# Modeling Gene-Regulatory Networks using Evolutionary Algorithms and Distributed Computing

**Martin Swain**
University of Ulster
Northern Ireland
*mt.swain@ulster.ac.uk*

**Thomas Hunniford**
University of Ulster
Northern Ireland
*tjc.hunniford@ulster.ac.uk*

**Johannes Mandel**
Fachhochschule Weihenstephan,
Germany and University of Ulster,
Northern Ireland
*johannes.mandel@fh-weihenstephan.de*

**Niall Palfreyman**
Fachhochschule Weihenstephan, Germany
*niall.palfreyman@fh-weihenstephan.de*

**Werner Dubitzky**
Univeristy of Ulster, Northern Ireland
*w.dubitzky@ulster.ac.uk*

**Abstract –** *Living organisms regulate the expression of genes using complex interactions of transcription factors, messenger RNA and active protein products. Due to their complexity,* gene-regulatory networks *are not fully understood, however, various modeling approaches can be used to gain insight into their function and operation. This paper describes an on-going study to use* evolutionary algorithms *to create computational models of gene-regulatory networks based on observed microarray data. Because of the computational requirements of this approach (which requires the discovery of gene network topologies), it is critical that it is implemented on a computing platform capable of delivering significant compute power. We discuss how this can be achieved using distributed and grid computing technology. In particular we investigate how Condor and JavaSpaces technology is suited to the requirements of our modeling approach.*

**Keywords:** Evolutionary algorithms, gene-regulatory networks, grid computing, distributed computing.

## 1 Introduction

In order to model the structure and dynamics of entire biological systems, systems biology focuses on analyzing the interactions between biological components in a holistic fashion. It is these interactions that give rise to an organism's complex dynamical behavior. *Gene-regulatory networks* (GRNs) are important for understanding such complexity. Within a GRN, genes and their products interact with one another – genes code for proteins that may in turn regulate the expression of other genes. The importance of gene expression networks is perhaps best understood by considering caterpillars and butterflies. Although both these organisms have identical DNA, the expression of this DNA is determined by the GRN, and it is this difference in gene expression that leads to their obvious physical differences.

The idea that an organism's functional state or physical characteristics are largely determined by gene expressions motivates the modeling of gene expression networks. It is important to understand that an organism's functional state acts recursively upon gene expression through a wide variety of regulatory mechanisms. The link between functional state and gene-expression may be viewed as a complex cellular control system in which information flows from gene activity patterns through a cascade of inter- and intracellular signaling functions back to the regulation of gene expression.

At the molecular level, the dynamic activity of GRNs has been revealed by microarray time series experiments that record gene expression. Since a GRN's structure is directly linked to its dynamic behavior, it has been hypothesized that a GRN's structure may be inferred from microarray data (Akutsu *et al.* 1999). This is a reverse-engineering problem in which causes (the GRNs) are deduced from effects (the expression data).

In this paper we describe the development of *evolutionary algorithm* (EA) methodologies that will help in the exploration of gene-regulatory networks. We have developed a formalism that describes such networks graphically and mathematically (Mandel *et al.* 2004b), and this has been coded using Java. Now, using this formalism, we are working on generating GRN models, and varying them, in order to 'guess' solutions. Here the solution will be the structure of a GRN that can be used, by computer simulation, to accurately reproduce the expression data observed by the microarray time series experiments. To do this we use evolutionary algorithms. However, a problem with using evolutionary algorithms is that they will require large amounts of computation in order to evolve a feasible GRN structure. Here we describe how distributed and grid computing can be used to provide the necessary computation, and we outline various technologies and architectures that can be used to achieve this.

### 1.1 Prior work

Initially we investigated computational formalisms that can be used to describe these bionetworks. Examples of related work include *asynchronous Boolean networks* (Thieffry & Thomas, 1998), *continuous logical networks* (Glass, 1975) and *stochastic networks* (Arkin, Ross and McAdams, 1998). An interesting approach to modeling gene regulatory networks has also been described by Matsuno and co-workers (Matsuno *et al.*, 2000), in which the authors use functional Petri nets to describe the lysis/lysogeny growth pathway in λ-phage.

From these investigations we have developed a novel formalism for modeling the structure and dynamics of concurrent networks. Called *mutuality nets*, this formalism combines features of *Petri nets* (Matsuno, 2003) and stock and *flow diagrams* (Sterman, 2000). Petri nets have problems with modeling functional change but can model non-conserved reaction flows while stock and flow diagrams cannot model non-conserved flows but can model functional change (Mandel *et al.* 2004a). By combining benefits of both formalisms the mutuality net approach can eliminate their problems. Mutuality nets have been discussed elsewhere (Mandel *et al.* 2004b) and will be described in detail in a forthcoming paper. This computational formalism provides the evolutionary algorithm with a symbolic language in which to express the dynamical structure of the reverse-engineered GRNs.

We are primarily interested in the *explanation*, as opposed to the mere *description*, of expression data. This tentative assumption of underlying explanatory dynamics is absolutely crucial to the construction of meaning from data, since only in the context of these dynamics does data attain meaning (Palfreyman, 2004). Several authors (Matsuno *et al.*, 2003, Platzer, 2003, Koza *et al.*, 1999 and D'haeseleer, 2000) have applied dynamical modeling to biological data. What these studies have in common is that they all postulate some framework of causal feedback that is assumed to have generated the data being explained. Matsuno adopts the standpoint that the bionetworks giving rise to biological data are essentially Petri nets; Platzer views these networks as Boolean switching nets; Koza as electronic circuits and D'haeseleer models them as artificial neural networks. None of these standpoints is particularly indicated by the data – and all involve a particular prior prejudice or presupposition on the part of the respective author.

## 2 Evolutionary Algorithms

The field of *evolutionary algorithms* has been growing rapidly over the last few years. Generally, three evolutionary techniques are distinguished: *genetic algorithms* (GA), *genetic programming* (GP) and *evolutionary programming* (EP) (Goldberg, 1989, Michalewicz, 1996).

An evolutionary algorithm (EA) is a search algorithm that is modeled on the mechanics of natural selection and natural genetics. It combines survival of the fittest among individuals with a structured yet randomized information exchange to form a search algorithm. EAs belong to the class of probabilistic algorithms, but they differ from random algorithms in that they combine elements of directed and stochastic (non-deterministic) search. Because of this EAs are more robust than directed search methods. Another advantage of EAs is that they maintain a population of potential solutions while other search methods process a single point of the search space.

EAs maintain a *population* of *individuals* or *chromosomes* for each iteration or *generation*. Each individual represents a potential solution to the problem at hand and is implemented as string structure (GA), tree structure (GP) or some arbitrary other representation (EP). Each solution or individual is evaluated by some *fitness function* to give some measure of its 'goodness' or 'fitness'. A new population or generation of individuals is then generated by selecting individuals from the current generation that are in some sense better (according to the fitness function) then others. Some members of this new population undergo alterations by means of *mutation* and *crossover* operations to form new solutions. Mutation arbitrarily alters one or more *genes* (i.e., string elements, tree-node, etc.) of a selected chromosome by a random change with a probability equal to the *mutation rate*. Crossover combines the features (i.e., substrings, branches of trees, etc.) of two or more parent chromosomes to form two similar offspring chromosomes by swapping corresponding segments of the parents' chromosomes.

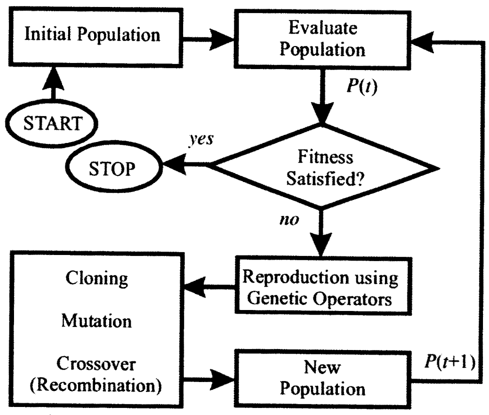

**Figure 1: Basic evolutionary algorithm.**

EAs are based on a set of processes that is repeated depending on the state of the computation. Figure 1 shows the basic process and algorithm for EA-like algorithms.

Recently, there has been increased interest in evolutionary program induction such as EP (Fogel, 1994) and GP (Koza, 1999). Evolving executable structures carries on the basic ideas from GAs with the exception that individuals or chromosomes are no longer represented by simple string structures but by more complex data structures such as trees, graphs, arrays, and so on, or even entire computer programs. With such chromosomes, the decision of what constitutes a gene and how such genes should be manipulated by the genetic mutation and crossover operators is not obvious.

This branch of EAs is ideally suited to address the complex search and optimization problems arising from reverse-engineering of GRNs from microarray data. EAs are often used on hard problems, known as non-deterministic polynomial problems (or NP problems[1] for short). EA approaches are used to solve NP problems, they can 'guess' a solution through mutation and recombination (crossover). In our application, the solution can then be selected for (or against) by means of a quick and simple test procedure; the results of one of our GRN simulations can be compared very quickly to time-series expression data.

Generally, EAs are more effective in finding optimal or near-optimal solutions than brute-force techniques. But they can still require significant computing resources (mainly compute cycles and sometimes also memory). Fortunately, EA techniques are relatively easy to parallelize, which means that multiple computers (worker nodes) can be used simultaneously to share the computational load and thus to increase overall performance. We are particularly interested in using grid technology to achieve this because of its potential to provide a scalable system without having to program many required components from scratch. There are three different approaches to parallelize EAs (Tomassi, 1999); these are briefly described below.

## 2.1 Simple master-worker model

One master process manages the population and hands out individual solutions to be evaluated by a number of worker nodes. Ideally, there is one worker node per individual to be evaluated:

- The master node needs to recognize/identify available nodes in the grid and use them as workers.
- The master node needs to transfer the individuals to the worker nodes, which then evaluate the individual solutions (in this case an executable simulation of a gene-regulatory network).
- After evaluation, each worker node returns the fitness value with the individual solution (or its identifier) to the master node.
- The master node then ranks the evaluated solutions and generates the next generation from it.

Even if there is only one solution to be evaluated per worker node, it is still possible that the time needed to evaluate a particular solution varies significantly. This means that at each generation cycle the master node needs to wait for the worker node that took the longest time before the master node can initiate the next cycle. This synchronization may significantly reduce overall performance. Sometimes reproduction operators themselves are very complex and time-consuming. In such a case, there is significant computation overhead at the master node.

## 2.2 Island-distributed approach

In the *island-distributed* approach (also known as *coarse-grained* approach), each worker node manages semi-independent subpopulations, sometimes called *demes*, of individuals or solutions. There is a loose coupling to other worker nodes. In this setup:

- An 'organizer' node has to recognize available worker nodes (islands) and use them to allocate subpopulations from the original population.
- Individual solutions migrate with between islands either at fixed intervals or in reaction to other criteria. However, the number of individuals migrating is usually kept small.
- At regular intervals the organizer node has to be informed about the development state of each subpopulation (e.g. report best fitness value).

Essentially, this approach allows solution populations to evolve largely independently. This means that the execution of both evaluation and genetic operators is distributed among the worker nodes. While making optimal use of available resources, this approach involves little synchronization overhead at the master node.

[1] Computational complexity refers to computing resources required to solve a given problem. Typical resources are computer cycles (the more cycles need the longer it takes to solve a problem) and memory. Two classes of problem/decision complexity are normally distinguished: (1) Problems that can be *solved* on a deterministic sequential machine in polynomial time (on the size of the problem). This class is known as *P*. (2) Problems that can be *verified* in polynomial time on the same machine are known as NP problems. Equivalently, NP problems are also problems that can be *solved* in polynomial time on a *non-deterministic* (Turing) machine (here: NP stands for *non-deterministic polynomial* time).

### 2.3 Cellular approach

Each worker node processes one individual solution. Each individual solution is mapped to a cell in a virtual grid. The fitness is evaluated simultaneously for all individuals and selection, crossover and mutation take place with neighboring cells on the grid. The configuration of this approach is as follows:

- An organizer node has to recognize available worker nodes and assign one individual to each node.
- Synchronization is needed for the fitness evaluation.
- Individuals migrate locally.

## 3 Determining a good solution

Mutuality nets essentially model gene-regulatory systems as a set of differential equations describing the concentration of different proteins over time. Arkin *et al.* (1998) use a simple pair of differential equations to describe the concentration of two proteins, P1 and P2.

$$\frac{dP_1}{dt} = \max\left[ER_1 - k \cdot P_2(t),\ 0\right] \qquad (1a)$$

$$\frac{dP_2}{dt} = k_2 \cdot P_1(t) - k_3 \cdot P_2(t) \qquad (1b)$$

Varying the unknowns, $k_1$, $k_2$ and $k_3$, results in different models for the behavior of this system over time. Figure 2 shows the way in which three different models predict the concentration of two proteins, $P_1$ and $P_2$.

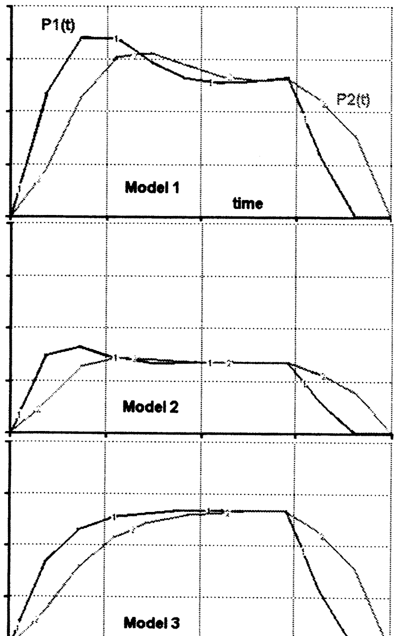

**Figure 2: Changing the unknown values in the differential equations produces different models for protein *concentrations* (y-axis) over *time* (x-axis).**

Comparing the observed protein concentrations with those predicted by a model should tell us how good the model is. There are a variety of statistical techniques that can do this, such as *sum of the squares* of the differences. Minimizing a metric such as this using EAs should tell us what values for $k_1$, $k_2$ and $k_3$ are optimal for modeling the gene-regulatory network. This assumes, of course, that the behavior of the gene-regulatory network is not chaotic. In this case even small variations of the unknown values would produce radically different behavior plots.

However robustness is an essential feature of biological systems (Kitano, 2002) which makes them relatively insensitive to minor alterations of their internal parameters. This significant property is achieved by regulatory feedback loops, modularity and redundancy of functional units and structurally stable network configurations.

Consequently, the topology of gene-regulatory networks which defines the interactions between the genes and their products has a strong impact on the robustness. In our EA approach, the topologies of GRN individuals are also directly evolved in mutuality net model representations.

## 4 Distributed implementations

In this study, the fitness of a particular GRN (GRN topology plus definition of expression level dynamics) is determined by executing the GRN (using given initial expression levels) and then comparing its output with the concentrations (RNA abundances) measured on real biological entities. The fitness calculation is the algorithm's most computationally intensive element. If these computations can be executed in parallel for all individuals or for subsets of individuals then the applications could be run on batch process and cycle-stealing workload management systems such as Condor (Thain *et al.* 2003) or Sun's Grid Engine (SGE). These systems can be used to construct pools of machines from either idle desktop computers or dedicated computing resources, and are currently available as open source software distributions. They allow machine pools to be used for executing computationally intensive jobs, and they offer relatively sophisticated facilities to queue, manage, checkpoint, and restart submitted jobs.

### 4.1 Condor

For the application discussed in this paper, we are interested in using Condor to construct and use a computer cluster from desktop machines, and we aim to implement the master-woker model described above using this infrastructure. The machines used in the cluster are typically in use during office hours but otherwise sit idle. In this approach, the master node will submit individual GRN simulations to the Condor pool; it will evaluate the individuals' fitness and execute the operations necessary to generate a new population of individual simulations. These simulations will again be submitted to the Condor pool, and so on, until an optimal solution to the GRN reengineering problem is found.

Java has been used before to develop genetic algorithms that may be run on a Condor pool. JavaGenes (Globus *et al.*, 2000) is an application that was developed to evolve graphs. These graphs represent molecules, composed of atoms and bonds, which often contain cyclical or ring structures. Each graph's fitness could be evaluated independently, which made the algorithm a natural candidate for Condor, and, in order to evaluate the statistics supporting the optimal graphs, 31 duplicate jobs were run differing only in the initial random seed.

JavaGenes was one of the first applications to use Java with Condor, andthe authors reported some serious technical difficulties. Expanding Condor to include a so-called Java Universe was not trivial, and involved significant developmental work. Since then the Condor Project has released a Java Universe that allows the JVM to be fully integrated within Condor's distributed, homogeneous environment (Thain and Livny, 2002).

While Condor is a good choice for implementing the master-worker model, in both the island-distributed and the cellular approach it is important to enable communication between individuals and subpopulations. Such scenarios are not so easy to parallelize as the master-worker model. Therefore we are investigating using JavaSpaces as an alternative to Condor.

## 4.2 JavaSpaces

JavaSpaces (Freeman *et al.*, 1999) is a Java-based technology that offers a simple approach to implementing parallel and distributed applications and, like Java, it is independent of hardware and software environments. It is currently distributed free of charge, and is an integral part of Jini technology (Edwards and Edwards, 2000). JavaSpaces is based on a tool called Linda (Gelemter, 1985) that was used to create distributed applications. Linda used a storage space for objects in order to simplify the implementation of parallel and distributed applications. The storage space was called a 'tuple space' and this inspired the name JavaSpaces.

JavaSpaces is designed to provide a simple but unified set of components for dynamic communication, coordination, and sharing of data and objects. In a distributed application, JavaSpaces technology acts as a virtual space between providers and requesters of network resources or objects. This allows participants in a distributed solution to easily exchange tasks, requests and information, and gives developers the ability to create and store objects with persistence. Specific features of JavaSpaces include:

- *Space sharing*. Multiple processes can concurrently interact with a space -- the JavaSpace handles the details of concurrent access.
- *Persistent Spaces*. JavaSpaces are persistent and facilitate the reliable storage of data and objects in a JavaSpace. It is also possible to request a *lease*, i.e., a duration of storage for an object. When the lease expires, the object will be removed by the system.
- *Associative Spaces*. Since JavaSpaces are associative, one does not need to know the name or location of an object to find it in a JavaSpace. The object can be discovered by associative content search based on a template. Associative lookup provides a 'loose' coupling allowing programmers to develop applications without knowing locations of other application components.
- *Transactions*. Spaces allow transactions using Jini's transaction service. Transactions are a crucial mechanism to cope with partial system failures.
- *Exchange of executable content*. Spaces allow you to exchange executable content. Objects residing in a JavaSpace cannot be invoked inside a JavaSpace. However, to execute an object, it is possible to retrieve an object from a JavaSpace, create a local copy, and invoke its methods.

In JavaSpaces Java objects can be used to represent GRN simulations, and can be written to a shared JavaSpace. Idle worker nodes can subsequently search the JavaSpace for objects waiting to be processed and read them, process them, and return the results in the form of result objects. It enables objects to be exchanged for processes so that parallel and distributed applications may be implemented in a relatively simple manner. Furthermore, with this system queuing and load balancing occurs naturally.

Setzkorn and Paton (2004) have implemented three frameworks for different parallel evolutionary algorithms: a synchronous master-worker PEA, an asynchronous master-worker PEA, and a coarse grained PEA (Setzkorn and Paton, 2004). The objective of these algorithms was to induce fuzzy classification rules. In the synchronous master-worker system, the master executes the evolutionary algorithm, while individuals are written into the JavaSpace where they are read and evaluated by the workers. In this case, either the master is idle while the workers evaluate the individuals, or the workers are idle while the master generates and initiates the next population. These problems were overcome by using the asynchronous and coarse-grained PEAs.

The coarse grained PEA is of particular relevant to modeling and computation task. This framework uses a similar structure to the synchronous system, but each worker runs an evolutionary algorithm in order to evolve a subpopulation of individuals, while the master node is able to migrate individuals from one subpopulation to another. As all subpopulations are virtually connected within the JavaSpace, individual migrations can be implemented easily, and if relatively few migrations are used then this approach can be faster than the simple synchronous master-worker system. Overall, Setzkorn and Paton concluded that JavaSpaces could be used to improve the performance of evolutionary algorithms in a simple and affordable manner.

We are currently exploring both Condor and JavaSpaces to implement two different solutions, depending on the level of communication required between individuals. In addition the GRN scenario discussed in this paper, our research in the DataMiningGrid Project (DataMiningGrid) is concerned with extending these technologies to perform data mining operations on the grid. Running our Java code under Condor will be comparatively simple and quick to implement so long as there is no communication between individual simulations. If the communication between simulations is essential, as it is for the island-distributed and cellular approaches, then there will be many advantages to using the shared object space available in JavaSpaces.

The DataMiningGrid Project addresses the current lack of a coherent framework for developing and deploying data mining applications on the grid. It will address this gap by developing generic and sector-independent data mining tools and services for the grid. A test bed consisting of several applications from a diverse set of sectors will serve as platform for demonstrating and promoting the technology developed by the DataMiningGrid. Because of its relevance across many sectors, the DataMiningGrid project has the potential to improve the sharing and exploitation of information across many R&D and business domains.

## 5 Approach

Since we wish to explore an EA approach in which there is significant communication between subpopulations of individuals, the remainder of this discussion focuses on the JavaSpaces approach to implementing the island-distributed model.

Initially, we will develop a generic EA system to create a scaleable system. Rather than using a single JavaSpace and asking each worker node to maintain a subpopulation of individuals, we will employ multiple JavaSpaces, each of which will contain a subpopulation of individuals.

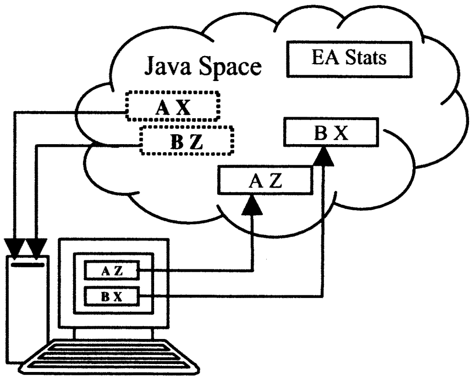

**Figure 3: A worker node (computer symbol) selects two individuals (boxes with upper case letters indicating chromosomes or individuals), performs crossover, mutation and evaluation and places the new individuals back in the JavaSpace.**

The structure is relatively simple – each JavaSpace contains up to 100 individuals as objects. An additional object is used to maintain sub-population statistics, such as the best individual found so far, the total fitness of the subpopulation, status flags and mechanisms for propagating information back to the user. Each worker node will search the JavaSpace for two individuals, download them, and then perform crossover, mutation and evaluation operations, before returning the results to the JavaSpace. Those individuals with a higher evaluation score will have a higher probability of being selected. Figure 3 above illustrates this technique.

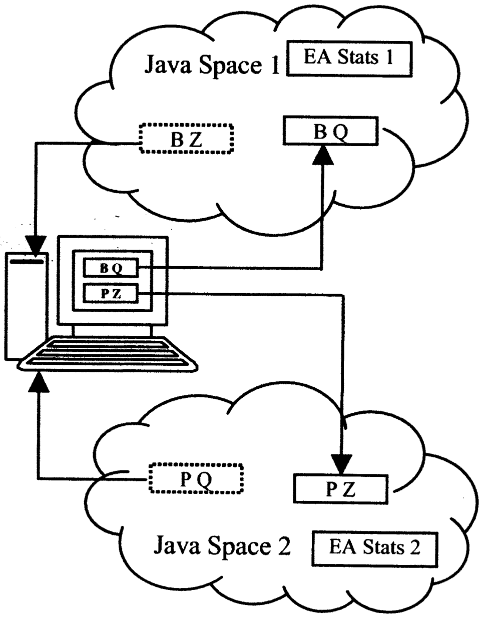

**Figure 4: A worker node connected to two or more JavaSpaces can select individuals from either, thus providing a bridge for migration.**

This in itself would be a sufficient system to find satisfactory solutions, but the real advantage in using JavaSpaces is when implementing the island-distributed approach. Worker nodes connected to more than one JavaSpace can choose to take individuals from any of them, as illustrated in Figure 4. Crossover, mutation and evaluation can take place as before and the evaluated individuals are placed back into these JavaSpaces to replace the individuals removed. This allows migration to take place between JavaSpaces.

To ensure that broken links will not result in objects taking up permanent residency in remote JavaSpaces, the lease feature of JavaSpaces will be used to remove unwanted objects if no renewal signal is received periodically from the originating node of the system (the node to which the results will ultimately be returned).

## 6 Conclusions and future work

Gene-regulatory networks can be modeled using mutuality nets that combine Petri nets and stock and flow diagrams (Mandel *et al.*, 2004a). However, these models require parameters representing unknown interactions between gene expressions and protein concentrations. Finding these unknown values is a difficult problem for which we propose to use evolutionary algorithms.

Determining mutuality network models of gene-regulatory networks using an EA approach requires considerable computational power. The search space is vast, and cannot be thoroughly searched in a deterministic manner. The more computational power that is available the better the solutions that we will be able evolve. Both Condor and JavaSpaces are technologies that facilitate the relatively easy implementation of applications requiring the sharing of computational load over multiple processors. In this paper we have explored the relative pros and cons of the two technologies for the study of gene-regulatory networks using mutuality nets. In particular, we have discussed using JavaSpaces to create a scalable system that will allow communication between individual subpopulations. We are currently preparing a comparative study and implementation of EA-based GRN topology discovery using mutuality nets (Mandel *et al.*, 2004b) based on both JavaSpaces and Condor.

## Acknowledgements

This work has been supported by the European Commission's FP6 IST DataMiningGrid Project (www.DataMiningGrid.org), contract number 004475.

## References

Akutsu, T., Miyano, S., Kuhara, S. (1999), "Identification of genetic networks from a small number of gene expression patterns under the Boolean network model", *Pacific Symp. Biocomp. 99*, 4: 17-28.

Arkin, A., Ross, J. and McAdams, H. (1998), "Stochastic kinetic analysis of developmental pathway bifurcation in phage lambda-infected Escherichia coli cells", *Genetics*, 149/4, 1633-48.

D'Haeseleer, P. (2000), *Reconstructing Gene Networks from Large Scale Gene Expression Data*, Ph.D. thesis, (University of New Mexico).

DataMiningGrid, DataMiningGrid – Data Mining Tools and Services for Grid Computing Environments at www.DataMiningGrid.org.

Edwards, W.K., Edwards, W. (2000), *Core Jini (2nd Edition)*, Prentice Hall PTR.

Fogel, D. (1994), "Applying Evolutionary Programming to Selected Control Problems" Comp. Math. App., 11(27), pp.89-104.

Freeman, E., Hupfer, S. and Arnold, K. (1999), "JavaSpaces™ Principles, Patterns and Practice", Addison-Wesley Longman Ltd.

Gelemter, D. (1985), "Generative communication" in Linda (ed.) *ACM Transactions on Programming Languages and Systems*, 1,80-112.

Glass, L. (1975), "Classification of biological networks by their qualitative dynamics", *J. Theor. Biol.*, **54**, 85-107.

Globus, A. et al. (2000), "JavaGenes and Condor: Cycle-scavenging genetic algorithms" in *Proceedings of the ACM Conference on Java Grande*, San Francisco, California, 134--139.

Goldberg, D.E. (1989), *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley.

Kitano, H. (2002), "Computational systems biology", Nature, 420:206-210

Koza, J., Bennett, F., Andre, D. and Keane M. (1999), *Genetic Programming III: Automatic Programming and Automatic Circuit Synthesis*, Morgan Kaufmann.

Mandel, J., Palfreyman, N., Lopez J., Dubitzky W. (2004a) "Representing Bioinformatic Causality", in *Briefings in Bioinformatics* (Henry Stewart Publications), 5/3, 270-283.

Mandel, J., Palfreyman, N. (2004b) "A Pen-and-Paper Notation for Teaching in the Biosciences", in J.A. López, E. Benfenati, W. Dubitzky (Editors), *Proc of Int'l Symposium Knowledge Exploration in Life Science Informatics 2004 (KELSI 2004)*, pp84-95, Milan, Italy,

Lecture Notes in Computer Science 3303, ISBN 3-540-23927-8 Springer Berlin Heidelberg New York.
Matsuno, H., Doi, A., Nagasaki, M., and Miyano, S. (2000), "Hybrid Petri Net representation of gene regulatory network", *Pacific Symp. Biocomp.*, 5, 338-349.

Matsuno, H. *et al.* (2003), "Biopathways representation and simulation on hybrid functional Petri net" in *Silico Biol.*, 3, 32.

Michalewicz, Z. (1996), *Genetic Algorithms + Data Structures = Evolution Programs*, 3rd edition, Springer-Verlag, Berlin.

Palfreyman, N. (2004), "The Construction of Meaning in Computational Integrative Biology", in W. Dubitzky (guest editor), *OMICS: A Journal of Integrative Biology, special issue: Data Mining meets Integrative Biology – a Symbiosis in the Making*, Vol. 8. No 2. 95-105, 2004.

Platzer, U. (2003), "Simulation of Genetic Networks in Multicellular Organisms", Ph.D. thesis, (Heidelberg University).

Setzkorn, C, Paton, R.C. (2004), "JavaSpaces – An Affordable Technology for the Simple Implementation of Reusable Parallel Evolutionary Algorithms", in J.A. López, E. Benfenati, W. Dubitzky (Editors), *Proc of Int'l Symposium Knowledge Exploration in Life Science Informatics 2004 (KELSI 2004)*, pp84-95, Milan, Italy, Lecture Notes in Computer Science 3303, ISBN 3-540-23927-8 Springer Berlin Heidelberg New York.

Sterman, J. (2000), *Business Dynamics: Systems Thinking and Modelling for a Complex World,* (McGraw-Hill/Irwin).

SGE, Sun Grid Engine at http://gridengine.sunsource.net/

Thain, D. and Livny, M. (2002), "Error Scope on a Computational Grid: Theory and Practice" in *Proceedings of the Eleventh IEEE Symposium on High Performance Distributed Computing (HPDC11)*, Edinburgh, Scotland.

Thain, D., Tannenbaum, T., and Livny, M. (2003), "Condor and the Grid" in Fran Berman, Anthony J.G. Hey, Geoffrey Fox (ed.), *Grid Computing: Making The Global Infrastructure a Reality* (John Wiley).

Thieffry, D. and Thomas, R. (1998), "Qualitative Analysis of Gene Networks", *Pacific Symp. Biocomp.*, 3, 66-76.

Tomassi, M. (1999), "Parallel and distributed evolutionary algorithms: A review" in K. Miettinen, M. Makela, P. Neittaanmaki, and J. Periaux (ed.), *Evolutionary Algorithms in Engineering and Computer Science* (Chichester: J. Wiley and Sons), 113-133.