Assignment 3: Distributed Smart Camera Networks and negotiation mechanisms

CSE5022 Advanced Multi-Agent Systems

DDL: 23:59, Wednesday, May 29, 2024

1 Overview

For this assignment, you must implement concepts related to *Negotiation and Agreement* in Multi-Agent Systems concepts utilising the *Distributed Smart Camera Network* use case, proposed by the authors in [1], following up the activities carried out in the laboratory. This paper considers a *Socio-economic inspired* approach to efficiently managing operations in a smart camera network. *Handover* and a *Vickrey auction mechanism* are described to handle objects of interest, tracking and optimising camera resources.
You are required to use the open-source Repast Symphony Simulation Multi-agent Toolkit[1], extending the appropriate classes in *java*, provided to achieve the requirements described in Section 3.

2 Overview

- In this assignment, both *Negotiation* and *Agreement* concepts are explored, using a network of smart *cameras* managed autonomously by a software agent. The array of these devices is fixed, but *objects of interest* being tracked are mobile. These objects can have any representation; they can be cars on a road or even persons in an airport. For this simulation, a set of n **fixed cameras** are instantiated in the environment.

  Each camera i can communicate with neighbouring cameras and have a limit k objects to track, stored in a set O_i; these objects should be in its corresponding Field of View (FOV). For the simulation, you have various m objects randomly distributed to be tracked by n cameras.

- **Negotiation and Agreement:** When an object is leaving the FOV of camera i, it can enter a region where other cameras can have the ”right” to track that object. To *negotiate* the tracking responsibility, a *Vickrey auction mechanisms* is proposed.

[1] https://repast.github.io/

- Since the multi-agent system negotiation in this scenario depends on the topology and network setup, you will evaluate how different scenarios generate different results. Moreover, besides *Vickrey Auction*, you are required to also assess a different *Auction mechanisms* for the same application, such as *English or Dutch auctions*.

3 Requirements

1. **Distributed Smart Camera Networks**: For each camera aiming to track certain object *j*, you need to implement the following characteristics:

   - *Field of View (FOV_i)*: The objects to be tracked by a camera *i* should be *inside* a region where the camera can *view* it. This study describes how an FOV can track a set of objects, evaluate the camera's performance, and manage its resources efficiently. The FOV is represented as a triangle (Figure 1). You must decide how to implement this information in the camera .java class.
   - *Visibility (v_j)*: For a given object *j*, a visibility value is generated (between 0 and 1), and it is related to the maximum distance of the FOV and the relative object position. For example, if the object is near, the value should be almost 1 and close to 0 in the opposite case. Consider a simple approach to generate this value.
   - *Confidence (c_j)*: A value between 0 and 1, indicating how confident the camera is to track the object *j*. For a camera *j*, both values c_j and v_j are between 0 and 1 as soon as the observed object is within the FOV of a camera, 0 otherwise. Consider a simple approach to generate this value.
   - *Utility*: An accumulated utility should be maintained for each camera. This information will be needed to evaluate the performance of the network system. How to calculate the utility is described below:

2. **Utility and Market Mechanisms:** Each camera will manage a utility function described by the following equation:

$$U_i(O_i, p, r) = \sum_{j \in O_i} u_i(j) - p + r \tag{1}$$

$$= \sum_{j \in O_i} [c_j \cdot v_j \cdot \phi_i(j)] - p + r \tag{2}$$

   Where $\phi_i(j)$: $O_i \rightarrow 0$, 1 is 1 if camera *i* attempts to track object j and 0 otherwise. In addition to utility earned by tracking objects, a camera *b* may make a payment to another camera *s* to "buy" the right to track an object from that camera. This requires that the "selling" camera *s* already owns the object *j*. If an exchange is agreed upon, the object is removed from O_s and added to O_b. *p* denotes the sum

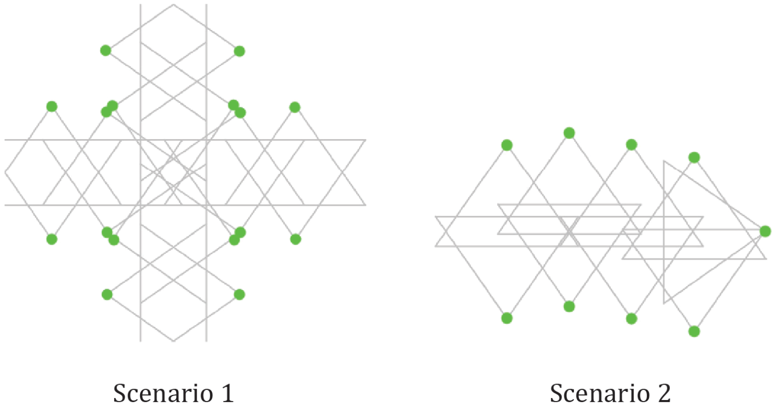

Figure 1: Fig. 1. Illustrations of the scenarios to be simulated. A green circle represents each camera, and its field of view is marked as a triangle.

of all payments (bids) made in trades in that iteration, and r conversely denotes the sum of all payments received.

The trade is facilitated by the **Vickrey auctions of second order**, implemented in *handoverEvent()* method, where the **buyer** camera offering the max bid or price is selected but pays the second max price in the auction. The bid is a value already implemented by the method *calculateBid()*. The utility for each camera is updated at every time step when a Handover is completed, implemented in *finalizeHandover()*

3. **Auctions mechanisms**: Since the Vickrey auction is not the only negotiation mechanism, you must implement a different mechanism, such as *English or Dutch Auctions* and evaluate the performance. Refer to [2] on how it varies from *Vickrey Auction of second order*.

4. **Simulations**: Once the methods required: FOV, utility and additional auctions are implemented, prepare 2 displays in Repast; each should correspond to 1 Scenario in Figure 1. The same amount of *Objects of interest* should be instantiated, but the number of cameras will change according to the scenario.

5. **Results:** Enable the **Data collection** process in the simulation interface [2] and compare the results obtained using *Vickrey auction* and the additional auction mechanism implemented. Each negotiation mechanism should be executed in different simulations to obtain separate results. Each scenario will evaluate results in a graph where the x-axis is **tick**, and the y-axis is **Total utility** for the number of cameras. An increasing curve is expected; however, how fast it increases will depend on the number of trades made.

[2] https://repast.github.io/docs/RepastJavaGettingStarted.pdf

4 Hints

- All classes and essential code needed to execute this simulation are provided. This assignment is an extension of the lab sessions. You may consider an extra *.java* class if necessary
- *Field of View (FOV)* can be seen as a partial region *sensed* by an *agent*. So far, we have considered full observability provided by the ***GridCellNgh()***, and ***getNeighborhood()*** methods. In this case, not all cells around the camera should be returned. Just the ones in the FOV.
- This assignment is partially inspired by the mechanisms described in [1]. Refer to sections 3 and 4.1 for more details on the problem formulation and aspects considered in Equations 1 and 2.

5 What to Submit

1. **A report in PDF format** describing each considered methodology for your simulation of the simulation. Illustrate your approach using screenshots and include as much detail as possible.
2. For each scenario, provide the requested charts described previously, your comments and reflections about the approach proposed; why are some scenarios more efficient than orders?
3. Considering the same amount of time steps (ticks), such as 1000. Which scenario is more efficient for the negotiation process?
4. **All source code files**, since it will be evaluated by executing the simulation during different time intervals.

Pack all files into `SID_NAME_A1.zip`, where `SID` is your student ID and `NAME` is your name (e.g., 11710106_ 张三 _A1.zip).

6 References

[1] Lukas Esterle et al. "Socio-economic vision graph generation and handover in distributed smart camera networks". In: ACM Transactions on Sensor Networks (TOSN) 10.2 (2014), pp. 1–24.

[2] Michael Wooldridge. An introduction to multiagent systems. John wiley & sons, 2009.