Lab 3

Searching and Planning, Computational Game Theory

Activity 1: Pay-Off Matrix and Nash Equilibrium. In this activity, we study competitive and collaborative games by designing pay-off matrices and by identifying an equilibrium *(no code requested)*.

A) We suppose that only two blockbusters movies are jockeying for position: Fox's Narnia and Warner Bros's Harry Porter. Suppose that both super-productions are released in November share a total of USD 500 million in ticket revenues, whereas blockbusters released in December share a total of USD 800 million. Formulate the game played by Fox and Warners Bros in normal form using pay-off matrix and determine the game's Nash equilibrium.

B) (Optional) Suppose that Ericsson and Nokia are the two primary competitors in the market for 4G handsets. Each firm must decide between two possible price levels: USD 100 and USD 90. Production cost is USD 40 per handset. Firm demand is as follows:

- if both firm price at 100, then Nokia sells 500 and Ericsson 800;
- if both firms price at 90, then sales are 800 and 900, respectively;
- if Nokia prices at 100 and Ericsson at 90, then Nokia's sales drop to 400, whereas Ericsson's increase to 1100;
- if Nokia prices at 90 and Ericsson at 100 then Nokia sells 900 and Ericsson 700.

We suppose that both firms choose prices simultaneously. Find the Nash equilibrium. Conclude on the best strategy assuming firms strategies will not change.

Activity 2: Stochastic Hill Climbing. In this activity, we propose to implement the *Stochastic Hill Climbing* algorithm. The implementation can be realized using the programming language of your choice (for instance C , C++ , Java or Python).

A) Explain the key idea and features of the class of *Hill Climbing* approaches. Why the Hill climbing optimization is adapted for local search? Is a solution existing all the time? Why this optimization is useful for Artificial Intelligence applications?

B) Implement the *Stochastic Hill Climbing* search (in the language of your choice, C , C++ , Java or Python) as described in the lecture notes.

C) (Optional) Apply the *Stochastic Hill Climbing* technique to solve the so-called *knapsack problem*. The knapsack problem is stated as follows: *"Consider a thief gets into a home to rob and he carries an old knapsack which has limited capacity. There are fixed number of items in the home - each with its own weight and value - Jewellery, with less weight and highest value vs tables, with less value but a lot heavy. He either takes it or leaves it"*. Here an instanciation of the problem:
Knapsack Max weight: $W = 10$ (units)
Total items : $N = 4$
Values of items : $v = [10, 40, 30, 50]$
Weight of items : $w = [5, 4, 6, 3]$

Activity 3: Backtracking and Search.

A) *Backtracking*. First consider the following cut-free program:

```
q(X,Y) :- i(X),j(Y).
```

Suppose we add a cut to the clause defining:

```
q(X,Y) :- i(X),!,j(Y).
```

How the resolution is made in Prolog? Study how both program verison behave. Then, explain the impact of the cut over the search. Finally, state about how the backtracking is done in Prolog.

B) *Tower of Hanoi* is famous puzzle is to move N disks from the left peg to the right peg using the center peg as an auxiliary holding peg. At no time can a larger disk be placed upon a smaller disk. The following diagram depicts the starting setup for $N = 3$ disks. Here is a recursive Prolog program that solves the puzzle. It consists of two clauses.

```
move(1,X,Y,_) :-
write('Move top disk from '),
write(X),
write(' to '),
write(Y),
nl.
move(N,X,Y,Z) :-
N>1,
M is N-1,
move(M,X,Z,Y),
move(1,X,Y,_),
move(M,Z,Y,X).
```

The variables filled in by '_' (or any variables beginning with underscore) are 'don't-care' variables. Write a version of this code to prevent the backtracking.

C) (Optional) Write a Prolog code searching for the maximum value in a binary tree. Use the cut operator to prune Prolog's search tree. Is the result different for a balanced and unbalanced binary search tree? Explain why the cut operator is not needed for an balanced binary search tree.

Activity 4: Rosenbrock and Steepest Descent. One classic hard test function for optimization methods is the Rosenbrock's banana function. In this activity, we propose to find the local minimum for the so-called Rosenbrock function.

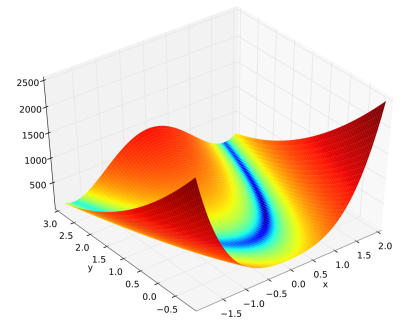

Figure 1: Rosenbrock Function

Given the Rosenbrock function as input defined as follows:

$$f(x, y) = 100\left(y - x^2\right)^2 + (1 - x)^2 \tag{1}$$

We supply its corresponding gradient:

$$\nabla f = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right) \tag{2}$$

where

$$\frac{\partial f}{\partial x} = 2x - 2 - 200x \cdot \left(y - x^2\right) \tag{3}$$

and

$$\frac{\partial f}{\partial y} = 200\left(y - x^2\right) \tag{4}$$

A) (Optional) Write the code for the *Hill Climbing* (in C,C++ or Java). Start a search at the origin and perform the *Hill Climbing* to find the global minimum of the given Rosenbrock function.

B) Write the code for the *Stochastic Hill Climbing* (in C, C++ or Java). Start a search at the origin and perform the *Stochastic Hill Climbing* to find the global minimum of the given Rosenbrock function.

C) Write the code for the *Gradient Descent* (in C, C++ or Java). Start a search at the origin and perform the *Gradient Descent* to find the global minimum of the given Rosenbrock function.

D) (Optional) Write the code for the *Stochastic Gradient Descent* (in C, C++ or Java). Start a search at the origin and perform the *Stochastic Gradient Descent* to find the global minimum of the given Rosenbrock function.

E) Compare the accuracy of all implemented methods. What is the more accurate technique? Explain the differences. How to improve the accuracy of this result?