Distributed Agent Evolution with Dynamic Adaptation to Local Unexpected Scenarios

Suranga Hettiarachchi, William M. Spears, Derek Green, Wesley Kerr

University of Wyoming, Laramie WY 82071, USA

Abstract. This paper introduces a novel framework for designing multi-agent systems, called "Distributed Agent Evolution with Dynamic Adaptation to Local Unexpected Scenarios" (DAEDALUS). Traditional approaches to designing multi-agent systems are offline (in simulation), and assume the presence of a global observer. In the online (real world), there may be no global observer, performance feedback may be delayed or perturbed by noise, agents may only interact with their local neighbors, and only a subset of agents may experience any form of performance feedback. Under these circumstances, it is much more difficult to design multi-agent systems. DAEDALUS is designed to address these issues, by mimicking more closely the actual dynamics of populations of agents moving and interacting in a task environment. We use two case studies to illustrate the feasibility of this approach.

1 Introduction

Engineering multi-agent systems is difficult due to numerous constraints, such as noise, limited range of interaction with other agents, delayed feedback, and the distributed autonomy of the agents. One potential solution is to automate the design of multi-agent systems in simulation, using evolutionary algorithms (EAs) [2, 9]. In this paradigm, the EA evolves the behaviors of the agents (and their local interactions), such that the global task behavior emerges. A global observer monitors the collective, and provides a measure of performance to the individual agents. Agent behaviors that lead to desirable global behavior are hence rewarded, and the collective system is gradually evolved to provide optimal global performance.

There are several difficulties with this approach. First, a global observer may not exist. Second, some (but not all) agents may experience some form of reward for achieving task behavior, while others do not. Third, this reward may be delayed, or may be noisy. Fourth, the above paradigm works well in simulation (offline), but is not feasible for real-world online applications where unexpected events occur. Finally, the above paradigm may have difficulty evolving different individual behaviors for different agents (heterogeneity vs homogeneity).

In this paper we propose a novel framework, called "Distributed Agent Evolution with Dynamic Adaptation to Local Unexpected Scenarios" (DAEDALUS), for engineering multi-agent systems that can be used either offline or online. We will explore how DAEDALUS can be used to achieve global aggregate behavior, by examining two case studies.

2 Distributed Agent Evolution with Dynamic Adaptation to Local Unexpected Scenarios

With the DAEDALUS paradigm, we assume that agents (whether software or hardware) move throughout some environment. As they move, they interact with other agents. These agents may be of the same species or of some other species [6]. Agents of different species have different roles in the environment. The goal is to evolve agent behaviors and interactions between agents, in a distributed fashion, such that the desired global behavior occurs.[1]

Let us further assume that each agent has some procedure to control its own actions, in response to environmental conditions and interactions with other agents. The precise implementation of these procedures is not relevant, thus they may be programs, rule sets, finite state machines, real-valued vectors, force laws, or any other procedural representation. Agents have a sense of self-worth, or "fitness". Agents that experience direct performance rewards have higher fitness. Other agents may not experience any direct reward, but may in fact have contributed to the agents that did receive direct reward. This "credit assignment" problem can be addressed in numerous ways, including the "bucket brigade" algorithm or the "profit sharing" algorithm [3]. Assuming that a set A of agents has received some direct reward, both algorithms provide reward to the set B of agents that have interacted (and helped) those in A. Further trickle-back rewards are also given to those agents in set C that helped those in B, and so on. Agents that receive no rewards lose fitness. If fitness is low enough, agents stop moving or die.

Evolution occurs when individuals of the same species interact. Those agents with high fitness give their procedures to agents with lower fitness. Evolutionary recombination and mutation provide necessary perturbations to these procedures, providing increasing performance and the ability to respond to environmental changes. Different species may evolve different procedures, reflecting the different niches they fill in the environment.

3 Transition of Offline Applications to DAEDALUS

Our prior applications of EAs to design multi-agent systems have used the offline approach – a global observer assigns fitness to agents based on their collective behavior. In the next section, we show how DAEDALUS could be applied to two different applications, namely, obstacle avoidance and the self assembly of machines, in an online environment. For both applications, recombination and mutation operators provide the ability to respond to environmental changes (which can include the addition and/or removal of agents).

3.1 Obstacle Avoidance

In prior work we have shown how our artificial physics framework can be used to self-organize swarms of mobile robots into hexagonal lattices (networks) that

[1] The work by [8] is conceptually similar and was developed independently.

move towards a goal (see Figure 1). We extended the framework to include motion towards a goal through an obstacle field. An offline EA evolved an agent-level force law, such that robots maintained network cohesion, avoided the obstacles, and reached the goal. The emergent behavior was that the collective moved as a viscous fluid [4].

Fig. 1. Seven robots form a hexagon, and move towards a light source.

The Artificial Physics Framework: In our artificial physics (AP) framework [7], virtual physics forces drive a swarm robotics system to a desired configuration or state. The desired configuration is one that minimizes overall system potential energy, and the system acts as a molecular dynamics ($\mathbf{F} = m\mathbf{a}$) simulation.

Each robot has position $\mathbf{p}$ and velocity $\mathbf{v}$. We use a discrete-time approximation of the continuous behavior of the robots, with time step Δt. At each time step, the position of each robot undergoes a perturbation $\Delta\mathbf{p}$. The perturbation depends on the current velocity, i.e., $\Delta\mathbf{p} = \mathbf{v}\Delta t$. The velocity of each robot at each time step also changes by $\Delta\mathbf{v}$. The change in velocity is controlled by the force on the robot, i.e., $\Delta\mathbf{v} = \mathbf{F}\Delta t/m$, where m is the mass of that robot and $\mathbf{F}$ is the force on that robot. F and v denote the magnitude of vectors $\mathbf{F}$ and $\mathbf{v}$. A frictional force is included, for self-stabilization.

From the start, we wished to have our framework map easily to physical hardware, and our model reflects this design philosophy. Having a mass m associated with each robot allows our simulated robots to have momentum. Robots need not have the same mass. The frictional force allows us to model actual friction, whether it is unavoidable or deliberate, in the real robotic system. With full friction, the robots come to a complete stop between sensor readings and with no friction the robots continue to move as they sense. The time step Δt reflects the amount of time the robots need to perform their sensor readings. If Δt is small, the robots get readings very often, whereas if the time step is large, readings are obtained infrequently. We have also included a parameter F_{max}, which provides a necessary restriction on the acceleration a robot can achieve. Also, a parameter V_{max} restricts the maximum velocity of the robots (and can always be scaled appropriately with Δt to ensure smooth path trajectories).

In this paper we are investigating the utility of a generalized Lennard-Jones (LJ) force law (which models forces between molecules and atoms).

$$F = 24\epsilon \left[\frac{2dR^{12}}{r^{13}} - \frac{cR^5}{r^7} \right] \quad (1)$$

$F \leq F_{max}$ is the magnitude of the force between two robots i and j, and r is the distance between the two robots. R is the desired separation between robot i and all other neighboring robots. The variable ϵ affects the strength of the force, while c and d control the relative balance between the attractive and repulsive components. In order to achieve optimal behavior, the values of ϵ, c, d, and F_{max} must be determined. Our motivation for using the LJ force law is that (depending on the parameter settings) it can easily model crystalline solid formations, liquids, and even gases.

Optimizing Parameters Using Genetic Algorithms (Offline Approach): Given the generalized force law, such as LJ, it is necessary to optimize the parameters to achieve the best performance. We achieve this task using a genetic algorithm (GA). Genetic algorithms are optimization algorithms inspired by natural evolution. We mutate and recombine a population of candidate solutions (individuals) based on their performance (fitness) in our environment. The individuals that have higher fitness than the average fitness of the population will reproduce and contribute their genetic makeup to future generations. One of the major reasons for using this population-based stochastic algorithm is that it quickly generates individuals that have robust performance. Every individual in the population represents one instantiation of a force law.

Further discussion of the genetic algorithm needs clear definitions of the parameter sets used for the GA individuals. The evolving parameters of the LJ force law are:

- ϵ : strength of the robot-robot interactions.
- c : non-negative attractive robot-robot parameter.
- d : non-negative repulsive robot-robot parameter.
- F_{max} : maximum force of robot-robot interactions.

and similar 4-tuples for obstacle-robot and goal-robot interactions.

Offspring are generated using one-point crossover with a crossover rate of 60%. Mutation adds/subtracts an amount drawn from a N(0, δ) Gaussian distribution. Each parameter has a $1/L$ probability of being mutated, where L is the number of parameters. Mutation ensures that parameter values stay within accepted ranges. Since we are using an EA that minimizes, the performance of an individual is measured as a weighted sum of penalties.

$$Fitness = W_1 * CollPen + W_2 * CohPen + W_3 * GoalNotReachedPen \quad (2)$$

The weighted fitness function consists of three components, a penalty for collisions, a penalty for lack of cohesion, and a penalty for robots not reaching the goal. Since there is no safety zone around the obstacles [1], a penalty is added

to the score if the robots collide with obstacles. The cohesion penalty is derived from the fact that in a good hexagonal lattice, interior robots should have six local neighbors. A penalty occurs if a robot has more or less neighbors. If no robot reaches the goal within the time limit, a further penalty occurs.

Experimental Methodology of Offline Learning Module: Our 2D simulation is 900 × 700, and contains a goal, obstacles and robots. Up to a maximum of 100 robots and 100 static obstacles with one static goal are placed in the environment. The goal is always placed at a random position in the right side of the world, while the robots are initialized in the bottom left area. The obstacles are randomly distributed throughout the environment, but are kept 50 units away from the initial location of the robots, to give the robots the opportunity to first get into formation. Each circular obstacle has radius R_0 of 10, and the square shaped goal is 20 × 20.

When 100 obstacles are placed in the environment, roughly 5% of the environment is covered by the obstacles (similar to [1]). The desired separation between robots R is 16, and the maximum velocity V_{max} is 20. Figure 2 shows 40 robots navigating through randomly positioned obstacles. The larger circles are obstacles and the square to the right is the goal. Robots can sense other robots within the distance of $1.5R$, and can sense the obstacles within the distance of R_0+1 (minimum sensing distance). The goal can be sensed at any distance.

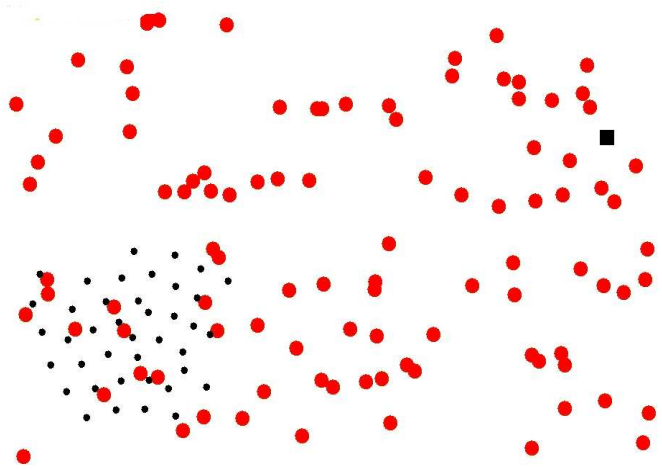

Fig. 2. 40 robots moving to the goal. The larger circle represent obstacles, while the square in the upper right represents the goal.

An LJ force law was evolved using the offline learning module of our simulation tool. The population size was 100 and the EA was run for 100 generations.

We trained over scenarios with 40 robots and 90 obstacles. Each individual (an instance of the force law) was evaluated for 1500 time steps, and averaged over 50 random instantiations of the environment.

Constraints with Offline Approach: The offline approach produces successful results [4], assuming that there is a global observer that assigns the fitness to individuals, the robots face scenarios in their current environment similar to the environment in which they were trained, and there are no communication delays or disruptions. In real world online applications, an offline approach can be problematic due to unexpected environment changes. We need an approach that allows the robots to rapidly adapt to the changing environment. We propose DAEDALUS, a fully distributed online approach that allows robots to adapt to the environment changes.

Online Approach with DAEDALUS: Each robot of the swarm is an individual in a population that interacts with its neighbors. Each robot contains a *slightly mutated* copy of the optimized force law rule set found with offline learning. This ensures that our robots are not completely homogeneous. We allowed this slight heterogeneity because when the environment changes, some mutations perform better than others. The robots that perform well in the environment will have higher fitness than the robots that perform poorly. When low fitness robots encounter high fitness robots, the low fitness robots ask for the high fitness robot's rules. Hence, better performing robots share their knowledge with their poorer performing neighbors.

When we apply DAEDALUS to obstacle avoidance, we focus on two aspects of our swarm: reducing obstacle-robot collisions and maintaining the cohesion of the swarm. Robots are penalized if they collide with obstacles and/or if they leave their neighbors behind. The second scenario arises when the robots are left behind in cul-de-sacs. This causes the cohesion of the formation to be reduced.

Experimental Methodology of Online Adaptation: Each robot of the swarm contains a slightly mutated copy of the optimized LJ force law rule set found with offline learning and all robots have the same fitness at the start. There are five goals to achieve in a long corridor, and between each randomly positioned goal is a different obstacle course with 90 randomly positioned obstacles. The online 2D world is 1650×950, which is larger than the offline world. In our changed environment, each obstacle has a radius of 30 compared to the offline obstacle radius of 10. So more than 16% of the online environment is covered with the obstacles. Compared to the offline environment, the online environment triples the obstacle coverage. We also increase the maximum velocity of the robots to 30 units/sec, making the robots moves 1.5 times faster than in the offline environment. The LJ force law learned in offline mode is not sufficient for this more difficult environment, producing collisions with obstacles (due to the higher velocity), and robots that never reach the goal (due to the high percentage of obstacles).

Robots that are left behind (due to obstacle cul-de-sacs) do not proceed to the next goal, but the robots that had collisions and made it to the goal are allowed to proceed to the next goal. We assume that damaged robots can be repaired once they reach a goal.

Results: To measure the performance of the DAEDALUS approach, an experiment was carried out with 60 robots, 5 goals in the long corridor, and 90 obstacles in between each goal. The experiment was averaged over 50 runs of different robot, goal, and obstacle placements. Each robot is given equal initial fitness and "seeded" with a mutated copy of the optimized LJ force law learned in offline mode. If a robot collides with an obstacle, it's fitness is reduced. Whenever a robot encounters another robot with higher fitness, it takes the relevant parameters pertaining to the obstacle-robot interaction of the better performing robot.

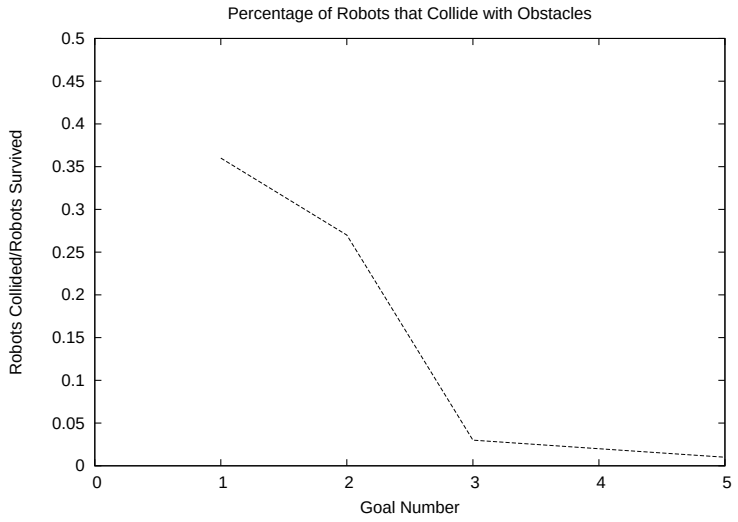

Fig. 3. The ratio of colliding robots versus the number of surviving robots, for 60 robots moving through 5 goals with 90 obstacles in between each goal.

Figure 3 shows the ratio of the number of robots that collided with obstacles versus the number of robots that survived to reach the goals. The graph indicates that after only 2 goals, the percentage of robots that collide with obstacles has dropped from about 36% to well under 5%. Inspection of the obstacle-robot parameters indicates that the repulsive component increased through the online process of mutation and the copying of superior force laws (this was confirmed via inspection of the mutated force laws).

This first experiment did not attempt to alleviate the situation where robots are left behind; in fact, only roughly 43% of the original 60 robots reach the final goal (see Figure 4, lower line). This is caused by the large number of cul-de-sacs produced by the large obstacle density. Our second experiment attempts to alleviate this problem by focusing on the robot-robot interactions. Our assumption was that the LJ force law needs to provide stronger cohesion, so that robots aren't left behind.

If robots are stuck behind in cul-de-sacs (i.e. they make no progress towards the goal) and they sense neighbors, they slightly mutate the robot-robot interaction parameters of their force laws. In a situation in which they do not sense the presence of neighbors and do not progress towards the goal, they rapidly mutate their robot-goal interaction causing a "panic behavior". These relatively large perturbations of the force law allow the robots to escape their motionless state.

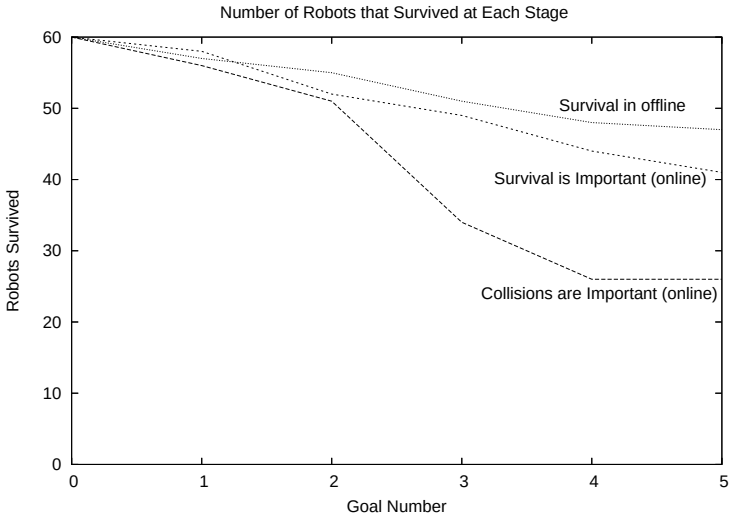

Fig. 4. A comparison of (a) the number of robots that survive when rules are learned using offline learning, (b) the number of robots that survive when using online learning (where the focus is on reducing collisions), and (c) the number of robots that survive when using online learning (and the focus in on survivability).

Figure 4 shows the results of this second experiment. In comparison with the first experiment (with survival rates of 43%), the survival rates have increased to 68%. As a control experiment, we ran our offline approach on this more difficult task. After five goals, the survival rate is about 78%. Recall that the offline results are obtained by running an EA with a population size of 100 for 100 generations, with each individual averaged over 50 random instantiations of the environment. As can be seen, the DAEDALUS approach provides results only

somewhat inferior to the offline approach, in real time, while the robots are in the environment.

Although not shown in the graph, it is important to point out that the collision rates were not affected in the second experiment. Hence, we believe that it is quite feasible to combine both aspects in the future. Collision avoidance can be improved via mutation of the obstacle-robot interaction, while survival can be improved via mutation of the robot-robot interaction and robot-goal interaction.

3.2 Evolution of Self Assembling Agents

For our second application we examine the self-assembly of robotic machines. These machines must determine their physical structures in order to consume some resource and output some product. A self-assembling machine may need to compete for resources and rapidly adapt in a changing environment. In this dynamic simulated environment, machines without an explicitly defined fitness function compete for survival. Each machine is slightly different from the others. Those machines that perform better send their "blue-prints" to those that perform worse. The poorer performing machines rebuild themselves according to the new blue-print. This study uses DAEDALUS to examine the issues in adapting L-Systems [5] to represent the different stages of development of these machines. We present our preliminary results of this on-going study.

In this simulation, a number of machines are placed within a virtual environment, where they must compete for three distinct resources. Acquisition of a sufficient quantity of the available resources determines survival. Two of these resources will be referred to as light and heat, and the third resource is simply space; machines are not allowed to overlap. Machines use the acquired resources as energy for survival and growth.

L-Systems Approach and Experimental Methodology: A machine's final structure is determined by a stochastic context-free bracketed L-system. An L-system is a system of string re-write rules similar to a Chomsky grammar [5]. The L-system is defined as an ordered triplet G = <V, w, P> where V is the alphabet, w is the axiom (or start symbol) and P is the set of production rules. As in any grammar, the start symbol is transformed through derivation steps involving the production rules, resulting in a new word at each step. In the stochastic L-system one or more production rules may have the same start symbol. A probability is attached to each start symbol and the probabilities for a set of production rules with equal start symbols sum to 1. During a derivation step, when a symbol with several rules is encountered a random value is generated to decide which rule to use.

A graphical interpretation of the word is presented at each step. The graphical representation is accomplished through turtle-graphics. Each letter in the alphabet is mapped to a turtle operation. We show an example L-system with the corresponding turtle interpretation of the alphabet used below.
V = {F, B, +, -, [,] }

w = F
P = {F (50%)→ FB, F (50%)→ F[+FB]F}
derivation example:
step 0: F
step 1: FB (at random the first F-rule was chosen)
step 2: F[+FB]FB (at random the second F-rule was chosen)

Turtle interpretation for the alphabet:
F = forward
B = blueprint
+ = turn right
- = turn left
[= push the turtle's current state onto a stack
] = pop the top of the stack into the current turtle

Initially, a set of machines is generated with minor random variations. At each time step machines may collect resources. The world is divided into columns representing the first resource, light. Any machine that is taller than all others in a column collects the light from that column for that time step. The second resource is distributed evenly by mass. Competition for space occurs naturally since machines may not overlap. At the end of each time step all machines are allowed to perform a growth step (L-System derivation), or to switch to an adult stage. The switch to adult stage occurs upon acquisition of a threshold amount of resource. The change arrests the L-System derivations and begins a propagation phase. Adult machines create copies of their genomes (blueprints) and distribute them in the environment. A machine constructor is assumed to exist. The constructor takes blueprints in pairs, recombines them, and builds the resulting machine with the current state of its L-System equal to its axiom.

A series of snapshots from an example run of the simulation is shown in Figure 5. In the first row we see a population of machines shortly after initialization. In the second row the right hand side of the environment is entirely controlled by a dense cluster of tall machines. The left hand side is relatively empty with only one small cluster of machines controlling any resources. The final two rows show the small population being overrun by the much taller machines as they react to the free resource area on the left. Throughout all frames we see the population adjusting its average height and structure for maximal energy absorption.

As mentioned above, the switch to adult stage occurs when the amount of resource acquired exceeds some threshold. We also used DAEDALUS to determine this threshold, as the environment changed. We assume that the machines must expend energy to acquire the resource. In some environments acquiring resources is inexpensive, while in others it is expensive.

Figure 6 illustrates the results. At first, the energy to operate per unit time is 0.1 (the units are arbitrary). Via mutation of the reproduction threshold parameter, machines evolve such that they reproduce when the threshold is approximately 15. There are a also a large number of machines (not shown in the

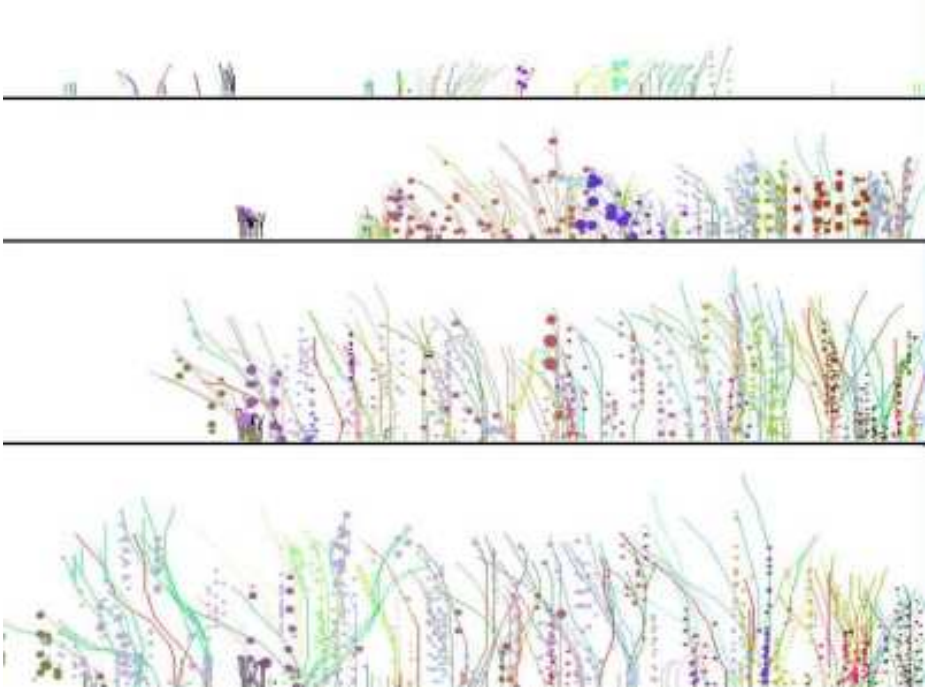

Fig. 5. Progress of Machines Competing for Resources.

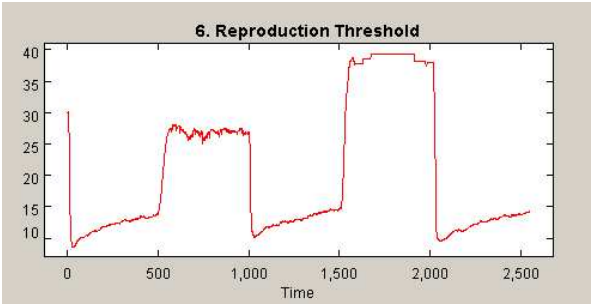

Fig. 6. The reproduction threshold depends on the difficulty in acquiring the resource.

graph). After 500 time units the operation energy is 0.5 (it is harder to acquire resources). Note that the threshold increases to roughly 27. There are fewer machines also. The operation energy decreases back to 0.1 and then increases to 0.9. In this case the threshold is very high, at approximately 39. There are also very few machines. Finally, the operation energy decreases back to 0.1.

The results are intuitively pleasing. First, in "easy" areas there are lots of

machines that reproduce often. In "hard" areas there are fewer machines that reproduce less often. Second, the results are reproducible in the sense that when the operation energy is 0.1, roughly the same threshold is evolved.

4 Conclusion

Traditional approaches to designing multi-agent systems are offline, and assume the presence of a global observer. However, this approach will not work in real-time online systems. This paper presents a novel approach to solving this problem, called DAEDALUS, where we show how concepts from population genetics can be used with swarms of agents to provide fast online adaptive learning in changing environments. Two case studies are used in this paper to illustrate the feasibility of this approach.

Future work will focus more on the issue of credit assignment. Current work in classifier systems uses mechanisms such as "bucket-brigade" or "profit sharing" to allocate rewards to individual "agents" appropriately [3] . However, these techniques rely on global blackboards and assume that all agents can potentially act with all others, through a bidding process. We intend to modify these approaches so that they are fully distributed, and appropriate for online systems.

References

1. Balch, T., Hybinette, M.: Social Potentials for Scalable Multi-Robot Formations. IEEE International Conference on Robotics and Automation. (2000)
2. Grefenstette, J.: A system for learning control strategies with genetic algorithms. Proceedings of the Third International Conference on Genetic Algorithms. Morgan Kaufmann (1989) 183–190
3. Grefenstette, J.: Credit Assignment in Rule Discovery Systems Based on Genetic Algorithms. Springer-Verlag **3** (1988) 225–245
4. Hettiarachchi, S., Spears, W.: Moving Swarm Formations Through Obstacle Fields. International Conference on Artificial Intelligence. CSREA Press **1** (2005) 97–103
5. Prusinkiewicz, P., Lindenmayer, A.: The Algorithmic Beauty of Plants. Springer-Verlag. (2004)
6. Spears, W.: Simple Subpopulation Schemes. Proceedings of the Evolutionary Programming Conference. World Scientific (1994) 296–307
7. Spears, W., Spears, D., Hamann, J., Heil, R.: Distributed, Physics-Based Control of Swarm of Vehicles. Autonomous Robots. Kluwer **17** (2004) 137–164
8. Watson, R., Ficici, S., Pollack, J.: Embodied Evolution: Distributing an evolutionary algorithm in a population of robots. Robotics and Autonomous Systems. Elsevier **39** (2002) 1–18
9. Wu, A., Schultz, A., Agah, A.: Evolving control for distributed micro air vehicles. Proceedings of the IEEE International Symposium on Computational Intelligence in Robotics and Automation. IEEE Press (1999) 174–179