Non-Homogeneous Swarms vs. MDP's
A Comparison of Path Finding Under Uncertainty

Michael Comstock

December 6, 2012

1 Introduction

This paper presents a comparison of two different machine learning systems tasked with creating a path to and from a goal point in a simulated world.

The first system which is tested in the simulated world is a very simple Markov Decision Process (MDP) which learns by sampling the world, estimating transition probabilities and recording rewards at each state. After estimating these transition probabilities and recording the rewards, learning is completed by performing value iteration.

The second system which will be implemented and tested is a modified non-homogeneous swarm algorithm which learns again by a process similar to pheromone trail creation in ant colony optimization algorithms.

2 World Description

Learning systems in this paper navigate through a two dimensional grid world, which should be viewed as a continuous space in R^2. In this case I examine the space $[0, 800] \times [0, 600]$. This choice was made as it goes well with rendering onscreen.

The world will have a starting location near the left side of the plane and a goal location near the right. Between the two locations is small barrier which the algorithms must learn to navigate around.

Each learning system will navigate this space with no apriori knowledge of the location of obstacles or rewards in this world.

3 The MDP Algorithm

The first learning algorithm implemented to solve this problem was a simple MDP. This choice algorithm was selected because it provides a very natural framework for dealing with the unknowns of this problem. Additionally this algorithm is widely used in reinforcement learning problems in many fields and thus serves as a reasonable basis of comparison.

The MDP handles the fact that obstacles are not known apriori by having transition probabilities between states. This means that if there is an obstacle between state s and state s' the MDP may learn this by estimating $P(s, a, s') = 0$ where a is the action "go to s'." Additionally the MDP can be easily extended to incorporate not rigid barriers by setting $P(s, a, s') = p \in (0, 1)$.

Rewards are in this model are also learned and set to be -1 for any state which is not a goal state and 1 if is a goal state. The learning algorithm is not told if s is a goal state until it occupies state s. Thus learning is done in an online fashion.

A more detailed description of the MDP algorithm is given in the appendix.

4 The Swarm Algorithm

The second learning algorithm implemented to solve this problem is a swarm algorithm with non-homogeneous agents. This algorithm's creation was inspired by nature - particularly ants' - ability to solve this exact problem on a regular basis.

This algorithm learns by having agents wander around randomly until they hit a goal state. Agents then communicate to each other how long it took them to reach the goal state. Agents that have visited a goal state recently will stop along their path to the goal in order to act as markers leading other agents to the goal. By repeatedly traveling to and from a goal state the swarm is able to rapidly build a (near shortest) path from the start state to the goal state. A (much) more detailed description of this algorithm is given in the appendix.

5 Results

A brief overview of results is given in the table below

	MDP	**Swarm**
RAM Usage	$\sim$ 1 GB	$\sim$ 90 MB
Elapsed Time	$\sim$ 300s	$\sim$ 6s
Solution Quality	Jagged; Grid-like	Smooth

5.1 Analysis

5.1.1 Ram Usage

The MDP needs to store the transition probabilities between each state for each action. Thus it needs $8 \times |states| \times |actions| \times |state|$ bytes of data are needed assuming each probability is stored as a 32-bit float. For a 80×60 discretization of the world, this takes about 1 gigabyte of space.

By contrast the swarm algorithm does not store any world states. It stores only the state of each member of the swarm. This takes 90 megabytes of space, since each agent stores only a limited number of values (see Appendix for details on what each swarm agent stores).

5.1.2 Elapsed Time

The elapsed time metric indicates how much time *seconds* elapsed before the algorithm converged on a solution. Note that the metric is seconds rather than clock cycles as neither the MDP nor the swarm was written in a manner which optimized for reducing CPU cycles. This metric is only meant to provide a very rough comparison in the amount of time required for each algorithm to solve the problem.

By this metric the swarm once again greatly out performed the MDP. This is due to a combination of multithreading the number of computations required for each algorithm. The MDP performs value iteration after several samplings of the transition probabilities. Value iteration requires many iterations (up to thousands) over each state for the values of each state to converge. Given there are thousands of states per iteration, this computation can take several seconds. By contrast the swarm algorithm explores states and assigns accurate values to each state simultaneously (see Appendix for more detail).

5.1.3 Solution Quality

The solution quality metric is a purely subjective metric which compares what each solution looks like. This is useful for visualizing how aesthetically pleasing each algorithm would be for controlling a game AI or what the expected behavior of a robot controlled by each algorithm would look like.

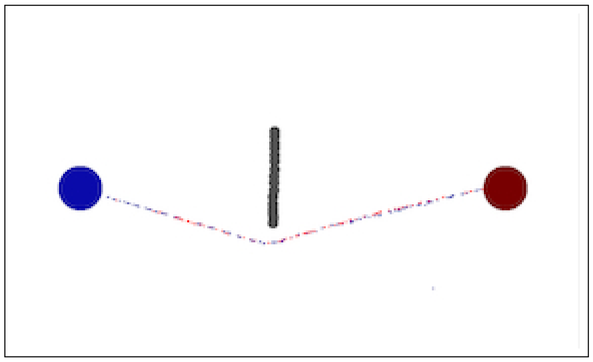

Figure 1: Swarm Algorithm demonstrates a smooth path after converging on a solution.

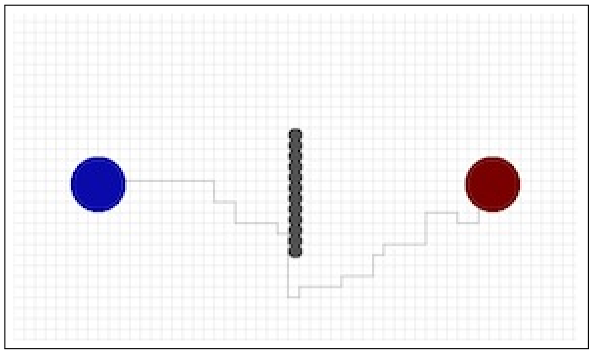

Figure 2: The MDP demonstrates a jagged path after converging on a solution.

6 Concluding Remarks

This document presents two very different algorithms for solving a path finding problem with no a priori knowledge of the environment in which each algorithm operates. While much further optimization is needed for both algorithms, the above results indicate the non-homogenous swarm algorithm is a potential improvement over MDP's for navigating through continuous spaces.

Please see the appendix for a detailed description of the novel way in which the swarm algorithm builds solution path.

A MDP Definition

The MDP learning system implemented in this paper had the stanford form [1]. Each action was produced from the following equation. Given a state s the action, a, taken at s is

$$a = \text{argmax}_a \sum_{s'} P(s, a, s')V(s')$$

where $V(s')$ is the value of being in s'.

The value $V(s)$ is computed by value iteration

$$V(s) = R(s) + \max_a \sum_{s'} P(s, a, s')V(s')$$

The values for $P(s, a, s')$ are computed by sampling the world while executing the current policy which chooses actions randomly in its first iteration.

B Swarm Agent Definition

Each agent in this system should be treated as a 7-tuple with the following fields:

$$\{x, v, t, d, g, m, b\}$$

x	The position of the agent in the problem space.
v	The velocity of the agent in the problem space.
t	The time since the agent has passed through its goal location.
d	The distance over which the agent may communicate.
g	The goal (either base or resource) which the agent is trying to reach.
m	A true false bit indicating if the agent is in the path marker state (1) or not (0).
b	A true false bit indicating if the agent is in the border marker state (1) or not (0).

Examining the m and b fields, one sees the agents may exists in up to four states: $(0, 0), (0, 1), (1, 0), (1, 1)$. Interally however we define the state $(1, 1)$ as unreachable and thus the agent may exist in only 3 states.

B.1 Agent States

An agent which is in state $(0, 0)$ is in the default state. In this state the agent moves through the problem space seeking its goal (either the resource or the base). Henceforth I will refer to this state as the *active state*.

An agent which is in the state $(1, 0)$ serves as a marker for other agents. This marker directs the agents in state $(0, 0)$ towards the goal which the agent in state $(1, 0)$ has previously visited. I will refer to this state as the *marker state*.

An agent which is in state $(0, 1)$ serves as a border marker for other agents. These markers direct agents in state $(0, 0)$ away from obstacles as well as limit the distance over which agents can communicate. I will refer to this state as the *bumper state*.

Agents in *active state* move through the problem space. The agents in *bumper* and *marker* states form the path structure in the problem space.

B.1.1 The Bumper State

The bumper state is the simplest state of the three states. Agents transition into the bumper state whenever the collide with an obstacle in the problem space. Agents transition out of the bumper state after a set period of time if no other agents are communicating with them.

Once in the bumper state agents limit their communication distance to a small radius around them.

Agents in this state communicate two pieces of data to other agents. The first piece of data the bumper agent communicates is that it is a bumper agent (it has hit an obstacle). The second piece of information the bumper agent communicates is its location.

B.1.2 The Marker State

Agents in the marker state form the main path structure of the system. Agents transition into the marker state whenever they are have the smallest value for t within half the communication distance of the nearest marker. If there are no other markers nearby (within communication distance) then an agent will transition into the marker state.

Agents transitions out of the marker state after a set period of time. Once an agent has transitioned out of the marker state, it may not transition back into the marker state until it has passed through a goal point.

When an agent is in the marker state it does not update its value for t.

Agents in the marker state transmit three pieces of information to other agents within their neighbor distance d. The first piece of information is the value of t which they have stored. The second piece of information is their location x in the problem space. The third piece of information which the marker transmits is which goal - g - the agent was seeking before it became a marker.

One crucial task the marker agents carry out is setting the proper neighbor distance d. This distance is set in the following manner. The marker agent sets its communication distance d to be the minimum distance from the marker location to a bumper. Setting a distance larger than this would cause the marker to pull agents in the active state into barriers. Setting a distance smaller than this would be inefficient. See figures below for examples of how setting the neighbor distances for markers affects the systems behavior.

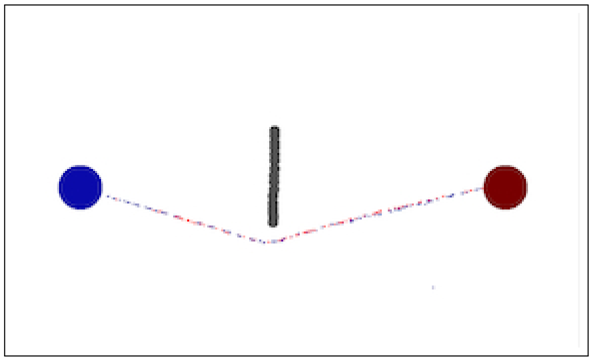
Figure 3: Markers with properly set distances d guide active agents around a barrier

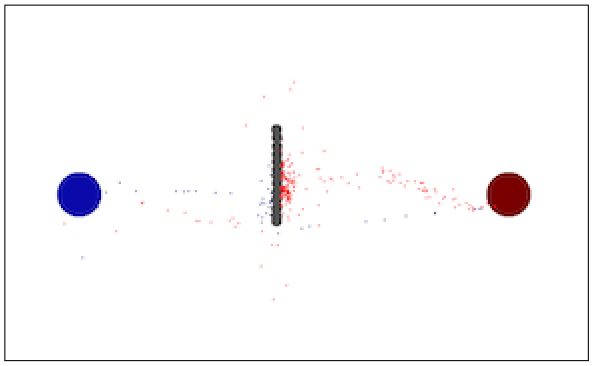

Figure 4: Markers with distances d set too large guide active agents into the barrier

B.1.3 The Active State

Agents in the active state move about the problem space alternating seeking the base and resource locations. The motion of the agents in this state is entirely governed by communication with other agents.

Agents in the active state set their neighbor distance d to be equal to that of the marker nearest to them.

During each iteration, the agents in the active state update their states fields as follows:

Let x_m be the location of the best marker within communication distance of the agent. Here the best agent is defined as the marker within the marker's communication distance who most recently visited the goal which the agent is trying to reach (marker $g_m \neq g$). In other words the best marker is the marker with the lowest value for t.

Let x_a be the location of the best active agent within communication distance of the agent. The best agent is defined as the agent with $g_a \neq g$ and minimum t.

Let x_b be the location of the nearest bumper agent within the bumper's communication distance.

Let c_0, c_1, c_2 be weights and r_1 and r_2 be random values in the range [0,1]. In this implementation $c_0 = 5.72$; $c_1 = 2.51$; $c_2 = 1.59$. This implementation is a slight modification of the implementation presented by Eberhart, Shi and Kennedy[2].

$$\begin{aligned} d &= nearestMarker(d) \\ t &= t+1 \\ v &= \begin{cases} (x - x_b) \text{ if } x_b \text{ is defined} \\ c_0 \cdot v + c_1 \cdot r_1 \cdot (x_m - x) + c_2 \cdot r_2 \cdot (x_a - x) \end{cases} \\ x &= x + v \end{aligned}$$

C Swarm Behavior

The swarm as a whole undergoes two distinct phases in this problem. In the first phase agents attempt to find the unknown goal location for the first time. In the second phase, agents build a path to and from this goal.

C.1 Finding the Goal

Given each agent has cannot observe the environment and, by definition, no agent has found the goal in this phase, agents move about the problem space randomly during this phase.

Agents start this phase by scattering randomly from the base location. They proceed to bump into obstacles, transition into the *bumper state* and direct other agents away from these barriers. Agents also transition into *marker agents* whenever they are the first agent to enter a new location, or whenever they reach a location in the shortest amount of time t since they last visited the base location. Thus the agents build a map of the environment as they search for the goal. This map, though is better though of as "bread crumbs" as it is overlayed onto the environment rather than kept in some internal state.

After some time (whose probability depends on the number of agents, distance to the goal and number and distribution of obstacles) an agent will randomly find the goal location and the swarm will transition into the second phase.

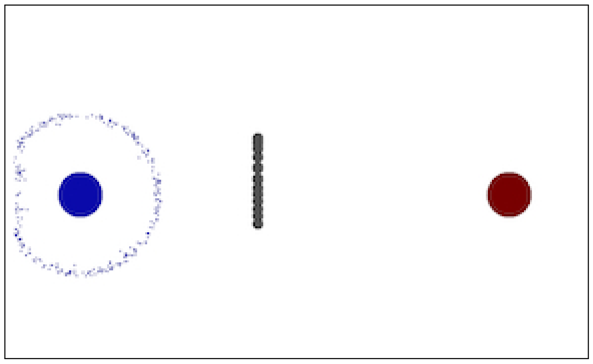
Figure 5: Agents scatter initially upon leaving the base for the first time.

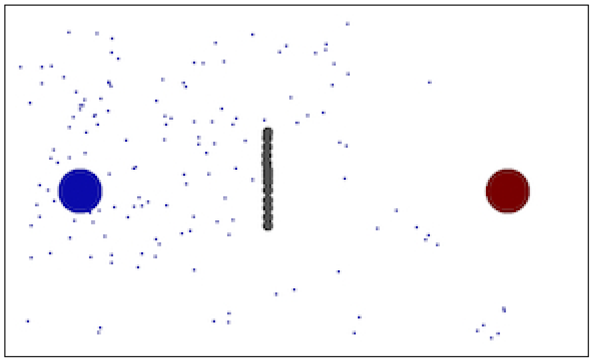
Figure 6: Agents search randomly for the goal.

C.2 Building A Path

Once an agent reaches the goal state the swarm transitions into a new phase where it builds a path to the goal. Because the first agent to hit the goal is, at that time, the only agent to hit the resource goal, it will immediately turn into a marker agent. Once in this state the agent transmits its location to all other agents within its neighbor distance d. This will cause all agents within this distance to no longer travel randomly, but rather head toward the marker agent, and thus head toward the goal.

As more and more agents reach the goal, they will leave the goal and head back towards the base location by following the *marker agents* which came from the base. Note that individual agents do not have a memory of where they have gone so they cannot simply "reverse" back to the base.

The process of transitioning from the first to the second state can be seen in Figure 7.

Examining Figure 7 we see several notable features. First we note that near the goal (red), swarm agents are beginning to demonstrate some organization. The red agents are now moving together since each one is drawn to the same *marker agents* on the bottom right. Note, *marker agents* are not shown in this figure.

Additionally we see the agents located to the left of the barrier are still moving about randomly. This is expected behavior as no agent will be able to communicate the location of the goal beyond the barrier if there are any *bumper agents* located along the barrier.

Over time the agents follow the markers back to the base. As they do so they too transition into *marker agents* and guide agents from the base around the barrier to the resource. Thus the path is first build from the base to the resource and then from the resource to the base.

A more detailed look at this process, especially the role of the *marker agents* is presented in the next section.

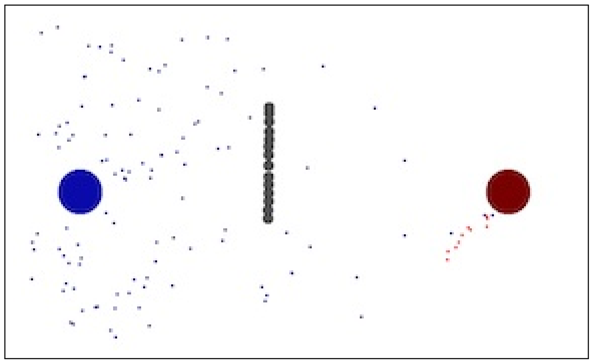
Figure 7: Agents have recently discovered the goal for the first time.

D Marker Agents

The preceding section examined the behavior of the swarm from with focus on the *active agents*. What is of far greater importance to this system however are the *marker agents*. Thus this section will take a more detailed look at how *marker agents* behave.

A useful way to think about the *marker agents* is to imagine the the above problem as a Markov Decision Process (MDP). MDP's are very useful in discrete spaces but become intractable in continuous spaces. One way of applying MDP's to continuous spaces is discretization. Choosing the proper way to discretize a space however can often prove difficult.

The role of the *marker agents* in this system is to automatically and simultaneously discretize a space and assign a value to this new discrete space.

In this context each discrete state S is a location x which is given by the agents location and a radius r which is given by the neighbor distance d. Thus an agent a is in state S if $||x_a - x_S|| \leq d_S$. The value of this state is simply $-t$ where t is the value t associated with the given *marker agent*.

Given this construct, we see a policy is generated by the *active agents* dynamics. It should be noted that this policy is generated greedily and is not necessarily optimal.

An illustration of the marker dynamics is given in the following figures. In these figures each state is shown as a hollow circle. Note that states may over lap. When an agent is in a location with overlapping states, then that agent is said to be in whichever state S which has the greater value.

Note that because agents transition out of the *marker state* periodically, there is redundancy built into the discretization. This helps prevent information loss when agents transition out of the *marker state*.

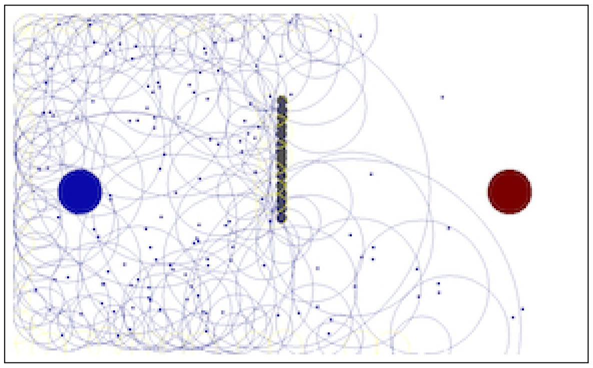
Figure 8: Agents initially scatter. During the scattering process may agents hit obstacles and transition into the bumper state. This abundance of bumpers limits the marker agents to a relatively small radius. As no goal has been found states to not demonstrate any structure and appear randomly selected.

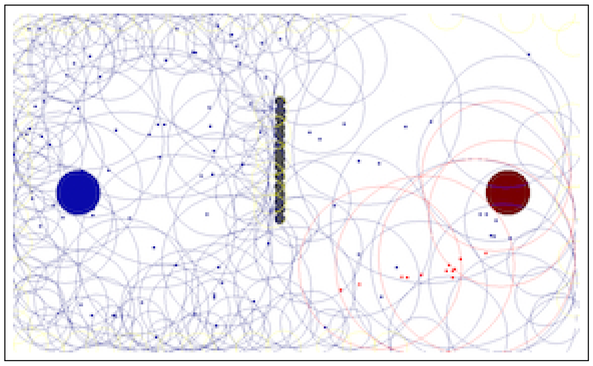

Figure 9: The goal has now been reached. We see the presence of two different kinds of states. The blue states provide insight into the problem of finding the base. The red states provide insight into the problem of finding the resources. Note that this is a picture of a proposed discretization immediately after the goal has been found. Not enough time has elapsed to allow for optimizing the discretization. Also note there are some areas still which are not covered by the discretization. These area's should viewed as states of unknown value.

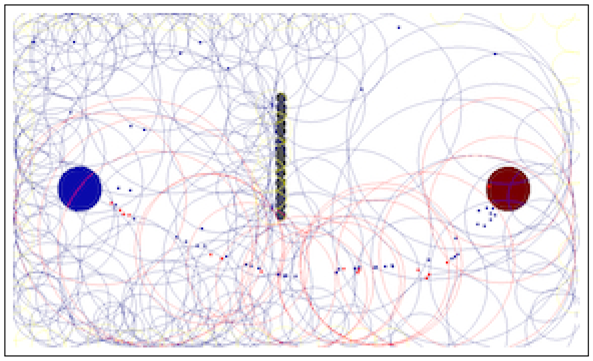

Figure 10: A path now exists both to and from the base. Discretization remains unoptimized.

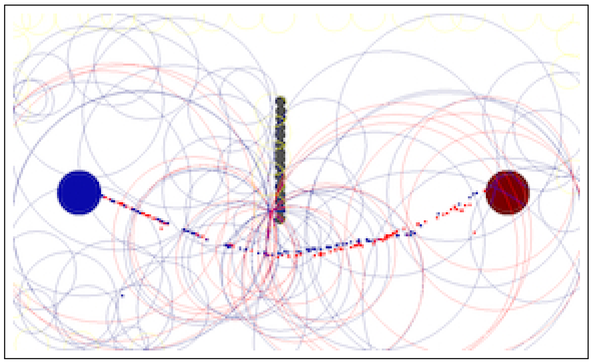

Figure 11: More agents converge onto the path. Previously created states are now removed as they go unused and become unnecessary.

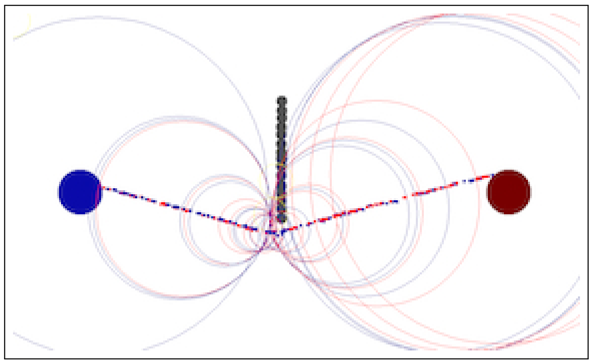

Figure 12: The swarm creates a minimal discretization of the problem space. In this stage all unnecessary segmentations are removed. One should note that states are as large as possible without going beyond the barrier. Additionally it should be noted that each state become smaller as it gets closer to the barrier. This allows the *active agents* to more precisely navigate near to the obstacle's edge. The discretization aligns with what one intuitely thinks of as the states of this system. There is a state where you are to the left of the obstacle, a state where you are to the right of the obstacle and a state where you are going around the obstacle.

References

[1] Stuart Russell & Peter Norvig, *Artificial Intelligence : A Modern Approach*. Prentice Hall, 3rd Edition, 2009.

[2] Russell Eberhart, Yuhui Shi & James Kennedy, *Swarm Intelligence*. Morgan Kaufman, 1st Edition, 2001.