Adaptivity in Cell Based Optimization for Information Ecosystems

Joseph A. Rothermich
Icosystem Corporation
10 Fawcett Street
Cambridge, MA 02138
jr@icosystem.com

Fang Wang
Intelligent Systems Laboratory,
BTexact
Adastral Park, Ipswich IP5 3RE
United Kingdom
fang.wang@bt.com

Julian F. Miller
School of Computer Science
University of Birmingham
Edgbaston, Birmingham
B15 2TT, United Kingdom
j.miller@cs.bham.ac.uk

Abstract. **A Cell Based Optimization (CBO) algorithm is proposed which takes inspiration from the collective behavior of Cellular Slime Molds *(Dictyostelium discoideum)*. Experiments with CBO are conducted to study the ability of simple cell-like agents to collectively manage resources across a distributed network. Cells, or agents, only have local information and can signal, move, divide, and die. Heterogeneous populations of the cells are evolved using Cartesian Genetic Programming (CGP). Several Experiments were carried out to examine the adaptation of cells to changing user demand patterns. CBO performance was compared using various methods to change demand. The experiments showed that populations consistently evolve to produce effective solutions. The populations produce better solutions when user demand patterns fluctuated over time instead of environments with static demand. This is a surprising result that shows that populations need to be challenged during the evolutionary process to produce good results.**

1 Introduction

Information Systems are becoming increasingly complex and dynamic. One example of this trend is the Internet. The Internet can be described as an Information Ecosystem where a complex web of information producers and information consumers interact in a constantly growing and changing environment (Marrow, et al., 2001). As complexity continues to increase, it becomes difficult to continue building systems using traditional design methods. Instead of the current centralized top-down methods, bottom-up approaches must be explored. Bottom-up methodologies start the design process with simple components and work towards a more complex whole. An example of a bottom-up process in nature is the emergent behavior in ant colonies, which are able to perform complex tasks even though the components, the ants themselves, are simple.

One problem with designing systems this way is that humans have a hard time understanding bottom-up systems. Humans have a centralized mind-set (Resnick, 1994), which often forces us to see systems using a centralized organization. Once systems become decentralized, it is hard for humans to understand how their components work together to perform its complex functionality. It is even harder for us to design this behavior. One way that we may be better able to understand decentralized systems is looking towards nature. Swarming insects such as ants and bees have decentralized functionality. Another example is Cellular Slime Molds *(Dictyostelium discoideum)*. Slime Molds spend most of their life as single celled organisms but occasionally exhibit behaviors associated with multi-celled organisms.

When Slime Mold cells begin to deplete their local food source, they aggregate to form a slug-like creature that helps relocate the population to a more fruitful environment. This is done via cell-cell signaling combined with a positive feedback mechanism. It was once thought that there was a leader for this aggregation but it is now known to be a decentralized process (Keller & Segel, 1970). The emergent behavior of Slime Molds is an example of order arising from simple decentralized parts. Their collective behavior has been demonstrated by showing that they can find the shortest path through a maze (Nakagaki et al., 2000).

This paper aims to test a model based on the behaviors of Cellular Slime Molds applied to an optimization problem in the Information Ecosystem domain. The Cell-Based Optimization (CBO) model was extended from a previous research project (Rothermich & Miller, 2002) to include simulated user requests and their processing by cell-like agents. Several experiments are conducted to study the ability of evolving populations of agents to effectively treat user requests in a changing environment.

There has been a large amount of study in the application of ideas from nature to computer science. Ant Systems (AS) and Ant Colony Optimization (ACO) (Dorigo & Gambardella, 1997; Dorigo & Di Caro, 1999) have been applied to optimization problems such as routing and scheduling. Previous cell-based models have been created by Agarwal (1995), Fleischer and Barr (1992), Ray (1994), Resnick (1994). This paper differs in that cell behavior is autonomously evolved through a form of genetic programming called Cartesian Genetic Programming (CGP) and in its application to an optimization problem.

Multi-agent systems have been used in Information Ecosystems in research such as InfoSleuth (Bayardo et al., 1997), Amalthea (Moukas, 1997), and the InfoSpiders project (Menczer & Monge, 1999). These projects differ in that they often focus on search and

identification of information instead of allocation of resources. There is a large amount of research in distributed load balancing optimization (Chavez et al., 1997; Schaerf, Shoham, & Tennenholtz, 1995; Pulidas, Towsley, & Stankovic, 1988). These projects consider the capacity at each location to be fixed, while the jobs that need to be performed can be shifted to different locations. This is the inverse of the problem faced in this paper since the user requests are generated at a certain database and cannot be transferred. The capacity, i.e., the agents, can however be transferred to different locations.

This paper proposes Cell-Based Optimization (CBO) to maximize the processing of user requests in a distributed information ecosystem. A set of simple information agents is utilized to retrieve information for users, each of which is equipped with a cell-based model. Without full knowledge of the ecosystem, cell-like agents are evolved to serve user requests in local environments or migrate to remote environments searching for additional requests. The collective behavior of the agents results in timely request processing, which is adaptive to changing user demands. This model has been implemented in a Java application and also within the DIET (Decentralized Information Ecosystem Technologies) Platform (Hoile et al., 2002). This is a mobile agent toolkit that creates a foundation for building scalable simulations and applications of interacting, lightweight Infohabitants (entities that can process information).

The remaining sections of this paper are organized as follows. The next section introduces the design of the cell-based model used for the creation of agents. Section 3 explains the resource optimization problem in information ecosystems and how cell-like agents are used to solve this problem. Experimental results are shown in Section 4. The last section draws conclusions from the experimental results and proposes future work.

2 Cell-based model design

Similar to Slime Mold Cells, cell-like agents have inputs and outputs, and various functions. This model does not intend to create a detailed model of natural cells, instead the abilities of cells are abstracted at a high level. The internal processing of agents is controlled by statements generated through evolution and the use of Genetic Programming.

2.1 Cell Inputs

Natural cells have the ability to sense and absorb chemicals from their environment. They can either absorb chemicals directly, or the cell can accumulate the chemical in a receptor that only allows chemicals to be absorbed after a certain threshold has been passed. A set of imaginary chemicals was used in the cell-based model. One chemical represented an energy source for the cells. The other chemical was emitted by the cells themselves, simulating cell-cell signaling.

Real cells do not have any information about their location. They only know what chemicals are present around them and the concentration gradients for each. The cell-based model followed a similar approach where cells do not know their location or neighbors but can sense the chemicals being emitted by them.

A threshold parameter similar to receptors in nature was used for cell inputs. This allowed the sensitivity of the cells to vary. Cells in the cell-based model also have the internal input of knowing how much energy they have. This means the actions they choose might be dependent upon their current health.

2.2 Cell Functions

2.2.1 Movement

Cells have the ability to move towards or away from chemical gradients, called chemotaxis. The cells created by the cell-based model have similar functionality. They can be programmed to move in relation to gradients of simulated chemicals. A small degree of randomness is also added to the direction of the cell's movement.

2.2.2 Cell Division

The cells in the cell-based model also have the ability to divide. Once a cell divides, the cell itself no longer exists and is replaced by two offspring. Each offspring has half of the energy of the original cell plus a small energy cost for the division process. The offspring inherit the genes of the parent cell plus some possible mutation.

2.2.3 Chemical Signaling

Cells have the ability to release a simulated chemical into the environment. This gives the cells a way to communicate with each other similar to the way that Cellular Slime Molds signal with chemical signals, called cyclic AMP, during their aggregation phase.

2.3 Cell evolution and Genotype Representation

2.3.1 Cartesian Genetic Programming

The cell-based model utilizes Cartesian Genetic Programming (CGP) to evolve cell functions. A genotype in Cartesian Genetic Programming is an integer string that encodes an indexed, feed forward, acyclic graph (Miller and Thomson, 2000). Unlike the parse tree representation in the standard GP (Koza, 1992), a genotype-phenotype mapping is used to create the graph phenotype from the integer string genotype. Each node in the genotype contains two types of genes: connection genes that represent how the inputs to the node are connected to program inputs or the outputs of other nodes, and a function gene that represents the operation that the node carries out on the inputs it receives. The

nodes that are not involved in the linked path between the inputs and outputs of the program are inactive in the phenotype. Such nodes have no effect on the behavior of the phenotype. However a point mutation operator can re-connect inactive nodes or disconnect active ones. This allows neutral drift to take place. This has been shown elsewhere to be extremely beneficial to the search process for a number of problems (Miller and Thomson, 2000, Vassilev and Miller, 2000, Yu and Miller, 2001).

2.3.2 Cell-Based CGP Representation

As in standard CGP, the genome consists of a number of genes that are linked to form a graph. The right-most gene is executed first, and each gene can be connected to any other gene on its left. The final gene on the left is always a 'do nothing' function. During the decoding of the phenotype, if no action has been reached while traversing the graph, then eventually the left-most 'do nothing' gene will be called and the cell will not perform an action. This case occurs less often as the genotype length parameter is increased. In the following optimization experiments, an empirical genome length of 20 genes was used. This genome allowed for complex behaviors (actions based on nested conditions) with a reasonable length.

Each gene in the CGP genome consists of an array of four numbers (e.g., 1012), representing the function (1) of the gene, an external chemical input (2) of the gene, and two connections (3) and (4) of the gene with other genes in the genome. A pictorial illustration of a single gene is shown in Figure 1. The node number in Figure 1 is an identifier used to indicate the location of a gene in a genome.

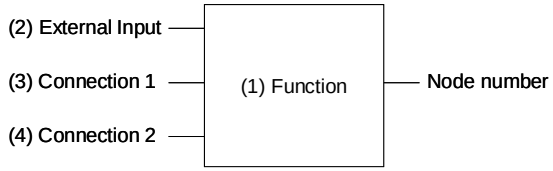

Figure 1: A Single Gene In The CGP Representation

In order to maintain the basic actions of Slime Mold Cells, a series of functions are designed including conditional and unconditional functions. The conditional functions provide the cells with the ability to act according to external stimuli or internal conditions such as a cell energy level. For the information-retrieving agents in information ecosystems, we use the functions shown in Table 1. These actions provide the basic functions and conditions of Slime Mold cells. As an example, if a gene is coded with function one, it will move towards the chemical referenced in the external input (2). If a cell is coded with function three, the cell will check for the presence of the chemical referenced in (2) and then execute the gene referenced in the connection (3) or (4) accordingly. The parameters UPPER_THRESHOLD and LOWER_THRESHOLD used in functions five and six are local variables for each cell that are considered part of the evolving genotype.

Table 1: Cell Functions

Function	Description
0	Do Nothing
1	Move towards (2)
2	Move away from (2)
3	If (2) is present do (3), else do (4)
4	If (2) is present do (4), else do (3)
5	If energy is above UPPER_THRESHOLD do (3), else do (4)
6	If energy is below LOWER_THRESHOLD do (3), else do (4)
7	Perform (3) then divide
8	Perform (3) then release chemical signal

In order to fully explain the decoding process for a phenotype, it is helpful to give an example. Figure 2 shows a sample genotype with five genes. The positions containing functions are underlined. The same genotype can be represented pictorially as shown in Figure 3.

```
   0    1    2    3    4    5
|0000|7100|1010|1012|3100|3024|
```

Figure 2: Example Cell Genotype In CGP

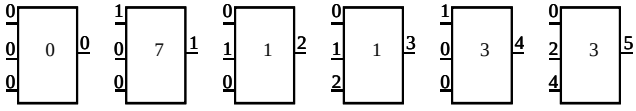

Figure 3: Pictorial Representation Of Example Genotype

```
If Chemical 0 is present
     Move towards Chemical 0
Else
     If Chemical 1 is present
            Do nothing
     Else
            Do nothing
```

Figure 4: Example Decoded CGP Cell Phenotype

The genotype is mapped to a phenotype by traversing the graph starting at the right-most gene. The function assigned to gene five is function 3: *If (2) is present, do (3) else do (4).* In this model, a zero represents the energy chemical and a one represents the chemical signal emitted from cells. The value in position (2) is zero, so if the energy chemical is present, decode and perform the gene identified by position (3): which is gene number two. Otherwise, perform the gene referenced in position (4): which is gene number four. This process continues until an action is found or the genotype has been

completely decoded without an action, which would result in a "Do Nothing" function.

The fully decoded phenotype for this example describes the cell's behavior and is a series of IF-ELSE statements. The phenotype is shown in Figure 4.

2.4 Evolution

2.4.1 Genetic Operators

When a cell divides, one of its offspring is mutated and the other inherits the parent's genes exactly. This allows for a type of elitism so that well conditioned genotypes are not lost in the next generation.

A first mutation consists of randomly changing one of the integer values in the genotype's CGP string. Only one value is changed per mutation. A mutation can modify the function, chemical input, or connections in a gene. Since only a portion of the genotype is decoded into the phenotype, mutations often do not affect the behavior of a cell.

A second mutation was implemented by adding or subtracting a random number to the upper and lower energy threshold variables. If thresholds overlap, e.g., the upper is lower than the lower threshold, the model still operates since thresholds are used in separate conditional functions.

2.4.2 Cell Population and Fitness Evaluation

A random population of cells is created for each execution. The initial, maximum and minimum number of cells in a population can be controlled through program parameters. If the population size falls below the minimum, new random cells are be created. When the maximum population size is reached, cells are no longer permitted to divide. Cells need to compete or cooperate for a limited amount of resources from the environment. If cells are not successful in getting energy, they die.

Fitness is not explicitly measured, but instead fit individuals are those that survive by collecting energy and/or passed on their genes to future generations. Therefore the concepts of fitness evaluations and generations from typical Evolutionary Algorithms are not needed in this model.

3 Test Design

3.1 Problem Definition

In an Information Ecosystem, databases or information services are distributed throughout a network. At each location in the network, users make requests, or queries, for information from these services. An appropriate number of agents are required at each location to treat a demand level for a database. The problem is shown pictorially in Figure 5.

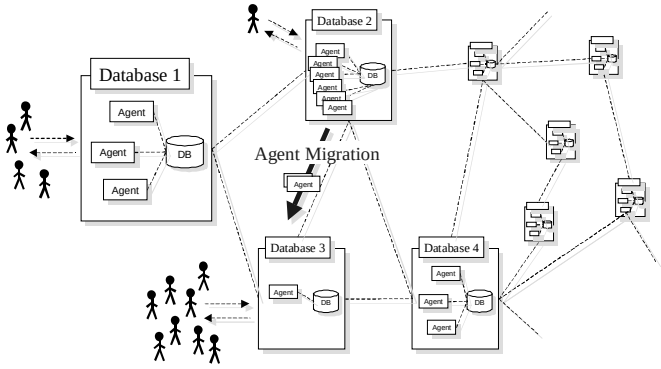

Figure 5: Resource Allocation Problem

As shown in Figure 5, several databases (1, 2, 3, …N) are distributed across a network. Assume each user makes one request per time-period, so Database 1 will have an average of four user requests per time-period; Database 2 will have an average of one request; and Database 3 an average of eight. There are six agents at Database 2 but only one user requesting service. Database 3 has a shortage of agents. The figure shows agents migrating from Database 2 to Database 3 to help balance out this discrepancy.

The task of allocating agents is easy to solve with a global knowledge of resources and demand across the network. However in open distributed networks, usually only local knowledge is known. The central maintenance of this information would not only prevent scalability and create risks of single points of failure, but may be impossible in a truly distributed network such as the Internet.

The problem is made more complicated in that the user demand may not always stay constant. An example of this would be web sites that are popular because of a current trend or news story but then fade in popularity after the topic has become outdated.

The problem can be stated as this: Can a group of information processing agents collectively distribute themselves across a network to maximize the processing of user requests given the following?

- The agents only have local information.
- The user demand patterns may or may not change with time
- The number of agents should be appropriate for the level of demand (i.e., there is a limited capacity for agents in the system).

3.2 Test design

3.2.1 Test Approach

The cell-based model introduced in Section 2 is utilized to create cell-like agents to solve the resource management problem discussed above. Cell-like agents can process one user request per time-step. Databases signal the number of untreated local requests as a chemical. The level of chemical at a single location is

equal to the number of outstanding local requests, plus the amount of chemical that has diffused from neighbor locations. A neighbor's chemical level indicates the level of unfulfilled demand at that location as well as the chemical it has received from its neighbor's locations. Since chemicals diffuse throughout the web of connected databases, the chemical signal is decreased as distance from the source database increases. Agents gain energy by processing user requests but lose energy continuously for other activities (e.g., migration) or doing nothing (e.g., their metabolism).

Tests were conducted using 50 trials per experiment. After several informal tests, 7,000 time-steps were determined to be an adequate experiment length to ensure completeness.

The network is represented as a grid pattern, however any type of network connectivity, including random connections, can be used. Each database is connected to its four adjacent neighbors. A ten by ten sized grid creates a network of 100 databases. This size is chosen so that performance is acceptable and there are enough databases to present a challenge to manage resources.

An average of ten new user requests per time-step are generated at each database. Using a Poisson distribution, λ represents the average number of new requests at each database per time-step. A unique λ is assigned for each database by using a Poisson distribution with mean of 10. Tests are performed using an initial population of 1000 randomly programmed cells. Each cell starts with 10 units of energy and 0.3 units are deducted during each time-step. Cells receive one unit of energy for processing a user request. The cells have 15% randomness in their direction of movement and have no polarity. This set of parameters provided reasonable results during several informal tests; it was decided that this set of parameters used across all experiments would serve as a consistent basis for experimentation.

Although the model is also implemented within the DIET platform, the experiments in this paper were performed in a separate stand-alone custom Java application. This application took advantage of previous work including visualization and the collection of metrics. However, unlike DIET, it does not scale well for large networks.

3.2.2 Experiments Conducted

A comparison is made between hard-coded cell designs and evolved designs. This provides a way to assess the performance of the evolving populations versus a human-designed baseline solution. The stability of evolved solutions is judged by monitoring the completed user requests across experiments.

Adaptability is assessed by monitoring the performance of cell populations when user demand patterns vary over time. Three methods are used to test the effects of changes in demand patterns over time:

Simultaneous Shift in Demand. One set of tests is carried out using a simultaneous shift in demand. Every 500 time-steps, the values for λ in the Poisson distribution are globally reassigned. The reassignment is carried out instantaneously across all locations. The global system demand remains the same overall, but demand within each environment changes.

Gradual Shift in Demand. Another test conducted is to change the demand patterns gradually. This approach still resets each environment's λ every 500 time-steps, but the new λ' is updated gradually. Each time-step after the global demand distribution is reset, a number of environments are called to increment or decrement their current λ in the direction of the new value λ'. This continues until the value λ in each environment equals its λ'.

Fixed Demand. Some tests are also performed using a fixed demand pattern for the duration of the test. This does not test the system's ability to be adaptive, but instead tested the ability of a population to learn the best solution for a constant demand.

4 Results

4.1 Population Dynamics

Populations of cells consistently evolve solutions with stable population sizes. Figure 6 shows experimental results using the immediate shift in demand method with 1,000 new requests being generated on average throughout the network per time-step.

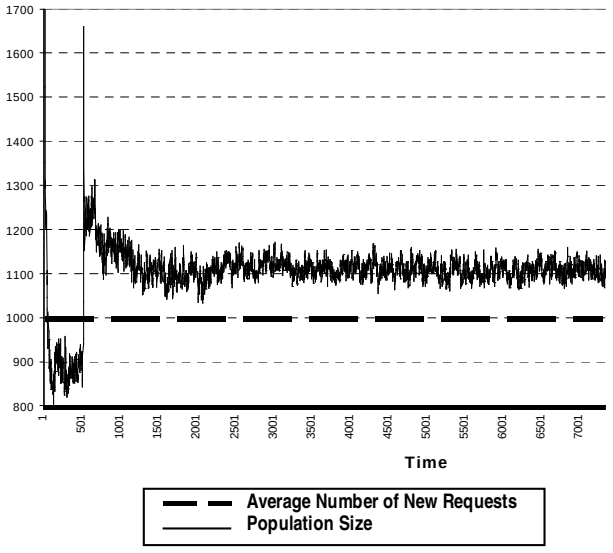

Figure 6: Population Size during 7,000 Time-steps

The population size stabilizes at around 1,100 cells and an average of 87.83% of user requests are completed. During an experiment, the number of cells usually varies greatly during the first 1,000 time-steps but then stabilizes for the remainder of the tests, even as demand

patterns continues to be reassigned every 500 steps. The average population size of 1,100 is a reasonable number of cells to handle the 1,000 new requests created on average at each time-step throughout the network. The average population size is 10% greater than the number of new requests. Completing 87.83% of user requests with a stable population size proves the ability of cells to optimize their performance.

4.2 Evolution versus Human Design

The CBO Algorithm is compared against several human designed agents. Many designs for agents were considered and tested. The best performing program is one where cells move towards the food source and divide when they are healthy (i.e., when there is food nearby). The program for this behavior was 'If energy is greater than 30, divide. Else, move towards the food chemical'.

As shown in Table 2, during 50 trials using the gradual shift in demand method, the evolved solutions perform better than the human designed solution. The evolved solution performs worse than human designed solutions during the initial time-steps of the program, but performs better after a few hundred time-steps.

Table 2: Average % Complete – Evolved vs. Human Design
(Average of 50 trials between t=5000 and t=7000)

	EVOLVED SOLUTION	HUMAN DESIGNED SOLUTION
AVERAGE % COMPLETE	93.74%	42.00%
STANDARD DEVIATION	0.07	0.03

An advantage that the evolved design has is that its population is heterogeneous. The human design is a homogeneous population of cells, all sharing the same behavior. Although it is easy to imagine cell programs that might work efficiently, it would be hard for a human designer to create a diverse population that can work collectively to allocate resources. In the evolved solution, certain cell groups evolve to play diverse roles. Some cells might be greedy (always following the database chemical), while others might be lazy (only following the database chemical when they are running out of energy). Also, some cells may even help the overall system's effectiveness by moving away from the database chemical. This could possibly help the population spread itself out in the network instead of cells clustering where demand is strong.

Planned heterogeneity could be tested by hard-coding groups of human designed cells. However, it would be difficult to determine how many roles are required, the proportions of cells in each role and specific thresholds each role should have in their programs. This type of design would probably have to be 'manually evolved' through sample runs and would not be adaptive to new types of networks and demand patterns. The benefit of naturally heterogeneous systems is discussed by Kephart et al. (1980) in their studies of computational ecosystems. They found that heterogeneity could create stability and that trying to design too much sophistication at the agent level lead to oscillations and chaos.

4.3 Adaptation

The adaptability of cell populations is tested by monitoring performance when patterns of user demand are shifted. The three methods for shifting demand mentioned in Section 3.2.2 are used for comparisons. The anticipated outcome was that the immediate, simultaneous shift in demand would have the worst performance, the gradual shift better, and no shift in demand would be the best.

The rationale for the anticipated result is as follows. An immediate shift in demand would require that cells be the most adaptive and respond to a change in demand as quickly as possible. A gradual demand shift would require the cells to become adaptive, but would be slightly more forgiving if the population was slow to adapt to the new pattern. Since the new demand pattern takes effect gradually, cells would have a generous amount of time to die off, divide or move to where demand was growing.

If there were no shift in demand, thus a constant demand pattern throughout the test, the cells would not have to adapt their learned behavior at all. It was assumed that this would be the easiest environment and would result in having the shortest user wait times.

The results of these tests are surprising. The anticipated comparison for gradual and immediate demand shift is correct. However, the anticipated result for tests with a constant demand level is false. The results for 50 trials are listed in Table 3.

The populations existing in a network without a shift in demand perform worse than both types of shifting demand tests. During the first 1,000 time-steps, the populations do perform better when there was a fixed demand pattern. However, during the rest of the experiments, they perform worse.

Table 3: Average % Complete – Different Methods for Shifting Demand
(Average of 50 trials between t=5000 and t=7000)

	IMMEDIATE SHIFT	GRADUAL SHIFT	NO SHIFT (CONSTANT DEMAND)
AVERAGE % COMPLETE	87.83%	93.74%	81.08%
STANDARD DEVIATION	0.11	0.07	0.11

It seems that forcing cells to be adaptive help them to eventually have better performance. Usually after a shift

in demand, there is a high level of cell death. After a couple of hundred time-steps stability is again reached. However, after several episodes of demand shifts, cells do not have a performance drop during a demand shift because they have evolved to be adaptive. It is possible that mediocre groups of cells that perform poorly are able to survive in an easier, constant environment whereas they may be made extinct otherwise.

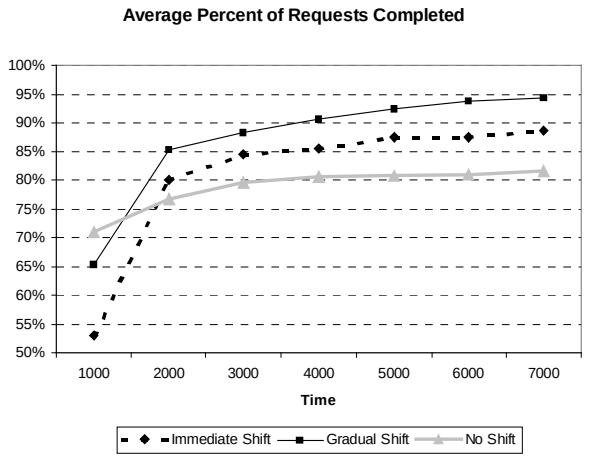

Figure 7: Comparison of Types of Shift in Demand

Figure 7 shows the average performance of the populations throughout different time-periods of the experiments. Cells in an easier environment only evolve to have acceptable performance, whereas cells in a constantly changing environment have to compete and find a new energy source every 500 time-steps. This competition may force them to evolve better solutions, which not only improve individual cell performance, but also the performance of the population.

5 Conclusions and future work

The goal of this research is to test the capabilities of a Cell Based Optimization algorithm at solving problems in an Information Ecosystem domain. Motivation for this work is to better understand bottom-up systems design and emergent behaviors in complex decentralized systems. The CBO algorithm uses cell-like functions and reactions to simulated chemicals in an artificial ecosystem. Cartesian Genetic Programming is used for genetic representation. The Cell Based Optimization algorithm is able to demonstrate successful group allocation across a network, maximizing the processing of simulated user requests.

Populations of randomly generated cells, or agents, are able to collectively manage their population size and their distribution throughout a network. This functionality is evolved without having to specify fitness functions or manage selection criteria. By making the treatment of user requests a requirement for cell survival (i.e., their energy source), the cells evolve to allocate themselves according to changing patterns in user demand. Each cell is autonomous and only has information about the chemicals in their local environment. Although each cell acts only to serve itself (i.e., survive), group functionality emerges so that the populations of cells collectively allocate themselves across the network. They also produce a stable and robust solution that is adaptive to changes in user demand patterns.

The experiments result in a heterogeneous population with cells serving varied roles. Some cells are greedy and always follow a signal for user demand. Other cells stay in one place if there is enough requests for it to survive. The evolved heterogeneous populations prove to be more effective than populations created by hand. On average, homogenous populations of human designed agents are not able to complete half of the user requests at each time-step. Their evolved heterogeneous counterparts average completion percentages above 90 percent. It was argued that it would be very difficult to design effective heterogeneous populations by hand.

When testing the adaptability of the evolved solutions, it is found that more challenging environments help to evolve more effective populations of cells. Cells that have to compete in an environment with changing user demand patterns are more successful in the long run then cells competing in static environments.

There are many extensions to this project that would serve for interesting research. The comparisons of evolved solutions to human designed cells could be furthered by attempting to create human designed heterogeneous populations for comparison. To prevent ‘manual evolution’, some rules would need to be enforced to define how much interaction a designer could have with the system before designing the cells. Also, the evolved populations could be studied in more depth to more precisely identify roles that the cells played and proportions of the populations in each group.

The cell-based model could be adapted to other problems. It would be interesting to see how Cell Based Optimization performs compared with more traditional techniques and other nature-inspired techniques (e.g., Ant Colony Optimization).

Currently the cells in the model consume energy at each time-step and when they divide. Additional functionality could be added so that other activities cost the cells energy. Examples include charging a cell for migration, signaling, retrieving local information, etc. This might make cells evolve to decrease the load on the system while at the same time evolve to treat user requests. An ecosystem model may be an interesting test-bed for this type of multi-objective problem.

Acknowledgements

This work was supported by the Future Technologies Group in BTexact and the DIET (Decentralised Information Ecosystems Technologies) project (IST-1999-10088) within the Universal Information Ecosystems initiative of the Information Society Technology Programme of the European Union. DIET is

also partly supported by the Enterprise Venturing Programme of BTexact Technologies. The authors are grateful to Paul Marrow, Cefn Hoile, Erwin Bonsma, and Mark Shackleton for their useful comments and helpful assistance throughout the work.

Bibliography

Agarwal, Pankaj (1995). The cell programming language. *Artificial Life*, 2 (1):3777.

Bayardo, R. J., Bohrer, W., Bric, R. et al. Infosleuth: Agent-based semantic integration of information in open and dynamic environments. In: *ACM SIGMOD*, pages 195—206, 1997.

Chavez A., Moukas A., Maes P. (1997) Challenger: A Multi-Agent System for Distributed Resource Allocation, In: *Proceedings of the First International Conference on Autonomous Agents*, Marina Del Ray, CA.

Dorigo M., Gambardella, L.M. (1997). Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem. *IEEE Transactions on Evolutionary Computation*. 1, 53—66.

Dorigo, M. Di Caro, G. (1999). The Ant Colony Optimization MetaHeuristic, in: D. Corne, M. Dorigo and F. Glover, eds, *New Ideas in Optimization*, McGraw-Hill.

Fleischer, Kurt and Barr, Alan H. (1992) A simulation testbed for the study of multicellular development: The multiple mechanisms of morphogenesis. In C. Langton (ed), *Artificial Life III*, London: Addison-Wesley, pp. 389-416.

Hoile, C., Wang, F., Bonsma, E., and Marrow, P. (2002) Core specification and experiments in DIET: a decentralised ecosystem-inspired mobile agent system In: *Proceedings of the 1st International Conference on Autonomous Agents and Multi-Agent Systems*, AAMAS2002, Bologna, Italy.

Keller, E. and Segel, L. (1970). Initiation of slime mold aggregation viewed as an instability, *Journal of Theoretical Biology*. 26, pp. 399-415.

Kephart, J. O., Hogg, Tag, and Huberman, Bernardo A. (1989). Dynamics of computational ecosystems. *Physical Review A*, 40.

Koza, John R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge, MA: MIT Press.

Marrow, P., Koubarakis, M., van Lengen, R.H., Valverde-Albacete, F., et al. (2001). Agents in Decentralised Information Ecosytems: the DIET Approach, *Proceedings of the AISB'01 Symposium on Information Agents for Electronic Commerce*. York, UK. pp. 109-117.

Menczer, F. and Monge, A. E. (1999). Scalable web search by adaptive online agents: An infospiders case study. In *Intelligent Information Agents*. Springer.

Miller, Julian and Thomson, Peter (2000). Cartesian Genetic Programming. In R. Poli, J.F. Miller, W. Banzhaf, W.B. Langdon, J.F. Miller, P. Nordin, T.C. Fogarty (eds), *Proceedings of the 3rd International Conference on Genetic Programming (EuroGP2000)*, Lecture Notes in Computer Science, Berlin: Springer-Verlag, Vol. 1802, pp. 15-17.

Moukas, A. (1996) Amalthaea: information discovery and filtering using a multiagent evolving ecosystem. In: *Proceedings of PAAM96*.

Nakagaki, T.; Yamada, H.; Tóth, Á. (2000). Intelligence: Maze-solving by an amoeboid organism, *Nature* 407, 470.

Pulidas, S, Towsley D., and Stankovic, J.A. (1988). Imbedding gradient estimators in load balancing algorithms. *In 8th International Conference on Distributed Computing Systems*, pp. 482—490.

Ray, Thomas S. (1994). An evolutionary approach to synthetic biology: Zen and the art of creating life. *Artificial Life*, 1(1/2), pp. 179-209.

Resnick, Mitchel (1994). *Turtles, Termites, and Traffic Jams: Explorations in Massively Parallel Microworlds*. Cambridge, MA: MIT Press.

Rothermich, J. (2002). *From Multicellularity to Cell Based Optimization: Studying the Cooperative Capabilities of Evolvable Cells.* MSc Thesis. University of Birmingham, UK.

Rothermich, J., Miller, J. (2002). Studying the Emergence of Multicellularity with Cartesian Genetic Programming in Artificial Life. *Proceedings of the UK Workshop on Computational Intelligence (UKCI-02)*, Birmingham, UK.

Schaerf, A., Shoham, Y., and Tennenholtz, M. (1995). Adaptive load balancing: A study in multi-agent learning. *Artificial Intelligence Research* 2, 475—500.

Vassilev, Vesselin K. and Miller, Julian F. (2000) The advantages of Landscape Neutrality in Digital Circuit Evolution. In J.F. Miller , A. Thompson, P. Thomson, and T. Fogarty (eds), *Proceedings of the 3rd International Conference on Evolvable Systems: From Biology to Hardware (ICES2000)*, Lecture Notes in Computer Science, Berlin: Springer-Verlag, Vol. 1801, pp. 252-263.

Yu, Tina and Miller Julian F. (2001) Neutrality and Evolvability of a Boolean Function Landscape. In J.F. Miller, M. Tomassini, P. L. Lanzi, C. Ryan, W. Langdon (eds), *Proceedings of the 4th International Conference on Genetic Programming (EuroGP2001)*, Lecture Notes in Computer Science, Berlin: Springer-Verlag, Vol. 2038, pp. 204-217.