5 Optimization of Microarray Experimental Design Using Genetic Algorithms

5.1 Introduction

Microarrays constitute a powerful tool for practical applications including, among others, diagnostics, target identification, screening, and genotyping. But they are also costly, both in time and resources, which make the careful design of microarray experiments critical to generate reliable and useful data. Statistical analysis of data generated from well designed experiments allows for meaningful biological correlation. Microarray experiments should be conducted as a group effort that requires efficient communication between those designing and running the experiments, those managing the databases and those doing the data analysis. This is paramount to generate statistically defensible data and is only possible if, from inception, this need is accounted for.

In this chapter the focus is on cDNA microarrays. This method for gene expression investigation allows comparison of expression levels in typically two biological samples for several thousand genes on a single slide. Since it was initially developed (Schena *et al.* 1995), microarray techniques have matured, contributing an important part in the generation of gene expression data at an unprecedented pace. Also referred to as spotted arrays, they usually consist of two reverse-transcribed RNA samples labeled to different color dyes, mixed and hybridized onto a slide. Microarray experimental techniques are fully described in (Schena 1999; Schena 2002; for a brief overview see section 2.4).

As with any other experimental approach, to succeed the objectives of the study must be clearly stated. This need has been brushed aside in the past since the quantity of data generated falsely suggests that almost any possible question can be addressed by simply analyzing the data (Simon *et al.* 2002). Regrettably that is not so, and with elevated costs and high demands on time, it has become clear that microarray studies have to be well defined as to their objectives and be well planned to ensure that the questions of interest can be effectively addressed. The planning stage among other considerations encompasses the experimental design of the study, which is the main topic of this chapter.

Many papers and books have been published over the last few years that discuss and present analysis methods for the interpretation of microarray data (Hegde *et al.* 2000; Dudoit *et al.* 2002; Knudsen 2002; Nadon & Shoemaker 2002; Pan 2002; Draghici 2003); of these, analysis of variance (ANOVA) methods, are growing in acceptance (Kerr *et al.* 2000; Draghici *et al.* 2001; Kerr & Churchill 2001; Kerr *et al.* 2002). But only recently has research focused on the issues involved in the effective design of microarray experiments (Draghici *et al.* 2001; Kerr & Churchill 2001; Churchill 2002; Dobbin & Simon 2002; Simon *et al.* 2002; Yang & Speed 2002; Simon & Dobbin 2003; Dobbin *et al.* 2003; Glonek & Solomon 2004; Vinciotti *et al.* 2005). The use of an appropriate design will not only reduce costs and save labor time; it will also ensure that the effects of interest will not be confounded with the many factors that influence gene expression data. The main sources of variability in spotted arrays are dye effects, array effects, spot effects (technical replicates on the same slide) and biological variability (biological replicates). All of these have to be accounted for to ensure an efficient and unbiased analysis.

Microarray experimental design is not straightforward. There are several ways an experiment can be designed; typical but not necessarily optimal designs include dye-swap designs, loop designs and

reference designs. Different constraints influence the design. Common practical considerations are to minimize the number of slides due to cost constraints, to reduce the number of steps between samples thus reducing variability, how to select the pair samples to hybridize and find an appropriate balance of the dyes. Unfortunately these different considerations are not necessarily linearly correlated. A reduction in the number of slides can unbalance the dyes or increase the average number of steps needed to compare a pair of samples. Depending on the level of interest on a specific aspect, different designs can be more efficient.

To find the best overall design that adequately balances conflicting constraints is not a trivial task. Microarray experimental design is essentially a multicriteria optimization problem. For this class of problems Evolutionary Algorithms are well suited for they can search the multicriteria solution space and evolve a design that maximizes the parameters of interest based on their relative value to the researcher and a given set of constraints.

Genetic Algorithms (GAs) – a class of evolutionary algorithms – are computational heuristics that use analogies of natural selection processes such as mutation, recombination and selection to evolve a population of candidate solutions based on an objective function (Goldberg 1987; Eshelman 2000; see section 2.5.4.3). In this chapter a Genetic Algorithm was developed to optimize the experimental design of spotted microarrays using a weighted multicriteria objective function. In section 5.2 the fundamentals of experimental design, the variability factors that influence spotted array data and current work in design optimization are reviewed. Section 5.3 introduces the Genetic Algorithm developed for design optimization and its implementation in the ArrayDesigner software. Results are presented in section 5.4, where simple GA-evolved designs are discussed to illustrate the method followed by a comparison of evolved designs with optimal designs. In section 5.5 some conclusions are drawn and future work possibilities are discussed.

5.2 Experimental Design of Microarrays

The objective of an experimental design is to ensure that the data gathered by an experiment has statistical value and can address the experimental questions as well as possible, given the experimental constraints. The principles behind the experimental design of microarrays are essentially the same as those developed by Fisher (1971) for agricultural research. The implication of this is that microarray experiments still fit into the classical statistical framework and should be treated as such (Yang & Speed 2002). For example the effect of different plots of land used in a crop experiment is analogous to the effect of different slides in a microarray experiment.

In the current context, microarray experimental design is about defining which samples will be hybridized together on a slide. A cDNA microarray is a comparison between two samples. How samples are paired in an experiment affects the comparisons that can be made from the data. In practical terms comparisons of interest should be closely connected in the design, preferably on the same array, thus removing the variability between slides. This is due to the fact that there is more variability between slides than within slides. Typically the correlation of measured intensities between duplicated spots on the same slide is around 95%; dropping to 60% – 80% on different slides (Churchill 2002). Further, the design determines if the effects of interest (sample x gene interactions) are confounded with other sources of variation. If the experiment is not carefully planned it may not be possible to separate the effects of interest from the other sources of variation, generating a biased dataset.

5.2.1 Sources of Variation

Spotted array experiments ultimately aim to measure the difference of expression levels between samples. The difficulty lies in distinguishing between the actual expression variation and the variation introduced by the method itself. The main sources of technical variability include: the method used to prepare the mRNA, the transcription steps, types of labeling, PCR amplification, variations in the pin surfaces, the amount of target that attaches to the slide, the parameters of the hybridization process, slide variation, non-specific hybridization, the gain settings of the scanner (PMT), high or low end saturation (dynamic range), alignment of the images, grid detection, background noise, contamination, shape of the spots and spot quantification (Draghici 2003:29).

For microarray analysis (within an ANOVA framework), and consequently for design considerations, four categories of variability should be considered: *varieties (V), genes (G), dyes (D)* and *arrays (A)* (Kerr & Churchill 2001). Varieties are the signal intensities of the factors of interest – different tissue samples or time points, for example. The genes are the sequences spotted on the slide – not necessarily genes; they can be ESTs or some other source of DNA, RNA or protein – this considers variation introduced by differences in the pin surfaces or target attached to the slide. Dye effects account for the variation introduced by the cDNAs labeled with the red and green dyes. Array effects consider the variation between slides; infrequently an experiment will consist of a single slide. An efficient design will account for these sources of variation and ensure that they are not confounded.

5.2.2 Replication

The starting point for a microarray design is replication. Churchill (2002) highlights two main types of replication: (1) technical and (2) biological. There are two types of technical replicates: (1) the same gene (*G*) is spotted multiple times on a slide and (2) multiple arrays (*A*) are hybridized with the same samples (*V*). Technical replicates do not consider biological variability; they are useful to reduce the noise and improve the precision of the measured signal. Spot replicates can be used to estimate the variance of G (σ_G^2). The variance of *A* (σ_A^2) can be estimated from array replicates. It is important to have a handle on technical variability; but it bears no relation to the biological variability. Biological variance (σ_V^2) can only be estimated from multiple measures of samples from a population. The importance of biological replicates cannot be over emphasized. No statistically sound inferences (assumptions that can be tested from the data) can be made if there are no biological replicates, since clearly a single sample is not representative of the population, or if residual variation cannot otherwise be measured and accounted for in a statistical model – for instance by fitting time of sampling over sufficient unreplicated samples.

While considering biological replicates it is worthwhile to briefly mention sample pooling. This strategy has been frequently used in microarray assays, particularly when the RNA extraction from a single individual is insufficient for hybridization. Apart from technical feasibility, pooling has been used to control biological variability. This is a valid approach to identify genes that are on average over or under expressed in two different varieties, but it provides no measure of confidence in the results achieved. Again, biological variance cannot be measured from a single pooled sample.

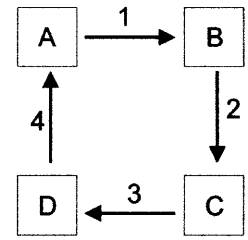
The precision of the study depends on the different sources of variability. The mean squared error (MSE) quantifies the precision of the estimation (decreases proportionally to the sample size); which in turn allows determining if estimated quantities are significantly different or not. MSE is estimated from

the residual variation or error variation observed within independent experimental units while precision is determined by the experimental residual degrees of freedom (Churchill 2002). As a rule of thumb at least 5 residual degrees of freedom should be available for an array experiment.

5.2.3 Graphical Representation of Designs

Microarray designs are commonly represented as directed graphs (see section 6.2 for a discussion on directed graphs). The nodes of the graph represent the varieties (*V*) and the edges represent the arrays (A). The direction of the arrow is used to represent the dye used to label each of the varieties. Customarily to the head of the arrow the red dye (Cy5) is assigned while the green dye (Cy3) is assigned to the arrow tail. Figure 5.1 illustrates a simple loop design with 4 arrays.

Figure 5.1 Microarray loop design with 4 arrays and 4 samples. Nodes represent the samples and the edges are the arrays. The arrow is used to represent the dye color: the head is the red dye (Cy5) and the tail is the green dye (Cy3). The samples are represented by the letters (A, B, C and D) and the arrays are numbered from 1 to 4. The samples co-hybridized on each array are: 1 – A/B, 2 – B/C, 3 – C/D and 4 – D/A.



Spotted arrays are comparative studies; as such it is the structure of the directed graph that defines which expression differences can be estimated, as well as the precision of these estimates, from the experiment. Consider two samples that are not hybridized on the same array; they can only be compared if there is a path in the graph that connects the nodes. The precision of this comparison depends on the number of edges that form the connecting path. The more steps needed to connect the samples the less precise the comparison is. More important experimental questions should be able to be addressed by short paths. As an example consider figure 5.1. The log ratio is commonly used to estimate the relative abundance of expression between samples. For samples *A* and *B* on the same slide the estimate is directly obtained from $\log(A/B)$; a less precise indirect comparison between *A* and *C* is obtained by $\log(A/C) = \log(A/B) - \log(C/B)$.

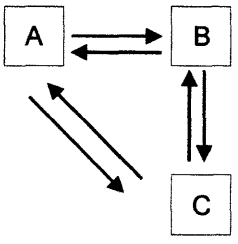
5.2.4 Types of Design

Only rarely does a microarray experiment consist of only two varieties. This means that in most cases not all samples appear on all arrays. Considering that an array can be viewed as an experimental block of size two, a microarray design with more than two varieties is an *incomplete block design*. In this section general characteristics of designs are discussed, followed by an overview of the two most adopted designs: reference and loop designs.

One of the main sources of systematic bias in cDNA microarrays is due to the labeling intensity of the dyes. A normalization step is usually performed to remove average dye biases (Schuchhardt *et al.* 2000; Quackenbush 2002; Leung & Cavalieri 2003; Smyth & Speed 2003); this leaves gene-specific dye biases to be accommodated. The reason why some genes (around 1%) present this bias is as yet unknown (Kerr 2003). An efficient method for accounting for this bias is through dye-swap where all

hybridizations are performed twice with the dyes swapped (figure 5.2). Full dye-swap of all slides is not necessary, it is sufficient to have an *even* design (Kerr & Churchill 2001; Kerr 2003). For a design to be even, each sample should be labeled with both dyes and each dye should be used the same number of times in the experiment. The loop design in figure 5.1 is an example of an even design – the dyes are balanced and each sample is used once with each dye. A condition for balance is that each variety is used an even number of times (the degree of all nodes must be even in the graph).

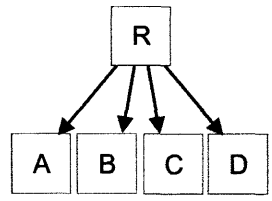
Figure 5.2 A Dye-swap loop design. For each pair of samples two hybridizations are performed with the dyes reversed in the second hybridization. In this example, for all three samples a direct comparison is possible.



Gene expression level comparisons can be direct – comparison between samples on the same slide or indirect – comparison between samples on different slides. If the variance for a single log ratio is σ^2 (direct comparison) it follows that the variance for a two step indirect comparison is $2\sigma^2$ (considering the same variance for within and between slide comparisons). The immediate implication is that an indirect comparison has a two-fold variance increase and demands two slides (two independent routes for comparison) instead of one; in this simple scenario costs and material are effectively doubled. If the same number of slides (2) are used the difference becomes even more evident. Two direct comparisons can be done (dye-swap can be used to account for dye bias) yielding two independent measurements and a variance of $\sigma^2/2$. At the same cost the variances between the two designs can vary by a factor of 4. Note that this factor is theoretical since in practice target spots are usually from the same biological sample and the independence assumption is not met. Nevertheless a direct comparison is usually more precise than an indirect comparison.

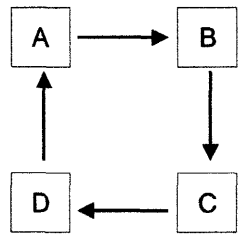
For experiments with just a few varieties, an all-pairs design is feasible (all samples are hybridized with one another). As the number of varieties increases this design becomes unfeasible due to lack of sample availability and/or cost constraints. For this scenario reference and loop designs are commonly used. The reference design has traditionally been the most widely used design in microarray experiments (figure 5.3). In this design a common sample reference is hybridized to all slides using the same dye to label the reference across all hybridizations. To use this design it must be assumed that there are no gene-specific dye effects; since the effects of the samples are completely confounded with the dye effects. The reference design is unbalanced. Another consideration is that half the measurements are collected from the reference, which is usually of little interest. Thus, for n varieties of interest $n-1$ arrays must be used. Further, since the comparisons of interest are usually between samples, all relevant comparisons are indirect. The positive aspects of reference designs are their extensibility and consistent number of steps between comparisons. Given that the reference is readily available, an experiment can be setup to initially include a small number of varieties and latter on further varieties can be attached to the original experiment. It is also noteworthy that, even if no direct comparisons are possible, the path to compare any two varieties is of size 2.

Figure 5.3 Reference Design. Four samples of interest are hybridized to a common reference. All comparisons are indirect with a step size of 2.



An alternative to the reference design is the loop design. In this design all varieties are connected to one another in a loop (figure 5.4). For *n* varieties *n* arrays are used. Each variety is used twice, once with each dye – the design is balanced, gene effects are not confounded with dye effects. In comparison to the reference design, twice the amount of information on the varieties of interest is collected using the same number of arrays (if the reference is not a variety of interest). A further advantage of the loop design is that direct comparisons are available between varieties of interest. In the simple example of 4 samples depicted in figure 5.4, out of the 6 possible comparisons, 4 are direct and only 2 indirect. The indirect comparisons are of step size 2. From the examples it seems clear that, for four varieties, the loop design is superior to the reference design. Unfortunately the problem is not so clear cut. Robustness is an important consideration in microarray experiments, particularly if the samples are limited and there is not enough material to duplicate an array if the hybridization fails. A reference design is amenable to technical problems with an array. If one of the hybridizations does not work only one variety is lost. With the loop design a single bad array can break the loop and jeopardize the entire experiment. A second disadvantage of loop designs is that as the number of varieties – and consequently arrays – increases, the number of steps in the path needed to compare the varieties also increases. If a comparison of all pairs of varieties is desired, loops with more than ten samples are inefficient (Churchill 2002). In practical terms, loop designs effectively double the labeling reactions since each sample is labeled with both dyes.

Figure 5.4 Loop Design. Four samples are hybridized one to another. The design is balanced – each sample is used twice, once with each dye. There are 4 possible direct comparisons (A/B, B/C, C/D, D/A).



There is no single optimal design that is optimal for all experimental questions. A microarray experimental design should try to balance three basic principles: (1) balance among the factors – particularly dyes, (2) use approximately the same sampling of varieties and (3) minimize the distances between pairs of varieties – especially the ones of interest which should preferably be hybridized on the same array allowing for direct comparisons (Kerr & Churchill 2001).

5.2.5 Design Optimization

As yet little has been done on optimization of microarray experimental designs. Kerr and Churchill (2001) studied different designs for up to 10 samples with various numbers of slides and compiled a list of

optimal designs (later extended to 13 varieties, designs available from http://www.jax.org/staff/Churchill/labsite/research/expression/design.html). Yang (2003) extended this work to 26 varieties with an equal number of slides.

The objective of an optimal design is to assign the varieties to slides in a manner that will maximize the precision of the resulting parameter estimates. The definition of *optimal* depends on the optimality criterion that is used. The most common criteria for design optimality are *D-optimality*, *E-optimality*, *A-optimality* and its derived *L-optimality*. Kerr and Churchill (2001) used *A-optimality* as the criterion for their work; even though Witt *et al.* (2005) argue that the proper term is *L-optimality*, a modified form of *A-optimality*; see 5.3.3 for details of optimality criteria).

A-optimality seeks to minimize the trace of the inverse of the design matrix. This criterion results in minimizing the average variance of the parameter estimates based on a pre-specified model. With this criterion, designs that minimize the average variance of a contrast are favored (equation 5.1).

$$\frac{1}{\binom{v}{2}} \sum_{k1 \neq k2} \mathrm{var}(VG_{k_1 g} - VG_{k_2 g}) \tag{5.1}$$

Where k_1 and k_2 are the varieties, g are the genes and $VG_{k_1 g} - VG_{k_2 g}$ are the contrasts of interest -- variety x gene interactions (Kerr & Churchill 2001).

Ideally, for a combination of any number of varieties using any number of arrays, an optimal design would be available. But an exhaustive search of the entire design search space, even for a relatively small number of varieties and slides, is unfeasible. An alternative is to use optimization heuristics that can explore the solution space within a reasonable timeframe. To our knowledge the only work (Witt *et. al* 2005 – prepress) in this field used Simulated Annealing to find near-optimal designs based on maximization of optimality scores (*D-optimality*, *A-optimality* or *L-optimality*).

5.3 A Genetic Algorithm for Optimization of Microarray Experimental Designs

In this work Genetic Algorithms (GAs) are proposed for the optimization of cDNA microarray experimental designs. The method allows optimization of designs in relation to the number of slides, definition of hybridization pairs and dye allocation. Optimization of the number of spot replications on slides was not considered in this study.

The most widely disseminated Evolutionary Computation branch, GAs derive from the seminal work of Holland (1975). GAs have been widely used in optimization problems where the solution space to be searched is too large to allow exhaustive search or the parameters are heavily constrained (Michalewicz & Fogel 2000). The fundamentals of GAs were presented in section 2.5 of chapter 2 and the reader is referred to that that section for an overview.

5.3.1 Design Representation

Optimization of spotted array designs can be viewed as an assignment problem with three optimization parameters: (1) number of arrays (slides), (2) allocation of hybridization pairs to the slides and (3) dye allocation for the variety pairs on the slides.

Computationally, each candidate design – a chromosome in the GA population – is represented as a numeric array of index$[n, m]$ where *n=2* (block size) corresponds to each one of the channels in the microarray, thus defining the labeling dye of a sample. Index *m* is the maximum number of hybridizations

allowed. The experimental samples (varieties) are assigned a unique numeric identifier s_i in the array. In this simple manner complex designs can be easily represented with the hybridization pairs defined by position *m* and the dye colors by *n*, as depicted in figure 5.5. An additional dimension is added to the array, corresponding to the population size of the GA.

Figure. 5.5 A GA chromosome with 5 varieties (numbered $s_0 - s_4$) used to represent a loop design. Dimension *n* is used for dye assignment and dimension *m* represents the number of arrays in the design.

Green Channel (Cy3) - n_0	s_0	s_1	s_2	s_3	s_4
Red Channel (Cy5) - n_1	s_1	s_2	s_3	s_4	s_0
Slide	m_0	m_1	m_2	m_3	m_4

To allow for variable design sizes (different number of slides) a vector is used to control the effective experimental size. The effective size (*ES*) is an integer denoted as $ES \in [s-1, m]$, where *s* is the number of samples in the study and *m* is the maximum allowed number of arrays. The minimum number of slides is defined as $s-1$ since this is the minimal criterion for connectivity. ES is an evolvable parameter (see below) used as a cutoff point in the design array.

5.3.2 Genetic Operators

5.3.2.1 Initial Population

Each chromosome in the GA consists of a candidate microarray design. The initial population is generated by randomly assigning hybridization pairs from the sample pool to the chromosome array up to the maximum allowed number of slides. The first sample of the pair is assigned the green channel (n_0 in figure 5.5) and the second sample is assigned to the red channel (n_1 in figure 5.5). If a pair is formed by the same sample (self hybridization) a new pair is randomly selected until a pair of two different samples is formed. The effective size (ES) integer is randomly selected and assigned to the chromosome. *ES* defines the actual number of slides in the design. Once a chromosome has been constructed the objective function is called and a fitness value is assigned to it.

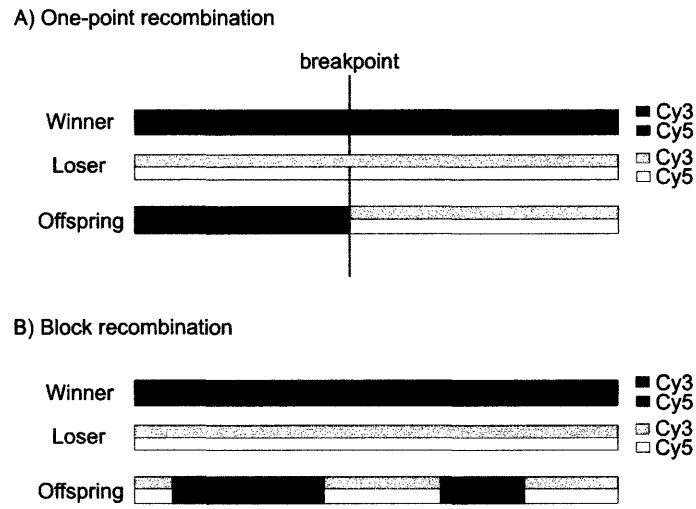
5.3.2.2 Selection

The GA uses steady-state generations and selection is elitist with tournament selection (Bäck *et al.* 2000a). The elitist approach ensures that the best solution is always retained in the population. ArrayDesigner (see 5.3.4) uses a fixed tournament of size 2; which in preliminary runs yielded a good balance between evolution rates while avoiding premature convergence (data not shown). The winner of the tournament remains in the population and the loser is replaced by its offspring. The recombination operator uses the tournament winner and the loser to generate an offspring which will replace the respective loser in the population. If recombination does not occur, offspring is a copy of the tournament winner. Once the offspring has been created (by recombination or by copying the tournament winner), it is modified by the mutation operators.

5.3.2.3 Recombination

In ArrayDesigner (see 5.3.4) the recombination probability is user defined. Two recombination operators are used with equal probabilities: (1) *one-point recombination* and (2) *block recombination*. The first method randomly selects a breakpoint in the chromosome (along the slides – array dimension *m*). An offspring is generated by randomly selecting either the winner or the loser of the tournament; the selected parent is copied into the offspring from the beginning of the design array up to the breakpoint. The remainder of the offspring is built by copying the other parent from the breakpoint until the end of the array. In block recombination the tournament loser is copied into the offspring and, with equal probability, a random number of blocks (between 1 and 3) of random sizes are selected from the tournament winner and grafted into the offspring in the same position they held in the tournament winner. A block consists of a variable number of slides with the hybridization pairs and channel assignments. The effective size of the offspring is defined by evaluating its fitness using the *ES* of both parents; the *ES* that yields the higher fitness is assigned to the offspring. Figure 5.6 illustrates the two recombination operators.

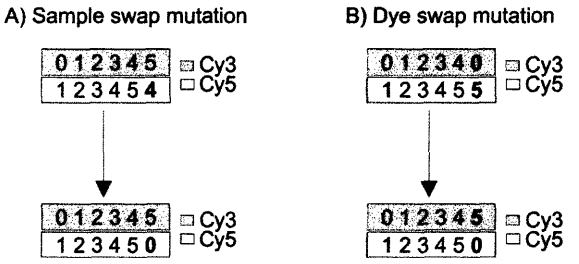
Figure 5.6 Recombination operators. A) One-point recombination – offspring is generated by copying one of the parents from the start of the array up to the breakpoint and copying the other parent from the breakpoint to the end of the array. B) Block recombination with 2 blocks – the tournament loser is copied into the offspring and blocks of variable size from the winner are grafted into the offspring.



5.3.2.4 Mutation

There are three mutation operators: (1) *sample swap*, (2) *dye swap* and (3) *effective size mutation*. The first operator tests both channels on all slides of the chromosome and randomly swaps the current sample with a different one if the operator returns true. This operator ensures that the entire solution space is accessible for exploration. The dye swap operator tests all slides and swaps the channels of the hybridization pair if the operator returns true. This operator is important to explore designs that are evenly balanced. Both operators are depicted in figure 5.7. The last mutation operator assigns a random number of slides – within the range: number of samples minus one to maximum allowed number of slides – as the effective size of the design. The *ES* is only adopted if it improves the fitness of the chromosome. This demands two fitness function calls, one with the original *ES* (from recombination) and one with the mutated *ES*. Designs are very sensitive to changes in the number of slides, for this reason the overhead of making three additional fitness calls for each chromosome is justified.

Figure 5.7 Mutation operators. A) Sample swap mutation – a sample in the array (in bold) in replaced by a new sample, with this single change the design becomes a loop design. B) Dye Swap mutation – the dye channels of a hybridization pair are swapped (in bold), the change turned the unbalanced design into a balanced one.



5.3.2.5 Repair Operator

A last operator – a *repair operator* is used to avoid self-hybridization. If a sample, through mutation, pairs with itself, one of the channels is replaced by a different sample. Even though self-hybridization could be penalized in the fitness function, preliminary investigations evidenced that inclusion of the repair operator increases the efficiency of the GA (data not shown).

5.3.3 Fitness Function

The fitness function is the key component in the genetic algorithm. A microarray design has to take into consideration the various factors that affect the experiment. These factors are mapped to key optimization parameters and the optimization problem is treated as a multi-objective problem since no solution can simultaneously completely satisfy all criteria.

In multicriteria problems there usually is no unique optimal solution but rather a Pareto front of solutions. Since objectives can conflict, improvements in one objective can degrade another one. Different combinations of values for the different objectives can yield the same total fitness; this implies that there is no unique optimal solution to the problem but rather a set of solutions with the same fitness (Pareto-optimal set). As an example, consider how to balance the number of slides (cost constraints) with the experimental questions (information constraints). With few slides the costs are low but there is not enough information to address the experimental questions; on the other extreme there is surplus data but at a very high cost.

With multicriteria problems the method used to determine the relative importance of each objective is critical for the optimization algorithm. A common approach to multi-objective optimization is to use a weighting scheme for the different objectives (Zitzler *et al.* 2000; Van Veldhuizen *et al.* 2000). There are several approaches to the weighting scheme, these methods range from a fully self adaptive approach – the scheme evolves alongside the optimization algorithm, in a similar way as mutation parameters evolve in Evolution Strategies – to a user-defined approach where the user modifies weights based on personal preferences (see Kinghorn (2000) for an example). The latter approach is less automatic but allows a higher degree of control by the user. The weighting scheme used in this work is user-defined; for each optimization parameter, a weight that reflects the relative importance of the parameter in relation to the others is defined by the user – the range used for the weights is $[0,1]$. This approach confers greater control of the search process to the user; which is a particularly important aspect, given that mathematically optimal designs might be inadequate due to practical constraints.

The basic aspect of the objective function is to evaluate the connectivity of the design. If slides are not connected no comparisons are possible and effectively there is no design. Before evaluating the

different objectives, design connectivity is tested; if the design is unconnected a very negative fitness is assigned to the chromosome ensuring that it will rapidly be eliminated from the population. If the chromosome represents a connected design, the other criteria are evaluated (no fitness value is assigned for connection). The criteria included in the objective function are: (1) Lower number of arrays, (2) lower steps for between slide comparisons, (3) higher power for specific contrasts of interest, (4) better balance dyes and (5) lower average variance of comparisons (A-optimality). The effective size (*ES*) parameter determines the number of slides in the first criterion. Evidently, if this criterion is used in isolation (weight 1, and 0 for the other criteria) the algorithm will rapidly reduce the number of slides to the number of samples minus one, which is the lowest value of the *ES* range that guarantees connection.

The second criterion uses Floyd's algorithm, to compute the shortest path between all possible pairs of nodes in the design. Floyd's algorithm uses dynamic programming (see 4.2.1) to find the all-pairs shortest path; the method makes use of an adjacency matrix to solve the problem simultaneously in $O(n^3)$ time (where n is the number of nodes in the graph). The criterion returns the sum of the shortest paths between all varieties. The underlying assumption is that by minimizing the overall number of steps between comparisons, precision is increased.

In an experiment not all contrasts are of the same interest. The third criterion accounts for these experimental interests. In ArrayDesigner the user can load a *contrasts file* with information on the relative importance of the different comparisons (see 5.3.4 for a description of the *contrasts file*). For each variety a numeric value can be assigned. The criterion either maximizes or minimizes the sum of squares of the differences between hybridization pairs (equation 5.2).

$$f = \sum_{m} (S_{n_0} - S_{n_1})^2 \tag{5.2}$$

Where m represents the slides and S the contrast values of the samples spotted on channels n_0 and n_1. Contrasts files can be used to optimize multifactorial designs.

The fourth objective attempts to balance the dyes on the red and green channels. The function is simply the number of balanced samples (samples that appear an equal number of times in each channel) divided by the total number of samples subtracted from one (equation 5.3, *BalS* – number of balanced samples and *TotalS* – total number of samples).

$$f = 1 - \frac{BalS}{TotalS} \tag{5.3}$$

The last criterion minimizes the variance searching for A-optimal designs (Kerr & Churchill 2001; Yang & Speed 2002); the shortest-path approach (criterion 2) is computationally more efficient than estimating the variance from the diagonal elements of the design matrix (X) which uses expensive matrix inversions (A-optimality minimizes the trace of $\sigma^2 (X'X)^{-1}$), but it is less robust since several designs can have the same average number of paths while still having different average variances. A further consideration, particularly for A-optimality, is that the A-optimal design depends on the chosen parameterization. Thus a design that is A-optimal for a given parameter set may not be optimal for a different one. For this criterion the average variance of all possible parameterizations is computed (Yang & Speed 2002). The method can easily be extended to include other forms of design optimality for which parameterization is not a problem: D-optimal designs – minimize the determinant of $\sigma^2 (X'X)^{-1}$ – or L-

optimal designs – minimize the trace of $\sigma^2 C(X'X)^{-1}C'$, where *C* is the *Contrasts Matrix* (Witt *et al.* 2005).

The fitness value of the chromosomes does not map directly to the objective function. Fitness is rank based – the fitness of a chromosome depends on its ranking in the population:

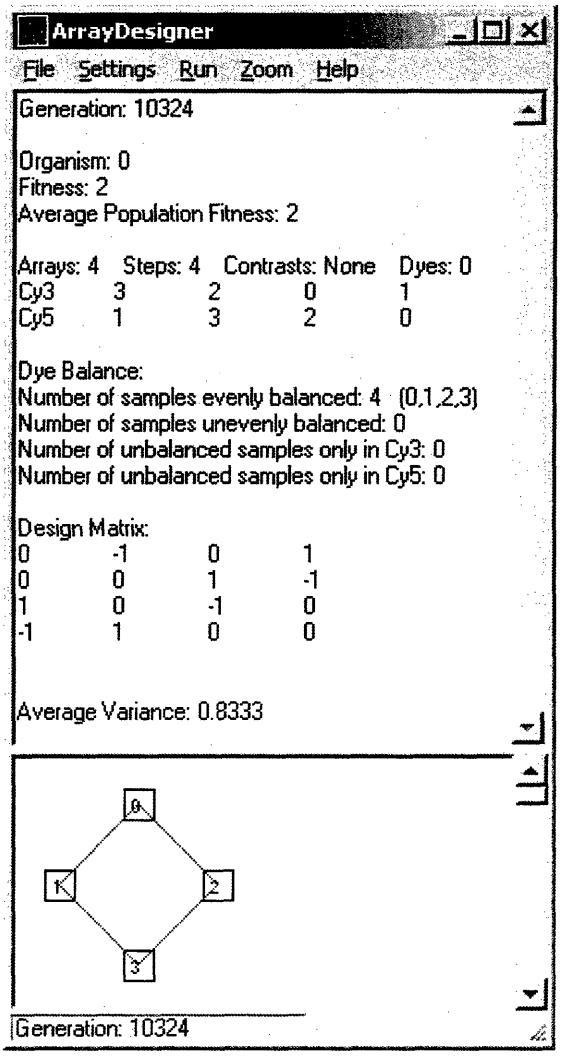
$$f = \sum_i 1/\sqrt{R_i} * W_i \tag{5.4}$$

Where R_i is the position of the chromosome based on the absolute values of criterion *i* ranked in ascending order and W_i is the relative weight of the criterion. Fitness is treated as a maximization problem; the best organism currently in the population gets a score between 1 and 5 (depending on the number of criteria selected) if the weight(s) are set at 1.

5.3.4 ArrayDesigner – Software for Microarray Design Optimization

The Genetic Algorithm was implemented in ArrayDesigner. The software was written in C# and runs under the .Net platform. The program allows setting the genetic algorithm parameters – population size, number of generations, mutation and recombination rates; as well as setting weights for the different criteria and defining the microarray parameters (figure 5.9). The software allows loading contrast files which can be used to define the importance of different comparisons. Optimization results are presented in graphic and text format (figure 5.8).

Figure 5.8 Screenshot of ArrayDesigner – a simple example with four samples and a fixed number of four slides. The optimization criteria were: balance dyes and reduce number of steps; equal weights (*1.0*) were used for both criteria. The upper pane presents result of the run, the optimized design in text format and dye balance information. The lower pane shows the optimized design as a graph.

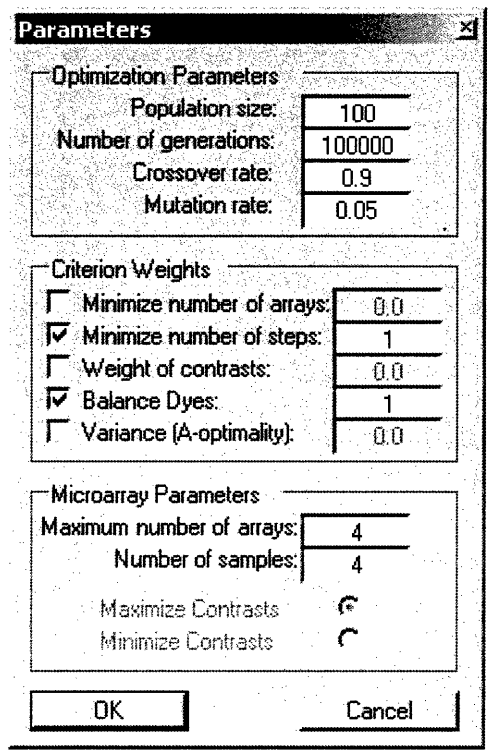


The optimization results shown in the upper pane in figure 5.8 are:

1. Generation – total number of fitness calls in the run.
2. Organism – the best chromosome of the run, if more than one chromosome has the same fitness the first one is shown.
3. Fitness – the fitness of the best chromosome. The maximum possible fitness is the number of criteria selected (between 1 and 5) multiplied by their respective weights. For example, if two criteria are selected with weights of respectively 1.0 and 0.5, the maximum fitness is 1.5. In figure 5.8 two criteria were selected (minimize number of steps and balance dyes) each with a weight of 1.0 – the maximum fitness is 2.0. Note that fitness is rank based, if during a run the maximum fitness is obtained it does not necessarily mean that the run has converged; it means that the best current solutions are aligned across criteria.
4. Average Population Fitness – is a useful measure of the degree of convergence of a run. When it equals the maximum possible fitness, all chromosomes are on the Pareto front – an indication of convergence.

5. Arrays, steps, contrasts and dyes – respectively the number of arrays in the design, the average number of steps between samples, sum of the squares of the contrasts and ratio of unbalanced samples.
6. The hybridization pairs and their respective dyes.
7. Details of dye balances.
8. Design matrix for the experiment.
9. The average variance of the design.

Figure 5.9 Screenshot of the parameter settings window in ArrayDesigner.



5.3.4.1 Contrasts Files

Contrasts files are plain text files that can be loaded into ArrayDesigner to provide information about comparisons of interest. They consist of row(s) of numerical values that are assigned to the samples. The data are used to optimize a design that either maximizes or minimizes the differences between hybridization pairs. Contrasts files can be used in two manners: (1) explicitly set up to convey information about the relative importance of hybridization pairs or, more interestingly, (2) they can contain information about the samples without any particular consideration as to the array experiment.

To illustrate the first case, consider a small time course experiment with five samples – one sample before a treatment was applied and 4 measurements at different time points – in which the main contrast of interest is between the untreated sample and each of the time point measurements. A contrasts file (maximization) that reflects this relationship would be:

1 0 0 0 0

With *1* for the untreated sample and *0* for the time course samples. If only four slides are available the optimal design is a reference design using the untreated sample as reference; which is the optimized design generated by the GA (see example in 5.4).

In the second case, the file can hold information on the varieties themselves. For example, an experiment aims to evaluate the gene expression differences between heavy and light livestock. The contrasts of interest are between heavy and light animals and the file (maximization) could consist of live weight measurements in kilograms of the animals such as:

250 270 382 235 395 410

The GA tries to match up pairs that maximize the weight differences and thus the contrasts (see example in 5.4). The contrasts file is not limited to a single source of data; additional rows can be added with other relevant information, for instance, daily weight gain, body fat and eye muscle area. The contrasts file is an interesting feature that allows incorporation of prior experimental data/knowledge into the design.

5.4 Optimization of Microarray Experimental Designs

To illustrate the method, six simple experimental scenarios were selected. Through these test cases it is easy to evidence how different criteria and weights can affect the final design. The method is then compared to more complex optimal designs found by Kerr and Churchill (2001).

The GA was used to construct designs for the following experimental parameters and design considerations: (1) time course with one main contrast of interest, five varieties (*V=5*) and four slides (*A=4*); (2) comparisons are of equal interest, *V=5*, *A=5*; (3) minimize number of arrays, *V=4*, *A=10*; (4) minimize number of steps, *V=4*, *A=6*; (5) dye balance, *V=3*, *A=6*; (6) comparisons are of different interest, *V=6*, *A=6*.

The GA parameters used were the same for all runs and are shown in table 5.1. The optimization criteria and design parameters for each test case are summarized in table 5.2. Figure 5.10 shows the graphs of the evolved designs. The *variance* criterion is not used in these examples (weight set at 0.0), it is used in the comparison study further down.

Table 5.1 GA parameters used for design optimization. Tournament size is hardcoded in ArrayDesigner.

Parameter	Value
Population size	100
Number of generations	100000
Crossover rate	0.9
Mutation rate	0.05
Tournament size	2

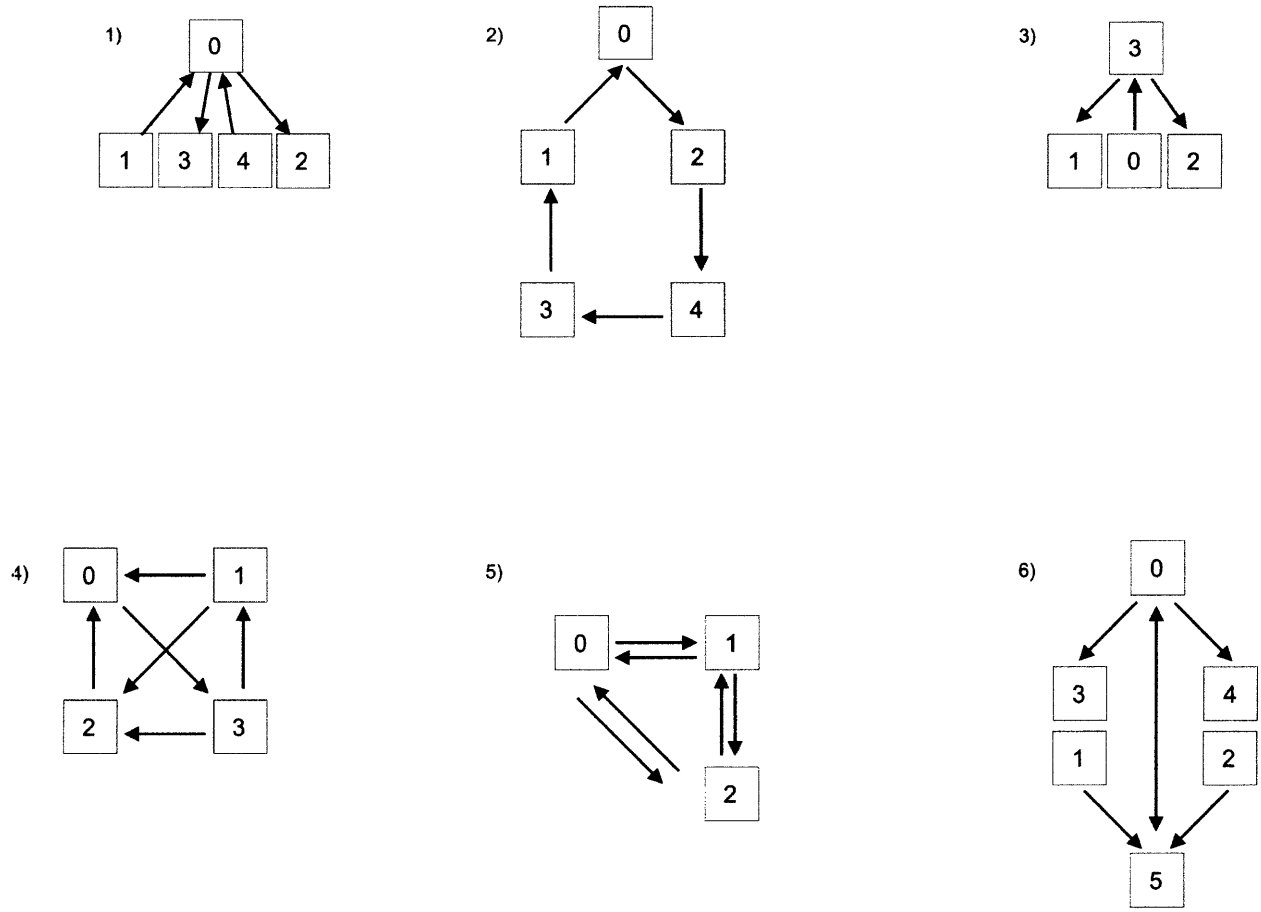
Due to the stochastic nature of GAs each run can evolve a different design. If a single objective is targeted, designs are usually equivalent for the same fitness value. With multiple criteria there may be several designs with the same fitness but structurally different. Five runs were performed for each test case. Differences in evolved designs and main points of each example are discussed below.

Table 5.2 Criteria and design parameters for 6 experimental designs. In test case 3 the value for slides refers to the maximum possible number of slides. The values for the criteria are the weights used – a value of 0.0 means that the criterion was not used.

	Design Parameters		Selected Criteria			
Design	Varieties	Slides	Minimize Arrays	Minimize Steps	Contrasts File	Balance Dyes
1	5	4	0.0	0.0	1.0	0.5
2	5	5	0.0	1.0	0.0	0.5
3	4	10	1.0	0.0	0.0	0.0
4	4	6	0.0	1.0	0.0	0.5
5	3	6	0.0	0.0	0.0	1.0
6	6	6	0.0	0.0	1.0	0.0

1. *Time course design* – the comparisons of interest are between a hypothetical untreated sample and 4 samples from different time intervals. A contrasts file (*1,0,0,0,0,0*) with the relative importance of hybridization pairs was used. The evolved design is the reference design (figure 5.10-1) where the reference is the untreated sample, which is the optimal design. Label assignment varies over runs and the samples are unbalanced. Four samples are used only once, since balance requires samples to be used an even number of times they cannot be balanced. The untreated sample can be balanced (used 4 times – even) by including the *balance dyes* criterion in the run. Dye balance weight should be kept low not to compete with the main criterion.
2. *Comparisons of equal interest* – since all contrasts are of equal importance, the main criterion is to minimize the number of steps between comparisons, which minimizes the total variance. The evolved design is a balanced (edges per sample = 2) loop design (figure 5.10-2). Loop designs are optimal for *V=5* and *A=5* (Kerr & Churchill 2001).
3. *Minimize number of arrays* – without any other constraints the GA rapidly evolves a design with the lowest possible level of connectivity between samples ($A = V - 1$). The evolved design is an unbalanced reference design with 3 slides (figure 5.10-3). Different runs produce alternative designs; either the reference sample changes or the design is linear (for example 0→1, 1→ 2, 2→ 3). A reference design can only be obtained if the *balance dyes* criterion is not used. In the reference design all samples are unbalanced; in the linear two varieties are balanced, thus the design always evolves to a linear design.
4. *Minimize number of steps* – similar to example 2 but with *V=4* and *A=6.* The evolved design is fully connected with direct comparisons between all samples (figure 5.10-4). For these values of *V* and *A* this is the design with the lowest variance (Yang & Speed 2002).
5. *Balance dyes* – with *V=3* and *A=6*, the evolved design is a fully connected dye swap (figure 5.10-5).
6. *Comparisons of different interest* – The contrasts of interest are between 6 hypothetical animals with high (samples 0,1,2) and low (samples 3,4,5) weights. The contrasts file is live weight in kilograms (200,210,220,300,310,320). The evolved design (figure 5.10-6) uses the extreme high and low values to hybridize with the other samples. The design is essentially two reference designs (the references are the extreme values) connected to each other through the references (dye swap).

Figure 5.10 Evolved designs for six example cases. 1) Time course design. 2) Comparisons of equal interest. 3) Minimize number of arrays. 4) Minimize number of steps. 5) Balance dye. 6) Comparisons of different interest.



The GA was benchmarked against optimal designs found by Kerr and Churchill (2001) through exhaustive search. Five designs were selected for comparison ($V = 6,7,8,9,10$ and $A = 2 * V$). The GA parameters were the same as used in the previous examples (table 5.1); five repeats were performed for each. The selected criteria were: *minimize error variance* (weight 1.0) and *balance dyes* (weight 0.5). Table 5.3 shows the average error variances for the best evolved designs and the benchmark designs. Variances were calculated as the average of the variance of all possible parameterizations, according to Yang and Speed (2002). Briefly, for each possible parameterization, a design matrix was constructed and the variance was obtained from the trace of $(X'X)^{-1}$ were X is the design matrix for a given parameterization.

Table 5.3 Average variances for all possible parameterizations of GA evolved designs and A-optimal designs from Kerr & Churchill (2001). *V* represents the number of varieties and *A* the number of arrays in the design.

Design Parameters	GA σ2	A-optimal σ2
V=6, A=12	0.4333	0.4333
V=7, A=14	0.4587	0.4587
V=8, A=16	0.4232	0.4643
V=9, A=18	0.4207	0.4891
V=10, A=20	0.4323	0.5

In all instances the GA evolved designs with the same or a lower error variance than those of Kerr and Churchill (2001). It is interesting to note that the benchmark designs were obtained through

exhaustive search and it should not be possible to improve on these results. This highlights the fact that different optimality criteria have different optimal designs. The designs of Kerr and Churchill are L-optimal (as per Witt *et al.* 2005) but not necessarily A-optimal. More importantly than the optimality criterion employed, though, is that the GA can find optimal or near optimal designs without exhaustive search of the solution space. In practical terms the method can easily be extended to any optimality criterion.

5.5 Conclusion and Future Work

Design optimization is a relevant topic for microarray studies given their high costs and time demands. It is critical before entering the experimental stage that the design can address all the biological questions. Further, since experiments are getting bigger, it is computationally intractable to exhaustively search all possible designs to select the best; this warrants the development of algorithms that can do a non-exhaustive *intelligent* search of the promising areas of the solution space.

In this chapter a Genetic Algorithm was presented for optimization of microarray experimental designs. The algorithm was implemented in the software ArrayDesigner. The method allows finding optimal or near optimal designs for large experiments (more than 20 samples/arrays) in a reasonable timeframe. Optimization is based on user defined multiple criteria, which allows exploring how different parameters and weights affect the experimental design. Thus, ArrayDesigner can also be used as an exploratory tool to investigate how, for example, a different number of slides affects the precision of the experiment – sometimes at a relatively small cost increase large efficiency gains can be attained.

Genetic Algorithms maintain a clear distinction between the optimization algorithm and the criteria which makes them well suited to test different objective functions. In this study, five criteria were implemented but these can easily be extended to include other objectives or different ones. For instance, the A-optimality method used in ArrayDesigner can be replaced by D-optimality or L-optimality without any changes to the GA.

To our knowledge this is the first time a GA is used to design microarray experiments. There is still a lot of work to be done in the area. Future work includes evaluating different objective functions, particularly optimality methods (D-optimality, E-optimality and L-optimality); an in-depth study of how the weighting scheme affects the outcome of the design; and, the relationship between criteria. An important aspect of design that was not considered in this study is the number of replicate spots per slide and how to optimize them – this warrants investigation.

Multiple criteria tend to yield several different designs that are equivalent in terms of fitness. ArrayDesigner will be modified to include, instead of a unique result, the Pareto set of equivalent designs. This should make it easier for the user to choose and compare designs. The current version of the program constructs the graph based on the nodes; this will be modified to build the graph(s) so as to minimize the number of edge overlaps in the graph, allowing easier visualization of the design.

6 SimArray, Software for Modelling Genetic and Biochemical Networks

6.1 Introduction

The vast quantities of data generated through genome projects have piqued the interest in the development of methods for modelling biological processes (Hofestadt & Thelen 1998; Covert *et al.* 2001; Wiechert 2002; van Someren *et al.* 2002; Stelling *et al.* 2002) which, ultimately, are targeted at modelling whole-cell dynamics (Tomita *et al.* 1999; Tomita 2001). Microarray time-series experiments are one of the preferred sources of data (Schulze & Downward 2001) as they can simultaneously provide a measurement of thousands of gene expression levels. Several studies have used gene expression data to infer the underlying genetic network of the system (DeRisi *et al.* 1997; Voit & Radivoyevitch 2000; Ronen *et al.* 2002; see chapter 7 for inference of biochemical systems from time-series data using Evolutionary Computation).

Alongside the methods, several simulation tools have been developed to aid modellers and experimentalists with the numerical simulation of mathematical models. Pettinen *et al.* (2005) give an overview and compare the benefits and drawbacks of the most widely used tools. In common, all these tools use models of differential equations to generate the simulations.

None of these tools allow for a straightforward visualization of the dynamics of gene expression over time resulting from these simulations as microarray data. Since this source of data is widely used in systems studies it is convenient to have a tool that allows simulation studies with an emphasis on gene expression. In this chapter method and code for a simulation tool – SimArray – was developed to allow modellers to graphically build models, perform simulations and easily visualize and treat results as a spotted microarray experiment.

SimArray is a pathway/modelling tool geared towards microarray experiments. An additional feature included in the program generates simulated time-series microarray datasets from the model with different user-defined noise levels. This is a secondary feature but still interesting as the simulated data can be useful as a first approach to compare different microarray data analysis methods (Slonim 2002; Pan 2002) or different analysis tools (Liu *et al.* 2004) using "well-behaved" data. It is important to mention that no attempt was made to simulate the dynamics of hybridization (Southern *et al.* 1999) or generate images, the simulations only add foreground and background noise to the deterministic equations. Several studies have addressed the simulation of hybridization dynamics (Bailey & Moore 1999; Balagurunathan *et al.* 2002; Wierling *et al.* 2002); these methods are generally more relevant to compare different image analysis techniques (Yang *et al.* 2002).

The main design issues and functionality considerations implemented in SimArray are:

1. An intuitive user friendly graphical user interface (GUI).
2. A design tool that permits graphical construction of a model.
3. Automatic conversion of graphs into systems of differential equations.
4. Flexible design of simulations and external stimuli.
5. Calculation of steady-states and basic stability and sensitivity analysis capability.
6. Parameter estimation of models (model fitting).

7. Flexible graphic and numeric outputs.
8. Generation of simulated microarray data.

In summary, SimArray allows the user to graphically build and modify biochemical systems as directed graphs which are then converted into S-systems of differential equations (Voit 2000). These equations can be used in simulation studies under different user-defined stimuli. The results of these simulations are presented as spotted microarray experiments allowing for easy visualization of only the 'spotted' products of the system as well as comparing how different treatments on the red and green channel affect the dynamics of these components. Results of simulations can be used to generate simulated microarray datasets. Further, steady-states and basic stability/sensitivity reports on the model are provided. A function for parameter estimation is available which uses Differential Evolution (Storn & Price 1997; see chapter 3) to fit a model to either time-series measurements or steady-state values.

The remainder of the chapter is organized as follows. In section 6.2 directed graphs are briefly discussed. Section 6.3 is a general overview of S-systems. Differential Evolution was previously discussed in chapter 3 and the reader is referred to that chapter for details of the algorithm. Section 6.4 is an overview of SimArray. In section 6.5 two examples are used to illustrate modelling and parameter estimation in SimArray. We draw our conclusions and suggest future work extensions for SimArray in section 6.6.

6.2 Directed Graphs

Graphs are an intuitive and straightforward way of representing networks. A graph is defined as a tuple (N,E) where E is a set of edges (connections) between the set N of nodes (elements). A graph can be directed, meaning that the set of edges encompasses information about the direction of flow between the nodes. An undirected graph does not define the flow direction. Graphs can be extended to include information such as activation/inhibition and regulatory interactions (De Jong 2002). The Kyoto Encyclopedia of Genes and Genomics (KEGG) is a leading example of graphs applied to chemical and genetic networks (Kanehisa & Goto 2000; Kanehisa *et al.* 2002). Graphs are essentially static representations of a dynamic system. They are particularly important to identify connections between regulatory systems, redundancy, missing interactions and overall complexity, among others (De Jong 2002). Directed graphs are static but are an important steppingstone to construct dynamic models (systems of differential equations, Boolean networks, among others); with this final goal there are three steps to construct a directed graph: define the components, define the processes and graphically arrange the components and processes.

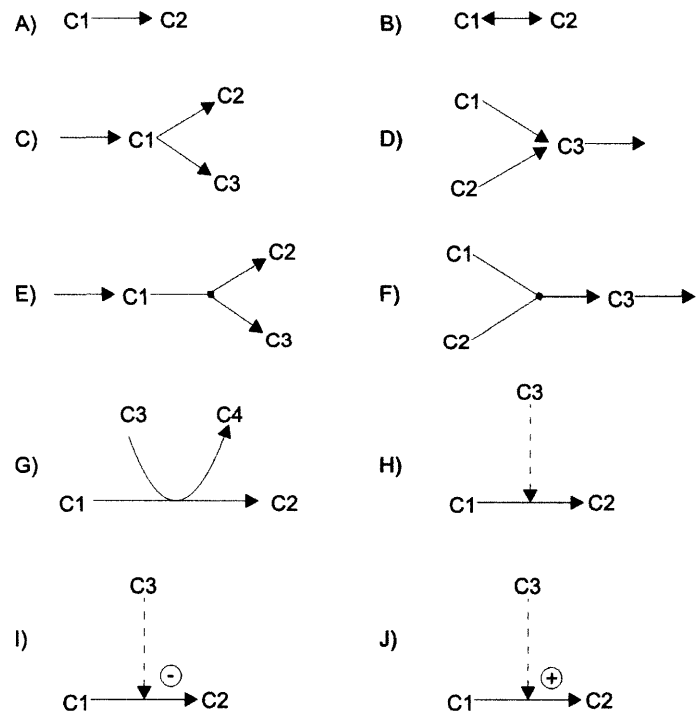
The components are a list of all the elements that affect the dynamics of the system. These are dependent variables, independent variables and parameters. The dependent variables are the components of the system whose values change over time and are affected by the system, as for example, metabolites. The independent variables are the components whose values do not change over time and are not affected by the system, such as, in some situations, enzymes. Usually dependent variables change their values in response to other dependent and independent variables; independent variables are not affected by other variables. The term independent variable is used here from a biochemical point of view and differs from the mathematical definition in which, in a biochemical reaction, *time* would be an independent variable. To avoid confusion time is simply referred to as time, even though it is an independent variable. The parameters are constant numerical values that quantify a

specific property of the system, for example pH or temperature. The distinction between these three classes is not always trivial and in different systems components can have different roles. The list of components is the critical step from which to build the dynamic model; the main components must be included but the model still should be mathematically tractable. There are no formal rules for assembling this list, but as a guideline the components should maximize the number of internal nodes (dependent variables) and minimize the number of external nodes (external variables) (Voit 2000).

The processes are a list of all the components that affect other components. That is, for each dependent variable, a list of the dependent and independent variables interacting with it. More formally it can be defined as a set of all the edges that connect the nodes and the edges that connect to other edges. Further, the direction of flow of material between the nodes has to be defined as well as the flow modulators (nodes that connect to edges). This is also a complicated step as it demands in depth knowledge of the system.

There is a vast literature available on the optimization and layout of graphs (Batista *et al.* 1994; Batista *et al.* 1999), even using evolutionary algorithms (Utech *et al.* 1998). But for most applications of modelling biochemical systems it simply consists of distributing the elements (nodes) of the system and the use of an appropriate graphical symbol to establish the links (edges) between the components and the direction of flow and modulators. There are several graphical symbols used to represent the different reaction types (figure 6.1), the terminology used is based on biochemical reactions which are more relevant in this context but the same principles apply to different networks.

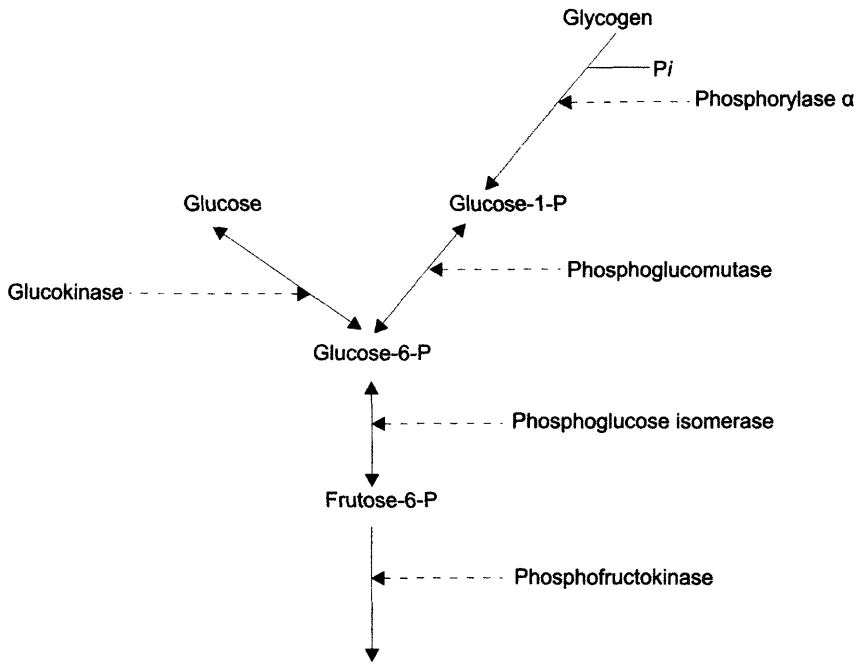
Figure 6.1 Graphical symbols used to represent reactions in a directed graph. A) irreversible reaction; B) reversible reaction; C) independent divergence branch point; D) independent convergence branch point; E) dependent divergence branch point; F) dependent convergence branch point; G) co-enzyme activation; H) modulation; I) inhibition; J) activation. Adapted from Voit (2000:26).



From a well defined directed graph (figure 6.2), a dynamic model can be constructed. There is a plethora of methods available (see section 2.3.1) with differential equations being the most extensively

used. In the next section S-systems, a form of differential equations which are used in SimArray, are reviewed.

Figure 6.2 An example of a directed graph – glycolysis pathway, modified from Voit (2000:29).



6.3 S-Systems

S-systems are canonical models of the Biochemical Systems Theory framework (Savageau 1969a; Savageau 1969b) which have been widely used in modeling biochemical systems (Voit 2000). The *S* in S-systems refers to their suitability to represent synergistic and saturated phenomena which are important characteristics of biochemical and biological systems. S-systems are a type of power-law formalism that uses nonlinear differential equations in which the right-hand sides of the equations consist of power-law functions. This formalism allows the construction of systems of differential equations suitable for the representation of virtually any differentiable nonlinear function (Savageau 1996; Voit 1991; Voit 2000). An S-system with n dependent and m independent variables consists of a production and degradation term and always takes the general form

$$\dot{X}_i = \alpha_i \prod_{j=1}^{n+m} X_j^{g_{ij}} - \beta_i \prod_{j=1}^{n+m} X_j^{h_{ij}} \quad \text{for } i = 1, 2, \ldots, n. \tag{6.1}$$

Where α_i and β_i respectively indicate the production and degradation rate constants for X_i (dependent variable) and both are ≥ 0. The indexes g_{ij} and h_{ij} are the production and degradation kinetic orders of the elements X_j (dependent and independent variables that affect the expression of X_i). The kinetic orders have activating effects of X_j on X_i if the values are positive and inhibitory effects if the values are negative (a value of zero results in X_j having no effect on X_i).

6.3.1 Characteristics of S-Systems

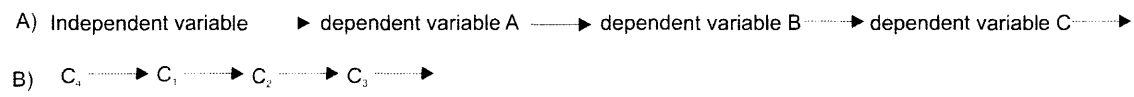
As with any type of model, S-systems have advantages and disadvantages. The main aspects of S-systems are:

1. Straightforward translation. The translation of a directed graph into S-systems can be easily automatized since the equations always have the same structure.

2. Scalability. Even systems with many components interacting are still reasonably tractable since it is a simple matter of adding a new component and its kinetic order to the production or degradation term.
3. Well defined parameters. In principle parameter values can be estimated from experimental measurements; there are no parameters that are mathematically necessary but have no biological meaning.
4. Mathematical tractability. Mathematical and numerical analysis of even complex S-systems is possible and often simple.
5. Difficult to parameterize. Even though all parameters relate to a biological value, numerically they are unrelated; thus it can be hard to find an adequate set of parameters even if biological measurements are available.

To illustrate the translation of a graph into S-systems, consider the simple linear pathway in figure 6.3. There are three dependent variables (C1, C2 and C3) and one independent variable (C4); thus there will be three equations in the system (one for each independent variable) It is convenient to change the names of variables (fig. 6.3A) to an indexed notation listing initially the dependent variables followed by the independent variables (fig 6.3B). Table 6.1 evidences the importance of the indexed notation (Voit 2000).

Figure 6.3 A simple linear pathway. A) graph with names of variables. B) graph with indexed notation; dependent variables are listed first and then the independent variables.



Each equation consists of a production and a degradation term with respectively α and β rate constants, all the variables that affect production and degradation of the component and their respective kinetic orders. The graphic symbols (fig. 6.1 determine the direction of flow of material and define which components affect the production or degradation of the dependent variables). With this information it is simple to construct table 6.1.

Table 6.1 Table of production and degradation constants, variables and kinetic orders for pathway of figure 6.3B.

S-system equation	production constant	variables	kinetic orders	degradation constant	variables	kinetic orders
C_1	α_1	C_4	g_{14}	β_1	C_1	h_{11}
C_2	α_2	C_1	g_{21}	β_2	C_2	h_{22}
C_3	α_3	C_2	g_{32}	β_3	C_3	h_{33}

For each equation there is a single production (α) and degradation constant (β). The number of variables depends on the number of components affecting the production or degradation of the considered element. The indexes of the kinetic orders can easily be obtained from the index of α or β and their respective variables. So, the procedure to translate a graph into S-systems is, for each independent variable (C_i), to include a rate constant α_i and all the variables that contribute to the production of C_i raised to a power (kinetic order) minus the β-term that is constructed in the same manner. The S-system of equations for the linear pathway is:

$$C_1 = \alpha_1 C_4^{g_{14}} - \beta_1 C_1^{h_{11}}$$
$$C_2 = \alpha_2 C_1^{g_{21}} - \beta_2 C_2^{h_{22}} \quad (6.2)$$
$$C_3 = \alpha_3 C_2^{g_{32}} - \beta_3 C_{31}^{h_{33}}$$

Note that each of these has just one production and one degradation term, as each of C_1 – C_4 have one arrow in and one arrow out. Even if the numeric values are unknown the equations themselves can still be formulated. This is clearly advantageous for automation of equation formulation from a directed graph. For an in-depth overview of construction of equations from graphs see Voit (2000:76-94).

Another property of S-systems which falls under scalability (item 2 above) is the telescopic property (Voit 2000:57). If a single variable of a higher level system represents an entire lower level system this variable can be substituted by the lower level system and still preserve the structure. To illustrate this property, with a linear model, if a variable is replaced by a lower level system with non-linear terms the structure usually breaks down. This is an important property for system studies as phenomena can be studied at different levels of resolution and higher level models can be easily connected to lower level models.

Steady-states are relevant for biological studies since most systems operate close to them. One of the main advantages of S-systems is the ease with which steady-states and other aspects of the system at or close to a steady-state can be analyzed. Algebraically, the set of steady-states can be found by equaling the left-hand side to zero (Voit 2000:193-217)

$$0 = \alpha_i \prod_{j=1}^{n+m} X_j^{g_{ij}} - \beta_i \prod_{j=1}^{n+m} X_j^{h_{ij}} \quad (6.3)$$

Since the inputs and outputs are balanced

$$\alpha_i \prod_{j=1}^{n+m} X_j^{g_{ij}} = \beta_i \prod_{j=1}^{n+m} X_j^{h_{ij}} \quad (6.4)$$

Assuming that none of the rate constants and variables in 6.4 is zero the logarithm of both sides can be taken to obtain

$$\ln \alpha_i + \sum_{j=1}^{n+m} g_{ij} \ln X_j = \ln \beta_i + \sum_{j=1}^{n+m} h_{ij} \ln X_j \quad (6.5)$$

Defining $y_i = \ln X_i$ and moving all terms with y_i to the left-hand side and the other terms to the right-hand side yields

$$\sum_{j=1}^{n+m} g_{ij} y_i - \sum_{j=1}^{n+m} h_{ij} y_i = \ln \beta_i - \ln \alpha_i \quad (6.6)$$

Renaming $a_{ij} = g_{ij} - h_{ij}$ and $b_i = \ln\beta_i - \ln\alpha_i$ for all i and all j an s-system with n dependent variables and m independent variables has a steady state defined by a system of linear equations of the form

$$\begin{aligned}
&a_{11} y_1 + a_{12} y_2 + \ldots + a_{1n} y_n + a_{1n+1} y_{n+1} + \ldots + a_{1n+m} y_{n+m} = b_1 \\
&a_{21} y_1 + a_{22} y_2 + \ldots + a_{2n} y_n + a_{2n+1} y_{n+1} + \ldots + a_{2n+m} y_{n+m} = b_2 \\
&a_{31} y_1 + a_{32} y_2 + \ldots + a_{3n} y_n + a_{3n+1} y_{n+1} + \ldots + a_{3n+m} y_{n+m} = b_3 \\
&\vdots \\
&a_{n1} y_1 + a_{n2} y_2 + \ldots + a_{nn} y_n + a_{nn+1} y_{n+1} + \ldots + a_{nn+m} y_{n+m} = b_n
\end{aligned} \quad (6.7)$$

Or, in matrix notation

$$A.\vec{y} = \vec{b} \tag{6.8}$$

Where the matrix *A* are the coefficients $a_{ij}=g_{ij}-h_{ij}$ and $\vec{y}$ is a row vector of the *n+m* components and $\vec{b}$ is a row vector with the solution coefficients $b_i=\ln(\beta i/\alpha i)$. Rearranging equation 6.7 the dependent variables can be separated from the independent variables such that

$$A_D.\vec{y}_D = \vec{b} - A_I.\vec{y}_I \tag{6.9}$$

Where *D* refers to the dependent variables and *I* refers to the independent variables. From 6.9 with matrix inversion the solution can be expressed as

$$\vec{y}_D = A_D^{-1}\vec{b} - A_D^{-1}A_I\vec{y}_I \tag{6.10}$$

Regular S-systems have a definite unique solution with non-zero rate constants and a non-zero determinant (det A≠0). If the determinant of A vanishes the S-system is irregular and there may be no solution or infinitely many. Irregular systems are of limited interest for modeling biological systems (Voit 2000). For regular systems the local stability and oscillatory behavior can be characterized from the evaluation of the eigenvalues of the linear systems. If all real parts are negative the steady-states of the system are locally stable. The numerical value of the real part refers to the relative time scale of the process. Small values relate to slow processes while high values refer to fast processes. A non-zero imaginary part of the eigenvalue is an indicator that the system will probably exhibit oscillations. From these simple concepts more in-depth analyses can be carried out; for example measurement of the robustness of biochemical systems (Chen *et al.* 2005).

S-systems are known to be difficult to parameterize with various different approaches having been devised (Voit 2000:143-192). Sands and Voit (1996) use the difference between the two flux flows (α and β-terms) to estimate parameters. Here we shall limit the discussion to parameterization from steady-states and dynamic data. The rigid structure of the equations makes them well-suited for parameterization through Evolutionary Computation. Tominaga and Okamoto (1999) used genetic algorithms for parameter estimation. Hybrid Differential Evolution in conjunction with local search is used by Tsai and Wang (2005) to parameterize systems to fit time-course data. Kimura *et al.* (2005) proposed using a method based on the problem decomposition strategy and a cooperative co-evolutionary algorithm to infer S-systems models of large-scale genetic networks.

S-systems have been used in several interesting studies. Vera *et al.* (2003) use a linear programming approach to simultaneously optimize various metabolic responses of biochemical pathways. Within inference of regulatory networks from gene expression data, the work of Thomas *et al.* (2004) and Kimura *et al.* (2005) are good examples. GeneNetwork is a tool for reverse engineering network architecture from time-series data as a Boolean network, a linear model, an S-system or a Bayesian network (Wu *et al.* 2004).

6.4 SimArray

SimArray was written using Microsoft's C#.Net and was designed to allow modelling of genetic and biochemical networks as S-systems of differential equations. Models are graphically constructed as directed graphs which are then converted into the S-systems. The program provides support for

simulations under different user-defined stimuli and for calculation of steady-states and stability/sensitivity analyses. A Differential Evolution algorithm is included for parameter estimation from either steady-state or time-course data. Results of simulations can be used to generate simulated microarray datasets. The GUI is geared towards microarrays but virtually any S-system model can be constructed. The main features of the program are:

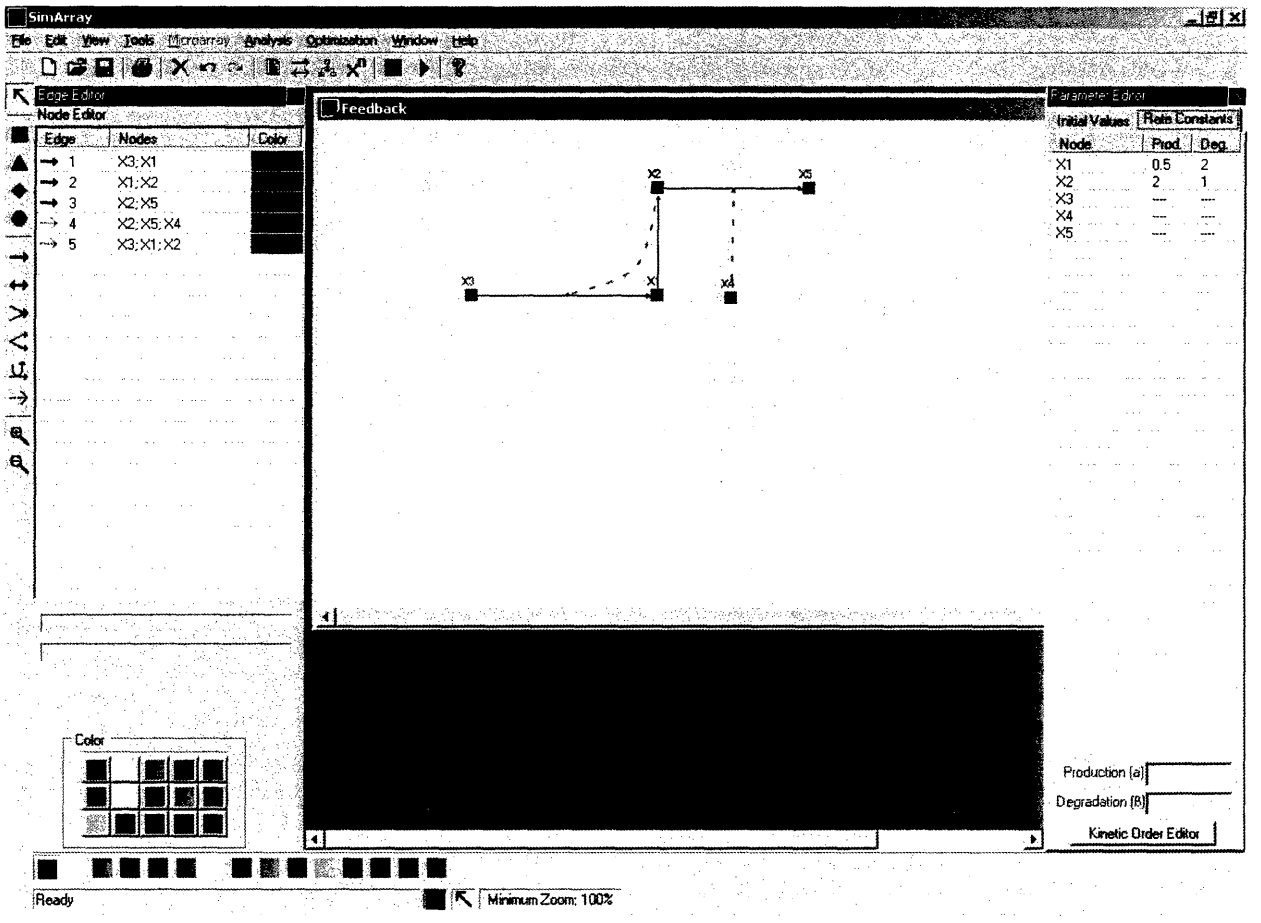
1. Graphical design of biological pathways.
2. Dynamic modelling with user-defined stimuli.
3. Calculation of steady-states and basic stability/sensitivity analysis.
4. Parameter estimation from steady-states values or time-course data.
5. Simulation of gene expression data.

Each of these features is discussed with some detail in the following sections.

6.4.1 Graphical Design of Networks

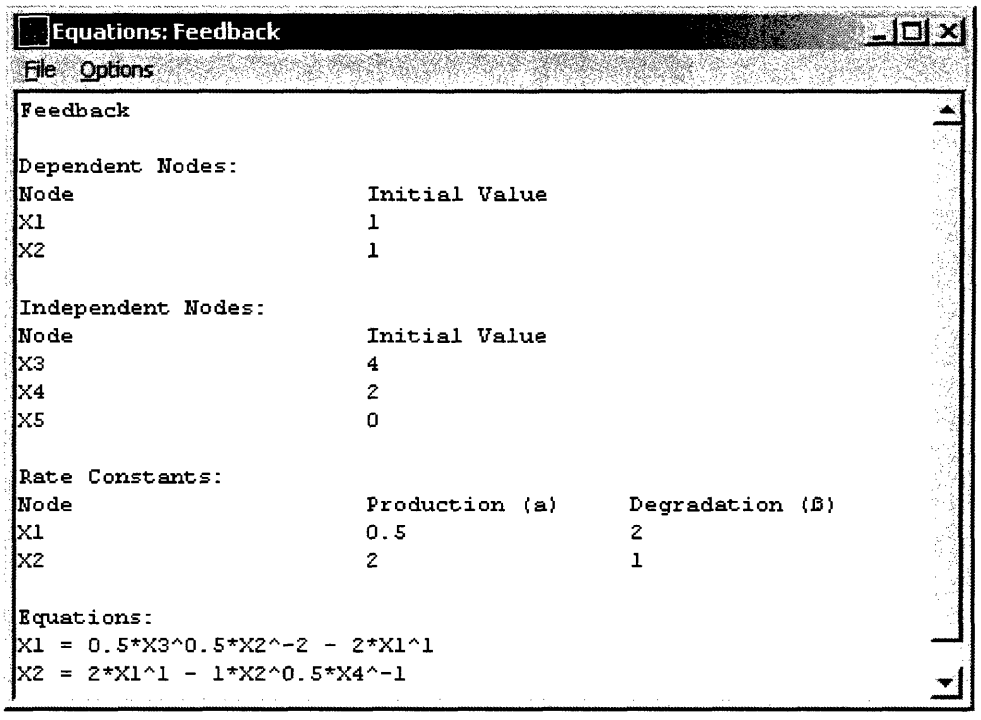
The graphic design tool of SimArray allows for easy 'point-and-click' drawing of two-dimensional networks. Figure 6.4 is a screenshot of SimArray with a simple feedback pathway (Voit 2000). There are four different node representations and six different flow/modulation arrows (fig. 6.1). Colours of nodes and edges are user-defined.

Figure 6.4 Screenshot of SimArray with a simple feedback pathway. On the left-hand side is graphic toolbar with a pointer to select and modify the layout of nodes, four node representations, six flow and modulator arrows and zoom tools. The node and edge editor on the left-hand side allows provides graph editing capabilities. The parameter editor on the right-hand side provides editing capabilities of initial values, variable settings, rate constants and kinetic orders.



The workflow to construct a pathway and its respective S-systems is as follows: select from the *graphic toolbar* (fig. 6.4, left-hand side) a node representation (square, triangle, diamond or circle) and add as many nodes as desired by clicking on the work canvas. A popup screen allows defining the name of the node, initial value, if it is a dependent or independent variable and if it is spotted on the microarray. Select a flow arrow or modulator and click on the nodes in the canvas to define a new edge. Use the *parameter editor* (6.4, right-hand side) to define the values of α and β constants for each dependent variable. Use the popup *kinetic order editor* to specify the values of each component affecting the α and β-terms of the dependant variables. These components are automatically defined from the edges. All nodes, edges, settings and graph layout can be modified using the editing tools and editing panes. The S-systems equations are automatically constructed from the graph. To view the equations click on the menu item *Analysis* (fig. 6.4), sub item *Equations* (fig. 6.5).

Figure 6.5 S-systems equations for the simple feedback pathway.



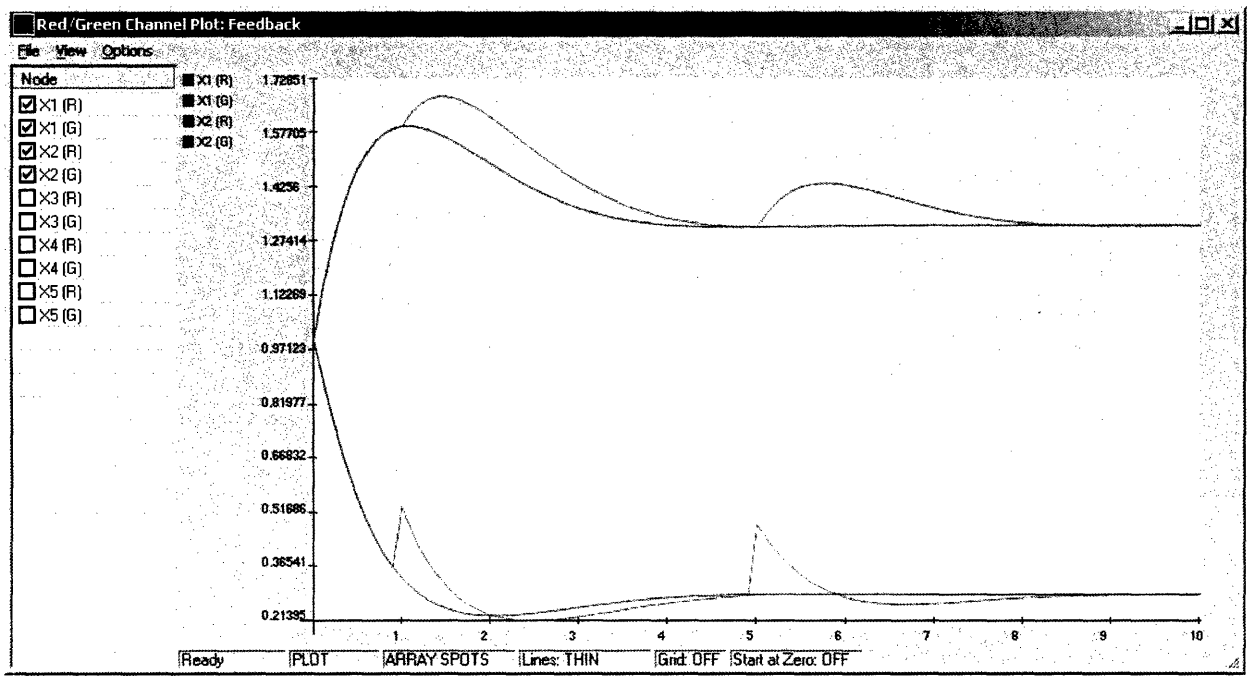
6.4.2 Dynamic Modelling of Networks

Simulations are one of the most important aspects of a modelling tool. SimArray is very flexible in setting up and visualization of simulations. Once a pathway and the numerical parameters of the S-system have been defined several simulation studies can be performed. The steps to set up a simulation involve definition of the time and report intervals; the numerical method (Euler or 4^{th} order Runge-Kutta) and the step interval. Currently only fixed step-size methods are available, this is important for the Differential Evolution (Cao *et al.* 2000; chapter 7) but is not necessary for the numerical simulations. A future version will include a modified Taylor method. The next step is to define the stimuli to the system. Any component – dependent or independent variable – of the system can be used as a stimulus. Stimuli are set by defining the component, the intensity (concentration) and the time points of the stimuli. Any number of components, time intervals and different concentrations can be used. Results of simulations are by default shown as spotted arrays with a red and green channel (fig. 6.6). When setting up a stimulus the channel to which it applies must be defined. Thus, different types of stimuli can be defined for each channel. Results of simulations can be viewed graphically or numerically. Several options are available for selecting which components are displayed, including visualization of single channels (red or green) or both channels together. Plots can be exported as bitmap files and numeric results as plain text files.

Figure 6.6 illustrates the results of a simulation using the simple feedback pathway from figures 6.4 and 6.5. This pathway consists of five nodes with three independent variables (*X3*, *X4* and *X5* in fig. 6.5; shown in red in fig. 6.4) and two dependent variables (*X1* and *X2* in fig. 6.5; shown in black in fig. 6.4). The latter were defined as being spotted on the array. The initial values of all components, the α and β constants of the independent variables *X1* and *X2* and the S-systems equations of the pathway can be seen in figure 6.5. The simulation was run over a ten minute interval, starting at time zero using the Euler method with a step of 0.001. The report interval was 0.1. In the red channel two bolus of 0.2 of *X1* at one

and five minutes were added. No stimuli were applied to the green channel. The effect of the stimuli on the dynamics of the system can be easily compared to the model without stimuli using the two channel plot.

Figure 6.6 Red and green channel plot for the feedback pathway. The simulation was run over a 10 minute interval, solved using Euler with a 0.001 step size. Two bolus of 0.2 of *X1* were added at one and five minutes on the red channel.

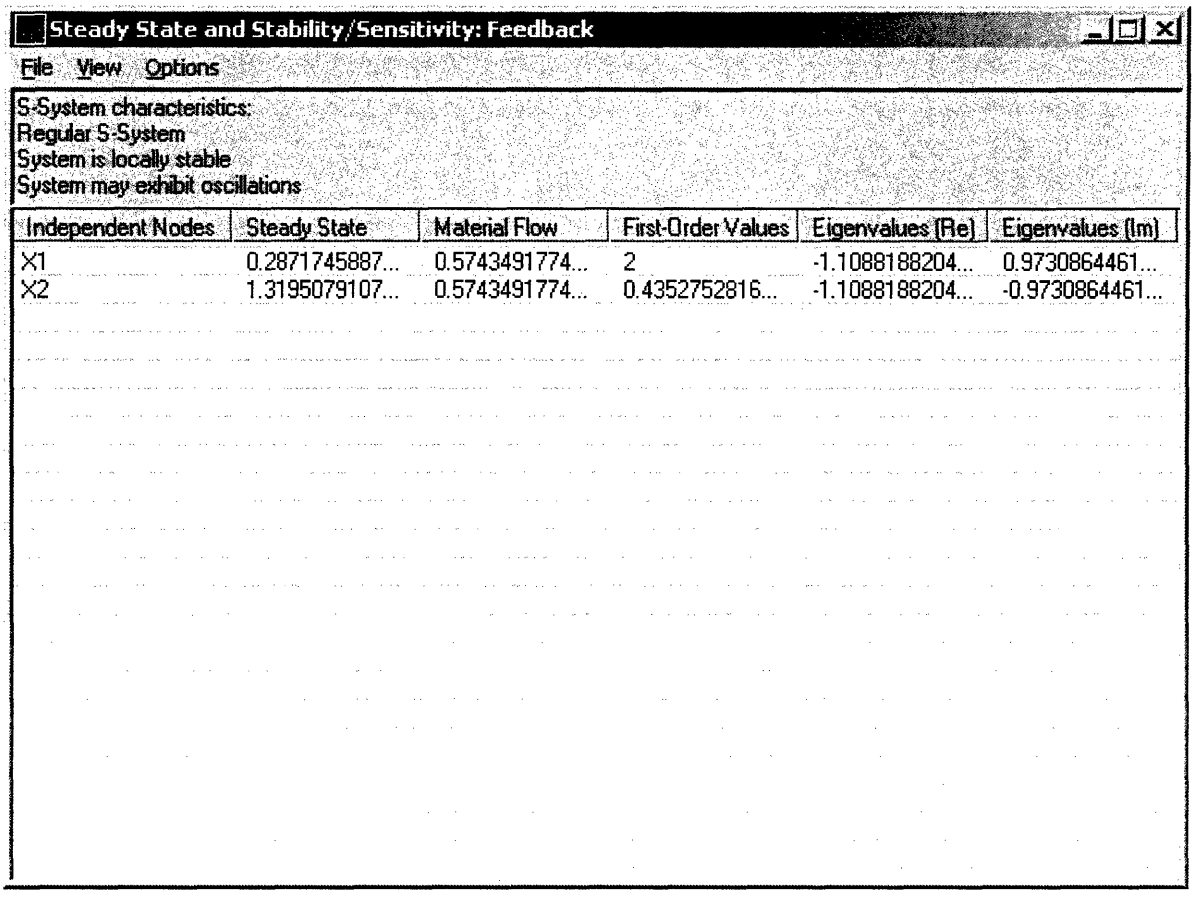


6.4.3 Steady-States and Stability/Sensitivity Analyses

Some basic analysis capabilities are included in SimArray. More extensive analyses can be performed externally using the text file numeric results of simulations. Figure 6.7 shows the *Steady State and Stability/Sensitivity* window for the feedback pathway. A summary of the S-system is shown in the upper pane – if the system is regular or irregular; locally stable or unstable and its oscillatory behaviour. For each dependent node the values of the steady-state, material flow, first-order values and the values of the real and imaginary parts of the eigenvalues are given. How to calculate the steady-states of S-systems was discussed in 6.3.1. To solve the matrixes LU decomposition with partial pivoting is employed. The results are further refined with an iteration method (Stewart 1998).

The material flow is the quantity of material that is flowing through each dependent variable in the steady-state. Since in the steady state the α-term equals the β-term; either can be used to calculate the flow of material. The apparent first-order rate constants (first-order values in fig. 6.7) are defined as the relative fluxes in the steady-state. That is, material flow divided by the steady-state value. First-order values provide information on the turnover time of products in the dependent variables. Small values indicate long turnover times and bigger values longer turnover times. The importance of eigenvalues was discussed in 6.3.1. To determine the eigenvalues SimArray uses a modified QR algorithm (Stewart 1998) adapted from the Math.NET open source library (http://nmath.sourceforge.net).

Figure 6.7 ***Steady State and Stability/Sensitivity*** **window for the feedback pathway (steady-states view). The steady-states of all dependent variables are shown. The material flow is the same for** ***X1*** **and** ***X2*** **following from the precursor-product relationship. The turnover time of** ***X1*** **is over four times the turnover time of** ***X2*** **(first-order values). The system is locally stable since the real parts of all the eigenvalues are negative. Non-zero imaginary parts of eigenvalues indicate possibility of oscillations.**



Besides the steady-states and the stability information, three types of sensitivities are available: the log-gain of the steady-states of the dependent variables in relation to a change in the independent variables and the sensitivities of the steady-states of the dependent variables in relation to changes in the rate constants and the kinetic orders. Figure 6.8 shows the sensitivity values window for the feedback pathway. All sensitivities are shown in relation to the steady-state and under the assumption of independence, which is the most important scenario. Sensitivities, in theory, only apply to very small changes but in practice they generally hold true for larger changes (this depends on the non-linearity of the system). They can be interpreted as the percentage of change in a dependent variable in relation to a one percent change to an independent variable, a rate constant or a kinetic order.

Figure 6.8 ***Steady State and Stability/Sensitivity*** **window for the feedback pathway (sensitivities view). The sensitivities of the dependent variables in relation to changes in the independent variables, rate constants and kinetic orders are shown. Sensitivities can be seen as a measurement of the percentage of change in a dependent variable that is caused by a 1% change to the independent variables and the parameters of the system.**

Steady State and Stability/Sensitivity: Feedback

File View Options

S-System characteristics:
Regular S-System
System is locally stable
System may exhibit oscillations

Variables Sensitivities	X1	X2
X3	0.1	0.2
X4	-0.8	0.4
X5	0	0
alpha (X1)	0.2	0.4
beta (X1)	-0.2	-0.4
alpha (X2)	-0.8	0.4
beta (X2)	0.8	-0.4
X3 - Prod. (X1)	0.1387255711...	0.2776435901...
X2 - Prod. (X1)	-0.1099628584...	-0.2198047986...
X1 - Deg. (X1)	0.2493452646...	0.4993122599...
X1 - Prod. (X2)	0.9951290618...	-0.4938815037...
X2 - Deg. (X2)	0.1107434598...	-0.0553257819...
X4 - Deg. (X2)	-0.5529831326...	0.2776435901...

The logarithmic gain (log gain) of a dependent variable in relation to an independent variable is given as the partial derivative in equation 6.11 and describes the logarithmic change in the dependent variable X_i caused by the logarithmic change in the independent variable X_j.

$$L(X_i, X_j) = \frac{\partial x_i}{\partial x_j} \tag{6.11}$$

Exploiting matrix algebra, all gains can easily be computed using equation 6.12, where the indexes *D* and *I* refer to the dependent and independent variables. For an in-depth discussion see Savageau (1969b).

$$L(X_D, X_I) = -A_D^{-1} A_I \tag{6.12}$$

Sensitivities in relation to changes in the rate constants can also be inferred from the partial derivatives defined in relation to α (6.13) and β (6.14) as:

$$S(X_i, \alpha_j) = \frac{\partial \ln X_i}{\partial \ln \alpha_j} \tag{6.13}$$

$$S(X_i, \beta_j) = \frac{\partial \ln X_i}{\partial \ln \beta_j} \tag{6.14}$$

The matrix of sensitivities S(X,β) can be obtained from the relation (Savageau 1996):

$$S(X_D, \beta) = A_D^{-1} \tag{6.15}$$

And since

$$S(X,\alpha) = -S(X,\beta) \tag{6.16}$$

The matrix *S(X,α)* is easily obtained as the negative of the *S(X,β)* sensitivities.

The sensitivities in relation to the kinetic orders $S(X_k,g_{ij})$ and $S(X_k,h_{ij})$ are again obtained through the calculation of partial derivatives (eqs. 6.17 and 6.18).

$$S(X_k, g_{ij}) = \frac{\partial \ln X_k}{\partial \ln g_{ij}} \tag{6.17}$$

$$S(X_k, h_{ij}) = \frac{\partial \ln X_k}{\partial \ln h_{ij}} \tag{6.18}$$

6.4.4 Parameter Estimation using Differential Evolution

Rate constants (α and β) and kinetic orders of the S-systems models can be estimated using Differential Evolution – DE (Storn & Price 1997; see chapter 3). To set up a DE run the user must specify the DE parameters: population size, number of generations, crossover and mutation rates, as well as which source of information is to be used to fit the model; either the initial values defined in the model or a time-series data file with measurements of the concentration of the dependent variables. The next step is to define which dependent variables (equations) are to be parameterized. Any combination of variables can be selected, the remaining variables are assumed to be correctly parameterized. This allows, for example, that only variables with time-course gene expression data are selected for parameterization. Which parameters of the S-systems will be estimated can be defined (α, β, production and degradation kinetic orders). This allows, for instance, estimation of only the rate constants of the selected variables using predefined kinetic orders. The search space can be constrained with individual upper and lower bounds for each rate constant and kinetic order.

If the data source selected is the initial values it is assumed that these are the steady-states of the components. Of course, if the initial values do not represent a steady state, the prevailing unstable state will still be targeted by the DE algorithm. The optimization algorithm builds the linear equations from the S-systems and searches for a parameter set that fits to these initial values. The fitness function is a simple sum of squares of errors (SSE) between the predicted steady-states (x_i) and the observed (y_i) steady-states (eq. 6.19). To use this approach it is important to have a reasonable estimate of the range of possible values for the parameters to set up constraints. Since there is a single data value for each variable and several parameters the DE can easily find a parameter set that is a perfect fit to the steady-state but these frequently are biologically unrealistic. This is not a robust approach, but nevertheless important since steady-state measurements of components are readily available which is seldom the case for time-series data. It can be used to obtain a 'first-try' parameterization that can be further evaluated by the modeller.

$$f = -1 * \sum_i^n (x_i - y_i)^2 \tag{6.19}$$

The second approach using time-course data is robust. The DE numerically solves the S-systems and optimizes a parameter set that fits the data for the selected variables. The fitness function used is

$$if(r>0) \qquad f=-1*(\sum_i^n \frac{\sum_j^m (x_{ij}-y_{ij})^2}{r(x_i,y_i)}+\sum_i^n\sum_j^m(\mathrm{cov}(x_i,x_j)-\mathrm{cov}(y_i,y_j))^2) \tag{6.20}$$

$$elseif(r\le 0) \quad f=-1*(\sum_i^n\sum_j^m(x_{ij}-y_{ij})^2*2+\sum_i^n\sum_j^m(\mathrm{cov}(x_i,x_j)-\mathrm{cov}(y_i,y_j))^2)$$

Where *r* is the correlation between the predicted and observed datasets for each dependent variable and *cov* is the covariance matrix. This function rewards both good correlation between predicted and observed values, and low sums of squared deviations; the importance of the latter increases for lower correlations. If the correlation for a given set is zero or negative, fitness is penalized by doubling the value of the SSE for that set. An example of parameterization is shown in section 6.5.1.

6.4.5 Simulation of Gene Expression Data

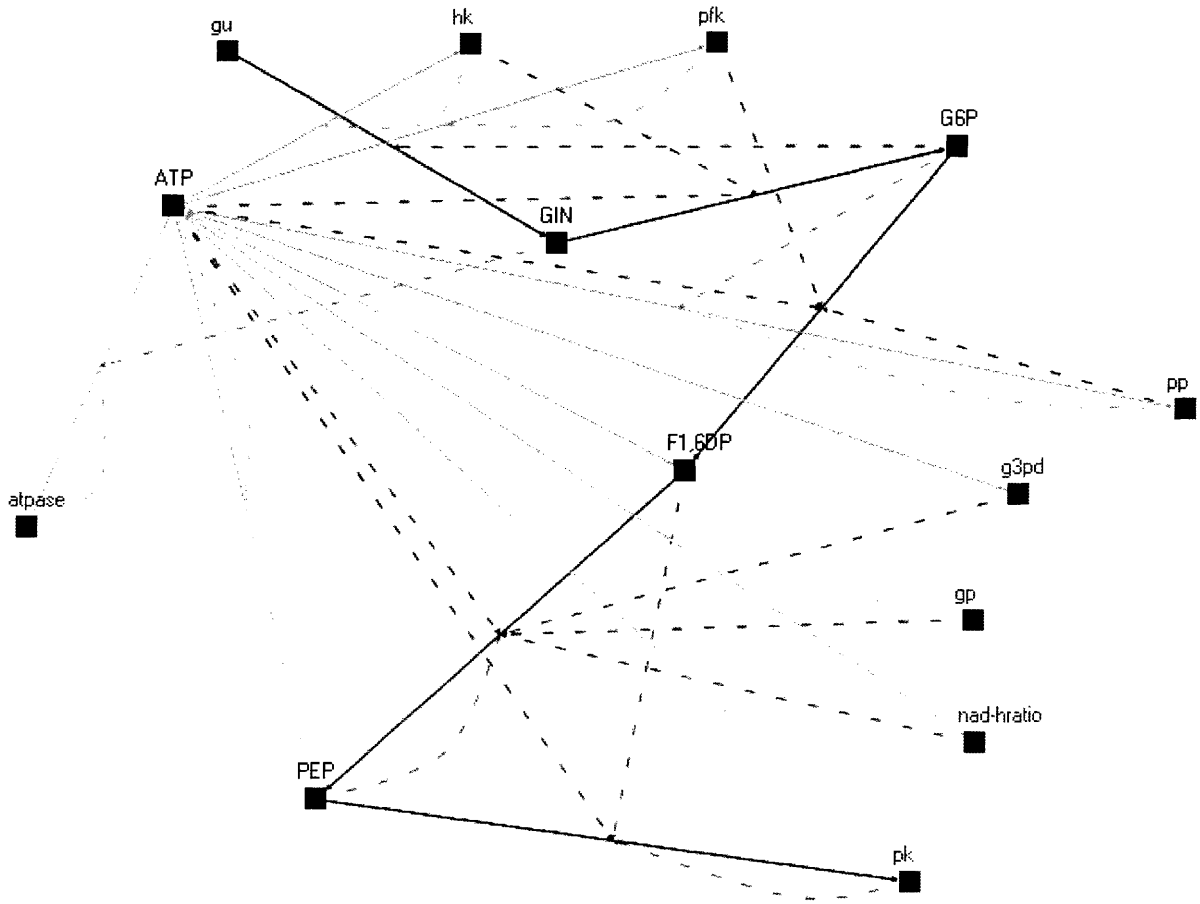
Simulated time-course microarray datasets can be generated from the results of model simulations. Output consists of a text file for each selected time point, with the concentration values in each channel of the dependent variables marked *'as spotted on array'* and user-defined foreground and background noise. Parameter settings include the total time interval of the simulation from which to sample data and the interval between sampling. The number of 'slides' is automatically defined from these two parameters. Noise is an important component in microarray studies. SimArray allows three options: no noise, random noise and Gaussian noise. Noise can be applied to the foreground ('spots'), background or both. Further parameters include minimum and maximum levels of noise and its standard deviation.

6.5 An Example of Model Building and Parameter Estimation Using SimArray

To illustrate the capabilities of SimArray, the fermentation pathway to produce ethanol in yeast (*Saccharomyces cerevisae*) is presented. This is an interesting model since the pathway has been broadly studied and has well known metabolites and kinetic processes as well as being important for industrial purposes. More importantly, an extensive discussion of modelling the pathway using S-systems and a full parameter set were presented by Voit (2000:260-290) and implemented in the software *Power Law Analysis and Simulations* (PLAS) written by Ferreira (http://www.dqb.fc.ul.pt/docentes/aferreira/plasdownm.html), which makes it ideal for testing purposes.

Briefly, the pathway describes the production of ethanol, glycerol, glycogen and trehalose from glucose. The graph for the model of yeast fermentation in SimArray is depicted in figure 6.9. For an in depth discussion of the components of the model and its limitations see Voit (2000:260-262).

Figure 6.9 Snapshot of directed graph in SimArray of the fermentation pathway of *Saccharomyces cerevisae* adapted from Voit (2000:261).



The model consists of five dependent variables (table 6.2) and nine independent variables (table 6.3). The α and β rate constants for each of the dependent variables is presented in table 6.2.

Table 6.2 Dependent variables in the fermentation pathway of *Saccharomyces cerevisae.*

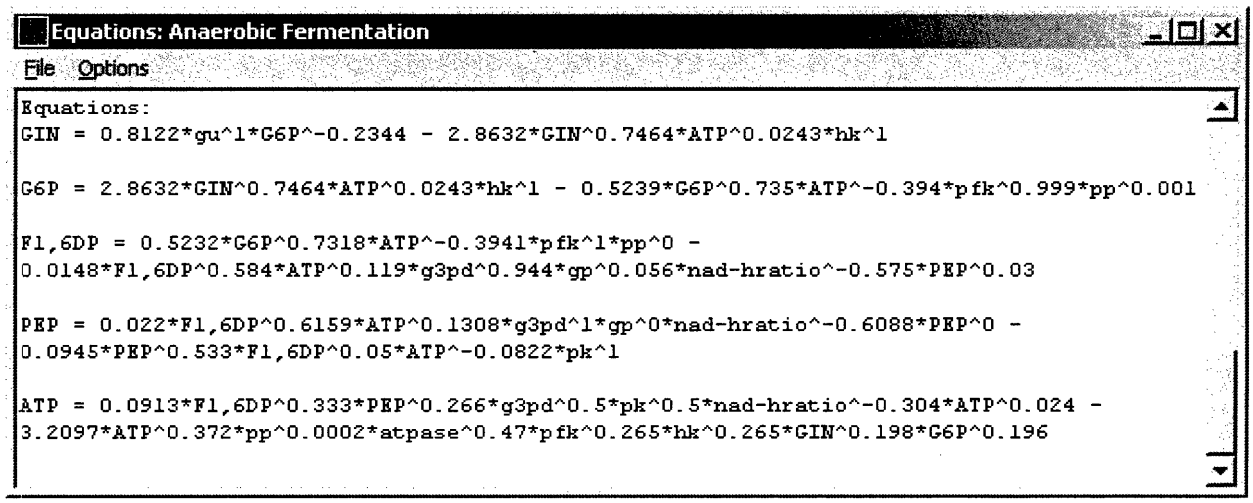
dependent variables	acronym	steady-state conc. (mM)	α rate	β rate
internal glucose	GIN	0.0345	0.8122	2.8632
glucose-6-phosphate	G6P	1.011	2.8632	0.5239
fructose-1,6-diphosphate	F1,6DP	9.144	0.5239	0.0148
phosphoenolpyruvate	PEP	0.0095	0.022	0.0945
adenosine triphosphate	ATP	1.1278	0.0913	3.2097

Table 6.3 Independent variables in the fermentation pathway of *Saccharomyces cerevisae.*

independent variable	acronym	value
glucose uptake	gu	19.7 mM min^{-1}
hexokinase	hk	68.5 mM min^{-1}
phosphofructokinase	pfk	31.7 mM min^{-1}
glyceraldehyde-3-phosphate dehydrogenase	g3pd	49.9 mM min^{-1}
pyruvate kinase	pk	3440 mM min^{-1}
polysaccharide production (glycogen+trehalose)	pp	14.31 mM min^{-1}
glycerol production	gp	203 mM min^{-1}
ATPase	atpase	25.1 mM min^{-1}
NAD+/NADH ratio	nad-hratio	0.042

The S-systems equations built from the directed graph and the kinetic orders of the components involved in the production and degradation terms are shown in figure 6.10. All parameters are from Voit (2000:278).

Figure 6.10 S-systems equations of the dependent variables in the fermentation pathway of *Saccharomyces cerevisae*.



Numeric results for the steady-states, stability and sensitivity values are the same as those obtained from PLAS with the exception of small variations in the sensitivities of the independent variables to changes in the kinetic orders. The partial derivatives (eqs. 6.17 and 6.18) used in PLAS yield approximate values; in SimArray the actual changes are calculated which are more precise but computationally expensive. Table 6.4 shows the values for the steady-states, material flow, first-order values, eigenvalues and the different sensitivities for the five independent variables of the yeast fermentation model.

Table 6.4 Steady-states and Stability/Sensitivity values for the independent variables of the *Saccharomyces cerevisae* model.

Independent Nodes	Steady State	Material Flow	First-Order Values	Eigenvalues (Re)	Eigenvalues (Im)
GIN	0.034520909	15.94800181	461.9809357	-1686.945634	0
G6P	1.014076087	15.94800181	15.72663237	-341.2459191	0
F1,6DP	9.182334902	15.93845039	1.735773151	-13.5883652	7.847460784
PEP	0.009551363	30.12879505	3154.397462	-13.5883652	-7.847460784
ATP	1.135328009	60.39711225	53.19794085	-1.833784267	0

Variables Sensitivities	GIN	G6P	F1,6DP	PEP	ATP
gu	0.9070656	1.3074685	0.9748421	1.3062934	0.6788323
hk	-1.339848874	0.000214771	-0.000195396	3.81E-06	0.000528486
pfk	0.286894854	-0.937332275	0.314441722	0.425490972	0.229315079
g3pd	0.000161255	-0.000409076	-1.623858284	0.151677633	-0.00100655
pk	-0.0177137	0.04494245	0.05051785	-1.7783473	0.1105766
pp	0.000576826	-0.001673141	-0.000642255	-0.00131272	-0.001578535
gp	0.017334845	-0.043981174	-0.050929066	-0.097316733	-0.10821155
atpase	0.145591148	-0.369387241	0.335966293	-0.006491828	-0.908842572
nad-hratio	-0.000312716	0.000793377	0.988585281	-0.09182699	0.001952063
alpha (GIN)	0.9070656	1.3074685	0.9748421	1.3062934	0.6788323
beta (GIN)	-0.9070656	-1.3074685	-0.9748421	-1.3062934	-0.6788323
alpha (G6P)	-0.5148719	1.5159548	0.7852189	1.3099575	1.1917933
beta (G6P)	0.5148719	-1.5159548	-0.7852189	-1.3099575	-1.1917933
alpha (F1,6DP)	-0.3095508	0.7853781	0.9094476	1.7377988	1.9323491
beta (F1,6DP)	0.3095508	-0.7853781	-0.9094476	-1.7377988	-1.9323491

alpha (PEP)	-0.1371705	0.3480227	-0.4079288	1.7852535	0.8562772
beta (PEP)	0.1371705	-0.3480227	0.4079288	-1.7852535	-0.8562772
alpha (ATP)	-0.3097684	0.7859303	-0.7148219	0.0138124	1.9337076
beta (ATP)	0.3097684	-0.7859303	0.7148219	-0.0138124	-1.9337076
gu - Prod. (GIN)	2.740495959	3.973996568	2.94825773	3.970355125	2.04394856
G6P - Prod. (GIN)	-0.002962812	-0.004270648	-0.003184191	-0.00426681	-0.002217326
GIN - Deg. (GIN)	2.289514426	3.316773102	2.462678238	3.313743439	1.70854445
ATP - Deg. (GIN)	-0.002797067	-0.004031741	-0.003006062	-0.004028118	-0.002093284
hk - Deg. (GIN)	-3.761447498	-5.376518179	-4.036756959	-5.371818427	-2.828537539
GIN - Prod. (G6P)	1.297014955	-3.723210337	-1.946142066	-3.225533247	-2.938895467
ATP - Prod. (G6P)	-0.001588413	0.004676964	0.002422499	0.004041416	0.003676855
hk - Prod. (G6P)	-2.152767822	6.617437234	3.374682593	5.69313172	5.166552623
G6P - Deg. (G6P)	0.00523153	-0.015401783	-0.007977956	-0.013309033	-0.01210857
ATP - Deg. (G6P)	-0.025865382	0.076195185	0.039459564	0.065837906	0.059897247
pfk - Deg. (G6P)	1.793677337	-5.099745834	-2.674826688	-4.422333133	-4.031575466
pp - Deg. (G6P)	0.001370062	-0.004033812	-0.002089413	-0.003485682	-0.003171262
G6P - Prod. (F1,6DP)	-0.003184665	0.008080442	0.009357004	0.017880395	0.019882343
ATP - Prod. (F1,6DP)	0.015367828	-0.038979955	-0.045136394	-0.086230197	-0.095879223
pfk - Prod. (F1,6DP)	-1.064202454	2.751694122	3.193263627	6.190432642	6.906892366
pp - Prod. (F1,6DP)	0	0	0	0	0
F1,6DP - Deg. (F1,6DP)	0.399513256	-1.006507808	-1.164580879	-2.21350839	-2.458245535
ATP - Deg. (F1,6DP)	0.004664736	-0.011834178	-0.013703545	-0.026183496	-0.029114368
g3pd - Deg. (F1,6DP)	1.149122782	-2.857263265	-3.301109378	-6.21293078	-6.884000345
gp - Deg. (F1,6DP)	0.092146029	-0.23340822	-0.270230841	-0.51572938	-0.573300581
nad-hratio - Deg. (F1,6DP)	0.565843846	-1.42138818	-1.644075552	-3.118013676	-3.46097404
PEP - Deg. (F1,6DP)	-0.043160463	0.109588395	0.126911518	0.242646356	0.269847755
F1,6DP - Prod. (PEP)	-0.186679358	0.475202489	-0.554140058	2.461685839	1.173251082
ATP - Prod. (PEP)	-0.002279742	0.005784291	-0.006779531	0.029675245	0.014232306
g3pd - Prod. (PEP)	-0.534903788	1.370076792	-1.58235722	7.229776338	3.404740156
gp - Prod. (PEP)	0	0	0	0	0
nad-hratio - Prod. (PEP)	-0.264381839	0.673926514	-0.784190475	3.505489888	1.666301056
PEP - Prod. (PEP)	0	0	0	0	0
PEP - Deg. (PEP)	-0.336276669	0.858285869	-0.99673156	4.481481524	2.124983902
F1,6DP - Deg. (PEP)	0.01521154	-0.038583642	0.045244107	-0.197765048	-0.094904683
ATP - Deg. (PEP)	-0.001432091	0.003633529	-0.00425881	0.018640327	0.008940194
pk - Deg. (PEP)	1.123272309	-2.794245947	3.37764635	-13.53039698	-6.735308765
F1,6DP - Prod. (ATP)	-0.227915716	0.580596265	-0.525154985	0.010174761	1.434564162
PEP - Prod. (ATP)	0.383990277	-0.967664544	0.888321753	-0.01708768	-2.364049883
g3pd - Prod. (ATP)	-0.60377044	1.548366774	-1.387764781	0.027007063	3.852785486
pk - Prod. (ATP)	-1.253336707	3.251754785	-2.868531802	0.056254631	8.191551629
nad-hratio - Prod. (ATP)	-0.298080548	0.760281303	-0.686510828	0.013311994	1.880997675
ATP - Prod. (ATP)	-0.000944025	0.002395177	-0.002178419	4.21E-05	0.005893211
ATP - Deg. (ATP)	0.014522263	-0.036835759	0.033514768	-0.000647491	-0.090606551
pp - Deg. (ATP)	0.000164856	-0.000418265	0.000380423	-7.35E-06	-0.001029098
atpase - Deg. (ATP)	0.470323618	-1.183427937	1.088658141	-0.020920138	-2.886597719
pfk - Deg. (ATP)	0.284127179	-0.717267585	0.656870443	-0.012650321	-1.755536723
hk - Deg. (ATP)	0.347577646	-0.876465523	0.803893812	-0.015470241	-2.142678395
GIN - Deg. (ATP)	-0.206375856	0.525524485	-0.47559067	0.009212127	1.297969409
G6P - Deg. (ATP)	0.000847363	-0.002149859	0.001955387	-3.78E-05	-0.005289444

6.5.1 Parameter Estimation for the Fermentation Pathway in Saccharomyces cerevisae

The yeast model was used to test the two available methods for model parameterization with DE. For the first approach only the steady-states are used to fit the model (see section 6.4.4). The run parameters were: 1,000,000 fitness calls, population size of 10; mutation rate of 0.4 and recombination rate of 0.5. Over ten repeats the best fitness was $-1.33e^{-4}$ (average fitness $-3.819e^{-4}$). In all runs the evolved parameters provided a very close fit to the numerical steady-states. But, as previously mentioned, this is not a robust approach and the evolved parameters do not reflect the original

parameters of the model nor are guaranteed to be stable. Nevertheless, all the kinetic orders had the appropriate signs (plus/minus) and considering the parameter set of the best fitness run the kinetic orders were all within a *(-1,1)* range and the highest rate constant was 0.6198 which is well within the usual parameters in S-systems (Voit 2000).

This approach is useful for an initial model parameterization if only steady-state data is available. Its efficiency improves if instead of full model parameterization a subset of the model is selected for parameterization. Over two trials of five runs with the same parameters and the DE optimizing α rate constants in the first trial and β rate constants in the second trial, the best fitness for α was $-4.02e^{-5}$ and $-1.01e^{-4}$ for β with the evolved rate constants closer to the original rates. Table 6.5 shows the best evolved values for α in the first trial and for β in the second trial.

Table 6.5 Optimization of rate constants using steady-states in the yeast model.

dependent variables	original α rate	predicted α rate	original β rate	predicted β rate
internal glucose	0.8122	0.8115	2.8632	3.0653
glucose-6-phosphate	2.8632	2.8774	0.5239	0.4897
fructose-1,6-diphosphate	0.5239	0.5280	0.0148	0.0180
phosphoenolpyruvate	0.022	0.0274	0.0945	0.0955
adenosine triphosphate	0.0913	0.0817	3.2097	3.1823

Alternatively, a subset of the equations can be selected for parameterization. Over five runs with the same parameters and the DE optimizing only the equation for *internal glucose* (equation 1 in figure 6.10) the evolved parameters of the fitted equation are reasonably close to the original parameters (table 6.6) with a fitness of $-1.28e^{-12}$. The best fitness of the five runs was $-1.62e^{-17}$ but the evolved parameter set was significantly different from the original values. This is an important consideration and once again a reminder that there is not enough information in steady-state data for a truly reliable parameterization. On the other hand, this approach is computationally inexpensive with runs taking under ten minutes on a desktop PC.

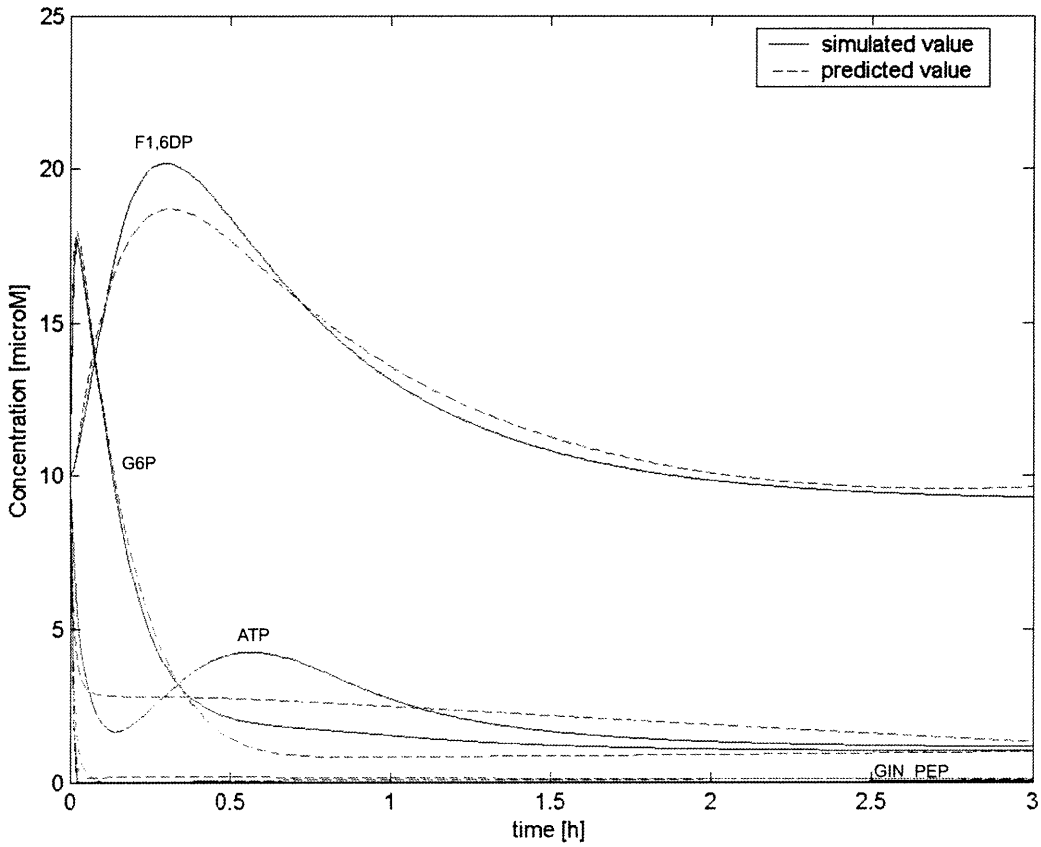
Table 6.6 Optimization of rate constants and kinetic orders for *internal glucose* using steady-states for model fitting in the yeast pathway.

internal glucose	original value	predicted value
rate constants		
alpha	0.8122	0.8122
beta	2.8632	2.8445
kinetic orders		
gu	1	0.9998
G6P	-0.2344	-0.9998
GIN	0.7464	0.7622
ATP	0.0243	0.0199
hk	1	0.9998

The second parameterization method evolves a parameter set to fit time-course data. The yeast model was used to generate a simulated dataset over a three hour period and sampled at approximately every fifteen minutes which was used as the source of data for the DE. The initial values for all dependent variables were set at 10. At three hours the model is very close to its steady-state. This approach is computationally expensive and runs can extend for over forty hours. The run parameters were: 1,000,000 fitness calls, population size of 10; mutation rate of 0.4 and recombination rate of 0.5. Over five repeats

the best fitness was -332.0350 (average fitness -446.6883). Figure 6.11 shows the fit of the best evolved parameters in relation to the original model over three hours.

Figure 6.11 Fit of the best evolved parameters in relation to the original parameters in the yeast model.



The full parameter set for the yeast model consists of 55 parameters. For such a complex model the evolved parameter set fits well to the original data and adequately reflects the dynamics of four out of the five dependent variables. All the evolved parameters are within the usual parameter values of S-systems and the model is stable. The dynamics of *ATP* were not adequately modelled with the evolved equation being essentially a linearization of the data points. This is still acceptable since the *ATP* equation is particularly complex with 15 parameters. The other four equations are a good fit to the data with a slight overshoot in *F1,6DP* and an undershoot in *G6PD*. For simpler models as the feedback example used in the previous section a virtually perfect fit was obtained (data not shown). For complex models the use of structural constraints could improve the optimization results (see 6.6).

6.6 Conclusion and Future Work

Modelling is an important component of systems studies. In this chapter we described SimArray, a tool for modelling genetic and biochemical networks. The main points of SimArray are:

1. A graphic design tool which reduces construction of networks to a simple 'point and click' process.

2. Automatic inference of differential equations from the network graph. This eliminates the need of manually building the equations from the graph which is a time consuming task and could also prove to be appealing to users less mathematically disposed.
3. Models are constructed as S-systems which always use the same rigid structure, have a solid theoretical framework, are capable of representing almost any system and have been widely used in the analysis of biochemical systems (Voit 2000).
4. The most widely adopted tool for working with S-systems is PLAS which is mathematically oriented. SimArray provides a graphical, more intuitive alternative to working with S-systems and includes an Evolutionary Computation algorithm for parameter optimization.
5. Due to the current importance of microarray data in systems studies, the 'theme' adopted revolves around the microarray framework. Simulated time-course microarray datasets with different noise levels can be generated from model simulations.

Parameterization of S-systems is a complex task (Voit 2000). SimArray uses a Differential Evolution (DE) algorithm for parameter optimization which is an efficient and fast heuristic well suited for this type of problem (see chapter 7 for a comparison of different heuristic methods). In 6.5.1 the DE was used to parameterize a complex model of fermentation in *Saccharomyces cerevisae* using steady-state and time series data. For the first case, the results show that very good fits to the data were obtained, but full model parameterization from steady-state data should be treated with caution as it may not be biologically meaningful or stable. This is not surprising, since the level of information is limited in relation to the number of parameters. For fitting subsets of parameters constrained by the available data, the approach is more reliable. For the second case, a full parameter set evolved that fits well four out of five of the time-course data points and adequately models the dynamics of the system.

Several studies have used simulated data to test network inference algorithms (Thomas *et al.* 2004); the simulated microarray datasets provide an easy way of generating simulated data from a well defined system which can be used to test these methods and how different levels of noise affect their performance. A potential application for SimArray is to bridge modelling studies with microarray studies. For example, a pathway of interest can be modelled and the effects of a treatment that inhibits expression of a particular gene in the pathway can be studied. Time-series microarray data can be used to tune the gene product variables in the model; a good correlation between measured data and simulated data is a solid indication that the model adequately explains the biological properties of the system, which allows further studies without the need of additional experimental data or, alternatively, can suggest further experiments worth pursuing.

Future extensions to SimArray include:

- Import/export capabilities to SBML (Systems Biology Markup Language) format, an XML-based language which allows the different tools to exchange models (Hucka *et al.* 2003; Hucka *et al.* 2004).
- Extend the sensitivities analyses functions to include how changes to independent variables and parameters affect the flux of dependent variables.
- In the current version parameterization constraints are limited to numerical upper and lower bounds; structural model constraints will be included, as for example, precursor – product relationships.

- S-systems have traditionally been used in modeling biochemical systems but have been used in other fields such as economics (Chaudhuri & Johnson 1990); a modified SimArray without the microarray thematic will be released to cater for a broader audience.
- Flow modulators in the graphs are not defined as activation or inhibition; thus the S-systems equations exhibit the component in the production and degradation terms of the modulated variables. Implementing positive and negative modulation will allow the simplification of equations and removal of neutral terms.

7 Hybrid Evolutionary Computation Algorithm

7.1 Introduction

Reconstruction of genetic networks and biochemical pathways is an important research topic in bioinformatics (De Jong 2002). High throughput molecular techniques are rapidly improving and providing data sources that would be untenable just a few years ago. These data can be regarded as a measured result of the complex interactions between genes and gene products; the reconstruction of these interactions is a major challenge for biologists and an important step in furthering our understanding of biological processes.

Many different approaches have been used for modelling biological processes (see section 2.3.1) with systems of differential equations (SDEs) arguably the most widely adopted method for modelling complex non-linear systems. SDEs model the differential relationships between the components of the system. The reconstruction of genetic networks or biochemical pathways involves inverting the problem and building the SDEs from time-series data which more frequently than not is more challenging than solving the systems themselves. The difficulties mainly arise from the high levels of noise present in time-series data, particularly microarray data (Schena 2002) which is the most common source of gene expression data. Chen *et al.* (1999) and De Hoon *et al.* (2002) used SDEs to infer models of gene regulatory networks from time-series gene expression data. A further difficulty is caused by the high dimensionality of the problem, with many genes and gene products interacting, thus originating many possible solutions and local optima.

A biological process modelled as a system of differential equations can be split up into two main components: the structure of the model per se and the parameters such as the constants or the coefficients. An efficient Evolutionary Algorithm (EA) to build SDEs from time-series data must be capable of constructing the structure of the model as well as defining its parameters.

Several prior studies have made use of EC methods to build models as systems of differential equations and evidenced their advantages over other modelling approaches. Ando *et al.* (2002) use Genetic Programming (GP) to infer the structure of gene network models and least mean square (LMS) as a criterion to determine the parameter set. Lanza et al. (2000) and Koza et al. (2001) use GP to evolve both the topology and parameters of metabolic pathways. In a recent study the balance slightly pends in favour of GP over Evolution Strategies – ES (Streichert 2004).

Genetic Programming (Koza 1992), is well suited for structure discovery but is not as efficient for parameter fitting since parameters are initially randomly created and new values can only evolve as mathematical functions of the original values. This characteristic of GP makes parameter values ephemeral since mutation and recombination operators can easily disrupt a good parameter. Add to this the need of allowing for deeper trees to evolve efficient parameters which can increase bloating of the entire function and the chance that the solution will get stuck in local optima due to inefficient parameterization. Differential Evolution – DE (Storn & Price 1997) is an efficient and robust heuristic for parameter optimization of continuous spaces (see chapters 3 and 6) but it is not adequate for model structure prediction.

A hybrid GP for structure discovery of a system of differential equations with an embedded Genetic Algorithm (GA) for the parameter optimization was proposed by Cao *et al.* (2000). Another hybrid approach using Linear Genetic Programming (LGP) with Evolution Strategies (ES) was proposed by Francone and Deschaine (2004). In this chapter a hybrid EC method is introduced to evolve models of biological processes as systems of differential equations and co-evolve a set of parameters for these models from time-series data. Gene Expression Programming – GEP (Ferreira 2001), a GP variant is used for model inference with an embedded DE for model parameterization. A simple method for reducing bloat in GEP – growth in the length of organisms which does not necessarily reflect an improvement in their respective fitness (see 7.4.4) – is also presented. Simulated time-series datasets derived from two models of the *lac operon* in *E. coli* (Mahaffy & Savev 1999; Yildirim & Mackey 2003) are used to evaluate the effectiveness of the hybrid algorithm. The main points of this chapter are to empirically evidence:

1. The construction of SDEs which closely predict the time-series data.
2. The inference of the model structure with the correct relationships between variables.
3. Efficient determination of an appropriate parameter set.
4. That the hybrid method is superior to GEP alone.
5. That bloat can be significantly reduced through our approach.

The remainder of the chapter is organized as follows. In section 7.2 a brief description of the lac operon and the models used to generate the time-series data is presented. Section 7.3 is a general overview of Gene Expression Programming. Differential Evolution was previously discussed in chapter 3 and the reader is referred to that chapter for details of the algorithm. In section 7.4 we discuss the hybrid algorithm and the bloat reduction method; a computational tool developed to compare different Evolutionary Algorithms and which implements the hybrid algorithm is briefly overviewed. In section 7.5 the effectiveness of the method is tested in the reconstruction of two models of the *lac operon*. Some conclusions are discussed in section 7.6.

7.2 *Lac Operon* in *Escherichia coli*

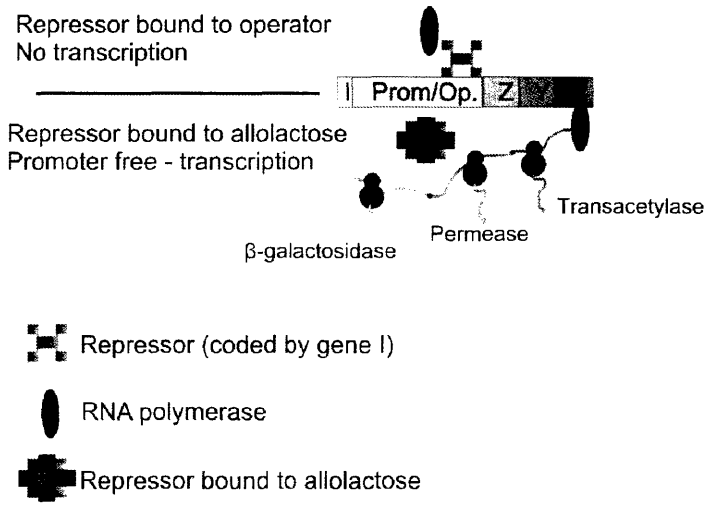
The most widely studied example of positive feedback is the *lac operon* in *E. coli*. The biological theory of feedback dates to the late 50s with the work of Jacob and Monod and was a breakthrough in evidencing gene regulation. Currently gene expression measurements (microarrays) are the most widespread source of data and, for this reason, models of genetic networks in which the primary source of gene regulation is through the control of transcription rates, become highly important.

In the absence of external glucose the *lac operon* (figure 7.1) is responsible for the intake of external lactose into the cell and for breaking it down into glucose and galactose. *E. coli* preferentially uses glucose as an energy source and the genes of the lactose cycle are not expressed. Inhibition is obtained through a repressor enzyme (coded by gene *I*) bound to a 24 nucleotide operator mainly downstream from the promoter region of the three structural genes that form the operon. While the repressor is bound to the operator, RNA polymerase is incapable of binding to the promoter and transcribing the downstream genes. Alongside the operator the operon consists of three structural genes *Z*, *Y* and *A* which respectively translate into the proteins β-galactosidase, permease and thiogalactoside transacetylase. Hydrolyses of lactose into glucose and galactose is carried out by β-galactosidase. Lactose is transported into the cellular environment by the permease. The third protein, thiogalactoside

transacetylase, does not seem to be involved in the lactose cycle. If glucose is available the lactose metabolism is suppressed since glucose inhibits permease and lactose does not enter the cell.

The repressor, coded by gene *I* just upstream from the operon, is a homotetramer protein with binding sites to the operator and to allolactose. When allolactose, an isomer of lactose, binds to the repressor it promotes a conformational change of the homotetramer which can no longer remain bound to the operator thus exposing the promoter region to RNA polymerase which will transcribe the three genes as a single mRNA (Griffiths et al. 2000:336-343). Figure 7.1 summarizes the dynamics of the *lac operon*.

Figure 7.1 Schematic representation of the *lac operon* in *E. coli*. In the absence of lactose the repressor binds to the operator and inhibits transcription (upper half). In the presence of lactose, the repressor binds to allolactose, changes its conformation and can no longer bind to the operator, thus freeing transcription of the structural genes *Z*, *Y* and *A* by RNA polymerase (lower half).



Several mathematical models were developed to analyse the dynamics of the *lac operon*. In this work we use the models of Mahaffy and Savev (1999) and Yildirim *et al.* (2004) to generate simulated time-series datasets of the dynamics of the *lac operon*. These models were selected because they consider transcriptional and translational delays, an important factor that is not taken into account in other models.

The first model consists of four differential equations (7.1), respectively for the concentrations of mRNA (*dM/dt*) with a delay for transcription and translation; permease (*dP/dt*), β-galactosidase (*dB/dt*) and lactose (*dL/dt*). There are eight parameters in the model (table 7.2; Mahaffy & Savev 1999).

$$\begin{aligned}
\frac{dM}{dt} &= \frac{1+k_1 y_4^{\rho}}{1+y_4^{\rho}} - b_1 y_1 \\
\frac{dP}{dt} &= y_1 - b_2 y_2 \\
\frac{dB}{dt} &= r_3 y_1 - b_3 y_3 \\
\frac{dL}{dt} &= S y_2 - y_3 y_4
\end{aligned} \tag{7.1}$$

The second model consists of three differential equations (7.2), respectively for the concentrations of mRNA (*dM/dt*), β-galactosidase (*dB/dt*) and allolactose (*dA/dt*). Delays are used to model the dynamics of mRNA and β-galactosidase. The concentration of internal and external lactose are

assumed to be close to the steady state across the membrane with a direct one-to-one relationship and the concentration of permease is constant. With these assumptions lactose and permease do not have to be considered in the model. There are a total of seventeen parameters in the model (table 7.3; Yildirim *et al.* 2004).

$$\begin{aligned}
\frac{dM}{dt} &= \alpha_M \frac{1+k_1(e^{-\mu\tau_M}A_{\tau_M})^n}{k+k_1(e^{-\mu\tau_M}A_{\tau_M})^n} - \tilde{Y}_M M \\
\frac{dB}{dt} &= \alpha_B e^{-\mu\tau_B} M_{\tau_B} - \tilde{Y}_B B \\
\frac{dA}{dt} &= \alpha_A B\frac{L}{K_L+L} - \beta_A B\frac{A}{K_A+A} - \tilde{Y}_A A
\end{aligned} \tag{7.2}$$

Where $A_{\tau_M} \equiv A(t-\tau_M)$, $M_{\tau_B} \equiv M(t-\tau_B)$, $\tilde{Y}_M M \equiv (Y_M+\mu)M$, $\tilde{Y}_B B \equiv (Y_B+\mu)B$ and $\tilde{Y}_A A \equiv (Y_A+\mu)A$.

7.3 Gene Expression Programming

Gene Expression Programming – GEP (Ferreira, 2001; Ferreira 2002) is a variant of Genetic Programming that instead of coding structures as non-linear entities, typically parse-trees in GP, uses linear strings of fixed size which are translated into non-linear entities of different sizes and structures. This two-step approach allows GEP to be viewed as a hybrid between Genetic Algorithms and Genetic Programming. GEP has essentially three main advantages over GP. Firstly, it allows implementation of the straightforward search methods common to GAs whilst maintaining the structural complexity attainable through GP. Secondly, all linear strings either code for valid structures or can be repaired with little overhead; such is not the case with GP where the parse-trees can easily breakdown into invalid structures and complex search operators must be used to ensure tree viability. The need to preserve valid tree structures limits search operators mainly to recombination between tree branches of the same arity. The third advantage of GEP is that it allows for more parsimonious solutions in contrast to GP where the entire parse-tree is the solution and the tendency is to bloat the tree to the maximum allowed size once the population stops evolving.

The linear strings of fixed size in GEP are referred to as chromosomes. Each chromosome has *n* genes with a head and a tail. The head consists of terminals and functions and the tail only of terminals. Terminals are numerical variables or constants, and functions are mathematical operators. The size of the head is a user-defined parameter while the tail is a function of the size of the head as defined in equation 7.3.

$$t = h(n-1)+1 \tag{7.3}$$

Where *t* is the tail size, *h* is the head size and *n* is the highest arity of the set of functions.

Each gene (*h+t*) can be translated into an expression tree (ET) from the coding region of the gene, referred to as an open reading frame (ORF) in a loose biological analogy. An ORF can be of the same size of the gene or smaller, in the last case there are non-coding regions downstream of the gene. The size of an ORF depends on the position and relationships between functions and terminals on the gene. This is a simple matter of looking at each position on the gene starting from zero and if in that position there is a function its arity is added to a counter. When the counter value becomes smaller than

the current position all terminal nodes in the ET are filled with terminals and no further functions can be added to the ET.

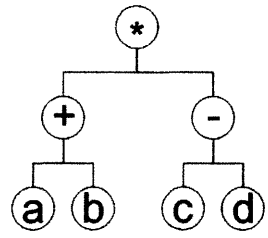
In summary a GEP structure consists of a chromosome with genes of fixed size which code for ORFs of variable sizes that are translated into expression trees (ETs). Search operators act on the gene structure (genotype) while selection acts on the expression trees (phenotype).

To illustrate these concepts consider equation 7.4.

$$y = (a+b)*(c-d) \tag{7.4}$$

This simple algebraic expression can be split into a set of terminals T={a,b,c,d} and a set of functions F={+,*,-). This equation can be represented as the ET in figure 7.2.

Figure 7.2 Expression tree of equation *y = (a+b)*(c-d)*. Terminal set T={a,b,c,d) and function set F={+,*,-}.



The ORF for this equation can be constructed copying the values of the nodes from top to bottom and from left to right, resulting in the string { * + - *a b c d* }. Figure 7.3 shows an example of this ORF as part of a gene of size 21. The head size of this gene is 10 and consists of terminals and functions. The tail (in bold) is formed exclusively with terminals and, from equation 7.3, is of size 11 since the function of highest arity is two. Note that the break off point in the gene is position 6, since in this position the total arities of the functions add up to 6 and the total number of elements is 7. Evidently from figure 7.2 it is clear that no further function could be added to the expression tree. Thus a gene of size 21 can code for an ORF of size 7.

Figure 7.3 GEP gene of size 21 with head (h) of size 10 and tail (t) of size 11. *t = h(n-1)+1*. The highest arity of the function set is 2, thus *n* = 2.

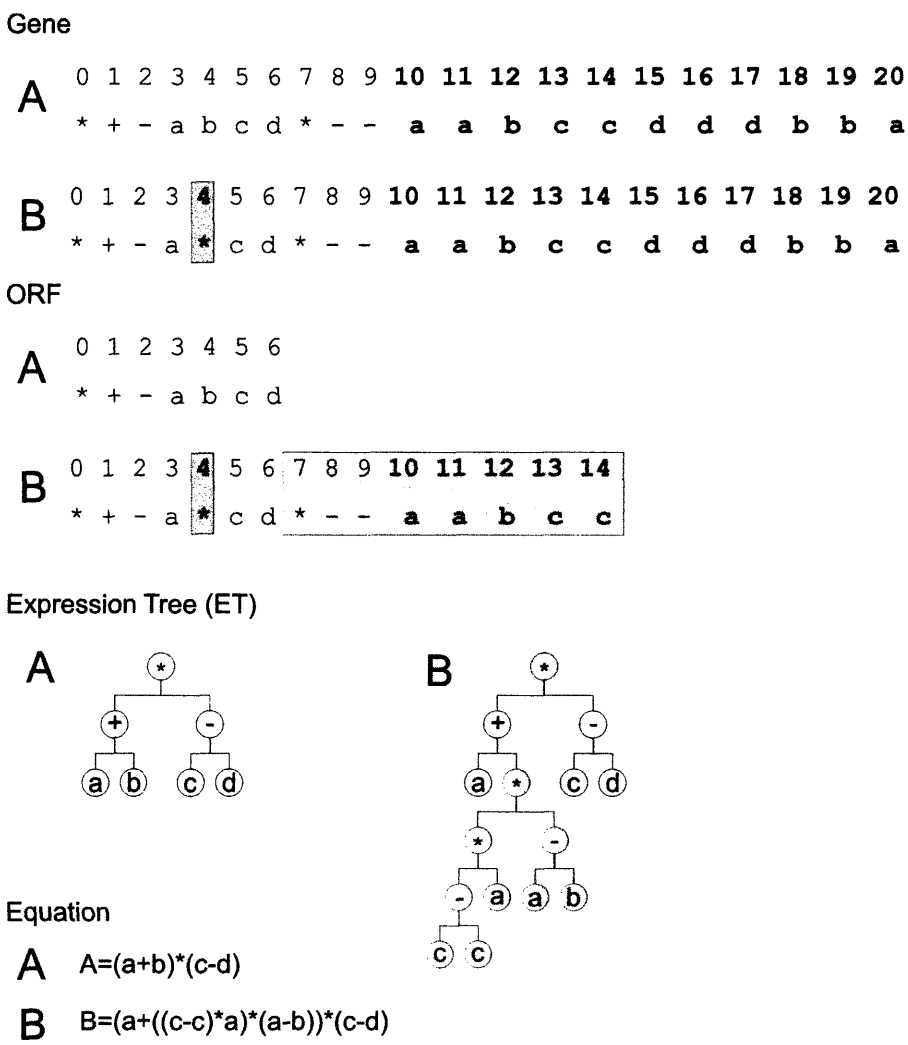
0	1	2	3	4	5	6	7	8	9	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**
*	+	-	a	b	c	d	*	-	-	**a**	**a**	**b**	**c**	**c**	**d**	**d**	**d**	**b**	**b**	**a**

At this point it should be clear that the tail is important to ensure integrity of the tree structure. By having a tail with only terminals, even if the head is solely formed of functions, there are enough terminals available to form complete trees with terminals in the outer nodes. In this scenario the gene and the ORF are of the same size. The value of the function with the highest arity is important to ensure tails with sufficient terminals for any combination of functions.

The distinction between head and tail is an important component of GEP. Tail integrity (only terminals in tail) must be preserved and search operators need to be used accordingly. GEP uses a wide range of search operators: mutation, Insertion Sequence transposition, Root Insertion Sequence transposition, gene transposition, one and two-point recombination and gene recombination (Ferreira 2001).

To illustrate how search operators can modify the structure of ETs yet still preserve structure integrity the mutation operator is considered. Mutation randomly changes the value in a gene position with another value given a certain probability. Since gene structure must remain intact, mutation in the head can replace the original value with either a terminal or a function; in the tail section only terminals can be used. Figure 7.4 shows a point mutation in position 4 in which a terminal (*b*) was replaced by a function (*), as a result instead of the ORF of length 7 a new ORF of length 15 is created. A new expression tree is created and the resulting equation is also very different. Notice that tree viability is still preserved.

Figure 7.4 Effect of mutation on the GEP structures. A single point mutation can completely change the size of ORFs, the ETs and the final product. In position 4 a terminal (b) mutated to a function (*) as shown in Gene B. From this mutation a new ORF was created of size 15 (shaded grey) replacing the original ORF of size 7, and a new tree (new segments shaded grey). The final products are shown under Equation. Tails in bold font.



7.4 Hybrid Evolutionary Algorithm

Genetic networks and biochemical pathways modelled as systems of differential equations have two main components: the mathematical structure of the equations and the parameters of the model. GEP is efficient in structure discovery but less adequate for parameter optimization. Ferreira (2003) presents methods for creating numerical constants in GEP but, within the scope of the models tested in this work, the hybrid algorithm proved superior to GEP alone (section 7.5). For parameter optimization there is a wide range of heuristic methods available: hill climbing, simulated annealing, evolution

strategies, genetic algorithms and their various branches (Michalewicz & Fogel 2000). Differential Evolution – DE (Storn & Price 1997) is a simple and robust heuristic for parameter optimization (see chapter 3) and more efficient than other methods in the test cases of this work (section 7.5).

A novel hybrid evolutionary algorithm was developed that incorporates the capacity of Gene Expression Programming to construct model structures added to the robustness of Differential Evolution in parameter optimization.

7.4.1 Hybrid Differential Evolution and Gene Expression Programming Algorithm

A simplified version of the hybrid algorithm is depicted in table 7.1. Initially random values are assigned to a given set of variables within the constraints defined by the user. Fully unconstrained values can also be used but tend to increase the search times. The algorithm iterates between GEP and DE by a user-defined number of iterations.

Table 7.1 Simplified algorithm of the hybrid method using Differential Evolution and Gene Expression Programming.

```
Initialize random values for variables within user-defined constraints
Do until (termination criterion)
{
        Iteration i
        {
                GEP
                Initialize random population of models
                Replace chromosome 0 with best model
                Do until GEPGeneration = GEPMaxGenerations
                {
                        Select
                        Crossover
                        Mutate
                        Evaluate
                        Replace
                        Generation++
                }
                If (GEP Best Model Improves Fitness)
                        Replace model with best model from GEP
                Else Keep original model

                Bloat Reduction Method

                DE
                Use Best Model to optimize variables
                Initialize random population of variables within constraints
                Replace chromosome 0 with best variables
                Do until DEGeneration = DEMaxGenerations
                {
                        Select
                        Crossover
                        Mutate
                        Evaluate
                        Replace
                        Generation++
                }
                If (DE Best Values Improve Fitness)
                        Replace variables with best values from DE
                Else Keep original variables
                i++
        }
}
```

For the first iteration a random population of models is generated using the initial variable set. GEP is used to select better models. At the end of the GEP run the best model is selected, simplified through a bloat reduction method (7.4.3) and used as the model for the DE to optimize the variable set. At the end of the DE run if the optimized variable set has a higher fitness than the original set it replaces it.

From the second iteration onwards, the GEP run will use the optimized variable set. The initial populations of GEP and DE are randomly generated apart from chromosomes zero, into which is respectively copied the current best model and the current best variable set, thus ensuring that the next round starts at least at the current best solution.

7.4.2 Selection and Search Operators in Gene Expression Programming

Originally GEP uses roulette wheel selection (Ferreira 2001). In this work the currently preferred tournament selection is adopted since it permits better control of the selective pressure and allows for a smoother evolution (Hancock 2000).

Several search operators are used in GEP (Ferreira 2001). Our GEP operators detract from the original set. This choice was based on better convergence in the considered test cases (data not shown) or simple computational efficiency. The operators can be subdivided into mutation and recombination.

Two types of mutation are employed: point mutation and block mutation. Point mutation replaces a single position with another random element from the function or terminal set in the head or just from the terminal set in the tail (Ferreira 2001). Block mutation replaces a randomly defined number of positions (up to 20% of the gene size) with new random elements; again, maintaining tail integrity. The probability of block mutation is set at 10% of the point mutation probability.

Recombination consists of one, two and three-point crossover, with an equal probability for each method. One-point crossover cuts both parents at the same position and the remainder of the gene downstream from the cut point is swapped to form the offspring. Two-point crossover selects a block of the same size, starting at the same position in both parents and this block is swapped in the offspring. Three-point crossover is a simple extension of two-point crossover where instead of one block being swapped, two blocks are swapped (notice that for three-point crossover *four* cut points are necessary). The first two methods are common operators in GEP (Ferreira 2001), the third is a new extension used in our implementation. Three-point crossover is the most disruptive operator employed and is used to replace transpositions, gene recombinations and gene transpositions which were excessively disruptive for these test cases and not implemented in the final algorithm.

7.4.3 Fitness and Objective Functions

The fitness and objective function selected are on a direct one to one mapping and henceforth referred to only as fitness.

Since the hybrid algorithm is geared to finding models of differential equations that reflect the dynamics of a biological system and since the primary source of information considered are time-series measurements of quantities of the components of the system; the fitness function is a measurement of goodness of fit between the predicted values of the model at a given time and the observed values. Equation 7.5 depicts the fitness function.

$$f = -1 * \sum_i^n \frac{\sum_j^m (x_{ij} - y_{ij})^2}{\sigma_{x_i}^2} \tag{7.5}$$

Where the upper term is the sum of the squares of differences between the observed (x_{ij}) and predicted (y_{ij}) values at time point j and the lower term is the variance of the observed data (x_i) for each component (i) of the system. The use of the variance in the lower term scales the sum of squared

deviations so that excessive emphasis is not given to a particular equation in detriment of the others. Fitness is treated as a maximization problem with worse solutions having highly negative values (due to the minus one multiplier) and the better organisms having values closer to zero, which is the maximum fitness.

Even though GEP will mostly yield valid trees, it still is possible that non viable structures will appear. These could be a division by zero or the square root of a negative number. These invalid organisms are assigned a highly negative fitness value. The same approach is employed for unstable equations that overflow.

Fitness evaluation is the most time consuming aspect of the algorithm. A modified Euler method with a fixed step-size of 0.01 is used. According to Cao *et al.* (2000) this step-size is the best trade-off between accuracy of solutions and computational effort.

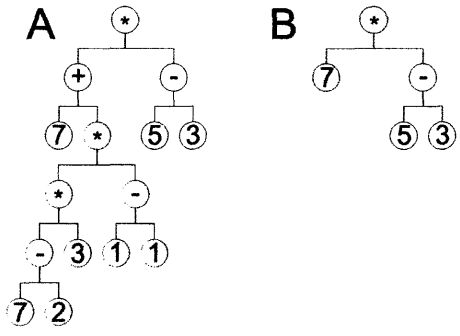
A scenario where tree structure can break down is with the use of time-delays in differential equations. A time-delay can be viewed as an operator that takes two arguments, on the right-hand side a list of the concentrations of a component over time and on the left-hand side a delay (t-τ), and returns the concentration at the delay point. For incorrectly placed time-delay operators instead of assigning a low fitness value, the GEP string is repaired to ensure a viable solution. Repair is carried out probabilistically, either replacing the time-delay operator with another randomly selected operator or replacing the right-hand side with a concentration and the left-hand side with a valid delay.

A last aspect of the fitness function is constraint-handling. If parameter constraints are defined, values that do not meet the constraints are replaced with the average value of the constraint range defined for the parameter, instead of penalizing the entire function. This approach ensures that a higher proportion of the population is formed of organisms that meet the constraint criteria.

7.4.4 Bloat

Bloat is a common phenomenon in Evolutionary Algorithms of variable length, particularly Genetic Programming and its variants. This essentially is the growth in the length of organisms which does not necessarily reflect an improvement in their respective fitness. Frequently the growth consists of regions that do not alter the structure of the solution giving raise to inert regions referred to as introns. An example is long trees attached to an addition branch that result in zero do not alter the value of the solution (figure 7.5).

Figure 7.5 Bloat. The shaded tree is inert and does not alter the final value of the organism. A) A bloated parse-tree with 15 nodes. B) The same tree can be reduced to 5 nodes.



A common cause of bloating in Evolutionary Algorithms is attributed to convergence. If the algorithm ceases to find better solutions or evolution slows down, there will be a greater number of similar solutions and the entire population tends to grow to the maximum allowed size (Langdon & Poli 2002:193-217).

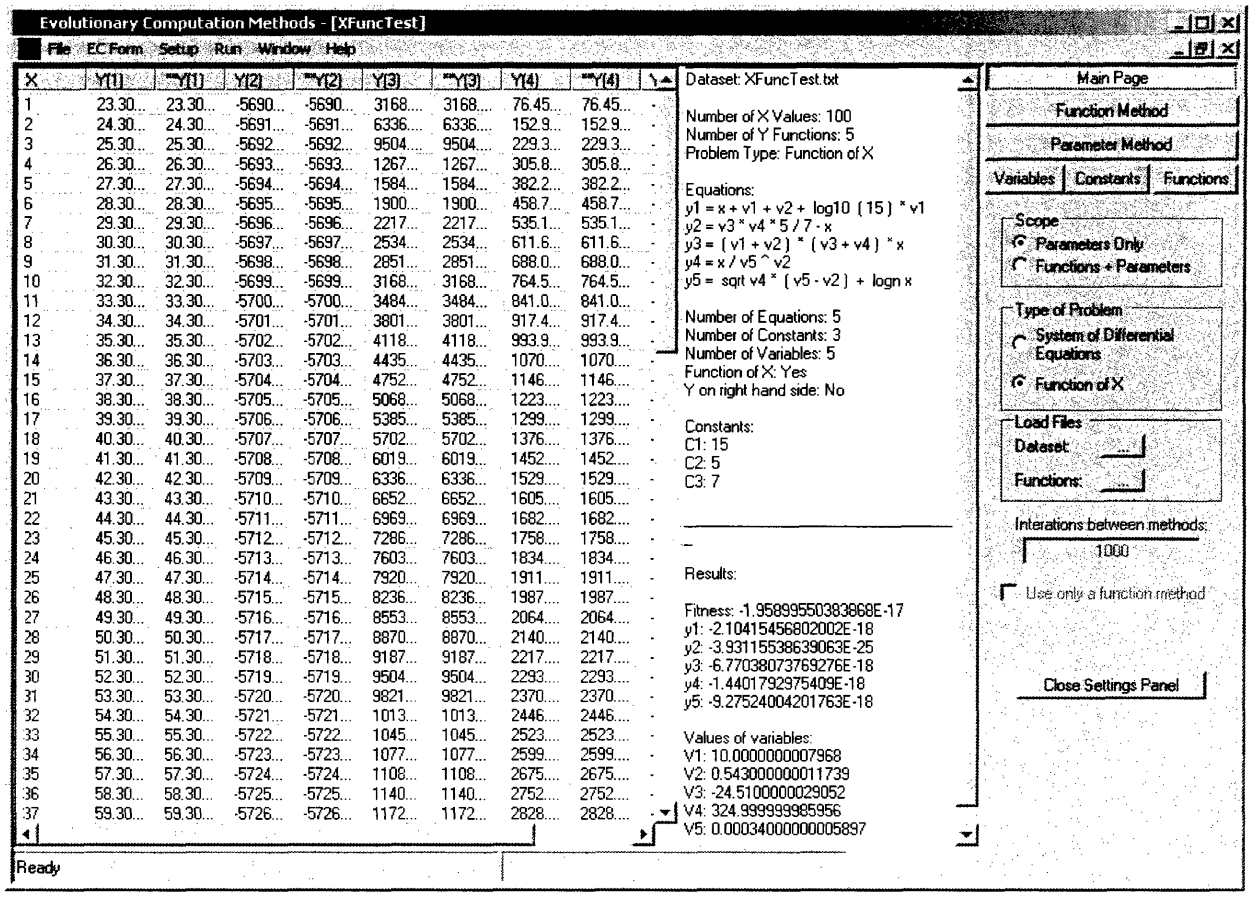
Several approaches have been suggested to reduce bloat. Common techniques include maximum size and tree depth limits (Koza 1992) used in almost all GP applications, a fitness function weighted by the size of the solution (Banzhaf *et al.* 1998) and code editing to identify non-coding regions and prune them out (Blickle 1996).

In this work two methods to reduce bloat in GEP are adopted. The first approach is the common maximum total size limit with no depth limit. The second, more interesting approach, involves the use of a pruning routine at the end of a GEP round before the model parameters are optimized using DE (table 7.1). The pruning routine removes inert structures and replaces subtrees of variables with a single variable thus generating more parsimonious trees without loss of accuracy. This simple approach significantly reduces the length of solutions (see section 7.5).

7.4.5 Computational Tool for Evolutionary Algorithms

A generic C# object-oriented computational tool was developed to allow testing the performance of the hybrid algorithm as well as comparing it to GEP for the considered test cases (figure 7.6). The tool is designed to be easily extensible to include other EAs and heuristic methods besides the ones currently available.

Figure 7.6 Evolutionary Computation Methods Tool. A C# tool that implements the hybrid DE-GEP algorithm. The tool also allows comparison of the efficiency of the hybrid approach to other heuristic methods. In the example Differential Evolution was used to find the values of five variables that, solving for the given equations fit the data in the columns (columns with no asterisks – given values, columns with asterisks – predicted values).



The current version allows for two classes of problems: discovery and parameterization of functions *f(x)* and differential equations *dx/dt*. Input data – the data for which a function and parameters are sought – consists, for the first class, of values for *x* and the respective values for each function *f(x)*. For the second class, data consists of time points and measured values of the components at each time point.

The user can select between using only a parameter optimization method (from Random Search, Stochastic Hill Climbing, Simulated Annealing, Genetic Algorithms, Differential Evolution and Evolution Strategies) with user defined equations (input as a plain text file) or, alternatively, the iterative method described in section 7.4.1 with GEP for model discovery. Even though emphasis is given to the hybrid method using DE and GEP, the tool allows the use of the hybrid approach with any available combination of parameter optimization heuristics and structure discovery methods. A third possibility is to use just a Genetic Programming variant to construct and parameterize the model.

Each heuristic method has its own set of parameters and these can be defined by the user through the Settings panel (right-hand side on figure 7.6). Further settings include definition of constants, variables, constraints on variables and the function set (operators).

7.5 Hybrid Algorithm in Model Discovery and Parameterization

Two mathematical models of the *lac operon* (see 7.2) were used to test the efficiency of the hybrid algorithm. A dataset was generated for each model using the parameters summarized in table 7.2

for the model of Mahaffy and Savev (1999), henceforth referred to as *Model I* and table 7.3 for the model of Yildirim *et al.* (2004), which will be referred to as *Model II*. These parameters are the same ones as used by the respective authors. The first dataset consists of a sampling of 40 points of the concentrations of mRNA, permease, β-galactosidase and lactose taken over forty minutes from *Model I*. The second dataset consists of 40 measurements of the concentrations of mRNA, β-galactosidase and allolactose sampled over a twenty minute time interval from *Model II*. At respectively forty and twenty minutes both models are very close to their steady states.

Table 7.2 Parameters for *Model I* of the lac operon from Mahaffy and Savev (1999). The parameters are not expressed in units since they are not derived from biological experimental data.

Parameter	Value
k1	5
τ	0.86
b1	1
b2	2
b3	0.1
r3	0.1
S	1
ρ	2

Component	Initial Value
mRNA	1
permease	1
β-galactosidase	1
lactose	1

The parameters of *Model I* are not expressed in units since they are not derived from experimental data, consequently they must be viewed purely from a mathematical standpoint (Mahaffy & Savev 1999). The same does not apply for *Model II*, where the authors went to great lengths to estimate biologically significant parameters which as closely as possible reflect experimental data (Yildirim & Mackey 2003; Yildirim *et al.* 2004).

Even though *Model I* consists of four equations (equation 7.1) while *Model II* has three (equation 7.2), the later is a more complex model in regards to the structure and parameterization (tables 7.2 and 7.3), with eight parameters more than *Model I* and all with different values whilst the first model has five numerical parameter values (three values are repeated). The parameter search space for *Model II* is also much broader.

Table 7.3 Parameters for *Model II* of the lac operon from Yildirim *et al.* (2004). The parameters used were estimated from experimental data and closely reflect biological values.

Parameter	Value
μ	3.03e-02 min^{-1}
α_M	9.97e-01 $\mu M min^{-1}$
α_B	1.66e-02 min^{-1}
α_A	1.76e+04 min^{-1}
γ_M	0.411 min^{-1}
γ_B	8.33e-04 min^{-1}
γ_A	1.35e-02 min^{-1}
n	2
K	7200
K_1	2.52e-02 $(\mu M)^{-2}$
K_L	9.70e+02 μM
K_A	1.95e+03 μM
β_A	2.15e+04 min^{-1}
τ_M	0.1 min
τ_B	2 min
L	50 μM

Component	Initial Value
mRNA	4.6e-4 μM
β-galactosidase	1.2e-3 μM
allolactose	4.27 μM

7.5.1 Comparison of Differential Evolution with Other Heuristic Methods

Our first approach was to compare model parameterization efficiency of Differential Evolution with other heuristic methods for the *lac operon* models. To ensure results are comparable, each method performed the same number of fitness evaluations (10,000 for *Model I* and 5,000 for *Model II*) not considering the evaluation of the initial populations. At these numbers of iterations a method might not have converged on a solution, but it is sufficient to provide a clear indication of efficiency within reasonable computational times. Table 7.4 summarizes the parameter settings for each heuristic. These parameters were selected based on published settings and the results of preliminary runs.

Table 7.4 Parameter settings for the heuristic methods.

Random Search		**Genetic Algorithm**	
iterations	10000	generations	1000
		population size	10
Stochastic Hill Climbing		crossover rate	0.8
restarts	10	mutation rate	0.1
iterations	100	tournament size	2
neighbors	10		
closest neighbor	0.01	**Evolution Strategies**	
furthest neighbor	0.1	generations	1000
		μ	10
Simulated Annealing		λ	100
restarts	10	ρ	3
neighborhood (%)	0.1		
retries	10	**Differential Evolution**	
max. temperature	0.5	generations	1000
min. temperature	0.1	population size	10
cooling ratio	0.01	crossover rate	0.5
		mutation rate	0.2

Ten runs were performed with each method and for each of the two models. Table 7.5 summarizes the results of these runs. The fitness value is as described in section 7.4.2, note that the closer the fitness is to zero the closer the predicted points fit the original data. For each model the average fitness of the ten runs and the best found fitness are shown.

Table 7.5 Comparison of efficiency of heuristic methods.

	Model I	*Model I*	*Model II*	*Model II*
Heuristic Method	**Average Fitness**	**Best Fitness**	**Average Fitness**	**Best Fitness**
Random Search (RS)	-891.864	-426.721	-422885.120	-392.133
Stochastic Hill Climbing (HC)	-308.016	-187.310	-391.880	-389.456
Simulated Annealing (SA)	-535.594	-224.687	-273.756	-136.406
Genetic Algorithm (GA)	-183.977	-77.122	-204.117	-124.913
Evolution Strategies (ES)	-127.346	-96.450	-92.839	-27.917
Differential Evolution (DE)	-1.186	-0.002	-62.193	-4.287

As would be expected Random Search (RS) was by far the worst performer, but it still is an interesting method that can be used as a benchmark, in the sense that it provides an estimate of how good an entirely randomly generated solution can be. RS performed particularly poorly with the more complex *Model II*.

Stochastic Hill Climbing (HC) adapted from Michalewicz and Fogel (2000) is highly dependent on the initial randomly generated parameters and can easily get stuck at a local optimum. With *Model II* all HC solutions got trapped close to the same local optimum. Surprisingly, given its wide spread use in optimization problems, Simulated Annealing – SA (Palshikar 2001, with modifications) performed worse than HC with the first test case. This inferior result may be due to a smoother fitness landscape of this model, with HC not getting trapped so easily at a local optimum. For *Model II,* SA performed better breaking out of the local optima where the HC algorithm got entrapped.

The other three methods, Genetic Algorithms (GAs), Evolution Strategies (ES) and Differential Evolution (DE) are population based heuristics and strictly speaking Evolutionary Algorithms. GAs usually are coded as binary strings (canonical GA), in this study a real valued GA was used which is a hybrid between GAs and ES, the usual recombination techniques of GAs are used in conjunction with an ES derived mutation operator that modifies a parameter value adding to it a sample from a normal random distribution *N(0,1).* No self-adaptation of mutation rates was used. The ES is of the type $(\mu/\rho,\lambda)$ adapted from Beyer and Schwefel (2002). The three EAs were significantly better than the other heuristics in the average of the runs and in the best solution (table 7.5). For *Model I* the GA's best fitness was higher than the best fitness of the ES. Nonetheless the average fitness of ES was higher than the GA's. The range of fitness values in ES (-158.415 to -96.4502) was tighter than with the GA (-326.136 to -77.122). Repeatability is an important consideration for adopting a heuristic, even if occasionally a method delivers an above average solution it is usually preferable to employ a method that yields consistent results, evidently without getting entrapped at a local optimum. This is particularly true for the hybrid iterative algorithm proposed. The variability in the results of the GA can be attributed to the use of a constant mutation rate which does not self-adapt as in ES. For *Model II* the best and average fitness of ES were significantly better than the GA.

Of all tested methods DE (see chapter 3 for a description of the algorithm) was by far the most efficient heurist, particularly for the simpler *Model I*. This advantage is less pronounced in *Model II* which

has a steeper and more complex solution space, but it is still highly significant. From these results DE was selected as the method of choice for parameter optimization in the hybrid algorithm.

7.5.2 Comparison of the Hybrid Algorithm with Gene Expression Programming

In this section the efficiency of the hybrid algorithm is compared to Gene Expression Programming (GEP) alone in their ability to construct a model and optimize the parameters for this model that most closely fit the simulated data points of the two *lac operon* models. Five runs were performed for each method and for each *lac operon* model. The parameter settings used are shown in table 7.6.

Table 7.6 Parameter settings for GEP and the hybrid algorithm. The symbol @ in the function set is the time delay function. The concentrations in the variables set are the values of the concentrations over time of each component of the model (4 in *model I* and 3 in *model II*) as estimated by the algorithm.

GEP		
Generations	20000	
Population Size	100	
Crossover Rate	0.9	
Mutation Rate	0.1	
Tournament Size	2	
Head Size	40	
Tail Size	41	
	Model I	*Model II*
Function Set	{+,-,*,/,^,@}	{+,-,*,/,^,@}
Constants Set	{0.1,1,10}	{0.1,1,10,100,1000,10000,e}
Variables Set	Concentrations	Concentrations

Hybrid Algorithm		
Iterations	10	
GEP		
Generations	2000	
Population Size	100	
Crossover Rate	0.9	
Mutation Rate	0.1	
Tournament Size	2	
Head Size	40	
Tail Size	41	
	Model I	*Model II*
Function Set	{+,-,*,/,^,@}	{+,-,*,/,^,@}
Constants Set	{1}	{1,e}
Variables Set	{Concentrations + 10 variables}	{ Concentrations + 20 variables}
DE		
generations	2000	
population size	10	
crossover rate	0.5	
mutation rate	0.2	

GEP is more efficient if numerical constants are left to evolve during the run instead of being hardcoded into the terminal set (Ferreira 2003). Thus, only basic building blocks were included in the constants set within the range of the parameters used in the models. The hybrid algorithm instead of constructing numerical constants uses a set of variables as parameters to optimize during the DE step. The number of variables used was selected as being slightly larger than the number of parameters in each model.

As a result of these runs for *Model I* the best fitness of the hybrid algorithm was -95.481 (average -105.292) compared to -219.838 (average -316.684) of GEP, whilst for *Model II* the best values were respectively -20.158 (average -29.669) and -80.885 (average -124.942).

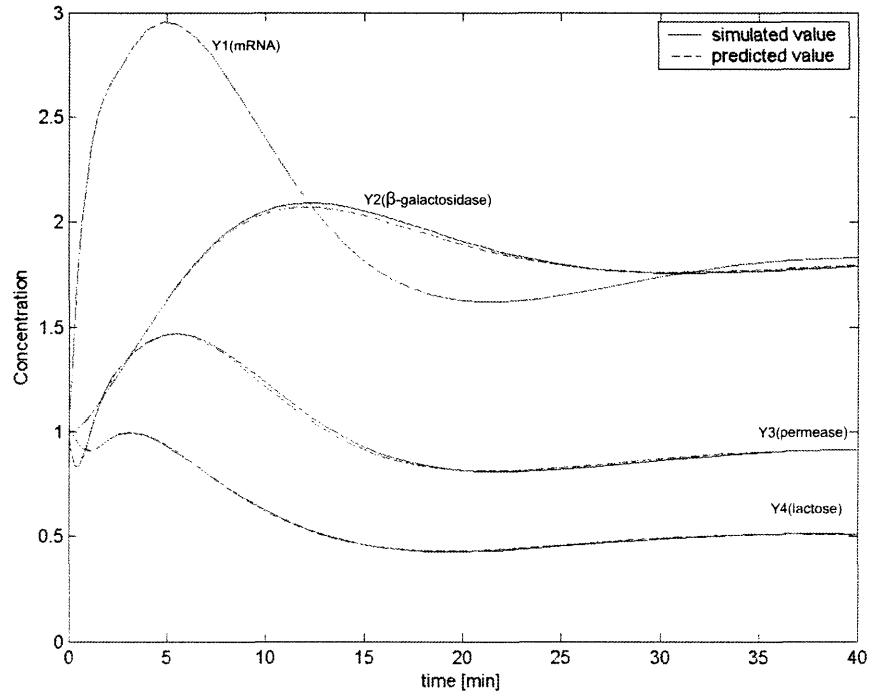
The hybrid algorithm is clearly more efficient for these test cases than only GEP. A further advantage of the method is the size of evolved solutions, with the hybrid algorithm producing solutions on average 47.5% shorter than GEP. This reduction can be attributed to the pruning algorithm (see 7.4.3)

and the lack of need to evolve constants from a basic set since the DE optimizes the constants of the equations.

7.5.3 Evolved System of Differential Equations and Parameters for *Model I*

Five more extensive runs of the hybrid algorithm were performed using the same parameters of table 7.6 with 100 iterations instead of 10. Figure 7.7 shows the fit of the equation set evolved with the hybrid algorithm to the original model of the *lac operon* of the best run. The predicted data and the simulated data points are virtually indistinguishable with a fitness value of $< -2x10^{-3}$.

Figure 7.7 Fit of predicted and simulated data points for *model I*.



The simplified and rearranged form of the run is shown in equation 7.6. Where *y1* is the concentration of mRNA, *y2* is permease, *y3* is β-galactosidase and *y4* is lactose. The original equations (7.1) are on the right hand side to facilitate comparisons.

$$
\begin{aligned}
y1 &= \frac{4.9987\, y_4^{t-0.64} y_4^{t-0.64} + 1}{y_4^{t-0.64} y_4^{t-0.64} + 1} - y_1 \\
y2 &= y_1 - (y_2 + y_2) \\
y3 &= y_1 / 10.1023 - y_3 / 10.1023 \\
y4 &= y_2 - y_3 y_4
\end{aligned}
\qquad
\begin{aligned}
\frac{dM}{dt} &= \frac{1 + k_1 y_4^{\rho}}{1 + y_4^{\rho}} - b_1 y_1 \\
\frac{dP}{dt} &= y_1 - b_2 y_2 \\
\frac{dB}{dt} &= r_3 y_1 - b_3 y_3 \\
\frac{dL}{dt} &= S y_2 - y_3 y_4
\end{aligned}
\qquad (7.6)
$$

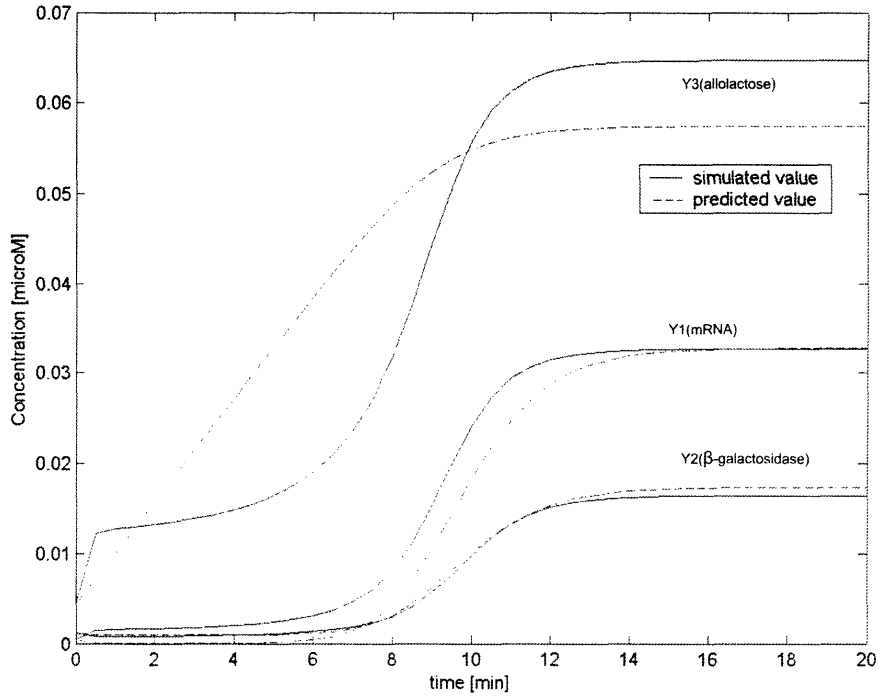
The evolved system of differential equations preserves the structure of the original model with the correct production and degradation components and the relationships between the elements. Out of the ten available variables for optimization only three appear in the final model. These do not necessarily mimic the original parameters but rather are adapted to the evolved equations. An appropriate time delay

was discovered even though it is not a perfect match to the original value (0.86) which is the main cause of deviation between the simulated and predicted data.

7.5.4 Evolved System of Differential Equations and Parameters for *Model II*

The same procedure as described in 7.6.3 was performed for *model II*. The fitness of the best run was -9.5030, not as good as for *model I*. Figure 7.8 shows the fit of the evolved system of equations to the simulated data. The values of allolactose were divided by 1000 to scale them down to allow for visualization on the same graph.

Figure 7.8 Fit of predicted and simulated data points for *model II*. Allolactose levels are scaled down on a 1:1000 ratio.



The evolved equations after simplification and rearrangement are shown in eq. 7.7. The original equations (7.2) are on the right hand side to facilitate comparisons.

$$y1 = 0.123003^{\,8.1379-\left(\frac{y3}{y1+y1+8.7614}\right)} - y1 \qquad \frac{dM}{dt} = \alpha_M \frac{1+k_1(e^{-\mu\tau_M} A_{\tau_M})^n}{k+k_1(e^{-\mu\tau_M} A_{\tau_M})^n} - \tilde{Y}_M M$$

$$y2 = y1 - y1^{t-0.53614} \qquad \frac{dB}{dt} = \alpha_B e^{-\mu\tau_B} M_{\tau_B} - \tilde{Y}_B B \qquad (7.7)$$

$$y3 = 0.12303 - (y2^{0.516968}) \qquad \frac{dA}{dt} = \alpha_A B \frac{L}{K_L + L} - \beta_A B \frac{A}{K_A + A} - \tilde{Y}_A A$$

Where *y1* is the concentration of mRNA, *y2* is β-galactosidase and *y3* is allolactose. Likewise to *model I* out of the 20 available variables only 8 were used in the equations.

The fit of the predicted values to the simulated data points (figure 7.8) is worse than for *model I* particularly for allolactose, but still a reasonable fit (R^2 0.987 – *y1*, 0.999 – *y2* and 0.836 – *y3*) for such a complex model. Changes of the EA parameters may improve convergence to a better fit. Of more concern are the equations which do not always reflect the true relationships between the different

components of the system as these are of key importance to understand a biochemical pathway or a genetic network.

7.6 Conclusion and Future Work

In this chapter a hybrid Evolutionary Computation algorithm was developed to infer the structure and parameters of biochemical pathways as systems of differential equations. GEP is used for model structure discovery with a nested DE for model parameterization. The choice of the latter is based on its better efficiency to determine the parameters of the two *lac* operon models test cases. These results conflict with those of Moles *et al* (2003) which presented evidence that ES is more efficient than DE for global optimization. In their work the model to parameterize consisted of 8 differential equations with 36 parameters while the *lac operon* models have fewer parameters. The difference between ES and DE was less evident in *Model II* which has more parameters. Further studies are needed to determine if DE becomes less efficient in comparison to ES with models that have a greater number of parameters. Further, the focus of our comparisons was on rapid optimization instead of precision. Fast convergence is paramount for the hybrid approach so that fewer iterations are necessary to achieve a reasonable set of parameters for the GEP to work on.

The hybrid approach is more efficient than GEP for these models, the predicted data points had a better fit to the simulated data and, more importantly, the evolved models more accurately reflected the true relationships between the components of the system (data not shown). A further advantage is that through the use of a pruning method between iterations bloat is greatly reduced, thus there is less premature convergence at local optima and the resulting equations are more tractable and simple.

Evolutionary Algorithms are computationally intensive but lend themselves well to parallelization (Whitley 2001; Alba & Tomassini 2002); with current cluster computing technologies the transition is not overly arduous. A future implementation of the hybrid algorithm will allow for parallelization and concurrent optimization of parameters which can be fed back into the GEP in real time. An eventual drawback of EC methods is the lack of consistency in results; some runs prematurely converge on poor solutions whilst other runs are at or close to the global optimum. This inconsistency is closely related to the empirical approach used to determine the initial settings which have to be fine tuned to suit different problems. Recent advances have been achieved in building a solid theoretical framework for EC (Langdon & Poli 2002) but it still is a very active and unexplored field of research. The choice of population size, mutation and recombination rates, the terminal and function set and constraints can greatly influence the outcome of a run. Further studies are needed to determine ideal parameter settings; particularly the size of the terminal set which seems to markedly influence the outcome. In these test cases only a fraction of the available terminals were used in the final evolved models but a certain surplus seems to be important to allow for an efficient search of the solution space. If the terminal set is too large or too small the efficiency of the algorithm was found here to be reduced (data not shown).

The models used in this study to test the algorithm are more complex than commonly used test cases since we wanted to test the efficiency of the algorithm under a more realistic scenario. For *model I* the fit, structure and parameters were very close to the original model, with the evolved model not only serving as a predictor but also capable of providing knowledge about the dynamics and interactions of the components of the system. For *model II* the same does not apply. A reasonable model to fit the data evolved but it is not necessarily good at predictions. More importantly, it is mathematically complex and

not informative of the real relationships of the elements and this is significant since the ability of the method to reverse engineer pathways and networks seems to decrease with more complex systems, thus scalability concerns should be addressed.

But how do we currently build models of systems such as the *lac* operon? This is done by experimental intervention, and, generally, simple statistical and human interpretation. The approach under discussion promises much greater power for the interpretation step. With appropriate manipulation of evolved models, it may also help point to new experiments to be carried out that will give most power to resolve those points about which most model ambiguity exists. Cycles of incisive experiments and powerful interpretation may lead to much faster understanding of biological systems.

Scalability issues will be addressed in future work. Use of S-systems (Voit 2000; chapter 6) to reverse engineer a system could provide a more robust framework since they use a rigid structure that is well suited for the DE to parameterize and for GEP to select the components involved in the underlying biological process.