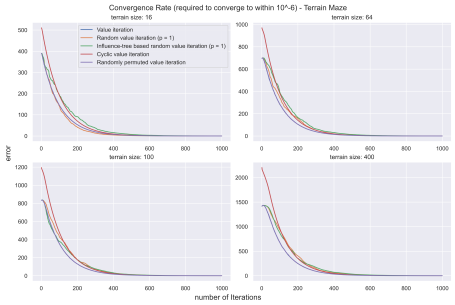


Figure 2: Terrain Maze ($\gamma = 0.99$, 20000 runs)

to solve the problem. However, it is more time-consuming when the size is larger. Additionally, classicalVI and rpVI exhibit the lowest number of iterations required for convergence, but their computational cost is high. Influence-tree-based randomVI, on the other hand, requires the highest number of iterations but has a lower computational cost.

4 Stochastic Approximation

Value iteration has been proven to be a simple and theoretically guaranteed method for solving MDPs. However, the implementation of value iteration can become computationally infeasible as the size of the state space increases. To address this issue, this project utilized state aggregation to overcome the computational cost while maintaining convergence.

4.1 State Aggregation

State aggregation involves grouping similar states to form aggregate states, dividing the original state space $\mathcal{S}$ of MDPs into K collections, and treating each collection as a mega-state. The state aggregation induces a partition on $\mathcal{S}$ that $\mathcal{S} = \cup_{i=1}^{K} S_i$, and they are mutually exclusive.

To approximate the optimal value V^*, we denote $W \in \mathbb{R}^K$ as the cost-to-go value function for the mega-state. The current value of W induces a value function $\tilde{V} \in \mathbb{R}^{|\mathcal{S}|}$ on the original state space, where

$$\tilde{V}(s, W) = W(j), \ for \ s \in S_i$$

4.2 Aggregation Algorithm

To improve the performance of pre-specified state aggregation, it is indeed to design the value function W that the approximation error $\left\|\tilde{V}(W) - V^\star\right\|_\infty$ is small. Without knowing the optimal value function $V^\star$ in advance, it can be challenging to design an accurate value function W. We utilized an adaptive state aggregation schemes to learn the true cost-to-go values online.Chen et al. (2021)

As the value iteration progresses and the value function V^k converges to the optimal value V^*, it is possible to adaptively update the mega-states and the value function W based on the current cost-to-go value vector $V^k \in \mathbb{R}^{|\mathcal{S}|}$. The current value function V^k can be discretized into intervals to form non-empty mega-states in order to reduce the original state size from $|\mathcal{S}|$ to at most $\lceil (b_2 - b_1)/\varepsilon \rceil$ mega-states, where $b_1 = \min_{s\in\mathcal{S}} V^k(s), b_2 = \max_{s\in\mathcal{S}} V^k(s)$. See Algorithm 1 for further details.

Algorithm 1 Value-based Aggregation

Input: $\varepsilon, \mathbf{V} = (V(1), \cdots, V(|\mathcal{S}|))^T$
$b_1 = \min_{s\in\mathcal{S}} V(s), b_2 = \max_{s\in\mathcal{S}} V(s), \Delta = (b_2 - b_1)/\varepsilon$
for $i = 1, \cdots, \lceil\Delta\rceil$ **do**
$\hat{S}_i = \{s | V(s) \in [b_1 + (i-1)\varepsilon, b_1 + i\varepsilon)\}, \hat{W}(i) = b_1 + (i - \frac{1}{2})\varepsilon$
end for
Delete the empty sets and the associated $\hat{W}(i)$ in $\{S_i\}_{i=1}^{\lceil\Delta\rceil}$ to form K mega-states and $W \in \mathbb{R}^{\mathrm{K}}$
Return: $\{S_i\}_{i=1}^{\mathrm{K}}$ and W

4.3 Periodical Implementation

When the optimal value function V^* is not known in advance, a two-phase algorithm can be used to update the mega-states and the value function W periodically. The two phases are:

- **Phase I (with $\mathcal{B}$):** algorithm performs global updates on $\mathcal{S}$.
- **Phase II (with $\mathcal{A}$):** algorithm performs state-aggregated updates.

At the beginning of Phase II, the algorithm performs the adaptive aggregation on the original state space $\mathcal{S}$ based on the last update of the value function V from Phase I. In the state-aggregated update, the algorithm takes a form similar to Stochastic Approximation by uniformly sample state s from each mega-state S_i, and

$$W^{k+1}(i) = (1 - \alpha_k)W^k(i) + \alpha_k T(\tilde{V}(W^k))_i$$

The algorithm divides the time horizon into intervals of $\{\mathcal{B}_i\}_{i=1}^{\infty}, \{\mathcal{A}_i\}_{i=1}^{\infty}$ with a pre-specified number of iterations n. These intervals have the form $\mathcal{B}_1, \mathcal{A}_1, \mathcal{B}_2, \mathcal{A}_2, \cdots$, and the algorithm performs value iteration on the state space $\mathcal{S}$ and state-aggregated updates alternatively according to the current adaptive aggregation. At the beginning of $\mathcal{A}_i$, the algorithm performs Algorithm 1 on $\mathcal{S}$ based on the last update on the value function V. See Algorithm 2 for further details.

4.4 Convergence

Theorem 1. If $\limsup \alpha_t \to 0$, $\limsup_{t\to\infty} |\mathcal{A}_i| < \infty$ and $\liminf_{t\to\infty} |\mathcal{B}_i| > 0$, we have

$$\limsup_{t\to\infty} \left\| V^t - V^* \right\|_\infty \leq \frac{2\varepsilon}{1-\gamma}$$

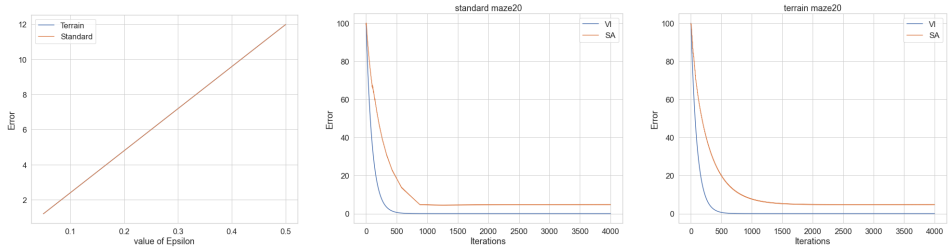
If the parameter ε in state aggregation is fixed, the approximation of the value function will have a ℓ^∞ error bounded by $\frac{2\varepsilon}{1-\gamma}$. This theorem states the asymptotic convergence for Algorithm 2. It is worth noting that Phase II has the potential to increase the error. Therefore, it may not be desirable to have $\limsup |\mathcal{A}_t| \to \infty$.

4.5 Experiments

To test the theory presented in Section 4.3, we conducted a series of experiments. We compared Algorithm 2 to the standard value iteration method on MDPs of different scales and complexities, as discussed in Section 3.3. Our results show that state aggregation has a significant reduction in computational cost while maintaining similar convergence performance when compared to standard value iteration.

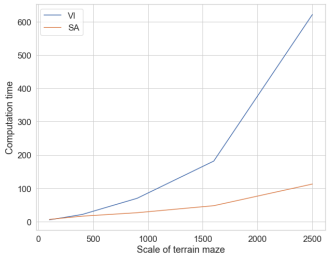
All experiments were performed with a discount factor of $\gamma = 0.99$. We fixed $|\mathcal{A}_i| = 2$ and $|\mathcal{B}_i| = 5$, with a learning rate $\alpha_t = \frac{1}{\sqrt{t}}$. The aggregation constant ε was set to 0.5. The error per is defined as the ℓ^∞ norm of the difference between the current value function V^t and the optimal value function V^*.

4.6 Results



(a) Influence of ε (b) Convergence on standard (c) Convergence on terrain

Figure 3: Left: The error of state-aggregated value iteration with different ε. **Middle:** Convergence speed on 20×20 standard maze. **Right:** Convergence speed on 20×20 terrain maze



Scale	Time (s) of SA	Time (s) of VI
100	6.59	5.22
400	16.35	21.51
900	26.69	69.91
1600	47.68	181.70
2500	112.72	621.45

Figure 4: Efficiency test

Influence of ε: We conducted experiments on a 20×20 maze with different settings of ε to investigate the effect of ε on the approximation error. The results, shown in Figure 3(a), indicate that the approximation error is proportional to ε.

Convergence: We tested the convergence of Algorithm 1 against standard value iteration on 20×20 standard and terrain mazes. The results, shown in Figure 3(b) and 3(c), suggest that the state-aggregated update $\mathcal{B}$ may increase the error.

Efficiency: We compared the computational time in 4000 runs on a large-scale terrain maze (50×50), repeated 20 times. The results are summarized in Table. As shown in Figure 4, state aggregation resulted in a significant reduction in computational time.

5 Tic-Tac-Toe Game

5.1 Value-Iteration Solution

In this problem, we are supposed to develop the optimal strategy for the cross-player. We assume that the cross-player plays first, and the opponent is a random player. To formulate the game as an MDP problem and find the optimal policy. We denote one complete game as one iteration k and keep the trajectory of all states from the start till the end of the game. Also, we keep the X player's and O player's values on each value separately. As for choosing an action, we let the agent choose the state with the highest value in its available moving space.

To update the value function of state s, we use the same value iteration method as what we did in the previous questions:

$$V_{k+1}(s) = \max_a(R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a)V_k(s')) \quad \text{where } P(s'|s,a) = \frac{1}{\text{num of current empty blanket}}$$

As for $R(s,a)$, we defined it as the reward given by the result of the game, if the player agent wins the game, the return would be 1, get 0 if it lost, and get 0.5 for other conditions. According to how we define the policy choosing above, we can get the policy function at the state s is:

$$\pi(s) = \arg\max_a R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a)V_{k+1}(s')$$

We extracted all the possible states and initialize their values, then run value iteration over all the states until convergence (The algorithm and pseudo-code can be referred to in the Appendix). Also, we set an indicator variable $\delta = max\{\delta, |(V - V_{new})|\}$ to judge whether the value iteration converges to an optimal global policy. After 5 processes, we can find it converges very quickly in figure 3. However, we find the time and space complexity largely increased in the same process. So we came up with another method with Q-learning.

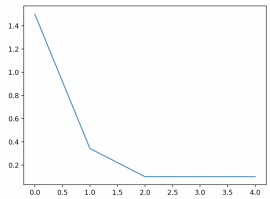


Figure 5: Convergence of value iteration

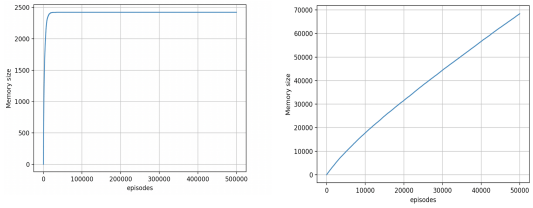


Figure 6: Comparison of size 3 and 4

5.2 Q-learning solution

Value iteration requires the state-to-state transition model given the action to learn the value function at every state. But by Q-learning, where q-values are simply the state-action value function. By state-action value function to take all the state-action pairs, for all s in S and a in A, and build a new MDP with transitions between pairs. This means for the q-value iteration part:

$$Q(s) = R(s) + \gamma \max_a \sum_{s'} T(s,a,s')Q(s') \qquad \delta \leftarrow \max_a[\delta, |Q_{old} - Q(s,a)|] \text{ until } \delta < \epsilon$$

Here to prove this right, we introduced a lemma that:
Lemma 4. Let Q, Q' be two Q-functions such that for all (s,a) we have $Q(s,a)$, $Q(s',a) \geq 0$ and let V, V' be the corresponding value functions such that for any $s \in S$, $V(s) = max_{a \in A} Q(s,a)$ and $V'(s) = max_{a \in A} Q'(s,a)$. Then

$$\|V - V'\|_\infty \leq \|Q - Q'\|_\infty$$

Then to get the policy for an agent to play with a random player, for each $s \in S$ do the:

$$\pi(s) \leftarrow \arg\max_{a \in A} Q(s,a)$$

After the experiment, we find the computation time increase largely in a larger size game, and it solves the problem we met in the previous part.

5.3 Deep-learning solution

Furthermore, if the game finally increased into a complex environment with high-dimensional state spaces, the process above will be difficult to explicitly represent this mapping. Therefore we think about using function approximation to learn a mapping from states to actions that maximize the expected reward. Where we complete the process like:

$$\text{simulation} \rightarrow \text{pass q-value to NN} \rightarrow \text{decision making}$$

After several trials, we found the result is not that satisfactory (the network we used and results are in Appendix). It can not work faster or more accurately than value iteration and q-learning very much in this tic-tac-toe game. Because of the limitation of this game setting, such as the total space is not large enough for deep learning, Neural Network will reduce the explainability for this model and so on. But we believe it can have great performance on many other more complicated conditions.

6 Conclusion

We discussed the value-iteration problem and its variance in different conditions. We found VI with Adaptive Aggregation shows improved efficiency in terms of computation time than basic VI. Randomly Permuted Value Iteration, potentially leads to further improvement in convergence speed. Because it explores the development of heuristic algorithms that can predict permutation. As for tic-tac-toe, we find Q-learning is better than Value Iteration in time and space complexity, while DRL only performs well in a more complicated problem. However, it should be noted that state aggregation may only show efficiency in large-scale problems. The state-aggregated update, $\mathcal{B}$, can introduce considerable variance, leading to an increment in approximation error in stochastic approximation. Furthermore, the mechanism of periodic implementation in alternating phases of $\mathcal{B}$ and $\mathcal{A}$ is still unclear and may require further investigation. In future research, we can explore ways to address these issues and improve the overall performance of the algorithm.

7 Acknowledgement

Thanks to the instructions from Ziniu Li.

References

Chen, G., Gaebler, J. D., Peng, M., Sun, C., and Ye, Y. (2021). An adaptive state aggregation algorithm for markov decision processes. pages 3

A Proof of preliminaries

A.1 Proof of Lemma 1

Firstly, we aim to simplify our linear program to standard form. Let

$$
\begin{aligned}
n &:= \sum_{i=1}^{m} |A_i| \\
c &:= \begin{bmatrix} c_{a_1}(1) \\ c_{a_2}(1) \\ \vdots \\ c_{a_1}(2) \\ \vdots \\ c_{a|A_m|}(m) \end{bmatrix} \in \mathbb{R}^n \\
p &:= \begin{bmatrix} p_1^T \\ p_2^T \\ \vdots \\ p_n^T \end{bmatrix} \equiv \begin{bmatrix} P(1|1,a_1) & P(2|1,a_1) & \cdots & P(m|1,a_1) \\ P(1|1,a_2) & P(2|1,a_2) & \cdots & P(m|1,a_2) \\ \vdots & \vdots & \vdots & \vdots \\ P(1|m,a_n) & P(2|m,a_n) & \cdots & P(m|1,a_n) \end{bmatrix} \in \mathbb{R}^{n\times m} \\
E &:= \begin{bmatrix} e_1^T \\ e_2^T \\ \vdots \\ e_n^T \end{bmatrix} \in \mathbb{R}^{n\times m}
\end{aligned}
$$

Thus our linear program can be compactly formulated as follows:

$$
\begin{array}{ll} \max & c^T x \\ \text{s.t.} & (E-\gamma P)^T x = \mathbf{e} \\ & x \geq 0 \end{array} \tag{7}
$$

Suppose basic columns of $(E-\gamma p)^T$ is $j(1) < \cdots < j(m)$, we want to show that $j(i) \in A_i$.
Note that our BFS (basic feasible solution) is $\sum_{i=1}^{m} x_{j(i)} e_{j(i)}$ for $x_{j(i)} > 0$. The first constrain of this program tells us that:

$$\sum_{i=1}^{m} x_{j(i)} (E-\gamma p)^T e_{j(i)} = \mathbf{e}$$

Expand it, then we can get:

$$
\left[\sum_{1\leq k\leq m : j(k)\in A_i} x_j(k) \right] - \gamma \left[\sum_{k=1}^{m} x_{j(k)} p_{j(k),i} \right] = 1 \quad \forall i = 1, 2, \cdots, m \tag{8}
$$

where $p_{j,i}$ represents the j,i-th entry of matrix [. Note that $[\sum_{k=1}^{m} x_{j(k)} p_{j(k),i}]$ is greater than 0, so for every i, at least one k satisfy $j(k) \in A_i$. Since there are exactly m basis vectors, so there is exactly one k satisfying $j(k) \in A_i$. Since $j(1) < j(2) < \ldots < j(m)$, we can conclude that $j(i) \in A_i$. So we can change formula (7) into :

$$1 = x_{j(i)} - \gamma \sum_{k=1}^{m} x_{j(k) p_{j(k),i} \leq x_{j(i)}} \tag{9}$$

So each basic variable value is no less than 1.
Now we sum up all the basic variable values by summing the equality part of (2) for i=1,2 ...,m. We have:

$$\begin{aligned} m &= \sum_{i=1}^{m}[x_{j(i)} - \gamma \sum_{k=1}^{m} x_{j(k)} p_{j(k),i}] \\ &= \sum_{i=1}^{m} x_{j(i)} - \gamma \sum_{k=1}^{m} x_{j(k)} \\ &= (1-\gamma) \sum_{i=1}^{m} x_{j(i)} \end{aligned}$$

A.2 Proof of Lemma 2

First of all, let's fix one specific $1 \le i \le m$. There are two cases.
case 1: $y_i^{k+1} \ge y_i^*$,so:

$$\begin{aligned} |y_i^{k+1} - y_i^*| &= y_i^{k+1} - y_i^* \\ &= (\min_{j \in A_i}(c_j + \gamma p_j^T y^k)) - (\min_{j \in A_i}(c_j + \gamma p_j^T y^*)) \\ &= (c_{j_0} + \gamma p_{j_0}^T y^k) - (c_{j_1} + \gamma p_{j_1}^T y^*) \qquad (10) \\ &\quad \text{(suppose } j_0 \text{ and } j_1 \text{ are the minimizing j for minuend and subtrahend)} \end{aligned}$$

And we also have:

$$c_{j_1} + \gamma p_{j_1}^T y^k \ge c_{j_0} + \gamma p_{j_0}^T y^k$$

Then we can do something to equation (1):

$$\begin{aligned} |y_i^{k+1} - y_i*| &= (c_{j_0} + \gamma p_{j_0}^T y^k) - (c_{j_1} + \gamma p_{j_1}^T y^*) \\ &\le (c_{j_1} + \gamma p_{j_1}^T y^k) - (c_{j_1} + \gamma p_{j_1}^T y^*) \\ &= \gamma p_{j_1}^T (y^k - y^*) \\ &\le \gamma \|p_{j_1}\|_1 * \|y^k - y^*\|_\infty \qquad \text{(by Holder's inequality)} \\ &= \gamma \|y^k - y^*\|_\infty \end{aligned}$$

case 2: $y_i^{k+1} \le y_i^*$, it is analogous. Then all entry of $\|y^{k+1} - y^*\|$ is less or equal to $\gamma \|y^k - y^*\|_\infty$, so $\|y^{k+1} - y^*\|_\infty \le \gamma \|y^k - y^*\|_\infty$.

A.3 Proof of Lemma 3

The same as proof of (2), suppose j_0 and j_1 be the minimizing j. Then we analyze the case that$y^0 \ge y^*$, and the opposite case is analogous. We use induction and assume $y^k \ge y^*$. Then we note that,for all i:

$$y_i^{k+1} - y_i^* \ge \gamma p_{j_0}^T (y^k - y^*) \ge 0$$

So $y^{k+1} \ge y^*$.If we suppose j_2 satisfy $y_i^k = c_{j_2} + \gamma p_{j_2}^T y^{k-1}$, then we have:

$$y_i^{k+1} = \min_{j \in A_i} c_j + \gamma p_j^T y^k \le c_{j_2} + \gamma p_{j_2}^T y^k = \gamma p_{j_2}^T (y^k - y^{k-1}) + y_i^k$$

Since by hypothesis $y^k - y^{k-1} \le 0$, so for all i we have $y_i^{k+1} \le y_i^k$, so $y^{k+1} \le y^k$.

A.4 Proof of Theorem.1

If we first switch to state-aggregated phase and denote $W(V^t)$ as the result of Algorithm 1, we will have

$$\left\|\tilde{V}(W(V^t)) - V^*\right\|_\infty \leq \left\|V^t - V^*\right\|_\infty + \frac{\varepsilon}{2}$$

For each step of stochastic approximation, as $\left\|v^t\right\|_\infty \leq \frac{1}{1-\gamma}$, we have

$$|W^t(j) - W^{t-1}(j)| \leq \alpha_{t_{sa}}(\left\|T(\tilde{V}(W^{t-1}))_s\right\|_\infty + \left\|W^{t-1}\right\|_\infty) \leq \alpha_{t_{sa}}\frac{2}{1-\gamma}$$

Condition: There exists $M > 0$ such that for any $\mathcal{A}_i$ coming after M state aggregation iterations, we have

$$\left\|W^{t_0+|\mathcal{A}_i|}\right\| - W^{t_0} \leq \sum_{i=0}^{|\mathcal{A}_i|-1} \alpha_{t_{sa}}(\left\|T(\tilde{V}(W^{t_0+i}))_s\right\|_\infty + \left\|W^{t_0+i}\right\|_\infty) \leq \frac{\varepsilon}{2}$$

Since

$$\limsup_{t\to\infty} |\mathcal{A}_i| < \infty, \liminf_{t\to\infty} |\mathcal{B}_i| > 0$$

There exists N such that when $i > N$, $a = \sup|\mathcal{A}_i|, b = \inf|\mathcal{B}_i|$. Then since $\{\alpha_t\}_{t=1}^\infty \to 0$ as $t \to \infty$, there exists a constant T that for $t > T$ we have $a \cdot \alpha_t < \frac{\varepsilon}{2(1-\gamma)}$. We have

$$\left\|W^{t_0+|\mathcal{A}_i|}\right\| - W^{t_0} \leq \sum_{i=0}^{a} \alpha_{t_{sa}}(\left\|T(\tilde{V}(W^{t_0+i}))_s\right\|_\infty + \left\|W^{t_0+i}\right\|_\infty) \leq \frac{\varepsilon}{2}$$

Condition 1 has been satisfied. Since $\left\|V^{t_i} - V^*\right\|_\infty \leq \gamma^{i-1}\left\|V^{t_1} - V^*\right\| + \frac{\varepsilon}{1-\gamma}$ and,

$$\limsup_{t\to\infty} \left\|V^{t_i} - V^*\right\|_\infty \leq \frac{\varepsilon}{1-\gamma}$$

Then, we aim to show that for any positive constant $\lambda > 0$, there exists a constant T_λ so that for any $t > T_\lambda$, thus we can find $i_\lambda > 0$ for any $\geq i_\lambda$ that

$$\left\|V^{t_i} - V^*\right\|_\infty \leq \frac{\varepsilon}{1-\gamma} + \lambda$$

Defining $T_\lambda = T + t_{i_\lambda}$, for $t \in [T_\lambda, +\infty) \cap \mathcal{A}_i$ and $t \in [T_\lambda, +\infty) \cap \mathcal{B}_i$, for some $i \in \mathbb{N}$, from the inequality

$$\left\|V^{t_i} - V^*\right\|_\infty \leq \gamma\left\|V^{\tau_i} - V^*\right\|_\infty, \quad \left\|V^t - V^*\right\|_\infty \left\|V^{t_i} - V^*\right\|_\infty + \varepsilon, \quad t \in (t_i, \tau_{i+1} - 1]$$

We have $\left\|V^t - V^*\right\|_\infty \leq \varepsilon$. Thus,

$$\begin{aligned}\left\|V^t - V^*\right\|_\infty &\leq \left\|V^t - V^{t_i}\right\|_\infty + \left\|V^{t_i} - V^*\right\|_\infty \\ &\leq \varepsilon + \frac{\varepsilon}{1-\gamma} + \lambda \\ &\leq \frac{2\varepsilon}{1-\gamma} + \lambda\end{aligned}$$

If $t \in [T_\lambda, +\infty) \cap \mathcal{B}_i$ for some $i \in \mathbb{N}$, based on the contraction property of Value Iteration, we have $\left\|V^t - V^*\right\|_\infty \leq \gamma^{t-\tau_i}\|V^{\tau_i} - V^*\|_\infty$ Hence, we can show that for all $t > T_\lambda$ and any $\lambda > 0$,

$$\limsup_{t\to\infty} \left\|V^t - V^*\right\|_\infty \leq \frac{2\varepsilon}{1-\gamma}$$

A.5 Proof of Lemma 4 (Q-Learning)

Lemma 4. Let Q, Q' be two Q-functions such that for all (s,a) we have $Q(s,a)$, $Q(s',a) \geq 0$ and let V, V' be the corresponding value functions such that for any $s \in S$, $V(s) = max_{a \in A} Q(s,a)$ and $V'(s) = max_{a \in A} Q'(s,a)$. Then

$$\|V - V'\|_\infty \leq \|Q - Q'\|_\infty$$

Proof: $Q(s,a) = Q(s,a) - Q'(s,a) + Q'(s,a) \Rightarrow Q(s,a) \leq |Q(s,a) - Q'(s,a)| + Q'(s,a)$
where we have used the fact that Q, Q' are always non-negative by definition of reward. So:

$$max_a Q(s,a) \leq max_a(|Q(s,a) - Q'(s,a)| + Q'(s,a)) \leq max_a(|Q(s,a) - Q'(s,a)|) + max_a Q'(s,a)$$

Thus we obtain the identity

$$max_a Q(s,a) - max_a Q'(s,a) \leq max_a(|Q(s,a) - Q'(s,a)|)$$

Swap the order and we have

$$\begin{aligned} |max_a Q(s,a) - max_a Q'(s,a)| &\leq max_a(|Q(s,a) - Q'(s,a)|) \\ \|V - V'\|_\infty &= max_s|V(s) - V'(s)| \\ &= max_s|max_a Q(s,a) - max_a Q'(s,a)| \\ &\leq max_s max_a|Q(s,a) - Q'(s,a)| \\ &= \|Q - Q'\|_\infty \end{aligned}$$

B Details of results

B.1 Example of Results for Maze Game with RandomVI and Influence-tree-based RandomVI among Different Sample Inclusion p

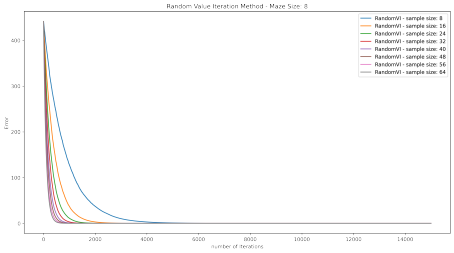


Figure 7: RandomVI-Standard Maze (size = 64, $\gamma = 0.99$, 20000 runs)

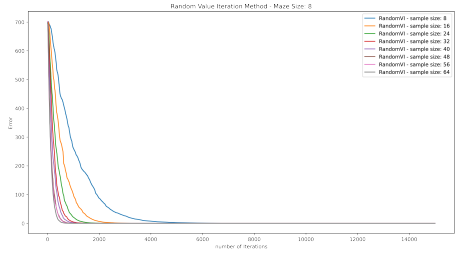


Figure 8: RandomVI-Terrain Maze (size = 64, $\gamma = 0.99$, 20000 runs)

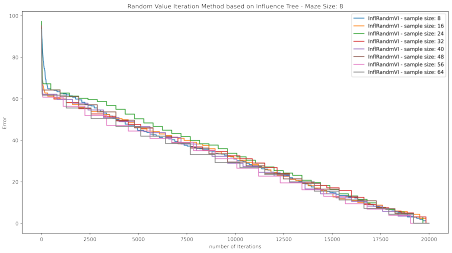


Figure 9: Influence-tree-based RandomVI-Standard Maze (size = 64, $\gamma = 0.99$, 20000 runs)

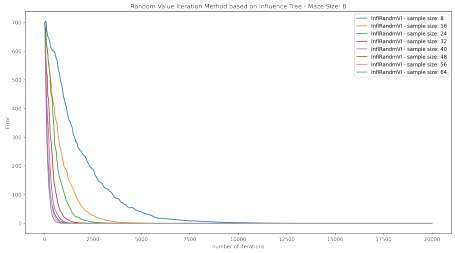


Figure 10: Influence-tree-based RandomVI-Terrain Maze (size = 64, $\gamma = 0.99$, 20000 runs)

B.2 Results for Number of Iterations to Converge with Different Methods in Maze Game

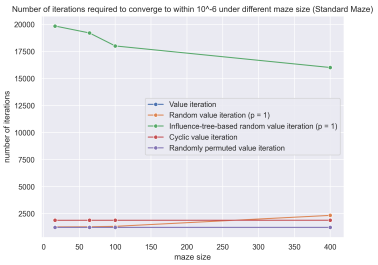


Figure 11: Standard Maze ($\gamma = 0.99$, 20000 runs)

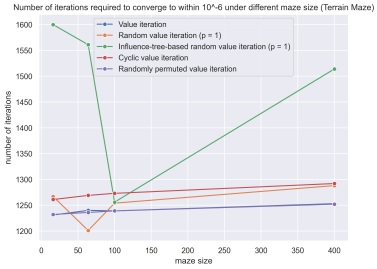


Figure 12: Terrain Maze ($\gamma = 0.99$, 20000 runs)

B.3 Value Iteration with Adaptive Aggregation

Algorithm 2 Value Iteration with Adaptive Aggregation

Input: $P, c, \varepsilon, \gamma, \{\alpha_t\}_{t=1}^{\infty}, \{\mathcal{B}_i\}_{i=1}^{\infty}, \{\mathcal{A}_i\}_{i=1}^{\infty}$
Initialize $W^0 = 0, V^1 = 0, t_{sa} = 1$
for $i = 1, \cdots, n$ **do**
  if $t \in \mathcal{B}_i$ **then**
    if t is the beginning of $\mathcal{B}_i$ **then**
      $V^{t-1} = \tilde{V}(W^{t-1})$
    end if
    for $j = 1, \cdots, |\mathcal{S}|$ **do**
      $V^t(j) = T(V^{t-1})_j$
    end for
  else
    if t is the beginning of $\mathcal{A}_i$ **then**
      Define $\{S_i\}_{i=1}^{\mathrm{K}}$ and W^t by Algorithm 1
    end if
    for $j = 1, \cdots, \mathrm{K}$ **do**
      Sample sate s uniformly from set S_j

$$W^t(i) = (1 - \alpha_{t_{sa}}) W^{t-1}(j) + \alpha_{t_{sa}} T(\tilde{V}(W^{t-1}))_j$$

    end for$t_{sa} = t_{sa} + 1$
  end if
end for
if $n \in \mathcal{B}_i$ **then**
  Return: V^n
end if
Return: $\tilde{V}(W^n)$

B.4 Details for Tic-Tac-Toe with Value Iteration (Pseudo-code)

```
For i in range(max_iterate=5):
        delta = 0
        for state in states:
                done != 0, state_index, player, legal_moves
                legal_values = []
                for action in legal_moves:
                        possible_indices(states, state, action) ---> list
                        p = 1/len(possible_indices)
                        action_value = 0
                        for i in possible_indices:
                                if player = 1: value = x_values[i]
                                if player = 0: value = o_values[i]
                                action_value += p*value
                        legal_values.append(action_value)
                if player:
        delta = max(delta, abs(x_values[state_index]-max(legal_values)))
        x_values[index] = gamma * max(legal_values)
```

```
18      else:
19          delta = max(delta, abs(o_values[state_index]-max(legal_values)))
20          o_values[index] = gamma * max(legal_values)
```

B.5 Details for Tic-Tac-Toe with Q-learning (Algorithm)

Algorithm 3 Q-Learning Algorithm

Input: $\alpha : learning\ rate, \gamma : discount\ factor, \epsilon : a\ small\ number$
Result:A Q-table containing $Q(S,A)$ pair defining estimated optimal policy $\pi^{\star}$
Initialize $Q(S,A)$ arbitrarily, except $Q(terminal,.)$;
$Q(terminal,.) \leftarrow 0$
for *each episode* **do**
  Initialize state S;
  for *each step in episode* **do**
    $A \leftarrow SELECT-ACTION(Q,S,\epsilon)$;
    Take action A, then observe reward R and next state S';
    $Q(s) = R(s) + \gamma \max_a \sum_{s'} T(s,a,s')Q(s')$

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 200)               2000

dropout (Dropout)            (None, 200)               0

dense_1 (Dense)              (None, 125)               25125

dense_2 (Dense)              (None, 75)                9450

dropout_1 (Dropout)          (None, 75)                0

dense_3 (Dense)              (None, 25)                1900

dense_4 (Dense)              (None, 3)                 78

=================================================================
```

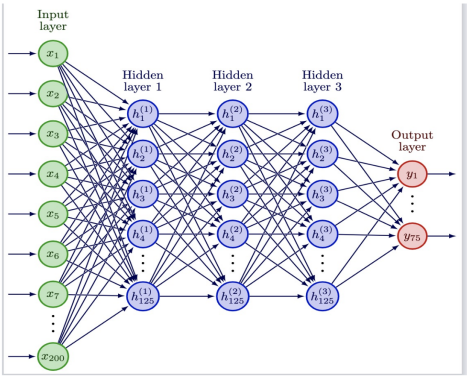


Figure 13: The structure of Neural Network

B.6 Result for Tic-Tac-Toe with Deep Learning

Result	Percentage
Wins	735 (73.5%)
Loss	45 (4.5%)
Draw	220 (22.0%)

Table 1: Results for O player trained

Result	Percentage
Wins	976(97.6%)
Loss	0 (0.0%)
Draw	24 (2.4%)

Table 2: Results for X player trained

Result	Percentage
X Wins	294 (29.4%)
O Wins	323 (32.3%)
Draw	383 (38.3%)

Table 3: Results for both player trained