

Decision Making in Intelligent Systems

Lab course 2004–2005

Faculty of Science,
University of Amsterdam

April 26, 2005

1 Introduction

This manual describes the lab course of the third year Bachelor Artificial Intelligence course “Decision Making in Intelligent Systems”. More information can be found on the lab course webpage at

<http://www.science.uva.nl/~mtjspaandmis>

The goal of the lab course is to practice computing decision making strategies for agents inhabiting stochastic environments. You will be asked to experiment with several dynamic programming and reinforcement learning techniques introduced in the lectures.

2 One player poker

Our running example will be an optimization problem resembling a simplified poker game. The game has only one player, which plays against the house. At the start of the game the player puts an ante to enter the game. The cards are dealt in alternating rounds: in each round either the player or the house receives a card, after which the player decides how much he will bet, i.e., how much he will increase the pot. The bet can range from 0 to a fixed integer (e.g., 1). At the end of the game, when player and house have been dealt an equal amount of cards, the player either wins or loses the pot, depending on whether or not his hand beats the hand of the house according to poker rules.

The objective of the player is to maximize his profit, i.e., to raise the stakes as high as possible when he is confident he will win, and to refrain from betting when he believes he will lose. Initially, we will focus on the following instance of one player poker:

- The deck consists of four suits of four cards each {J,Q,K,A}, resulting in a total number of 16 cards.
- In total four cards are dealt, two to the player and two to the house.

- In each of the three betting rounds the player can choose to bet one mark or no marks.
- The ante is one mark.
- In this simplified poker setting the only possible card combination is a pair, which beats any single card.
- In case of a draw the player receives 0.

3 Assignment 1

Your first assignment is to solve the one player poker instance mentioned above using value iteration. Figure 1 shows an example trace of a particular game. Solving one player poker requires modeling the game as a Markov Decision Process, and you should start by modeling the state space S . Use the states to define a transition model $p(s'|s, a)$ which for every state $s \in S$ gives the probability that if the player executes action $a \in A$ in s , it will result in a jump of the system to state s' . The reward should be a function $r(s, a)$, which for every s and a returns the immediate reward for the player. Implement value iteration and solve your MDP. Test the resulting policy by simulating a large number of games. Compare with a random policy, as well as with a heuristic policy (e.g., how you would play this game).

Matlab is the suggested programming language, but you're free to implement your program in any programming language you want (but note that our programming support may vary). You're expected to write a report describing your approach and findings and to hand in your code. The lab report should be two to three pages, and \LaTeX is preferred.

4 Assignment 2

The 2nd assignment is identical to the 1st one, except that each player is now dealt a total of 3 cards (so there are 5 betting rounds), and an additional poker rule is added: a triple beats a pair (e.g., $\{J, J, J\}$ beats $\{A, A, J\}$). The state space is now much larger than in the 2-card version, which makes on-line methods more appropriate than dynamic programming. In this assignment you are asked to learn an optimal policy for the player using the following methods (and compare results):

1. Monte Carlo (a method of your choice from Chapter 5 of Sutton & Barto),
2. Q-learning,
3. TD(λ).

5 Assignment 3

The final assignment involves computing policies for a one player poker game in which not all cards are revealed. You should use the poker setting as defined in section 2, i.e., with 2 cards dealt per player. The difference in this assignment is that our player does not observe the first card the house receives, but does see the second one. As the state is now only partially observable to the player, this turns the problem into a POMDP. In order to solve the POMDP, you will need to define an observation model $p(o|s)$ that for every state s specifies the probability of observing a particular card o . Using the transition and observation model you compute at every time step the belief $b(s)$ of the player, which is a probability distribution over all states. The belief is a Markov state signal for the planning problem, and there are many methods for solving POMDPs based on belief states. In this assignment we ask you to explore and compare two simple methods:

1. Compute a heuristic policy using Q_{MDP} , based on the Q values computed in assignment 1.
2. As our one player poker game has particular structure there is only a finite set of reachable beliefs in the POMDP, which allows mapping the POMDP model into a discrete-state MDP and then solve it using standard dynamic programming methods, e.g., value iteration.

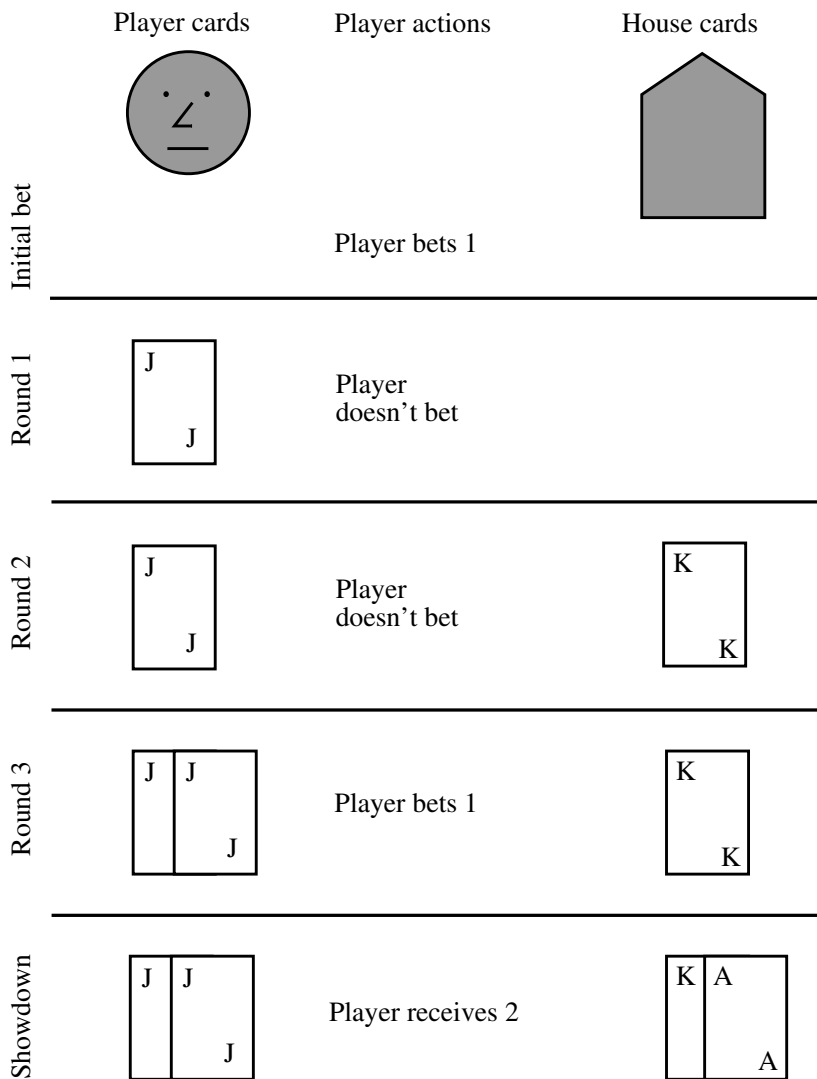


Figure 1: Example trace of a one player poker game. Left column shows the cards our player receives, right column the cards of the house and the middle column the actions the player takes. Time progresses from top to bottom. In each betting round the player can decide to bet (add 1 to the pot) or decide not to bet (add 0 to the pot). He starts with an ante of 1 and the game moves to round 1. The player receives a jack and decides not to bet (as it is the lowest ranking card in the deck). The game moves to round 2 and the house receives a king. Our player again decides not to bet, but in round 3 he is dealt another jack. As the two form a pair, he decides to raise the stakes and bets. The house receives an ace as final card, and is beaten by the pair of jacks. At showdown our player receives the pot in cash.