IGGI GD2/CE810: Game Design (Part II)

Lab: AI Agents for Game Design

16th May 2017

This lab is composed of two parts, using MCTS and RHEA for game design. You can do either of them, or both if you have the time! The code for both parts is on Github: `https://github.com/ljialin/spaceBattleGame`, **branch gameDesignII**.

In the RHEA part, you'll implement a (1+1)-EA solver based on a given RMHC solver. You will test both solvers on OneMax problem and the *spaceBattleGame*. Then you will use RMHC or (1+1)-EA to search the game space.

In the MCTS part, you'll work on an incomplete code for MCTS and fill in the main methods to complete the algorithm. As an optional point, you can work on improving the algorithm to play the Space Battle game in a particular way of your interest.

RHEA

The Evolutionary Algorithms (EAs) and Rolling Horizon EAs (RHEAs) have been presented in the previous lecturer. In this lab, we are going to work on the simple (1+1)-EA and Random Mutation Hill Climber (RMHC).

1.1 Solving OneMax problem using RMHC and (1+1)-EA

The objective of the OneMax problem is to maximise the number of 1s occurring in a binary string, i.e., for a given n-bit string $\mathbf{s}$, the fitness function to maximise for that string is given by

$$f(\mathbf{s}) = \sum_{i=1}^{n} s_i, \tag{1}$$

where s_i denotes the i^{th} bit in the string $\mathbf{s}$, and is either 1 or 0.

The OneMax problem has been implemented in `OneMax.java`[1]. The `trueFitness` returns the noise-free fitness given string $\mathbf{s}$ (Equation 1).

```
1  public class OneMax {
2    private double noiseStd;
3    public OneMax(double noiseStd) {
4      this.noiseStd = noiseStd;
5    }
6
7    public double trueFitness(int[] x) {
8      double sum = 0;
9      for (int i: x) {
10       sum += i;
11     }
12     return sum;
```

[1] `https://github.com/ljialin/spaceBattleGame/blob/gameDesignII/src/labSimpleRHEA/OneMax.java`

```
  }
}
```

1.1.1 Solvers: RMHC and (1+1)-EA

The pseudo-code of RMHC can be found in the lecture note[2]. The RMHC solver has been implemented in `TestOneMax.java`[3]:

```
/**
* RMHC solver
* @param d
* @param budget
* @return
*/
// TODO: 14/05/17 Read the code to solve the OneMax
public static int[] solveRMHC(int d, int budget) {
  int[] x = init(d);
  Random rdm = new Random();
  int t = 0;  // counter
  System.out.println("Optimisation starts with true fitness=" + oneMax.
     trueFitness(x) + ", budget=" + budget + " evaluations.");
  while (t<budget) {
    // reproduction and mutation
    int[] xp = x;
    int mut = rdm.nextInt(d);
    if (rdm.nextDouble()>0.5) {
      xp[mut] = 1 - xp[mut];
    }
    // evaluation
    double fitnessParent = oneMax.fitness(x);
    double fitnessChild = oneMax.fitness(xp);
    // comparison and selection
    if (fitnessChild >= fitnessParent) {
      x = xp;
    }
    t++;
  }
  return x;
}
```

[Exercise] Solving the OneMax problem using RMHC and (1+1)-EA. *Hint: You may need to repeat the experiment several times as the solvers are stochastic.*

1. Please read and run the following code to solve the OneMax problem using RMHC (code already in the project). You can vary the problem dimension and/or the budget.

```
public static void main(String[] args) {
  int d = 10;             // dimension on OneMax problem, genome
     length
  int budget = 2000;      // optimisation budget, fitness evaluations
  /** Noise-free case */
  double noiseStd = 0;
  // TODO: 14/05/17  Uncomment the following lines to test with noisy
     OneMax
  //        /** Noisy case with std = 1 */
  //        double noiseStd = 1;
  oneMax = new OneMax(noiseStd);
  /** Solve the OneMax using RMHC */
  int[] solutionRMHC = solveRMHC(d, budget);
```

[2] http://orb.essex.ac.uk/ce/ce810/gd2/2017/lectures/RHEA.pptx
[3] https://github.com/ljialin/spaceBattleGame/blob/gameDesignII/src/labSimpleRHEA/TestOneMax.java

```
12      System.out.println("The true fitness of optimised solution is "
13      + oneMax.trueFitness(solutionRMHC));
14  }
```

2. Please implement the (1+1)-EA to solve the OneMax problem. Hint: You can copy/paste the method `solveRMHC` and change the mutation operator.

3. Try to solve the same OneMax problem using (1+1)-EA and RMHC given same budget and compare the performance.

1.1.2 (Optional) Noisy OneMax problem

The OneMax corrupted by some additive Gaussian noise with constant variance 1 is formalised below:

$$f'(\mathbf{s}) = f(\mathbf{s}) + \mathcal{N}(0,1), \tag{2}$$

where $\mathcal{N}(\mu, \sigma^2)$ denotes a Gaussian noise, with mean μ, and variance σ^2. The code is provided.

```
1 public double fitness(int[] x) {
2   double sum = trueFitness(x);
3   if (noiseStd == 0) {
4     return sum;
5   }
6   Random rdm = new Random();
7   return (sum + noiseStd * rdm.nextGaussian());
8 }
```

[Exercise] (Optional) Solving the noisy OneMax problem using RMHC and (1+1)-EA. *Hint: (i) While evaluating a string, you may need to re-evaluate it several times, i.e., resampling, as the fitness is noisy. (ii) You may need to repeat the experiment several times as the solvers are stochastic.*

1. Solve the noisy 10-dimensional OneMax problem using (1+1)-EA and RMHC **without** resampling given same budget (e.g. 500) and compare the performance.

2. Solve the noisy 10-dimensional OneMax problem using (1+1)-EA and RMHC **with** r **resamples** (e.g., $r = 2, 5, 10$) given same budget (e.g., 500) and compare the performance.

1.2 Design new agents for *spaceBattleGame*

Please download the framework on Github `https://github.com/ljialin/spaceBattleGame.git`, branch "gameDesignII". This *spaceBattleGame* is adapted to the General Video Game AI (GVGAI) framework[4], the agents given in the package `controller` are copied from GVGAI. This means that the agent you implement for the *spaceBattleGame* can be submitted to the GVGAI 2-player planning track :)

Remark: A *sampleGA* is provided, you can work on GA instead of RMHC or (1+1)-EA.

[Exercise] Implementation of AI agents.

1. Look at the agents provided in the package `controller` and play the game between human players or against an AI agent. You can use the `playOne` method in `GameTest.java`[5].

2. Implement an AI agent using rolling horizon RMHC or (1+1)-EA or any algorithm you want to use or your new designed one. *Hint: the genome length is the optimisation horizon.*

3. Include the *ShiftBuffer* to your agent.

4. Play against your new agent.

5. Submit your agent to the GVGAI 2-player planning competition. It's notable that no additonal work is required other than creating an account!

[4] `http://www.gvgai.net/`

[5] `https://github.com/ljialin/spaceBattleGame/blob/gameDesignII/src/test/GameTest.java`

1.3 Design new instances for *spaceBattleGame* using RMHC and (1+1)-EA

The game parameters of the *spaceBattleGame* are listed in `Constants.java` [6]. We are going to vary the parameters and create new game instances, like what Simon has demoed using *FlappyBird* (though we don't have the version in JavaScript)!

[Exercise] Game design using RMHC and (1+1)-EA.

1. Look at the agents provided in the package `controller` and play the game between human players or against an AI agent. You can use the `playOne` method in `GameTest.java`[7].

2. Edit some parameters (e.g., cooldown time, speed, radius, kill award) manually and play the game to see how different/hard/easy/strange the new game instance could be.

3. Read the code `SpaceBattleGameSearchSpace.java`[8]. Add the parameters you want to search and optimise, and the possible values for each of the parameters. The method `playNWithParams` in `GameDesign.java`[9] should be edited as well. An example has been given in the code below:

```
// SpaceBattleGameSearchSpace.java
static int[][] values = {
  { 4, 6, 8, 10}, // SHIP_MAX_SPEED
  { 1, 2, 3, 4, 5},  // THRUST_SPEED
  { 0, 1, 5, 10, 20, 50, 75, 100}, // MISSILE_COST
  { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10}, // MISSILE_MAX_SPEED
  { 1, 2, 3, 4, 5, 6, 7, 8, 9}, // MISSILE_COOLDOWN
  { 10, 20, 30, 40, 50} // SHIP_RADIUS
};
```

```
// GameDesign.java
public static double[] playNWithParams(int ai1, int ai2, int[] params,
    int resample) {
  Constants.SHIP_MAX_SPEED = params[0];
  Constants.THRUST_SPEED = params[1];
  Constants.MISSILE_COST = params[2];
  Constants.MISSILE_MAX_SPEED = params[3];
  Constants.MISSILE_COOLDOWN = params[4];
  Constants.SHIP_RADIUS = params[5];

  double[] res = GameTest.playNAndMean(resample, ai1, ai2);
  System.out.println(t);
  return res;
}
```

   emphPlease feel free to edit the code and make it more reasonable!

4. Run `RMHCTest.java` to search in the deisgn space and optimise the game instance [10]. Can you tell the fitness used in this code for evaluation and selection?

5. Can you change the solver to (1+1)-EA?

[6] https://github.com/ljialin/spaceBattleGame/blob/gameDesignII/src/ontology/Constants.java
[7] https://github.com/ljialin/spaceBattleGame/blob/gameDesignII/src/test/GameTest.java
[8] https://github.com/ljialin/spaceBattleGame/blob/gameDesignII/src/test/SpaceBattleGameSearchSpace.java
[9] https://github.com/ljialin/spaceBattleGame/blob/gameDesignII/src/test/GameDesign.java
[10] https://github.com/ljialin/spaceBattleGame/blob/gameDesignII/src/test/RMHCTest.java

MCTS

2.1 How to run this

- The MCTS agent to be completed is located in the Java package: `controllers.labMCTS;`
- The game entry point is at `test.GameTest.java`.
- Look for the **TODO: 16/05/17** comments in the code (or, in *IntellijIDEA*: View → Tool Windows → TODO).

2.2 Informative Read

The first set of TODOs are simply for you to familiarize yourself with the code. Follow TODOs from 1 to 4, read the code and ask questions if you're unsure of anything:

- TODO #1: controllers.labMCTS.Agent.java; Analyze the main class of the Agent, with the methods that are used to initialize the agent and to supply an action at every frame.
- TODO #2: controllers.labMCTS.SingleMCTSPlayer.java; Analyze the starting point of MCTS: the methods that start the algorithm and run it at every tick.
- TODO #3: controllers.labMCTS.SingleTreeNode.java: Have a look at the overall class, attributes and methods.
- TODO #4: controllers.labMCTS.SingleTreeNode.mctsSearch(); Method that contains a loop that performs the iterations of MCTS.

2.3 Building MCTS

In this set of TODOs, you'll implement the main functions of MCTS:

- TODO #5: controllers.labMCTS.SingleTreeNode.treePolicy(); Implement the tree policy. You need to determine at each step if you need to expand() or use uct(). Note that there are functions that do these things, so you only need to implement when to call these.

```
1  /**
2   * Executes the tree policy from a state received as parameter.
3   * @param state state to start running from
4   * @return A new node added to the tree, in the Expand phase of MCTS,
5   * or a node already in the tree if a terminal state was reached.
6   */
7  public SingleTreeNode treePolicy(StateObservationMulti state) {
8
9    SingleTreeNode cur = this;
10
11   //Keep going down the tree until depth reached or game is over
12   while (!state.isGameOver() && cur.m_depth < Agent.ROLLOUT_DEPTH)
13   {
14       //TODO: 16/05/17 Exercise Lab MCTS (5): Tree policy: Determine
             if you need to EXPAND or Navigate the tree using UCT
15
16       //...
17   }
18   return cur;
19 }
```

- TODO #6: controllers.labMCTS.SingleTreeNode.uct(); Build the UCB1 value for each node in the UCT method. You need to provide the calculations for the exploration and exploitation terms.

```
/**
  * UCT policy of the MCTS algorithm. It requires a state to choose an
      action from.
  * @param state state to apply UCT in
  * @return The selected node.
  */
 public SingleTreeNode uct(StateObservationMulti state) {

   SingleTreeNode selected = null;
   double bestValue = -Double.MAX_VALUE;
   for (SingleTreeNode child : this.children)
   {
       //TODO: 16/05/17 Exercise Lab MCTS (6): Build the UCB1 value
          for each one of these children.
       // 1. EXPLOITATION term: average of rewards, normalized within
          the bounds so it's in [0,1]
       double exploitation = 0.0; // --- FILL HERE ---
       exploitation = Utils.normalise(exploitation, bounds[0], bounds
          [1]);

       //2. EXPLORATION term: to value highly states rarely visited
       double exploration = 0.0; // --- FILL HERE ---

       //UCB1 Equation build up.
       double uctValue = exploitation + Agent.K * exploration;

       [...]
}
```

- TODO #7: controllers.labMCTS.SingleTreeNode.rollout(); Implement the random moves that happen at each step during the rollout phase.

```
/**
  * Rollout phase of MCTS
  * @param state state to start the rollout from.
  * @return the Value (aka reward, fitness) of the state reached at the
      end of the rollout.
  */
 public double rollOut(StateObservationMulti state)
 {
   int thisDepth = this.m_depth;

   //Check that we don't have to finish the rollout.
   while (!finishRollout(state,thisDepth)) {

     //TODO: 16/05/17 Exercise Lab MCTS (7): Execute one random move
        forward (note that the opponent(s) has to move as well!)
     //1. Random move for all players
     // ...

     //2. Roll the state forward
     // ...

     [...]
}
```

2.4 Improving your agent

In this final set of exercises, you'll improve MCTS adding information to the value function and the random rollouts. The objective is that your agent behaves more to your liking (this could be a better player, or following some specific behaviour you want to obtain).

- TODO #8: controllers.labMCTS.SingleTreeNode.myHeuristicScore(); Improve the evaluation of the final state so the agent behaves more to your liking.

```
1   /**
2    * Heuristic for a score on steroids. Do we want something else apart
         from the score?
3    * @param a_gameState state to evaluate.
4    * @return Value for this state
5    */
6   public double myHeuristicScore(StateObservationMulti a_gameState) {
7       //This is the normal game score
8       double rawScore = a_gameState.getGameScore(Agent.id);
9
10      //TODO: 16/05/17 Exercise Lab MCTS (8): Open: what would you add
            to the score to make your AI stronger?
11      /// rawScore += ?;
12
13      return rawScore;
14  }
```

- TODO #9: controllers.labMCTS.SingleTreeNode.rollOut(); How could you modify the rollouts, so instead of performing random moves, they are biased towards more interesting parts of the search space?

```
1  /**
2  * Rollout phase of MCTS
3  * @param state state to start the rollout from.
4  * @return the Value (aka reward, fitness) of the state reached at the
       end of the rollout.
5  */
6  public double rollOut(StateObservationMulti state)
7  {
8    int thisDepth = this.m_depth;
9
10   //Check that we don't have to finish the rollout.
11   while (!finishRollout(state,thisDepth)) {
12
13     [...]
14
15     //TODO: 16/05/17 Exercise Lab MCTS (9): Can you think of a way to
         bias the rollouts so MCTS explores other parts of the search
         space?
16
17     thisDepth++;
18   }
19
20   //Reached the end of the rollout, we need to retrieve the value of
       the state.
21   double delta = value(state);
22
23   //Update the bounds with the newly seen reward
24   if(delta < bounds[0])
25     bounds[0] = delta;
26   if(delta > bounds[1])
27     bounds[1] = delta;
28
29   return delta;
30 }
```

- Open TODO #10: Can you think of other places in the code where you could modify the agent's behaviour?