

PLANNING OF COMPLEX INDUSTRIAL SYSTEMS USING A NOVEL PARALLEL GENETIC ALGORITHM

Turner S⁺, Tindle J⁺, Fletcher I⁺, Mellis J*, Mortimore D*, Tindle S⁺

⁺ *School of Computing and Technology, University of Sunderland, Sunderland, SR6 0DD, UK*
Tel: +44 (0) 191 515 2000 Email: *steve.turner, john.tindle, ian.fletcher, sonia.tindle@sunderland.ac.uk*

* *Evolved Networks Limited, B55 Phoenix Place, Adastral Park, Martlesham Heath, Suffolk, IP5 3RE, UK*
Tel: +44 (0) 1473 663 060 Email: *john.mellis, david.mortimore@evolvednetworks.com*

Keywords: Parallel Genetic Algorithm Grid Cluster

Abstract

Today's network planning problems are made up of many thousands of parameter variables all requiring optimisation to produce a good solution. At this level of complexity single processor Genetic Algorithm (GA) solutions begin to struggle due to the enormous search spaces associated with the problems. Parallel GAs can overcome this issue by distributing the problem over a number of processors. However, they can be hampered by a communications bottleneck between processors or inadequate sharing of information. This paper presents a new type of parallel GA which has been developed to create a distributed parallel processing environment in which near-optimal solutions can be generated for complex large parameter set problems in an acceptable length of time. The processing environment comprises a set of servers or nodes interconnected via a Local Area Network (LAN) with the algorithm sharing data amongst the nodes.

1 Introduction

There are numerous planning problems prevalent in society particularly those facing the utility companies in providing a reliable supply to customers. Following discussions with a number of industry experts it has been determined that within the next few decades there will be a severe skills shortage as many experienced planners near retirement, leaving their younger counterparts grappling with planning problems without the necessary skill set or experience.

To counter this problem and ease the burden on the planner, there are various commercial planning tools available such as *accessfirst* for telecoms [1], *CYMDIST* for electricity [2] and *InfoWorks WS* for water [3]. Some of these tools such as *accessfirst* employ evolutionary computing (EC) methods to produce near optimal solutions within a fraction of the time that would take even an experienced network planner days to accomplish [4]. The research underpinning *accessfirst* was undertaken in conjunction with the University of Sunderland [5]. Two further EPSRC funded research projects have been completed at Sunderland which also employed EC

techniques to address network planning problems [6],[7]. In all cases, the work was undertaken on single processors. Results from [7] in particular, clearly indicated that for some network planning problems additional computing capacity is required to scale up to industry-sized problems.

Recent research undertaken at the University of Sunderland has focussed on the feasibility of producing a multi-utility optimiser which is capable of solving network planning problems for the telecoms, gas, water and electrical distribution networks. As a consequence of the earlier research at Sunderland there has been an emphasis on using parallel computing techniques in conjunction with aspects of EC to solve large complex problems. Indeed it has been noted that the nature of evolutionary algorithms such as Genetic Algorithms (GA) and Particle Swarm Optimisation (PSO) is 'intrinsically parallel and distributed' [8].

While parallel processing has been a topic for research for more than two decades, recent advances in technology have seen a resurgence of interest in cluster computing and grid computing. Cluster computing refers to a single set of nodes residing in one location, whereas a grid computer may be composed of many clusters together with other kinds of resources such as networks and storage facilities. In the past, grid computing has normally been associated with large-scale scientific projects such as the Large Hadron Collider (LHC) at CERN [9] and the search for extra-terrestrial life (SETI@home) [10]. More recently however, it has been recognised that there is huge potential for the application of cluster and grid computing to everyday problems within business and industry.

In order to accelerate the application of grid technology within the commercial world, the UK Department of Trade and Industry (DTI) are funding *Grid Computing Now (GCN)* [11]. GCN operates two Knowledge Transfer Networks (KTNs) with a view to bridging the gap between research and industry: the *IECnet* (Inter-Enterprise Computing network) for large enterprises and *Grid UK* for small and medium-sized businesses. Further discussion of the potential impact of grid computing on the business world may be found in [12].

Current research at the University of Sunderland employs a cluster computing facility that was funded by the Science Research Investment Fund (SRIF1).

This paper examines what the authors believe to be a new type of parallel GA which incorporates elements of

PSO to solve large scale problems that would otherwise be insolvable on single processor systems.

2 Parallel Genetic Algorithms

Genetic Algorithms are arguably the best known type of Evolutionary Computing [13]. GAs, developed by John Holland in 1975 [14], model nature's evolutionary characteristics by 'breeding' new solutions over a number of generations. Each solution consists of a set of *chromosomes* which contain *genes*, each gene represents a variable in a solution (the chromosome). These solutions are subject to *crossover*, *mutation*, and *selection* to form new solutions as in nature where these operations produce new, slightly different offspring. Over time the *best* solutions are more likely to be used in the production of the new generation through survival of the fittest. To determine which solutions are better than others a *Fitness Function* is used in conjunction with a *Penalty Function* to rank solutions in terms of suitability. This ensures that the more favourable solutions are used in the production of the next set of solutions.

GAs by their very nature can be easily adapted for use in a parallel environment. Parallel Genetic Algorithms have been around for at least three decades [15] and consequently there are several major models [16], [17]:

- Master-Slave
- Multiple-deme aka coarse-grained or distributed
- Hierarchical parallel GA
- Fine grain

They offer different methods for distributing the workload of the GA over a parallel computing architecture such as a grid computer cluster. There are many factors in deciding the structure of a parallel GA.

- Are single or multiple populations used?
- Are the GA operations or fitness calculations distributed to other machines?
- How is population data shared amongst processors?

2.1 Master-Slave

The master-slave configuration uses a central processor which carries out the central GA operations (selection, crossover, mutation) and distributes the fitness evaluation to other processors.

This is regarded as one of the easiest ways to implement a parallel GA [16]. The overall operation of the algorithm is the same with the performance improvement being in the calculation of fitness values. The population is split between the slave processors and a portion is sent to each one for evaluation. Studies have indicated that this does indeed improve the performance of the algorithm over a single processor implementation.

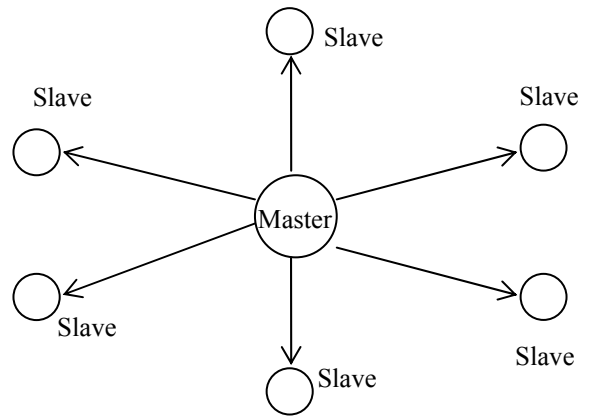


Figure 1. Master-Slave GA

However, research has shown that as the number of processors is increased the efficiency of the parallel algorithm is diminished [16]. This is mainly due to the increase in communication overheads.

2.2 Multiple-Deme

These are also known as *coarse-grained* or *distributed* GAs. There is less communication involved as the populations are much more isolated from each other than in the Master-Slave approach; this has resulted in multiple-demes being referred to as the *Island model*. Yet this approach is much more complicated because there are many variables to consider, the main three being:

- The size and number of demes (sub-populations)
- Topology of connections between demes
- How many individuals migrate each generation

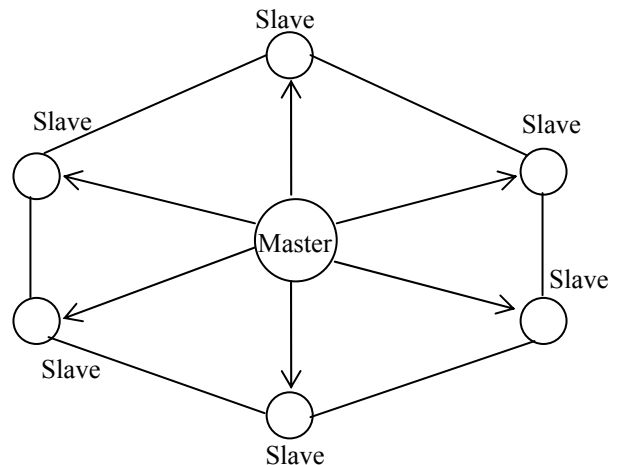


Figure 2. Multiple Deme (coarse grained) GA

2.3 Hierarchical parallel GA

A Hierarchical Parallel GA is a combination of the aforementioned Multiple-Deme (MD) GA and the Master-Slave approach. The idea behind it is to take advantage of the lower communication overheads of the MD whilst

using the master-slave technique to reduce the isolation between populations that the MD model can suffer.

This can be implemented by having a set of linked Master-Slave GAs, thus the multiple Master-Slave GAs can communicate their population amongst one another which effectively parallelises the Master-Slave cluster.

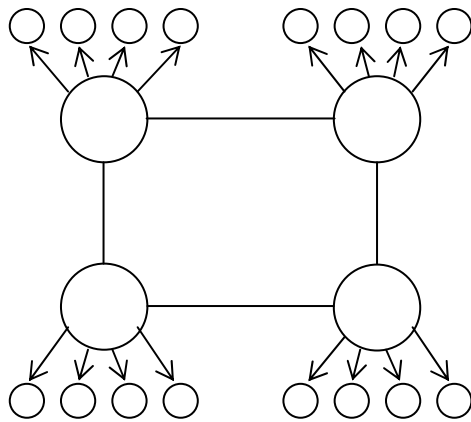


Figure 3. Hierarchical Parallel GA

2.4 Fine Grain

Fine Grain parallel GAs consist of a large set of fully connected “workers” each of which shares its population with all the other members. Migration of the population of each worker occurs after it has converged, at which point the worker shares its solution with its peers. After migration has occurred, the workers restart and run until convergence.

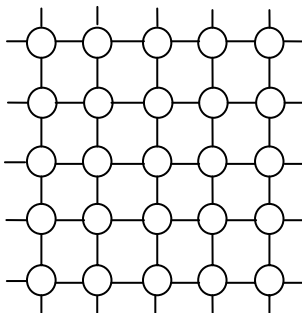


Figure 4. Fine Grain GA

This type of configuration is suited to large clusters of processors. Communications between the workers is very low since it only occurs during migration hence there is little in the way of a communications overhead.

An advantage of using this technique is that the expected quality of solutions in each worker is vastly improved compared to the simple GA or the Island model. This is due to the sharing of solutions between all workers in the cluster so that all will eventually have a copy of the best solution.

2.5 Speedup using parallel algorithms

In 1967, Amdahl theorised that the maximum speed up obtained from adding an infinite number of processors to a

problem was limited by the amount of serial processing required in the algorithm [18]. This gave way to Amdahl’s law which states:

$$\text{Speedup} = 1 / (s + p/N) \quad (1)$$

Where

s = serial processing time

p = parallel speedup

N = number of processors.

Therefore it is imperative to minimise any serial processing within the parallel algorithm. However, there are some flaws with the above formula when applied to large scale parallel problems. Gustafson argues that the serial portion of the program such as start up time, loading and I/O operations does not grow with problem size whilst the amount of work that can be done in parallel ‘varies linearly with the number of processors’ [19].

He notes that E. Barsis gives an alternate law:

$$\text{Scaled speedup} = N + (1-N) * s' \quad (2)$$

Where

N = number of processors

s' = serial time spent on parallel system.

3 A new type of parallel GA

The main focus of this paper is on a new type of GA which the authors have termed ‘Daisy Chain Parallel GA’. It has a similar structure to the Island model but with unique communication and control mechanisms which enable it to be highly scalable and easily implemented over a distributed network or cluster of processing nodes.

3.1 Daisy Chain algorithm

The daisy chain GA was constructed by decentralising the processing of the algorithm so that all nodes (servers) in the parallel algorithm equally contribute to the solving of the problem. This was accomplished by connecting the nodes up in a loop so that each node regularly passed its best solution to another in the chain via File Transfer Protocol (FTP). Separate Java threads for communication were used to ensure that the sending of solutions did not create a bottleneck in the algorithm.

3.2 Structure of Daisy Chain GA

The algorithm is initiated by an HTML form residing on the trigger server (node) which allows the user to input the variables used to configure the algorithm and the server IP addresses of the nodes used in the calculation. These variables are passed using a JavaServer Page (JSP) on the web server to the Java application via a JavaBean which then starts the first node in the chain of nodes. The first node then contacts the second node via a URL object call to a copy of the JSP residing on the neighbouring node, starting up the second node that in turn activates the third node and so on until the last node in the chain is activated.

Each node runs a GA on a population randomly generated on the local machine which over time sends and receives 'elite' solutions between its neighbouring nodes.

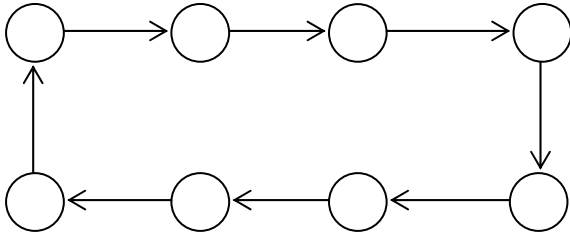


Figure 5. 8 node daisy chain set up

Each elite solution passed is a simple data file containing the best solution of the local population array; data is separated by white space and the solution also contains the calculated error and fitness values as seen in figure 6.

var1	var2	var3	varN	error	fitness
------	------	------	-------	------	-------	---------

Figure 6. Contents of an elite solution data file

The GA itself employs standard tournament selection, single point crossover and a creep factor instead of mutation so that as the algorithm approaches a near optimum solution the level of creep applied to individual variables of the solution is reduced.

4 Experiments

To benchmark the algorithm, test functions 1, 3 and 5 from De Jong's PhD Thesis [20] were used which provide a good set of problems covering a broad range of environments from continuous to discrete models all with the common goal of minimising a function. To test the optimiser to its limits, multiple instances or sets of the functions were solved simultaneously. This was because the GA converged on a single function too rapidly to show any improvement through parallelisation due to communication overheads. Each test was given an upper limit of 5 minutes to complete to meet the industrial requirements of the research, after which if it had not converged it was marked as Did Not Finish or DNF.

The test environment was a suite of 8 networked PCs running Windows XP Pro, Apache Tomcat 5.5.17, JRE 1.5 and WS_FTP Server 3.14 over a 100Mb LAN.

4.1 De Jong function 1

This is a simple 3 dimensional problem within a range of +/-5.12 with a minimum value of 0

$$F1(X) = \sum_{i=1}^3 x_i^2 \quad (3)$$

4.2 De Jong function 3

Function 3 is a 5 dimensional step function where each possible solution is a flat surface arranged in a series of steps increasing in value. This function was restricted to the range +/- 5.12 with a minimum value of -30

$$F3(X) = \sum_{i=1}^5 [x_i] \quad (4)$$

where $[x_i]$ is the greatest integer less than or equal to x_i

4.3 De Jong function 5

This is a problem with 25 local minima lying on an otherwise flat plane, also known as the 'foxholes' problem on account of the series of narrow holes which can easily trap algorithms within a local minima.

$$\frac{1}{F5(X)} = \frac{1}{K} + \sum_{j=1}^{25} \frac{1}{f_j(X)} \quad (5)$$

Where

$$f_j(X) = c_j + \sum_{i=1}^2 (x_i - a_{ij})^6 \quad (6)$$

$$K = 500$$

$$c_j = j$$

The function was restricted to +/- 65.536 with the local minima set at the vectors

$$\{a_{ij}\} = \begin{cases} -32, -16, 0, 16, 32, -32, -16, \dots, 0, 16, 32 \\ -32, -32, -32, -32, -32, -16, -16, \dots, 32, 32, 32 \end{cases}$$

with the global minimum lying at (a_1, a_2) where the function minimises to 1. The minima lie at 1 - 25 since when x_i is very close to a_{ij} , $f_j(X)$ tends towards c_j therefore $F5(X)$ tends towards j ($1/K \ll 1/j$).

5 Results

5.1 De Jong function 1

This function was tested by solving 1, 100 and 500 sets of the problem concurrently. When solving only 1 problem set it was shown that a single machine was better than multiple due to the simplicity of the test function and lack of communication.

However, when the number of sets was increased to 100 the parallel architecture began to show an improvement over the single processor with the single processor taking on average 3.5 seconds compared to 0.96 when using 8 machines. The improvement is more marked when the number of sets reach 500, the single processor machine can no longer cope and fails to produce any results within the 5 minute upper limit. Using 2 machines produces results in an average of 2mins 41 seconds whilst 4 machines produces a time of 2m 34s and 8 machines with a time of 2m 21s.

5.2 De Jong function 3

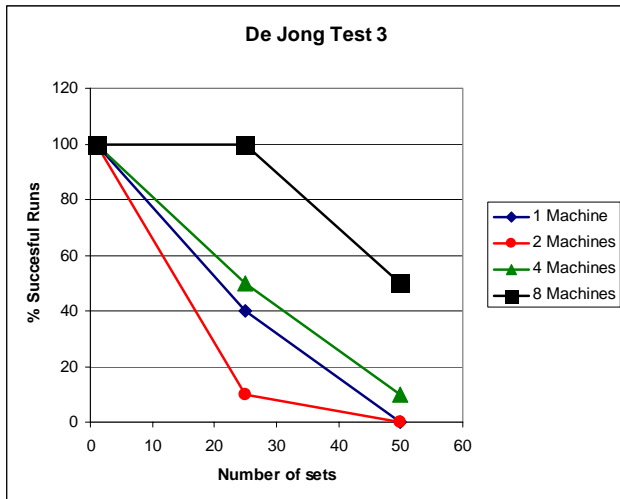


Figure 7. De Jong test 3 results

The single set of problems showed no improvement with more machines, but by placing a greater strain on the algorithm with more problem sets to solve, the parallel improvements became apparent. When evaluating 25 sets at once only the 8 machine implementation successfully completed all 10 runs with 4 machines completing 50% of the runs, 2 machines completing 10% of the runs and the single machine failing to complete any. The improvement through parallelism was also evidenced in evaluating 50 sets though the upper limits for evaluation were being reached on the 8 machine set up; 50% successful runs vs. 10% for 4 and 0% for 1 or 2 machines.

5.3 De Jong function 5

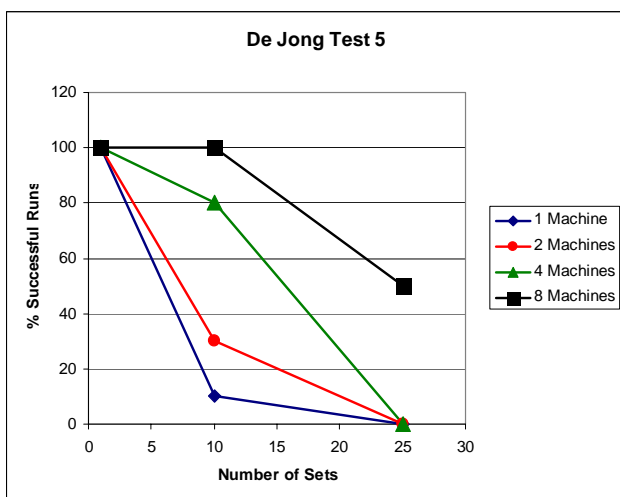


Figure 8. De Jong test 5 results

The final test again showed very little improvement when adding more machines for the single set of problems but when this was increased to 10 sets the improvement became apparent. When using a lower number of machines the number of DNF occurrences rose significantly, indeed only the 8 machine architecture successfully completed all 10 runs rapidly within an average time of 15 seconds. By further increasing the number of problem sets to 25 even

the 8 machine set up struggled and only returned 5 successful runs out of 10. All lower set ups failed to complete their runs.

6 Conclusions

In our experiments the authors aimed to obtain satisfactory solutions within five minutes. To prove the robustness and precision of the optimiser, ten runs for each function set were carried out to determine the average solution time and standard deviation. Experiments have clearly shown that if a single processor system is able to solve a simple problem there is no benefit to be gained by using a cluster computer. When a complex problem is considered, for example the fifth De Jong function (5) with a large number of variables, the problem can only be solved by using an eight processor cluster computer. In most cases, the more complex problems cannot be solved in a reasonable period of time by less than an eight processor cluster.

The optimisation system may be classed as a loosely coupled system where a small proportion of the processing time is devoted to data communication. Experiments have shown that the elite data only needs to be transmitted once in every five hundred generations. In this way the serial component(s) in equation (1) has been reduced to maximise the speedup factor. It has not been possible to calculate an average speedup factor with the particular cluster computer currently available. This is because in many cases it was not possible to solve the very complex problems within the five minute time limit.

The experiments that have been carried out demonstrate that there is a need for a more powerful computer to solve more complex problems. In the near future a large cluster computer funded by the SRIF 3 initiative will be installed at the University of Sunderland with two hundred processors. Further experiments will be undertaken to validate the performance of the new cluster computer by using the daisy chained parallel GA solver/optimiser and the benchmark functions described in this paper.

Acknowledgements

This paper has been supported by an EPSRC sponsored industrial grant, number 430247X in conjunction with Evolved Networks Limited and the University of Sunderland. The authors would also like to acknowledge the Science Research Investment Fund (SRIF1) for providing the computing facilities used in the development and testing of the algorithm.

References

- [1] Evolved Networks: [accessfirst, http://www.evolvednetworks.com/product/afds.html](http://www.evolvednetworks.com/product/afds.html), Last accessed 29th June, 2006.
- [2] CYME International: CYMDIST, Distribution System Analysis, Available from

- <http://www.cyme.com/software/cymdist/> Last accessed 29 June 2006.
- [3] Wallingford Software: InfoWorks WS, Wallingford Software Ltd, Available from http://www.wallingfordsoftware.com/products/infoworks_ws/ Last accessed 29 June 2006
- [4] Asumu, D.E., Mellis, J., Performance in planning – smart systems for the access network, *BT Technology Journal* Vol 16 No 4 (1998).
- [5] Tindle, J. and Poon, K.F. “Addressing Optimisation Issues in Network Planning with Evolutionary Computing”, *Telecommunications Optimisations: Heuristics and Adaptive Techniques*, (eds.) Corne, D.W., Oates, M.J., Smith, G.D., John Wiley and Sons Ltd., England, pp. 79-97, (2000) ISBN 0-471-988553.
- [6] Tann, P.L. “Development of a Symbol Recognition System Using Evolutionary Computing Methods” *PhD Thesis*: University of Sunderland, EPSRC Industrial CASE Award, grant reference 01305600. (2005).
- [7] Tindle, S.J. “Intelligent Methods for Optimising Network Solution Design” *PhD Thesis*: University of Sunderland, EPSRC Industrial CASE Award, grant reference 00313350. (2005).
- [8] Alba, E., Tomassini, M., Parallelism and Evolutionary Algorithms, *IEEE Transactions on Evolutionary Computing*, Vol 6 No 5 pp 443-462 (2002).
- [9] Large Hadron Collider (LHC) Computing Grid: <http://lcg.web.cern.ch/LCG/>, last accessed 28th June, 2006.
- [10] Search for Extraterrestrial Intelligence (SETI): <http://setiathome.berkeley.edu/>, Last accessed 28th June, 2006.
- [11] Grid Computing Now (GCN): <http://www.gridcomputingnow.org>, Last accessed 28th June, 2006.
- [12] Knight, W., Unlocking the Grid, *IET Engineering and Technology*, 1, No.3, 43-45, (2006), ISSN 1750-9637.
- [13] Morley, M.S., Atkinson, R.M., Savic, D.A., Walters, G.A., GAnet: genetic algorithm platform for pipe network optimisation. *Advances in Engineering Software* Vol 32, pp467-475 (2001).
- [14] Holland, J.H., *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*, University of Michigan Press, Ann Arbor, MI (1975).
- [15] Bethke, A.D., Comparison of Genetic Algorithms and Gradient-Based Optimizers on Parallel Processors: Efficiency of Use of Processing Capacity, *Tech Rep* No 197, University of Michigan, Logic of Computers Group, Ann Arbor, MI, (1976).
- [16] Cantú-Paz, E., Goldberg, D.E., Efficient parallel genetic algorithms: theory and practice, *Computer Methods in Applied Mechanics and Engineering*, No. 186 pp221-238 (2000).
- [17] Eklund, S.E., A massively parallel architecture for distributed genetic algorithms, *Parallel Computing*, Vol 30 pp 647-676 (2004).
- [18] Amdahl, G.M., Validity of the single-processor approach to achieving large scale computing capabilities. *AFIPS Conference Proceedings* Vol 30 pp 483-485 (1967).
- [19] Gustafson, J.L., (1988) Reevaluating Amdahl’s Law, *Communications of the ACM* May 1988 Vol 31 N. 5.
- [20] De Jong, K.A., An analysis of the behaviour of a class of genetic adaptive systems. *PhD thesis*, University of Michigan, U.S.A. (1975).