Maastricht University

Foundation of Agents

Assignment Report:

Tower of Hanoi

Maastricht University

Faculty of Science and Engineering

Department of Data Science and Knowledge Engineering

Authors

Alessandro Scoppio, i6200653

Giorgos Patsiaouras, i6198785

1 Introduction

The ”Tower of Hanoi” problem to solve in this assignment is a mathematical puzzle consisting of three pins and two disks of different sizes, namely $diskA$ and $diskB$ such that $diskA > diskB$. For each step the agent can perform an action: move the top disk from a source pin to a target pin. The puzzle is considered solved when $diskA$ and $diskB$ have been transposed to the third pin, with smaller disk on top of the bigger one. Reaching this configuration will declare the end of the puzzle, as well as its solution. Having a finite number of states S and actions A, this problem can be easily modeled as a Markov Decision Process (MDP).

1.1 States Description

The number of possible states for a "usual" Tower of Hanoi puzzle with n disks is 3^n. Differently, in our specific environment, it is possible for each of the three pins to have $diskA$ **on top** of $diskB$, that means adding three more possible states, resulting in a total of 12:

State	pin1	pin2	pin3
s1	B	|	|
	A	|	|
s2	|	|	|
	A	B	|
s3	|	|	|
	A	|	B
s4	|	A	|
	|	B	|
s5	|	|	|
	|	B	A
s6	|	|	B
	|	|	A

State	pin1	pin2	pin3
s7	|	|	|
	B	|	A
s8	A	|	|
	B	|	|
s9	|	|	|
	B	A	|
s10	|	B	|
	|	A	|
s11	|	|	A
	|	|	B
s12	|	|	|
	|	A	B

Table 1.1: State Representation

State S_6 in Table 1.1 is the absorbing state: the reward in this state will be 100 and the probability of performing every action will be zero in order to ”trap” the execution in that final state.

1.2 Actions Description

An action is defined as the movement of the top disk from a source pin to a target pin. All of the actions can be found in the table below:

Action	Source pin [From]	Target pin [To]
a_1	1	2
a_2	1	3
a_3	2	1
a_4	2	3
a_5	3	1
a_6	3	2

Table 1.2: Action Representation

2 Optimal Policy

Both **Policy Iteration** and **Value iteration**, produced the same optimal policies $\pi^*(s_i)$, with the exception of the absorbing state which is never being updated with a new policy.

$\pi^*(s_1) = a_1$	$\pi^*(s_7) = a_2$
$\pi^*(s_2) = a_2$	$\pi^*(s_8) = a_2$
$\pi^*(s_3) = a_6$	$\pi^*(s_9) = a_4$
$\pi^*(s_4) = a_4$	$\pi^*(s_{10}) = a_3$
$\pi^*(s_5) = a_4$	$\pi^*(s_{11}) = a_5$
$\pi^*(s_6) = a_1$	$\pi^*(s_{12}) = a_5$

By using the Optimal Policy starting from State 1, the Optimal Route is defined as:

$S_1 \rightarrow S_2 \rightarrow S_5 \rightarrow S_6$

3 Optimal Utility

Optimal Utility $U^*(s)$ calculated using the Optimal Policy $\pi^*(s)$, for $s \in S$

$U^*(s_1) = 75.387157$	$U^*(s_7) = 98.791209$
$U^*(s_2) = 85.928781$	$U^*(s_8) = 86.754469$
$U^*(s_3) = 75.387157$	$U^*(s_9) = 85.928781$
$U^*(s_4) = 86.754469$	$U^*(s_{10}) = 75.387157$
$U^*(s_5) = 98.791209$	$U^*(s_{11}) = 66.848441$
$U^*(s_6) = 0.000000$	$U^*(s_{12}) = 75.387157$

It is worth mentioning that Value iteration and Policy Iteration returned exactly the same values. Nevertheless, Policy Iteration converged in 2ms against 3.5ms of value iteration; Furthermore, the iterations needed to converge were 4 for policy iteration, instead of the 22 iterations needed by value iteration method.

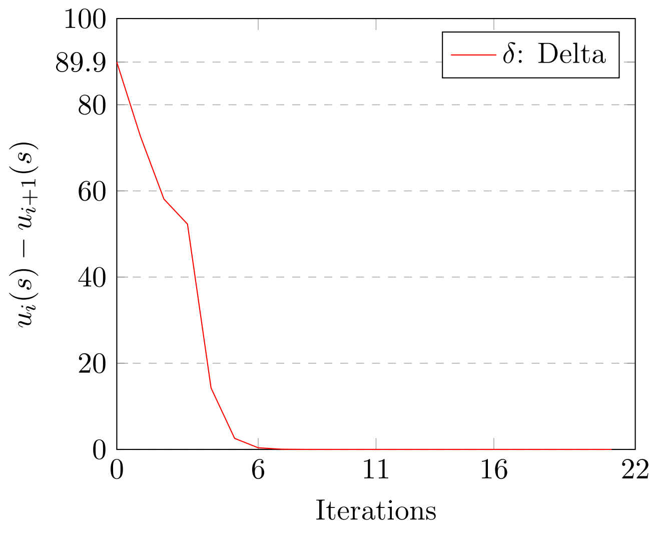
4 Evaluation of Convergence Speed

Value iteration reaches delta values close to zero already from the 6_{th} iteration. The Value Iteration was performed with the minimum e possible which is 2.220446049250313e-16.

$$\boxed{\delta = \max_{s \in S} |u_i(s) - u_{i+1}(s)|}$$

Iteration	*Delta*	*Iteration*	*Delta*
0	89.9	11	$1.0266141060810696E-5$
1	72.72900000000001	12	$1.129589620063598E-6$
2	58.10049000000001	13	$1.2199848242744338E-7$
3	52.29044100000001	14	$1.2978688346265699E-8$
4	14.255543970000005	15	$1.3627783346237266E-9$
5	2.567119845600004	16	$1.4155432381812716E-10$
6	0.38578896513000416	17	$1.4566126083082054E-11$
7	0.05208820644912748	18	$1.4637180356658064E-12$
8	0.006568322665856385	19	$1.8474111129762605E-13$
9	$7.88246501755907E-4$	20	$1.4210854715202004E-14$
10	$9.125137931675908E-5$	21	0.0

Table 4.1: Delta value for each iteration



5 Implementation Details

The problem was modeled and implemented in Java [1], using Jama [2] for solving the linear equations system during policy iteration. All methods used are declared in the class Hanoi and called in the Main class.

To execute the program run the following command

```
$ java -jar hanoi.jar
```

Requirements: Java JDK 1.8

[1]Is it worth to mention that State and Actions in the implementation are indexed starting from 0.
[2] https://math.nist.gov/javanumerics/jama/