Multi-Agent Reinforcement Learning to Watershed Management

Final Report

Integrated Master in Informatics and Computing Engineering

Agents and Distributed Artificial Intelligence

Group 06_1:
Alexandre Ribeiro - up201205024@fe.up.pt
João Loureiro - ei08101@fe.up.pt
José Mendes - up201200647@fe.up.pt

Faculdade de Engenharia da Universidade do Porto
Rua Roberto Frias, sn, 4200-465 Porto, Portugal

December 10, 2017

Contents

1 Objectives

1.1 Scenario description

Water is an essential and limited resource, for that reason it's management must be efficient. Consider the existence of a river and a set of entities interested in using it: water supply of a city, irrigation of agricultural area (2 farms), power generation, fish protection ecologic zone.

The different entities interested in the use of water have separate goals that are mapped in different utility functions. Each entity is represented by an agent that controls the quantity of water to be retrieved from the river for it to use, in order to meet its goals and some restrictions (for instance, the minimum water level of the river after the water to supply the city being retrieved).

The utility functions that will be used will always depend on how the various agents are disposed in the field. In order to restrict the scope of this project a specific layout for the problem was chosen as the diagram from the figure 1 illustrates.

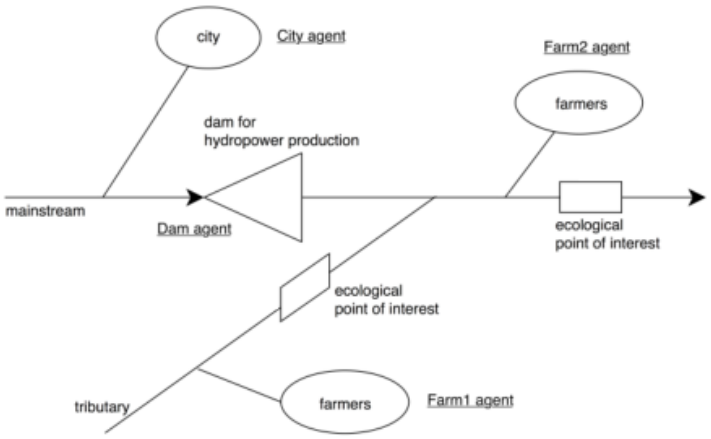

Figure 1: Watershed schema.

1.2 Work objectives

The Watershed problem is a resource management problem which consists of multiple agents. Each of these agents are withdrawing water from a common and finite supply for their own purposes.

The main goal is to implement a multi-agent reinforcement learning system that must be able to regulate the water used by all the interested entities, insuring that the balance between the objectives/interests of all the agents will not collide with any of the restrictions applied.

$$P(x) = \sum_{i=1}^{N} b_i(x_i) - \sum_{j=1}^{N} p_j(x_j) \tag{1}$$

In other words we will be maximizing a function $P(x)$ that represents the overall performance of the system, *i.e.* the sum of all the benefits $b_i(x_i)$ that each individual agent obtains by withdrawing water from the river minus a penalty $p_j(x_j)$ for when the agents violate the constraints.

2 Specification

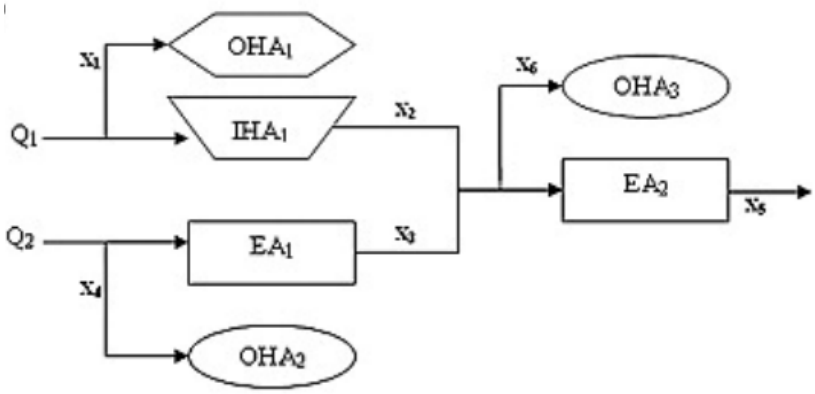

Figure 2: Watershed logical view from the research of Yang *et al* [1].

The Watershed problem that is represented on the figure 2 has multiple goals and constraints, and consists of continuous variables, *i.e.* the water that can be withdrawn by the river by any of the interested parties has an infinite number of possibilities. Each scenario of this problem will depend on three initial parameters, the monthly mainstream(Q_1) and tributary(Q_2) inflows and the dam capacity(S).

There are six variables that must be optimized for this particular example, four variables are controlled directly and two of them are reactive and indirectly optimized. The four direct variables are the water withdrawn by the city(x_1), by the dam(x_2) and by the farms($x4$ and $x6$). The two reactive variables are the water available to the ecosystems($x3$ and $x5$).

Since the reactive variables, $x3$ and $x5$, are not directly controlled they must be calculated using the following equations:

$$x_3 = Q_2 - x_4 \tag{2}$$

$$x_5 = x_2 + x_3 - x_6 \tag{3}$$

To each of these variables it must be assigned an objective function representing the benefits that an agent gets for withdrawing the water. The objective functions that will be used for every variable are shown in the following equations.

$$f_1(x_1) = a_1x_1^2 + b_1x_1 + c_1 \tag{4}$$

$$f_2(x_3) = a_2x_2^2 + b_2x_2 + c_2 \tag{5}$$

$$f_3(x_3) = a_3x_3^2 + b_3x_3 + c_3 \tag{6}$$

$$f_4(x_4) = a_4x_4^2 + b_1x_4 + c_4 \tag{7}$$

$$f_5(x_5) = a_5x_5^2 + b_5x_5 + c_5 \tag{8}$$

$$f_6(x_6) = a_6x_6^2 + b_6x_6 + c_6 \tag{9}$$

The a_i, b_i and c_i in the above equations are dimensionless constants[1] and the following table has the values their values along with the values for α_i which represents the minimum for the variables that will be optimized(x_i).

Parameter	Value	Parameter	Value	Parameter	Value	Parameter	Value
a_1	-0.20	b_1	6	c_1	-5	α_1	12
a_2	-0.06	b_2	2.5	c_2	0	α_2	10
a_3	-0.29	b_3	6.28	c_3	-3	α_3	8
a_4	-0.13	b_4	6	c_4	-6	α_4	6
a_5	-0.056	b_5	3.74	c_5	-23	α_5	15
a_6	-0.15	b_6	7.6	c_6	-15	α_6	10

Table 1: Watershed constants

In addition to minimum value of each variable we also need to take into account the following constraints.

$$\alpha_1 - x_1 \leq 0 \tag{10}$$

$$\alpha_2 - Q_1 + x_1 \leq 0 \tag{11}$$

$$x_2 - S - Q_1 + x_1 \leq 0 \tag{12}$$

$$\alpha_4 - x_3 \leq 0 \tag{13}$$

$$\alpha_3 - x_4 \leq 0 \tag{14}$$

$$\alpha_4 - Q_2 + x_4 \leq 0 \tag{15}$$

$$\alpha_6 - x_5 \leq 0 \tag{16}$$

$$\alpha_5 - x_6 \leq 0 \tag{17}$$

$$\alpha_6 - x_2 - x_3 + x_6 \leq 0 \tag{18}$$

The Watershed problem can be formulated as the optimization of the next equation subject to the constraints highlighted above.

$$F(x) = \max \sum_{i=1}^{6} f_i(x_i) \tag{19}$$

2.1 Identification and characterization of the agents

By means of the purposed formulation of the problem, we identify four agents, each one representing a different entity interested in the use of the water, but only three kinds of agents. Thus, there will be a *CityAgent*, a *DamAgent*, a *FarmAgent*, that must be replicated to regulate the two existing farms.

2.1.1 Architecture

There is a set of core classes that allow the problem modeling:

- *EnvironmentAgent*: implements a CyclicBehavior responsible for providing the data that the agents will perceive and regulates the actions that each agent must take.
- *CityAgent*: implements a SimpleBehavior responsible for controlling the withdraw flow of the city.
- *DamAgent*: implements a SimpleBehavior responsible for controlling the withdraw flow of the dam.
- *FarmAgent*: implements a SimpleBehavior responsible for controlling the withdraw flow of the farms.
- *SuperAgent*: responsible for implementing the reinforcement learning *Q-Learning* algorithm, that the agents will use as behavior.
- *Watershed*: the model that will wrap all the entities above referred.

The figure 3 helps to understand the inheritance structure of the agents.

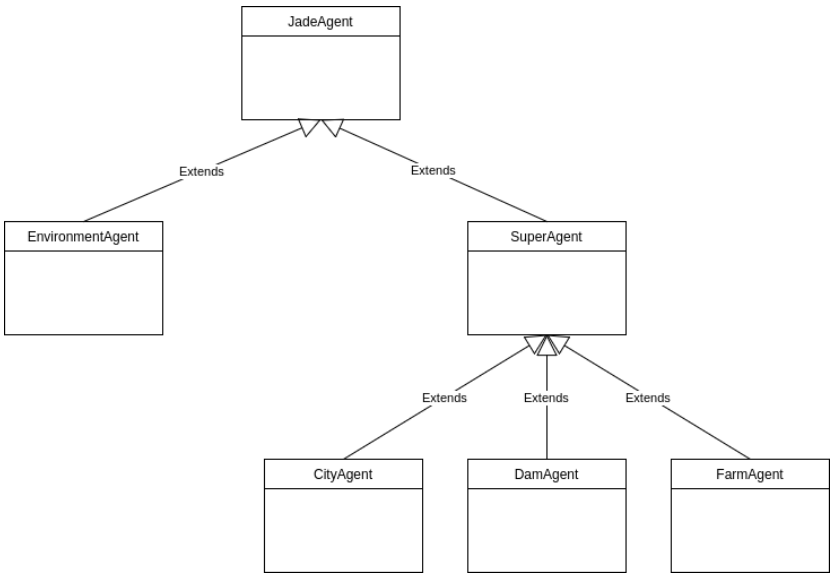

Figure 3: Agents architecture UML.

2.1.2 Behavior

Reinforcement Learning is an area of Machine Learning research, in which autonomous software agents have the capability to learn through trial and error by interacting with the environment. The agent learns to execute a given action using a process of reward/punishment linked to the best/worse results when searching for a goal.

The agents that withdraw water from the river will implement a well known reinforcement learning algorithm, the *Q-Learning*, as a behavior.

Q-Learning is an off-policy, model-free reinforcement learning algorithm that has been proven to converge to the optimum action-values with probability 1, so long as all actions are repeatedly sampled in all states and the possible actions which the agent may take are discrete [2].

The algorithm works as follows:

```
Initialize Q(s, a) arbitrarily
Repeat (for each episode):
   Initialize s
   Repeat (for each step of episode):
      Choose a from s using policy derived from Q
         (e.g., ε-greedy)
      Take action a, observe r, s'
      Q(s, a) <-- Q(s, a) + α [r + γ max_α,Q(s', a') - Q(s, a)]
      s <-- s';
   until s is terminal
```

Figure 4: Q-Learning pseudocode.

As shown on the figure 4, the algorithm first arbitrarily initializes the Q values of the Q table ($Q(s, a)$), which contains the values for the state and actions pairs.

Then in each episode it selects an action using a strategy, for this particular problem it was used a time decreasing ϵ-greedy strategy, where a random action is selected with probability ϵ, and the highest valued action is selected with probability $1-\epsilon$. The value of ϵ decreases by 0.99ϵ at each iteration. This will cause the agents to converge on a solution.

After selecting the action(a) and observing what the reward(r) and the next state(s') are, the algorithm updates its Q values using the following equation:

$$Q(s, a) \rightarrow Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)] \tag{20}$$

where $\alpha \in [0, 1]$ is the learning rate and $\gamma \in [0, 1]$ is the discount factor.

2.2 Interaction protocols

The interaction between the environment agent and the other agents is done through the *ACLMessage*, a class implemented in *JADE*.

The *EnvironmentAgent* will collect the information about the actions performed by all of the other agents and retrieve to each of them the respective reward and the next state that it must perform.

3 Development

3.1 Platform and development environment

JADE

Java Agent DEvelopment Framework is a software Framework that simplifies the implementation of multi-agent systems through a middle-ware that complies with the Foundation for Intelligent Physical Agents specifications. It is a distributed agents platform, in which, each agent only acts within its own domain, although they can be cooperative.

In this project we used some of the abstractions provided by the JADE API such as the Agent class, the SimpleBehaviour class, the CyclicBehaviour class and the peer-to-peer communication, based on asynchronous messages, implemented by the ACLMessage class, which allows the communication between the various agents that compose the Multi-Agent System.

Java Swing

Swing is a GUI widget toolkit for Java. We used it to provide a more friendly interface to configure the simulation parameters as well as show the actions performed by the agents in real time. Basically it allows a graphical view of the model simulation.

3.2 Application structure

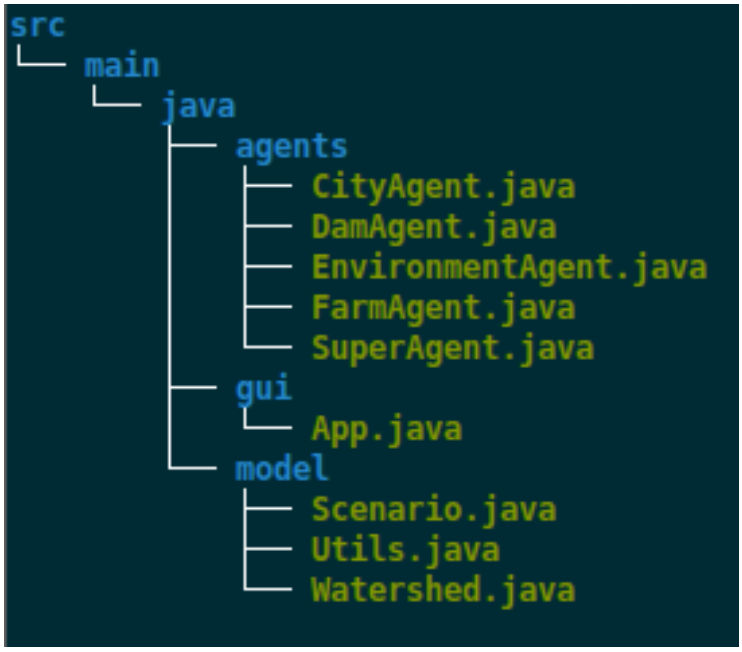

Figure 5: Application structure.

As shown in the figure 5, the application is structured into three packages:

- **Agents**
  - This package was already described above in the architecture section.
- **Model**
  - Watershed: the model that wraps all the elements of the project and builds the logical simulation.
  - Scenario: contains some preseted values that will be passed to the model, as well as the ability to create new ones.
  - Utils: contains some static fields, used across the application.
- **GUI**
  - App: builds a graphical user interface on top of the Watershed logical model.

4 Experiments

4.1 Objectives

4.2 Results

5 Conclusions

5.1 Experiments results analysis

5.2 Developed work and applicability of MAS to the proposed scenario

The proposed problem could be easily transposed to software since it is well described by two related papers.

The Watershed management is indeed a well suited problem for the application of a Multi-Agent System because, there is a set of distributed points with interest in the use of the water and that use should be extremely well prudent given that water is a vital and limited resource.

6 Improvements

6.1 Proposed program improvements

7 Resources

7.1 Software

The project development environment consisted of:

- Ubuntu 16.04 LTS
- JetBrains IntelliJ IDEA
- JADE (Java Agent DEvelopment Framework)
- Gradle Build Tool

7.2 Group effort

- Alexandre Ribeiro - 40%
- João Loureiro - 40%
- José Mendes - 20%

8 Appendix

8.1 User manual

To launch the program simply run *gradle run* on a terminal inside the provided source code folder.

A Graphical User Interface will be launched and allows to configure some model parameters like selecting one of the preseted scenarios, the number of episodes of the simulation, the agents learning rate and discount factor. When all set, click the launch button and the simulation will start running, showing the amount of withdrawn water of each agent.

References

[1] Francesco Amigoni, Andrea Castelletti, and Matteo Giuliani. Modeling the management of water resources systems using multi-objective dcops. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*, AAMAS '15, pages 821–829, Richland, SC, 2015. International Foundation for Autonomous Agents and Multiagent Systems.

[2] Karl Mason, Patrick Mannion, Jim Duggan, and Enda Howley. Applying multi-agent reinforcement learning to watershed management. 05 2016.