Konrad Zdanowicz 180394

Markov Decision Problems and Reinforcement Learning

Introduction

The goal of the exercise was to implement the algorithm that will find the solution for the simple Markov decision problem. Besides the algorithm itself, the scope of the project was to develop a formal description of the problem that would serve as an input to the program, and the appropriate output visualisation was expected in the form of Gnuplot scripts/graphics.

Solution

For implementation the Java language was used. The compressed package contains both Maven source project as well as the executable jar. Also the mechanism of property reader was used for easy and integrated input data parsing and reading, and the output data was marshalled to the gnuplot dat files, then plotted by the gnuplot gpl file and saved to the png file of the same name.

Input data

To run the program, the appropriate *.properties file of following syntax must be created:

```
# board params
N=4
M=4

# default reward for empty field
r=-1

# board description
states.start=(1,1)
states.prohibited=(3,1)
states.terminal=(4,1,100)
states.special=(3,2,-20)


# uncertanity model
a=0.8
b=0.1

# discount factor
d=0.99
```

```
# error maximum value
e=0.01
```

Besides the basic parameters following convention in the task description, there are states defined in the board description. Following notation:

```
states.start=(1,1)            # S
states.prohibited=(3,1)       # F
states.terminal=(4,1,100)     # G
states.special=(3,2,-20)      # B
```

Corresponds to the following board layout:

		B	
S		F	G

This input can be easily modified and extended, for example by adding another states in CSV standard.

Usage

Such data stored in property file may be easily used for the problem resolution, using following command:

> java -jar markov-meu.jar board

That will load „board.properties” to the program and do the calculation using value iteration procedure. The algorihm runs as long as the error attribute is set. The standard output is a convenient way of previewing the algorithm result. For such input, the following output is printed:

```
Reading data from task1.properties...
(i) The board was parsed correctly.
Now executing value iteration algorithm...
_       _       _       _
_       _       _       _
_       _       B       _
S       _       F       G
-------------------------
Algorithm ended after 28 steps.
_ 65,4344 ▶     _ 74,6431 ▼     _ 77,7445 ▼     _ 74,7639 ▼
_ 73,9268 ▶     _ 77,6118 ▶     _ 80,7491 ▶     _ 85,9435 ▼
_ 70,5266 ▶     _ 73,4350 ▲     B 60,4652 ▶     _ 91,5045 ▼
S 61,1110 ▲     _ 63,2082 ▲     F 0,0000 #      G 100,0000 100.0
```

```
-------------------------
▶       ▼       ▼       ▼
▶       ▶       ▶       ▼
▶       ▲       ▶       ▼
▲       ▲       #       100.0
-------------------------
Saving results to task1Out.png
```

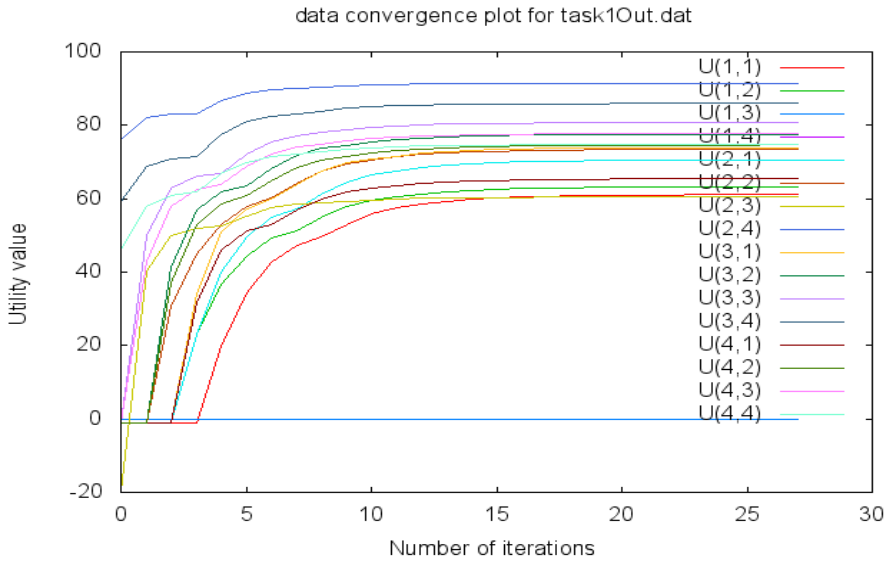
The standard output displays information about the last utility results with the directions presented in the form of small triangles. For the more detailed information, there are specially generated files compatible with gnuplot.

Output

Following results are expected to appear on the line with the property file:

- boardOut.png
- boardOut.dat
- boardOut.gpl

Where the first is a an image of converge data plot, that contains utility value for each field during each iteration. The sample look of that file for the input data from above is as follows:



Although not too readable because the density of data, all information needed that was calculated by the algorithm is shown. In case the manual look at the data result is needed, the corresponding [name]Out.dat file contains all data in the obvious gnuplot convention.

For the custom graph displaying, the [name]Out.gpl must be edited.

The analysis of this example is not too interesting, as the general attitude of the agent is to move in the direction of the terminal state without bigger worry about steping on the special state that has a negative influence

(although the likeness of agent going through the unlucky field is fairly slow (we can see that the line which 0-iteration value begins with -20 utility value has a relatively close value to the others, although obviously lower).

Altered reward parameters

```
states.terminal=(4,1,50)
states.special=(3,2,-100)
```

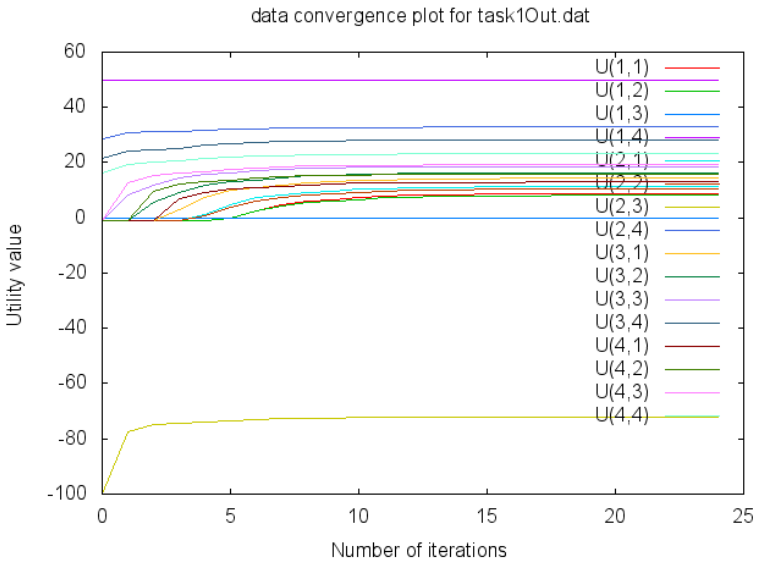
Following parameters were modified comparing to the initial file. The change implies, that the terminal state now is not so expensive for the user, and the special state is much more dangerous.

It may be noticed, that for such parameters, the system answers as it is expected:

```
>       >       >       v
>       >       ^       v
^       <       >       v
^       ^       #       50.0
```

The orange square is the occurance of the very dangerous special field; the algorithm tries to minimize the chance of the agent stepping on this field (however the risk of stepping on this field is still unavoidable if trying to reach the optimal solution).

Taking a closer look at the complete data convergence plot emphases now the difference in likeness of stepping on the bonus field comparing to the ordinary ones.



Alter action uncertanity model

This time, the next experiment was performed – the uncertanity model was modified completely to twist the standard approach:

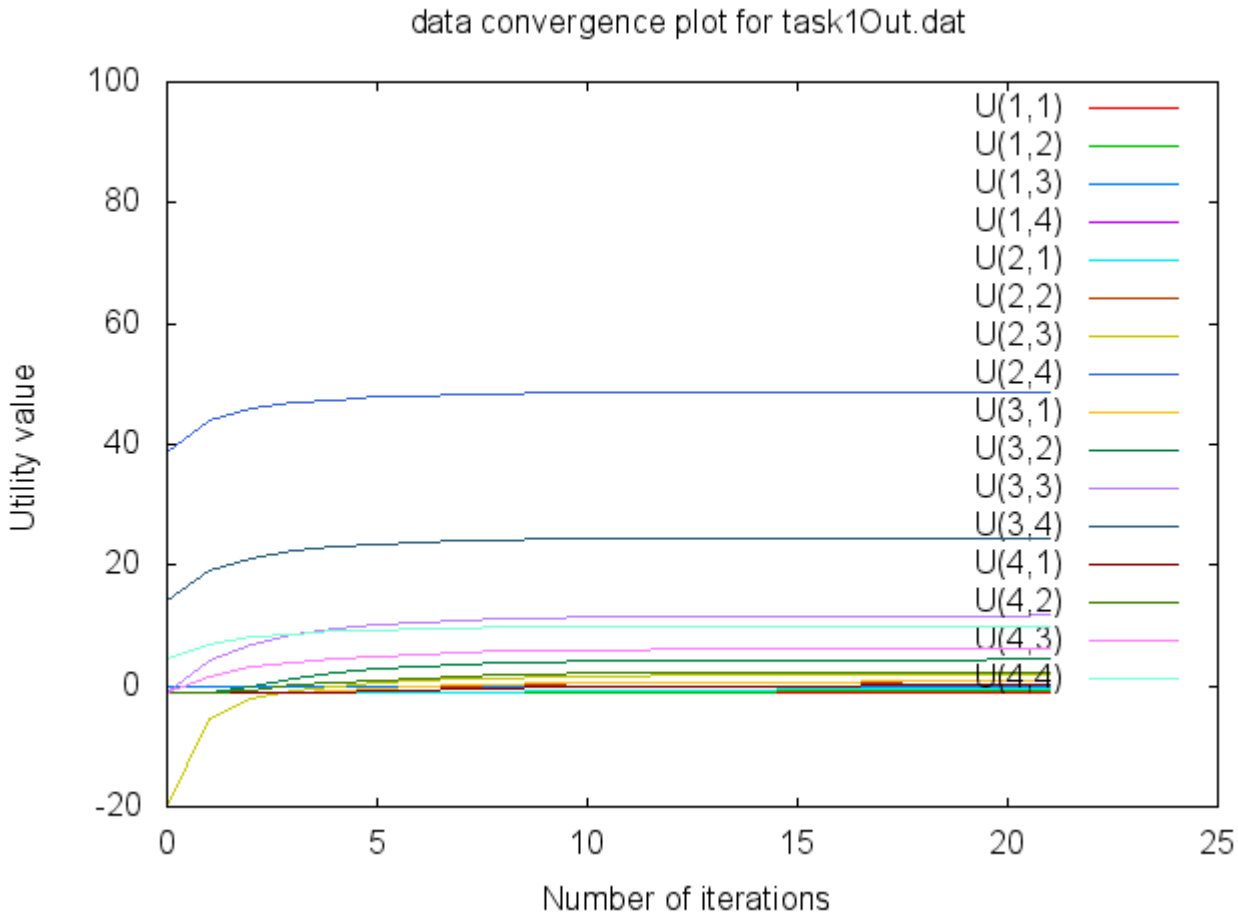
```
# uncertanity model
a=0.2
b=0.4
```

What was changed is the sanity of the agent – now the agent is very likely to make an irrational decision, while the probability that the desired direction will be followed is 0.2, that gives the very high frequency of the wrong decision being made – 80%!

The algorithm however finds the solution that tries to minimize the possible loss of such agent's behaviour.

```
v       v       v       <-
^       ^       ^       <-
^       >       ^       <-
-       <       #       100.0
```

The solution suggest the user in fact to follow the irrational directions as it may seem, however taking the closer look at the moves explains, that those directions actually lead to the optimal solution.



It is interesting how the most values stabilize around 0, while the very few are more polarized:

```
_ 0,0699 v      _ 2,3330 v    _ 6,1638 v      _ 9,9292 <-
_ 0,7430 ^      _ 4,3713 ^    _11,6589 ^      _24,5172 <-
_-0,5609 ^      _ 0,7423 >    B 1,8783 ^      _48,6799 <-
S-1,0000 -      _-0,9055 <    F 0,0000 #      G100,0000 100.0
```

The utility grows as it is closer to the terminal state, and is suprisingly high around the bonus value itself, which is certainly the least desired field to step on.

Change the discount factor

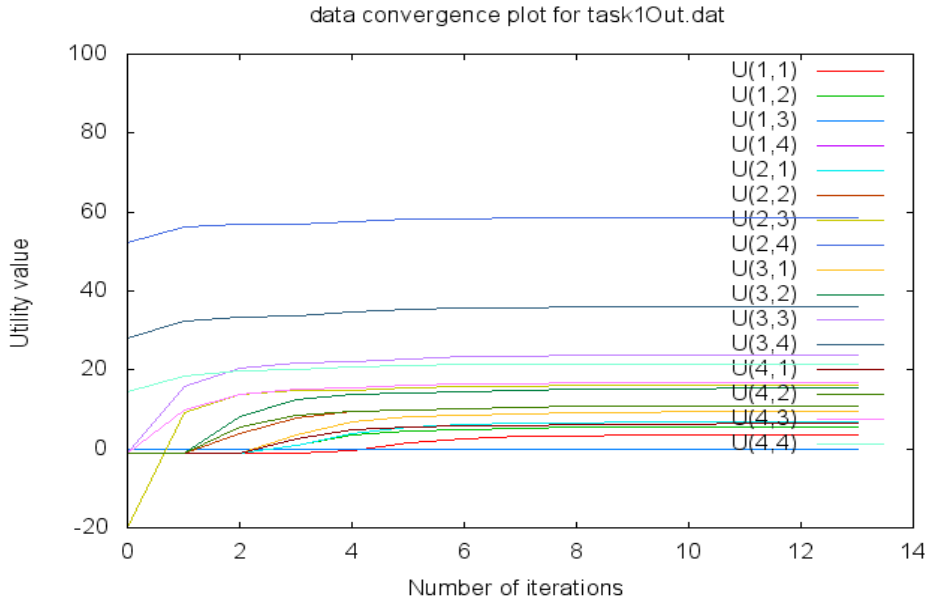
The last analysis of the algorithm touches the discount factor modification, according to following property:

```
d=0.7
```

The small change in discount factor (from 0.99 to 0.7) results in very different character and shape of the solution:

```
>       v       v       v
>       >       >       v
>       >       >       v
^       ^       #       100.0
```

Converge plot:



Those attributes show how the discount factor affects the utility value

distribution, ergo solution – the agent doesn't pay too much attention to the fact, that the negative bonus state is going to be visited. That can be observed also by flattening of the charts.

Results

Following exercise turned out to be a good lesson on few topics – software development techniques, development tools usage, AI theory investigation and developing project research skills while performing the comparision of parameters influence on the solution.