TEK5010

Fredrik Jaibeer Mahal Nordeng

September 2020

https://github.com/FredrikNM/Multi-Agent-Systems/tree/main/Random%20Agents

1 Abstract

In this paper I will make a environment where we implement a multi-agent system (MAS). We will see that, even tho we make very little hard coding in how they should move and interact, they still can be pretty efficient in trying to solve tasks we create.

2 Introduction

We are asked to create a square area where agents move around solving tasks they come over. The variables we consider here are :

Area of the environment (A) : How big/small (we have 1000*1000 in all our simulations)
Tasks (T) : How many of them
Task radius (Tr) : If agents inside this distance from task they will move towards it
Task capacity (Tc) : Agents needed to solve a task
Agents (R) : How many agents
Agents velocity (Rv) : Since we are iterating over time steps, this is the distance an agent can cover per iteration
Agents communtication radius (Rd) : Used for signaling task found
Time agents follows a signal (Rt) : In case task solved before an agent reaches signal, we use this to set agents back in to search modus
Call Off : Agents working can, instead of relying on Rt, send signal saying that the task is solved

3 One agent scenario

First we will implement task radius to 50 (as in meter/foot or what you want to think of it as), one agent moving around randomly, solving a task every time it is in the task radius. It moves around at a speed of 25 (meter/foot/etc) per iteration.

We are asked for a good model for moving around randomly. Lets consider the case of one agent. This is actually purely a definition of what moving randomly is. If we are to consider completely random, we are only left with a choice, a brownian motion at every time step. If we introduce memory, we can alternatively keep the direction until something is hit, task or a wall. Then our agent might only be steered by the shape of the walls, if we don't explicitly tell it to move randomly when hitting a wall, or it is done with a task. This is random in the sense that it depends on where you initialize your agent. As we see in the heatmap Fig. 15 it can actualy get stuck in a specific shape. We avoid this by telling the agent to move randomly after finishing a task or hitting a wall Fig. 15 .

The more you take away from a brownian motion the more systematically you can tell your agent to move. So lets rather see this from the perspective of how systematically can we actually move? This is very hard question. So lets imagine that our agent can not see, but it has memory of where it has been. It is also aware of it only being one task available to solve. If our agent do not have a map of the environment, we first realise that the agents vision in some sens is the radius of a task. Our agent should then ideally move in a bigger and bigger area around its origin, while the new vision is next to what it has seen before but never overlapping. One example would be a spiral, but with what shape it grows its area of seen space which is most efficient in an arbitrary shaped space is for the reader to look in to if they want.

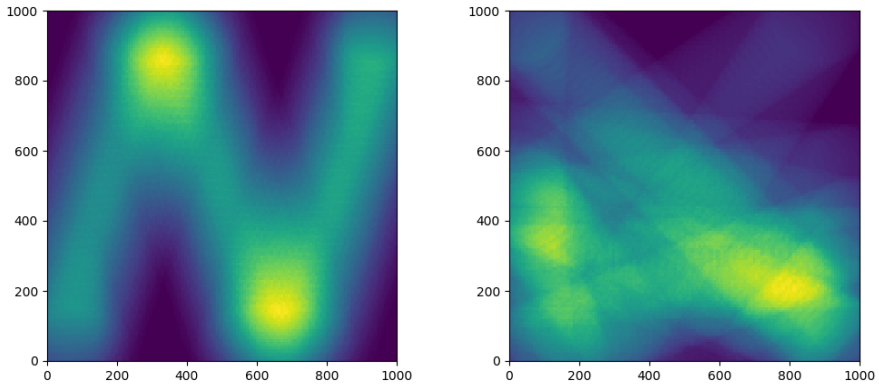

Figure 1: Left picture is agent steered by wall, getting stuck in a N shape. Right picture the agent move randomly after hitting a wall. In both pictures the area of heat is the radius of a task, since that is in a sense our agents vision area

Next thing we are asked is to plot how many tasks the agent finish on average at each time step. We also get the question if this a good measure in this search and task allocation STA problem.

To be fair, if our agent is moving around in a very large area, with only one task available do to at a time, this measure might not be very good. If we are

moving around not as a brownian motion, but with some kind of memory or something else setting the direction, we might want to know how much time the agent spend in the same area. Because if it is going around in a little circle, it is very bad, but if it on the other hand spend very little time in the same places, it is actually doing a very good search job event tho it might not solve any task because the environment is to big for it to find it. So my proposal for assessing how good an agent is doing is time spent in the same area. Worst case this measure could lead to creating agents that avoid spots it has been in and at the same time avoid tasks. In our case this would never happen as they are obligated to work when a task is found. So lets then imagine a agent that knows where tasks are spawned, and by that manages to avoid the task radius. We have then created the inverse of a perfect agent. In multi-agent systems this can be used for shepherding if agents are programmed to avoid collisions, since the inverse of an perfect agent occupy the spaces where tasks are not spawned.

4 Multi-agent scenario

From figure we see that after around 1000 iterations we usually get into a steady-state where the performance stay almost the same. Even tho as we will se later we might not even reach it after 4000 iterations. If we end up in steady it could still be a possibility that our system have several steady-states it could end up in. From the N in Fig. 15 we see how this could happen. To get a good measure of performance, for different amount of agents and task available at each time, we then would have to simulate each scenario such that we can take advantage of the Central Limit Theorem, which has a rule of thumb of n > 30. ref modern mathematical statistics with applications. The cases we see on with no communication among the agents could be considered the "random" benchmark for the STA problem, where we have touched on what really random. We will in the rest of our test use agents that keep their direction, and gets a new random direction when hitting wall, other agents, are finished with a task, or a signal they followed has been called off.

Several different scenarios have been simulated, and to make them comparable I have chosen to see it as an average off task done compared to how many task that is available.

$$\frac{Average\,task\,finished}{Available\,tasks} \tag{1}$$

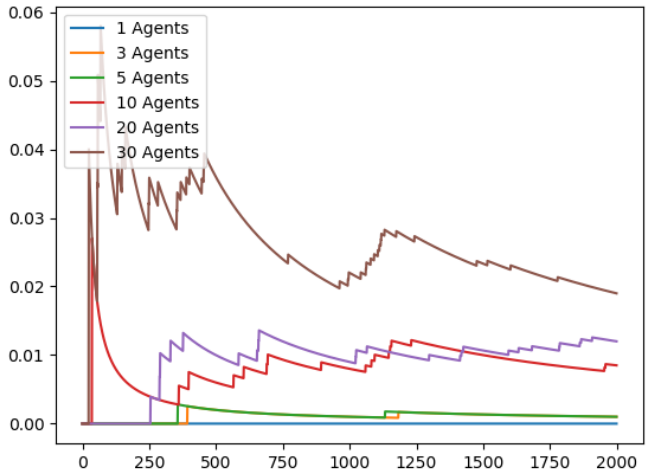

Figure 2: One agent needed to finish a task, and one task available at each time.

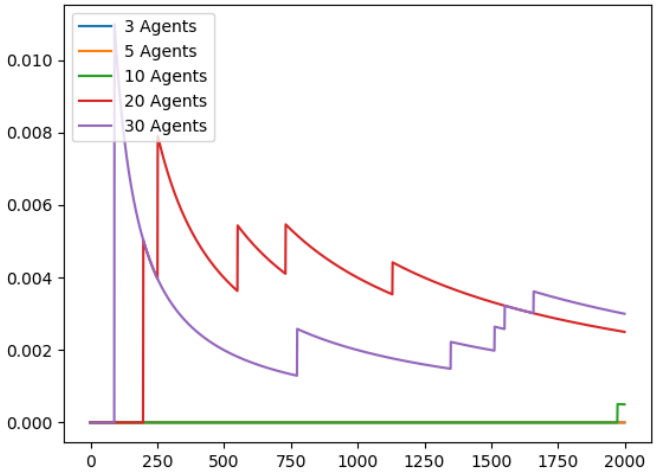

Figure 3: Three agent needed to finish a task, and one task available at each time. This will be our standard to check up against. Since more agents are need at each task we see the efficiency goes down from when only one agent was needed.

Lets see if adding more task will make it easier for the agents to find them.

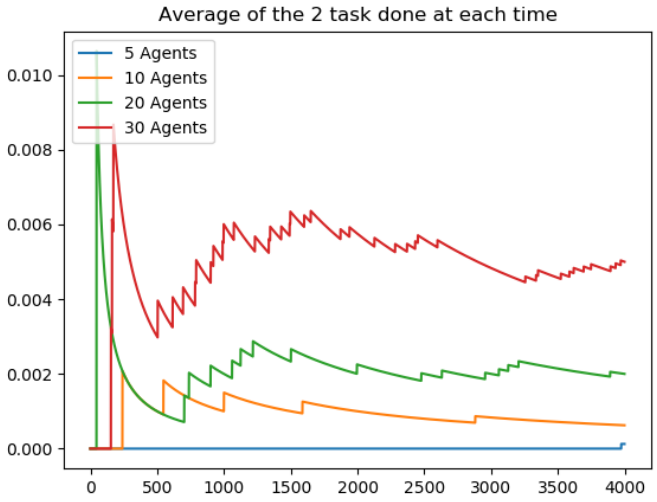

Figure 4: As we see efficiency goes a little bit up, but I also removed the scenario with three agents because it would be possible for 2 of the agents stuck at one of the task, with the last one waiting for help at the other task.

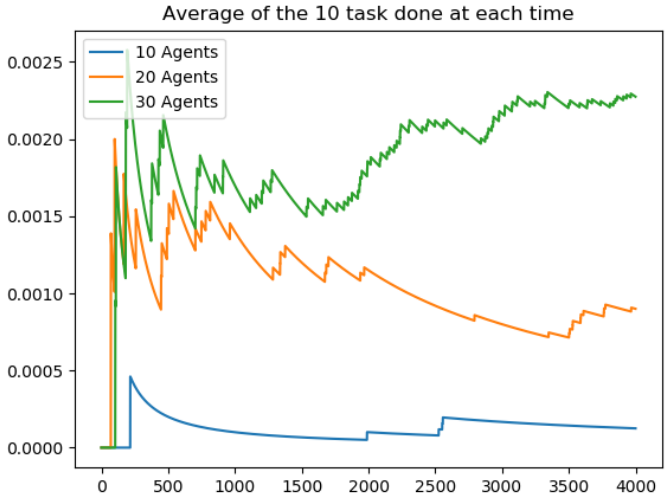

Figure 5: The efficiency is starting to fall as a consequence of agents waiting for help. Surprisingly the simulation with ten agents still have free agents at 2500 iterations.

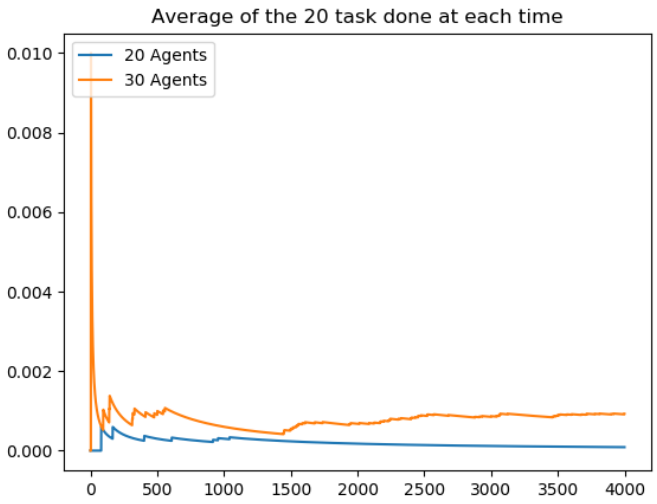

Figure 6: Here we see the waiting time for agents to come help each other become so large that even 20 and 30 agents almost gets nothing done. To many of them find different tasks.

For the last part of our assignment we implement communication between the agents, by sending a signal when they find a task. They send the signal only once and we simulate it with different radius which we then compare to simulation where they use call off. Call off is a new signal telling agents that has not reached the task before it is done that they can start searching for new tasks again. We have 30 agents,

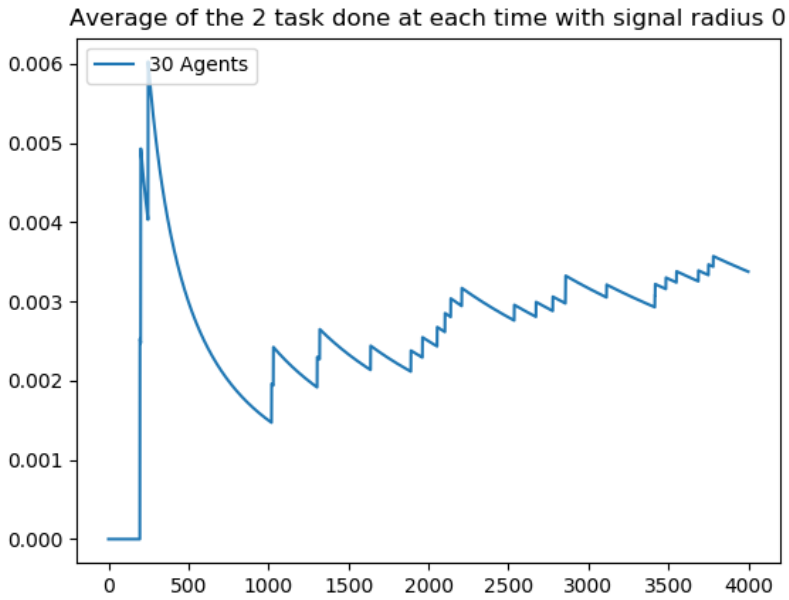

Figure 7: 30 agents with no signal and 2 tasks.

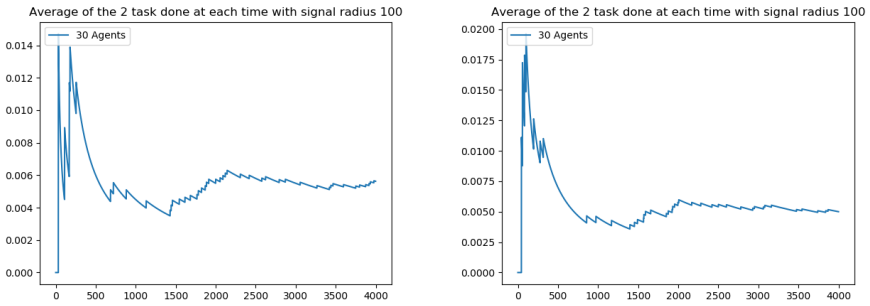

Figure 8: Left picture signal with out call off, right with call off.

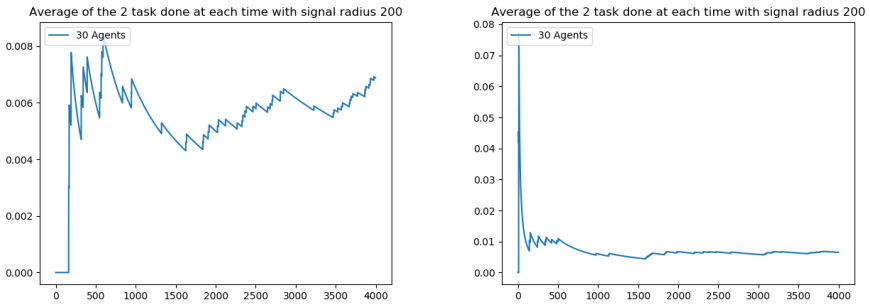

Figure 9: Left picture signal with out call off, right with call off.

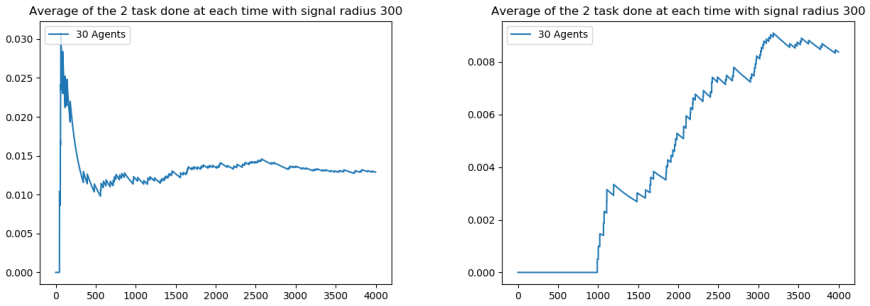

Figure 10: Left picture signal with out call off, right with call off. Here the call off simulation did not even reach steady state after 4000 iterations.

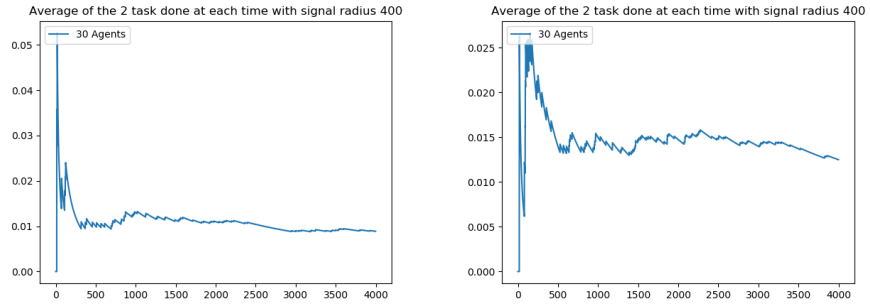

Figure 11: Left picture signal with out call off, right with call off. This is where we get our best result.

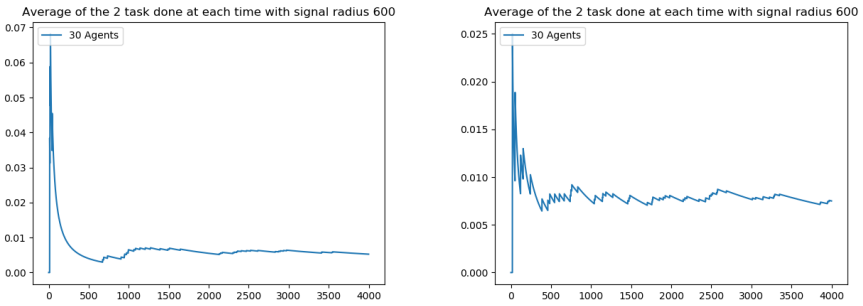

Figure 12: Left picture signal with out call off, right with call off. Our efficiency is starting to go down, due to the signal radius being so large that it is a nuisance and prevent agents from searching. The end up more clustered.

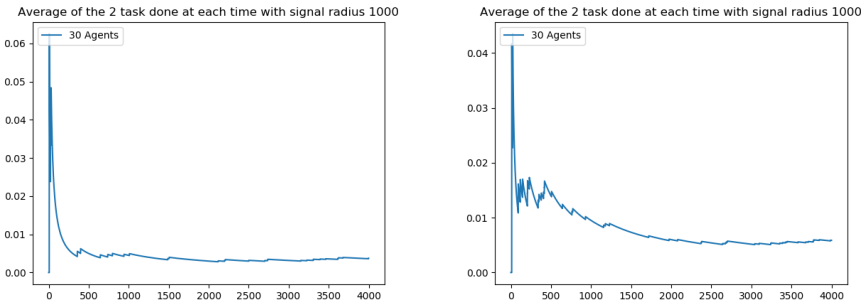

Figure 13: Left picture signal with out call off, right with call off.

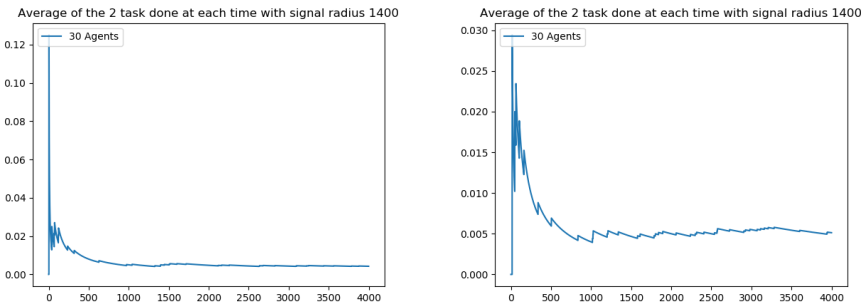

Figure 14: Left picture signal with out call off, right with call off. Here we see how bad it can get. I strongly suggest running the program to see it.

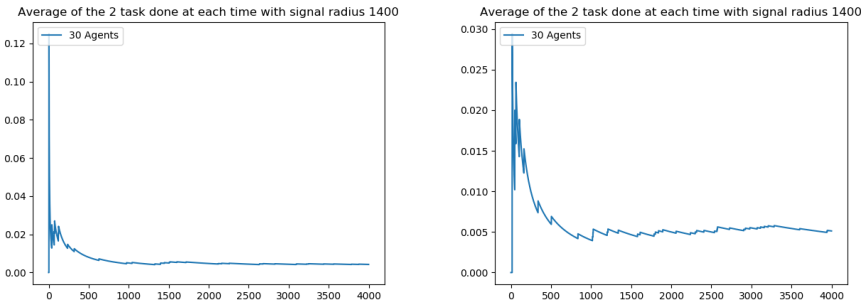

Figure 15: Left picture signal with out call off, right with call off. Here we see how bad it can get. I strongly suggest running the program to see it.

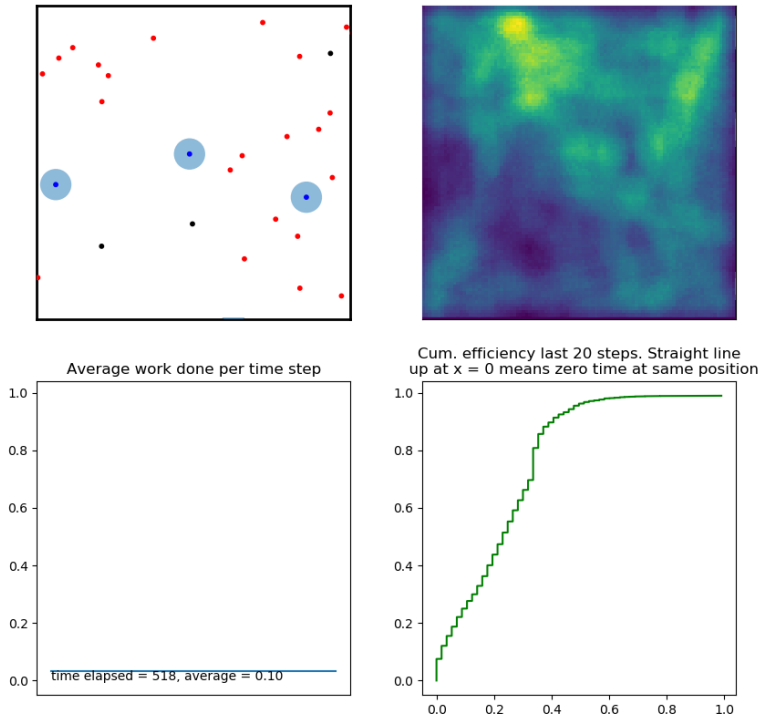

Figure 16: Example of a simulation and how it can look. The efficiency plot at bottom right is average time spent in same area last 20 time steps. Straight line up at 0 would mean that none of the agents have been at the same place again, individually.