Report: Towers of Hanoi

Richard Polzin – Student Number : i6145946

Introduction

This report summarizes the results of the Towers of Hanoi practical assignment for the course Foundation of Agents. It is part of the master Artificial Intelligence at Maastricht University.

The assignment was solving the Tower of Hanoi problem with three pins and two disks. It was only possible to move one disk at a time and there is a chance of making mistakes. This problem should be modeled as an MDP and solved using Value iteration and Policy iteration.

The report will describe the states and actions as well as the optimal policy. The utility of each state for that policy will be noted and the convergence speed will be analyzed. Finally the differences between Value Iteration and Policy Iteration will be discussed.

States and Actions

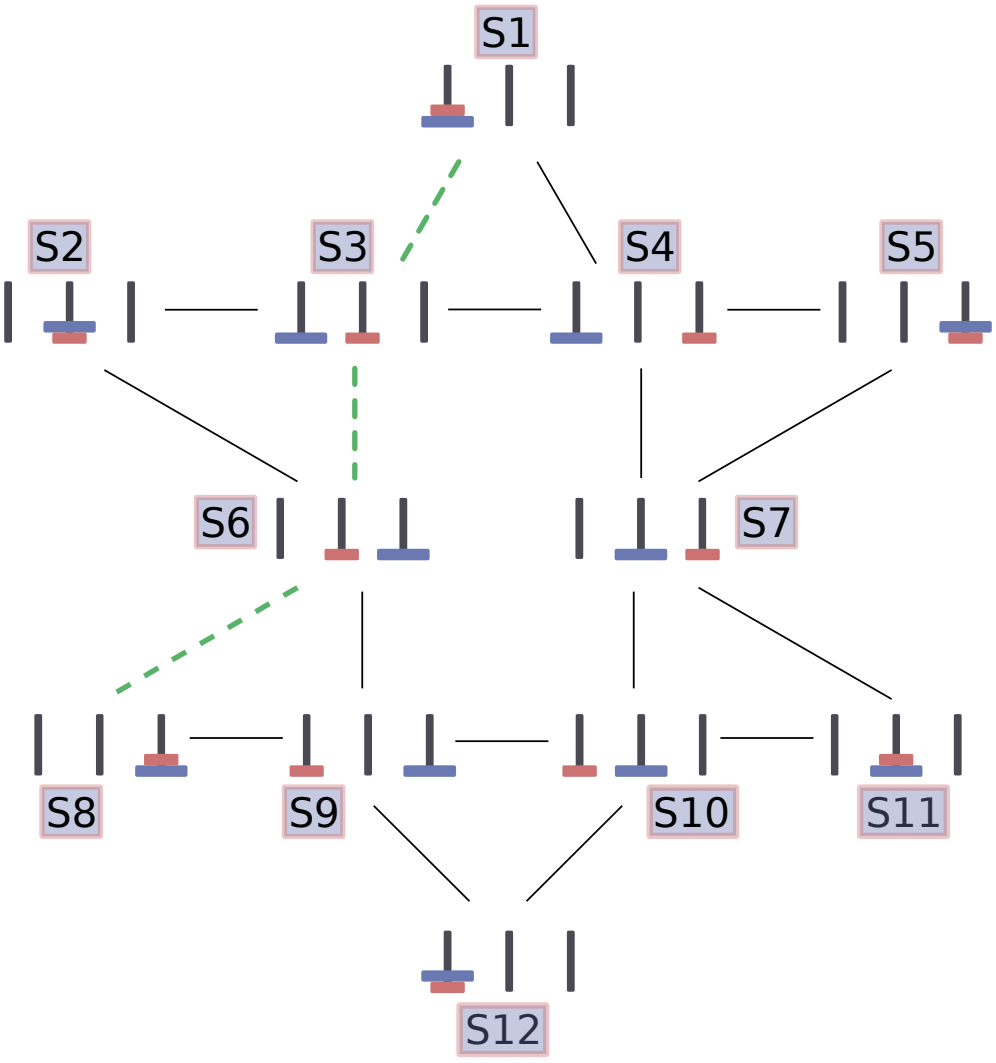

Illustration 1: Visualization of the State-Space

Illustration 1 shows the state-space. The starting state is S1. The principal variation is highlighted. Every line connecting a state to another corresponds to an action for those states. All the actions are bidirectional. The final state is state S8. Actions leading to S8 are not bidirectional. On every state the agent can decide to do nothing. This is achieved by introducing an implicit action leading from *s* to *s* at every state *s*. It yields a reward of *-1* if no bigger disk is on top of a smaller one. If the big disk is on top of the small one staying in that state adds a reward of *-10*. In the final state there is no punishment for staying in the state. Furthermore this is the only possible action in that state.

Optimal Policy

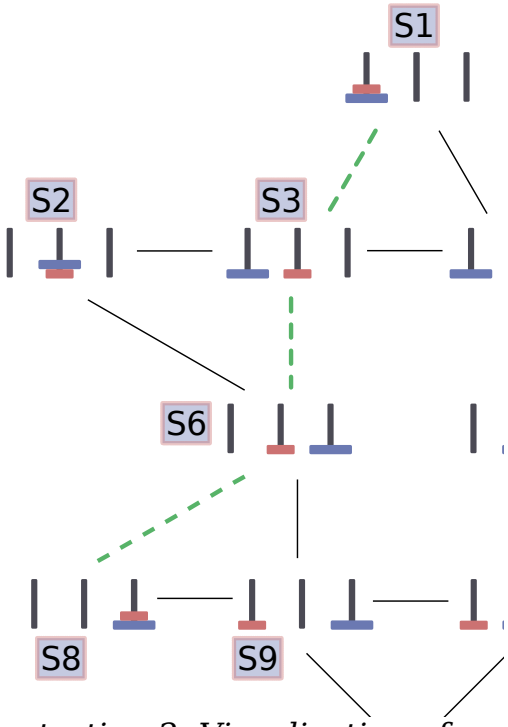

Illustration 2: Visualization of the Principal Variation

The optimal policy is highlighted in the state space as seen in illustration 2. First the small disk is moved to the center pin (S3). Then the big disk is moved to the rightmost pin (S6). Finally the small disk is positioned on top of the big disk on the right S(8). In conclusion the problem is solvable in three moves.

Value and Policy iteration both calculate utilities for every state. The best move is the one with the highest utility.

Table 1 shows the utilites that are calculated by the Policy iteration. Those are the utilities that are reached when the optimal policy is calculated.

Initial State *s*	**Action = Move to State *s'***	**Utility**
S1	S3	75.387
S2	S6	86.754
S3	S6	85.929
S4	S3	75.387
S5	S4	66.848
S6	S8	98.791
S7	S10	75.387
S8	S8	0
S9	S8	98.791
S10	S9	85.929
S11	S10	75.387

Table 1: Optimal Utilities and best Actions based on Policy Iteration

Illustration 3 visualizes the best actions for every possible state. The principle variation is highlighted again. Analyzing the resulting actions shows that all the actions lead towards at the goal state as expected.

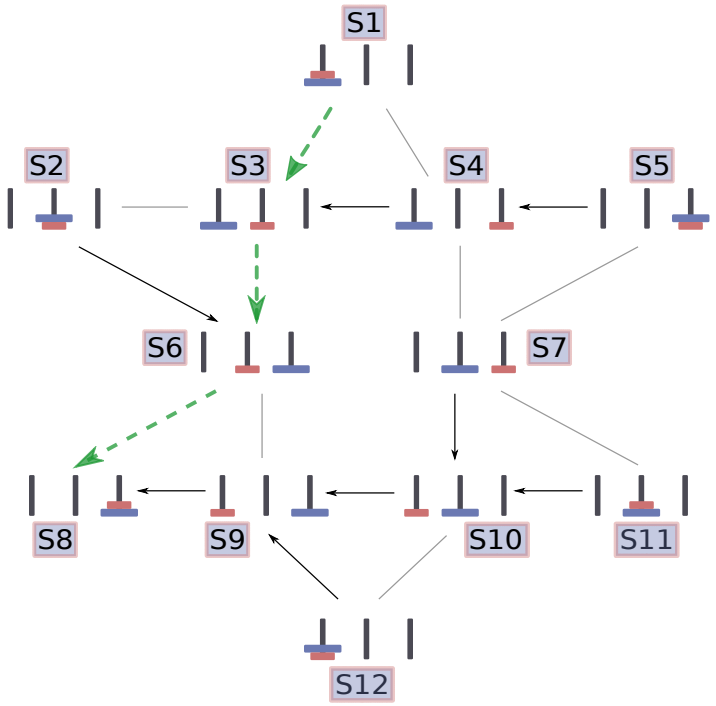

Illustration 3:
Visualization of the State-Space and the best Actions

Convergence Speed

Table 2 and Illustration 4 show how many iterations the Value iteration needed to reach a improvement smaller than epsilon.

Epsilon	Iterations
10	4
1	5
1×10^{-1}	6
1×10^{-5}	8
1×10^{-10}	10
1×10^{-15}	12

Table 2: Overview of different setups for the Value iteration

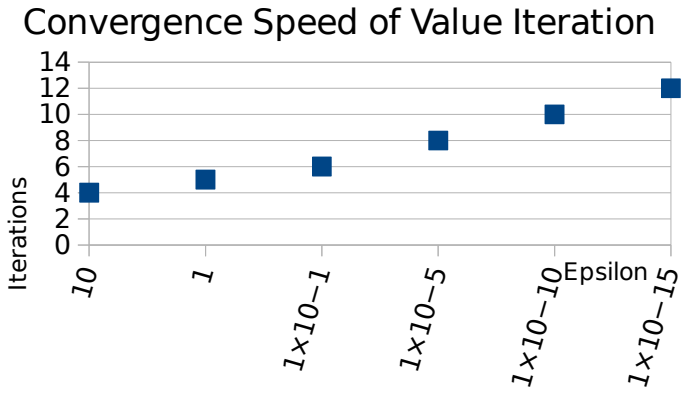

Illustration 4: Convergence Speed

Discussion

The main focus of this part is to elaborate on the difference between Value and Policy iteration. First of all Value and Policy iteration will be described on their own. The effort in implementation, the speed as well as the general outlines will be presented for each strategy. In the next part the advertised comparison will be executed.

Value iteration calculates the utility for every action in every state and updates it in every iteration. The action with the highest utility is assumed to be the optimal action. When the utility does not update more than a certain threshold the iteration is stopped. Implementing Value iteration is rather simple. The core of the implementation is a twofold loop that iterates on the states and every actions that can be executed on this state. In addition to that the calculated utilities have to be tracked and updated when a better utility becomes available. The time saved implementing Value iteration is lost again when it comes to performance. First of all with Value iteration only an approximate solution can be generated. Secondly the iteration speed can be pretty slow. More details will follow in the comparison.

Policy Iteration improves a policy in every iteration. The starting policy defines some action for every state. Utilities are calculated and compared to the utilities of the actions the policy did not choose. If an action is better than the one chosen by the policy, the policy is updated. When the policy does not change anymore the optimal policy is found. Implementing Policy iteration is more complex than value iteration. First utilities for the policies actions have to be calculated and the resulting linear equation has to be solved. Then the utilities for the other actions have to be calculated and compared to those chosen by the policy. Policy iteration always generates the guaranteed optimal policy, as it improves a policy until all the utilities are at their best. On the other hand solving a linear equation can become pretty expensive.

Comparing Value and Policy iteration the implementation effort differs noticeable but not significant. On the one hand Policy iteration generates the exact solution at the drawback of solving linear equations. Those can become pretty expensive for large state-spaces. On the other hand Value iteration generates an approximate solution with less computational effort, but at the risk of getting a policy that is not optimal.

In Conclusion Policy iteration should be used whenever sufficient. Policy iteration provides the exact solution which is favorable in most cases. Only if Policy iteration is too expensive, or optimality is not required Value iteration should be used.