**Game Theoretic Crowd Evacuation Management in IoT enabled Facilities Documentation**

PROBLEM/ABSTRACT

The code is simulating crowd evacuation management using Stochastic Learning Automata and the Minority Game sequentially. Stochastic Learning Automata is centralized, meaning that in order to operate, it requires information about how the users are interacting with the environment. It organizes users into their optimal escape route cluster, which the minority game acts on. The minority game is distributed, making it much simpler in its function, and therefore faster. The minority game simply broadcasts whether the target group was under the threshold value. The users of the *go* and *not go* group update the strategy accordingly. When the threshold for convergence is high, the minority game ensures that the target group population is less than or equal to the preset threshold. When the convergence needed to exit the minority game is low, users in routes can exceed capacity and decrease the quote of service of that route. Ultimately, routes can become more distinguished in their effectiveness to manage crowd evacuation when the minority game is quicker and less effective, simulating real life conditions. Information crucial to users making better decisions is updated after the minority game, and the number of iterations needed for all the users to evacuate the critical area is recorded.

CONCEPT

*Stochastic Learning Automata*

The code simulates a set of users evacuating a critical area. Each user has a vector of action and reward probabilities the length of the number of escape Routes (therefore, there is an element dedicated to each route). The escape route cluster the user is placed in is randomly selected according to the action probabilities (the higher probabilities have more weight)\*. For a user, Every escape route has a reward probability calculated using equation 1. The reward probability for the chosen route is updated every time slot using variables describing how users are placed in the environment, including the number of users in a route, their distance from a route, and the sum of QoS for a route from every time slot before [1].

Again, the reward probability is updated for only the chosen route. This value is put over the sum of the non - updated reward probabilities and the newly updated prob [2] . For each user, the randomly chosen route’s action probability is updated using [3], and every other action prob is updated using [4]. The normalized reward prob used is the same for all cases.

The second method of determining clusters relies on the convergence of the user’s action probabilities beforehand. In the first method, because reward probabilities are updated every time slot (after the minority game is played and the escapees are determined) the action probabilities would update using the same reward probability every iteration. The action probabilities will change and possibly converge, however not optimally. The incorporation of a new variable, the size of the cluster, is incorporated into the reward probability formula [5]. The lengths of the updated clusters are used after each iteration to update the reward probability for a randomly chosen route. This new reward probability is used to create a normalized reward probability (6). Next, the action probabilities are updated; [7] updates the chosen route’s action probability, [8] updates the other routes’ action probabilities . The final clusters are determined after every users action probabilities are converged to one route’s cluster

\*for the first iteration, a user’s action probs are congruent for all routes.

*Minority Game (MG)*

The need of the minority game exists solely when the route is not filled to capacity to the point that some but not all of the users in a cluster can escape. If there is enough capacity for all the users in a cluster, all users can go. If there is no capacity for users, all return to the critical area.

For each user, there are two blank vectors used in the minority game, MGScores and MGProbs. These 1 by 2 vectors keep track of the score and probability of selecting the go and not go strategy respectively. When the user enters the minority game, MGScores is set to two scores of zero respectively, and MGProbs is set to probabilities of 0.5 for each strategy. Each user in the cluster randomly chooses one of the two groups based on the probability distributions of their MGProbs. If the number of users in the go group is less than or equal to the previously calculated threshold (number of users in route subtracted from route capacity), their score for the go group increases by one and the scores of the members in the not go group remained unchanged. If the user in the go group is greater than the threshold, their scores remain unchanged while the no go strategy score of the users in the winning no go group increases by one.

The minority game used in this code uses exponential learning to update the user’s probability to go or not go. During each iteration, after each user’s scores have been updated, their MGProbs are updated using the exponential learning formula. After all the users’ MGProbs have converged to one group, the escapee and returnee groups are finalized.

*Updating Route Information*

After the minority game is played, information used in calculating the reward probability is updated, including the number of users in routes, the evacuation rate for the escape route [9] and the quote of service sum of every route. If the number of users in the route does not exceed capacity, the cumulative QoS rating of the escape route increases by [10]. Else, the QoS simply increases by 0.1 [11].

In the second stochastic learning method (requiring convergence of all users to a threshold before users are put in clusters), the action and reward probabilities for each user are reset in addition to all the updates mentioned above.

DESIGN (refer to comments in the code for more detail)

It is important to implement classes for escape routes and users as they have unique information attached to them that changes with every iteration (Refer to EscRt.m and User.m). It makes the code easier to read and understand.This code is somewhat object oriented; objects of escape routes and users are made to be used in the main code.

When ConvMain.m or NonConvMain.m are run, the population of the critical area and the counter of how many timeslots have passed will be printed onto the Command Window. However, in the Workspace, matrices will store how cluster sizes, evacuation rates, Users in routes etc. changed as the simulation went on and make it easier to notice trends and point out conceptual errors or bugs.

ANALYSIS

*The Quotes of Service of the routes diverge when the minority game is imperfect and allows the routes to exceed capacity.*

*On the subject of Converged Clusters*

Seemingly, the primary benefit of making clusters based on users’ converged action probabilities is their decisions reflect updated information about escape routes. They should objectively choose a new cluster every time slot without any bias from the previous time slot’s probabilities. These clusters should reflect the QoS of the Escape Route. As a result, the escape Routes’ capacities should be used to their fullest extent. More people exit every iteration, taking fewer time slots for the Critical area to clear out.

It would seem that making clusters independent of convergence means that users slowly learn how the efficiency of the routes are changing; if they have favored one route in the past, it will take longer for their action probabilities to favor another action route even when the information for that route is updated . For example, if the QoS for every route is similar, the number of users choosing each cluster should be even as well. In addition, because users leave and the action and reward probabilities do not reset, this correspondence between QoS and cluster size should decrease as the simulation runs .

However, after running NonConvMain.m and looking at the distribution of users in clusters, we can see that NonConvMain achieves cluster distributions that are comparable to the QoS of routes without taking as much time as ConvMain.m does in converging all users to one cluster *every* time slot.

*Dynamic evacRate formula change [9]*

With the dynamic evacuation rate formula, it is possible for the evacuation rate for a route to **increase** with the addition of users . This is not the intended effect of a dynamic evacuation rate, which strives to simulate a bottleneck effect, or the effect of congestion on outflow. In addition, as the formula is recursive, there is never a decrease in evacuation rate, even when the number of users in the route starts to approach capacity (Compare the first few rows of RouteUsersMat and EvacRateMat after running ConvMain.m). One way to improve this is to eliminate the recursion. Make the formula simply rate = capacity/users as evacuation rate is inversely proportional to the number of users.

If the intent of the original formula is for the rate to stay the same as the initial value when the route is filled to capacity, then the simulation runs as it should. The model as a whole simulates what would happen in a real world situation; first there is congestion, but gradually people leave faster and faster (Refer to the increased decrease of people returning to the Critical Area with every time slot in CritAreaMat after running ConvMain.m, excluding the first iteration).

*Change in QoS update formula for under capacity routes. [10]*

Previously, the QoS would increase by one when the route was filled below capacity and increase by zero otherwise. However, the new formula, ( +(capacity - users)/capacity + 0.1 for good service, +0.1 for lacking service) is more effective in updating the quote of service because it accounts for how much under capacity the route is. If there are no users, then 1.1 will be added to the route’s QoS. If the route is almost filled to capacity, the quote of service will only increase by a small fraction greater than 0.1. With this new formula, users can make better decisions of which cluster to choose. In addition, it is easier for us to see whether the routes’ are being used to their fullest extent. A larger quote of service would mean that escape routes were left under capacity more times than a route with a smaller quote of service.

*Prevention of divide by zero errors*

The denominator of the reward probability formula and the the Evacuation Rate formula were all added to 1 so that if the users in route were zero, the program would not crash.

*On Converged Clusters*

The primary benefit of making clusters based on users’ converged action probabilities is their decisions reflect updated information about escape routes. They objectively choose a new cluster every time slot without any bias from the previous time slot. These clusters reflect the QoS of the Escape Route. As a result, the escape Routes’ capacities are used to their fullest extent. More people exit every iteration, taking fewer time slots for the Critical area to clear out.

Making clusters independent of convergence means that users slowly learn how the efficiency of the routes are changing; if they have favored one route in the past, it will take longer for their action probabilities to favor another action route even when the information for that route is updated . For example, if the QoS for every route is similar, the number of users choosing each cluster should be even as well. However, because users leave and the action and reward probabilities do not reset, this correspondence between QoS and cluster size decreases as the simulation runs (after running ConvMain.m, compare how the standard deviations of QoSMat’and ClusterMat’ are changing through timeslots ).