Assignment on Extensive-Form Games

We consider the following problem: your agent is placed on an island with a map. The goal of the agent is to reach the destination square on the island while collecting as many gold treasures as possible while stepping at each square at most once. There are bandits on the island that want to ambush the agent and steal the collected gold. The bandits may be placed at specific squares (termed “dangerous places”); there can be at most one bandit at one dangerous place. Your agent only knows the overall number of bandits, however, he does not know the exact places where they are waiting. The bandits want to harm your agent as much as possible by deciding at which dangerous places they are going to wait. When your agent visits a place with a bandit, an attack occurs. The attack is successful with some probability p (the agent will lose all the gold and the game ends). Otherwise, the attack was unsuccessful and your agent can continue. Your agent **can react** to the information gained during the course of the game - e.g., in the case of an unsuccessful attack, the agent can exploit this information (there is one less bandit in the remaining dangerous places).

The bandits primarily decide their position at the beginning of the game. Moreover, the bandits can use sensors. If your agent enters an unoccupied dangerous place, an alarm is triggered and 1 bandit can relocate to another empty dangerous place. However, this can happen at most once. The following holds:

- the alarm is triggered only when the agent visits an unoccupied dangerous place for the first time
- there is at most 1 alarm during the game,
- if the first dangerous place the agent visits is occupied with a bandit, there will be no alarm triggered (now, nor any later in the game)
- the bandit cannot relocate to the dangerous place where the agent has triggered the alaram

Your goal is to:

1. formalize this problem as a two-player zero-sum extensive-form game, and
2. implement the sequence-form linear program (using the external CPLEX LP solver) for this game that computes the value of the game specified via the input file.
3. write down a report that will contain the extensive-form game tree (or a part of it) corresponding to the ~~second~~ third example listed below (the one with exp. value of the game 5.5). It is ok to write it down by hand as long as it is readable and understandable.

Deadline:

~~6.12.2016 4:00 CET~~ 13.12.2016 4:00 CET

IMPORTANT

- You can use CPLEX (links for the download are below) or Gurobi for solving LPs
- If you want to use other programming languages than Java or Python, consult with me [http://cs.felk.cvut.cz/en/people/bosanbra] first (submitting your solution in an unsupported language can result in a significant grading penalization)
- The assignment is graded manually since the most important part is to correctly model the problem as an extensive-form game, which means to correctly identify states, actions, sequences, and information sets.
- Construct 2 LPs (one for each player) and export both LPs into the text file.

Input

The assignment is to implement the program for solving described games. The program takes one argument (the input file) with the following format:

```
number: number of rows of the maze (M)
number: number of columns of the maze (N)
M rows, N symbols representing squares of the maze
number: the overall number of the bandits
float number: the probability of a successful attack
```

Squares of the maze are defined as follows:

```
# - obstacle
- - empty square
S - start of your agent
D - destination of your agent
G - gold treasure
E - dangerous place
```

Constants for Utility Calculation:
Your agent receives 10 points for reaching the destination and he receives 1 additional point for each collected gold treasure. If a bandit successfully attacks the agent, your agent receives 0 points no matter how many gold treasures he has been carrying before the attack. If your agent follows a path that does not end in the destination square (e.g., if it leads to a dead end), your agent receives 0 points.

Input Example:

```
7
9
#########
#E----EG#
#-##-##E#
#S##-##-#
#-##-##-#
#E-----D#
#########
2
0.5
```

Output

The output of your program consists of several parts. First, on the standard output write out all the following information.

(1) a list of the sequences for each of the players:

```
AGENT:
S1:[first_sequence]
S2:[second_sequence]
...
ATTACKER:
Q1:[first_sequence]
Q2:[second_sequence]
...
```

(2) calculated extended utility function **g** for each combination of sequences (you can exclude the zero ones):

```
AGENT\ATTACKER |  Q1      |    Q2    | .... |
-------------------------------------------
S1             | g(S1,Q1) | g(S1,Q2)| ...  |
S2             | g(S2,Q1) | g(S2,Q2)| ...  |
...
-------------------------------------------
```

(3) calculated realization plans of the sequences in Nash equilibrium (list only sequences that are in the support of NE (i.e., they are played with a strictly positive probability)):

```
SOLUTION_AGENT:
S1:[realization_plan_of_first_sequence]
S2:[realization_plan_of_second_sequence]
...
SOLUTION_ATTACKER:
Q1:[realization_plan_of_first_sequence]
Q2:[realization_plan_of_second_sequence]
...
```

(4) calculated value of the game:

```
SOLUTION_VALUE:[value_of_the_game]
```

(5) export the linear program into the file `[your_login_name].lp`

Conventions & FAQ

- Your agent moves in the maze in 4 directions (up, down, left, right).
- Use the name Main.java for the main java class (or main.py) and place all your classes in one package (or directory) named by your login-name.
- As soon as the agent reaches the destination square, the game ends.

Examples and Results

The value of the game must be exactly the same (modulo double arithmetic imprecision). If the difference is larger (e.g., >0.001), your solution is still incorrect!

```
7
9
#########
#G-----E#
#-#####-#
#S--E--D#
#-#####-#
#G-----E#
#########
2
0.5
```

the value of the game: `7.096774193548387`

```
7
9
#########
#G-E---E#
#-#####-#
#S-E--ED#
#-#####-#
#G-E---E#
#########
2
0.6
```

the value of the game: `3.4064516129032265`

```
7
9
```

```
#########
#E----EG#
#-##-##E#
#S##-##-#
#-##-##-#
#E-----D#
#########
2
0.5
```

the value of the game: `5.5`

```
7
9
#########
#E--G-EG#
#-#---#E#
#S##-##-#
#-##-##-#
#E-----D#
#########
2
0.5
```

the value of the game: `6.0`

CPLEX-Java Integration

As a linear-program solver you can use IBM ILOG CPLEX [http://www-01.ibm.com/software/integration/optimization/cplex-optimizer/]. It can be downloaded using the following links:

```
MacOS : https://www.dropbox.com/s/4u6uab74o8z7udd/cplx_mos.rar
Linux 64bit : https://www.dropbox.com/s/5oxsvji4adbr13w/cplx.rar
Win 64bit : https://www.dropbox.com/s/3dr7emfv1w440d7/cplx-w64.rar
```

If you do not know the password, ask us (or your colleagues).

Using ILOG CPlex

Integrating CPLEX with java project: It is necessary to add cplex.jar to your project from …/CPLEX_STUDIO/cplex/lib/cplex.jar, javadocs can be found in …/CPLEX_STUDIO/doc/html/refjavacplex/html.

Running the program: `java -Djava.library.path=[path_to_ILOG]/CPLEX_STUDIO/cplex/bin/[architektura:x86-64_sles10_4.1] …`

LP initialization: `IloCplex cplex = new IloCplex();`

Variable initialization: `IloNumVar V = cplex.numVar([lower_bound],[upper_bound],[typ:IloNumVarType], [name]);`

Creating a generic mathematical expression:

```
IloNumExpr zero = cplex.constant(0);
IloNumExpr sum = cplex.sum(cplex.prod(2,zero),V); // (2*0 + V)
```

Creating a generic inequality in the LP: `cplex.addGe(sum,V);`

Creating dual variables in the LP: `IloRange constraint = cplex.addGe(cplex.diff(sum,V),0);`

Adding theobjective to the LP: `cplex.addMaximize(V);`

Exporting the LP to a text file: `cplex.exportModel("file_name.lp");`

Solving the LP: `cplex.solve();`

Reading the state of the algorithm (whether the solution is 'optimal' or some error has occurred): `cplex.getStatus();`

Reading the value of the variable: `cplex.getValue(V);`

courses/be4m36mas/assignment2-gt.txt · Last modified: 2017/12/10 15:37 by bosanbra