1 Introduction

Biology has been under the spotlight several times over the last twenty years. Breakthroughs such as cloning and the sequencing of the human genome changed the world. These ground breaking discoveries triggered a change in attitudes to biology. What used to be a reductionist science became holistic. What used to be event-driven became data-driven (Aggarwal & Lee 2003). Biology became big. High-throughput techniques churning out huge amounts of data are the norm. Databases are growing exponentially (Roos 2001). Rutherford might have insisted that it still is stamp collecting, but it certainly is a collection that even he would be proud of.

At the International Genetics Conference of 2003 in Melbourne a renowned scientist mentioned that his entire PhD research could be done in less than fifteen minutes on a modern sequencer. More importantly, there is no need to repeat the work since it is readily available just a mouse click away on the Internet. Biology is being swarmed by data and the main challenge for the next decades is to integrate, use and make sense out of all this information (Chong & Ray 2002). As John Mattick from the University of Queensland puts it 'It's all about putting Humpty-Dumpty back together again'. Biological systems are complex and not prone to axiomatization. A simple list of components will not explain how they work, but it is the basic starting point for reconstructing them and understanding them.

Modern Biology is heavily dependant on data, high-throughput techniques and on the development of computational tools and analytical methods that allow the extraction of new knowledge from all the data available. This approach may lack the finesse of a mathematical proof; but sheer brute force to generate large datasets and vast computer resources to analyse them seems the most viable and immediate alternative to tame the complexity of life.

But technology is still scratching at the surface of biological complexity. We still do not have widespread resources capable of optimizing large systems or solving complex problems purely through exhaustive search nor algorithms guaranteed to converge on optimal solutions. To advance in the field we have to rely on intelligent computational methods that are capable of reducing the search space that must be explored and can guide the search to the most feasible areas. These methods might not always give optimal results but at worst they offer good approximations that can be further tuned. A blooming field of research for such methods is Evolutionary Computation (EC).

Evolutionary Computation is a vast field of research in optimization methods loosely inspired by biological evolutionary processes. The central idea behind EC is to create populations of candidate solutions of a problem and evolve these populations by selection based on an objective function which emulates natural selection. EC methods are very efficient to explore and find solutions in large, complex non-linear search spaces (Fogel 1999; Bäck 2003; Bäck *et al.* 2000a; Bäck *et al.* 2000b).

The driving theme behind this thesis is to develop effective Evolutionary Computation methods and apply them to solve biological problems. An overview of each individual chapter is presented further down but, essentially, the thesis can be split up into two main branches: finding solutions to biological problems and reconstruction/parameterization of biological systems. For the first topic, three complex problems of practical importance which experimentalists routinely face are addressed. These problems, apart from the practical spin-off applicability, can be seen as a proof of concept of the validity and

usefulness of EC methods. The second topic is of a more theoretical nature, in which an EC method to construct and/or parameterize mathematical models of biological systems is developed. Emphasis is given to the parameterization of models as S-Systems – power law functions of differential equations (Voit 2000) and generation/parameterization of systems of time delay differential equations.

To help with a wider adoption of these methods several computational tools were developed. A full description of each tool is presented in the relevant chapter. Software design and development was an important component of this project; all programs were designed following the current graphical user interface standards and adequate error handling was implemented. The programs are freely available from the author and/or the Internet.

The remainder of this introductory chapter is a breakdown of the different chapters in the thesis. Initially an overview of the structure of the thesis is given, followed by a brief description of the contents of each chapter.

Chapters 3 through 7 are the production chapters per se, with brief details given below. They are presented as independent pieces of work since each chapter addresses a different biological problem. They all fit under the common umbrella of Evolutionary Computation, but the nature of the specific problem and the EC methods developed to solve them differ from problem to problem. Thus, instead of the typical format of a comprehensive literature review and a methods chapter followed by the production chapters, each chapter contains its own review of the relevant literature for the biological problem followed by a methods section for the EC adopted or developed. Where a topic has been previously introduced the reader is referred to the relevant chapter and section. Each chapter has been kept as independent as possible which should allow the reader to select the topics of interest without much loss of context.

These five chapters all follow the same basic outline. The first section is a brief presentation of the problem followed by an overview of the relevant literature in the field. Even though the title of the headings vary, subsequent sections include a methods section in which the adopted EC approach is discussed along with a description of the computational tool developed; trailed by a results and discussion section – generally an evaluation of the efficiency of the method. Some conclusions and possibilities for future work are drawn in the closing section.

Chapter 2 is a global literature review chapter. It can be viewed as a contextualization chapter. To get started we have a look at Computational Biology and Bioinformatics, how they are reshaping Biology into a quantitative science and further argue that modern Biology could not be where it is without the tools and methods developed in these rapidly advancing fields (Mount 2004). Next, the reader will have a brief encounter with the emerging field of Systems Biology (Kitano 2001a), the philosophy behind it and the research methods employed. Moving on, we delve into the world of modelling biological systems, which is a crucial aspect of Systems Biology studies. We briefly review the most prominent approaches that are being used (Bower & Bolouri 2001) and some of the computational tools that have been developed. This is followed by an overview of microarrays (Schena 2002; Draghici 2003; Knudsen 2002), a method which delivers great quantities of parallelized data of the states of biological systems. Microarrays are a valuable source of information for modelling systems and ultimately for use in Systems Biology. The final section of this review and also the most extensive explores the fast growing field of Evolutionary Computation. Ultimately this thesis is focused on applying Evolutionary Computation to biological problems and a more extensive exploration of ECs and their applications in Biology is prominent.

Chapter 3 studies the use of Differential Evolution (Storn & Price 1997), an EC method, to optimize nutrition parameters in beef cattle. Using growth models as objective functions the DE searches for the best combination of selected inputs that enable a target output. A computational tool developed for running the optimization routine is used and its results are validated using experimental data.

Chapter 4 addresses the Multiple Sequence Alignment problem which is an important research topic in Bioinformatics and an essential tool in molecular biology. An algorithm using Genetic Algorithms and dynamic programming was developed and scored against the Clustal W alignment program using the BaliBase benchmark (Thompson *et al.* 1999b).

Chapter 5 looks into the use of Evolutionary Computation algorithms to optimize the experimental design of spotted microarray experiments. Experimental design is moving to the centre stage with the microarray community as researchers are increasingly becoming aware of the need for well planned experiments if they are to expect experiments to yield significant results (Churchill 2002). A genetic algorithm was developed to optimize experimental designs using weighted multicriteria objectives as an objective function. Experimental designs evolved by the EC algorithm are compared to optimal designs found through exhaustive search.

In Chapter 6 a computational tool that allows the user to graphically build networks of biological processes and generate simulated microarray datasets was developed. The underlying engine converts these graphs into S-Systems of differential equations (Voit 2000) which can then be parameterized by the user or inferred from available data through a Differential Evolution optimization algorithm (Storn & Price 1997). The capacity of the optimization algorithm to discover the correct parameters of biochemical systems is evaluated.

Chapter 7 presents a hybrid Evolutionary Computation algorithm which merges Gene Expression Programming (Ferreira 2001) with Differential Evolution (Storn & Price 1997). A simple pruning method to reduce bloat is implemented in the algorithm. This hybrid algorithm explores the best features of each method and enables concurrent model discovery and parameterization. Two models of the *Lac operon* (Mahaffy & Savev 1999, Yildirim & Mackey 2003) are used as a benchmark to compare the hybrid method with the individual methods and other alternative EC representations. A computational tool that implements several optimization heuristic methods was developed.

Chapter 8 concludes the thesis with some general considerations about the development of Evolutionary Algorithms followed by a summary of the entire project focussed on the algorithms developed and their computational implementation. Contributions to the biological problems are also highlighted.

Evolutionary Computation has already gone mainstream. It has left the realm of theory and found its way into real life scenarios as a powerful tool to solve optimization problems and to extract knowledge from complex systems. Apart from the use of the tools and methods presented; I trust that at the end of this thesis the reader will share my enthusiasm for Evolutionary Algorithms and decide to test them on their own set of problems.

2 Literature Review

This chapter should provide the reader with the background information needed to understand the chapters that follow. As mentioned in the Introduction, chapter-specific literature is discussed within each chapter. Here, discussions are limited to the broader scope of each field. Each of the following sections addresses an entire field of science, and so this brief review can only breach the surface of each subject and look into the topics of interest in this study. The reader interested in a broad view of a specific area is advised to start with the reference texts which are cited in each section. The Evolutionary Computation section is more extensive and provides a general overview of its different branches, with special attention given to the topics of interest for this thesis.

2.1 Computational Biology and Bioinformatics

Computational Biology is an interdisciplinary field anchored on Computer Science and Biology, which deals with the development and implementation of methods to study biological systems. The term is frequently and freely interchanged with Bioinformatics. Research interests between the two overlap quite extensively, a fact which is clearly evidenced by the similarity of focus of the articles published in two of the main journals in the area – *Journal of Computational Biology* and *Bioinformatics*. To avoid this confusion NIH – the National Institutes of Health – released a functional definition of the two terms (Huerta *et al.* 2000). In practical terms the difference between Bioinformatics and Computational Biology resides in a more applied, data-analysis and data-storage driven approach to the former and a more theoretical, model/simulation driven approach to the later. Further, Bioinformatics is normally associated with computational research applied to data obtained from molecular biology whilst Computational Biology is broader in scope encompassing not only genetic and biochemical networks but also higher level systems such as ecological systems and population dynamics. For the scope of this thesis we will not be overly burdened to distinguish between Computational Biology and Bioinformatics. Loosely chapters 4 and 5 are in the realm of bioinformatics and chapters 3, 6 and 7 fall under computational biology.

Molecular biology has advanced at a staggering rate over the last decades. A large number of organisms have been fully sequenced and many more are in the pipeline. The number of DNA base pairs that have been sequenced is growing exponentially. In 2004 there were close to forty-five billion base pairs at the National Center for Biotechnology Information (NCBI) deposited in the GenBank database (GenBank 2005). Protein sequences and microarray studies are following the same path. Bioinformatics emerged from the need to maintain, analyze and interpret these ever expanding data (Thornton 2003:xiii-xiv). No modern molecular biology lab operates without relying heavily on computers to store and analyze their data as well as accessing knowledge from the public domain databases. Without the computational tools and databases developed in bioinformatics, working with such vast quantities of data would be untenable. These two factors – massive datasets and intensive use of computers – have shaped modern biology. Biology has become an event-driven, information-centered science. And as such it has been crucial to incorporate new techniques from computer science, information technology, mathematics and

statistics. These methods applied to biology form the framework of the multidisciplinary field of bioinformatics.

But not only is biology profiting from the allegiance to the technical sciences. In an interesting turn of events, biology is driving the IT industry and leading the shift from custom mainframe computers to clusters of computers based on commodity-priced components. These clusters are rapidly growing in importance due to their relative low costs and the need for greater computational resources. Clusters are a natural course for bioinformatics due to the parallel nature of biological problems which, not infrequently are referred to as being "embarrassingly parallel" due to the ease with which they can be split into smaller problems (Augen 2003).

The full scope of bioinformatics can be rather daunting as evidenced by the bioinformatics curriculum suggested by Altman (1998). A good entry point into the field is Mount's (2004) bioinformatics book which is a comprehensive introductory text and covers the most relevant topics of bioinformatics at an introductory level. Research in bioinformatics includes databases and datamining, pair-wise alignment of sequences and multiple sequence alignment (see chapter 4), prediction of RNA secondary structure, gene prediction, phylogeny, protein structure prediction, analysis of gene expression data, genome analysis and reconstruction of biological pathways, just to name some of the current topics. Bioinformatics is a booming field with four major journals exclusively devoted to the field. The Journal of Bioinformatics and the Journal of Computational Biology follow the traditional publishing scheme; BMC Bioinformatics and the newly launched PLoS Computational Biology follow the open-source model.

A growing area of interest within the bioinformatics community resides in the integration of the different sources of data to construct models which allow system level studies of biological processes. These models are usually based on simulations which try to predict the dynamics of systems and permit testing the soundness of the postulations built into them. This contrasts with typical knowledge discovery problems as for example the inference about clusters of genes from expression profiles, which tend to use heuristic or statistical methods to derive insights (Kitano 2002a). Growth in this area is clearly evidenced by the expanding number of articles published in the leading journals. Campbell and Heyer (2003) base their book entirely around this paradigm, making for a solid introductory text to develop the conceptual mindset for the emerging field of Systems Biology. These two areas overlap considerably in their objectives yet the methodologies are quite diverse.

2.2 Systems Biology

Kitano (2001b:1) defines Systems Biology as 'a new field in biology that aims at system-level understanding of biological systems, such as cells and organisms'. Alongside RNA interference, Systems Biology is an area that is attracting a lot of attention in the Life Sciences. But the concept is not new, as early as the 1930's researchers of systems theory and cybernetics were interested in biological systems due to their inherent complexity. Researchers such as Wiener, Kalman, Bertalanffy, Rosen and Mesarovic were involved in the 'first wave' of Systems Biology. At the time the lack of data and limited computational resources impaired advances in the field (Wolkenhauer 2001).

This has changed. Advances in molecular biology techniques have provided a previously untenable amount of information at the molecular level. Genomics research has generated quantitative transcriptome and proteome data that has only recently become available. And most certainly molecular

biology techniques and high-throughput methods will continue evolving rapidly, generating even more data and even more detailed information of biological entities at a molecular level (Kell 2004).

As the components of biological systems are being laid out; it is becoming clearer that these systems cannot be seen as a simple collection of parts. More importantly, they are formed by their constituent elements and the interactions between them (Kitano 2000, Kitano 2002b) which coincides with the usual view of a system as a collection of components and the interactions among them. Thus Systems Biology can be seen as Systems Theory applied to Genomics (Wolkenhauer 2001).

Systems Biology is the exponent representative of the shift that life-science research is undergoing. Focus is changing from a reductionist approach centered at identifying and understanding the function of individual components to a holistic approach geared towards an integrative understanding of biological systems (Aggarwal & Lee 2003). Notably a system-oriented approach is not viable without relying on knowledge derived from reductionist studies, so the approaches should not be seen as conflicting but rather as complementary. The need for an integrative approach is clear from the fact that single levels of information cannot fully explain the dynamics of biological processes. A common example that is often mentioned is the lack of a linear relation between mRNA and protein expression levels in cells (Gygi *et al.* 1999; Anderson & Seilhamer 1997). To reconstruct even simple networks, information from mRNA and protein expression levels is required (Hatzimanikatis & Lee 1999). As Kacser (1987:327) put it 'one thing is clear: to understand the whole one must study the whole.'

A holistic approach to studying biological systems implies operating with complex systems. Here it is necessary to distinguish between complex systems and biological complex systems. Complex systems are commonly defined as a large collection of elements which interact to product complex behaviors. Biological systems differ in the sense that the elements of the collection are, per se, complex. Thus, the complex behavior does not exclusively result from the interactions of the network but rather from the combination of the complex elements that form it, the interactions between these elements and the control nodes of the system (Kitano, 2002a).

2.2.1 Characteristics of Biological Systems

Biological systems and complex systems show many characteristics in common. But biological systems are the product of billions of years of evolution. Selective pressures over such long time spans have evolved features that are unique to biological systems. Even though most species exhibit great genetic variation and are subjected to different environmental conditions, the phenotypic variation is relatively low. That is, organisms are robust to genotypic and environmental changes. This principle is known as canalization and has long been recognized as a mechanism evolved to channel a range of genotypes to express similar phenotypes that are close to optimal, helping to conserve genetic variation (Stearns 2002).

Robustness is the most discussed characteristic of biological systems. This is an essential feature which ensures that the system will be able to maintain its functional properties when exposed to various disturbances. Understanding robustness is central in Systems Biology, namely to understand how systems respond to environmental changes and internal failures (Kitano, 2001b). Robustness is of fundamental importance in treating diseases, for regrettably robustness is a common feature of biological systems, with pathogens and cancer cells being robust as well (Kitano 2002a).

Kitano (2001b) suggests that the robustness of biological systems is essentially achieved from the use of system controls, redundancy, modular design and structural stability. Kitano further suggests that these properties are intrinsic to complex systems.

Various system control methods are employed by biological systems such as feedforward and feedback which are present in almost any organism. A classical example of positive feedback is the *lac* operon of *Escherichia coli* (Jacob *et al.* 1960) consisting of a promoter region followed by three structural genes (*lacZ*, *lacY* and *lacA*) and a repressor operon further upstream.

Redundancy is a commonly adopted strategy to increase the robustness of a system to damage to certain components of the network. By employing multiple pathways to accomplish the same task, alternative routes can be found to ensure the continuity of the process. Gene duplication and genes with similar functions exemplify redundancy; but redundant pathways and mechanisms can also involve quite different methods or routes.

Adoption of modular designs mitigates system-wide deleterious effects of disruptions to a certain pathway. A natural consequence is that modular designs help contain damages from spreading throughout the system. A further evolutionary advantage presents itself, with the modules serving as building blocks for extended or derived functions. Modularity is clearly evidenced through the addition of further complexity levels from cellular organelles to cells, tissues, organs and finally systems.

It is not clear the extent to which structural stability applies to biological systems (Kitano 2001b). Nevertheless many systems are able to operate within a wide range of parameters and maintain/resume homeostasis after a disruption. Higher level systems can operate within, for example, a wide range of temperature, salinity and hydration.

Another aspect of biological systems which tends to be disregarded is that they are sub-optimal. Biology is opportunistic and processes tend to converge on solutions which are functional but rarely the overall best. This characteristic limits the application of deterministic optimization techniques to biological systems since these solve for the optimal solution. Further, control theory assumes that states are initially defined, whereas in biological systems these states are constantly modified by the system itself (Kitano, 2002a). To explore these sub-optimal systems and evolving state dynamics, the use of stochastic methods, such as Evolutionary Computation (discussed in section 2.5) can prove efficient since they emulate selection processes, explore different areas of the solution space and can dynamically modify the states of the system.

2.2.2 Methods in Systems Biology

Systems Biology is essentially a multi-disciplinary subject converging knowledge from statistics, mathematics, computer science, engineering and biology. Kitano (2001b) identifies four-key areas of research for Systems Biology: genomics and other molecular biology research; computational studies including simulations, bioinformatics and the development of software; systems dynamics analysis and the development of high-throughput measurement technologies.

Methods in Systems Biology will usually include the following five research steps (Campbell & Heyer 2003:267-268):

1. Define all components that characterize and influence the system.
2. Build a model of the system to make inferences.
3. Disrupt each component of the system in wet-lab experiments.

4. Use system-wide, high-throughput techniques data (e.g. microarrays).
5. Refine the model when experimental and predicted data diverge.

These methods provide the framework for a systems-level understanding of a process which, according to Kitano (2002b) is composed of the structures of the system, the dynamics of the system, the control methods and the design method. A benchmark example of this integrated approach is a study on the utilization of galactose in the metabolism of yeast (Ideker *et al.* 2001) in which they build a model of a cellular pathway using perturbations of system components measured through microarrays, quantitative proteomics and databases of interactions.

Technological advances are the main impulse behind Systems Biology. Advances in high-throughput techniques will make or break the field. But a computational backbone is also of the utmost importance. Several modeling tools and simulation programs are being developed. It is necessary to ensure that these tools can communicate and exchange data, operating under a common standard. That is what the Systems Biology Markup Language (SBML) proposes to be – a common XML (Extensible Markup Language) based language which allows the different tools to exchange models (Hucka *et al.* 2003; Hucka *et al.* 2004).

It is not a trivial task to adapt the different modeling approaches to SBML standards due to the different nature of models employed and the modeling task at hand. Nevertheless, modeling is crucial to Systems Biology and it is of paramount importance that the different approaches can be integrated under a common framework. Section 2.3 discusses different modeling methods commonly used in Systems Biology and some of the computational tools developed.

2.3 Modelling of Biological Systems

Computational advances and availability of vast amount of biological data have combined to create a fertile ground for the development of models of biological systems. The sheer complexity of biological systems demands the use of modelling tools to interpret biological data. Models are of undisputed value to integrate data and gain insights about the functioning of a system. Experimentalists might argue that biological data is still limited but even under this scenario models are useful to understand the limits of knowledge and even suggest where further experiments are needed.

High-throughput techniques are providing data that allow testing more complex models, thus quantitative models anchored on real biological measurements are gaining terrain. Biological systems are complex and it is becoming clear that simplistic reductionist models are not sufficiently detailed to explore this complexity. Researchers should not try to force data into simple models but rather develop models which can handle it (Gondro 2002). With more realistic models not only are predictions more reliable but the models can be driven with actual experimental data which helps to test and improve them. Further, a conceptual shift is ensuing with models. Traditionally models were constructed from a preconceived idea whereas the current approach is to develop models based on experimentally measured relationships. Thus, models can be seen as a method for formalizing relationships and organizing experimental data (Bower & Bolouri 2001).

The constituents of biological systems not infrequently operate in different pathways, making system prediction quite counterintuitive. Without a rigorous quantitative analysis of the systems it is difficult to make predictions of the dynamics of the system. Add to this the nonlinearity of the relationships

and states of the components which demand a mathematical analysis (parameter, sensitivity) if we are to make effective predictions (Voit, 2000:4).

For these reasons powerful modelling approaches must be used to gain insights of the underlying biology. Modelling of biological systems is an active field of research (Bower & Bolouri 2001; De Jong 2002; D'haeseleer *et al.* 2000). A great number of modelling methods have been applied to biological systems. The next section overviews some of the most commonly adopted.

2.3.1 Modelling Approaches

Models range from graphs and Boolean networks all the way through to models of systems of differential equations. We will limit this discussion to a brief description of graphs, Boolean networks, stochastic models and differential equations commonly applied to systems biology; or, more specifically, applied to genetic and biochemical networks. A good entry point to the field of modelling is provided by Bower and Bolouri (2001), Voit (2000) and Haefner (1996).

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). But they do not allow studies of the dynamics of the pathway as Boolean networks do.

Boolean networks are based on Boolean logic (Kauffman 1993) in which, in the case of genetic networks, each gene can be either expressed (active/on) or unexpressed (inactive/off). This structure naturally confers discrete and binary (0/1) states to the network. A network with n elements will have a state space of 2^n states. New states are computed from a previous state $t-1$ according to a set of Boolean functions that represent the interactions between the elements. All outputs are synchronous, that is all outputs are simultaneously updated at time point t. A further characteristic of Boolean networks is their deterministic nature, meaning that there is a unique output for a specific input (De Jong 2002; Gibson & Mjolness 2001; D'haeseleer *et al.* 2000).

Boolean Networks provide an initial overview of complex pathways with many elements and limited data availability. Also, due to their discrete and binary structure these networks are computationally less expensive, making them attractive for large networks. Several approaches which offer greater granularity to Boolean networks have been proposed: kinetic logic models, continuous logical models, fuzzy Boolean models (Gibson & Mjolness 2001). Boolean models and variants have been used in reconstruction of genetic networks (Shmulevich *et al.* 2002; Liang *et al.* 1998; Maki *et al.* 2001) and inference of networks using microarray data (Akutsu *et al.* 2000; Somogyi et al. 2001).

At a molecular level the common assumption that systems vary continuously and deterministically may no longer be valid (Gibson & Mjolness 2001). This becomes evident in systems with a small number of molecules such as some transcription factors. Stochastic models (Gillespie 1977) can be used to

model these systems or systems with short time-spans; instead of taking the continuous and deterministic avenue, these models use discrete and stochastic approaches with exact numbers of molecules instead of concentrations. Stochasticity appears in state changes which are defined by a joint probability distribution with final states not guaranteed to converge on the same values (De Jong 2002).

Stochasticity is the key concept behind stochastic models. State changes are probabilistic instead of deterministic. Monte Carlo simulations are commonly used to construct a probability distribution of the system (McAdams & Arkin 1998; Gibson & Bruck 2001). Stochastic models are closer to the biological reality of molecular pathways but it is computationally expensive to calculate the probability distribution and they demand an in-depth knowledge of the modeled system (Gibson & Mjolness 2001; De Jong 2002). Further, over longer time-spans and systems with a large number of molecules the stochastic effects may average out, making deterministic models an adequate and less expensive approximation (Gillespie 2000).

The most widely adopted formalism used to represent dynamical systems is Ordinary Differential Equations (ODEs). ODEs have been extensively used to simulate genetic networks and biochemical systems (Jones & Sleeman 2003; Allman & Rhodes 2003; Voit 2000, for introductory texts). Under this approach states are non-negative continuous concentrations of the different components of the system and change deterministically according to differential equations. Thus, systems are modeled as time-dependant functions of the concentrations of the other elements of the system, taking the form of the rate equation

$$dx_i/dt = f_i(x) \tag{2.1}$$

Where $x = [x_1,\ldots,x_n] \geq 0$ is the vector of concentrations and the function f_i is generally nonlinear (De Jong 2002).

Most realistic networks modeled as ODEs become too complicated to be solved analytically and can only be solved numerically. To solve these systems, not only the equations themselves are necessary but also a vector of the initial concentrations. The main hindrance to the use of numerical techniques is the lack of the actual kinetic parameters for the model. More often than not parameter values are selected that enable the model to reflect observed measurements. This problem should become less of an issue since there is a growing availability of time series information which can be used to infer adequate parameters (Gibson & Mjolness 2001; De Jong 2002). Arguably parameterization, even though highly relevant, may not need to be so accurate; the structure of the network might be more important to confer stability than the actual parameters (von Dassow et al. 2000).

Several differential equations approaches have been applied to biological systems (De Jong 2002), of these many approaches S-Systems (S refers to synergism and saturation) are of particular interest due to their rigid structure and mathematical tractability. S-Systems use power-law functions for the canonical modeling of the states of nonlinear systems. This formalism allows the construction of systems of differential equations suitable for the representation of any differentiable nonlinear function (Savageau 1996; Voit 1991; Voit 2000). S-Systems always take the general form

$$\dot{X}_i = \alpha_i \prod_{j=1}^{n+m} X_j^{g_{ij}} - \beta_i \prod_{j=1}^{n+m} X_j^{h_{ij}} \quad \text{for } i = 1, 2, \ldots, n. \tag{2.2}$$

Where α_i and β_i respectively indicate the production and degradation rate constants for X_i and both are ≥ 0. Likewise g_{ij} and h_{ij} are the production and degradation kinetic orders of the elements X_j

involved in the process. This rigid structure makes analysis of these models relatively straightforward and well-suited to parameterization through Evolutionary Computation (see chapter 6). Further details on S-systems are discussed in chapter 6; for a comprehensive overview of S-Systems and biochemical systems applications the reader is referred to Voit (2000).

Ultimately the type of model selected must be able to address the experimental questions of interest. For this it is important to consider issues of complexity, tractability, interpretability, level of detail and computational cost (Voit 2000:1-10; Wessels *et al.* 2001).

2.3.2 Computational Tools for Modeling Biological Systems

Many computational tools have been developed which allow the simulation of biochemical systems. Usually these tools implement the reactions as systems of differential equations which are solved through numerical integration methods.

Some of the most extensively used tools include E-Cell (Tomita *et al.* 1999), Gepasi (Mendes 1993), GENESIS (Bower & Beeman 1998) with Kinetikit (Bhalla 2002) and Jarnac with JDesigner (Sauro 2000; Sauro 2001). All these different tools have strong and weak points; Pettinen *et al.* (2005) compare the efficiency and usability of some of the different tools. Their comparison study highlights some general important aspects for biochemical simulation tools:

1. Usability – it is paramount that the tools offer a well developed GUI (Graphical User Interface) which allows visual construction of pathways and analysis/visualization of results via graphs.
2. Import/export of models – adoption of a common format as SBML becomes evident if these tools are to find widespread practical usage.
3. Parameter Estimation – support for estimation of model parameter values.

Biological systems modeling is under constant development; with new algorithms constantly being introduced and new computational tools being developed. Chapter 6 introduces our contribution to the area with a novel computational tool based on S-Systems for simulations of dynamic systems.

2.4 Microarrays

In the above sections the importance of high-throughput techniques was highlighted. Microarrays, alongside protein chips, RT-PCR, RNAi knockout and other methods, revolutionized the ability of generating high quantities of system-wide, parallelized data. The widespread adoption of microarrays as the tool of choice for gene expression studies has produced what is currently the greatest volume of data available for system level studies. It is therefore only fair that we dedicate a brief section to introduce this powerful technique. Chapter 5 is dedicated to the optimization of microarray experimental designs.

The seminal paper in microarrays (Schena *et al.* 1995) introduced the concept of a massively parallel version of the Northern and Southern blot technique; allowing for thousands of hybridizations at the same time. With microarrays, instead of studying a single gene and mRNA at a time, researchers can often simultaneously study all known mRNA of a tissue. This opened the door to compare all gene expression levels between normal and tumour cells, evaluate expression levels of different tissues of an organism, study population variability, genotyping, test drug resistance and test environmental effects of gene expression levels. An excellent overview of microarray technologies and applications is presented in the *The Chipping Forecast* (1999) and *The Chipping Forecast II* (2002).

Essentially, microarrays consist of a substrate to which are adhered thousands of probes (oligonucleotides, cDNA) separated from each other by micrometric distances. The target sample will hybridize to the probe if they are complementary. Interpretation of the microarray experiment is made possible by labelling the target with a fluorescent dye or a radioactive element which can be detected and quantified quantitatively or semi-quantitatively (Schena 1999).

2.4.1 Types of Microarrays

There are two major microarray technologies in common usage; the GeneChip system from Affymetrix which is a proprietary system, and spotted arrays on a glass slide. A third less common method is Serial Analysis of Gene Expression (SAGE). Since in this thesis only spotted arrays (chapters 5 and 6) are addressed, the discussion is centred on this technology.

SAGE is run on standard DNA sequencing equipment and mRNA is measured by sequencing concatenated fragments of their respective cDNA. The count of these cDNA fragments corresponds to the abundance of mRNA in the sample (Velculescu, *et al.* 1995; Powell 2000; Yamamoto *et al.* 2001; Oien 2003).

The proprietary GeneChip from Affymetrix uses a photolithographic process analogous to computer silicon chip production. Masks are used to control the light-driven synthesis of oligonucleotides ('oligos') on the chip surface. Each oligo is up to 25 nucleotides long and there are up to 40 oligos to detect a gene product on each chip. Around half the oligos are perfect matches to characteristic regions of a gene; the other half uses an oligo with a mismatch in position 13 which allegedly allows detection of nonspecific hybridization and experimental noise (Lockhart *et al.* 1996; Lipshutz *et al.* 1999).

Spotted arrays give the relative abundance of two target sets of mRNA which compete to hybridize with the probes. These (cDNA, PCR products or oligonucleotides) are spotted on the array using a robot which transfers a small sample from a microtiter plate onto a polylysine coated glass plate. Spotted arrays are not as uniform as Affymetrix chips but they allow for greater flexibility since any probe can be designed to spot on the array, this is particularly important for organisms which Affymetrix does not cover. Cost-wise spotted arrays are cheaper than GeneChips but a probe library is costly and time consuming to construct.

The most common spotted array experiments consist of a competitive hybridization between two samples with each one tagged with a different fluorophore using the ubiquitous fluorophores cyanine-3 (Cy3) and cyanine-5 (Cy5) from Amersham. The slide is read by a scanner which uses lasers to excite the fluorophores (absorption of Cy3 – 550nm and Cy5 – 649nm) and a CCD converts the analog light signal of the fluorophores into a digital signal of light intensities (emission Cy3 – 570nm and Cy5 – 670nm) which are saved in TIFF graphical format. Since the hybridization is essentially competitive, the sample with a higher expression of mRNA will attach more to its probe and this is detected by the intensity of emission of the fluorophores. Scanned images can be viewed as an overlay of the two intensity files (Green – Cy3, Red – Cy5, each color is referred to as a channel) with the probes that did not hybridize in black (no expression in both samples), the probes in which the two samples have similar expression levels in yellow and variants from pure green (only the Cy3 labeled sample is expressed) to pure red (only Cy5). For a comprehensive review of spotted arrays refer to Schena (2002) and Schena (1999).

2.4.2 Analysis of DNA Microarrays

Data analysis of microarrays, due to its dimensionality and many sources of error, is a complex and still very open problem, with the scientific community actively researching the best methods to analyse array data. Data analysis of SAGE data, spotted arrays and Affymetrix chips is similar. The main difference between spotted arrays and GeneChips is that the latter handle a single sample at a time, whereas spotted arrays handle two samples on the same slide (Knudsen 2002:1-14).

Briefly, data analysis involves spot recognition from the scanned images and evaluation of fore and background intensities (Yang *et al.* 2001), normalization to remove dye effects – Cy3 and Cy5 fluoresce with different intensities – and other sources of variation (Quackenbush 2002; Schuchhardt *et al.* 2000; Smyth & Speed 2003; Yang *et al.* 2002), a statistical test to detect differentially expressed genes (Nadon & Shoemaker 2002; Dudoit *et al.* 2002), ANOVA is growing in popularity (Kerr *et al.* 2000; Kerr *et al.* 2002); and, cluster analysis to group genes with similar expression patterns, typically in time-series experiments (Sherlock 2000; Jiang *et al.* 2004). Knudsen (2002), Draghici (2003) and Hegde *et al.* (2000) offer a good overview of microarray analysis. The annual CAMDA (Critical Assessment of Microarray Data Analysis) Conference is solely dedicated to the various techniques of microarray data analysis (the proceedings are available from Springer-Verlag). A multitude of analysis software is available either commercially or as free software. Of notice is Bioconductor which runs under the statistical package R (Gentleman *et al.* 2004), an open-source and community driven package which is growing in popularity due to being open-source – researchers can freely modify a method – freely available and, as new methods are published they are rapidly implemented.

On the other side of the spectrum lies the experimental design of microarrays. With the enthusiasm generated by the new technology and researchers overwhelmed by the possibility of generating huge datasets, little thought was initially given to experimental design. It is now clear that the classic approach of a well defined experimental question and a well planned design are crucial (Simon *et al.* 2002). The descriptive view of microarrays considers the technology as a tool to survey gene expression, this approach usually does not address an experimental question and may result in data that has little practical value or is biased, impairing the analysis (Simon *et al.* 2002; Yang & Speed 2002). Further, due to high experimental costs, an effective design is important to maximize material and labor efficiency. Chapter 5 further discusses experimental design issues with spotted arrays and presents an Evolutionary Computation approach to optimization of experimental designs.

2.5 Evolutionary Computation

The previous sections discussed how biology is becoming an information-centred science, the importance of a holistic approach to understand biological phenomena, different modelling techniques that aid in constructing this system view of biology, and microarrays as an important source of biological data. In this section we review the fundamentals of Evolutionary Computation (EC), a powerful method to address the issues of our data saturated and, paradoxically information restricted, contemporary biology.

Evolutionary Computation is about solving problems. For some relatively simple problems, closed-form optimal solutions can be calculated directly. For other classes of well defined problems, deterministic algorithms guaranteed to converge on optimal solutions are available. An example of such an algorithm is exhaustive search, in which the entire solution space is evaluated and the optimal solution selected. This class of problems is the exception rather than the rule; especially for biological problems.

Except for the most trivial problems the search space is so large that exhaustive search is prohibitive. Another scenario is the class of ill-defined problems. An optimal solution implies absolute knowledge of the problem; frequently knowledge of the problem's domain is incomplete and sparse, the solution space is unknown and data sources are noisy. In other types of problems optimality may vary over time or there are multiple optimal solutions. And, sometimes problems are heavily constrained and simply finding a single set of parameters that comply with the bounds is unwieldy (Michalewicz & Fogel 2000). Biological problems fall under all these categories, they have large search spaces, are non-linear, complex and open-ended, available data is noisy and incomplete and, they are interlinked and constrained by the components of the system.

For such complex problems a compromise must be sought. Instead of deterministic algorithms which yield optimal solutions but can only be used in specific and frequently unrealistic problems, stochastic algorithms that are not mathematically guaranteed to converge on a optimal solution can be adopted (Michalewicz & Fogel 2000). Evolutionary Computation falls under this last category. EC is the general umbrella for a group of stochastic problem solving methods loosely inspired on evolutionary processes such as selection, mutation and recombination. EC methods are commonly referred to as Evolutionary Algorithms (EA). All algorithms have in common the use of populations of candidate solutions which reproduce, compete, and are subjected to selective pressures and random variation – the four basic elements of evolution (Atmar 1994).

2.5.1 Evolution of Evolutionary Computation

EC, alongside Fuzzy Systems and Artificial Neural Networks is a subfield of Computational Intelligence and the umbrella under which reside closely related stochastic methods of simulating evolution (Schwefel et al. 2003). The main branches of EC are Evolutionary Programming (EP), Evolution Strategies (ES), Genetic Algorithms (GA), Genetic Programming (GP) and Learning Classifier Systems (LCS); see section 2.5.4 for a description of the different types of EC.

EC is a young field; the term was coined in 1991 in an attempt to unite the different branches. The origins of using computers as a means to emulate and understand evolution dates back to the late 1950s and early 60s through the pioneering work of Bremermann, Fraser and Friedberg. The different EC branches evolved quite independently from each other, despite having much in common. The current trend is to merge the best aspects of the different branches and develop solid theoretical foundations for the entire field (De Jong *et al.* 2000).

2.5.2 Overview

Evolution can be seen as a dynamic and opportunistic optimization process. Effectively it is a method to search through a vast solution space and find a solution that allows organisms to survive and reproduce in a certain environment. It is dynamic in the sense that solutions (organisms) can change to adapt to environmental changes and it is opportunistic in the sense that solutions are not necessarily globally optimal but rather tend to move to the next available solution that ensures viability, even if in detriment of a more globally optimal solution. Interestingly enough, the high-level rules that govern evolution and account for the great variability of organisms are quite straightforward. Organisms – which can be seen as candidate solutions – evolve through random variation due to mutation, recombination and manipulations on their genetic material; these candidates are subjected to selective pressures which

evaluate their adaptiveness and determine their capacity of generating descendants, thus propagating better fit genotypes into the future generations. These characteristics are the inspiration of Evolutionary Computation. In a nutshell, EC tries to mimic the mechanisms of biological evolution to solve complex problems (Mitchell & Taylor 1999; Fogel 2000a; Fogel 2000b).

Even though specific implementations can vary significantly and algorithms are not constrained to using only biological mechanisms, there are three common features which are shared by the different branches of EC (Mitchell & Taylor 1999; Bäck 2000):

1. A population. A number (*n*) of candidate solutions (representations of the problem) compete against each other to remain in the population and generate offspring. Since ECs use populations, they can be seen as a parallelized search of the solution space (see section 2.5.5 for population representations).
2. Selection. Organisms from the population pool are selected for culling or reproduction based on their fitness. Fitness is a function measurement of how 'good' a representation is at solving the problem. The two most adopted methods for assigning fitness are as a direct mapping to the problem or as a relative measurement of performance in relation to the remainder of the population. Arguably, the choice of a fitness function that clearly states the problem is the most important step in determining the success or failure of the EC algorithm (see section 2.5.6 for a discussion of selection methods and fitness mapping).
3. Search operators. EC uses stochastic methods to solve a problem; these biologically inspired operators provide the variability necessary for the EC population to explore different areas of the solution space. The two main sources of variability are mutation, which are randomly generated new sources of variability (see section 2.5.8) and recombination, which exploits the available variability within the population to form new combinations of candidate solutions (see section 2.5.7).

Thus, a general EC algorithm combines these features and through iterations improves the overall fitness of the population, gradually converging on a solution. The following steps form the general structure of an EC algorithm:

1. Create an initial population – randomly or based on prior information;
2. Assign a fitness value to all organisms (also referred to as chromosomes);
3. Select organisms for reproduction based on their fitness and a selection scheme;
4. Create descendants from the selected parents;
5. Modify the descendants with the search operators;
6. Evaluate the fitness of the descendants;
7. Cull organisms from the parental population and replace them with the descendants according to the selection scheme;
8. Repeat from step 3 until a termination criterion is met, for example, a specified number of iterations or a predefined fitness value is reached.

2.5.3 Pros and Cons

There are different classes of optimization problems: assignment, model parameterization, model discovery, combinatorial and minimax, just to name a few. Selection of the best method for a certain optimization problem is not trivial. The no-free-lunch (NFL) theorem (Wolpert & Macready 1996)

essentially states that there is no optimal method for solving all types of optimization problems. A method adequate for a certain class of problems may breakdown with a different problem. A wide range of global optimum finding methods such as dynamic programming and linear programming are available for specific types of optimization problems, these are traditional methods that are guaranteed to converge on the global optimum.

Traditional methods can be split into two main classes, the algorithms that evaluate complete solutions and the algorithms that evaluate partial solutions (Michalewicz & Fogel 2000:55-109). In the first class are exhaustive search, gradient methods and linear programming. These can be applied to specific domains; for smooth differentiable problems a gradient approach is suitable, linear programming is well tailored for a problem of linear variables. For a small confined search space exhaustive search is adequate. The second class of methods evaluates partial solutions and builds on them; these include greedy algorithms, dynamic programming and branch-and-bound algorithms. As an example dynamic programming is efficient in pair-wise alignment of amino acid or nucleotide sequences.

Regrettably most of the above algorithms are confined to particular classes of problems which frequently cannot be applied to real-world problems. Real-world optimization problems tend to exist in a non-linear, discontinuous and complex landscape. Further, the systems can be dynamic with objectives shifting over time thus demanding that the optimization method be capable of modifying its strategy *in situ*. Other common difficulties are the presence of noise in the available information and lack of knowledge about the nature of the problem itself (Michalewicz & Fogel 2000:11-30). These difficulties can hinder the adoption of global optimum finding algorithms, even though they have been successfully adopted to solve a wide range of specific problems.

EC algorithms offer a compromise solution. Whilst not guaranteed to converge on optimal solutions they can be used on a wider range of problems albeit at the sacrifice of efficiency. This loss of efficiency is particularly evident in simple scenarios for which specifically tailored optimum seeking algorithms are available. Thus, classes of problems for which specific algorithms are available should not be solved through ECs for they will not be able to solve these problems faster or with higher precision (Schwefel 2000).

A common approach to solving a complex problem is to linearize it or make use of some other simplification method that will allow the problem to be solved by an available optimum seeking method. However these simplifications can yield results that are further away from the true answer than an approximate result obtained through an EC approach. Even with quite incorrect initial conditions, an EC approach can still produce reasonable results (Schwefel 2000).

EC methods are not guaranteed to converge on an optimal solution. But worse, it is also not possible to evaluate how close to optimal a result is. Attempts to develop a formal analytical framework for EC are still incipient, mainly due to the fact that EC methods present the same analytical difficulties of the problems they are used to solve – complex, noisy, dynamic and non-linear (Forrest 1993).

Probably the greatest limitation to the use of EC methods is the dimensionality problem. As the number of variables increases the computational effort can increase exponentially. But, since by their very nature EC methods are well suited for parallelization and there is a growing interest in developing parallel algorithms, the importance of this problem is becoming less significant (Alba & Tomassini 2002).

In summary, real-world problems are complex and demand flexible tools that can be adapted to the nature of the problem and not tools that force the problem to adapt itself to them. EC provides a powerful framework for solving these problems.

2.5.4 Types of Evolutionary Computation

2.5.4.1 Evolutionary Programming (EP)

EP in its basic form consists of generating an initial population μ and a fitness value is assigned to each individual. The iterative loop (each loop is commonly referred to as a generation) usually consists of duplicating each parent μ_i until a predefined number λ_i of offspring are generated. The offspring are modified through a mutation process – commonly a Gaussian distribution with zero mean and variance of one, recombination is not used in classic EP. All offspring are evaluated as to their fitness and along with the parental population a selection operator is used to cull the population size back to μ.

EP shares more than a few similarities with ES (see section 2.5.4.2). The main difference between EP and other EC methods is the global optimization method employed by EP. No attempt is made to break the problem down into subcomponents; the fitness evaluation is based solely on the whole organism. In this sense the genotype is of little importance and focus is on optimization of the phenotype, for this reason recombination is not used in EP. A further distinction is that traditionally EP uses continuous-valued variables instead of the discrete representation common in GA.

Current versions of EP are self-adaptive, with the mutation parameters (variance, covariance) adapting to the current state of the population. An overview of EP is given by Bäck (1996), Fogel (1999) and Porto (2000).

2.5.4.2 Evolution Strategies (ES)

ES were initially developed to solve technical optimization problems. There are two main general notations for the strategy: $(\mu + \lambda)$ where the ES generates λ offspring from a parental population μ and selects the best μ from all $\mu+\lambda$ individuals. Alternatively the (μ , λ) strategy generates λ offspring from μ parents and selects the μ best from the λ offspring. Weak selective pressures seem to yield a better response thus the μ/λ ratio should not be too small. Of course, a *1:1* mapping of $\mu:\lambda$ reduces the algorithm to a random walk.

Typically ES use recombination between two randomly selected parents to generate the offspring; commonly adopted is the multipoint crossover (2.5.7). After recombination the offspring are mutated. ES mutation schemes are not particularly straightforward; each organism, besides the element that maps their position in the search space, can have several parameters controlling the mutation distribution which customarily follows a multivariate normal distribution with zero mean and a covariance matrix that is symmetric and positive definite. At least two mutation parameters are commonly used: angles (σ) and standard deviations (ω). These mutation parameters can be self-adaptive as in EP algorithms.

The original ES strategy was $(\mu + 1)$ with a single replacement per iteration loop which is called steady-state (an analogy to overlapping generations) in opposition to the generational approach. Even though the steady-state approach is the preferred choice for other EC methods, modern ES adopt a generational approach similar to EP.

As with Evolutionary Programming, ES does not attempt to break down the problem into smaller subcomponents. Optimization is solely based on the phenotypic values of the organism. Schwefel and Rudolph (1995) and Rudolph (2000), provide an overview of ES.

2.5.4.3 Genetic Algorithms (GAs)

The most widely disseminated EC branch, GAs date back to Holland's (1975) seminal work. GAs distinguish themselves from the other methods by the emphasis that is placed on recombination. Traditionally GA organisms are represented as linear bitstrings which are referred to as chromosomes; this is the canonical GA (Holland 1975; Goldberg 1987). The value in each position of the bitstring is an allele (0 or 1) and the position itself is a gene or locus. The combination of values (alleles) in the bitstring (chromosome) maps to a phenotypic expression, such as a parameter to be optimized. From the above it is clear that GAs operate at two structural levels: a genotypic and a phenotypic one. Selection operators are carried out based on the overall chromosome value (phenotype) while search operators act on the genotype, modifying the chromosome which may or may not change the phenotypic expression.

GAs are the class of EC which most closely mimic evolutionary processes at a genetic level. Recombination swaps chromosome parts between parents to form the offspring and mutation changes the value of alleles at randomly selected loci. From this notion derives the concept of schema in GAs (Holland 1975); a good solution consists of a set of good small building blocks. Thus, the assumption is that the chromosomes in the population are formed by small schemas that add up to yield the final fitness. Under the schema model, recombinations between good schemas are preserved since they increase the overall fitness of the chromosome while recombinations which break up a good schema are eliminated since they reduce the fitness. This assumes that the GA will concentrate on searches which are optimally allocated. The schema theory has been under attack recently, with many arguments for and against but still limited solid proofs (Whitley 2001; Langdon & Poli 2002). To the author's knowledge no work has attempted to describe schema theory within the framework of quantitative genetics; such a study might evidence the appearance and maintenance of haplotypes and linkage disequilibrium studies could provide a more in-depth understanding of the dynamics of GAs.

Recombination is often regarded as the main search operator of a GA, with mutation seen more as a mechanism of ensuring a robust gene pool to be explored by recombination (Eshelman 2000).

2.5.4.4 Genetic Programming (GP)

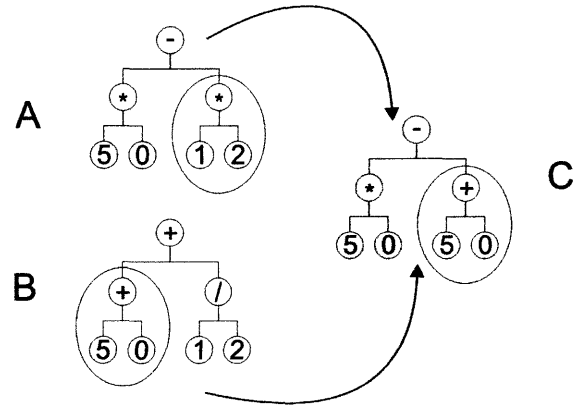
Often regarded as a specialization of Genetic Algorithms, GP has evolved to become a branch of EC in its own right. Initially GP was devised as a method to optimize data structures as executable computer programs with the fitness value assigned based on the results obtained when executing the instructions contained in each member of the population. In this context, GP evolves populations of computer programs or other algorithmic processes to solve a specific problem (Banzhaf *et al.* 1998; Koza 1989; Koza 1992; Koza 1994; Koza *et al* 1999; Koza *et al* 2003).

Original implementations of GP used tree-structured representations implemented in LISP. Tree-structures have the terminal nodes of the tree containing inputs (referred to as terminals) and the internal nodes holding functions. This type of construct demands significant overhead to ensure viability of the trees (handle, for instance, division by zero or infinite loops) or correct tree structures which can break-up due to mutation and recombination (Heywood & Zincir-Heywood 2000; Banzhaf *et al.* 1998:143-170).

Currently LISP implementations are rarely used and alternative language implementations such as C++ (Keith & Martin 1994) are used. The same applies to new GP structures developed to improve on the original Tree-GP. Alternative methods are numerous and include Linear-tree Genetic Programming (Kantschik & Banzhaf 2001), Linear Genetic Programming (Nordin 1994; Banzhaf *et al.* 1998:243-265), Linear Page-based Genetic Programming (Heywood & Zincir-Heywood 2000) and Gene Expression Programming (Ferreira 2001).

GP uses recombination and mutation in a similar fashion as GAs. Commonly recombination involves swapping entire subtrees between parents. Figure 2.1 illustrates a typical tree GP and recombination between two parents. Mutation replaces a subtree with a random subtree of a randomly defined depth. GP structures use variable-length representations which have a tendency to grow in tree depth through the addition of subtrees of functions and terminals which contribute only slightly to improve the fitness. This growth is referred to as bloat and is an active field of research in the GP community. A Common practice is to place a cap on tree depth (Langdon & Poli 2002).

Figure 2.1 Recombination in a tree GP. Recombination between parents A and B generate offspring C, the function set is {*,+,/,-} and the terminal set is {2,0,1,5}. The trees code for A=(5*0)-(1*2); B=(5+0)+(1/2); C=(5*0)-(5+0).



2.5.4.5 Learning Classifier Systems (LCS)

No LCS were used in this thesis so they are only briefly overviewed. The interested reader is referred to Wilson (1994), Smith (2000) and to Vlachos *et al.* (2004) for an application of LCS in biology. But in short, LCS are more of a concept than an actual algorithm; they can be seen as a hybrid between a machine learning technique and a GA. LCS is a rule-based system which uses agents that receive an external signal (message), process it and produce a response. The genetic material consists of a set of rules which map an input (detector) to an output (effector), modifying the behaviour of the agents to environmental changes. The GA component of the LCS is to adapt the rules to obtain a desired behaviour from the agents.

2.5.4.6 Notes on Evolutionary Algorithms (EA)

The NFL theorem applies to EA. No single approach is always superior for all problems or can solve any type of problem. The choice of an appropriate EA depends on the nature of the problem at hand. There have been advances in developing a formal framework for EC but largely the field is still anchored on a trial-and-error approach. There are no widely applicable rules for selection of population parameters apart from the collective empirical experience of practitioners (Banzhaf *et al.* 1998:334-338).

On the bright side, the methods are robust and even suboptimal parameter selection can still lead to good results.

As a rule of thumb, GAs are well suited for discrete problems such as sorting, ranking or allocation problems; EP and ES are a good first choice for continuous problems such as model parameterization; GP allows tackling problems such as model discovery.

Within each EC branch there is vast number of different algorithms. Selecting the best one for a given task can be quite daunting. From a practical standpoint considerations of ease of implementation, computational and convergence speeds and repeatability of results are important.

2.5.5 Computational Representation and Implementation

An appropriate choice of representation for the populations is crucial for EA and largely depends on the nature of the problem. A parameterization problem is usually represented as a real-valued vector; if using an ES or EP the vector consists of the solution vector and variability parameters. Finite-state representations are also frequent with EP. A GA classically uses binary strings. GP has to store information on the functions, the terminals and the relations between the two; lists, stacks, parse-trees and vectors are commonly used (Bäck *et al.* 2000:127-165). Over the following chapters, examples of different representations are shown that are best suited to the nature of the particular problem.

The choice of programming language is of secondary importance to the algorithms and they can usually be easily ported between languages. In this thesis all software applications were developed in Microsoft C# and run under the Microsoft.Net platform. Without swimming in the dangerous waters of defending a specific language, C# was chosen for being a modern fully object-oriented language which allows rapid development of GUIs and algorithms with a low overhead. On the downside C# is slower than either Fortran or C++ under the latest release of the .Net platform. Gilani (2004) presents further arguments for adoption of C# in scientific computing.

2.5.6 Selection

Selection is an integral component of all EC methods. Through selection, solutions with a higher fitness are emphasized in the population. There are several selection operators (Bäck *et al.* 2000a:166-234) but all essentially select better solutions for reproduction and delete less fit solutions which are replaced by the offspring of the better performing ones. Selection does not generate new solutions; it simply directs the evolution of the population. Of notice is that not only the best organisms are always selected; the process is stochastic, which *can* allow inferior solutions to be selected over better ones with a low probability. This preserves the diversity of the population and avoids a premature convergence on local optima.

The main selection methods are proportionate, rank-based, Boltzmann and tournament. Proportionate selection assigns a probability of generating offspring based on the relative fitness of the organism. The simplest form of proportionate selection is roulette wheel; where each solution is assigned an area in the wheel proportional to its fitness – fitter organisms have a bigger area and consequently a higher probability that the wheel when spun will stop in their area. Rank-based selection ranks the entire population based on their fitness and then assigns a selection probability based on these ranked values. Boltzmann selection uses a probability distribution with a T term similar to the temperature term in the Boltzmann distribution which decreases as the iterations progress; initially all solutions have similar

chances of being selected since a large T is used but as T reduces the stringency increases and only better solutions are chosen. Tournament selection chooses a certain sample size from the population to compete and the ones with the highest fitness are selected; the selective pressure is defined by the size of the tournament.

Tournament selection is rapidly becoming the selection method of choice for EC applications. There is no need to evaluate the entire population or maintain population statistics which makes the selection process faster. For the same reasons it is also well suited for parallel implementations. The major drawback of roulette wheel is avoided; in which the size of the areas rapidly become the same as the population converges on a solution, forcing the use of a fitness scaling mechanism between the upper and lower limits of the fitness range. Tournament selection is inherently noisy and can rapidly lead to a loss of diversity. To counterbalance this effect small tournaments are preferred in association with slightly higher mutation rates. Hancock (2000) presents a comparison of the different selection mechanisms.

EC uses two generational structures: steady-state and generational. The steady-state uses an overlapping generation approach in which parents and offspring simultaneously compete in the population. Tournament selection is typically steady-state with only a few new organisms in each generation. The generational approach uses non-overlapping populations with the offspring entirely replacing the parental population. Steady-state runs tend to have a higher variance, thus in small populations the effect of drift in more pronounced and can lead to the loss of variability. To counteract this effect, bigger populations should be used in steady-state systems (De Jong & Sarma 1993).

Selection operators act on the fitness of the organisms. Fitness is arguably the most important aspect of any EA, if the fitness function is not well constructed the whole EA will breakdown. The fitness function can be seen as a measure of the probability that an organism will survive and reproduce in the population. The selection scheme and the fitness function are inextricably connected. A good fitness function should allow for a range of intermediary values which can be explored by the EA. All-or-nothing fitness functions are ineffective for an EA which must be capable of evaluating if a certain solution is better or worse; the less granular the fitness function, the higher the probability that the EA will converge on an adequate solution. At this point it is important to highlight the distinction between fitness and objective function. The fitness function maps to the objective function which is external to the EC and depends on the nature of the problem. The terms fitness and objective function are frequently used as synonyms, especially when the mapping between the fitness function and the objective function is 1:1. On the other hand, if a population scaling scheme is adopted it is quite clear that fitness and objective function are necessarily distinct. In summary, the objective function assigns a value to an organism which can be directly translated as its fitness or which can be mapped to a fitness value based on the fitness function of the EA.

In multi-objective optimization the fitness function is even more critical as there usually is no unique solution to a problem 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). More formally a solution is Pareto optimal if there is no feasible set of variables which would improve a criterion without simultaneously decreasing at least one other criterion. An example of such a problem is an optimization problem in microarrays in which a balance must be found between the number of slides (cost constraints)

and 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 (see chapter 5). 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 EA in the same fashion as mutation parameters in ES – to a user-defined approach where the user modifies weights based on personal preferences. In this case, weightings can be varied in the light of the response surface of component outcomes generated during analysis – the best direction to take depends on how far can be gone in each direction. The latter is used in Total Genetic Resource Management (TGRM), a tool for optimization of animal breeding programs, where an animal breeder can define weights and constraints interactively (Kinghorn 2000).

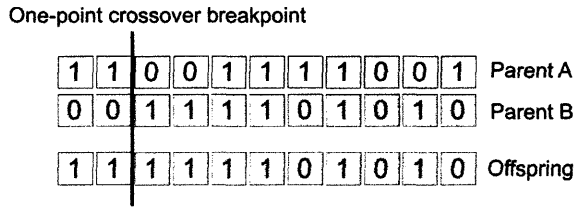
2.5.7 Recombination

Recombination is a search operator that does not generate new sources of variability in the populations albeit introducing new variation. It operates by combining parts from two or more parents to generate one or more offspring. The drive behind recombination is to generate new variability in the population by manipulating the component sources of variation to explore new combinations which might be better solutions to the problem.

There are many different recombination algorithms depending on the EC method and the representation of the problem: binary strings, real-valued vectors, finite-state machines or parse trees. Booker et al. (2000) reviewed different recombination methods. The most straightforward recombination is used in canonical GA and is a good model to illustrate the principle. Figure 2.2 illustrates a one-point crossover in a binary GA. Briefly, two parents are selected for recombination, a breakpoint in the chromosome is randomly determined, and from the breakpoint onwards the two chromosomes swap the remainder of their bitstrings.

Figure 2.2 depicts how recombination can produce an offspring with a higher fitness than the parents. Consider that the trivial objective function of a GA is to maximize the number of *ones* in the binary string. The parents have respectively seven and six *ones* in their strings, in the example their offspring has eight *ones*, after recombination.

Figure 2.2 One-point crossover in a binary genetic algorithm. A breakpoint is randomly selected and the two chromosomes swap bitstrings after the breakpoint. Recombination is a search operator which explores available population variability by testing new combinations. No new allelic variability is generated through recombination but it does generate new variation in fitness values.



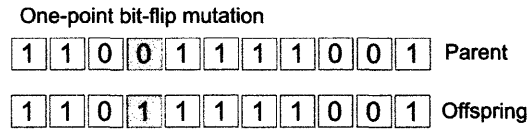
2.5.8 Mutation

In contrast to recombination, mutation generates new allelic variability in the population. The general principle is that new offspring are created by a stochastic change to a single parent. Like recombination there is a plethora of mutation algorithms for the different EAs (Bäck *et al.* 2000c). For

example, a real-valued EA would change the value at a selected allele with a new randomly selected value.

Again, the canonical GA is the most pictorial example of mutation. Figure 2.3 shows a point-mutation bit-flip in which an allele of a parent is randomly selected to be flipped. The most common approach is to assign a small uniform probability that mutation will occur and test each position of the bitstring; if the mutation operator returns true the bit at the position is flipped. Notice that mutation can also produce an offspring with a higher fitness than the parent. After mutation the offspring has eight *ones* instead of seven.

Figure 2.3 Point-mutation bit-flip in a binary genetic algorithm. New offspring are produced by a random change to the parent. In the example allele at position 4 mutated and was flipped from zero to one. Mutation is a source of new variability in a population.



Mutation is an important operator to generate new sources of variability and expose new areas of the solution landscape whilst recombination can only shuffle available variability. Consider that in the example in figure 2.3 no member of the population had a *one* in allele four; no amount of recombination would generate a solution with *one* in this position, thus an optimal solution would be unobtainable.

Mutation and recombination are the main search operators used in EC. Frequently both are used in an EA and the parameter settings for these operators are critical for a successful run. High mutation rates can reduce the method to a random search. If too low, there will be little or loss of variability in the population. The same applies to recombination, if too high good constructs will be broken up. If too low there will be little exploration of the search space. A balance always has to be achieved between the two search operators as well as selective pressure and population size, which are the four main parameters in an EA (Banzhaf *et al.* 1998:334-338).

2.5.9 Applications in Bioinformatics

Evolutionary Computation is being widely employed to solve bioinformatics problems. And that is not particularly surprising; bioinformatics problems are complex, noisy and non-linear – the perfect setting for Evolutionary Computation to thrive. Some of the current efforts include sequence reconstruction from shotgun sequencing data, multiple-sequence alignment of protein or DNA sequences, tertiary protein folding inference, identification of coding regions in DNA sequences, microarray data clustering and reconstruction of metabolic and genetic pathways. Fogel and Corne (2003) provide a comprehensive review of the current research topics in EC applied to Bioinformatics.

2.5.10 Summary

In a nutshell, Evolutionary Computation uses computer algorithms which search through complex solution landscapes. A population of candidate solutions is created, a fitness value is assigned to each member of the population and depending on their fitness value an organism has a higher or lower probability of being selected to remain in the population and generating offspring. New members are created through recombination and mutation thus exploring the solution space.

This brief overview of EC aimed at providing the reader with a general idea of how Evolutionary Algorithms work, what are their applications, the main types of EC and the operators that drive the processes involved. At this point it must be clear that it is a broad field and that we have merely glimpsed the tip of the iceberg; nevertheless it should suffice to keep the reader afloat over the next five chapters where different EC methods are applied to solve biological problems. As each new method is introduced it is discussed in greater depth. In the last chapter the arguments for adopting each specific method are highlighted.

2.6 Conclusion

Some of the most interesting challenges of modern biology involve solving complex problems; this is fertile ground for Evolutionary Computation. The following chapters address some of these challenges. From practical real-world applications such as optimization of diets for beef cattle, day to day research tasks such as the multiple alignment of sequences and the optimization of experimental design for microarrays all the way through to basic innovative research in parameterization and reconstruction of biochemical pathways and genetic networks, EC offers powerful and flexible methods to address these topics.

Over the next decades the integration of biology with mathematics and computer science will continue evolving and ever more exciting new discoveries will result from this partnership. Evolutionary Computation will definitely be a key player in some of these discoveries.

3 Nutrition Parameter Optimization in Beef Cattle Using Differential Evolution

3.1 Introduction

Several mathematical models have been developed to help predict nutritional requirements, feed utilization and weight gain in beef cattle (eg. NRC 2000; INRA 1989; SCA 1990; AFRC 1993; Fox *et al.* 2004). These models are economically important as they can help reduce cost of production, reduce waste and assist in making production decisions. They are generally deterministic models that depend on extensive initial parameter sets for the cattle traits such as those used in the SCA (1990) model which includes information on breeds, initial and standard weights ('the base weight of an animal when skeletal development is complete and condition score is in the middle of the range' (SCA 1990), as well as feed information such as daily intake, type of feed supplied (pasture, roughage, concentrates) and feed specifications. Outputs of these models are in the form of, for example, required and supplied metabolizable energy and/or protein, daily weight gain and feed intake predictions. These models have proven to be adequate predictors of the above mentioned outputs and valuable computational tools for cattle producers have been developed (Freer *et al.* 1997). The main drawback of these models is that they rely on a well known set of parameters that not infrequently are only partially defined. They may also not be applicable to certain management conditions, for instance the AFRC (1993) system is not adequate for grazing animals.

Given a well defined set of initial parameters, it is relatively straightforward to implement a model and calculate its outputs from those parameters. The inverse is not true. To find a set or subset of input parameters for a target output is not a trivial exercise. To illustrate the concept in the first scenario, given a specific quantity of feed, a model can be used to predict the daily weight gain of the cattle. The inverse problem is to calculate the quantity of feed required to meet a target daily weight gain. This problem is commonly solved through an iterative process (McDonald *et al.* 2002:296) such as Linear Programming (LP).

Linear Programming is a deterministic optimization method widely adopted in economics and engineering to find the maximum of a linear combination of variables constrained by also linear equalities or inequalities. The first and most used method for solving linear programs is the simplex method (Michalewicz & Fogel 2000:76-80). LP has been widely used for ration formulation (Glen 1980; Crabtree 1982; Van de Haar & Black 1991; Martinez *et al.* 1998), optimization of the least cost diet formulations (Tedeschi *et al.* 2000), feed allocation across a herd (Wang *et al.* 2000a), reduction of Nitrogen and Phosphorus mass balance on a farm (Wang *et al.* 2000b) and comparisons of nutritional management strategies (Nicholson *et al.* 1994). LP is an efficient approach when there is a single objective, for instance, minimize the cost of ration. The problem becomes more complicated when there is more than one objective, for instance to balance nutrition requirements (eg. required and supplied daily metabolizable energy) and achieve a target daily weight gain. When there are several goals to optimize, traditional LP can result in infeasible solutions. Recently multiple-objective programming (MOP) has been used for solving multiple objective nutrition problems in broilers (Zhang & Roush 2002). MOP is a more flexible alternative to LP that transforms inequalities into constraints and uses deviation variables, which

are the optimization target, in the objective function. The downside of these methods is that for practical systems they can become computationally too intensive and in complex heterogeneous systems the simplified assumptions used to structure the problem can lead to wrong solutions since the model of the problem is a poor representation of the *real* process; or as Michalewicz and Fogel (2000) put it "the right answer to the wrong question". Computationally, LP and MOP are both complex to implement and specific details vary with the type of problem.

Alternatively, the model can be algebraically inverted in the sense that the inputs become the outputs and vice-versa. This method is applicable for simple models with a unique output for any given set of inputs (Doeschel-Wilson *et al.* 2006). But model inversion is usually mathematically intractable since many parameter sets can generate the same output. In addition, selection of optimal parameters for multiple targets (multicriteria objectives) is generally not possible since the objectives can be imperfectly correlated, thus solutions tend to present themselves as Pareto fronts which are functions of the weights attributed to each criterion.

A third approach to model inversion is through the use of stochastic iterative methods including direct search methods such as the simplex method (Nelder & Mead 1965) which must not be confused with the simplex algorithm used in LP, hill climbing, tabu search, and simulated annealing. These have the advantage of not being limited to linear problems and are capable of exploring multidimensional nonlinear solution spaces as well as multiple objective problems. For complex agricultural models evolutionary algorithms perform better than direct-search methods, tabu search and simulated annealing (Mayer *et al.* 1996; Mayer *et al.* 1998; Mayer *et al.* 2001). Interestingly, even though Evolutionary Computation is widely used in optimization problems, including agricultural systems (Mayer *et al.* 1996; Mayer *et al.* 1999; Mayer *et al.* 2005; Meszaros 1999; Kinghorn 1998; Kinghorn 2000), to our knowledge it has not been used for the optimization of cattle nutrition models.

In this chapter method and code was developed to implement the main nutrition aspects of the AFRC (1993) and SCA (1990) cattle models. An evolutionary algorithm is used to search for an optimal set of parameters for multiple objective target outputs. The optimization engine is based on Differential Evolution – DE (Storn & Price 1997) an efficient real-valued evolutionary algorithm that rapidly converges on optimal or near-optimal parameter sets for multidimensional real valued problems with complex solution spaces. The nutrition model and the DE are implemented in the software LaGraZ. The program can be used directly as a modelling tool to calculate the outputs for a given parameter set or, the DE algorithm can be used to search for a parameter set that targets the balance between required and supplied metabolizable energy and/or protein, the balance between fermentable metabolizable energy (FME) and effective rumen degradable protein (ERDP) as well as the target daily weight gain. Optimized parameters consist of an optimized daily ration selected from a mix of user supplied available feeds. The scope of is work is currently limited to optimization of nutrition parameters; an important component for practical applications that was not considered is cost optimization. Nevertheless, the methods described can easily be extended to include costs as an additional objective function.

The remainder of the chapter is structured as follows. Section 3.2 overviews the main aspects of the nutrition model implemented in LaGraZ. Section 3.3 discusses the Differential Evolution algorithm. In section 3.4 the software LaGraZ is introduced. A comparison of predicted daily live weight gains with experimental data from field trials of grazing animals in Uruguay is shown in 3.5. Some conclusions and future work are discussed in 3.6.

3.1.1 Authorship Note

This work was developed in conjunction with Alvaro Simeone from the University of Uruguay. The selection of the nutrition model used in LaGraZ, the validation experiments and the analysis of results were contributed by him. The computational methods, code, Evolutionary Algorithm and the program LaGraZ were developed by the author of this thesis. For this reason the discussion of the nutrition model will be restricted to a brief overview of the main aspects of energy systems with further literature references for the interested reader. The validation experiments will only be indicated. The discussion will be centred on the main topic of this thesis – Evolutionary Computation applied to biological problems.

3.2 Mathematical Model of Cattle Nutrition

The main aspects for a cattle production system, in terms of nutrition, are to determine the nutrient demands of animals and the quantity and types of feeds that must be given to meet those demands. Each nutrient must be supplied so as to meet the demand, ensuring it does not lack, which would result in suboptimal growth, nor is excessive which would result in waste and consequently higher costs. Of the different nutritional requirements, energy is the principal component. The largest nutritional component any typical ration consists of energy sources, thus a formulation that does not meet the energy requirements will have to be severely modified. On the other hand, if for instance a vitamin is lacking, this can be corrected by addition of a small quantity from a concentrate. Further, energy sources directly affect production traits, such as live weight gain in cattle which shows an immediate response to modifications of the available quantities of energy, as well as the demands of other nutrients (McDonald *et al.* 2002:292-294).

Given the importance of energy metabolism, the emphasis of nutrition models resides in relating the intake of energy with the performance of the animals. Essentially a cattle nutrition model consists of a set of equations and coefficients that predict the nutrient requirements and how the feed will be used by the animals. Models usually predict nutritional requirements taking into account the type of animal, environmental factors, management systems, feed composition and biological processes. Several nutrition mathematical models have been developed. In the United States, the *National Research Council* model (NRC 2000) was developed with two levels of solutions, depending on available information. The first level uses tabular feed energy values and the metabolizable protein system. The second level uses the *Cornell Net Carbohydrate and Protein System* – CNCPS (Fox *et al.* 2004) rumen model. The CNCPS is a mechanistic and deterministic mathematical model built upon basic principles of rumen function, microbial growth, digestion and physiology. In Europe there are two main models: the French system from the *Institut National de la Recherche Agronomique* (INRA 1989) and the British model from the *Agricultural and Food Research Council* (AFRC 1993). In Australia the *Commonwealth Scientific and Industrial Research Organization* model (SCA 1990) was developed as an extension of the original AFRC (ARC 1980) model to include the grazing conditions of Australia. The INRA, AFRC and SCA are systems of deterministic empirical equations. A review of these models was presented by Tedeschi *et al.* (2005). The nutrition model used in this work is based on the SCA (1990) model for grazing animals and the AFRC (1993) model for feedlot cattle.

3.3 Differential Evolution

No optimization heuristic is superior to all others for all types of optimization problems (Wolpert & Macready 1996). But, given that an algorithm is capable of finding optimal or near-optimal solutions, it will ideally be simple to implement, fast to converge and will not overwhelm the user with a plethora of initial settings. Differential Evolution – DE (Storn & Price 1997), an evolutionary algorithm for global optimization over continuous spaces, meets these criteria. It can be implemented in approximately 20 lines of code; converges faster than other heuristics (see chapter 7 for a comparison of different methods) and uses a small number of parameter settings. DE has been successfully used in a wide range of optimization problems; frequently outperforming other heuristics (see *www.icsi.berkeley.edu/~storn/code.html* for a bibliography on DE and chapters 6 and 7). For numerical optimization of complex agricultural systems, DE has proven to be a robust optimization algorithm (Kinghorn 1998; Kinghorn 2000; Mayer *et al.* 2005).

Differential Evolution lies on the intersection between real-valued Genetic Algorithms (GAs) and Evolution Strategies (ES), using the conventional population structure of GAs and the self-adapting mutation of ES. In a sense DE can loosely be viewed as a population based Simulated Annealing (SA) with the mutation rate decreasing (analogously to the temperature in SA) as the population converges on a solution.

The principle behind DE is straightforward. An initial population of candidate solutions (*chromosomes*) of user defined size is randomly generated. Each chromosome consists of a numeric vector where each position in the vector corresponds to a numeric parameter to be optimized. The size of the vector is equivalent to the number of parameters. The positions in the vector are referred to as *genes*. On initialization each chromosome is assigned a fitness value according to the objective function. The population evolves by iteratively generating a *challenger* for each chromosome using search operators (see 3.3.1). If the challenger has a higher fitness than the original chromosome, it replaces the latter in the population. If not, the challenger is discarded. Once all chromosomes in the population have been challenged (one *generation*), the process starts again with the new population formed by the surviving chromosomes of the original population and the challengers that had a higher fitness. Differential Evolution uses discrete generations with elitism (the best solutions are always kept in the population). In each generation all chromosomes are challenged and are only replaced if the challenger has a higher fitness than the parent chromosome. Since an elitist approach is adopted, at the end of each generation the average fitness of the population should increase or remain unchanged. The process is repeated until a maximum number of generations are attained, a fitness threshold is reached or the fitness value does not improve over a certain number of generations.

In its basic form, DE uses only four user defined settings (1) number of generations, (2) population size, (3) recombination rate and (4) mutation rate. The number of generations necessary for convergence varies from problem to problem and the user should perform some test runs to get an indication of the evolution of the process. Storn and Price (1997) suggested a population size of 5 to 20 times the number of genes to be optimized. Recent work by Mayer *et al.* (2005) indicates that small populations (1.4 times the number of genes) can be considerably more efficient. The search operators recombination and mutation are discussed in the next section.

3.3.1 Differential Evolution Search Operators

The effectiveness and simplicity of the method resides in the search operators used to construct the challengers. The focus of DE is numerical optimization; as such each gene in the challenger is a numeric value resulting from the interaction of the search operators and the corresponding genes in other solutions. With this in mind, the operators can be divided into three categories (1) parent selection, (2) recombination and (3) mutation.

3.3.1.1 Parent Selection

Consider that *P1* (parent) is the chromosome to be challenged and *C1*, *C2* and *C3* are three different chromosomes randomly selected from the population. For each gene the challenger (*O* - offspring) is constructed such that with a probability equal to the recombination rate

$$O_n = P1_n \tag{3.1}$$

where *n* is the gene (vector position), the value of the challenged gene is simply copied into challenger. If not

$$O_n = C1_n + M*(C2_n - C3_n) \tag{3.2}$$

where $M*(C2_n - C3_n)$ is the mutation value, whose magnitude is partly controlled by the mutation parameter *M*. The new gene value is simply the value of $C1_n$ plus the difference (Hence "Differential Evolution") at the prevailing gene for other two chromosomes multiplied by the mutation parameter/operator *M*. In this manner four parents are always involved in generating the challenger.

3.3.1.2 Recombination

Differential Evolution uses a very simple form of uniform recombination. A user defined rate between 0.1 and 1.0 defines the probability with which a gene value is copied from the challenged chromosome (eq. 3.1) or a new value is generated from the other three parents (eq. 3.2). Low recombination rates are more meticulous in exploring the solution space but are also slower. Higher rates converge faster but there is the risk of getting trapped at a local optimum. An initial recombination rate of 0.5 seems to work well is most situations (Mayer 2005; see also chapters 6 and 7).

3.3.1.3 Mutation

Mutation in DE is a user defined real valued parameter. Differently from canonical binary Genetic Algorithms (Goldberg 1987) in which the mutation rate defines a random uniform probability of flipping a bit at a certain position, in DE the mutation rate is self-adapting. The mutation operator *M* is used as a multiplier of the difference between two randomly selected chromosomes (eq. 3.2) which is then added to a third random chromosome. As the optimization process converges on a solution the population variance decreases and the magnitude of the mutation reduces accordingly. This mimics the self-adapting operators used in Evolution Strategies (Beyer & Schwefel 2002) without the complexity of having to store and calculate variance and covariance information for each gene to tune the operators.

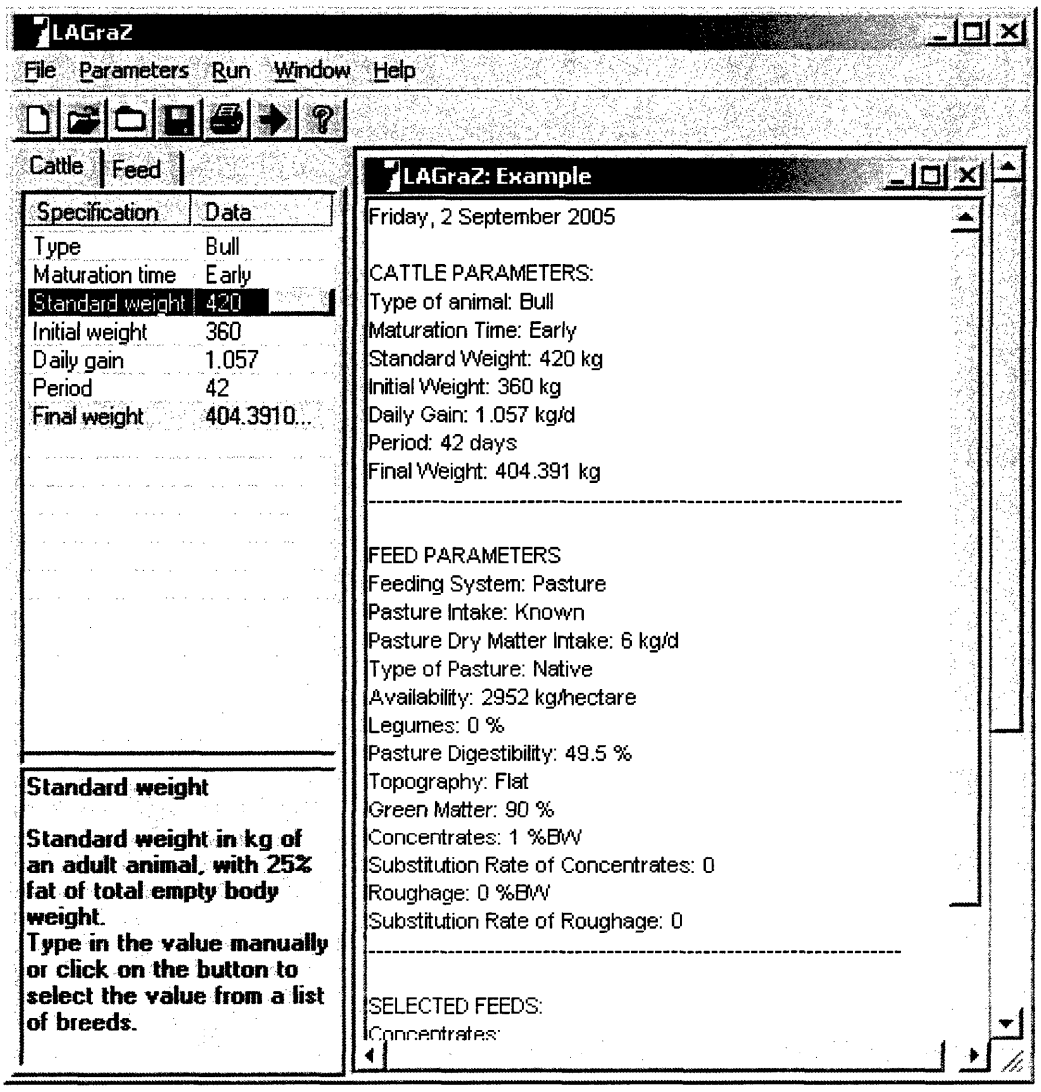
Storn and Price (1997) suggested a mutation operator between 0.4 and 1.0. Lower rates tend to generate intermediates between the parents and higher rates tend to be extrapolative outside the bounds of the parental values (Mayer *et al.* 2005). To avoid entrapment at a local optimum, particularly at the latter stages of the optimization, Mayer *et al.* (2005) suggest changing the mutation rate to a higher level

every few generations to provoke extrapolative mutation. For example every ten generations the rate can be changed to 5.0 and then back again to 0.5.

3.4 LaGraZ

LaGraZ was written using Microsoft's *C#* and runs under the *.Net* platform. The program was designed as a support tool to allow producers to explore how different feed scenarios affect the performance of their animals. An optimization function based on Differential Evolution is included to optimize ration formulations that meet user defined nutritional objectives. LaGraZ is also geared towards the students of Rural Sciences and Agriculture as a simulation tool for classroom applications. A balance was sought between simplicity of use while still providing complete information of the model. To ensure that the program can easily be incorporated into a production system, a graphical user interface based on side panels was developed (figure 3.1). Model parameters are presented to the user in two groups: (1) information about the cattle and (2) information about the feed, which are accessed by clicking on the tabs on the upper side of the panel. The lower box on the panel provides a brief description of the selected parameter. Clicking on a parameter will allow changing the settings directly on the panel or through a dropdown box or a popup menu. Shaded fields cannot be modified and are automatically updated when a parameter is changed. Greyed out fields indicate that the parameters are not applicable for those settings (figure 3.2). The window on the right hand side describes the selected parameters and the results of the simulation. Each time a parameter is modified the results window is automatically updated. LaGraZ uses a multiple document interface which allows simultaneously viewing the results of different simulations.

Figure 3.1 Screenshot of LaGraZ. The panel on the left hand side is used to set the parameters of the model. These are grouped into two classes: *Cattle* and *Feed* parameters, which are accessed by clicking on the tabs on the upper side of the panel. The lower box on the panel provides a brief description of the highlighted parameter. The window on the right hand side describes the selected parameters and the results of the simulation. The arrow on the tool bar runs the optimization routine.



3.4.1 Cattle Parameters

The user is required to provide the *type* of animal, *maturation time*, *standard weight*, *initial weight*, *daily gain* and *period*. The first three parameters are used to select the appropriate coefficients for the model (SCA 1990). The animal type can be *Bull*, *Heifer* or *Steer*. The maturation times are *Early*, *Medium* and *Late*. These two parameters are used to calculate a combined condition score coefficient between 0.7 and 1.3. The *standard* reference *weight* of the animal is the weight at a mature skeletal size with an average condition score. The standard weight can be user defined or selected by breed from a popup menu. Table 3.1 shows the breeds and standard weights used in LaGraZ. The *final weight* is automatically calculated from the *initial weight* added to the *period* (number of days) multiplied by the *daily* weight *gain*. The model assumes linearity, thus it is only applicable within certain limits; conservative constraints as to maximum weight and growth period are enforced in the program.

Table 3.1 Breeds and Standard Reference Weights (SRW) used in LaGraZ.

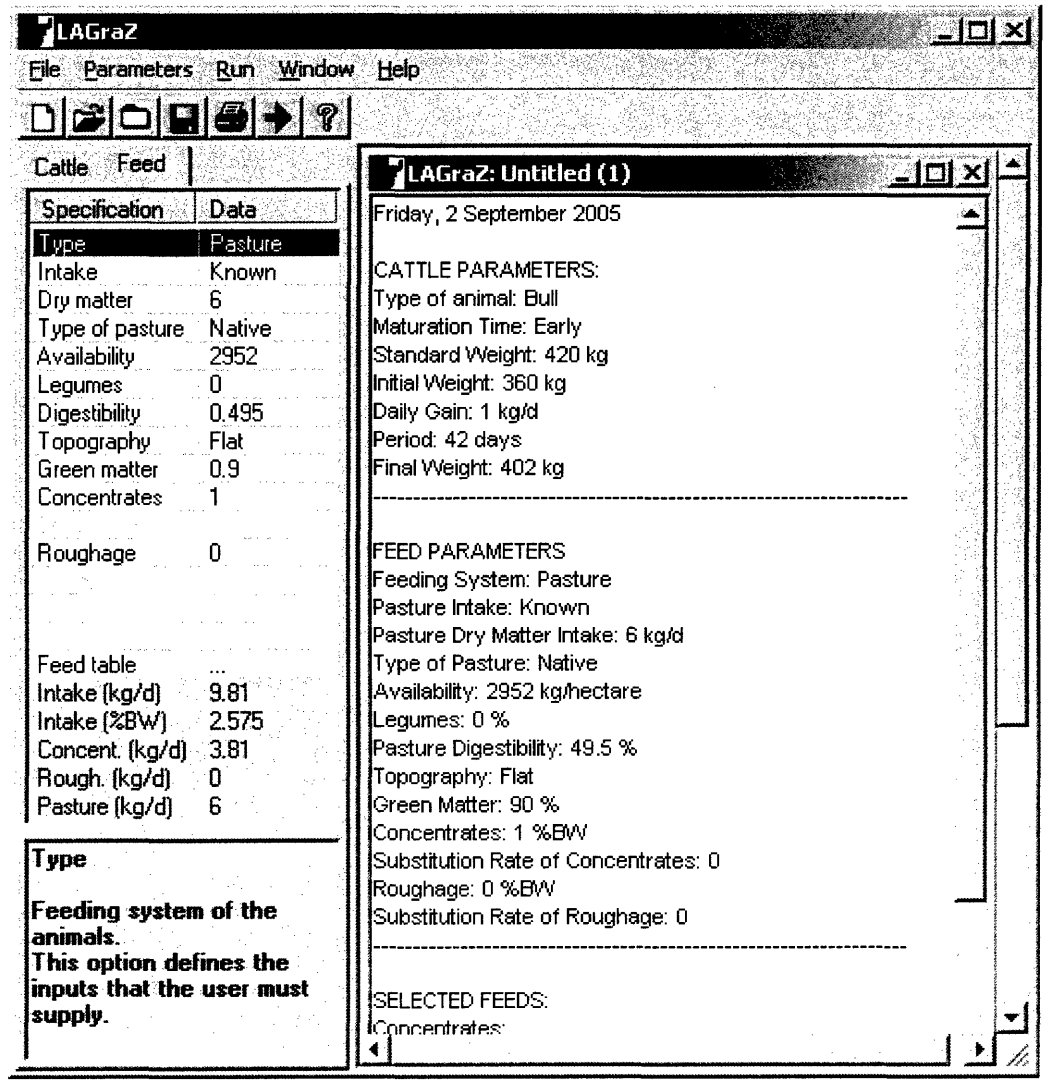
Breed	SRW
Brahman	498.042
Brangus	483.981
Charolais	526.164
Chianina	509.835
Devon	469.012
Hereford/Angus	484.434
Holstein	493.960
Jersey	457.219
Limousin	489.877
Longhorn	435.446
Nellore	496.227
Piedmontese	492.599
Red Poll	464.930
Salers	520.721
Santa Gertrudis	503.031
Shorthorn	524.350
Simmental	520.721

3.4.2 Feed Parameters

Feed parameters are set under the Feed tab (figure 3.2). There are two main classes of feed defined by the *Type* setting: Pasture and Feedlot. Different parameters are required for each (parameters not used are greyed out). Briefly, the parameters for grazing animals are:

- *Intake* – known or unknown. If the daily consumption of pasture is known, the value in kg/day is inserted in the field *Dry Matter*. If unknown is selected, the program will estimate the daily intake from pasture information and the substitution rates – ratio of reduction in pasture consumption in relation to the weight of supplement supplied – for concentrates and roughage.
- *Dry Matter* – daily intake of dry matter from pasture.
- *Type of Pasture* – Native or improved. Is only used if no specific information about the pasture is provided (see *Feed Table*).
- *Availability* – kilograms of dry matter of pasture available per hectare.
- *Legumes* – percentage of legumes available in the pasture. Is only used if no specific information about the pasture is provided (see *Feed Table*).
- *Digestibility* – proportion of dry matter that is digestible. Can be manually defined or automatically calculated from the *Feed Table* (see below) pasture information.
- *Topography* – Flat, medium or steep, affects the maintenance nutrient requirements.
- *Green matter* – proportion of green pasture. Corrects for water content.
- *Concentrates and Roughage* – dry matter quantity of concentrates and roughage offered per day expressed as a percentage of live weight.
- *Concentrate SR and Roughage SR* – Substitution rates for concentrates and roughage. The values can be manually edited or automatically defined from the *Feed Table* information.
- *Feed table* – defines the types and percentages of concentrates, roughage and pasture supplied. The feed table is discussed in details in 3.4.2.1.

Figure 3.2 Feed parameters in LaGraZ. Parameters not relevant for a given scenario are greyed out. Shaded fields are only informative and automatically updated when a parameter is changed.



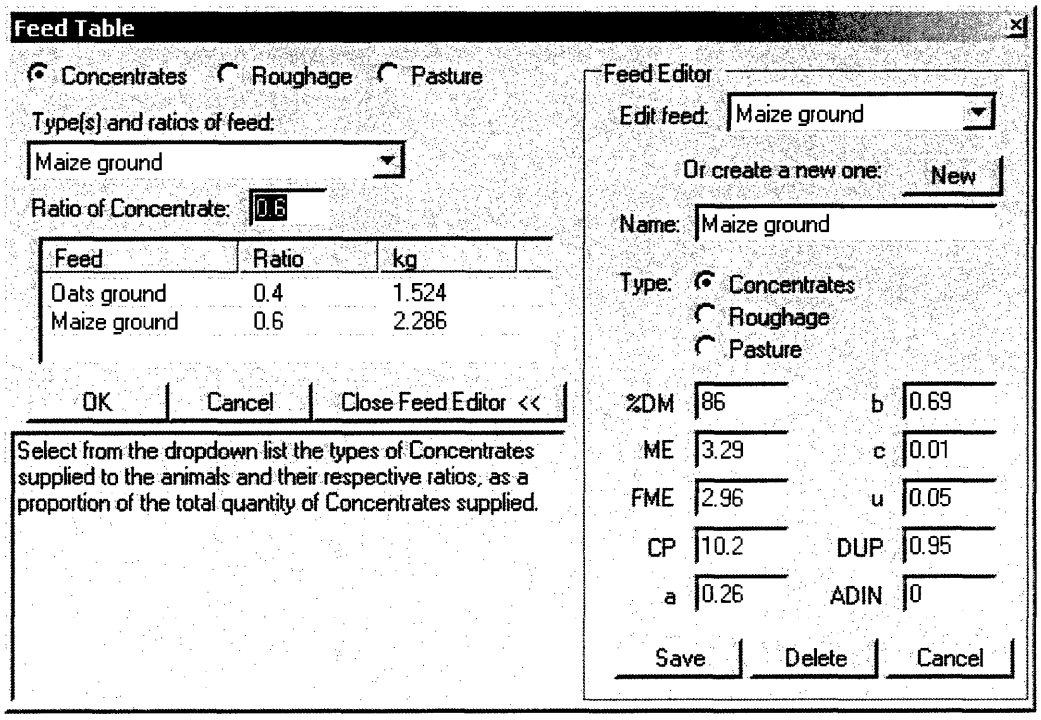
The user defined parameters for feedlot are *Intake* and *Dry Matter* (if intake is known), the proportion of concentrates and roughage in the ration, substitution rates (if intake is unknown) and the feed table (3.4.2.1). Shaded fields (figure 3.2) cannot be modified and are automatically calculated from the parameters. The shaded fields in the feed panel summarize the total daily intake in kilograms and as a percent of body weight, and the daily intake of concentrates, roughage and pasture in kilograms.

3.4.2.1 Feed Table

Even though LaGraZ will provide a rough estimate of energy and protein availability in a generic pasture, the *Feed Table* provides a more efficient method for estimating nutrient availability. Three different classes of feed (concentrates, roughage and pasture) can be selected from a dropdown list (figure 3.3). Each feed chosen as a constituent of the ration is defined as a percentage of its respective class. Actual weights of the feed are automatically calculated from the parameters. There are over forty feeds currently available and these can be modified by the user using the *Feed Editor* (figure 3.3). New feeds can be included via the Feed Editor or directly through a semi colon delimited text file. For each feed the following parameters are specified in the Feed Editor: percent of dry matter (%DM), metabolizable energy (ME), fermentable metabolizable energy (FME), crude protein (CP), degradability

parameters (a – soluble fraction, b – not soluble but fermentable fraction, c – constant rate of degradation of b and u – undegraded Nitrogen fraction), digestible undegraded protein (DUP) and acid detergent insoluble nitrogen (ADIN).

Figure 3.3 Screenshot of *Feed Table*. Feeds from three different classes (concentrates, roughage and pasture) can be selected from the dropdown list on the right hand side. The user defines the percentage of each selected feed for the three classes. The daily allowances in kilograms are automatically calculated from the parameters. On the left hand side is the Feed Editor. Feeds and their parameters can be created, modified and deleted by the user.



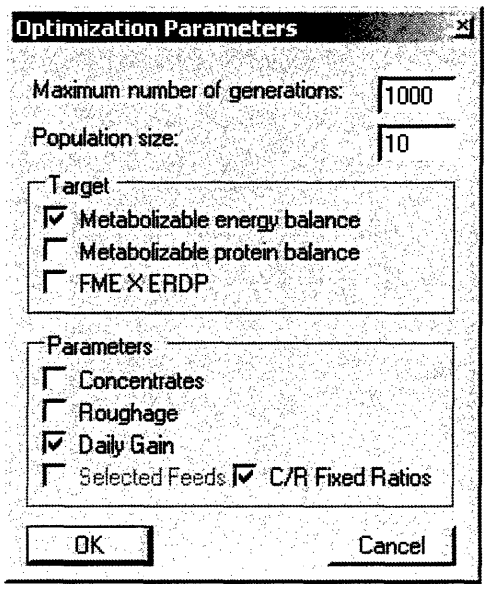
3.4.3 Optimization

The Differential Evolution (DE) algorithm included in LaGraZ allows multi-criteria optimization of model parameters based on user-defined *Targets* and *Parameters* (figure 3.4). The *Targets* are the outputs of the model under a given set of parameters and are used in the objective function. The *Parameters* are the actual model parameters to be optimized.

The goal is to balance the nutritional requirements of the animals to ensure there is no lack or surplus of ration. The objective function is simply construed as the unweighted sum of the absolute arithmetic differences between supplied and required energy needs of the selected targets. Any combination of the three following targets can be selected:

1. Metabolizable energy balance – the difference between the supplied metabolizable energy obtained from the feed information and the required metabolizable energy needed for maintenance and daily weight gain.
2. Metabolizable protein balance – the difference between the supplied metabolizable protein obtained from the feed information and the required metabolizable protein needed for maintenance and daily weight gain.
3. FME X ERDP – the difference between the Fermentable Metabolizable Energy and Effective Rumen Degradable Protein.

Figure 3.4 Screenshot of *Optimization Parameters*. User defined DE parameters are the maximum number of generations and population size (number of chromosomes) in a run. Any combination of *targets* and *parameters* can be selected. *Targets* are used in the fitness evaluation of the chromosomes. *Parameters* are the model parameters that are optimized by the DE.



The DE uses the objective function to assign a fitness value to the chromosomes of the population; optimization consists of minimizing the fitness function (*f*):

$$f = -\sum_{1}^{n}(R_n - S_n)^2 \tag{3.3}$$

where *n* is a target, *R* is the required and *S* is the supplied nutrient. The model must be constrained to ensure that only biologically realistic scenarios are considered. Various constraints are evaluated in the objective function: (1) feed limits – an animal cannot consume unlimited quantities of feed nor excessive quantities of certain types of feed, for instance Urea; (2) mathematical constraints – for example, the substitution rates can cause animals to eat negative quantities of pasture to allow for an increased consumption of supplements; (3) weight gains – daily weight gains are constrained within realistic limits. Model constraints are handled by moving an invalid parameter to its closest valid limit or, if the model still returns an invalid solution, by assigning a very negative fitness value to the chromosome. This ensures that all chromosomes are valid or, alternatively, rapidly removed from the population through selection. This approach is not ideal for optimization algorithms but it is necessary since there are weaknesses in the model that have to be accomodated. The most sensible values for limits would be variable, depending on age, weight and possibly other factors (for example, maximum daily weight gain at birth is much less than at 1 year). This in itself means another growth model is needed to define what the limits should be – or else practitioner experience must be used. The need for any such limits does show a potential weakness in a model for optimisation purposes. This means that it is possible to have a model that robustly predicts animal performance as a function of food eaten as different levels and qualities, but which, when optimised, suggests that best food conversion efficiency is to be had by starving the animal until a millisecond before slaughter, then feeding it a ridiculous amount of feed in the remaining short period.

The parameters that can be optimized are *concentrates*, *roughage*, *daily gain*, *selected feeds* and *concentrates/roughage (C/R) ratio*. Each model parameter selected for optimization is a gene in the DE

chromosome and its length is automatically defined. The parameters interact with one another and different combinations should be used depending on the optimization problem. Briefly, optimization of *concentrates* and *roughage* tries to find the best fit to the selected targets by modifying the supplied daily quantity (in percent of body weight) of concentrates and/or roughage without changing the relative composition of each of these feeds previously defined in the Feed Table (3.4.2.1). The *daily gain* is a useful parameter to estimate the effective daily gain/loss for a given diet. *Selected feeds* is the most useful option for practical applications. This will optimize the relative contents of feeds within a class (concentrates and/or roughage). For instance, given a set of ten possible concentrates (maize, barley, oats, etc) selected from the *Feed Table*, the DE will optimize the relative proportion of these concentrates in the ration to fit the objective function. If *C/R fixed ratio* is used, the solution is constrained by having to maintain the concentrates and/or roughage ratio to body weight constant. This means that, in the above example, the types and proportion of the feeds are optimized but the percent of concentrates to body weight cannot change. If this option is not selected the relative participation of the feeds in the diet and the relation to body weight are simultaneously optimized. A simple optimization example is illustrated in section 3.4.5.

To reduce the apparent complexity of the optimization, the only DE parameter settings exposed to the user are the *maximum number of generations* and the *population size* (figure 3.4). These are the most critical parameters. For the search operators (see 3.3.1), recombination was set at a fixed value of 0.5 and the mutation rate at 0.4. During a run, every 10 generations the mutation rate is changed to 10.0 and then back again. With these parameters the DE converged well in our test cases (data not shown).

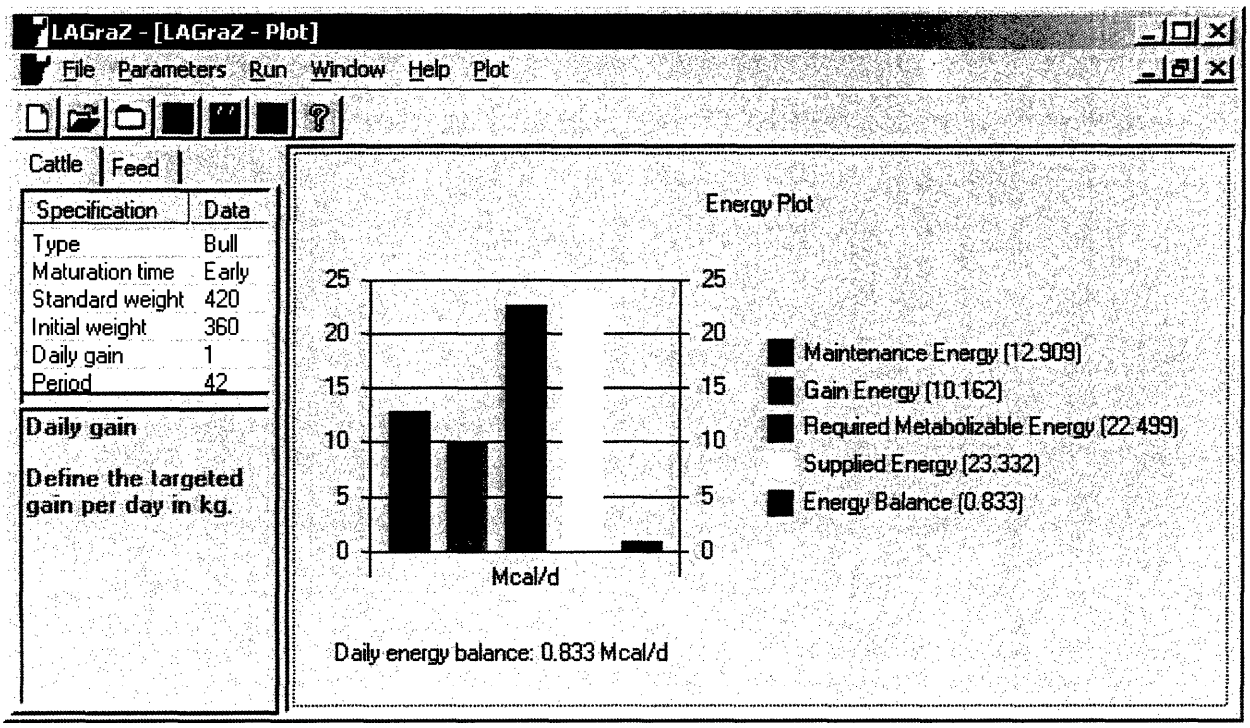
3.4.4 Visualization of Results

Results are presented in the results pane (figures 3.1 and 3.2) as a descriptive summary of the model. This includes the cattle and feed parameters used in the model; a full description of the formulation of the daily ration, either user defined or optimized; summary of the daily requirements for maintenance and growth and metabolic balances of required and supplied energy and protein. Also included are intermediate model parameters of greater interest to students (such as microbial crude protein (MCP), rumen degraded protein (RDP), undegraded dietary protein (UDP) and others).

Visually, different plots can be selected from the *Plot* menu. There are four different plots available in LaGraZ: (1) feed plot – with the distribution of the three classes of feeds (concentrates, roughage and pasture), (2) a feed distribution plot – with a full summary of all feeds in each class, (3) energy plot – summarizes the energy requirements and availability (figure 3.5) and (4) protein plot – the protein requirements and availability.

In the example (not optimized) in figure 3.5, there is a surplus of supplied energy (balance of 0.833 Mcal/d) indicating that further growth can be obtained from this diet. After optimization the balance should be at or close to zero (see example in the next section). Notice that the maintenance and gain energy do not add up to the total required metabolizable energy since a correction for energy retained for the given consumption level is applied.

Figure 3.5 Screenshot of the *Energy Plot*. It describes the total required metabolizable energy, the energy requirements for gain and maintenance, the available energy and the balance between supplied and required energy.



3.4.5 A Simple Optimization Example Using LaGraZ

To illustrate the use of Differential Evolution in LaGraZ a simple optimization problem is presented. A producer wants to estimate the daily gain of his animals over a 40 day period. These animals are early maturation Nelore bulls with an average weight of 400kg. They are grazing animals with an unknown daily dry matter intake of native unimproved pasture supplemented with 6kg/day of ground maize and a substitution rate of 0.5 calculated from the feed coefficients. The pasture digestibility is 49.5% with 90% green matter. The estimated daily dry matter pasture intake is 0.771kg/day. Table 3.2 shows the initial model parameters for cattle and feed. The optimization target is the *metabolizable energy balance* and the parameter optimised by the DE algorithm is *daily gain*. The DE parameters are the default values (maximum generations 1000, population size 10). Notice that it is an optimization problem for the DE; from the user's perspective it is a simple matter of *calculating* the estimated live weight daily gain.

Table 3.2 Cattle and Feed model parameters used in the optimization example.

Model Parameters			
Cattle		**Feed**	
Type	*Bull*	Type	*Pasture*
Maturation Time	*Early*	Intake	*Unknown*
Standard Weight	*496kg (Nelore)*	Type of Pasture	*Native*
Initial Weight	*400kg*	Availability	*2952kg/hectare*
Daily Gain	*0.0kg*	Legumes	*0%*
Period	*40 days*	Digestibility	*49.50%*
		Topography	*Flat*
		Green Matter	*90%*
		Concentrates	*1.5% body weight*
		Substitution Rate	*0.5*
		Feed	*Ground Maize*

These parameters before optimization yield a required metabolizable energy of 11.828 Mcal/day of which 11.828 Mcal/day are required maintenance energy and 0.0 Mcal/day are for gain (daily gain set at 0.0). The supplied metabolizable energy is 19.74 Mcal/day and there is a surplus balance of 7.912 Mcal/day. Running the DE (converges in less than 100 generations and the run is automatically stopped) the balance between supplied and required metabolizable energy is 0.0 Mcal/day (both 20.768 Mcal/day) and the estimated daily weight gain is 1.041kg/day per animal. This is a simple method for estimating the daily weight gain at nutrient balance for a given diet composition.

A more interesting problem is to find a mix of concentrates that will allow a targeted daily gain. The above parameters were modified to a daily gain of 1.0kg per day and the following concentrates were selected from the *Feed Table*: ground maize, ground barley, ground wheat, ground oats and rice bran in equal proportions (20% – 1.26kg/day). The model yields a total (gain plus maintenance) required energy of 22.057 Mcal/day. The supplied energy is 18.005 Mcal/day. For the optimization the same balance between supplied and required metabolizable energy target was used. The parameters to be optimized were the proportion of the available concentrates (*Selected Feeds* box in figure 3.4) with the daily ration of concentrates fixed at 6.3kg/day. After optimization the balance is 0.0Mcal/day (less than 200 generations) with the required and supplied metabolizable energies at 20.457Mcal/day. The new ration formulation is shown in table 3.3.

Table 3.3 Optimized ration formulation for a daily gain of 1kg in Nelore cattle.

Concentrate	Percentage	Weight
Barley ground	14.32%	0.902 kg
Maize ground	75.75%	4.772 kg
Oats ground	3.55%	0.224 kg
Wheat ground	5.92%	0.373 kg
Rice bran	0.46%	0.029 kg
Total	**100.00%**	**6.3 kg**

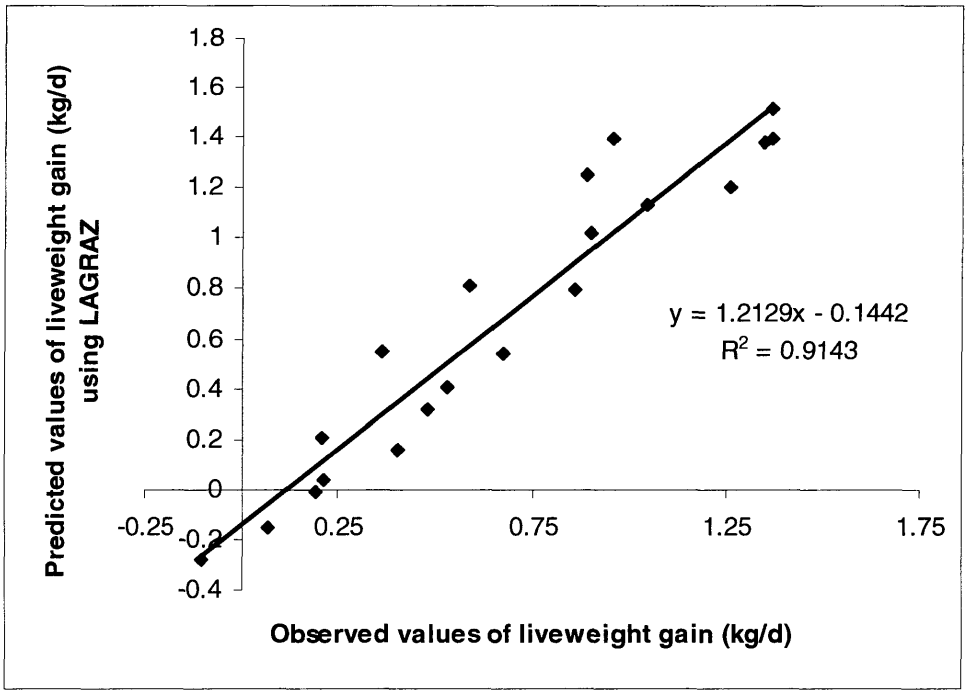
3.5 Experimental Results from LaGraZ

A series of experiments was performed to compare the predictions from LaGraZ with experimental data from twenty trials in Uruguay (Alvaro Simeone pers. comm.). The DE in LaGraZ was used to predict the daily live weight gain in cattle and the results were compared to the measured daily gains observed in the trials. The model parameters (see 3.4.1 and 3.4.2) were adapted from the experimental data to reflect as closely as possible the conditions of the trial (Alvaro Simeone pers. comm.).

For all experiments an optimization run was conducted using the default DE parameters (maximum number of generations 1000, population size 10; see 3.4.3 for details of the DE in LaGraZ). A single run was sufficient since in all cases the DE converged on a perfect fit in less than 500 generations. The target was the balance between required and supplied metabolizable energy; the optimization parameter was the daily weight gain.

Figure 3.6 shows the predicted live weight gain estimated by LaGraZ in comparison to the experimental data from the trials in Uruguay. The calculated R^2 for the 20 data points was 0.9143 which is a good fit to the data, suggesting that the model adequately reflects the nutritional requirements in cattle.

Figure 3.6 Comparison of predicted daily live weight gain with LaGraZ and experimental data from trials in Uruguay.



The DE and the model are tightly interwoven in LaGraZ. This makes it unfeasible to evaluate the DE independently from the model using real data. Runs with simulated data evidenced that the DE rapidly and consistently converged on the global optimum (data not shown), but, more importantly, in the above trial tests the DE converged on a perfect fit to the objective function in all twenty cases in less than 500 generations. This is a good indication that the DE can easily optimize these problems and any poor solutions are essentially due to limitations in the underlying model or over parameterization and not due to the optimization method.

3.6 Conclusion and Future Work

Daily live weight gain prediction in cattle for a given diet and elaboration of efficient rations are of great importance to the cattle industry. Nutritional components of the SCA (1990) and AFRC (1993) cattle models were implemented in a computational tool – LaGraZ – that was developed to assist producers to evaluate the effect of different diets on the nutritional requirements in grazing or feedlot cattle. Differential Evolution (Price & Storn 1997), a robust and simple Evolutionary Algorithm (Mayer *et al.* 2005) is used for multicriteria optimization. In practical terms the algorithm allows the prediction of daily live weight gain for a given diet as well as the optimization of ration formulations and daily allowances to meet specified nutritional targets.

LaGraZ was designed with producers and students in mind. Simplicity of use was an important consideration in this project; particularly in regards to the optimization algorithm. We opted to sacrifice parameter flexibility in the DE by hard-coding parameter settings straight into the algorithm, leaving only the two most sensitive parameters to be defined by the user (maximum number of generations per run and population size). This approach seems to work well since in all test cases the default parameters rapidly converged on a solution without any user input. A further consideration was to simplify visualization of results without compromising information delivery. To this end, results can be viewed

either as a series of simple graphs that summarize the main points or in text format with a full description of the model numerics.

Linear programming has been successfully used in optimization of feed formulations; but evolutionary algorithms can be more generic, this is particularly true for multiple objectives optimization, where Evolutionary Computation has proven more efficient than linear programming (Michalewicz & Fogel 2000) and also easier to implement. The DE used in LaGraZ allows multiple objectives to be selected for optimization: (1) balance between required and supplied metabolizable energy, (2) balance between required and supplied metabolizable protein and (3) balance between fermentable metabolizable energy (FME) and effective rumen degradable protein (ERDP). By balancing supplied and required nutrients, daily weight gains for a given diet can be predicted and ration formulations can be optimized.

Comparisons of predicted live weight gain in LaGraZ with experimental data from animal trials evidenced a good fit, suggesting that the program adequately predicts cattle growth (Alvaro Simeone pers. comm.). It was not possible to evaluate how efficient the model and the DE are at optimizing feed formulations. This would demand specific animal trials feeding optimized rations. Adoption of LaGraZ by producers would allow collection of data to study this aspect of the program. The software is freely available from the author.

In its current version, optimization is focused exclusively on nutritional aspects. The flexible nature of Differential Evolution allows extending the objective function without modifying the algorithm. Future work includes adding economic parameters of feeds (cost of concentrates, roughage and pasture) and cattle (sale price per kilogram) with optimization targeting profit maximization; thus diets will not only be nutritionally efficient but also cost effective. Multiple objectives tend to conflict with one another and a compromise must be sought to balance the non-correlated objectives. The objective function will be modified to allow for user defined weights, where the relative importance of the different criteria can be set. This should allow for greater flexibility of the optimization process.

4 A Simple Genetic Algorithm for Multiple Sequence Alignment

4.1 Introduction

Multiple sequence alignment (MSA) is an important part of molecular sequence analysis being routinely used to identify and measure similarities between samples of DNA, RNA or protein. An alignment is the arrangement of two or more sequences of 'residues' (nucleotides or amino acids) that maximizes the similarities between them. It can be pivotal in the reconstruction of phylogenetic trees or an important tool in the prediction of the function and/or structure of an unknown protein by aligning its sequence with others of known function and/or structure. A third use is the prediction of probes for the same family of sequences in the same or different organisms.

The relationships between sequences are very complex since they have been exposed to evolutionary pressures and mutations over millions of years. Research interests span evolutionary, structural and functional relationships of the sequences. Obviously, the best way to infer these relationships would be from a solid knowledge of the evolutionary history and the structural properties of the sequences. Unfortunately this wealth of information is very seldom available; instead, generic models of protein evolution based on sequence similarity are used (Henikoff & Henikoff 1992) in a scoring system which penalizes substitutions and gap insertions. The highest scoring alignment can be found through a Dynamic Programming (DP) algorithm as the Needleman-Wunsch (Needleman & Wunsch 1970) or Smith-Waterman (Smith & Waterman 1981). But these algorithms are computationally demanding in all but the most trivial problems. To circumvent this limitation, most multiple alignment methods implement approximate heuristic algorithms. Currently the main approach to multiple sequence alignment is the progressive method (Feng & Doolittle 1987) implemented in CLUSTAL W (Thompson *et al.* 1994), MULTAL (Taylor 1987) and T-COFFEE (Notredame *et al.* 2000) for instance. This method is very fast and straightforward but it can easily get caught at local minima. A second method is the exact method, such as MSA (Lipman *et al.* 1989) which tends to give better results than the progressive method but is computationally too intensive for bigger problems; the practical limit is around 10 sequences. The third method is the iteration-based approach. This method uses algorithms that produce an alignment and tries to improve it over successive iterations. This approach includes Hidden Markov Models (HMMs) (Eddy 1998; Karplus & Hu 2001), Simulated Annealing (Kim *et al.* 1994), tabu search (Riaz *et al.* 2004), Genetic Algorithms (Notredame & Higgins 1996; Zhang & Wong 1997; Anbarasu *et al.* 2000; Nguyen *et al.* 2002; Shyu *et al.* 2004) and Evolutionary Programming (Chellapilla & Fogel 1999). A hybrid method using progressive alignment and iteration was suggested by Thomsen *et al.* (2002). No single approach is ideal for all scenarios as evidenced by McClure *et al.* (1994) and Thompson *et al.* (1999a) where the latter evaluated the performance of several alignment programs using the alignments in the BaliBase (Thompson *et al.* 1999b; Bahr *et al.* 2001) benchmark as test cases. Wallace *et al.* (2005) compared iterative alignment methods.

From the above it should be clear that the methods and tools for sequence alignment are numerous and there is as yet no optimal approach to the problem. Progressive alignment methods are fast and deterministic, in the sense that they always provide the same result. These are two important

considerations for routine applications. Their major problem is that errors in the initial alignments are propagated to the other sequences and cannot be corrected. This is not a problem with iterative methods but they tend to be much slower and since most are stochastic, results may vary between runs. Iterative methods are well suited for complex problems where no other alternative is available or where the best possible alignment is important irrespective of computational cost. From these methods, Evolutionary Algorithms have an important advantage over progressive methods in that the alignment component is independent of the scoring function. This means that different objective functions can be tested without modifications to the alignment routine which makes them particularly well suited for testing new scoring functions. Also, by their very nature, EC algorithms are easily parallelizable which meets the current trend of low-cost clusters and multi-core processors. This can shift the time-cost balance since parallelization of DP algorithms is not a trivial task (Ebedes & Datta 2004).

In this chapter a simple genetic algorithm was developed for the multiple sequence alignment of biological sequences. The algorithm is implemented in MSA-GA, a multiple sequence alignment tool with an intuitive graphical user interface, developed within this project. Two methods are available for creating initial candidate solutions: (1) from pairwise alignments, where sequences are aligned in pairs alone, or (2) from a combination of pairwise alignments and a user defined multiple sequence alignment. A set of hand-curated alignments from BaliBase (Thompson *et al.* 1999b; Bahr *et al.* 2001) were used to compare the alignments produced by the two approaches in MSA-GA with the alignments generated by Clustal W.

The remainder of the chapter is organized as follows. Section 4.2 briefly reviews the background information on sequence alignment. In section 4.3 iterative methods based on EC are briefly reviewed, our genetic algorithm and its implementation in the MSA-GA software are discussed. Section 4.4 shows the results of MSA-GA in comparison to the alignments of Clustal W. Conclusions are drawn in section 4.5.

4.2 Sequence Alignment

Evolutionary forces mould living organisms and their biological sequences. Genetic information is inherited from common ancestors through millions of years of evolution (Ridley 2003). Therefore, sequence alignment can be seen as a measurement of evolutionary divergence. The relationships between sequences reflect evolutionary, functional and structural biological connections. From a bioinformatics standpoint sequence alignment is the first step to predict structures, for phylogenetic analysis and to detect critical functional residues (Mount 2004). Moreover, these functions depend heavily on the quality of the alignments. Ideally alignments would derive from the evolutionary history and the structural characteristics of the sequences; but this level of information is seldom available. Instead, ad-hoc models of evolution are used (4.2.4) as part of a scoring scheme and optimized with a heuristic method, usually DP in progressive alignments or a stochastic algorithm (Simulated Annealing, Genetic Algorithms and Evolutionary Programming among others) in iterative alignments.

A pairwise alignment is the arrangement of two sequences that maximizes the similarities between them. The term similarity refers to the number of matches of residues between the sequences; which is distinct from homology which refers to common evolutionary roots. Sequences that align well are considered homologous; with the distinction between homologous and nonhomologous not clearly distinguishable at around 30% identity – the twilight zone (Sander and Schneider 1991). The residues in aligned sequences can be a match, a mismatch or a gap. For homologous sequences a match suggests

an evolutionarily conserved region; a mismatch shared between two sequences can indicate derivation from a common ancestor and a gap is usually explained through insertions or deletions in the sequences. Mismatches and gaps are used to bring as many identical or similar residues into register following a scoring scheme. Alignments can be global or local; the former approach uses the entire sequences to maximize the number of matched residues and the latter approach maximizes the alignment of similar subregions. Multiple sequence alignments are simply an extension of pairwise alignment with three or more sequences.

4.2.1 Dynamic Programming

DP is widely used in optimization problems (Bellman 2003); it is mathematically guaranteed to find the optimal alignment (Shyu *et al.* 2004) and is routinely used for pairwise alignments. For three or more sequences the algorithmic complexity grows significantly (Stoye *et al.* 1997) and since it is a large combinatorial problem (NP-hard) the computational effort becomes prohibitive (Bonizzoni & Della Vedova 2001; Just 2001). In a nutshell, DP is a recursive procedure that splits a problem into a set of interdependent sub-problems in which the next intermediate solution is a function of a prior sub-problem and the next solution depends only on its immediate neighbours. For a pairwise alignment problem DP starts at the end of the sequences and attempts to match all possible pairs of residues according to a scoring scheme for matches, mismatches and gaps (4.2.4) generating a matrix of score values for all possible alignments between the two sequences. The highest score identifies an optimal alignment. The score matrix has dimensions *i, j* – where *i* and *j* are the lengths of the two sequences – and is constructed from top to bottom; to reach a position (*i, j*) in the matrix from a previous move there are three possible paths: a diagonal move with no gap penalty (*i-1, j-1*); a move from position (*i-1, j*) to (*i, j*) and (*i, j-1*) to (*i, j*) with a gap penalty. The matrix is built recursively according to equation 4.1. The actual alignment is obtained from a second matrix, the trace-back matrix – which stores information of the moves through the matrix by backtracking the moves made to obtain the highest score (see Durbin *et al.* 1998 for detailed examples of pairwise alignment using DP).

$$F(i,j) = \max\begin{Bmatrix} F(i-1,j-1) + s(x_i, y_j), \\ F(i-1,j) - d, \\ F(i,j-1) - d \end{Bmatrix} \quad (4.1)$$

In equation 4.1 the function value *F(i, j)* is the highest score for position *i* in sequence *x* and position *j* in sequence *y*; *d* is the cost of the gap penalty and *s* is the score for a match/mismatch between residues x_i and y_j. At position *F(0, 0)* the value is set to zero. More than one path can lead to the same value and a choice has to be made as to which path to follow. Most programs report a single optimal alignment (Mount 2003:81). Another aspect that must be considered is that the choice of scoring scheme and gap penalties influence the alignment produced by the DP algorithm.

4.2.2 Progressive Alignment

Progressive alignment (Feng & Doolittle 1987) is the most widely used heuristic for aligning multiple sequences but it is a greedy algorithm that is not guaranteed to be optimal. DP is used to build the multiple alignment which is constructed by aligning pairs of sequences from the most closely related sequences to the furthest apart with the order of alignments defined by a guide tree of edit distances. To

build the tree the pairwise distances between all sequences are calculated and these are used in a phylogenetic method such as neighbour-joining (Saitou & Nei 1987). The main drawback of progressive alignment methods is that once a sequence has been aligned it is not modified even if it is suboptimal when other sequences are aligned subsequently. This means that information from more distantly related sequences cannot be used to correct initial misalignments. The approach is most efficient when aligning closely related sequences without outliers and without long insertions or deletions in the sequences (Notredame 2002).

4.2.2.1 Clustal W

Clustal W (Thompson *et al.* 1994; Chenna *et al.* 2003) is the most popular multiple sequence alignment program. It uses a global progressive alignment method. The alignment steps are: (1) pairwise alignment of all sequences using dynamic programming or a fast approximate k-tuple method. (2) The scores of the pairwise alignments are used to build a distance matrix of genetic distances. Initially the number of matched positions divided by the total number of residues without gaps is obtained; these scores are divided by 100 and subtracted from 1.0 to get the actual distances. These are used to build the guide tree using the neighbour-joining method (Saitou & Nei 1987). (3) Dynamic programming is used to align the sequences from the most closely related to the least closely related guided by the distances from the tree.

4.2.3 Scoring Systems and Objective Functions

Alongside the local minimum problem due to the greedy nature of the progressive method, Thompson *et al.* (1994) highlighted the importance of the parameter choice in Clustal W. Iterative methods are equally affected by the choice of parameters which if inadequate will yield false global optima. Thus, a critical step in any MSA is the definition of a relevant scoring system.

A scoring system includes scores for matches, mismatches, substitutions, insertions and deletions. In practical terms it can be split into two components: substitution matrices which provide a numerical score for matches and mismatches; and gap penalties which allow numerical quantification of insertions and deletions.

A substitution matrix is a table of numbers of dimension 20x20 (table 4.1) for amino acids and 4x4 (table 4.2) for nucleic acids which represents all possible transitions between the 20 amino acids or the 4 nucleic acids. They provide a measure of the probability of a substitution or conservation occurring. Since the direction of a substitution is unknown the matrices are symmetric. For DNA sequences a simple matrix commonly used assigns a positive value for a match and a negative value for a mismatch (table 4.2). For protein sequences the most common matrices are the Percent of Accepted Mutation (PAM) matrices (Dayhoff 1978) and the Blocks Amino Acid Substitution Matrices – BLOSUM (Henikoff & Henikoff 1992). Values in the matrix are commonly given as the log odds score of the substitution occurring (table 4.1) with substitutions or conservations more frequent that randomly expected assigned positive values and underrepresented substitutions/conservations assigned negative values. Both types of matrices are followed by a number (PAM250, BLOSUM62) which in the PAM matrices refers to the evolutionary distance – a PAM1 matrix is the probability of an amino acid changing into any other to a total probability of 1 roughly representing a period of 50 million years of evolution. Thus greater numbers indicate greater distances. In the BLOSUM matrices the number refers to the minimum percent identity of

the blocks used to construct the matrix – in a BLOSUM62 the sequences had a 62% identity. Thus greater numbers indicate shorter evolutionary distances. BLOSUM is regarded as a preferable matrix choice since each matrix was constructed from actual data while the PAM matrices extrapolate from PAM1 assuming a fixed and independent mutation rate at all sites and over time, while in reality sites vary in their mutability and the rate of mutation over time is also not invariable (George *et al.* 1990). A further criticism of PAM matrices is the small size of the original dataset. A new set of PAM matrices derived from a large dataset was presented by Jones *et al.* (1992). Other substitution matrices have been developed as for example the matrices of Gonnet *et al.* (1992) constructed using data from the entire Swiss protein database, which are currently regarded as being the most accurate.

Table 4.1 BLOSUM62 Substitution Scoring Matrix. The BLOSUM 62 is a 20x20 matrix in which every possible conservation and substitution is assigned a score based on the actual observed frequencies in alignments of proteins with 62% similarity. Identities are assigned the most positive scores. Frequently observed substitutions are assigned positive scores and infrequent substitutions are assigned negative scores (Henikoff & Henikoff 1992).

	C	S	T	P	A	G	N	D	E	Q	H	R	K	M	I	L	V	F	Y	W
C	**9**	-1	-1	-3	0	-3	-3	-3	-4	-3	-3	-3	-3	-1	-1	-1	-1	-2	-2	-2
S	-1	**4**	1	-1	1	0	1	0	0	0	-1	-1	0	-1	-2	-2	-2	-2	-2	-3
T	-1	1	**4**	1	-1	1	0	1	0	0	0	-1	0	-1	-2	-2	-2	-2	-2	-3
P	-3	-1	1	**7**	-1	-2	-1	-1	-1	-1	-2	-2	-1	-2	-3	-3	-2	-4	-3	-4
A	0	1	-1	-1	**4**	0	-1	-2	-1	-1	-2	-1	-1	-1	-1	-1	-2	-2	-2	-3
G	-3	0	1	-2	0	**6**	-2	-1	-2	-2	-2	-2	-2	-3	-4	-4	0	-3	-3	-2
N	-3	1	0	-2	-2	0	**6**	1	0	0	-1	0	0	-2	-3	-3	-3	-3	-2	-4
D	-3	0	1	-1	-2	-1	1	**6**	2	0	-1	-2	-1	-3	-3	-4	-3	-3	-3	-4
E	-4	0	0	-1	-1	-2	0	2	**5**	2	0	0	1	-2	-3	-3	-3	-3	-2	-3
Q	-3	0	0	-1	-1	-2	0	0	2	**5**	0	1	1	0	-3	-2	-2	-3	-1	-2
H	-3	-1	0	-2	-2	-2	1	1	0	0	**8**	0	-1	-2	-3	-3	-2	-1	2	-2
R	-3	-1	-1	-2	-1	-2	0	-2	0	1	0	**5**	2	-1	-3	-2	-3	-3	-2	-3
K	-3	0	0	-1	-1	-2	0	-1	1	1	-1	2	**5**	-1	-3	-2	-3	-3	-2	-3
M	-1	-1	-1	-2	-1	-3	-2	-3	-2	0	-2	-1	-1	**5**	1	2	-2	0	-1	-1
I	-1	-2	-2	-3	-1	-4	-3	-3	-3	-3	-3	-3	-3	1	**4**	2	1	0	-1	-3
L	-1	-2	-2	-3	-1	-4	-3	-4	-3	-2	-3	-2	-2	2	2	**4**	3	0	-1	-2
V	-1	-2	-2	-2	0	-3	-3	-3	-2	-2	-3	-3	-2	1	3	1	**4**	-1	-1	-3
F	-2	-2	-2	-4	-2	-3	-3	-3	-3	-3	-1	-3	-3	0	0	0	-1	**6**	3	1
Y	-2	-2	-2	-3	-2	-3	-2	-3	-2	-1	2	-2	-2	-1	-1	-1	-1	3	**7**	2
W	-2	-3	-3	-4	-3	-2	-4	-4	-3	-2	-2	-3	-3	-1	-3	-2	-3	1	2	**11**

Table 4.2 Example of a simple 4x4 substitution matrix for nucleic acids.

	A	T	C	G
A	**1**	-1	-1	-1
T	-1	**1**	-1	-1
C	-1	-1	**1**	-1
G	-1	-1	-1	**1**

To obtain the best possible alignment, gaps are introduced in the sequence and a scheme for penalizing these gaps must be adopted. The most common approach is the affine gap penalty model where an initial penalty is used to start a gap and a smaller linear penalty is used for extending the gap an extra position (eq. 4.2). The assumption of the model is that a single mutation event of x residues in more likely than x adjacent mutations of a single gap; thus in the alignment there are preferably few longer gaps than a large quantity of short gaps. In equation 4.2 the total gap penalty (d) is the penalty of opening a gap (g) plus the cost of extending the gap (r) times the size of the gap (x).

$$d = g + rx \tag{4.2}$$

The values of gap penalties depend on the choice of matrix and must balance their values (Altschul 1989). A high gap penalty in relation to the values in the matrix will impede the appearance of gaps. On the other extreme too low gap penalties will cause gaps to appear everywhere in the alignment.

Several scoring schemes have been developed. A frequently used scoring scheme is the sum of the score of all pairwise alignments – sum of pairs (SP). But this approach assumes that the sequences are independent from each other and there is an overestimation of the number of substitutions. To account for this effect the SP scores, *s*, over *k* sequences can be weighted by a factor, *w*, that accounts for this effect by reducing the influence of the most closely related sequences (eq. 4.3). Weighted SP is used in Clustal W – the W stands for weighted – and are usually obtained from the distances in the guide tree (Thompson *et al.* 1994). Other weighting methods have been suggested, with a comparison of four of these methods discussed by Vingron and Sibbald (1993).

$$Weighted\ Sum\ of\ Pairs = \sum_{i=2}^{k}\sum_{j=1}^{i-1} w_{ij} s_{ij} \tag{4.3}$$

Other scoring schemes use consistency scores which are a measure of consistency of the MSA with a library of pairwise alignments. This scheme is used in the *Tree-based consistency based objective function for alignment evaluation* – T-COFFEE (Notredame *et al.* 1998; Notredame *et al.* 2000)

In summary, the final MSA is a function of the substitution matrix, gap penalty function, scoring scheme and optimization algorithm. The first three form the objective function which is condensed into a numerical value with the scoring scheme. The objective function is the criterion from which the optimization algorithm must try to find the global maximum. It is important to mention that even if the global maximum is found, the alignment is only optimal in the mathematical sense. The true optimal alignment represents the actual identity by descent of the components under test, and this is generally elusive.

4.3 Multiple Sequence Alignment with Evolutionary Computation

Progressive methods have the problem of propagating initial misalignments into the entire MSA which becomes particularly evident in more distantly related sequences (Thompson *et al.* 1994). Iterative methods try to correct for this problem by iteratively realigning subgroups of the alignments and then aligning them into the entire MSA. With the use of an objective function such as the SP the aim is to improve the MSA alignment score. Iterative methods can be deterministic or stochastic; here the discussion will be limited to a brief overview of previous Evolutionary Computation approaches to MSA, which are stochastic methods. Comprehensive evaluations of iterative algorithms were presented by Hirosawa *et al.* (1995) and Wallace *et al.* (2005).

The seminal work in the field, SAGA (Notredame 1996) is the best known MSA algorithm using Evolutionary Computation. SAGA uses a Genetic Algorithm with 22 different types of complex search operators which are optimized during runtime using dynamic scheduling. The objective function is the weighted sum-of-pairs. Such complexity may be unnecessary, Thomsen and Boomsma (2004) showed that operator scheduling did not improve the quality of alignments in comparison to a uniform selection of operators, they also evidenced that crossover operators contributed little to improve alignments with mutation operators being the main determinant in successful alignments. Zhang and Wong (1997) used a simpler GA but the scoring scheme was based on the number of fully matched columns which limited the algorithm to sequences of high similarity. EC approaches are computationally intensive; to account for

this time demand, Anbarasu et al. (2000) developed a parallelized GA based on the island model. A second parallel GA, limited to multi-processor single machines, was presented by Nguyen *et al.* (2002). Their approach uses a parallelized hybrid GA based on the coarse-grained parallel model. Shyu et al. (2004) used GAs to optimize the guide tree used for progressive alignment methods and also presented a method using a "vertically scalable encoding scheme" for evolving the MSA which demands approximately the same number of iterations to converge irrespective of the number of sequences to be aligned. Both methods were only used and evaluated on simulated DNA sequences limiting their practical application. Chellapilla and Fogel (1999) used an Evolutionary Programming (EP) approach with five different variation operators. Their results indicate that their EP is more efficient than Clustal W for DNA sequences of low similarity. It should be emphasized that these approaches are slow compared to progressive methods. Speed can be improved by parallelization; another approach that not only improves the speed but also the quality of the alignments is to seed the initial population with pre-alignments, as for example, use the results of Clustal W to form the initial population (Thomsen & Boomsma 2004, section 4.4) or a hybrid method using progressive alignment and iteration (Thomsen *et al.* 2002). In the next section our simple genetic algorithm for MSA is discussed followed by an overview of its implementation in the MSA-GA program.

4.3.1 A Simple Genetic Algorithm for Multiple Sequence Alignment

For simplicity the algorithm is illustrated using DNA sequences but it can easily be extended to RNA and protein sequences. MSA-GA allows alignment of any type of sequence; comparisons with other alignment programs (section 4.4) are done using protein sequences which are more challenging due to their structural properties.

4.3.1.1 Matrix Representation

A group of sequences to be aligned consist of *n* sequences of DNA of different lengths. An alignment is represented as a matrix with *n* rows in which each row represents a sequence. Each position in the array is occupied by a symbol from the alphabet {*A,T,C,G,–*}, with the character symbols respectively corresponding to the nucleotides *adenine, thiamine, guanine* and *cytosine*. Gaps are represented by the symbol "–". Evidently, the order of the nucleotides in the sequences has to be preserved and is only interspaced with gaps. The number of columns in the matrix was defined as

$$L = \max(l_1, l_2, l_n) * k + x \tag{4.4}$$

Where the number of columns (*L*) is the size of the longest sequence multiplied by a scaling factor of *k* plus an initial offset *x*. An alignment, after gaps have been inserted, is infrequently more than 20% longer than the longest sequence. An adequate scaling factor is in the range {1.2,1.5}. The offset is the maximum number of gaps inserted at the beginning of a sequence. $k=0.2*l_{max}$ is an adequate offset, but this can be increased is the solution uses most or all of this leading space.

The final alignment is obtained from the matrix by pruning all columns consisting only of the gap alphabet symbol (–) and aligning all l_i symbols across the *n* sequences.

4.3.1.2 Population Initialization

Each organism in the GA consists of a candidate alignment. The organisms of the initial population are generated from pairwise alignments of all the sequences. Initially all global pairwise

alignments between the sequences are computed with Dynamic Programming using the Needleman-Wunsch algorithm (Needleman & Wunsch 1970). For each sequence one of the pairwise alignments corresponding to that sequence is randomly selected to form the organism. At the beginning of the sequence a randomly defined number of gaps is placed (offset). The number of gaps is an integer that varies between zero and the size of the offset. Table 4.3 summarizes the pseudo code algorithm for population initialization.

Table 4.3 Population initialization algorithm.

```
n = 1;
for i = 1 to NumSeq-1
        for j = i+1 to NumSeq
                PairAlign( ) = DP_Align(i, j);
                Alignments(i, n) = PairAlign(1);
                Alignments(j, n) = PairAlign(2);
        n++;
for i = 1 to PopSize
        for j = 1 to NumSeq
                Population(i, j) = InitGaps(Random(MaxOffset))+Alignments(j, Random(NumSeq-1));
        Fitness(i) = Calc_Fitness(i);
```

Once an organism is constructed the objective function is called and an initial fitness value is assigned to the organism. Even accounting for the overhead to calculate the pairwise alignments, an initial population seeded from pairwise alignments is overall faster and greatly improves the scores with reduced convergence times when compared to randomly generated alignments (data not shown). With this approach the initial population starts with a high mean fitness.

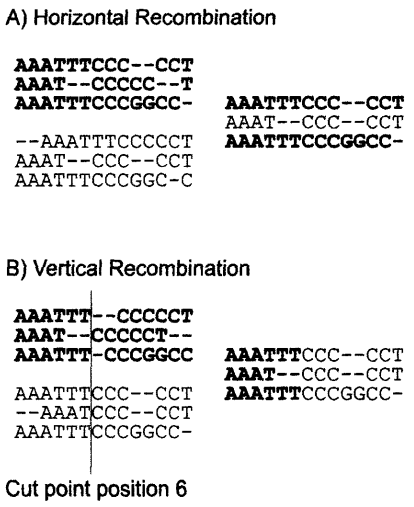
A second method available in MSA-GA is to include a pre-alignment which is inserted into the initial population.

4.3.1.3 Variation Operators

An MSA is defined by the position and size of the gaps in the sequences. From an EC perspective it can be viewed as a "gap-shuffling" operation. Two types of search operators were included in the algorithm: recombination between parents to produce offspring alignments and gap mutations.

Two types of recombination were adopted with independent probabilities of occurring. Offspring are created by one of the recombination types: (1) *horizontal recombination* which builds an offspring by randomly selecting each sequence from one of the parents and (2) *vertical recombination* which randomly defines a cut point in the sequence and the offspring is built by copying the sequence from position 1 up to the cut point from one parent and from the cut point to the end of the sequence from the other parent. The same cut point is used throughout all sequences; initially for each sequence a new cut point was randomly selected but this approach proved to be excessively disruptive to the alignments. With vertical recombination the position of gaps are accounted for to ensure integrity of the structure of the sequences. Figure 4.1 exemplifies the two types of recombination. There are no optimal probability settings for the recombinations; empirical observations suggest that an equal chance of either method being selected with a 30% probability for *horizontal recombination* and 50% for *vertical recombination* are generally appropriate. If no recombination occurs the offspring is copied from one of the randomly selected parents.

Figure 4.1 ***Horizontal*** **(A) and** ***vertical*** **(B) recombination in the MSA Genetic Algorithm. (A) Offspring are generated by randomly selecting entire sequences from either of the parents. (B) A randomly defined cut point splits the sequences of the parents in two; offspring are generated by selecting one substring from each parent.**



Mutation operators can only act on gaps and there are four possible operations: open a new gap, close an existing gap; extend gap size or reduce gap size. Three mutation operators were used to manipulate gaps. To open a new gap a (1) *block mutation* operator is used; a position in a sequence is randomly selected and a block of gaps of variable size is inserted into the sequence. (2) For *gap extension*, a block of gaps is randomly selected and an extra gap position is added. The third mutation operator is (3) *gap reduction*, a block of gaps is randomly selected and a gap position is removed, the probability of a gap position being removed is an inverse function of the size of the gap, meaning that the smaller the number of gap positions the higher the probability that a gap position will be removed. If the selected gap block consists of a single position it will always be removed, and the gap will be closed.

4.3.1.4 Hill Climbing

If over 1000 objective function evaluations the best fitness in the population does not improve, a greedy hill climbing (HC) algorithm (Michalewicz & Fogel 2000) is run to try to further improve the alignments. For the best candidate alignment the HC algorithm goes through each block of gaps in each sequence, adds an extra gap position and evaluates the objective function. If the fitness value improves another gap is added to the block and the fitness is re-evaluated. The process is repeated while the fitness improves. When the new gap results in a worse fitness the gap is removed and the algorithm moves on to the next block. If the best fitness improves, the GA resumes the run; if not the same process is repeated but deleting a gap position instead of adding one. At the end of the process the GA resumes the run.

4.3.1.5 Objective Function

The objective function (OF) is not an integral part of the GA which makes it particularly well suited for testing different scoring schemes. In our GA the fitness value is a direct 1:1 mapping of the objective function, meaning that the scores from the objective function are directly assigned as the fitness of the candidate alignment. The objective function used is the weighted sum-of-pairs (eq. 4.3). In MSA-GA the weights are user-defined and can be derived from the scores of the pairwise alignments used to seed the

initial population with the neighbour-joining method (Saitou & Nei 1987) or some other weighting scheme. Weights are normalized following the method of Thompson *et al.* (1994). Initial gap opening and gap extension penalties are modified following Thompson *et al.* 1994.

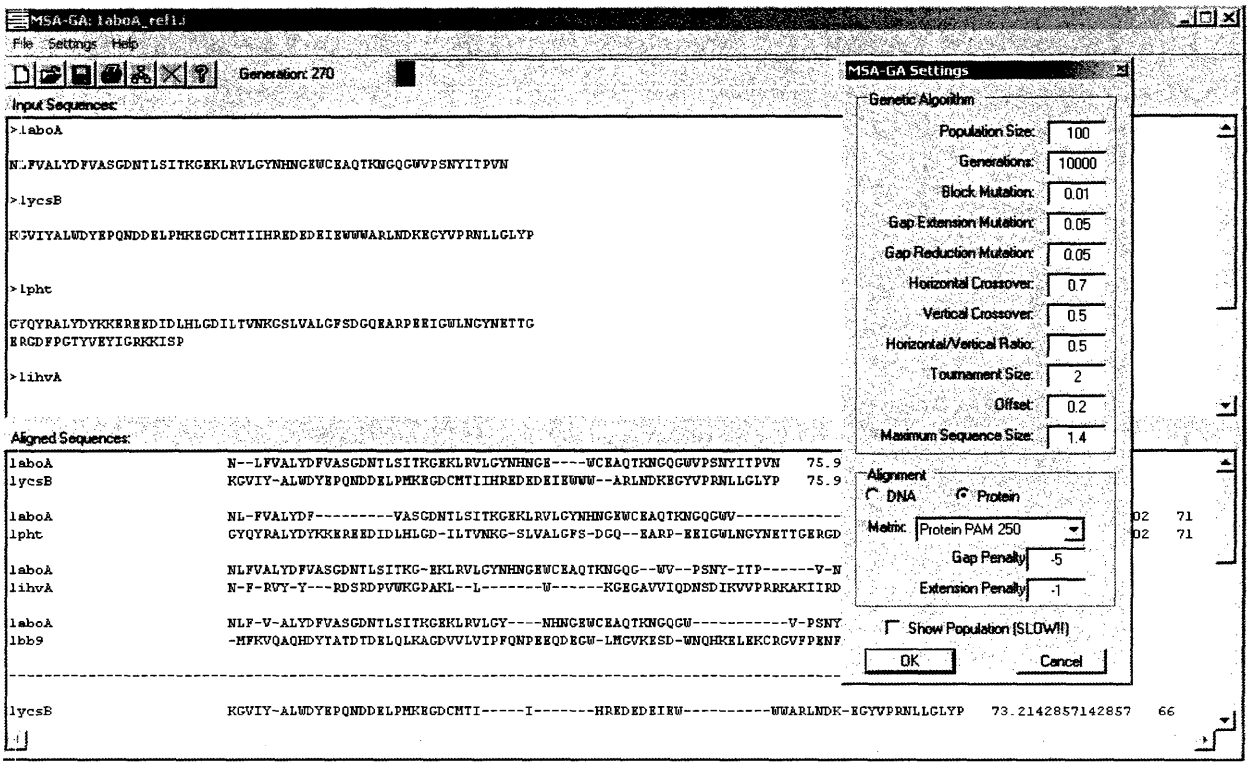
The GA uses steady-state generations and selection is elitist with tournament selection (Bäck *et al.* 2000a). The winner of the tournament remains in the population and the loser(s) are replaced by its (their) offspring. Recombination uses the tournament winner and each of the losers to generate an offspring which will replace the respective loser in the population. If there is no recombination the winner is used as a template for the mutation operators.

4.3.2 MSA-GA

MSA-GA was written in Microsoft C#.Net and implements the simple GA described in the previous section. The main drive in the software design was to ensure a clear separation between the GA and the objective function, for this an object oriented approach was used. The GA communicates with the objective function by sending out a candidate alignment as a character matrix of dimension (n, j), where n is the number of sequences in the alignment and j is the maximum number of columns as per equation 4.4. The objective function returns a numerical score which is the fitness of the candidate alignment. This clear distinction between GA and objective function make it very simple to implement different objective functions.

FASTA format sequences can be pasted directly into the input pane (fig. 4.2) or read from a file (.txt, .india or .fasta extensions). The GA run parameters can be set in the *MSA-GA Settings* window (fig. 4.2). This window also includes settings for the alignment: type of alignment (DNA or protein), RNA is not included since RNA sequences usually are converted into amino acid sequences for alignments; scoring matrix and initial penalty values for opening and extending gaps. Hill Climbing settings can also be changed (*OFF* by default). The current version includes the most commonly used scoring matrices; for DNA the identity and Blast matrices and for proteins the main PAM, BLOSUM and GONNET matrices.

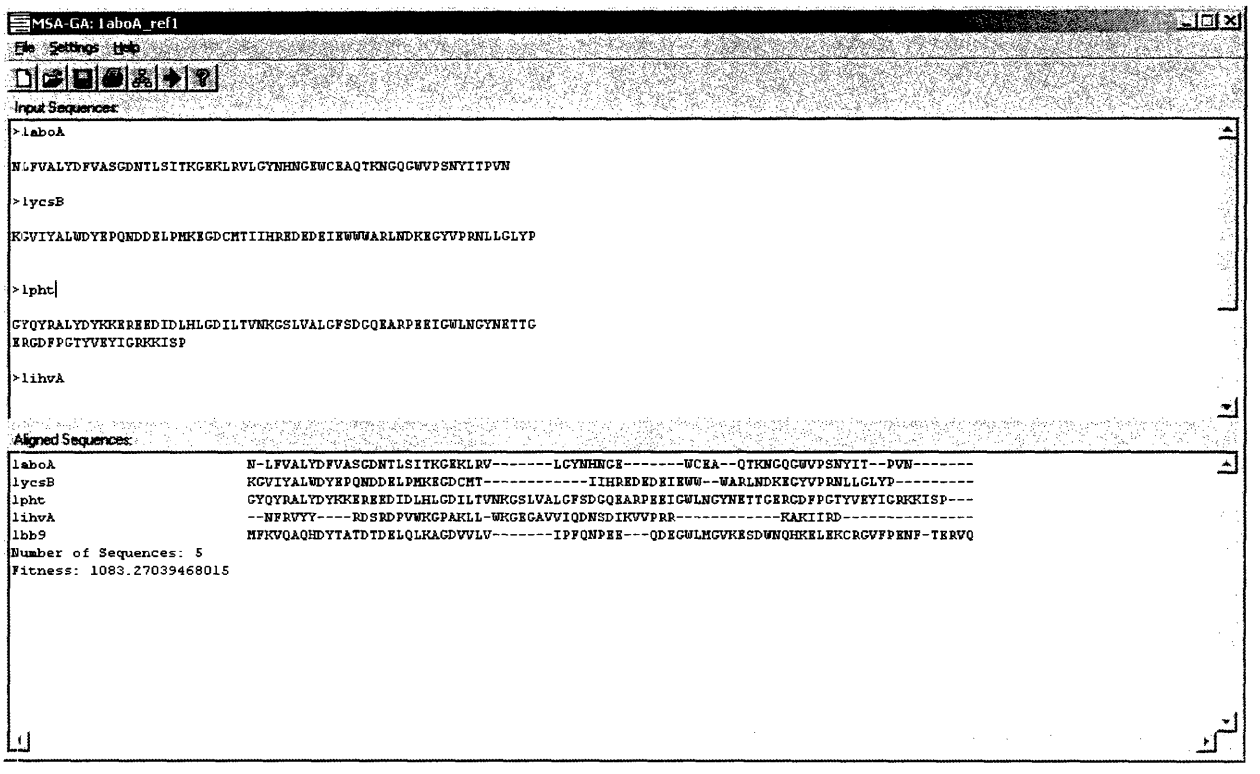
Figure 4.2 Screenshot of MSA-GA. The upper pane shows FASTA format sequences to be aligned. The lower pane displays the pairwise alignments of all sequences. The *MSA-GA Settings* window allows selecting the run parameters of the GA and the alignment settings.



At the beginning of a run MSA-GA will automatically search for weights and prealignment files with the same name as the current file. If a weight file is not available the program will use a default weight of 1.0 for all sequences – unweighted sum-of-pairs. If an alignment file is available MSA-GA will check its integrity and assign the alignment as the first candidate alignment in the population.

On initialization the lower pane displays the pairwise alignments for all sequences (fig. 4.2) with the alignment scores and the size of each alignment. At the end of the run the best alignment is displayed in the lower pane (fig. 4.3). Run results can be saved as text files in FASTA format.

Figure 4.3 Screenshot of MSA-GA showing the best alignment at the end of a run.



4.4 Evaluation of MSA-GA for Multiple Sequence Alignment of Proteins

To evaluate the performance of MSA-GA a set of test cases from the *Benchmark Alignment Database* – BaliBase (Thompson *et al.* 1999b; Bahr *et al.* 2001) were chosen. The original BaliBase (Thompson *et al.* 1999b) consists of a set of 142 reference alignments with over 1000 sequences. Version 2 of BaliBase (Bahr *et al.* 2001) improved some alignments from the original database and extended it to 167 reference alignments and over 2100 sequences including sequences with repeated regions, transmembrane sequences and circular permutations.

BaliBase 2 is divided into eight classes of reference sets (1) equidistant sequences with different levels of conservation, (2) sequences with a highly divergent sequence, (3) groups with less than 25% identity, (4) sequences with N/C-terminal extensions, (5) internal insertions, (6) repeats, (7) circular permutations and (8) transmembrane proteins (Bahr *et al.* 2001). Group 1 is subdivided by sequence sizes. From each group (and subgroups in group 1) sequences were randomly selected for the test cases, in a total of 32 which is a representative sample of the entire database. The sequences in the test cases are shown in table 4.5.

Two sets of five runs for each test case were performed using MSA-GA. The parameter settings of the runs are summarized in table 4.4, Hill Climbing was not used since it is very time demanding and initial tests indicated that less than 1% of the times HC would improve the alignment. For the first set, initial populations were generated using only pairwise alignments. For the second set, pairwise alignments and the alignments from Clustal W were used to seed the initial population. The weights were obtained from the guide trees used in Clustal W.

Table 4.4 GA Parameter settings used in MSA-GA.

Parameter	Value
population size	1000
number of generations	20000
block mutation	0.1
gap extension mutation	0.05
gap reduction mutation	0.05
horizontal recombination	0.8
vertical recombination	0.8
horizontal/vertical ratio	0.5
tournament size	10
offset	0.0
maximum sequence size	1.2

MSA-GA alignments are compared to alignments produced by Clustal W, which is the standard reference alignment program (Chenna *et al.* 2003). To compare the results with Clustal W, the sum-of-pairs score from BaliScore (Thompson *et al.* 1999a) was used. BaliScore compares the aligned sequences from BaliBase with an alignment produced by an MSA program and returns an alignment score between zero (bad alignment) and one (good alignment) which can be used to estimate the quality of an alignment in relation to a BaliBase reference, as well as being directly comparable across the different alignment algorithms. Since MSA-GA outputs results in FASTA format, and BaliScore takes MSF formats as inputs, a small graphical wrapper was written in C# to convert from FASTA to MSF using ReadSeq written by Gilbert (http://iubio.bio.indiana.edu/soft/molbio/readseq/java/), a common converter of sequence formats and then run a modified version of BaliScore (Julie Thompson pers. comm.) with the scores saved as text files. Table 4.5 shows the score of the best run of MSA-GA for the test cases in comparison to the scores from Clustal W. The default parameters of Clustal W were used to generate the alignments and the same settings were used in MSA-GA: initial gap opening penalty of 10.0, gap extension penalty of 0.2 and a positive matrix. Instead of the Gonnet series matrix, only the Gonnet 250 matrix was used. BaliScore returned an error when trying to score the Clustal alignments for reference groups 6 and 7. For this reason the results for these groups were omitted from table 4.5.

Table 4.5 BaliScore score results of MSA-GA, MSA-GA with prealigned sequences and Clustal W for 28 test cases selected from BaliBase 2. In reference 1 the abbreviations are short (*S*), medium (*M*) and long (*L*). References from groups 6 and 7 were not included since BaliScore returned an error when trying to score the Clustal W alignments. The figures in *bold* show a higher score compared to Clustal W.

Ref. 1	MSA-GA	MSA-GA w/prealign	Clustal W
Ref. 1 - S, <25% identity			
1idy	0.427	0.438	0.521
1tvxA	**0.294**	**0.209**	0.06
Ref. 1 - M, <25% identity			
1uky	**0.443**	**0.405**	0.392
kinase	0.295	**0.488**	0.479
Ref. 1 - L, <25% identity			
1ped	0.501	**0.687**	0.592
2myr	0.212	**0.302**	0.296
Ref. 1 - S, 20% - 40% identity			
1ycc	**0.65**	**0.653**	0.643
3cyr	**0.772**	**0.789**	0.767
Ref. 1 - M, 20% - 40% identity			
1ad2	**0.821**	**0.845**	0.773
1ldg	**0.895**	**0.922**	0.88
Ref. 1 - L, 20% - 40% identity			
1fieA	0.843	**0.942**	0.932
1sesA	0.62	0.913	0.913
Ref. 1 - S, >35% identity			
1krn	**0.908**	0.895	0.895
2fxb	0.941	0.985	0.993
Ref. 1 - M, >35% identity			
1amk	**0.965**	**0.959**	0.945
1ar5A	0.812	0.946	0.946
Ref. 1 - L, >35% identity			
1gpb	0.868	**0.948**	0.947
1taq	0.525	0.826	0.826

Ref. 2 - 1 orphan sequence	MSA-GA	MSA-GA w/prealign	Clustal W
2pia	0.761	**0.768**	0.766
1pamA	0.755	**0.758**	0.757

Ref. 3 - Sub-groups of sequences	MSA-GA	MSA-GA w/prealign	Clustal W
kinase	0.58	0.619	0.619
1pamA	0.703	**0.744**	0.743

Ref. 4 - N/C terminal extensions	MSA-GA	MSA-GA w/prealign	Clustal W
1dynA	**0.038**	**0.034**	0
kinase2	**0.71**	**0.635**	0.63

Ref. 5 - Internal insertions	MSA-GA	MSA-GA w/prealign	Clustal W
2cba	0.422	0.621	0.628
S51	0.528	0.73	0.75

Ref. 8 - Transmembrane proteins	MSA-GA	MSA-GA w/prealign	Clustal W
gsh	**0.085**	0.075	0.075
lectin2	**0.158**	0.146	0.146

The average scores for the 28 alignments were 59.04% for MSA-GA, 65.29% for MSA-GA seeded with prealignments and 63.97% for Clustal W. MSA-GA performed better than Clustal W on 12 test cases and worse on 16. The number of test cases for each reference set is insufficient to draw

conclusions as to the suitability of MSA-GA for any particular class of problems; nevertheless, MSA-GA seems to perform better with short or medium length sequences and with sequences of low identity which is in close agreement to the results for the iterative method using Genetic Algorithms implemented in SAGA (Notredame & Higgins 1996; Thompson *et al.* 1999). The orphan sequences in reference 2 seem to bias the alignment, entrapping the GA at a local optimum, which also follow the results from SAGA (Thompson *et al.* 1999). MSA-GA scored higher for the tested references in groups 4 and 8, but only in reference 4 did the alignments improve by more than two percent.

MSA-GA with seeded prealignments improved on the original Clustal alignments in 17 test cases. Of these, six improved the alignment by more than two percent. The average improvement was 1.32% which closely agrees with the average 1.6% improvement obtained with iterative approaches applied to Clustal W alignments (Wallace *et al.* 2004). Even though Notredame and Higgins (1996) observed that seeding could entrap the GA at a local optimum, our results suggest that better solutions can be found if a run is seeded with prealignments. Out of the 28 test cases, in only three the GA did not change the original alignment score. In seven cases the BaliScore value did not change, but in four of these the fitness value in MSA-GA increased. Similar results were obtained by Thomsen and Boomsma (2004) seeding Clustal W alignments into SAGA, with these outperforming randomly initiated alignments. The remaining four alignments yielded worse results than Clustal W even though the fitness values increased, which indicates that the objective function does not adequately map to the 'biological' alignment. An example of this effect was found in *1idy* in reference 1. In an MSA-GA run the SOP score of the Clustal prealignment was 641 and 655 at the end of the run, which yielded a worse BaliScore value (0.521 – 0.438). More interestingly is that the score for the BaliBase alignment itself is 659, clearly evidencing the non linear relation between the fitness score and the 'biological' alignment. Further, the MSA-GA score without a prealignment evolved to a final value of 663 (BaliScore 0.427), beyond the theoretical 'global optimal' value.

4.4.1 Discussion

Genetic Algorithms are computationally expensive, which hinders their mainstream adoption as a tool for multiple sequence alignments. On the other hand, the clear distinction between the scoring scheme and the GA itself make them particularly well suited for testing or implementing different objective functions. The inadequacy of the scoring methods for certain types of alignments is a well known problem and is an active field of research in MSA.

MSA-GA without prealignments yielded better results than Clustal in 43% of the test cases and worse in the remaining, with the alignments on average 4% inferior. These results do not suggest that MSA-GA, even ignoring the time demand, is a superior alternative to Clustal W. Nevertheless, MSA-GA can be used as an additional tool to generate a second alignment from which the researcher can select the best alignment. Alternatively, the alignments produced by Clustal can be used to seed the MSA-GA alignments. For this scenario, the results from the test cases improved the original Clustal alignments in 61% of the times and were worse in only 14% (in 3 alignments 2% or less worse and 8.3% in one alignment – the *1idy* reference, mentioned above, for which the objective function and the 'biological alignment' do not seem to relate well); for the remaining 25% the alignment did not change. The average improvement was 1.32% and the best almost 15% better. Depending on the application of the alignment even small gains can justify the time expense. A further aspect in MSA-GA is that it allows any type of

prealignment and weights to be used. This means that the initial population can be seeded with a prealignment not only from Clustal, but also from other MSA programs or from structural databases.

MSA-GA with prealignments was superior to MSA-GA without prealignments in 71% of the cases. In the eight cases were the latter performed better, MSA-GA with prealignments also improved the original alignment but to a lesser degree. This is possibly due to entrapment at a local optimum as a result of the seeding.

A last consideration is the importance of GA parameter settings. MSA-GA seems to be sensitive to changes in these settings. Changes in population size, mutation and recombination rates and tournament size affect the final alignment and converge times. We performed a series of long runs with *kinase* in reference 1 changing the GA parameters (data not shown). BaliScore results were in the range 0.201 – 0.492. This evidences the stochastic nature of the GA, which can be a matter of concern for the user. These settings are still empirically defined, with no theoretical framework for defining the best settings. SAGA uses a complex operator scheduling to try to optimize the parameters during run time, unfortunately the results of Thomsen and Boomsma (2004) evidenced that there was no significant difference with or without operator scheduling.

4.5 Conclusion and Future Work

A simple Genetic Algorithm for the MSA problem was described and implemented in a novel alignment tool – MSA-GA. References from the BaliBase dataset were used to compare alignments from MSA-GA using pairwise seeding and seeding with alignments from Clustal W.

Our results indicate that the GA seeded with prealigned sequences can improve the alignments derived from Clustal W. MSA-GA without prealignments is less guaranteed to find the best alignment and the stochastic nature of results can render the approach not practical for daily usage. MSA-GA is computationally expensive as, evidently, an iterative approach cannot compete in terms of speed with a progressive method. But, more importantly, the main motivation behind this work was to develop a general framework for testing and implementing different scoring functions. We have shown that a simple GA is capable of optimizing an alignment without the need of excessively complex operators and the object oriented approach used in MSA-GA makes extending and/or replacing objective functions a trivial task.

As noted before, an advantage of an EA approach in which the objective function is disjointed from the optimisation method is that extra components can be easily added to the OF. For example, if relatively large regions (eg. >100bp) have been duplicated, probably as a single event, this should not have a dramatic effect on scores, at least for some applications such as phylogeny. Including this and other such events (eg. reversal) in a strong-arm optimisation method would be highly complex, but easy in an EA approach. Any type of feature or constraint on solutions can be targeted – features that we might not anticipate until appropriate problems prevail. Prior adoption of an EA approach will make such hurdles much easier to overcome.

Future work includes extending the available set of scoring matrices and implementing a function for user-defined matrices. We also want to include different scoring schemes in MSA-GA as the consistency-based objective functions COFFEE (Notredame *et al.* 1998) and T-COFFEE (Notredame *et al.* 2000) and the recent Log Expectation (LE) scoring function (Edgar 2004). Further studies are needed to test the best scope of application for the GA and to evaluate if it is getting entrapped at local optima

with longer sequences or alignments with many sequences as these initial results seem to indicate. This might require the development of new operators in the GA.