

A Comparison of Parallel Global Optimisation Algorithms for Reverse Engineering Gene Networks

Luke Jostins

August 21, 2008

Abstract

The approach of reverse engineering gene networks by fitting models has been highly successful, but increasing model complexity means that more powerful global optimisation techniques are required for model fitting; for this, we require faster parallel algorithms. I examine the modeling of the gap gene network in *Drosophila*, for which a gene network model, in the form of a set of differential equations, is fitted to high-resolution spatio-temporal expression data. Previously model fitting has been performed with Parallel Lam Simulated Annealing, but it has been shown that an island Evolutionary Strategy would be more efficient. Until now a parallel Evolutionary Strategy has not been applied to this problem.

I study in detail the performance of the island Evolutionary Strategy when applied to the gap gene problem, including the effect of the distribution of individuals across islands, and demonstrate that the per-island speed of the algorithm increases with number of islands. By splitting up the islands throughout a number of processors, I apply a new coarse-grained parallel Evolutionary Strategy to problem, and study how its performance varies depending on number of nodes. It is found that both the reliability and the speed of the algorithm increase with increased number of nodes, though the efficiency drops off beyond 20 nodes. Finally, I compare the performance of the parallel ES to Parallel Lam Simulated Annealing and find that the Evolutionary Strategy is both faster and more reliable than Simulated Annealing.

I discuss ways in which model fitting by optimisation can progress in the future, including the potential for specific improvements that could be made to both the Evolutionary Strategy and Simulated Annealing. I conclude that as well as its superior performance to Simulated Annealing, the Evolutionary Strategy also has more possibilities for future developments, and I discuss such improvements, including asynchronous algorithms, hierarchical algorithms and hybrid algorithms.

Contents

1	Introduction	3
2	Background	5
2.1	The Problem	5
2.1.1	The Model	5
2.1.2	The Data	6
2.2	Optimisation and the Inverse Problem	7
2.3	Evolutionary Algorithms	7
2.3.1	A Brief History of Evolutionary Algorithms	7
2.3.2	Formulation of Evolutionary Algorithms	9
2.3.3	Parallel Evolutionary Algorithms	10
2.3.4	EAs in Gene Networks	12
2.4	Simulated Annealing	12
2.4.1	A Brief History of Simulated Annealing	12
2.4.2	Parallel Simulated Annealing	14
2.4.3	SA in Gene Networks	15
3	Methods	16
3.1	Algorithms	16
3.1.1	Restrictions to Search Space	16
3.1.2	Parallel (μ,λ) -Evolutionary Strategy	16
3.1.3	Parallel Lam Simulated Annealing	18
3.2	Implementation	19
3.2.1	Software	19
3.2.2	The Message Passing Interface	20
3.2.3	The Darwin Cluster	21
4	Results	22
4.1	General Procedure	22
4.2	Island Dynamics	22
4.3	Parallel Efficiency of the Evolutionary Strategy	24
4.4	Parallel Lam Simulated Annealing versus the Parallel Evolutionary Strategy	28
5	Discussion	31
5.1	The Parallel (μ,λ) Evolutionary Strategy	31
5.2	Simulated Annealing vs Evolutionary Strategy Efficiency	32
5.3	Tuning	33
5.4	Future of the Algorithms	34
6	Conclusion	35
7	Acknowledgements	37
	References	38

1 Introduction

The driving aim of systems biology is the understanding of complex interacting systems, and a powerful tool for this is the reverse engineering of such systems. In the reverse engineering approach, we attempt to use data to directly infer parameter values for a model to an entire system, and analyse this model to further our understanding; this differs from the traditional approach of building up a larger picture through individually measured simple interactions. One of the most exciting areas is the reverse engineering of gene regulation networks; exactly how genes and their gene products interact to regulate expression, with the overall result of allowing a range of behaviours to occur, such as developmental patterning, cell differentiation and responses to environmental changes. Reverse engineering in this context means inferring an underlying gene network, which will contain information on whether each gene activates, represses or otherwise affects other genes.

There are multiple approaches to both the experimental and the computational side of this problem. Gene expression microarray data has been used successfully to infer regulatory aspects of gene networks⁶², and the relationship between the statistical techniques involved in microarray analysis and those used in network theory has been given significant theoretical study²³. Bayesian network models have also been successfully applied to gene expression information from both microarrays⁵⁰ and intracellular flow cytometry data⁶⁰.

One drawback of many network inference methods is that they do not consider how expression changes over time or throughout the organism being studied, as they generally average expression together across both time and space. While time-series of microarray data can be used to fit network models that have a time component, using either difference equations⁴⁷ or Dynamic Bayesian networks⁴⁹, it is very difficult to measure spatial differences in gene expression throughout an organism with microarrays to reasonable resolution. While it is possible to use microarray analysis on single cells³⁰, it takes a significant period of time per cell, and does not allow data collection on a significant enough scale for network analysis. In addition Bayesian network models have not yet proved effective for spatio-temporal modelling. However, there are many systems for which both space and time are integral aspects; for instance in pattern formation, or pathways that involve intercellular signalling, and thus a spatio-temporal approach is required. It has even been shown that spatial localisation is important in many pathways of single celled organisms⁶³.

I am going to consider a technique which allows the inference of explicitly spatio-temporal gene networks. This approach involves collecting spatial data at nuclear resolution for various stages of development by confocal scanning of immunofluorescent stainings for particular gene products. The gene networks are inferred by fitting a spatial gene network model, a set of deterministic non-linear differential equations that include parameters that specify how genes activate or repress each other, from which we can infer the network structure. These parameters are phenomenological, in the sense that they summarise the effect of transcription factors as simple values; as such, it is not possible to estimate them individually, and thus the entire model must be fitted to the data simultaneously, and thus we employ a global optimisation procedure to reduce the least-squares difference between the model's predictions and the data. This technique has been used successfully to gain insights into gene networks in which spatial patterns develop over time^{56 27}. This method had the advantage of producing an explicit model of how the gene products change over time, which can be used to investigate aspects such as robustness, and assigning specific expression features to the action of particular regulatory interactions.

The only developmental system that this approach has been applied to so far is metameric pattern formation in the early *Drosophila* embryo; the process that determines where the segments will form. The general procedure by which this occurs is well understood³; a number of maternal mRNAs become localised during pre-fertilisation egg development; protein gradients then form by diffusion from their localised source after translation. One to three hours later, at the blastoderm

stage (at which body segments become determined), these maternal factors regulate the expression of a set of transcription factors called the gap genes. The gap genes in turn regulate a further set of transcription factors, the pair-rule genes, which finally regulate the position of the segment polarity genes, which mark where the segments will appear.

I use the term ‘regulate’, rather than ‘determine’; while it was once believed that each layer of the hierarchy was fully determined by the layer above it (gap genes by maternal gradient, segment polarity genes by pair-rule genes, and so on), it is now known that such vertical interactions are insufficient to explain the pattern formation of their gene products²⁶. To fully explain the observed patterns, we need to consider the (largely repressive) horizontal interactions, i.e. the complex gene regulatory networks between genes on the same layer. It is to this end that we model the full spatio-temporal behaviour of the gap genes, and parameterize the model using gene expression data for the gap genes and maternal gradients throughout the embryo and across developmental time; from the model’s parameters, we can infer the gene network.

One of the main problems with this approach is that the number of parameters for the model, and thus the time required to run the optimisation and fit the model to the data, increases rapidly as more genes are considered. It is thus extremely important to design efficient global optimisation algorithms to keep up with the ever-increasing scope of systems biology; efficiency in this case means both directly increasing the speed of the algorithms, as well as allowing the algorithms to run efficiently in parallel. The most commonly used algorithm for this purpose has been Parallel Lam Simulated Annealing^{32,33,9}; however, it has been shown¹⁴ that an appropriate Evolutionary Strategy, the so-called island-ES, can be used to solve the problem more rapidly.

Up to this point, the main advantage of the Simulated Annealing approach for the gene network optimisation problem was that it had a proven efficient parallel algorithm, whereas no parallel version of the island-ES has been applied to this problem before. In the hope of harnessing the same parallel utility that has been successful for Simulated Annealing, I have developed such a parallel island-ES algorithm.

The focus of this study will be on the relative efficiencies of Parallel Lam Simulated Annealing and the parallel ES, when applied to the gap gene problem. While I specifically focus on their application to gene regulatory networks in general, and the gap gene network in particular, I also aim to place this in the context of optimisation by Evolutionary Algorithms (EAs) and Simulated Annealing (SA) in general. To this end, I begin with background to the problem, including a description of the gene network model and the *Drosophila* embryo data, and with a history of Evolutionary Algorithms and Simulated Annealing in general.

I go on to investigate further the application of the island-ES to the gap gene network problem, especially with regards to how the number of islands, and the way population size scales with number of islands, affects how effective the algorithm is. I implement the parallel ES algorithm on a large computing cluster in order to study how the efficiency of the algorithm scales with number of processors. Finally, I compare the parallel ES to Parallel Lam Simulated Annealing when applied to the same problem; such an investigation has not been performed on an actual data-set before, as previous studies have focused on bench-mark tests of simulated problems⁴⁴. In all cases, I investigate both the speed and the reliability of the algorithm.

Finally I discuss implications for future gene network problems and model fitting by optimisation in general.

2 Background

2.1 The Problem

2.1.1 The Model

Mjolsness et al. ⁴³ developed a spatio-temporal model of gene concentrations during animal development, based on the recursive neural net model of Hopfield ²². They formulated a version of this model for gene concentrations in gene networks, and specifically for gene expression in the *Drosophila* blastoderm. The production rates of all gene products are assumed to be influenced by concentrations of the other gene products in the cell, as well as by external inputs. There are additional dimensions to consider in gene networks that are not present in neural networks; each cell nucleus can have different protein concentrations, and thus we have to model the concentrations of different gene products in different spatial locations. As we are considering segmentation, and as anterior-posterior pattern formation is relatively independent of dorsal-ventral pattern formation, only the anterior-posterior dimension was included in the model, with N_{nuc} nuclei considered to be laid out in a line, each one having an index $i \in \{1, \dots, N_{nuc}\}$, thus giving $N_{nuc} \times N_g$ gene expression values $\{g_i^a\}$.

The model is given by

$$\frac{dg_i^a}{dt} = R_a \Phi\left(\sum_{b=1}^{N_g} W_a^b g_i^b + m_a g_i^{Bcd} + h_a\right) - \lambda_a g_i^a + D_a(n) [(g_{i+1}^a - g_i^a) + (g_{i-1}^a - g_i^a)] \quad (1)$$

The three terms correspond to protein production, decay and diffusion, and I will discuss each separately.

The production term is equal to some fraction of the maximum production rate R_a . This fraction is given by the sigmoid function

$$\Phi(u_a) = \frac{1}{2} \left[\left(\frac{u_a}{\sqrt{u_a^2 + 1}} \right) + 1 \right] \quad (2)$$

which takes on values between 0 and 1. In this case $u_a = \sum_{b=1}^{N_g} W_a^b g_i^b + m_a g_i^{Bcd} + h_a$, which contains three terms. The first term $\sum_{b=1}^{N_g} W_a^b g_i^b$ is the sum of activation or repression by regulators b , where the contribution of each b on gene a is given by its contraction times by the weight of the connection between them W_a^b . This weight matrix W defines the gene interaction network; a positive value of W_a^b represents activation of gene a by gene b , a negative value represents repression, and a zero value represents no interaction. The second term $m_a g_i^{Bcd}$ is the effect of the concentration of the maternal factor Bcd, where m_a is the effect of the Bcd gradient on gene a and g_i^{Bcd} is the concentration of Bcd in nucleus i ; Bcd is laid down maternally during egg development, and is thus treated separate from the other genes. The final term h_a represents other factors, including ubiquitous maternal factors.

The decay term $\lambda_a g_i^a$ assumes that gene products decay exponentially, with a decay constant λ_a for gene a .

The diffusion term $D_a(n) [(g_{i+1}^a - g_i^a) + (g_{i-1}^a - g_i^a)]$ corresponds to Fickian diffusion, in which gene product diffuses in from neighbouring nuclei at a rate proportional to the difference between their concentrations, and $D_a(n)$ is the diffusion constant of each protein; as diffusion is proportional to the distance between the nuclei, $D_a(n)$ is a function of the number of cell divisions that have occurred prior to time t , as every time the cells divide the distance between their nuclei halves and thus the rate of diffusion doubles, giving $D_a(n) = D_a(n+1)^2$, where D_a is the diffusion constant at $n = 0$.

The model takes account of cell division. The lengths of interphase and mitosis occur according to a very well determined schedule¹¹, and are modeled using three rules. During interphase a continuous rule is applied, in which equation 1 holds. After this is a period of mitosis, during which a second continuous rule is applied, in which the production term of equation 1 is set to zero. Finally, a discrete division rule is applied, in which n is incremented, and each nucleus splits into two, with equal concentrations of all gene products. The precise timing is given in Jaeger et al.²⁶.

2.1.2 The Data

While the model alone can provide some insight into gene networks (not least of which is stating a complex hypothesis in one relatively simple equation), the model becomes especially useful when applied to experimental data. If we can determine the values of the various parameters of equation 1 for particular data, we can extract relevant information (such as the weights of the gene network). However, producing data at a high enough resolution, with quantitative measures of gene product concentration for each nucleus, is a non-trivial task.

Surkova et al.⁶⁴ discuss a method of producing spatio-temporal protein concentration data for segmentation genes in the early *Drosophila* embryo that can produce quantitative data at the resolution of individual nuclei. The method involves scanning fluorescently-stained embryos, and using a high-throughput image processing pipeline to extract the per-nucleus concentrations and combine them into a high-resolution time-series of integrated data.

The data collection consists of the fluorescent staining of fixed embryos using antibodies raised against three target transcription factors per embryo (one of which is always the pair-rule gene *eve*), as well as a nuclear stain to identify the positions of the nuclei. Each of the four fluorophores emit at a different wavelength, and thus can each be scanned separately using a confocal microscope at various depths. A large collection of embryos is scanned (in the case of Surkova et al.⁶⁴, 1600), in order to have a representative sample of gene expression data across the time series for all the genes of interest. Surkova et al.⁶⁴ scanned 15 genes, but we are only interested in genes required to study the gap gene network: they include maternal genes *bicoid* (*bcd*) and *caudal* (*cad*), and the gap genes *Krüppel* (*Kr*), *knirps* (*kni*), *giant* (*gt*), *hunchback* (*hb*) and *tailless* (*tll*).

The image processing pipe-line involves five stages. Firstly, each embryo image is segmented; the boundaries of each nucleus is determined using a combination of a watershed algorithm and an edge detection algorithm, and the location of each nucleus, along with the intensity of each gene product it was scanned for, is stored. Secondly, each image undergoes background correction to remove the signal from non-specific binding; this is performed based on the observation that non-specific binding follows a two-dimensional parabolic distribution⁴⁶, and thus it can be removed by fitting such a distribution to the intensity readings outside of the nuclei, and subtracting this from the overall signal. Thirdly, the developmental age of each image is determined, by visual inspection based on *eve* expression and embryo morphology. Fourth, the expression data are registered; the position of the *eve* stripes are determined using splines, and an affine co-ordinate transformation is applied to each image to minimise the over-all least-squared distance between the stripes, ensuring that they line up. Finally, the images are averaged together for each time-step and gene, in order to create an integrated data-set.

This approach has been used to create FlyEx⁵², an online dataset of segmentation gene expression data for *Drosophila*, and by fitting the gene network model to this data Jaeger et al.²⁷ were able to gain biological insight into the functioning of the gap gene network.

2.2 Optimisation and the Inverse Problem

There are essentially two problems to be addressed with gene network models. The first is the so-called *direct problem*, which consists of solving the equations that make up the model, using given parameter values that have been measured or guessed. Examples of questions we may address with such a solution are: what does the model predict in different circumstances (such as if certain gene products are removed, as in a gene knockout experiment), can we infer how robust the developmental processes are to various changes in parameters or variables (such as differences in maternal inputs), and how evolvable is the system⁵⁴? However, in many cases we have no measurements or accurate estimates for parameter values. Thus, we must address the *inverse problem*; assigning values to the various parameters in the model based on experimental data. For instance, the sections above discussed the existence of a spatio-temporal gene network model for early *Drosophila* genes, and an extensive dataset of spatio-temporal gene expression data; in order to make full use of both of these, we need to solve the inverse problem. This is often a non-trivial task.

We wish to find what values the particular parameters have to take to best explain the data. We can frame this as an optimisation problem in which we attempt to find the set of parameters \vec{x}_{opt} that gives the minimum value of an objective function $E(\vec{x})$. We define the objective function for a particular set of parameter values \vec{x} as the sum of squared distances between the values predicted by the model and the experimental data for all data-points we have, or

$$E(\vec{x}) = \sum_{(i,t,a)} [g_i^a(t, \vec{x})_{model} - g_i^a(t)_{data}]^2 \quad (3)$$

where the sum is over all i, t and a for which we have data, and where

$$\vec{x} = (W_1^1, \dots, W_{N_g}^{N_g}, m_1, \dots, m_{N_g}, h_1, \dots, h_{N_g}, \lambda_1, \dots, \lambda_{N_g}, D_1, \dots, D_{N_g}, R_1, \dots, R_{N_g}) \quad (4)$$

is the vector of all parameters to be estimated, with a length of $N_g(N_g + 5)$.

Specifically, we want to use the optimisation procedure to fit the model for a 6 gene problem. A dataset exists, created using the method described above, that contains expression data for 6 genes (plus the maternal gradient of Bcd) across 58 nuclei and 9 equally spaced time-classes from cleavage cycles 13 to 14 of early *Drosophila* development. The genes of which we wish to model the expression are the 5 gap genes *hb*, *Kr*, *gt*, *kni* and *tll* and the maternal gene *cad*.

As already mentioned, optimisation for complex problems such as these is non-trivial. The system of equations is non-linear with a large number of parameters to be estimated, and the fitness landscape is not smooth. A full search is not possible, and a local search (which moves downhill until it finds the lowest energy) is likely to get stuck in a local minimum. Instead, we need an approach that searches the entire landscape, but does so in an efficient way. Many so-called Global Optimisation methods have been developed to overcome this problem. I will discuss two broad classes of approaches; Simulated Annealing, and Evolutionary Algorithms.

2.3 Evolutionary Algorithms

2.3.1 A Brief History of Evolutionary Algorithms

I am discussing Evolutionary Algorithms as used for searching or optimisation, not in order to simulate evolution for its own sake.

Algorithms that were in effect elementary evolutionary optimisation algorithms were in use as early as the 1950s. For instance, Richard Friedberg published a pair of papers^{15 16} on his Learning Machine; a computer program that had elements of its programming that underwent random

changes and selection, in order to adapt to the problem it has been set. Another early and innovative example of the application of evolution to optimisation was George Box's "Evolutionary Operation" scheme, for use in industrial plants⁶. This was explicitly modeled on biological evolution, and involved making random changes in the day-to-day running of the plant, and selecting which changes to carry on with based on the success of the plant.

While these early evolutionary algorithms had modest success, they tended to get trapped in local minimum. In particular, Friedberg's algorithm could take up to 100 times longer to converge on a good solution than random chance alone did, due to the inability to escape troughs in the fitness landscape.

Hans Bremermann made this problem explicit in a talk at the Self-Organising Systems Conference in 1962⁷. He noted that the then current models of evolution assumed a single fitness for a given gene, independent of the other genes present, and that this is not generally true, either in biological evolution or in optimisation. He proposed the introduction of recombination, the "mating game", into evolutionary algorithms in order to overcome this problem, and showed that for certain problems it stopped the solution becoming trapped.

In general, the EAs that were developed after this point fell into three categories: Genetic Algorithms (GAs), Evolutionary Programming (EP) algorithms and Evolutionary Strategies (ESs).

Genetic Algorithms grew out of a general theory of adaptive systems formulated by John Holland, first outlined in the early 1960s²⁰, and more specifically from his discussion of the application of this adaptive theory to a number of separate systems, including optimisation, published as *Adaptation in natural and artificial systems* in 1975²¹. The technique was applied to parameter optimisation by Kenneth De Jong²⁹. Inspired by the genetic code, the procedure involves representing the entire parameter set as a long binary string by concatenating binary representations of the parameters together. Mutation occurs by randomly inverting single bits at a low rate, and recombination (occurring at a higher rate than mutation) is performed just like genetic recombination, i.e. by selecting two individuals at random and swapping one or more sections of the binary string between the two. The offspring thus produced are randomly selected, with the probability of each offspring surviving being related to their fitness.

In 1965 Lawrence Fogel and colleagues published a book, *Artificial Intelligence Through Simulated Evolution*¹³, in which they described an EA, which they called Evolutionary Programming. They applied it to the evolution of finite state automata, to be used in prediction (a common application for many EAs), but it has been applied widely since. The EP algorithm does not include recombination, but it does have a number of mechanisms for ensuring that the search space is well sampled. Firstly, mutation is biased, and is dependent on the fitness (values with high fitness change less). Secondly, selection is performed by a method called q -tournament selection; each individual has its fitness compared with the fitness of q other randomly picked individuals, and is assigned a score based on how many of these individuals it is fitter than. The individuals with the highest such score are then selected. This technique is stochastic but elitist; the best individual will be selected for the next generation, but there is also a chance that lower fitness individuals will also be selected (with the parameter q determining how likely this is). This prevents the algorithm becoming trapped in a local minimum.

The Evolutionary Strategy was originally devised at the Technical University of Berlin for designing hydrodynamic devices, with changes applied directly to experimental devices (for instance, Lichtfuss³⁴ used such an ES to optimise the flow rate in a bent pipe). The ES was first implemented on a computer by Hans-Paul Schwefel⁶¹. ESs are explicitly continuous algorithms, with operations specifically designed to handle continuous variables. Evolutionary Strategies contain a number of features that make them well suited to optimisation; each individual has its own mutation rate associated with it, allowing the mutation rate to adapt as the individuals are selected. They also involve recombination between parameter values, rather than between binary representations as in the GA; offspring are produced from two or more individuals by giving each

offspring parameter a differently and stochastically weighted mean of its parent's values. ESs are relatively general, and exactly how recombination occurs varies across implementations. Finally, ES selection is deterministic: the top λ individuals are selected from the population.

By far the most popular type of EA in use today is the Genetic Algorithm, largely due to its versatility. However, it is worth noting that for a large number of optimisation problems, ESs tend to outperform GAs. In his 1996 book *Evolutionary Algorithms in Theory and Practice*⁵ Thomas Bäck performed a large empirical comparison of the three different approaches across a range of objective functions, including both continuous and step functions, judging them based on convergence velocity and reliability, and found that Evolutionary Strategies tended to perform best, followed by Evolutionary Programming and finally Genetic Algorithms. Surprisingly, given the popular claim that GAs tend to perform better on discrete problems than other EAs, the ES out-performed the GA even on a step function. Similar results were obtained in another related analysis⁴. Both these references also give a more detailed history of the specific algorithms, and their different formulations.

2.3.2 Formulation of Evolutionary Algorithms

We can specify the general Evolutionary Algorithm, of which all the EAs described in the previous section are a special case. We will use the notation used by Bäck and Schwefel⁴. Evolutionary Algorithms operate on individuals $\vec{a} \in I$ (where I is the set of all possible individuals), and on object variable vectors $\vec{x} \in \mathbb{R}^n$ (these vectors are either associated with or identical to the individuals in the population). We have the objective function to be optimised, $f : \mathbb{R}^n \rightarrow \mathbb{R}$, and a fitness function $\Phi : I \rightarrow \mathbb{R}$, which will be dependent on, but not necessarily identical to, the objective function.

We define a parent population at generation t as $P_p(t) = \{\vec{a}_1, \dots, \vec{a}_\mu\}$ where $\mu > 1$ is the size of the parent population, which is initialised in some way to create $P_p(0)$. During each generation the parent population undergoes mutation and recombination to produce an offspring population $P_o(t) = \{\vec{a}_1, \dots, \vec{a}_\lambda\}$, where $\lambda > 1$ is the size of the offspring population. Fitnesses are then determined for the individuals in the offspring population, and some combination of individuals from the parent and offspring populations are selected to produce a new parent population $P(t+1)$. This is repeated until some stop criterion is achieved.

Recombination and mutation are achieved by a recombination function $r_{\Theta_r} : I^\mu \rightarrow I^\lambda$ and a mutation function $m_{\Theta_m} : I^\lambda \rightarrow I^\lambda$, where Θ_r and Θ_m are parameter sets. We will take recombination as occurring first, though this does not have to be the case; if mutation goes first then it maps $I^\mu \rightarrow I^\mu$. The mutation function treats each individual separately, and thus can be reduced to the local operator $m'_{\Theta_m} : I \rightarrow I$. In contrast, recombination makes use of two or more individuals in the parent population to produce each individual in the offspring population, and thus is reduced to the local operator $r'_{\Theta_r} : I^\mu \rightarrow I$.

The fitness associated with each individual is calculated using the fitness function Φ , and selection is achieved by the selection function $s_{\Theta_s} : (I^\lambda \cup I^\mu) \rightarrow I^\mu$. A termination function $\iota : I^\mu \rightarrow \{true, false\}$ is used to determine whether the algorithm should perform another step or not.

The general Evolutionary Algorithm can thus be specified as

1. Initialise the parent population: $P_p(0) = \{\vec{a}_1(0), \dots, \vec{a}_\mu(0)\}$
Evaluate the parent population: $\{\Phi(\vec{a}_1(0)), \dots, \Phi(\vec{a}_\mu(0))\}$
2. Check termination criteria $\iota(P_o(t))$, and finish if true
3. Produce the offspring population from recombination: $P_o(t) = r_{\Theta_r}(P_p(t))$
4. Mutate the offspring population: $P'_o(t) = m_{\Theta_m}(P_o(t))$
5. Evaluate the offspring population: $\{\Phi(\vec{a}_1(t)), \dots, \Phi(\vec{a}_\mu(t))\}$
Produce the new parent population by selection: $P_p(t+1) = s_{\Theta_s}(P'_o(t))$
6. Increment the generation $t \rightarrow t+1$ and return to 2.

The individual types of EA described above can be classified based on this system. For instance, we can specify all the individual variations that make up the canonical EP algorithm as follows. Firstly, it has a fitness function $\Phi(\vec{a}) = \delta(f(\vec{a}), \kappa)$, where δ is a scaling function that keeps the fitness positive and κ is an optional randomisation element. Secondly, it has no recombination stage. Thirdly, it has mutation determined by the local operator $m'_{\{\beta_1, \dots, \beta_n, \gamma_1, \dots, \gamma_n\}} : I^\mu \rightarrow I$, $m'(\vec{x}) = \vec{x}'$, with the pre-mutated object vector x and the post-mutated object vector x' related by

$$x'_i = x_i + N(0, 1)\sqrt{\beta_i\Phi(\vec{x}) + \gamma_i} \quad (5)$$

for $i \in \{1, \dots, n\}$, where β_i and γ_i are mutation parameters. Finally, selection is performed by q -tournament selection (as described above).

Specification of the Evolutionary Strategy that we shall use is given in the “Methods” section below.

2.3.3 Parallel Evolutionary Algorithms

Evolutionary Algorithms are inspired by biological evolution, and biological evolution is a massively parallel operation: every individual can be considered an evaluation of genetic and epigenetic parameters, through the fitness function of development and competition, and countless numbers of individuals perform this evolution simultaneously. Allowing EAs to mimic this function becomes increasingly important as grid and cluster computing become more powerful. At any given time for any top-line computing technology it is generally cheaper to buy two computers than to buy one that is twice as fast, or as William Gropp put it so succinctly, “to pull a bigger waggon, it is easier to add more oxen than to grow a gigantic ox”¹⁹.

There have essentially been three approaches to parallel EAs; the master-slave approach, the diffusion approach, and the migration approach. Most of the examples discussed here are of parallel GAs, which are by far the most numerous (and have been extensively reviewed^{8,2}).

The master-slave approach is the simplest form of parallel GA. A master node implements all aspects of the GA itself, other than calculating the fitness function; this has the advantage of introducing no new parameters. This approach is used when calculating the fitness function is a very costly operation compared to recombination and mutation. This has been especially useful in Genetic Algorithms; the string representation of parameters makes recombination (as simple crossing over) and mutation (as bit inversion) very simple, and can thus be easily run on the master node. Fogarty and Huang¹² reported a speed-up for a Genetic Algorithm, specifically a learning rule for a pole balancing application, when implemented on a parallel computer made

up of transputers (small, cheap processors that were popular in the 1980s, made to be connected together to form larger parallel computers). They noted that communication overhead became too high for higher implementations, though this was probably due to both the master-slave nature of the implementation, and the limitations of transputer communication. A master-slave ES with similarities to migration ESs is discussed below, along with migration.

The diffusion approach (also called the ‘fine grained’ approach) concentrates on producing a large, interacting population over a number of nodes; often with one or few individuals per node. The population is often spatially structured, with selection and recombination occurring between individuals that are nearby (i.e. on nodes that are well connected). Thus good solutions propagate in space, and can merge with other solutions as they migrate across nodes; it has been proposed that this intermingling of spatially localised solutions can help avoid local minima¹⁸. The method does have the disadvantage of changing the EA it is based on heavily, with often unknown and certainly highly implementation dependent changes to results.

The literature on fine-grained GAs is large. There have been some very successful implementations; an example of note being ASPARAGOS¹⁸, one of the first. ASPARAGOS was implemented on a ring of transputers, each of which ran one individual, with a width of two individuals and a circumference of $N/2$ individuals (where N is the number of nodes); individuals interacted (recombined with, and were selected relative to) all individuals that they could reach by 2 moves (i.e. 7 other individuals). It was shown that ASPARAGOS (using 64 transputers) could get better solutions faster to large Travelling Salesmen Problems than both Simulated Annealing (the fastest algorithm in use at the time) and k-opt (the highest quality algorithm). Cantu-Paz⁸ reviews the success of other algorithms with other structures that followed, including grids, toruses and hypercubes, with varying degrees of success.

Rumarsson and Sigurdsson⁵⁸ produced a parallel ES for model selection in support vector machines which was half way between a fine-grained and a master-slave approach. It uses continuous selection and mutation, such that low fitness individuals in the population are persistently deleted and replaced by offspring of the fitter individuals; different processors asynchronously select the worst and best individuals not being accessed by another processor and perform the necessary operations (including evaluating the new individual and changing the ranks appropriately). They report good results for 4 nodes, though whether this algorithm will scale well beyond such a small number of nodes is currently unknown.

The migration approach, often called the ‘coarse-grained’ approach, involves running a number of separate and largely independent EAs, each on a separate processor, which occasionally exchange information with each other. Whereas the diffusion approach has much in common with mainland population biology, this approach is inspired by island population biology, with small, well mixed populations connected together by weak migration.

Once again there are numerous examples of coarse-grained parallel GAs. Early implementations are reviewed elsewhere⁸, and modern GAs are applied successfully in a wide range of applications. An impressive example of that is the demonstration by Pereira and Lapa⁵¹ that a parallel migration GA could be applied effectively to the design of the core of a nuclear reactor; also notable was that the communication overhead of the algorithm was so low that it could be implemented on an off-the-shelf Local Area Network, with little loss of efficiency.

It has been noted that ESs would be particularly well suited to the migration approach; Lohmann³⁶ notes that an island-ES (serial ESs that contain multiple populations) could solve problems that a standard ES could not. Ngom⁴⁸ produced a parallel ES for use in protein structure determination, which behaved mostly like a master-slave ES, but with coarse-grained elements. Selection is done in parallel, with each slave node evaluating a subset of the population; however, each slave node also performs a brief mutation-only ES in their subset to see if they can improve on the best result, with the best individual being taken up by the master node and used in the next round of the main ES. However, no extensive study has been done on the implementation

and efficiency of a purely migration based parallel ES.

Finally, it is worth noting that hybrid parallel EAs exist that combine these different approaches. These can sometimes be called hierarchical parallel EAs, as they often involve different levels of parallelism; for instance a hierarchical migration/master-slave hybrid, involving a number of master-slave parallel EAs migrating amongst each other. Lim et al.³⁵ implemented a hierarchical GA to be used in Grid computing (which are often highly heterogeneous), which they successfully applied to an airfoil shape optimisation problem using a representative collection of clusters.

2.3.4 EAs in Gene Networks

The earliest example of the use of an EA in gene network optimisation was by Marnellos³⁸, who used a coarse-grained parallel GA, as well as Simulated Annealing, to fit a gene network model for *Drosophila* neurogenesis; the GA had a faster initial convergence, though was ultimately slower than Simulated Annealing.

Mendes and Kell⁴¹ compared two techniques, a local search and Simulated Annealing, to fit a model of HIV Proteinase inhibition. They noted that the local search method often failed to find a good solution, but that Simulated Annealing took a very long time; though they did not attempt to solve the problem with an EA, they did apply the Evolutionary Programming algorithm to similar problems, and on the basis of those results suggested that EP may be faster than SA but more reliable than the local search for the Proteinase problem.

Mendes⁴⁰ used an EP algorithm and two local search methods included with the kinetics package Gepasi, to fit a gene-expression/enzyme kinetics model to noiseless simulated data. The model involved three enzymes, their mRNAs, and products that they catalyse (all of which either activate or repress transcription). They found that the EP algorithm fitted the data far better than either of the local searches, but found that a hybrid approach, with an EP followed by a local search, could achieve far better results than either could alone. However, the lack of noise may well exaggerate the effectiveness of the local search, as it allows it to converge on a theoretically perfect solution, which is not attainable in real applications.

The power of EAs, and especially ESs, to fit gene models was demonstrated by a comprehensive comparison, performed by Moles et al.⁴⁴, of off-the-shelf Global Optimisation methods applied to the same gene network problem as Mendes⁴⁰. They included a number of EAs, including ESs and a hybrid GA/adaptive search algorithm, as well as a range of other Global Optimisation algorithms, and found that the EAs, and the ESs in particular, performed far better than the other methods. However, it did not include Simulated Annealing, and again did not allow for noise.

Mendes and Banga³⁹ implemented a hybrid ES/local search algorithm on the same gene network; this approach increased the algorithm speed by an order of magnitude over any of the methods used in Moles et al.⁴⁴. They also showed that this hybrid approach was effective even when the simulated data was subject to noise.

Motivated by these simulation studies, Fomekong-Nanfack et al.¹⁴ used a state-of-the-art island-ES, followed by a local search, to fit the gap gene network model to gene expression data. This method was shown to produce solutions faster and more reliably than previous Simulated Annealing applications to the same problem, without sacrificing solution quality.

2.4 Simulated Annealing

2.4.1 A Brief History of Simulated Annealing

What eventually became Simulated Annealing began as the Metropolis Method, a method to numerically sample from a Boltzmann distribution⁴². The idea behind it was that if you wished

to calculate a value F for an ensemble of particles at equilibrium, you needed to average across all possible states (for instance, combinations of positions) that the ensemble can be in, weighted by the probability of it being in that state. The probability of an ensemble taking on a particular state is dependent on that state's energy E , and is proportional to $\exp(-E/kT)$, where T is the temperature of the ensemble and k is Boltzmann's constant. Averaging across all possible states is often impossible, and thus states are often randomly sampled; however, sampling at random is likely to be inefficient, since it will involve sampling states with very high energy, and thus low contributions to the average value of F .

The Metropolis Method is a more efficient way of sampling states; instead of sampling states uniformly and weighing the average by the ensemble probability, the Metropolis Method samples states with a probability $\exp(-E/kT)$, allowing you to merely average all the sampled values of F together as an unweighted mean. The algorithm works by starting with a random state, and then making one or more random changes to it: if a change decreases the energy, we accept the move, if it increases the energy, we accept the change with a probability $\exp(-\Delta E/kT)$, where ΔE is the change in energy. Once a move is accepted, we sample F at the point we have moved to and then perform another move from there; if it is not accepted, we perform another move from the old position. Over time, this method samples all of the energy space, but spends more time at the low energy parts of the state-space.

This property; sampling the entire state-space, but preferentially sampling lower energies, is a useful property in optimisation, as we can equate "energy" with objective function to be minimised. The degree to which lower energies are preferentially sampled is given by the temperature, T ; when T is high, increases in energy happen nearly as often as decreases, and when T is low energy increases become very unlikely. When the temperature is zero, the search only moves downhill. Kirkpatrick et al.³¹ stated that, by starting off with a high temperature, and decreasing it over time, it would be possible to find the globally lowest energy state. They drew an analogy with physical annealing; the gradual lowering of the temperature of a liquid substance allows the formation of highly stable (low energy) crystals. It was demonstrated that a Simulated Annealing algorithm could be used to find a state that gives a low energy; be that low energy state a good solution to a Travelling Salesman Problem, or a good positioning of components and wires on a computer.

Exactly how to decrease the temperature is not immediately obvious. Geman and Geman¹⁷ proved analytically that a temperature schedule of $T(t) = T_0/\ln(1+t)$ would converge on a global optimum, but this temperature decrease was so slow as to make the algorithm very inefficient. It was later shown analytically⁶⁵ that a temperature schedule of $T(t) = T_0/(1+t)$ would also be guaranteed to converge, providing moves were drawn for a Cauchy distribution (which allows occasional large moves); this was called "Fast Simulated Annealing". It was later argued that so-called "Very Fast Simulated Annealing" would reliably allow even faster temperature decrease²⁴, with a temperature schedule $T(t) = T_0 \exp(ct^{1/N})$, where N is the number of parameters being sampled; while this hasn't been proved to always converge, it has been applied successfully to many problems.

However, we ideally want the temperature decrease to be optimal at any given time; the change in temperature at any given point should be based on the measured properties of the system at that time. Lam and Delosme³² derived a temperature schedule based on keeping the system in "quasi-equilibrium". A Simulated Annealing process functions best at thermal equilibrium (i.e. it is, at any point, sampling from the Boltzmann distribution associated with the current temperature T); however, moving the temperature at a non-infinitesimal rate disturbs this equilibrium. The Lam method defines the system to be in "quasi-equilibrium" if the system is within certain bounds of thermal equilibrium; how close it needs to be is determined by a quality parameter λ . The Lam schedule is derived by decreasing the temperature such that energy decrease is maximised, without violating the quasi-equilibrium, and gives a temperature decrease at each time step dependent on

the variance of the energy and the acceptance ratio of previous moves. This energy decrease is given explicitly in the “Methods” section below.

Lam and Delosme³² applied Lam Simulated Annealing to the Travelling Salesman Problem and the Graph Partition Problem, and found that it outperformed a number of the leading methods for solving both of these problems.

2.4.2 Parallel Simulated Annealing

As described above, the SA algorithm seems to be an inherently serial algorithm. However, as parallel computing became increasingly important, many attempts were made to produce a parallel version.

One such approach is based on the method of Macready et al.³⁷, instead of generating moves for all parameters at each time step, we have various processors that each change a randomly generated subset of the parameters, and occasionally pool their solutions. The degree of parallelism is determined by a parameter $0 < \tau \leq 1$, which is the probability of any given parameter being changed on each processor; if we have N parameters, $\tau = 1/N$ corresponds to an average of only one parameter per processor changing, and $\tau = 1$ corresponds to all parameters changing (i.e. the serial case).

Macready et al.³⁷ applied such an algorithm to select a state, a vector of binary variables $\vec{s} = (s_1, \dots, s_N)$ given an NK energy function

$$E_s = \frac{1}{N} \sum_{i=1}^N E_i(s_i; s_{i_1}, \dots, s_{i_K}) \quad (6)$$

where s_{i_1}, \dots, s_{i_K} are the K ‘neighbours’ of s_i and each value of the subenergy $E_i(s_i; s_{i_1}, \dots, s_{i_K})$ for the 2^{K+1} different states are preselected from a uniform distribution $[0, 1)$. Thus, the parameter K determines how independent the subenergy E_i of each binary variable s_i is of all other binary variables; if $K = 0$, then they are fully independent, and if $K = N - 1$ then each variable effects the energy of all others.

They applied the Simulated Annealing algorithm (with an exponential temperature schedule) to this problem for various values of τ and K when $N = 500$, and measured the final energy. They found that the parallel algorithm functioned perfectly well for low values of K and high values of τ , but for values of K larger than 2 there was a critical value of τ , above which the energy catastrophically failed to converge; these values rapidly approached 0 as K went above about 10. As high K values represent non-linearity in the energy function, and non-linear energy functions are the problems most likely to have SA applied to them, this suggests a fundamental problem with this approach.

A different approach involves mixing of states. In this method, each processor runs an SA process, but occasionally solutions are mixed across processors, with lower energy processes being more likely to be copied to a different processor; this is a similar process to selection in an EA, and has a similar effect. An early attempt at this was made by Aarts et al.¹, which pictured the SA processes as propagating through processors, mixing with other processes, and dying; K processors were connected together in a directed ring structure, and an initial SA process was seeded on one of them. This process would survive for L iterations, and every L/K iterations it attempts to propagate to the next processor along in the ring; if there was no process on the next processor, it would copy itself to this processor, and the new copy would start up as a new process, behaving just like the old one. If there was already a process on the target processor, the process on the target processor takes up the state of the parent process with a probability roughly equal to $\exp(E_p/T)/[\exp(E_p/T) + \exp(E_t/T)]$, where E_p is the energy of the parent process and E_t is the energy of the target process (this is not entirely correct, as the algorithm in fact allows

different processes to have different temperatures, but it captures the spirit of it). When correctly tuned, this algorithm allows all the processes to remain in quasi-equilibrium, and the increase in sampling allows you to decrease the temperature faster, and thus speed up the algorithm.

Aarts et al.¹ applied this algorithm, along with another less successful parallel SA, to the Travelling Salesman Problem, and found a 75% efficiency of speed-up on 8 processors (75% efficiency relative to the time the algorithm would take on a serial computer 8 times as fast). Other similar algorithms have been applied with variable effectiveness, as reviewed elsewhere⁹.

In order to bring the full effectiveness of the Lam schedule to bear, Chu et al.⁹ detailed a mixing-of-states parallel SA algorithm that included a collective pooling of statistics (i.e. mean and variance of the energy over time). The algorithm involves K SA processes spread across K processors, and every τ/K iterations of the algorithm (i.e. τ samplings across all processors) all processors broadcast their statistics, averaged over the last τ/K iterations. A time-decaying running average of the statistics is kept, and this averages is used to keep an optimal Lam schedule by treating all the processes as one process. The algorithm involves full mixing of states, in which every τm iterations each process broadcasts its energy, and each process receives a new state from process i with a probability $\exp(E_i/T) / \sum_{j=1}^K \exp(E_j/T)$. m needs to be small enough to keep all processes in quasi-equilibrium, but large enough to allow different processes with the same initial conditions after mixing to become decorrelated enough to search different parts of the state-space.

Chu et al.⁹ demonstrated that this algorithm could keep near-100% efficiency for up to 50 processors, and 80% efficiency for 100 processors, when used to fit the gap gene network model described above to data simulated from the model. However, this measurement failed to take into account communication time, as it only measured processor cycles. In addition, the algorithm requires the number of processors K to be a factor of, and no larger than τ , which can restrict the number of processors used.

2.4.3 SA in Gene Networks

The first use of Simulated Annealing to fit a gene network model was Reinitz et al.^{56 57}, which used Lam Simulated Annealing to fit the *Drosophila* gene network model for segmentation genes to spatio-temporal data. They compared the predictions of this fitted model to known *Drosophila* biology, and concluded that the model was accurate in many parts, and where it was not the model was missing known regulators (such as in the far anterior and posterior regions). They used this method to produce a model of how *eve* stripes form via the action of activation and repression by gap genes⁵⁵. This did not come cheap, however; fitting the 5-gene model took around a week of computation.

As mentioned above, Chu et al.⁹ applied the Parallel Lam SA to simulated data from the gap gene network model. However, they only used a 2-gene model; in addition, they did not include noise, and fixed the parameters for one of the genes, thus only fitting seven parameters. The same algorithm was successfully used by Jaeger et al.^{27 26} to parameterize a full 6-gene, 66-parameter model from gene expression data; they used the fitted model obtained to analyse how gap gene domains migrate after formation in the early embryo. On a 10-node Beowulf cluster of 2.4GHz machines, the model fitting took between 8 and 160 hours for each run; this was likely to be largely due to communications overhead, as the cluster ran internode communication via Ethernet. Jaeger et al.²⁸ also applied Lam SA in serial to a simpler set of network models (6-9 parameters each) to study early gap gene circuits.

Mendes and Kell⁴¹ reported the use of Simulated Annealing in parameterizing a model of HIV Proteinase inhibition given experimental data. They found that Simulated Annealing gave a better solution than a local search, but took about 750 times longer. However, the authors do not mention what SA algorithm was used, and only refer to Kirkpatrick et al.³¹, suggesting that they were not using a fully up-to-date method.

3 Methods

3.1 Algorithms

3.1.1 Restrictions to Search Space

We do not need or want to search the entire parameter space; there are certain values that we know *a priori* that parameters should not take, and we do not want these parameters to grow without bound or shrink to zero in the model. To represent this, we may either introduce absolute criteria that prevent the optimisation from taking on those values, or we can produce a penalty function, which increases as the parameters move further into areas of unacceptable solutions. The penalty function may be added to the energy function (as occurs in Simulated Annealing), or it may be kept separate and handled in an algorithm-specific way (as occurs in the Evolutionary Strategy).

For the gene network problem, we give a penalty function to the parameters associated with protein production of

$$\Pi(\vec{x}) = \max \left(0, \exp(\Lambda \sum_a [\sum_b (W_a^b v_{max}^b)^2 + (m_a v_{max}^{Bcd})^2 + (h_a)^2]) - \exp(1) \right) \quad (7)$$

where Λ is a control parameter, and v_{max}^b is the maximum observed intensity of gene b . The justification for using this penalty function is that it remains 0 for

$$\sum_a \left[\sum_b (W_a^b v_{max}^b)^2 + (m_a v_{max}^{Bcd})^2 + (h_a)^2 \right] < 1/\Lambda \quad (8)$$

i.e. when the absolute value of the total regulatory input is below a certain threshold, but rises steeply outside of those bounds. We took $\Lambda = 0.001$.

The production, decay and diffusion constants, R_a , λ_a and D_a are given explicit limits, such that any parameter value outside these limits is considered unacceptable and rejected. The ranges are $10 < R_a < 30$, $0 < D_a < 0.3$ and $5 < \lambda_a < 20$ for all a .

3.1.2 Parallel (μ, λ) -Evolutionary Strategy

The EA that we will be using is a Parallel (μ, λ) -ES, and is a parallel version of the Island ES algorithm that Fomekong-Nanfack et al. ¹⁴ applied to the same Gene Network problem. The only change that has been made to the algorithm is to allow the island populations to be placed on separate processors. Optimisation parameters are taken from Fomekong-Nanfack et al. ¹⁴.

The island algorithm has N_{isl} different populations, each with a parent population size μ and an offspring population size λ , and each of which acts as a separate ES, with each one separately initialised and with selection, recombination and mutation performed only within populations. Each population runs on a separate processor, and thus all these operations can be performed locally, keeping down communication time. The populations are connected together by a migration operation, which is performed by communication between processors. While we experimented with various values of N_{isl} and λ , we always took $\mu = \lambda/5$.

The fitness function $\Phi(\vec{a}) : I \rightarrow \mathbb{R}$ is equal to the objective function $E(\vec{x})$ from equation 3, as well as the penalty function $\Pi(\vec{a}) : I \rightarrow \mathbb{R}$ from equation 7. Here the penalty function is not added to the objective function, as it is handled separately during the selection operation.

The selection operator $s_{P_f} : I^\lambda \rightarrow I^\mu$ selects the top μ individuals of the parent population, according to a Stochastic Ranking based on both the fitness function and the penalty function. This Stochastic Ranking method, produced by Runarsson and Yao ⁵⁹, uses a bubble-sort-like procedure, in which an arbitrary ranking is produced, and λ sweeps are performed in which each

individual in turn, starting from the top, is compared to the one directly below. If the result of the penalty function for both individuals is less than or equal to zero, the fitnesses of the two are compared, and the pair is ordered accordingly. If the penalty function of the top individual is greater than zero, then there is a probability P_f that the individuals will be ordered according to their fitness, and a probability $1 - P_f$ that they will be ordered according to their penalty value. This procedure is ended if there is no change in the order in any given sweep. We use $P_f = 0.45$, which is in the suggested range given by Runarsson and Yao⁵⁹, and is the value used by Fomekong-Nanfack et al.¹⁴.

The local recombination operator $r'_\gamma : I^\mu \rightarrow I$ occurs in two parts. First, $\lambda - \mu$ individuals are produced as direct (asexual) copies of the individuals in the parent population, with the fittest $\lambda \pmod{\mu}$ individuals being represented twice if λ is not a multiple of μ . Second, μ additional individuals are produced by recombination; each individual \vec{a}_c in the parent population produces an offspring \vec{a}'_c with object vector \vec{x}'_c by recombination between its own object vector \vec{x}_c , the next fittest individuals object vector \vec{x}_{i+1} , and the fittest individuals object vector \vec{x}_0 , using the equation

$$\vec{x}'_i = \vec{x}_i + \gamma(\vec{x}_{i+1} - \vec{x}_0) \quad (9)$$

where γ is the recombination factor. We have taken $\gamma = 0.85$.

The local mutation operator $m_{\{\varphi, \alpha\}} : I \rightarrow I$ is a non-isotropic self-adaptive mutation rule, in the sense that each individual has a different step size for mutation. This allows the step size to undergo evolution under selection, which leads to an adaptive step-size similar to SA, but without having to specify a step-size adaptation rule. Mutation is applied to the $\lambda - \mu$ individuals which did not undergo recombination, and starts with a random change to the step size σ_{OJ} , given by

$$\sigma'_{ij} = \sigma_{ij} \exp(\tau' N_i(0, 1) + \tau N_{ij}(0, 1)) \quad (10)$$

for $i \in \{\mu + 1, \dots, \lambda\}$ and $j \in \{1, \dots, n\}$, where $\tau = \varphi^* / \sqrt{2\sqrt{n}}$ and $\tau' = \varphi^* / \sqrt{2n}$ are tuning parameters. We have used $\varphi^* = 1$. N_i and N_{ij} are a vector and a matrix of values sampled from a normal distribution with zero mean and unit variance, which is generated afresh each generation.

Next, we mutate the parameters \vec{x}_{ij} themselves, using

$$\vec{x}'_{ij} = \vec{x}_{ij} + \sigma'_{ij} N'_j(0, 1) \quad (11)$$

for $i \in \{\mu + 1, \dots, \lambda\}$ and $j \in \{1, \dots, n\}$, where $N'_j(0, 1)$ is another randomly sampled normal unit vector, generated each generation.

Finally, we apply exponential smoothing to the step sizes, to reduce fluctuation

$$\sigma''_{ij} = \sigma_{ij} + \alpha(\sigma'_{ij} - \sigma_{ij}) \quad (12)$$

for $i \in \{\mu + 1, \dots, \lambda\}$ and $j \in \{1, \dots, n\}$, where α is a smoothing parameter. We have taken $\alpha = 0.2$. σ''_{ij} then becomes the step size for the next round of mutation.

A migration operation $mg_m : I^{N_{isl} \times \mu} \rightarrow I^{N_{isl} \times \mu}$ is applied across all populations (and thus all nodes) every m generations (we took $m = 200$, though experimental runs showed little variation within the range 50 to 200, data not shown). A node designated the master node generates a migration schedule, in which every population is assigned another population to migrate an individual to, and this schedule is broadcast to all nodes. The individual nodes then communicate with each other point-to-point, with each individual sending the parameter values for its highest-ranking individual to its designated receiver, and replacing its lowest ranking individual with the best individuals of the population for which it is a designated receiver (or, if it was the designated receiver for multiple populations, with the individual from the last population to communicate with it).

Finally, the collection of data related to descent speed and the checking of termination criteria were performed together. Every τ generations, the best individual in each population was backed up, and the processors communicated between each other to find the lowest energy of any individual across all populations, which the master node records to a log file along with a time-stamp. We took $\tau = 20$. The program has two termination procedures; it either runs for a preset number of generations, or it halts when the lowest energy remained below a particular preset amount for $\rho \times \tau$ generations. Preliminary investigation showed that the convergence time was relatively constant across runs, so we use a constant number of generations. This number of generations varied, but was generally 40 000, which we found to be long enough for virtually all runs to converge.

3.1.3 Parallel Lam Simulated Annealing

We will use the Parallel Lam SA algorithm given by Chu et al. ⁹, with details taken from Jaeger ²⁵, which runs on K processors, with one processor being arbitrarily defined as the master node. I have used the optimisation parameters selected in Jaeger et al. ²⁶, which were based on extensive sampling of a simplified 2-gene problem by Chu et al. ⁹.

The energy is given by the objective function $E(\vec{x})$ plus the penalty function $\Pi(\vec{x})$, defined in equations equation 3 and 7 respectively.

The basic operation of the algorithm is straightforward: each iteration of the algorithm, each of the K processors change the state, the parameter set \vec{x} that they have stored, to a new state \vec{x}' , according to a move generation strategy (described later). They then evaluate the energy difference between the old and new states $\Delta E = E(\vec{x}') - E(\vec{x})$; if this is negative, the move is accepted, if not then it is accepted with a probability $exp(\Delta E/T)$. The temperature starts at T_0 , which we have taken to be $T_0 = 10^6$, and is decreased according to the Lam schedule

$$s_{n+1} = s_n + \lambda \left(\frac{1}{\sigma(s_n)} \right) \left(\frac{1}{s_n^2 \sigma^2(s_n)} \right) \left(\frac{4\rho_o(s_n)(1 - \rho_o(s_n))^2}{(2 - \rho_o(s_n))^2} \right) \quad (13)$$

where $s_n = 1/T(n)$ is the inverse temperature at the n th iteration, $\sigma(s_n)$ is the standard deviation of the energy at the n th iteration, ρ_0 is the acceptance ratio (the ratio of the number of moves accepted to the total number of moves proposed) and λ is a quality factor, determining how close the system stays to equilibrium. We took $\lambda = .00001$. It should be noted that the final term contains only the acceptance ratio ρ_0 , and is maximised when $\rho_0 = 0.44$.

Lam and Delosme ³³ developed a set of estimators for calculating the mean energy, $\sigma(s_n)$ and ρ_0 . ρ_0 is relatively simple; merely averaging over the last 100 iterations gives a good estimate. However, this does not work for the mean and standard deviation, as the energy changes rapidly, and energy states are autocorrelated, meaning that means based on short-term averaging would be subject to unacceptable noise, and means based on long-term averaging would involve sampling the energy at a now outdated temperature. To solve this problem a more complex statistical estimation procedure was derived. The estimation technique works by modeling the energy density at a given temperature as a sum of gamma functions, and fitting the model by weighted least-squares to the previously observed energies. The weights exponentially decay with time in the past, and are given for each observation by w_a^n for the mean and w_b^n for the standard deviation, where n is the number of iterations ago the observation was taken. Larger values of the w parameters mean that past observations are given close to as much weight as recent ones, whereas low values mean that past observations are given little weight. We used $w_a = 0.995$ and $w_b = 0.99999$, taken from Chu et al. ⁹.

The Parallel Lam schedule is global, in that each node uses the same temperature for any given iteration. This means that we need to keep average statistics across all nodes. To this end, every τ/K generations all processors send their statistics to a master node, which averages them together and keeps further running estimates of these local statistics using the method described above. It

is this set of global statistics which is used to calculate the temperature changes according to the Lam schedule; the rate of temperature decrease is based on all iterations occurring in parallel, and thus is made to decrease K times faster than the serial case. We took $\tau = 100$.

Another important aspect of the algorithm is move generation. Each parameter is changed every iteration according to the exponential change rule

$$x'_i = x_i \pm \bar{\theta}^i \ln(\xi) \quad (14)$$

where ξ is drawn from a uniform random distribution between 0 and 1, and $\bar{\theta}^i$ is a scaling parameter designed to keep ρ_0 near to 0.44 and is updated every v iterations according to

$$\ln(\bar{\theta}^{i'}) = \ln(\bar{\theta}^i) + 3.0(\rho_0 - 0.44) \quad (15)$$

We took $v = 100$.

Every $m\tau/K$ iterations, the algorithm performs a mixing of states. Every node sends its current energy to the master node, and the master node randomly assigns each node another node from which to receive the current parameter values. The probability of any given node being assigned node i is given by $\exp(E_i/T) / \sum_{j=1}^K \exp(E_j/T)$, where E_i is the current energy of node i . This mixing of states allows the best results to propagate, but also allows nodes to explore higher energy solutions. We use a mixing rate $m = 13$ for 10 nodes.

In order to avoid the final solution being affected by the initial conditions the algorithm performs an “initial burn”, in which each processor spends n_{init} iterations running as normal serial SA at a constant temperature T_0 . After that, the algorithm runs for n_{init}/K iterations to calculate initial statistics. n_{init} is chosen to allow auto-correlation to drop to near zero; we have chosen $n_{init} = 10^5$.

The algorithm has two different types of stopping conditions; the absolute condition, and the freeze condition. In the absolute condition, the algorithm terminates after the absolute mean value of E remains below a target energy for $5 \times \tau/K$ iterations. In the freeze condition, the algorithm terminates after the absolute energy E changes by less than a proportion κ over $5 \times \tau/K$ iterations. As preliminary investigations showed that the final energy and the convergence time were both very variable, we use the latter case, and take $\kappa = 0.0001$.

3.2 Implementation

3.2.1 Software

The Parallel (μ, λ) -ES code is implemented in C++, and is based on code written and used by Fomekong-Nanfack et al.¹⁴ (available at http://www.science.uva.nl/research/scs/3D-RegNet/fly_ea/). The code was modified to run in parallel; main communication areas were migration, in which an arbitrary master-node generates and broadcasts a migration schedule, and then nodes communicate point-to-point to send parameters; backup and logging, in which individuals keep backups of their best individuals, and the master-node keeps a time-stamped log of the best individuals every τ generations; and termination, with all nodes broadcasting their energy every τ generations and terminating when the criteria described above are met.

The Parallel Lam SA code is implemented in C, with code taken from the supplementary material from Jaeger et al.²⁷, (available at <http://flyex.ams.sunysb.edu/lab/gaps.html>). Little modification is made to the original code, other than minor alterations to allow finer-scale time-stamping in the log files for descent curves.

For both algorithms we had a scrambling procedure to give the problem different starting conditions; the SA algorithm reads in starting parameter values, which a scramble program randomised prior to starting each instance of the program, and the ES algorithm generates the starting conditions according to a random seed, which was also scrambled prior to starting each run.

Both implementations were compiled using the Intel C++ Compiler ICC, and both implementations make use of the QLogic implementation of the Message Passing Interface (MPI, described below). Analysis of data was performed using the statistical programming language R⁵³.

Source code is available from <http://www.srcf.ucam.org/~lj237/thesis.html>.

3.2.2 The Message Passing Interface

There are a number of models of parallel computing that can be used when creating a parallel program¹⁹. One such model is the shared-memory model, in which all processors are assumed to have full access to a common memory, and each runs a separate process. The programs all perform load and store operations directly on the memory, with their combined actions making up the parallel program. A related model is the threads model, in which all processors run the same process, with a single memory address and a single set of state variables, but the process subdivides into a set of ‘threads’; this has the advantage that creating and destroying threads, and switching between threads, occurs quickly (as you do not need to begin new processes to do so). The parallel ES described by Ngom⁴⁸ operates under this model, using the POSIX Threads standard.

However, these models all have the disadvantage of being relatively inflexible; they require a set of processors to be physically connected to the memory that they are using, which generally requires a pre-designed architecture that cannot be easily modified. The message-passing model is more flexible: each processor is assumed to have its own local memory, which no other processors have automatic access to. Transfer of information occurs by explicit sending of information between processors over some form of communication network. This has the advantage of being highly portable, as it makes no underlying assumptions about the network, other than that it allows messages to be passed. It can thus function on heterogeneous networks of nodes of arbitrary and variable size¹⁹.

One implementation of the message-passing model is the Message Passing Interface (MPI), a communications protocol that is implemented via a library containing functions that allow information to be exchanged between nodes. This can be done in one of two ways; blocking, or synchronous communication (in which all nodes involved in a particular message exchange do so simultaneously) and non-blocking, or asynchronous communication (in which a node sends messages to a buffer associated with a second node, and the second node receives it when it at a later date). The former has the disadvantage that a node that wishes to send information must wait until the other nodes involved are ready to do so; however, the latter makes the program less predictable, as the order in which nodes exchange information differs each time the program is run. I have used blocking communication in both algorithm implementations, as using non-blocking communication would involve modifications to the algorithms beyond the scope of this study.

The MPI functions that are used in the code implement one-to-one communication, one-to-many communication, many-to-one communication and many-to-many communication. One-to-one communication is used in migration (in the ES) and mixing of states (in SA), where single nodes either exchange states or send and receive individuals from each other; the associated functions are `MPI_Recv`, which receives data from a specified node, and `MPI_Send`, which sends data to a specific node. One-to-many communication comes in the form of the `MPI_Broadcast` function, which is used in both algorithms for the master node to send out migration or mixing schedules to all other nodes. Many-to-one communication is used to condense information from other nodes, which is used in both algorithms for when the master node updates the ensemble- or population-wide statistics. The function `MPI_Reduce` is used to efficiently combine information from many processors; it has a function associated with it (usually either `min` or `sum`), which maps all the information sent by the nodes to a single value, received by the master node. Many-to-many communication is used to keep all nodes informed of the energy of all others, and uses

the `MPI_Allreduce` function, which operates the same as `MPI_Reduce`, except that it sends the information to all nodes, rather than just the master node.

Another important use was in timing the programs. Both the programs write out regular log files, which contain information on the value of the objective function, along with a time-stamp (kept using the `MPI_time` command).

3.2.3 The Darwin Cluster

The programs were run on the Darwin Supercomputer of the University of Cambridge High Performance Computing Service (<http://www.hpc.cam.ac.uk/>), provided by Dell Inc. using Strategic Research Infrastructure Funding from the Higher Education Funding Council for England.

Darwin is a 2340 core computing cluster, made up of 595 boxes with four 3.00 GHz processors each (two 3.00 GHz dual core Intel Woodcrest processors). Each box has 8GB of RAM (2GB per processor) and 80GB of hard-disk space, and are all connected together by an Infiniband interconnect, with 900 MB/s bandwidth with a 2 microsecond MPI latency.

Programs, or jobs, are written as specialised shell scripts that contain information on how much time and memory and how many processors are needed. This information is passed on to a Portable Batch System (PBS), which monitors available nodes and schedules the jobs to run when the resources requested become available. Up to 512 nodes may be requested for job lengths of up to 36 hours at a time. All the tests described below were run via the Darwin PBS system.

4 Results

4.1 General Procedure

In this section I will describe the results of various tests performed on both algorithms. These include tests of the performance of the serial island-ES and the parallel ES under different circumstances, and a comparison of the performance of the parallel ES and Parallel Lam SA.

There are two aspects of algorithm performance that I wish to measure. The first is how fast it is: how long the algorithm takes to reach a given quality of solution. The second is how reliable it is: what proportion of the time it can produce a solution of a given quality. I measured how the ES performed under various circumstances using these metrics, and also used these metrics to compare the ES and the SA algorithms.

I used an R script to extract the descent curves from optimisation log files, which give a detailed description of how the value of the objective function changes over time; each descent curve was created by averaging over a number of runs (which differs test-to-test). In order to gain more definite, quantitative measures of the algorithm performance, I also recorded the proportion of runs that reached a given energy criterion; we used energy criteria ranging from good solutions to poor solutions (350 000, 450 000, 550 000 or 650 000). I also calculated the average amount of time taken for the runs to reach the various criteria, though obviously only for those runs that did achieve the criteria.

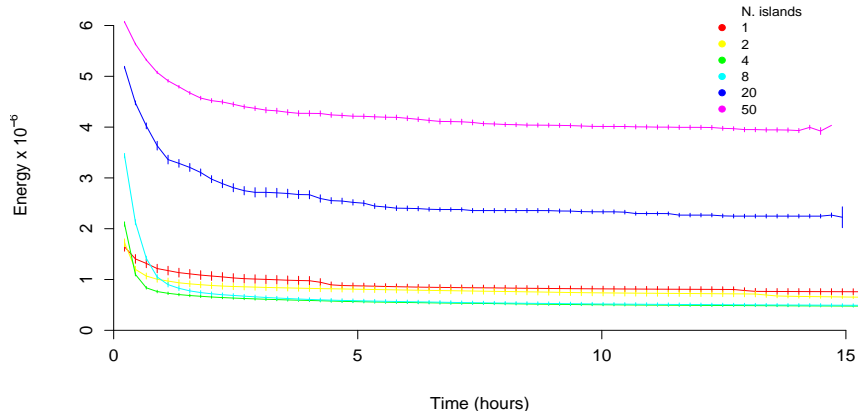
4.2 Island Dynamics

Before I directly study the parallel ES algorithm, I am going to study how the number of islands affects the performance of the serial island-ES algorithm, which has not been addressed previously. There are two justifications for this. First, the behaviour of the serial algorithm is useful information, and how to choose the number of islands to maximise descent speed and reliability is an important tuning question. Second, studying the effect of adding additional islands gives an indication of how efficient the parallel algorithm could be, and by comparing the results to the parallel case we can judge how much algorithm speed is lost due to communication overhead and waiting times.

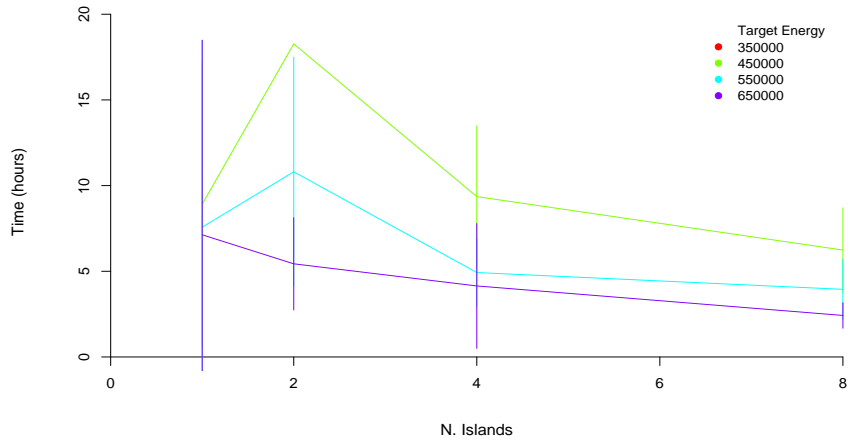
There are two different approaches to adding islands to the island-ES. The first approach is to keep a constant metapopulation size, meaning that adding new islands decreases the number of individuals per population. This means that the processing time required for the algorithm does not increase with the number of populations, and in fact decreases due to the $O(n^2)$ complexity of the stochastic sorting. The trade off in this approach is between number of populations and population size; with a large number of populations, you will allow many populations to search different parts of the state-space, but the search abilities of the individual populations shrink as their sizes dwindle. The second approach is to keep a constant population size, meaning that the meta-population size grows as the number of islands increases; this trades off the increased number of populations with decreased running speed per generation.

I tested both of these approaches. In both cases, I ran the algorithm 10 times each for different numbers of islands $N_{isl} \in \{1, 2, 4, 8, 20, 50\}$. In the case of the constant metapopulation, I used a metapopulation size of 500, giving $\lambda = 500/N_{isl}$ individuals per island; every run consists of 25 000 generations. In the case of the constant population size, we used $\lambda = 125$, for a metapopulation size of $N_{isl}\lambda$. In this case the number of generations per run varied; initially I used $100000/N_{isl}$, but I also ran the 20 and 50 islands for the full 36 hours the PBS system allowed, in order to compare their per-island descent speed to other runs. Figures 1 and 2 show results for the serial island-ES with constant metapopulation size and constant population size respectively.

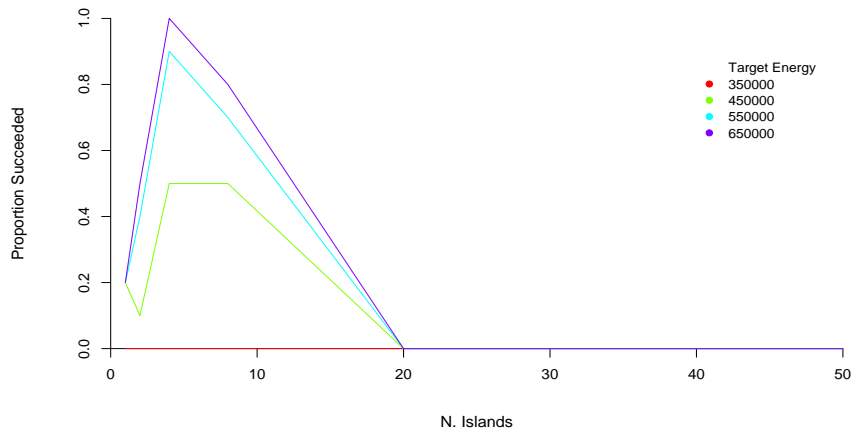
The constant metapopulation size approach (Figure 1a) runs for approximately the same



(a) Descent Curves for Various Numbers of Islands



(b) Time Taken to Reach Various Energies



(c) Success Rates for Various Energies

Figure 1: Descent curve, times taken to reach target energies and success rates for the serial island-ES with various numbers of islands and with constant metapopulation size $\lambda N_{isl} = 500$ and variable population size $\lambda = 500/N_{isl}$, with 10 runs per data-point. In 1a and 1b the error bars represent 1.96 standard errors around the mean (σ/\sqrt{N} for N data-points with standard deviation σ), and correspond to 95% confidence intervals for normal data. Data for islands 20 and 50 are not plotted in 1b, since none of the solutions converged on any of the energy targets.

amount of time regardless of N_{isl} , with a slight reduction in running time for 20 and 50 islands, as predicted above. The most striking aspect of the graph is that with high numbers of islands, and thus low population sizes, the ES catastrophically fails to converge on a good solution. As the number of islands increases from 1 to 8, the initial speed of descent increases, but the initial energy also increases; the 4-island solution is initially better, though the 8 island solution converges on it with time.

In terms of the time taken to reach target energies, Figure 1b shows a general decrease in the time taken to reach the criteria given, though the difference between nodes 4 and 8 is slight, and the small number of 1-island runs that reached any of the energy criteria means that the error is large. Figure 1c shows a definite peak in terms of success rate at 4 islands, with success being very low for very high and very low numbers of islands. None of the runs achieved the strictest energy target applied.

The constant population size approach (Figure 2a) shows the expected increase in running time as the number of populations increases. The descent curves for 1, 2 and 4 populations are similar, with the 4-island case seeming slightly more effective, but the large computation time rapidly swamps any advantages when population sizes become larger. Figure 2b confirms this, with populations 1, 2 and 4 taking approximately the same amount of time for most energy targets, but the time taken increasing steadily beyond that. Figure 2c shows a decrease in the reliability across the board as the number of islands increases, until around 8 populations, after which it increases again.

The constant population-size approach is comparable to the parallel approach, and allows direct simulation of what a perfect parallel algorithm would look like. I divided the time taken in the serial case by the number of islands used, in order to give an estimate of perfect parallel performance, assuming that the processing power is perfectly spread across N_{isl} processors, with no communication overhead. The results are shown in Figure 3. We can see that the increase in computation time from increasing islands grows more slowly than the increase in speed from adding additional processors, at least up until 20 nodes. Figure 3a shows that the speed of descent increases for up to 20 nodes, though the 50 node case does not seem to be faster, with Figure 3b confirming this.

4.3 Parallel Efficiency of the Evolutionary Strategy

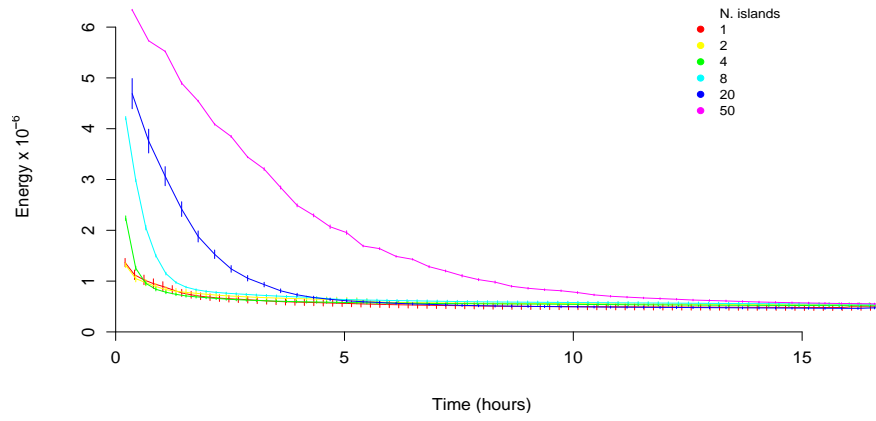
In order to test the actual efficiency of the parallel ES, I performed a similar set of tests as serial island-ES tests with constant population size. I ran the parallel ES 10 times sequentially for different numbers of processors $N_{isl} \in \{1, 2, 4, 8, 20, 50\}$, with a population size $\lambda = 125$. Each run terminated after 40 000 generations. The results are shown in Figure 4.

The descent curve in Figure 4a shows a small but persistent speed increase with increasing number of processors, though this is unsurprisingly not as pronounced as it is in Figure 3a. Increasing the number of nodes also decreases the final mean convergence energy. However, the difference between 20 and 50 nodes in both these respects is so small as to be almost unnoticeable.

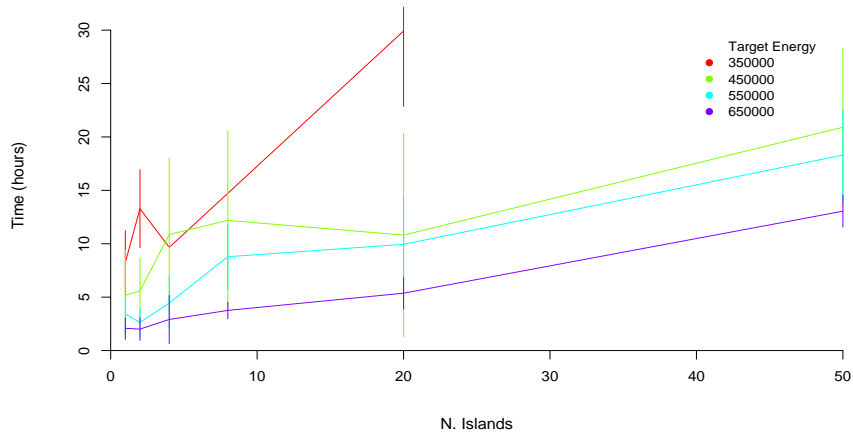
Figure 4b shows very noisy data below 4 nodes, but after that shows a gradual decrease in the time taken. However, this decrease is nowhere near as dramatic as Figure 3b. If we define the parallelisation efficiency as the mean value of

$$\frac{\text{Speed for } N_A \text{ nodes}}{\text{Speed for } N_B \text{ nodes}} \times \frac{N_B}{N_A} \quad (16)$$

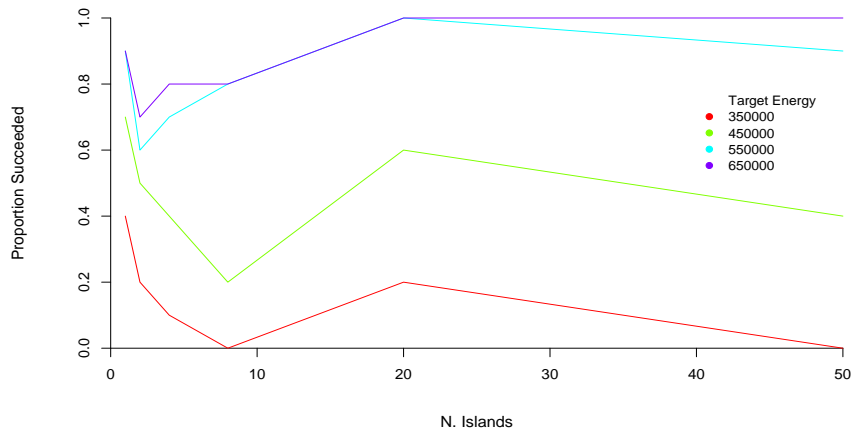
for all adjacent pairs of numbers of nodes (i.e. (1,2),(2,4),(4,8),(8,20) and (20,50)), we find at best a parallelization efficiency of 59% (for the 550 000 condition) and at worst a parallelization efficiency of 44% (for the 350 000 condition). However, if you exclude the 50 node case the largest



(a) Descent Curves

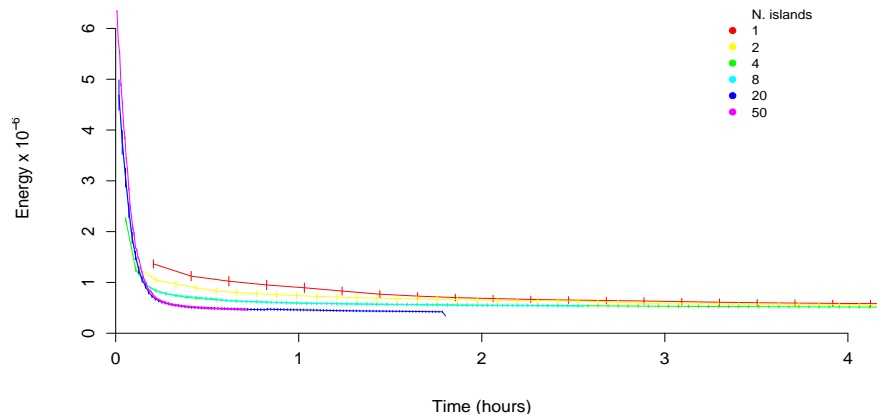


(b) Time Taken to Reach Target Energy

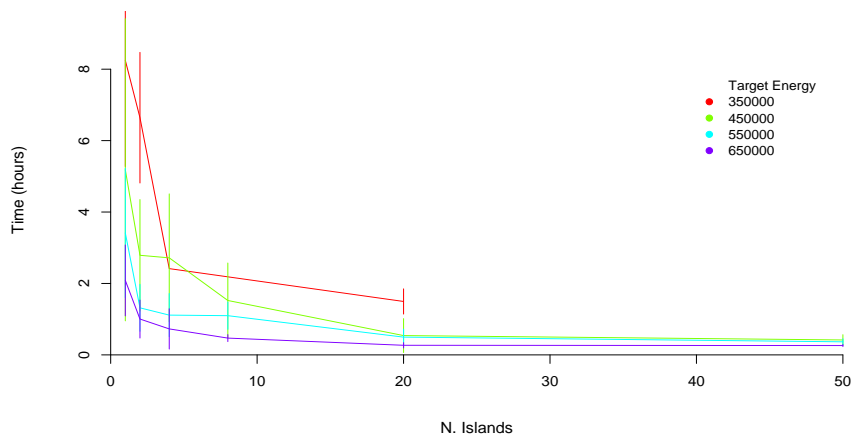


(c) Success Rates for Various Energies

Figure 2: Results for the serial island-ES for various numbers of islands and with variable metapopulation size $125N_{isl}$ and constant population size $\lambda = 125$, with 10 runs per data-point. Errors calculated as in Figure 1.



(a) Descent Curves



(b) Time Taken to Reach Target Energy

Figure 3: Predicted descent curve and time taken to reach target results for a perfectly efficient parallel ES with various numbers of nodes, calculated by dividing the times for the constant population size serial ES in Figures 2a and 2b by N_{isl} .

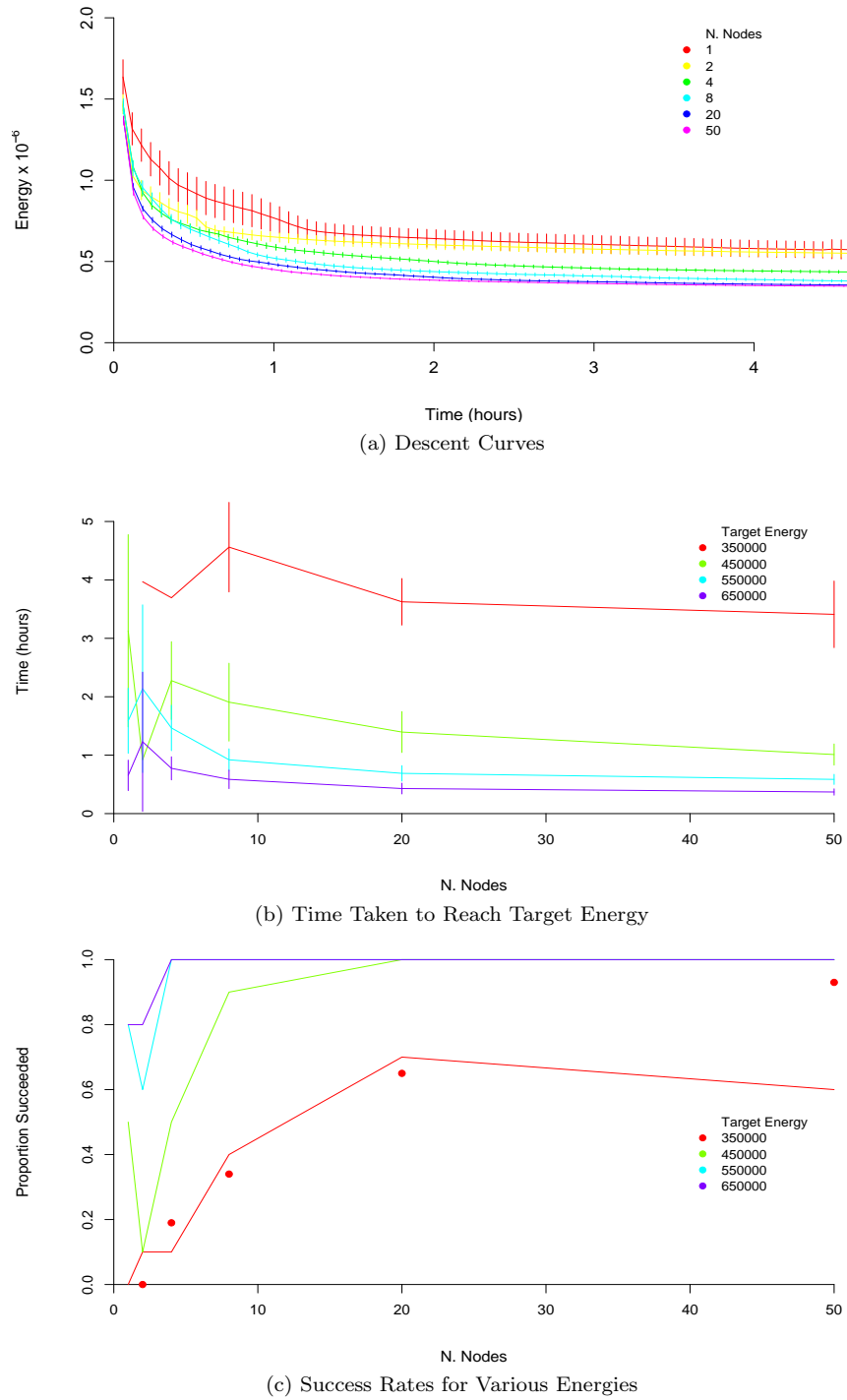


Figure 4: Descent curves, times taken to reach target energies and success rates for the parallel ES on various numbers of nodes, with population sizes $\lambda = 125$, with 10 runs per data-point. The red dots represent the best possible efficiency that could be gained by running multiple instances of smaller-node runs, according to equation 17.

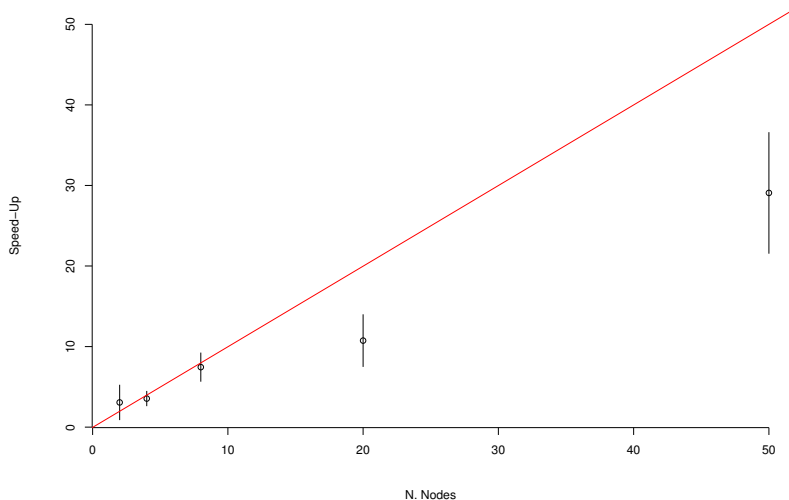


Figure 5: The Speed-Up of the parallel algorithm using N_{isl} nodes, compared to the serial algorithm using N_{isl} islands. The red line shows a theoretical perfect speed-up curve.

efficiency rises to 64%. Figure 4c shows a sharp increase in success rate for all energies up to 20 nodes, after which it ceases to increase.

We designate the reliability of an algorithm on K_i nodes for energy E as $r_{K_a}(E)$ (the probability of achieving target energy E). In order to quantify whether the reliability of the algorithm for a given number of nodes is an improvement over those that use fewer nodes, we observe that an algorithm running on K_a could be replaced by an ensemble of $\text{floor}(K_a/K_b)$ algorithms running on K_b nodes each. The reliability of this ensemble would be given by $1 - (1 - r_{K_b}(E))^{\text{floor}(K_a/K_b)}$. We can thus give the minimum reliability that an algorithm using a particular number of nodes would have to achieve in order to be an improvement over the best possible ensemble, or

$$r_a^*(E) = \max_b \left(1 - (1 - r_{K_b}(E))^{\text{floor}(K_a/K_b)} \right) \quad (17)$$

I have plotted this minimum reliability for the lowest energy requirement ($E = 350000$) as the red dots in Figure 4c; you can see that the reliability scales well with the number of node, with the exception of 50 nodes.

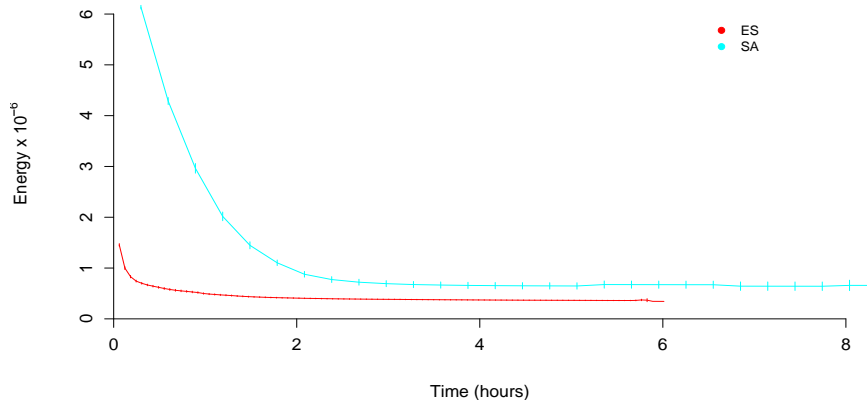
We can combine the data from the serial runs with the data from the parallel runs to measure the relative speed-up of the parallel algorithm, defined as

$$\frac{\text{Time taken in serial}}{\text{Time taken in parallel}} \quad (18)$$

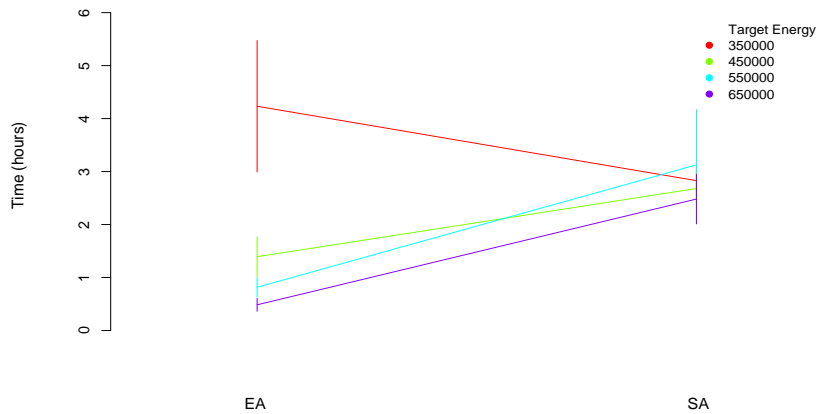
measured for a particular value of N_{isl} . Figure 5 shows the speed-up for various numbers of nodes; the red line shows a theoretical perfect speed-up. Below 10 nodes the results are essentially perfect; the speed-up is indistinguishable from the perfect speed-up. However, after this point the line drops off, with the speed-up dropping to about 60%.

4.4 Parallel Lam Simulated Annealing versus the Parallel Evolutionary Strategy

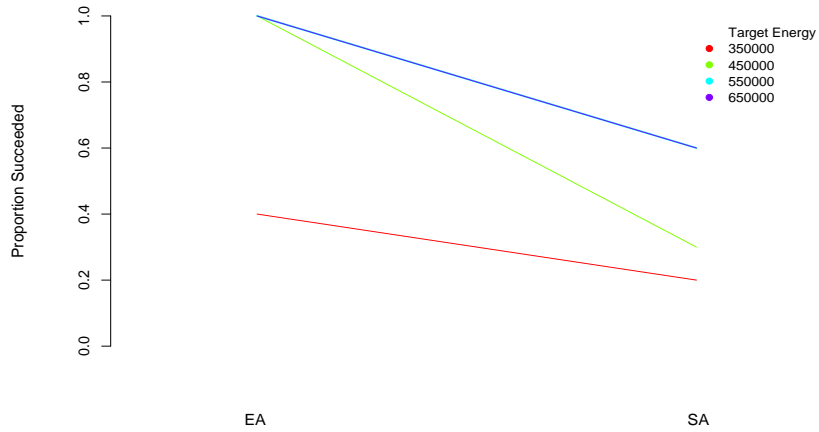
To determine which algorithm performs better at the model-fitting task, we applied both the Parallel Lam SA algorithm and the Parallel EA algorithm to the gap gene network problem. In



(a) Descent Curves



(b) Time Taken to Reach Target Energy



(c) Success Rates for Various Energies

Figure 6: A comparison of descent curves, time taken to reach target energies and success rates between the parallel ES and the Parallel Lam SA, using 10-nodes, with 50 runs per data-point.

both cases we used 10 processors per run, and for the ES we used a population size of $\lambda = 125$. We did a total of 50 runs for each algorithm, in order to have a large enough sample size to detect definite differences in performance.

The results of the comparison tests are shown in Figure 6. The descent curves show that the ES begins at a lower energy and descends at a much faster rate. The running time of the Simulated Annealing algorithm was highly variable, ranging from 1 to 30 hours, with a median of 12 hours. The final energy was also highly variable, from as low as 240 000 to as high as 2 850 000, with a mean of 750 000. However, I did not see the bimodal behaviour described by Chu et al. ⁹; there appeared to be monomodal distribution of final energies.

Once a single outlier program (which ran for 62 seconds) was excluded, the final EA energy only went down to 330 000, but never went above 470 000 either, with a mean final energy of 360 000. The running time was also highly consistent, never deviating more than 45 minutes from the 5.5 hour run-time.

Figure 6b shows that for all energy targets but the lowest, the ES finished in half the time as SA. However, for the lowest energy, SA was significantly faster. Figure 6c shows that the ES performed significantly more reliably, achieving three of the energy targets every time, and the most stringent 40% of the time, while SA only reached even the highest energy target 60% of the time. Note that the last two pieces of information show us that the SA algorithm rarely achieves high quality solutions, but when it does do so, it does it relatively fast; i.e. there are a limited number of ‘good runs’, that achieve low energies at high speeds.

5 Discussion

5.1 The Parallel (μ, λ) Evolutionary Strategy

The behaviour of the serial island-ES with constant population size partly confirms Fomekong-Nanfack et al.¹⁴'s choice of 4 islands for their optimisation problem, as 4 islands does give the maximum reliability (as shown in Figure 1c). In terms of maximum algorithm speed, however, the optimal number of islands is 8, thus giving a trade-off between speed and reliability. Above 8 islands there is no measured speed-up, since none of the runs converged. The fact that the algorithm decreases in efficiency so dramatically for 20 and 50 islands compared to 1-8 islands suggests that there is a minimum population size required for the algorithm to converge at all, and below this value it does not matter how many islands you have, since no populations will succeed. We can estimate this minimum population size as being between the population size in the 8 island case and the 20 island case; i.e. somewhere between 62 and 25. What the limiting factor in small populations is cannot be directly addressed, but it seems likely that it is the lack of diversity within the populations which prevents any population searching the local state-space very effectively.

The results for the serial algorithm with variable population size are relatively straight-forward. There is a trade-off between the increase in search capacity due to increasing the number of islands and the decreasing speed of each island having to be processed. As Figure 2 shows, for up to 4 nodes this trade-off is well-balanced; there is no detectable change in efficiency. Beyond 4 nodes, the increase in processing time outweighs the advantage of the increased search space. This is especially true for the low energy cases, which suggests that the increased number of islands mostly increases the initial speed of descent, i.e. the speed at which high- and mid-energy targets are reached, and has less effect on the the final convergence.

Our estimates of the reliability of the variable population-size algorithm are somewhat contradictory, as Figure 2c shows a *decrease* in reliability with increased number of islands, despite the parallel algorithm showing an *increase* (see Figure 4c). Since the algorithm should perform identically in all respects other than speed, I expect that there is a confounding technical factor. As the run-time grows linearly with the number of populations in the serial runs, the increased unreliability may be due to crashes or interruptions from Darwin's PBS system; this problem would not be present in the parallel case, as running time does not increase significantly with number of populations.

Looking at the ideal parallel results predicted from the variable population size results, we can see that the per-island speed of descent rises significantly with an increase in islands. Figure 3a shows that every increase in the number of islands up to 20 brings a significant increase in the per-island speed of descent, and Figure 3b shows a similar result. However, the 20 and 50 nodes appear to have roughly the same per-island speed, which suggests that at 50 nodes the increase in efficiency that results from adding more nodes is too small to be observed with the number of samples we took.

These results act as a baseline for the parallel ES, on top of which inter-node communication is added. The difference between the predicted parallel results (Figure 3) and the actual parallel results (Figure 4) is summed up as the difference between the perfect speed-up curve and the actual speed-up curve, shown in Figure 5. For up to 8 nodes the speed-up is 100% to within the accuracy we can measure; the communication overhead is small compared to other processes occurring. However, beyond that the communication overhead causes a drop in efficiency. A large proportion of this will be a simple increase in communication; if there are a more processors, more information must be collected and broadcasted. More communication overhead will be introduced by the asynchronous nature of the algorithm; each τ generations takes as long for the entire group of nodes as it does for the slowest node, as all populations have to wait to exchange information

with it. As the number of nodes increases, the total variation in speeds increases, and thus the time spent waiting for slow individuals increases.

The results for the parallel algorithm (Figure 3) show that the algorithm scales well with the number of nodes. In terms of speed, we have a parallel efficiency of up to 64% for 20 nodes or fewer, although there appears to be little increase in speed for 50 nodes over 20, as would be predicted from the variable population size ES. It is in increased reliability that the parallel ES especially excels. Figure 4c shows that the probability of obtaining target energies rises very fast, with high- and mid-energy solutions being achieved on every run by 20 nodes, and low energy solutions being achieved 60% of the time, and the reliability scales well compared to the minimum reliability values. Once again, however, there are no real advantages to using 50 nodes over using 20.

The general conclusion from this analysis seems to be that the algorithm scales well, when both speed-up and increase in reliability are considered, for up to 20 nodes. However, beyond this, and especially for very low energy targets, the parallelization becomes less effective.

5.2 Simulated Annealing vs Evolutionary Strategy Efficiency

The most striking feature of the descent curves of Parallel Lam SA and the parallel ES algorithm (Figure 6a) is how much faster the ES algorithm converges. Examining the data closely, we can see that there are three aspects that characterise the difference in the curves.

Firstly, the ES curve begins at a much lower energy than the SA curve. This is caused by differences in the initialisation behaviour of the two algorithms. The SA algorithm begins with a single starting solution, which undergoes a high temperature ‘burn period’ to cause the processors to become decorrelated and to lose dependence on the initial condition, followed by a period of statistics collection at high temperature in order to prime the Lam schedule. This places a hard limit of K initial states on the algorithm, giving a smaller sample of states and thus a lower expected minimum energy across processors. In contrast, the parallel ES starts with a number of randomly created initial states equal to λN_{isl} , allowing a far higher diversity of energies, and thus a lower expected minimum energy.

Secondly, the initial speed of descent is higher for the ES. This is probably due to a particular difference in the early operation of the two algorithms. During the early stages, the SA temperature is high, and thus there is little selection for better solutions; once the temperature is lowered the selection for better solutions increases, but simultaneously the solution is getting closer to a minimum, and the slowing associated with the decreased move size counteracts the decreased temperature. The ES, however, begins a full selection schedule straight away, allowing descent at maximum speed from the very start of the algorithm. Note that the reason that the ES can afford to start fast, but the SA cannot, is that the multi-individual nature of Evolutionary Algorithms allows a diversity of individuals (and thus lower energy solutions) to remain despite a decrease in mean energy; if the SA was to decrease at this rate, it would lose quasi-equilibrium and fail to converge, becoming stuck in a local minimum.

Thirdly, the parallel ES converges to a lower mean energy across all runs than the parallel SA. The reason for this appears to be driven by the unreliable nature of the SA, shown in Figure 6c; while the SA can achieve low energy, and has a minimum energy across all runs far lower than the ES, the large proportion of runs that fail to reach low energy increases the mean energy to above that of the ES.

The unreliable nature of the SA has been commented on before^{26 14}, and that island-EAs can be highly reliable function optimisers in large parameter estimation problems has been well established⁴⁵. It is difficult to say precisely why an island-EA should be more reliable than a parallel SA; both involve within-processor means of landscape searching (mutation/recombination in ES, move generation in SA), and both of them involve a process by which solutions propagate

through from processor to processor. It is possible that the diversity within each population in the EA prevents the propagation of local minima solutions throughout the population; even if a locally minimal solution spreads to all populations, there will still be higher-energy individuals that will continue to search the state-space outside of this minimum, and once they achieve a lower solution than the local minimum they will begin to propagate. This does not occur in Simulated Annealing; if every processor is stuck in a local minimum, then there are no back-up individuals to allow it to escape.

Addressing this question, either by running numerical tests on altered algorithms (e.g. is the ES still more effective if the population diversity is periodically eradicated?) or by the difficult task of studying both systems analytically, may allow us to discover more aspects of the algorithms that contribute to these differences. This, in turn, may allow us to make improvements to the algorithms.

5.3 Tuning

One subject of note that is worth addressing is the importance of tuning. There are a large number of optimisation parameters associated with both optimisation procedures that need to take on the correct values for the optimisation to run at peak efficiency. For the ES the important ones are the offspring and parent population sizes λ and μ , the mutation and recombination parameters φ , α and γ and the migration interval m . For SA we have the pooling, mixing and step-size adaptation intervals τ , m and v , as well as Lam estimation parameters w_a and w_b . Finally we have the common penalty parameter Λ , and all the parameter bounds.

Significant effort has gone into parameter tuning, and the parameter values that I used in the above tests have been chosen for good reasons, both empirical and theoretical. Many values are taken from Chu et al.⁹, who made an extensive study of the parameter values used in Simulated Annealing by running the algorithm a large number of times on a simple 2-gene problem using a variety of values of the tuning parameters, and more were taken from the theoretical considerations of Lam and Delosme³³. In the case of the ES, many of the values were based on those used by Fomekong-Nanfack et al.¹⁴, including many that were based on an extensive empirical study of the selection regime by Runarsson and Yao⁵⁹.

Our results have contained a small amount of tuning investigation; I have shown that for the island-ES the ideal value of N_{isl} is 4 if we wish to maximise reliability, and 8 if we wish to maximise speed. The differences between the tuned and not tuned algorithms was significant; a change of a single island, from 4 to 3 (with speed extrapolated from the 2 and 4 island cases) would cause an estimated increase in the time taken of 35%. While this parameter is probably one of the most important in the algorithm, with the largest effect on efficiency, it nonetheless demonstrates the huge impact tuning can have on efficiency. It also demonstrates how hard tuning can be for real problems; this value alone took around 750 hours of CPU time to discover.

Such large computational load makes extensive tuning of the algorithms difficult even on the scale of a direct study of the algorithms, and certainly beyond the range of a systems biologist who only wishes to use the algorithm to solve a practical problem. As such, this study has operated as a ‘practical guide’, an example of the algorithms being applied given our current best knowledge of the optimal optimisation parameters.

It is unknown whether the optimal values of the optimisation parameters will be the same for the small test problems as they are for the large parameter estimation problems on noisy data, and the large number of parameter value combinations makes it unlikely that we have achieved peak performance for either algorithm. We can feel certain that picking the optimal parameters is itself an optimisation problem of at least comparable difficulty to the one that we have solved in this study, and approaches that tune each parameter individually are unlikely to find optimal values. The example of finding the ideal N_{isl} demonstrates both that finding new ways to choose

optimisation parameters is required, and that doing so potentially result in as much of an increase in algorithm speed as producing parallel algorithms or developing new optimisation techniques.

An interesting observation on this situation was given by Bäck⁵. It was noted that the problem of choosing tuning parameters for a Evolutionary Algorithm is exactly the kind of parameter estimation problem that Evolutionary Algorithms are able to solve; specifically, the mixed-data nature of the problem (with continuous and discrete parameters) is well suited to a Genetic Algorithm. To demonstrate this, he produced what he called a meta-evolutionary algorithm; a GA that optimises the tuning parameters of a different GA. This meta-GA was applied to problems with known ideal parameters, and was shown to be able to recover them effectively. Of course this approach requires a very large number of iterations of the EA that it is optimising, and thus it would be more suited to simulated versions of the test problem, as used by Chu et al.⁹.

5.4 Future of the Algorithms

The complexity of models in systems biology is constantly increasing, and thus the speed required of optimisation is always growing. The *Drosophila* segmentation system alone consists of interactions between dozens of genes and gene products³, and to model all of them would create a drastic increase in model complexity. Future developments in systems biology and in optimisation will have to address how this increasing complexity will be handled.

The simplest way of getting around this problem is identifying gene regulatory subprocesses, or modules; parts of the system for which most of the complexity is contained within interactions between the components. These subprocesses can be modelled individually, and these models incorporates as 'black boxes' in the modeling of the larger systems they are part of. For instance, experiments with knockout mutants have shown that the gap gene network is not regulated by the lower layer of the pair-rule genes³, and thus can be considered a subprocess and thus the model can be parameterized separately from latter stages of segmentation. Once this has been achieved, the parameters obtained can be used in larger models of the segmentation systems, drastically decreasing the complexity of parameterizing this model. This is sometimes called the 'Divide and Conquer' strategy.

However, even taking this into account, that models will increase in complexity beyond our current optimisation capacity is unavoidable. The Parallel Lam SA algorithm begins to lose efficiency beyond 50 nodes⁹, and this decrease in efficiency will become even worse beyond 100 nodes, as the value of τ will have to be increased beyond the current value, causing a loss in efficiency of the Lam schedule. In fact, the decrease in efficiency was certainly underestimated by Chu et al.⁹; as mentioned previously, they failed to take into account waiting times due to blocking communication. Such communication delays are going to be at least as high for the SA as they are for the ES, as the ES and SA communication schemes are very similar.

While our parallel ES is faster and more reliable than the SA algorithm, it does not in the current case scale well beyond 20 nodes. However, on larger problems a larger percentage of the time will be spent calculating the fitness function, which will bring down the proportion of time spent in communicating, and thus is likely to increase the efficiency of the algorithm such that it will scale efficiently beyond 20 nodes, and possibly beyond 50.

There are modifications that can be made to the parallel ES algorithm that will increase its parallel efficiency. As Figure 5 shows, a significant portion of the loss of efficiency appears to come from communication overheads, almost certainly due to the the nature of node-to-node blocking communication. It would not require any major changes to the algorithm for it to be able to communicate using a non-blocking, asynchronous method; there is no reason that migration must occur all at once, and it is entirely possible for populations to migrate when they are ready to do so. In this scheme, when a population comes to migrate, it will pick a partner according to some predefined method, send an individual to that population's buffer and then carry on with the

algorithm. All populations will integrate individuals from their buffer when they are next free to do so. There are some issues to be resolved, such as whether populations that are running faster should migrate more often or not (i.e. have migrations occur per generation or per unit time), but these are far from insurmountable. In contrast, the SA algorithm requires information on all nodes before it can perform mixing of states, with the result that an asynchronous method would require major changes to the algorithm and be far harder to develop.

However, we have also seen that the speed of the serial island-ES decreases as the number of islands beyond 8 are added, and thus even with hypothetically perfect communication the efficiency will continue to drop as more processors are added beyond that. To solve more complex problems we need either a method to increase the speed of the algorithms, or a method to allow the algorithms to scale up efficiently to increasing numbers of processors.

One method for increasing the speed comes from the observation that local searches virtually always fail on complex problems, but if they happen to start near the global minimum, they tend to converge on it very fast⁴¹. This suggests that the so-called hybrid algorithms, discussed above, could be especially effective. Once one has got a solution close enough to the global minimum, one can apply any of the range of local search techniques that exist, including many that function well in parallel⁶⁶. The role of the global optimisation algorithm now becomes to descent as fast as possible to a low-enough energy for the local search to converge. The island-ES has been shown to be able to achieve significant speed-up for the gap gene problem using this approach¹⁴, and the parallel ES is clearly perfect for this task, as it achieves a relatively low energy very fast, at least compared to Simulated Annealing.

One issue with this hybrid approach is that it either introduces one or more tuning parameters by requiring a set position to switch over, or it requires a way of theoretically determining the ideal switch-over point. This is important, as Mendes and Banga³⁹ showed that the changes in the switch-over point can have a large impact on the efficiency of the algorithm, with a late switch wasting time on slow global descent, and an early switch causing the local solution to fail to converge.

One method for increasing the parallel efficiency, specific to the parallel ES, is the hierarchical approach. As explained above, a hierarchical algorithm consists of small groups of nodes that run one aspect of an EA (either a master-slave EA, or a fine-grained EA) which are in turn part of a larger EA (either a fine-grained EA or an island EA). This becomes especially relevant to our parallel ES as the complexity of the model, and thus the computation difficulty in calculating the fitness function, increases. An appropriate hierarchical implementation would involve a number of master-slave clusters, each of which runs a single island population; the master node would perform selection, mutation and recombination and a set of slave nodes would evaluate the fitness functions for the individuals in the population. Then migration would occur between populations (i.e. clusters) normally. This algorithm becomes especially plausible as multi-core computers become increasingly popular; each population could run on the multiple cores in one computer, keeping down intra-population communication time (where the majority of communication overhead would be).

6 Conclusion

Systems biology, as with much of modern biology, is largely driven by technology and innovation. In the case of the gap gene network, it was an innovative technique that allowed the extraction of spatial expression data at high resolution that in turn allowed explicit modeling of the system, and it was a combination of increasing computer technology and the innovative parallel SA algorithm that allowed the 6-gene model to be fitted.

I have investigated in detail new innovations appearing now will allow further progress in the

reverse engineering of biological systems, and specifically how the fruits of a long tradition of Evolutionary Algorithms and their application to parameter estimation can be brought to bear on the problem.

The investigation has shown that the parallel Evolutionary Strategy is more efficient than Parallel Lam Simulated Annealing when applied to a real empirical problem, both in terms of speed and reliability. I have shown that it has reasonable parallel efficiency, such that the full force of the algorithm can be applied to a 10-node problem without loss of efficiency, and provided a detailed description of why it outperforms the SA algorithm on various accounts. I have also discussed issues related to the tuning of both algorithms, and shown how important this aspect of algorithm function can be.

The parallel ES algorithm is certainly a new and powerful tool, that is ready to solve actual problems. It will hopefully allow another incremental increase in the complexity of models that can be successfully fitted, and thus increase the breadth of our knowledge of the complexity of natural systems. However, I think it important that this parallel Evolutionary Strategy demonstrates not just the power, but also the potential of Evolutionary Algorithms. Research into the Evolutionary Strategy and EAs in general is ongoing, and I have discussed two simple ways by which the island-ES algorithm could be extended, one by giving a direct speed-up and another by allowing the algorithm to more effectively make use of modern technology.

Evolutionary Algorithms are inspired by the processes of the evolution of life, and as such they potentially have available to them the tools that lead to the most successful optimisation run we have yet examined¹⁰. It is apparent that there is a whole array of modifications and improvements that can be made to such algorithms, some of which are already known, and many more that are yet to be developed.

7 Acknowledgements

I would like to thank my supervisor, Johannes Jaeger, for ample support, advice and tutorage, and to the lab PI, Michael Akam. I would also like to thank Stuart Rankin and Wojciech Turek from the Cambridge High Performance Computing Service for computing support, and keeping such good care of Darwin, and to apologise to them for crashing more computers than I ever thought possible.

I am especially grateful to the Cambridge Computation Biology Institute for making this project possible, and the EPSRC for providing funding.

Computing resources were provided by the Cambridge Department of Zoology.

References

- [1] E.H.L. Aarts, F.M.J. de Bont, E.H.A. Habers, and P.J.M van Laarhoven. Parallel implementations of the statistical cooling algorithm. *Integration, the VLSI Journal*, 4(3):209–238, 1986.
- [2] P. Adamidis. Parallel Evolutionary Algorithms: A Review. In *4th Hellenic-European Conference on Computer Mathematics and its Applications*, 1998.
- [3] M. Akam. The molecular basis for metameric pattern in the Drosophila embryo. *Development*, 101(1):1–22, 1987.
- [4] T. Bäck and H.P. Schwefel. An Overview of Evolutionary Algorithms for Parameter Optimization. *Evolutionary Computation*, 1(1):1–23, 1993.
- [5] Thomas Bäck. *Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms*. Oxford University Press, 1996.
- [6] G.E.P. Box. Evolutionary Operation: A Method for Increasing Industrial Productivity. *Applied Statistics*, 6(2):81–101, 1957.
- [7] H.J. Bremermann. Optimization through evolution and recombination. In M. Yovits, G.T. Jacobi, and G.D. Goldstein, editors, *Self-Organizing Systems-1962*, pages 93–106. Spartan Books, Washington D.C., 1962.
- [8] E. Cantu-Paz. A survey of parallel genetic algorithms. *Calculateurs Paralleles, Reseaux et Systems Repartis*, 10(2):141–171, 1998.
- [9] K.W. Chu, Y. Deng, and J. Reinitz. Parallel Simulated Annealing by Mixing of States. *Journal of Computational Physics*, 148(2):646–662, 1999.
- [10] C. Darwin. *On the Origin of Species*. 1859.
- [11] V.E. Foe and B.M. Alberts. Studies of nuclear and cytoplasmic behaviour during the five mitotic cycles that precede gastrulation in Drosophila embryogenesis. *J Cell Sci*, 61(1):31–70, 1983.
- [12] T.C. Fogarty and R. Huang. Implementing the Genetic Algorithm on Transputer Based Parallel Processing Systems. In *PPSN I: Proceedings of the 1st Workshop on Parallel Problem Solving from Nature*, pages 145–149, London, UK, 1991. Springer-Verlag. ISBN 3-540-54148-9.
- [13] L.J. Fogel, A.J. Owens, and M.J. Walsh. *Artificial Intelligence through Simulated Evolution*. John Wiley, 1966.
- [14] Y. Fomekong-Nanfack, J.A. Kaandorp, and J. Blom. Efficient parameter estimation for spatio-temporal models of pattern formation: case study of Drosophila melanogaster. *Bioinformatics*, 23(24):3356–3363, 2007.
- [15] R.M. Friedberg. A Learning Machine: Part I. *IBM Journal of Research and Development*, 2(1):2, 1958.
- [16] R.M. Friedberg, B. Dunham, and J.H. North. A Learning Machine: Part II. *IBM Journal of Research and Development*, 3(3):282, 1959.

- [17] S. Geman and D. Geman. Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *IEEE transactions on pattern analysis and machine intelligence*, 6(6):721–741, 1984.
- [18] M. Gorges-Schleuter. ASPARAGOS, A Parallel Genetic Algorithm and Population Genetics. *Lecture Notes In Computer Science*, pages 407–418, 1989.
- [19] W. Gropp, E. Lusk, and A. Skjellum. *Using MPI : Portable parallel programming with the Message - Passing Interface, 2nd Edition*. MIT Press, 1999.
- [20] J.H. Holland. Outline for a Logical Theory of Adaptive Systems. *J. ACM*, 9(3):297–314, 1962.
- [21] J.H. Holland. *Adaptation in natural and artificial systems*. MIT Press, Cambridge, MA, USA, 1975.
- [22] J.J. Hopfield. Neurons with Graded Response Have Collective Computational Properties like Those of Two-State Neurons. *Proceedings of the National Academy of Sciences of the United States of America*, 81(10):3088–3092, 1984.
- [23] S. Horvath and J. Dong. Geometric Interpretation of Gene Coexpression Network Analysis. *PLoS Comput Biol*, 4(8):e1000117, 2008.
- [24] L. Ingber. Very fast simulated re-annealing. *Mathl. Comput. Modelling*, 12:967–973, 1989.
- [25] J. Jaeger. *Dynamic Regulatory Analysis of the Gap Gene Network in Drosophila melanogaster*. PhD thesis, Stony Brook University, New York, 2005.
- [26] J. Jaeger, M. Blagov, D. Kosman, K.N. Kozlov, E. Myasnikova, S. Surkova, C.E. Vanario-Alonso, M. Samsonova, and D.H. Sharp J. Reinitz. Dynamical Analysis of Regulatory Interactions in the Gap Gene System of *Drosophila melanogaster*. *Genetics*, 167(4):1721–1737, 2004.
- [27] J. Jaeger, S. Surkova, M. Blagov, H. Janssens, D. Kosman, K.N. Kozlov, M.E. Manu, C.E. Vanario-Alonso, M. Samsonova, D.H. Sharp, et al. Dynamic control of positional information in the early *Drosophila* embryo. *Nature*, 430(6997):368–371, 2004.
- [28] J. Jaeger, D.H. Sharp, and J. Reinitz. Known maternal gradients are not sufficient for the establishment of gap domains in *Drosophila melanogaster*. *Mechanisms of Development*, 124(2):108–128, 2007.
- [29] K.A. De Jong. *An analysis of the behavior of a class of genetic adaptive systems*. PhD thesis, 1975.
- [30] F. Kamme, R. Salunga, J. Yu, D.T. Tran, J. Zhu, L. Luo, A. Bittner, H.Q. Guo, N. Miller, J. Wan, and M. Erlander. Single-Cell Microarray Analysis in Hippocampus CA1: Demonstration and Validation of Cellular Heterogeneity. *J. Neurosci.*, 23(9):3607–3615, 2003.
- [31] S. Kirkpatrick, CD Gelatt, and MP Vecchi. Optimization by Simulated Annealing. *Science*, 220(4598):671–680, 1983.
- [32] J. Lam and J.-M. Delosme. An Efficient Simulated Annealing Schedule: Derivation. Technical Report 8816, Electrical Engineering Department, Yale, New Haven, CT, 1988.

- [33] J. Lam and J.-M. Delosme. An Efficient Simulated Annealing Schedule: Implementation and Evaluation. Technical Report 8817, Electrical Engineering Department, Yale, New Haven, CT, 1988.
- [34] H.J. Lichtfuss. *Evolution eines Rohrkrümmers*. PhD thesis, 1965.
- [35] D. Lim, Y.S. Ong, Y. Jin, B. Sendhoffand, and B.S. Lee. Efficient Hierarchical Parallel Genetic Algorithms using Grid computing. *Future Generation Computer Systems*, 23(4): 658–670, 2007.
- [36] R. Lohmann. Application of Evolution Strategy in Parallel Populations. *Proceedings of the 1st Workshop on Parallel Problem Solving from Nature*, pages 198–208, 1990.
- [37] W.G. Macready, A.G. Siapas, and S.A. Kauffman. Criticality and Parallelism in Combinatorial Optimization. *Science*, 271(5245):56–59.
- [38] G. Marnellos. *Gene Network Models Applied to Questions in Development and Evolution*. PhD thesis, 1997.
- [39] M. Rodriguez-Fernandezand P. Mendes and J.R. Banga. A hybrid approach for efficient and robust parameter estimation in biochemical pathways. *BioSystems*, 83(2-3):248–265, 2006.
- [40] P. Mendes. Modeling large scale biological systems from functional genomic data: parameter estimation. In H. Kitano, editor, *Foundations of Systems Biology*, pages 163–186. MIT Press, 2001.
- [41] P. Mendes and D.B. Kell. Non-linear optimization of biochemical pathways: applications to metabolic engineering and parameter estimation. *Bioinformatics*, 14(10):869–883, 1998.
- [42] N. Metropolis, A.W. Rosenbluth, M.N. Rosenbluth, A.H. Teller, and E. Teller. Equation of state calculations by fast computing machines. *The Journal of Chemical Physics*, 21(6): 1087–1092, 1953.
- [43] E. Mjolsness, D.H. Sharp, and J. Reinitz. A connectionist model of development. *J Theor Biol*, 152(4):429–53, 1991.
- [44] C.G. Moles, P. Mendes, and J.R. Banga. Parameter Estimation in Biochemical Pathways: A Comparison of Global Optimization Methods. *Genome Res.*, 13(11):2467–2474, 2003.
- [45] H. Mühlenbein, M. Schomisch, and J. Born. The parallel genetic algorithm as function optimizer. *Parallel computing*, 17(6-7):619–632, 1991.
- [46] E. Myasnikova, M. Samsonova, D. Kosman, and J. Reinitz. Removal of background signal from in situ data on the expression of segmentation genes in Drosophila. *Dev Genes Evol*, 215(6):320–6, 2005.
- [47] D. Nam, S.H. Yoon, and J.F. Kim. Ensemble learning of genetic networks from time-series expression data. *Bioinformatics*, 23(23):3225–3231, 2007.
- [48] A. Ngom. Parallel evolution strategy on grids for the protein threading problem. *Journal of Parallel and Distributed Computing*, 66(12):1489–1502, 2006.
- [49] I.M. Ong, J.D. Glasner, and D. Page. Modelling regulatory pathways in E. coli from time series expression profiles. *Bioinformatics*, 18:S241–248, 2002.

- [50] D. Pe'er, A. Regev, G. Elidan, and N. Friedman. Inferring subnetworks from perturbed expression profiles. *Bioinformatics*, 17:S215–224, 2001.
- [51] C.M.N.A. Pereira and C.M.F. Lapa. Coarse-grained parallel genetic algorithm applied to a nuclear reactor core design optimization problem. *Annals of Nuclear Energy*, 30:555–565, 2003.
- [52] E. Poustelnikova, A. Pisarev, M. Blagov, M. Samsonova, and J. Reinitz. A database for management of gene expression data in situ, 2004.
- [53] R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2008. URL <http://www.R-project.org>. ISBN 3-900051-07-0.
- [54] G.T. Reeves, C.B. Muratov, T. Schüpbach, and S.Y. Shvartsman. Quantitative Models of Developmental Pattern Formation. *Developmental Cell*, 11(3):289–300, 2006.
- [55] J. Reinitz and D.H. Sharp. Mechanism of eve stripe formation. *Mech Dev*, 49(1-2):133–58, 1995.
- [56] J. Reinitz, E. Mjolsness, and D.H. Sharp. Model for cooperative control of positional information in Drosophila by bicoid and maternal hunchback. *J Exp Zool*, 271(1):47–56, 1995.
- [57] J. Reinitz, D. Kosman, C.E. Vanario-Alonso, and D.H. Sharp. Stripe forming architecture of the gap gene system. *Developmental Genetics*, 23(1):11–27, 1998.
- [58] T.P. Runarsson and S. Sigurdsson. Asynchronous parallel evolutionary model selection for support vector machines. *Neural Information Processing—Letters and Reviews*, 3(3):59–68, 2004.
- [59] T.P. Runarsson and Xin Yao. Stochastic ranking for constrained evolutionary optimization. *Evolutionary Computation, IEEE Transactions on*, 4(3):284–294, Sep 2000.
- [60] K. Sachs, O. Perez, D. Pe'er, D.A. Lauffenburger, and G.P. Nolan. Causal Protein-Signaling Networks Derived from Multiparameter Single-Cell Data. *Science*, 308(5721):523–529, 2005.
- [61] H.P. Schwefel. *Kybernetische Evolution als Strategie der experimentellen Forschung in der Strömungstechnik*. PhD thesis, 1965.
- [62] E. Segal, M. Shapira, A. Regev, D. Peer, D. Botstein, D. Koller, and M. Friedman. Module networks: identifying regulatory modules and their condition-specific regulators from gene expression data. *Nat Genet*, 34(2):166–76, 2003.
- [63] K.A. Shepard, A.P. Gerber, A. Jambhekar, P.A. Takizawa, P.O. Brown, D. Herschlag, J.L. DeRisi, and R.D. Vale. Widespread cytoplasmic mRNA transport in yeast: Identification of 22 bud-localized transcripts using DNA microarray analysis. *Proceedings of the National Academy of Sciences*, 100(20):11429, 2003.
- [64] S. Surkova, E. Myasnikova, H. Janssens, K. N. Kozlov, A.A. Samsonova, J. Reinitz, and M. Samsonova. Pipeline for acquisition of quantitative data on segmentation gene expression from confocal images. *Fly*, 2(2):58–66, 2008.
- [65] H. Szu and R. Hartley. Fast simulated annealing. *Physics Letters A*, 122(3-4):157–162, 1987.
- [66] M.G.A. Verhoeven and E.H.L. Aarts. Parallel Local Search. *Journal of Heuristics*, 1:43–65, 1995.