CSCE 475/875

Final Project Report

Treasure Hunters Simulation: Local Decisions vs. Global Coherence

Team: GZ

5/08/2020

# Abstract

In this paper, we present a design for a multiagent system that models treasure hunter (explorer) agents searching for one of multiple treasures that depreciate in value over time. The explorer agents’ goal is to maximize their individual utility which corresponds to a treasure’s value that they obtain either alone or in a team with another explorer agent. If alone, the treasure’s value upon uncovering one is their utility while if in a team the treasure’s value is split evenly between the teammates and given as the utility for each of them. In our simulation, we model the explorer agents with many different parameterizations in order to better understand the underlying local decisions being made by the explorer agents (pertaining to whether or not the explorer agents want to search for a treasure alone or in a team with another) and their effect on global coherence.

# Introduction

In order to investigate the impact of local decisions on global coherence within a multiagent system, we designed a treasure hunter simulation using the Repast Simphony simulation toolkit in the Java programming language. The simulation will consist of one type of agent, the explorer agents, and one type of item, the treasures. The explorer agents are designed to perceive only what is within their perception region, know the places they have perceived in the past through utilizing their imperfect memory, know the treasures’ starting values as well as their depreciation rates, and know the identity of the explorer agent that is nearest to them at any given time step. The explorer agents must make a decision on how to move and whether or not they should team up at every time step. All treasure items are designed to start at the same value and depreciate at the same constant rate. The desired emergent behaviors of the simulation is that the explorer agents will discover the treasure in an efficient manner and that the agents who start the simulation nearest to a treasure will team up less frequently than those starting further away. We conceived a few hypotheses that ponder the effects of the explorer agents’ decisions on emergent behavior and designed experiments that adjust various environmental parameters to verify them.

# Simulation Design

The multiagent system will have an area of the size *L* x *L* square units and there will exist *T* treasure items and *N* explorer agents located randomly in the area. The multiagent system will consist of one type of agent, the explorer agent, and one type of items, the treasure. Each explorer agent’s goal is to uncover a treasure item in an efficient manner, which is to say in a manner that maximizes their reward while considering the lack of knowledge of the location of any treasures in the unmapped area.

The treasure items, once randomly placed, will not move and will not be able to sense anything in its environment, but they will have a preset value assigned to them. This value will then slowly deteriorate at a constant rate over time until they have been found by an explorer agent. The treasure items do not make any decisions and do not perform any other actions besides the deterioration of their value.

The explorer agents will be assigned a percentage memory, *S*, from within the interval 10-50% which determines their navigation ability. This radius will be set up as the Moore neighborhood with *K* being the radius in the north, south, east, and west directions, as shown in **Figure 1** below. The explorer agents, once randomly placed, will be able to move in the area in four directions: north, south, east, and west, one unit at a time. After each move, the explorer agents will first check if, in a radius of *K* units from their location, they have found the treasure.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  | EA |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |

**Figure 1**: Explorer Agent’s (EA) treasure detecting radius with *K* = 3 (gray area)

### Algorithms

To decide what actions to take, an explorer agent’s reasoning algorithms are shown in the algorithms below.

**Algorithm Exploring(ExplorerAgent explorerAgent)**

1. If explorerAgent.getUncoveredTreasure() is false then
   1. currentLocation ← explorerAgent.getCurrentLocation()
   2. treasureFound ← explorerAgent.checkNeighborhood(currentLocation)
   3. If treasureFound is true then
      1. If adjacentToTreasure is true then
         1. explorerAgent.setUncoveredTreasure(true)
      2. Else
         1. Move(explorerAgent, currentLocation, treasureFound)
   4. Else // treasure is not found
      1. If Alone(explorerAgent) AND Alone(nearestExplorerAgent) then
         1. If TeamUp(explorerAgent, nearestExplorerAgent) is false then
            1. Move(explorerAgent, currentLocation, treasureFound)
         2. Else
            1. explorerAgent.addToTeam(nearestExplorerAgent)
            2. nearestExplorerAgent.addToTeam(explorerAgent)
      2. Else
         1. Move(explorerAgent, currentLocation, treasureFound)

**End Algorithm**

**Algorithm TeamUp(ExplorerAgent explorerAgent, ExplorerAgent nearestExplorerAgent)**

1. unknownArea ← totalArea - explorerAgent.getSearchedArea()
2. currentSpeed ← explorerAgent.getSearchedArea() / timeTaken /\*Get speed at

which new

areas have

been

searched\*/

1. prospectiveSearchedArea ← nearestExplorerAgent.getSearchedArea() + explorerAgent.getSearchedArea()
2. potentialSpeed ← prospectiveSearchedArea / timeTaken /\*Get speed at which

new areas can be searched with new search area\*/

1. timeToDiscoverTreasureCurrent ← (unknownArea / currentSpeed) / numTreasures
2. timeToDiscoverTreasurePotential ← (unknownArea / potentialSpeed) / numTreasures
3. treasureValueWithCurrent ← timeToDiscoverTreasureCurrent \* treasureDecayRate
4. treasureValueWithTeam ← timeToDiscoverTreasurePotential \* treasureDecayRate
5. individualPayoutCurrent ← treasureValueWithCurrent / numTeamMembers
6. individualPayoutPotential ← treasureValueWithTeam / numTeamMembers + 1
7. If individualPayoutCurrent < individualPayoutPotential then
   1. return true
8. Else
   1. return false

**End Algorithm**

**Algorithm UpdatePermanentMemory()**

1. numLocationsToAdd(percentSMemory \* workingMemory.size())//percentSMemory corresponds to variable parameter S
2. For number in numLocationsToAdd
   1. permanentMemory.add(random(workingMemory)) /\*add random location from working memory to the agent’s permanent memory\*/

**End Algorithm**

**Algorithm Move(ExplorerAgent explorerAgent, Location currentLocation, boolean treasureFound)**

1. If treasureFound is true then
   1. oldPerceptionRegion ← explorerAgent.getPerceptionRegion()
   2. moveTowards(explorerAgent.getTreasureLocation())
   3. newPerceptionRegion ← explorerAgent.getPerceptionRegion()
   4. workingMemory ← oldPerceptionRegion \ newPerceptionRegion /\*”\” stands for “set minus”\*/
2. Else
   1. oldPerceptionRegion ← explorerAgent.getPerceptionRegion()
   2. moveTowards(random(undiscoveredLocations))
   3. newPerceptionRegion ← explorerAgent.getPerceptionRegion()
   4. workingMemory ← oldPerceptionRegion \ newPerceptionRegion /\*”\” stands for “set minus”\*/
3. UpdatePermanentMemory()

**End Algorithm**

The Exploring algorithm is responsible for determining what action to take after each time step. It will determine whether or not the explorer agent has found the treasure in its neighborhood, if so then it will either uncover the treasure (if it is adjacent to it) or it will move towards it. If the explorer agent has not found the treasure, then it will call the TeamUp function to determine if it is more efficient to team up with another agent or to stay with its current party size. If it doesn’t team up, then it will move towards a random undiscovered location.

The TeamUp algorithm determines whether or not the explorer agent should team up based on the number of treasures, the speed at which the explorer agent and the nearest explorer agent are discovering new locations, and the speed of the treasure’s deterioration of value. This information is used to simply calculate an estimated reward for the explorer agent in the scenario in which it teams up or stays as is. The two estimated rewards are then compared to one another, and whichever is larger, the explorer agent chooses to do. The TeamUp algorithm naturally accounts for the memory and perception radius of each of the explorer agents by utilizing each agent’s current searched area using the getSearchedArea algorithm (in which the current searched area is calculated using their memory and perception radius) in the calculations. The TeamUp algorithm is made mutual (i.e. agent A and B will each add each other to their team only if both agent A and agent B have determined they want to team up with each other) through utilization of successive calls from the Exploring algorithm that ascertains each agent adds the other to its team.

The UpdatePermanentMemory algorithm is used to update the explorer agent’s searched areas in which it has remembered. This is done by keeping track of the agent’s “workingMemory” which is updated after each Move to be the locations that need to be further processed (by multiplying the navigation percentage memory S) before being put into “permanentMemory.” The agent’s “permanentMemory” represents permanently remembered locations that the agent has perceived before.

The Move algorithm moves the given explorer agent one unit in either the direction of the found treasure or a random unknown location. In addition to this, the Move algorithm also updates the “workingMemory” of the explorer agent.

### Environment

Regarding the environment, of the five key properties (incomplete, stochastic, non-episodic, dynamic, real-time), the properties that are made available in the multiagent system are:

* Incomplete
  + Explanation: Explorer agents cannot detect where the treasures are until in perception radius nor can they see the location of every other agent
* Stochastic
  + Explanation: When an agent tries to establish a teammate, but may not always be successful
* Non-episodic
  + Explanation: The actions the explorer agent has taken, such as teaming up with another, has an effect on the next action that agent takes, therefore, the effect of the agent’s actions need to be considered for the future
* Dynamic
  + Explanation: The treasure items within the environment modify it by way of their value deterioration
* Not real-time
  + Explanation: The environment only has a discrete number of actions capable of being performed within it.

# Desired Emergent Behavior

One of the desired emergent behaviors is that the explorer agents will discover the treasure in an efficient manner. This is determined by their starting position relative to the location of the treasure, the time it would take for them to reach it given incomplete information, and the reward they are given if they are to reach the treasure in a given time with a given number of teammates. Another desired behavior is that agents that are placed further away from the treasure should team up more frequently. On the other hand agents closer should remain single for longer.

# Hypotheses

### Hypothesis 1

The shrinkage of the *L* x *L* area will result in more explorer agents choosing to work alone, rather than teaming up. In addition, the smaller area will lead to a smaller amount of time for the explorer agents to find a treasure. This hypothesis may be validated in the experiments due to the logic of a lesser general area corresponding to a shorter distance from the explorer agents to the treasure on average. This, in turn, will result in fewer explorer agents choosing to cut their reward by teaming up as it would not be worth it considering the trade-off between the area size and the time it takes for an explorer agent to find the treasure (which is directly proportional to the reward).

### Hypothesis 2

The explorer agents that find a treasure fastest will not be the explorer agents with the highest reward values. This hypothesis may be validated in the experiments due to the logic that the explorer agents that have teamed up will be able to search areas much faster than those that have not, thereby leading them to discover the treasure sooner than those who are alone. Due to the consequence of teaming up being the splitting of the treasure’s value, these explorer agents will not earn as great of a reward as those that found the treasure alone.

### Hypothesis 3

Explorer agents with smaller perception radiuses (*K*) and lesser navigation memory (*S*) will be more successful if they team up with other explorer agents.￼ This hypothesis may be validated in the experiments due to the logic of this type of explorer agent not being able to search for the treasure fast enough to get a reward (due to the degradation of the treasure’s value) when alone.

# Experiments

Our experiments are designed to test the previous hypotheses.

|  |  |
| --- | --- |
| Parameter | Range of Values |
| Treasures’ Starting Value (Constant) | 1000 |
| Treasures’ Depreciation Rate (Constant) | .01 |
| Number of Treasures (T) | 10, 30, 50 |
| Side Length of the Square Area (L) | 40, 70, 100 |
| Number of Explorer Agents (N) | 100, 200, 300 |
| Perception Radius (K) | 3, 4, 5, 6 |
| Percentage of Retained Memory (S) | 10%, 30%, 50% |

We originally planned to have 1 x 1 x 3 x 3 x 3 x 4 x 3 = 324 combinations. We would run 10 simulations for each combination, resulting in 3240 runs. Each set of 10 simulations would then be averaged together to produce the results of the system. Each simulation will also collect data on: the distance the agent who has found the treasure was from the treasure at the beginning, the time it takes for an agent to discover the treasure, and the number of teams that have been formed. Unfortunately, time constraints did not allow for this so we opted to perform targeted data analysis in which we chose a handful of parameter combinations (from our parameter and range of values table) to test.

# Results

The following tables show graphical results of the total number of explorer agent teams formed over time.

Note: When not explicitly stated via the tables, parameter values are:

T = 10

L = 40

N = 100

K = 3

S = .1

|  |  |  |  |
| --- | --- | --- | --- |
|  | S = .1 | S = .3 | S = .5 |
| K = 6 |  |  |  |
| K = 3 |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  | N = 100 | N = 200 | N = 300 |
| L = 40 |  |  |  |
| L = 100 |  |  |  |

|  |  |  |
| --- | --- | --- |
| T = 10, L = 100 | T = 30, L = 100 | T = 50, L = 100 |
|  |  |  |

# Conclusions

The trials that were conducted using the above parameter configurations prompted the following conclusions:

* As area of environment increases completion time increases and number of teams formed increases at the start (nearly all agents form teams immediately)
* As the number of explorer agents increases completion time increases and number of teams formed increases overall (the number of teams formed is spread out throughout the simulation)
* As retained memory increased completion time increased, but number of teams formed stayed steady (any team that formed, formed at nearly the same time as well)
* As perception radius increased completion time decreased and time to create teams shortened (teams formed faster), steady number of teams formed.
* As number of treasures increased completion time decreased, and teams formed stayed the same (0)

# Future Work

The experiments conducted within this paper were used to discover the efficiency of agents. This work did not form a real world scenario. So in the future, models based on this one can expand on it by incorporating factors such as, walls or other obstacles, fatigue, a hierarchy of agents, or others. The model in this paper was also made to get the job done. It can be refined and improved upon to increase efficiency and provide more complete data. As stated previously there are many factors that would help this model function as a more real world model. Increasing the communication of agents and adding in obstacles or enemies for the agents to overcome would cause the model to perform more like a real world scenario. In addition, adding hierarchy of agents, sort of like a company, would change how the agents interact and could possibly have interesting outcomes. Potentially the agents could all belong to different companies and share information between only their company. This could modle how companies interact with regards to a scarce resource. They might be able to pay for information or steal information from other companies and simulate corporate conflict or cooperation. Another way that the results of this paper can be improved is by increasing the ranges of the parameters, or testing more parameters within the ranges already given. The model allows for any integer to be used for each parameter, but this paper only focused on the 324 combinations that were given earlier in the paper. A future version could test specifically how changing the size of the area affects the utility of the agents by using many more variations of the L parameter than this paper did. This method of future testing can be done for all the parameters. This would give a better idea on how each parameter affected the agent’s utility is a more holistic sense. Another way to increase the reliability of the model’s results is to increase the number of trials performed. This would make the average result of the trials be a truer estimate of the combination’s effectiveness. There are many ways to improve upon this paper’s findings.