SAPIENZA
UNIVERSITÀ DI ROMA

Report on a project

for Artificial Intelligence course

Multi Agent Systems (MAS)

"Fire Control"

i) Task allocation strategy:
Contract Net Protocol (CNP) - Decentralized Contract Net Protocol

ii) Inside-task exploration strategy:
Random Walks (RW) - Selection of valid cells

Nazgul Mamasheva

i) Task allocation strategy: Contract Net Protocol (CNP) - Decentralized Contract Net Protocol

Decentralized CNP: many managers, where each manager with UAVs in communication range composes one entity. Each entity applies Centralized CNP for its auctions.

Let's assume there is a manager named K, it has two UAVs (KN_1, KN_2) in communication range. K can make bids and propose to itself without sending packets to itself.

K can make auctions in two ways (winner utilities are denoted by green):

Table 1: Auctions for each task one by one

Auctioneer K	Task_1	Task_2	Task_3
KN_1	0.3	0.8	0.06
KN_2	0.2	0.1	0.09
K	0.5	0.9	0.7
Winners	KN_2	KN_1	K

Table 2: Auctions for all tasks at once

Auctioneer K	Task_1	Task_2	Task_3
KN_1	0.3	0.8	0.06
KN_2	0.2	0.1	0.09
K	0.5	0.9	0.7
Winners	K	KN_2	KN_1

In the first scenario, the auctioneer will send bids for specific tasks to all bidders. Then the auctioneer will wait for all proposals for that task and will choose a winner. As shown in Table 1, for Task_1 the winner is KN_2, it has the smallest utility result. Then for the second auction, i.e. for Task_2 the bidder with the smallest utility (0.1) is KN_2, but this bidder already got a task, then the winner will be the next available bidder with smallest utility proposal, it is KN_1 with 0.8. And the last winner will be K itself.
The problem here is that KN_1 instead of moving to its nearest task Task_3 will move to the farthest task Task_2. And KN_2 instead of moving to its nearest task Task_3 will move to its farthest task Task_1.

The second scenario, Table 2 can be a good option to solve such a problem which was implemented in this program. The auctioneer will make auctions for all tasks at once. Then it will wait for all the proposals. After getting all proposals, it will choose the smallest utility among all proposals, here it is 0.06. The first winner is KN_1 for Task_3. The second smallest utility is 0.09, but the auctioneer can't assign two bidders to the same task. In this example, each task can get only one winner. So, the auctioneer will search for the next smallest utility and it is

0.1. The auctioneer will check if this task already has a winner, in this example - no, then the winner for Task_2 is KN_2. The next smallest utilities are 0.2 and 0.3, but their owners are already winners. The auctioneer will skip them and stop on 0.5, its owner is available, then the winner is K for Task_1.

As results show, the second option is a better choice. The utility values for two scenarios are the same, but results for two scenarios show us that winners can be completely different according to the chosen options.

Utility function: an UAV needs to calculate the cost of its performance according to a task. When an agent gets a bid, it will call this function to calculate the utility and send the result to the auctioneer as a proposal.

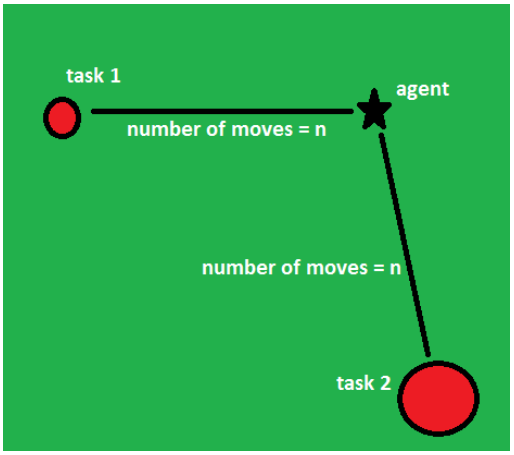
$$u(Agent, Task) = 0.7 * (numMoves / maxMoves) + 0.2 * (1 - taskSize / maxTaskSize) + 0.1 * (1 - taskRadius / maxTaskRadius)$$

0.7, 0.2 and 0.1 are weights, where $0.7 + 0.2 + 0.1 = 1.0$

$0 <= u(Agent, Task) <= 1$

The utility heavily depends on the number of moves from the agent's current position to the task's centroid position, therefore it weighs 0.7 out of 1.0.

Utility Function could consist of only a number of moves. But size and radius of task should also be taken into account.



In the scenario, illustrated on the left, the agent has to choose one of two tasks, where the number of moves from agent to two tasks are the same. Then we can see that decisive factors will be the task's size and radius. In such a situation, the agent should choose the task with a big size and large radius, since such a task (fire) is more dangerous, and it can ignite more neighbor cells.

Here we can see that size and radius are also important for calculating the utility of an agent to a specific task. Therefore, these two criteria are also included in utility function calculation.

Task priorities: this addresses the question of determining the optimal number of agents assigned to a task (fire), as specified in the project. The function sets a heuristic for determining the appropriate number of agents for each task.

First, we need to define the priority of each task according to its size and radius. The task with the largest size and radius has the highest priority and so on. Then it makes some calculation:

w = numUAVs / numTasks

(numUAVS - number of all UAVs)

r = numUAVs % numTasks

(numTasks - number of all Tasks)

Each task has to get a minimum number of UAVs, so each task will get “w” number of UAVs. Then the “r” tasks with highest priorities will get by “1” UAVs.

For example, numUAVS = 23, numTasks = 5.

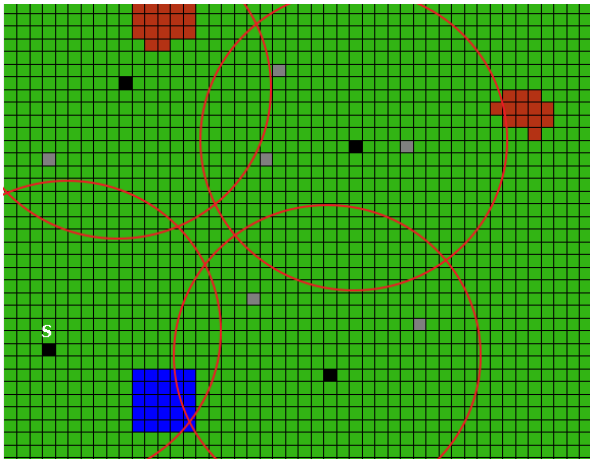
int w = 23 / 5 4

int r = 23 % 5 3

Tasks order (by priorities)	Task_1	Task_2	Task_3	Task_4	Task_5
Number of UAVS	4 + 1	4 + 1	4 + 1	4	4

Each manager will apply this function in order to assign the right number of UAVs to each task (fire).

Defining managers: when the program starts execution, it will first define managers. The managers are those agents (UAVs) with the highest count of neighboring agents in their area. Each agent will initially count the number of neighbor agents within its communication range. Subsequently, the program will compare the results and select the agents with the highest number of neighbors, assigning to them "manager” status. Agents listed under these managers cannot hold manager status in the same area. This process will iterate until no agents remain in the list.



If the agent doesn't have any neighbors within its communication range, it can be the manager in its area, because no other manager can send bids to this agent. In program code this kind of manager will be assigned with "_manager" status. It will bid and propose to itself.

Example for _manager with no neighbors, denoted by "S"

ii) Inside-task exploration strategy: Random Walks (RW) - Selection of valid cells

Cells selection: after choosing a task, UAVs will choose valid cells, which will increase efficiency of the extinguishing process. The centroid of a task will be the very first position of each agent. This position will be saved in the local knowledge of an agent. In the next step the agent will choose the nearest cell of the last saved cell in its local knowledge. It will be one of 8 positions.

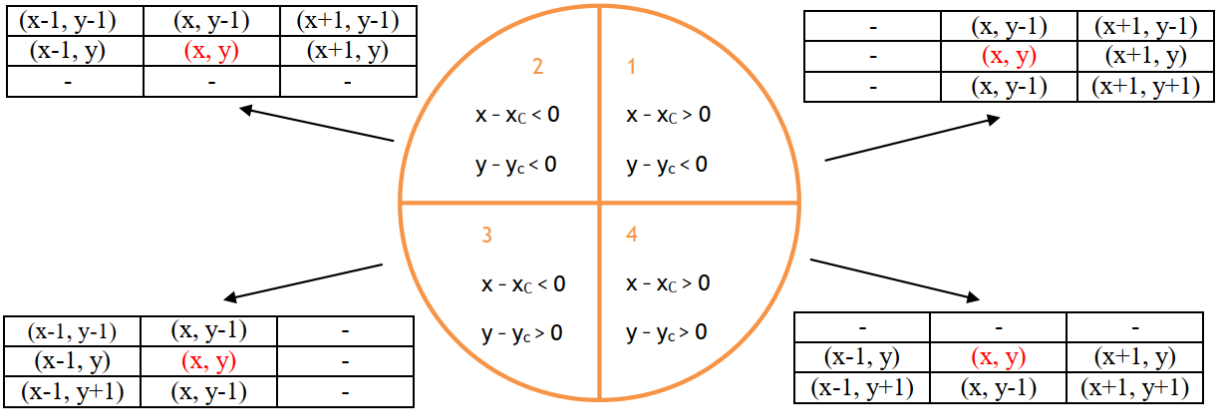
If agent is on position (x, y), then the coordinates of the nearest cells are as follows:

(x-1, y-1)	(x, y-1)	(x+1, y-1)
(x-1, y)	(x, y)	(x+1, y)
(x-1, y+1)	(x, y+1)	(x+1, y+1)

The agent will randomly choose one of 8 options. The program each time will check a new position for not being out of the boundary of a field and for being in the area of a task. If the nearest cells are already extinguished or normal, then the program will gradually increase the distance between last cell position and new cell position.

After a while, when cells will start burning out and the agent's next position will coincide with the burned cell, the agent will use another strategy.

The tasks (wildfires) propagate outward, covering neighboring areas from all its sides.



x: x coordination of new position
y: y coordination of new position

x_c: x coordination of centroid position
y_c: y coordination of centroid position

If the agent's new position is a burned out cell, then it will calculate $x - x_c$ and $y - y_c$. And then will define which quadrant it belongs to. In figure above, for the 1st quadrant the agent can choose only 5 nearby positions, i.e. the agent should move to the north-east side, but not to the west side. Because on the north-west side most probably cells nearby are already burned out and it will not make much sense for the agent to move there. For 2nd quadrant – to the north-west side, for 3rd quadrant – to the south-west and for 4th quadrant – to the south-east side.

Results evaluation:

Number of UAVs	Number of Tasks	Communication Range	Burned Cells
2	3	30	3233
3	3	30	3080
4	3	30	2753
8	3	30	2525
15	3	30	1708
5	2	30	2838
5	3	30	2818
5	4	30	2925
5	8	30	2894
5	15	30	2923
8	3	10	2670
8	3	20	2398
8	3	40	2417
8	3	50	2551
8	3	60	2709

The results also depend on UAVs' and tasks' positions, which are random each time. Therefore, the number of burned cells sometimes differed significantly for each combination of parameters and average results were chosen as demonstration results.

1. As we see, with a constant number of tasks (3) and with constant communication range (30), if the number of UAVs will increase, the number of burned cells will decrease.
2. With a constant number of UAVs (5) and with constant communication range (30), if the number of tasks will increase, the number of burned cells will overall increase a little bit.
3. With a constant number of UAVs (8) and constant number of tasks (3), if the radius of communication range will increase, the number of burned cells will vary.

So, according to the results, the number of burned or extinguished cells depends on the number of UAVs, rather than the communication range.