Artificial Intelligence - Multi Agent Systems

Albani Dario*, Nardi Daniele

Practical Exercise

As in precision agriculture, the world is a sown field that has to be managed autonomously by a set of agents. The tasks are different: initially agents have to explore the field and to map weeds that can be found in the field. Then, given the position of the weeds, the agents have to seek and destroy the weeds using a specific tool.

The assignment consists in choosing one problem (between field exploration and weed removal) and one cooperation technique (among Swarm Intelligence (SI), CNP or DCOP) and developing a working implementation in JAVA using the given simulator. The implementation should be tested with varying initial condition and environment sizes.

Optional: if you work in a group, each member of the group should solve the chosen exercise with a different method (e.g. DCOP and Swarm or CNP and DCOP) and compare the approaches.

The discussion will take place at the oral exam and consists in showing the running implementation and reporting on the performances of the proposed approach.

*albani@dis.uniroma1.it

JAVA Simulator

In order to accomplish the homework you have to use the simulator enclosed to this file. Below we provide a brief introduction to the simulator.

NOTE: any change has to be documented and accurately explained.

Compiling and Running

To run the simulator, after you have implemented the behaviour of the agents, you should follow the following three steps:

- Open a terminal and reach the folder where the package multiagent is contained.
- Compile the file WorldPanel.java by typing:
  javac multiagent/WorldPanel.java
- Run the code:
  java multiagent.WorldPanel

Documentation

Every function and class is documented and several comments are also present within specific functions.

For further information and details regarding the simulator you can refer to the documentation. Just open the doc folder and the index.html that is available in it.

How it Works

The entry point of the simulator is the class WorldPanel. This class is used to graphically display the state of the world. The world is represented as a grid containing white-coloured cells (i.e. empty cells) and green-coloured cells (cells that contain weeds). You can add or modify colors to display different cell's status; refer to the method **paintComponent(Graphics g)**.

Once any button is pressed, the simulation starts and the method **actionPerformed(ActionEvent evt)** is executed. This last method calls repeatedly calls the class AgentSim with the action **doOneStep()** until all the tasks are eventually completed.

World Description

You can consider the field as a grid world of $n \cdot m$ cells, with n and m not necessarily equal. Each agent has its own local representation of the world. A cell has the following properties:

- **visited**: True, False. True if the cell has been visited by at least one agent.
- **weed**: True, False. True if there are weeds in the cell.
- **sprayed**: True, False. True if the agent performed the spraying of the herbicidal at the current location.

According to the task that you will chose, weed's locations are initially unknown or known or (see section 1, 2 or 3 for further specification).

At each time step the world class queries each agent with a polling strategy, according to a predefined, fixed order. For each agent the world:

1. requests the next action to be executed
2. executes the requested action and returns the corresponding outcome
3. ask the agent which task it is performing

For the set up of **Coverage Problem** and the **Weed Removal Problem** two already defined methods are given.

For the **Coverage Problem** you have to ignore the variable sprayed of the cells. All the cells are initially set as not visited (i.e. visited = False). Weed's locations are not known and their spread does not follow any particular distribution. Agents start from random initial cells. A task is identified as a non explored cell.
For the **Weed Removal Problem** you have to consider the variable sprayed. All the cells are initially set as visited and not sprayed. Weeds' locations are known and their spread does not follow any particular distribution. Agents start from random initial cells. A task is identified as a non sprayed cell.

Agents

An agent is an unmanned aerial vehicle (UAV). Agents present an internal system for collision avoidance (i.e. you do not have to worry about collisions).

Agents' communication protocol

You can assume that the agents have internal antennas and a wireless systems, thus are able to communicate. The power of such devices is enough to guarantee a full connection among the agents.

Agents' Actions

The actions to be performed are different with respect to the task, hence you will have different implementation.

1) action **NoOp**. No action is performed and the agent continue to hover over the current location.

2) action **moveToLocation(Task t, Agent a)** moves the agent a to cell c referenced by the task t. Once c is reached the agent has to perform one between the action Spray or Check in order to complete the assigned task. Note that only the last cell is updated because UAVs are able to scan or to spray a cell only while hoovering over it and not during motion. You can use the World class to retrieve the information about the visited cell.

3) action **publishNextTask(Agent a)** return target Task t associated with a target Cell c. The target task is the task that has to be executed by the agent in the next step (e.g. explore or spray the herbicidal). This action represents the core logic of the exercise (notice that the task under execution is communicated to the world mainly for visualization purposes, while the decision of who is doing what depends on the specific method chosen).

4) action **spray(Task T, agent a)**: Sprays the herbicidal on the cell C associated with task T. The action will result in the update of cell's and task's information and returns the updated task: the task assigned to the cell is marked as done and cells variables are updated.

5) action **check(Task T, agent a)**: Scans the cell C associated with the task T. The action will result in the update of the cell's and task's information and returns the updated task: the task assigned to the cell is marked as done and cells variables are updated.

Problem

Coverage Problem In order to accomplish the task UAVs have to explore the whole field; i.e. every cell in the field has been visited, no remaining pending tasks are present.

Weed Removal Problem In order to accomplish the task UAVs have to spray the herbicidal on every weed; i.e. every cell in the field has been visited and no weeds are left, hence no remaining pending tasks are present.

1 Swarm Intelligence

For either the Coverage or the Weed Removal problem :

1) Deliver a working implementation of a swarm intelligence approach with the simulator in which you implement your own coverage or weed removal solution.

2) Estimate a performance measure for your approach and discuss possible solution and/or improvements. Some comparison parameters can be the time needed to explore the field, robustness, the number of visits for each cell and possibly other.

Some tips

- For every reachable cell in the, otherwise local map, compute the probability of being chosen for that cell by defining a cell selection function. When dealing with the coverage problem, you can define it as a variation of the Random Walk strategy. For the weeds removal problem you already know the location of the weeds.
- Remember to normalize overall cells probabilities to one and to extract one Cell c and its related task randomly according to the pre-computed probabilities. E.g. Imagine a 3-cells grid world. After some computations cells have values. [1,2,1] normalized values are thus [.25,.5,.25]. Cell two has double the probability to be chosen than cell one and cell two.

2 Contract Net Protocol and Sequential Auctions

For either the Coverage or the Weed Removal problem :

1) Deliver a working implementation of both CNP and Sequential Action with the simulator in which you implement your own coverage or weed removal solution.

2)Estimate a performance measure for your approach and discuss possible solution and/or improvements. Some comparison parameters can be the time needed to explore the field, robustness, the number of visits for each cell and possibly other.

Some Tips

For CNP an Combinatorial Auctions, the next Task is the next cell that has not been treated or visited in the grid. You should skip already sprayed or visited cells. Your implementation will be a sequence of tasks allocated from the list of pending tasks in the world class. You can define different distance metrics (e.g. Manhattan distance).

3 Distributed Constraint Optimization Protocol

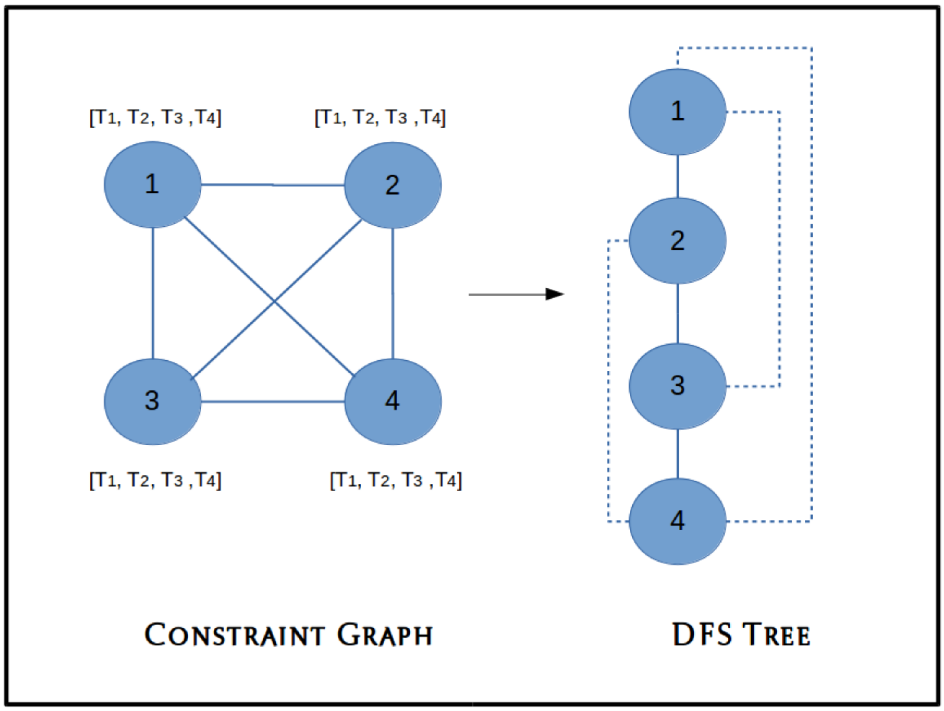
For either the Coverage or the Weed Removal problem:

1) Deliver a working implementation of DPOP with the simulator in which you implement your own coverage or weed removal solution.

2) Estimate a performance measure for your approach and discuss possible solution and/or improvements. Some comparison parameters can be the time needed to explore the field, robustness, the number of visits for each cell and possibly other.

Some Tips

To simplify the DPOP problem you can assume that all the computations are made by a central unit (i.e. the world or a central agent) and that all the value function are sent via broadcast at the end. For the DPOP one has to build a constraint graph where: the variables represent the tasks, and there is one agent per node. Assuming that the each agent can take only a task, the constraint graph will result in a fully connected structure. In order to simplify the problem, consider 4 tasks and 4 agents at a time and build the DFS tree arranging the visit according to the id of the agents (nodes). This will result in the same DFS tree as in the image below. Implement the assignment of the first 4 tasks to 4 agents Optional: try to generalize by grouping the tasks (in sets of four units and the agents in fixed teams of four agents).


[T1, T2, T3 ,T4]
[T1, T2, T3 ,T4]
1
2
3
4
[T1, T2, T3 ,T4]
[T1, T2, T3 ,T4]
1
2
3
4
CONSTRAINT GRAPH
DFS TREE