# Distributed Intelligent Systems

# Home Work 3


# Afroze Ibrahim Baqapuri


---

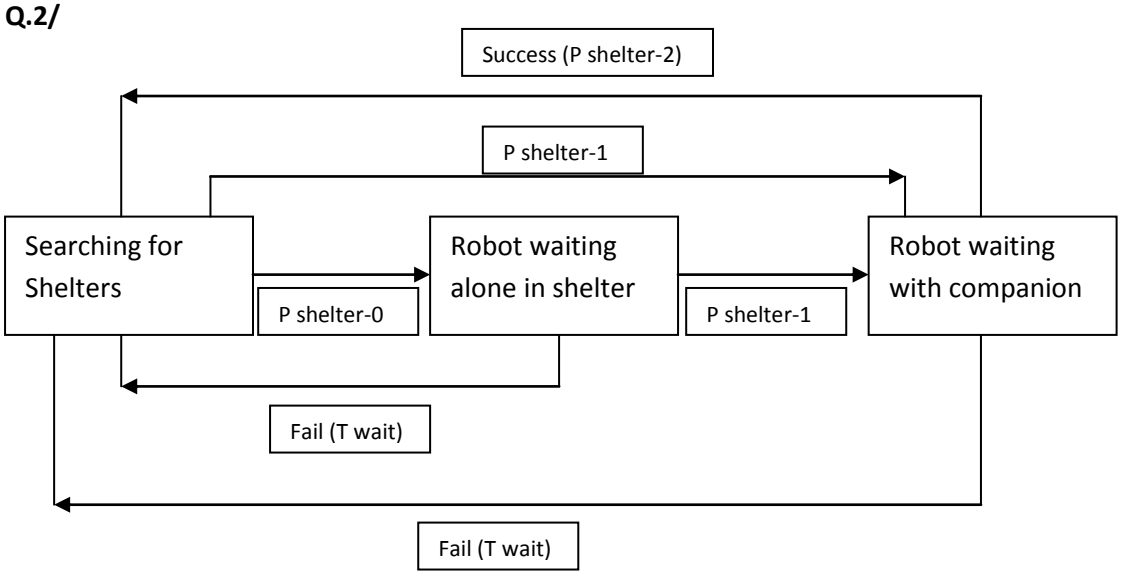
**Q.1/**

As you keep on increasing the waiting time, the collaboration rate first increases, then after reaching a maximum point is starts to decrease.

The reason for this is that now we running our program using 3 robots and total number of shelters is also 3. The first increase in collaboration rate is because the chances of 2 robots finding each other in the same shelter would be very low if both of them wait very less time for the other (so this chance/probability would initially improve as waiting time increase. But with equal number of robots and shelters there is a possibility that each robot waits in its own shelter and there is no robot out in the searching mode. So as the waiting time increases the time wasted by robots when no robot is in searching class increases which has the overall effect of decreasing collaboration rate (after a certain value of waiting time)

From my simulations the maximum collaboration rate of approximately 0.85% is reached around 100 seconds of waiting time.

**Q.2/**



Where;
P(shelter-0) = P(shelter) * { [M0 – Nw1(K) – (Nw2(K)/2)]/M0}
P(shelter-1) = P(shelter) * [Nw1(K)/M0]
P(sheler-2) = P(shelter) * [Nw2(K)/2*M0]

**Q.3/**

1)

Delta-Shelter-1(K) = N-search(K) * prob. of entering any shelter * ratio of empty shelters
Delta-Shelter-1(K) = N-search(K) * P(shelter) * { [M0 – Nw1(K) – (Nw2(K)/2)]/M0}
(note we have divided Nw2 with 2 because two robots equals to one shelter in this case)

Delta-Shelter-2(K) =2* N-search(K) * prob. of entering any shelter * ratio of 1-filled shelters
Delta-Shelter-2(K) =2* N-search(K) * P(shelter) * [Nw1(K)/M0]
(note: 2 is multiplied since robot waiting alone also joins in Nw2 along with shelter entering robot)

2)

Delta-Collab(K) = 2*N-search(K) * prob. of entering any shelter * ratio of 2-filled shelters
Delta-Collab(K) = 2*N-search(K) * P(shelter) * [Nw2(K)/2*M0]
(note: 2 is multiplied since the 3$^{rd}$ robot never waits in the shelter and never leaves search state)

**Q.4/**

Gamma-1 represents the fraction of robots that entered empty shelter at k – t(wait) and did have any other robot enter their shelters till t(wait).

Gamma-2 represents the fraction of robots that entered 1-filled shelter at k – t(wait) and did have any other robot enter their shelters till t(wait).

**Q.5/**

C(j) = Delta-Collab(j) / 2
(since for each collaboration 2 robots return back to search state. 3$^{rd}$ is already in search)
C(j) = N-search(j) * P(shelter) * [Nw2(j)/(2*M0)]
(note j is just a unit of time and can be replaced with K, like Nw2(j) = number of robots in 2-filled state at time = j)

**Q.6/**

Since the global objective function is to minimize overall energy expended and time spent, so that will remain same.

We should change the local objective function in such a way that robots can only bid if the intersection of their functionality with the task being bid is not an empty set (in other words robots have the functionality of performing this task).

**Q.7/**

Again the global objective function remains the same.

The local objective function changes in the following way.
Let t(j) be the $j^{th}$ task being bid on and dist(r(i),s(j)) = distance of robot i from task s(j)

If s(j) = photography task:

Cost of bid of robot I = [dist(r(i),s(j)) / V(average)] + P(r)

If s(j) = sample extraction task:

Cost of bid of robot I = [dist(r(i),s(j)) / V(average)] + S(r)

**Q.8/**

The robots with lower capacity have a risk that even if they are very efficient at conducting tasks and might be the best greedy locally optimal option for task assignment, but if they travel very far to go deep in a region concentrated with a large number of tasks then it would run out of capacity and another less efficient robot may need to be called for its replacement from a far away distance. Hence overall there would be loss of resources and redundancy (because the inefficient robot with much greater capacity would alone have had been able to carry out all tasks and the efficient robot would not even have to travel so much distance to begin with).

In light of above discussion, robots with lower capacity should be inclined more towards doing jobs closer to them and less inclined towards doing far away jobs.

Hence the cost bid of the robot should contain a factor which should be inversely related with its available capacity. So if the available capacity is low the multiplying factor would make the overall bid-cost higher, and vice versa.

**Q.9/**

Consider that there is a sample extraction task which is in fact unreachable and there is a sampling robot very far away from the task and a photography robot very close to that task. So instead of having the sampling robot travel a great distance only to realize that the site is inaccessible it may be useful to have the close-by photography robot to first inspect the site and if the site is reachable then it should bid again with probability $p = 0$. Otherwise it should do nothing and consider the job done.

In light of above it maybe be useful for even photography robots to bid for a sample extraction job with a very high relative cost, so that they may only be interested in going if the photography robot is indeed very far away from the site as compared to them. Also the cost function should be dependent on probability *p*, so that if it is very low then the photography robots should not bother inspecting the site (since it would most probably just be wasting resources going there just to realize that it is an ok site).

Bidding of sampling robot: dist(r(i),s(j)) * (1-p(j))
Bidding for photography robot: dist(r(i),s(j)) * p(j) * k

Where p(j) is the uncertainty probability of sample extraction task j and k is a suitably large constant, k >= 1. And the above bids are only valid for sample extraction tasks.

**Q.10/**

Bid in the same way as above. Only difference is that now if the photography robot arrives at the site and finds it to be inaccessible it photographs the site before considering the job done (otherwise as above it triggers are re-bid with p=0). Also the value of k may need to be adjusted for this environment for specific optimization.

**I.11/**

Implementation file is attached with this report for reference. I have used 2 threshold values so for that I had to slightly change the code for supervisor file also (to include the possibility of using two thresholds). Therefore kindly use the supervisor file also attached with the report checking the algorithm.

I have implemented a specialization strategy in which each robot at random tries to “specialize” more towards doing one task than any other tasks. It is a piece-wise linear model in which the more you do a particular type of task the more you are inclined to do it, and the less you do it the less inclined you are do it. Overall result is convergence of specialization states.

This might be useful in number of situations. For example in a bee colony different bees have different task specializations and they mainly do their specified task more than any other task, and do not shift to some other task unless there is a dire need to. The result is overall increase in the efficiency of the bee colony. This is also observed in how humans behave, for example we specialize in a job/profession and are more comfortable in doing that than any other job in which we have no specialized. Result is that when it comes to performing that one job we may be much better than a person who is good in all jobs but not best in any. It would be like jack of all trades and master of none.

By using specialization in our robots we make them better at doing one task than any other task, while still giving them a possibility to perform the other task if the need arises. This may be useful in robots designed for specialized tasks and we want them to pick up randomly in accordance with some constraints and basically specialize and “learn” on their own. For example if we have robots

playing a football game, which are initially uniform, we may want each of them to become specialized in one football tactic in such a way to maximize the overall fitness function.

**Q.12/**

Numerically solving on matlab the answer is approximately 3.985

Matlab script has been attached for reference.

**Q.13/**

The probability of this is zero because only the minority class changes its direction with 0.5 probability. This means that robots in majority class at beginning (or any time of the experiment) will remain in majority class and no robot in majority class will change its direction. Hence there is no probability of this event occurring.

**I.14/**

**Note:** The algorithm doesn't print that it has converged, it just prints 10 arrays and when it has printed the $10^{th}$ array it is in the stable state. It usually takes less than 20 minutes simulated time.

The algorithm works in such a way that first it follows the exploration technique and tries to collect unique id's of all the robots in the arena. Once it has all the unique id's it simply sorts them and selects the first 70% of sorted id's and assigns them either right or left direction (implementation choice) and the rest in the other direction. Note that if we choose the first 70% to be right then we must make sure that first 70% of the id's in the sorted lists of all robots must turn right (and this sorted list would contain the id of current robot as well, which stores the list). It takes approximately 20 minutes simulated time to converge.

The critical point of the algorithm is that each robot must eventually know the id of every other robot in the arena (so that the sorted list of each robot is essentially the same and all robots agree on which robots to turn right and which to turn left).

Whenever a robot comes in the vicinity of another robot (or more than one) it communicates its id and state to it and the receiving robot stores it in a list, and decides upon which direction to assume depending on a probability of the states of robots currently in its list (since the states and id's of previous robots it encounters gets stored in the list). The decision is made in such a way to prevent the robots getting stuck for a long time and increase their exploration ability. In this way given sufficient time every robot eventually meets with every other robot and condition for convergence to stable state is satisfied.

**I.15/**

No our strategy would not scale. Reason is that it relies on the assumption that every robot must atleast come in the radio communication range (meet) of every other robot. However if the number of robots become very large then it may not be scalable for this condition/assumption to hold true in a reasonable time limit, hence it would not scale.

**Q.16/**

If we want the absorbing state to be 7 left and 3 right:
Stimulus if state left = NL
Threshold if state left = 7.5
stimulus if state right = NR
Threshold if state right = 3.5

If we want the absorbing state to be 3 left and 7 right:
Stimulus if state left = NL
Threshold if state left = 3.5
stimulus if state right = NR
Threshold if state right = 7.5

**Q.17/**

No not all conditions are satisfied because absorbing state will have to be either 7R,3L or 3R,7L but it cannot be both. Moreover since there is local information only this means that robots will not be able to know about the states of all other robots at any given instant with complete certainty henceforth it may not make local optimal choice. However it would converge over time into one of the 2 states (depending upon the stimulus and threshold pair we choose according to our choice of which absorbing state we want, since we can't have both).