

A Framework for modelling stochastic optimisation algorithms with Markov chains

Edward Blair Clark

PhD Thesis

University of York

Department of Electronics

November 2008

Abstract

While various aspects of nature: evolution, clonal selection and annealing, have been the source of inspiration for optimisation algorithms, it is not always clear how and why the algorithms work well on some problems and poorly on other problems. In this thesis we consider properties of exact models of optimisation algorithms and relate these properties to specific operators. To facilitate this lower level of understanding of how optimisation algorithms work, we perform a case study of modular modelling on an example of an Artificial Immune System (AIS) algorithm, the B-Cell Algorithm (BCA). From a case study of modular modelling of the BCA, we derive a framework for modelling stochastic optimisation algorithms with Markov chains. Based on a Markov chain model of the BCA we obtain a proof of convergence of the algorithm, bounds for the rate of convergence of the algorithm and a brief numerical analysis of the Markov chain model of the algorithm. The framework demonstrates that optimisation algorithms can conceptually be split into two parts: search operators and state operators. Search operators are represented by a "sample matrix". State operators are represented by a "possible transit matrix". These matrices can be combined by two equations to form the transition matrix of a Markov chain model of the algorithm. Equations (5.11) and (5.12) are the key to the framework, they allow the construction of the transition matrix from the sample matrix and possible transit matrix. We apply the framework to create a Markov chain model of a Hill Climber, re-write an existing model of Simulated Annealing in terms of the framework and produce a novel Markov chain model of the $(1+\lambda)$ Evolutionary Strategy with fitness proportional mutation. These models, along with the model of the BCA, demonstrate that the framework is not restricted to a particular field of optimisation.

Contents

A Framework for modelling stochastic optimisation algorithms with Markov chains	1
Abstract	2
Contents	3
Algorithm list	7
Figure list	8
Table List.....	10
Acknowledgments.....	11
Author's Declaration	11
Chapter 1: Introduction.....	12
1.1 Maximising, Minimising and Optimising	14
1.1.1 Optima	15
1.2 The nature of space in computers.....	16
1.3 All representations map to binary strings	21
1.4 Black box optimisation	23
1.5 Operators.....	25
1.5.1 Search operators	25
1.5.2 State operators	26
1.6 Convergence	27
1.7 Publications	29
1.8 Order of material to come	30
Chapter 2 Background.....	33
2.1 Markov chains.....	33
2.1.1 The Markov property	34
2.1.2 Matrix properties	34
2.1.3 Optimisation algorithms and the Markov property	35
2.1.4 Tractability of numerical Markov chain calculations	37
2.2 Simulated Annealing.....	37
2.2.1 Simulated annealing Markov chain models	39
2.3 Evolutionary Algorithms	41
2.3.1 Genetic Algorithms.....	41

2.3.2 Evolutionary Strategies	42
2.3.3 Theory of Evolutionary Algorithms	42
2.3.4 Schema Theorem.....	43
2.3.5 Markov chain models of Evolutionary Algorithms.....	44
2.4 Artificial Immune Systems.....	50
2.4.1 Markov chain models of clonal selection algorithms	51
2.5 Discussion of the literature	53
2.5.1 Defining the state of an algorithm.....	54
2.5.2 Motivation for a new approach to Markov chain modelling	57
2.6 Summary	58
Chapter 3 The B Cell Algorithm	61
3.1 The B Cell Algorithm	62
3.1.1 The BCA from a modelling perspective.....	65
3.2 Mutation masks.....	66
3.3 BCA with clonal pool of 1, population of 1 and no elitism	67
3.3.1 The Contiguous Region Hypermutation Operator (CRHO)	68
3.3.2 Counting states in the Contiguous Region Hypermutation Operator (CRHO).....	70
3.3.3 Contiguous Region Hypermutation Operator for a bit string of length one	71
3.3.4 Contiguous Region Hypermutation Operator for a bit string of length two.....	72
3.3.5 From mutation masks to state tables	77
3.3.6 Choosing to use the Markov chain representation	79
3.4 The BCA with population of 1 and a clonal pool of 1	81
3.4.1 The Sample Matrix.....	83
3.4.2 The Possible Transit Matrix.....	85
3.4.3 Constructing the Transition Matrix	86
3.4.4 Worked Example	86
3.5 Proof of convergence	89
3.5.1 General proof of convergence.....	96
3.6 Demonstration of numerical application of the model	97
3.7 BCA with clonal pool of C, population of 1 and elitism	100
3.7.1 How cloning affects the state space of the BCA.....	102

3.7.2 Exploring a modification of the sample matrix	102
3.7.3 General construction of a clonal matrix.....	107
3.8 Summary	109
Chapter 4: Bounding the rate of convergence and approximate calculations	111
4.1 Calculation methods	115
4.2 The Lower Bound	117
4.3 The Upper Bound	119
4.4 Comparison with current bounding results	122
4.5 Chapter discussion	126
4.5.1 The Isomorphism between bounding and plotting	126
4.5.2 On the usefulness of bounding results in stochastic search.....	128
4.6 Summary	129
Chapter 5: Modelling Framework	131
5.1 Method of construction.....	132
5.2 Defining the state of an algorithm	133
5.2.1 Algorithms with populations and algorithms with population based operators	134
5.2.2 Memory Sets and elitist operators.....	136
5.3 The Generalised Sample Matrix	138
5.3.1 Special case – random search	138
5.3.2 Special case – composite operators.....	140
5.3.3 Special case – forbidding null mutations	141
5.3.4 Special case - The clonal matrix	142
5.4 General Possible Transit Matrix	143
5.4.1 Indiscriminate search	143
5.4.2 Elitism.....	144
5.4.3 Simulated Annealing	144
5.5 General Transition Matrix	146
5.6 Dynamic algorithms and landscapes	147
5.7 A practical guide to modelling with the framework.....	150
5.7.1 Definition of a state of the algorithm	151
5.7.2 Constructing the sample matrix	152
5.7.3 Constructing the possible transit matrix.....	153

5.7.4 Constructing the transition matrix.....	154
5.8 Summary	154
Chapter 6: Applying the framework	157
6.1 Algorithm model – hill climber.....	158
6.1.1 Determine the minimal definition of a state of the algorithm: ..	159
6.1.2 The Hill Climber Possible Transit matrix	160
6.1.3 The Hill Climber Sample matrix	160
6.1.4 Hill climber example.....	161
6.1.5 The hill climber transition matrix	162
6.2 Algorithm model – Simulated annealing	163
6.2.1 Determine the minimal definition of a state of the algorithm ...	164
6.2.2 The Simulated Annealing possible transit matrix	164
6.2.3 The Simulated Annealing Sample matrix	165
6.2.4 Simulated Annealing example	165
6.2.5 The simulated annealing transition matrix.....	166
6.3 Algorithm model – $(1+\lambda)$ Evolutionary Strategy	167
6.3.1 Definition of state of the algorithm.....	169
6.3.2 Possible transit matrix.....	169
6.3.3 Sample matrix.....	170
6.3.4 $(1+\lambda)$ Evolutionary Strategy example.....	171
6.3.5 $(1+\lambda)$ Evolutionary Strategy transition matrix	174
6.4 Summary	175
Chapter 7: Discussion and Conclusion	177
7.1 The framework	177
7.2 Markov chain models as algorithm specifications	178
7.3 General observations about bio-inspired algorithms.....	180
7.4 Convergence proofs and theoretical analysis.....	181
7.4.1 Proof of convergence.....	181
7.4.2 Bounds on the rate of convergence	183
7.4.3 Numerical analysis	184
7.5 Thesis conclusions	186
7.6 Future work.....	187
References.....	189

Algorithm list

Algorithm 1: The B-Cell Algorithm adapted from (Kelsey and Timmis, 2003)	62
Algorithm 2: The B-Cell algorithm with a clonal pool of 1, population of 1 and no elitism	67
Algorithm 3: The B-Cell Algorithm with a population of 1, a clonal pool of 1 and elitism.....	82
Algorithm 4: The B-Cell Algorithm with a clonal pool of C, population of 1 and elitism.....	101
Algorithm 5: Hill Climber	159
Algorithm 6: Simulated Annealing.....	163
Algorithm 7: $(1+\lambda)$ Evolutionary Strategy.....	168

Figure list

Figure 1. Mapping a continuous function onto a discrete search space grid. The grey dots show the search space grid..... 19

Figure 2. Chapter dependency diagram 32

Figure 3. The Contiguous Region Hypermutation Operator. Figure adapted from (Clark et al., 2005) 65

Figure 4. The probability distribution of the Contiguous Region Hypermutation Operator: String Length $l=8$, Probability of mutation $r=0.5$. The theoretically determined probability distribution is shown in A. The probability distribution in B was determined experimentally by averaging one million results from the implemented CRHO. Figure adapted from (Clark et al., 2005). 76

Figure 5. Examination of the effect of mutation rate, r , on the B cell algorithm convergence rates for a one-dimensional quadratic function. String length $l=8$. Figure taken from (Clark et al., 2005)..... 99

Figure 6. Bounding the rate of convergence. The curved black line is the true rate of convergence on the optima, while the dashed line is a line that bounds it. Whether the dashed line is an upper or lower bound is subjective, depending on the direction you project onto the black line. Projection 1: for a given value of probability of being in the optima δ , the dashed line will be an upper bound on the number of iterations required $t_1(\delta)$. Projection 2: for a given number of iterations t , the dashed line will be a lower bound of the probability of being in the optima $\delta_1(t)$ 112

Figure 7: The transition matrix for an elitist algorithm optimising a needle in a haystack function. The column of the optimum contains all the values f_T for $T > 0$ exactly once. $f_{T=0}$ is not explicitly represented in the matrix as it occurs where the diagonal intersects the column of the optima where there is a 1, as all transitions from an optima are forbidden. All other diagonal elements take the complementary value $1-f_T$ for a given value of T for the same reason. It should be noted that while the positioning of the column is

placed arbitrarily to reflect the unknown location of the optima, all transition matrices that represent needle in a haystack functions can be permuted to be lower triangular. 114

Table List

Table 1. Mutation masks from state 00 to all states.	77
Table 2. Probabilities of mutation masks occurring from state 00 to all states.	77
Table 3. Mutation masks from state 01 to all states.	78
Table 4. Probabilities of mutation masks occurring from 01 to all states. ..	78
Table 5. Mutation masks from state all states to all states.	78
Table 6. Probabilities of mutation masks occurring from all states to all states.	79
Table 7. Generic probabilities of sampling all states from a starting state of 00, for a clonal pool of arbitrary size.	105
Table 8. Example algebraic probabilities of sampling all states from a starting state of 00, assuming a clonal pool size of two	106
Table 9. Generic probabilities of sampling all states from a starting from all states, for a clonal pool of arbitrary size	106
Table 10. Example algebraic probabilities of sampling any states from a any starting state, assuming a clonal pool size of two	107
Table 11. Minimal state of the algorithm guide	151

Acknowledgments

I would like to thank my supervisor, Professor Jon Timmis. I would also like to thank Dr Andy Hone, who was originally my co-supervisor when I started my PhD at the University of Kent. Once I had transferred to York he was no longer able to be my supervisor in an official capacity, nevertheless he continued to watch over the progression of this thesis and provide guidance and support all the way to the end.

Author's Declaration

I hereby declare that the work contained in this thesis is my own and has not previously been published with the following exceptions:

The content of sections 3.4, 3.5 and 3.6 and their subsections (where applicable) were published in 2005 in a conference paper (Clark et al., 2005). Parts of section 3.5 also appear in a journal paper of which I am not an author (Stepney et al., 2005). This publication precedes (Clark et al., 2005); however, the section of (Stepney et al., 2005) that contains material the same (or similar) to the material in (Clark et al., 2005) and section 3.5 of this thesis is referenced as a footnote "Edward Clark (2004) private communication". The material published in (Clark et al., 2005) is revisited in (Timmis et al., 2008c).

Chapter 1: Introduction

Like other bio-inspired paradigms, such as Genetic Algorithms (GA) (Holland, 1975), Simulated Annealing (SA) (Kirkpatrick et al., 1983) and Evolutionary Algorithms (EA) (Rechenberg, 1971), the Artificial Immune System (AIS) paradigm (Timmis and Andrews, 2007) has given rise to a variety of function optimisation algorithms. With a legacy of benchmark functions from the older bio-inspired optimisation algorithms, empirical studies show AIS algorithms are comparable to existing optimisation algorithms, but did not show them to have any clear advantage over any of the other paradigms (Timmis et al., 2008b). The field of AIS has matured to the point where the AIS community is seeking stronger theoretical understanding the optimisation algorithms produced by the AIS paradigm (Timmis et al., 2008c, Cutello et al., 2007). It is this push for stronger theoretical understanding by the AIS community that provided the motivation for the theoretical modelling and analysis undertaken in the case study presented in chapter 3 of this thesis.

We hypothesise that it is possible to construct modular models of stochastic optimisation algorithms and shall derive a modular modelling framework based on a case study. We shall demonstrate that the models produced can be used to further understanding of optimisation algorithms through theoretical and numerical investigations.

This thesis proposes a framework for modelling of stochastic optimisation algorithms with Markov chains, which we derive from a case study of an AIS optimisation algorithm. An exact mathematical model of the B-Cell Algorithm (BCA) (Kelsey, 2004, Kelsey and Timmis, 2003, Kelsey et al., 2003) is presented in chapter 3. The tools developed by modelling the BCA are used to construct a general framework for modelling of stochastic optimisation algorithms with Markov chains which is presented in chapter 5.

The Markov chain models of optimisation algorithms published prior to this work are of non-elitist algorithms (Nix and Vose, 1992, Häggström, 2002, Villalobos-Arias et al., 2004, Laarhoven and Aarts, 1987) and have therefore taken the form of irreducible Markov chains (Vose, 1995). The work of Villalobos-Arias et al (Villalobos-Arias et al., 2004) on the MISA algorithm (Coello and Cortes, 2002) present an explicit transition matrix for an irreducible Markov chain model of a non-elitist version of the MISA algorithm and also a proof of convergence for an elitist version of the algorithm, but do not explicitly provide a transition matrix for the reducible Markov chain, which corresponds to the elitist algorithm. The work presented here provides the first complete model of an elitist algorithm, giving the transition matrix explicitly and shows that model takes the form a reducible Markov chain; result has been published (Clark et al., 2005). Construction of the explicit transition matrix for the Markov chain model of BCA required the development of a novel system for modelling optimisation algorithms.

In order to develop a novel modelling system, every branch of assumptions must be reconsidered at its most fundamental level. Many assumptions will of course be the same as in existing models, as they are required by the structure of what is being modelled; however there are some choices to be made. The algorithm to be modelled provides the arbitration of these choices.

Some aspects of the model will be determined by the specifics of the algorithm. However, there are also some considerations about computational optimisation in general that need to be examined. These preliminary, but fundamental aspects of modelling computational optimisation algorithms are explored in this chapter.

1.1 Maximising, Minimising and Optimising

There are two important areas that require attention in order to have a full understanding of computational optimisation of mathematical functions:

- the finite, discrete representation of the function
- the behaviour of the algorithm

This thesis shall focus on algorithm behaviour. We shall illuminate the problems of representation in this chapter in order to fully contextualise the thesis, but no attempt to tackle the problems associated with the choice of representation will be made.

Instead, having distinguished between the two parts of the problem, we shall demonstrate that they are separable and extricate the problem of the algorithm behaviour and focus on that aspect of optimisation. In order to demonstrate the two halves of the problem are separable we shall have to consider computational optimisation from its most fundamental level.

Let us define some objective function f , where

$$f : S \rightarrow \mathbb{R} \tag{1.1}$$

for some set of values S , and let us restrict the objective functions we consider to continuous mathematical functions by defining $S = \mathbb{R}^d$.

A value x^* is defined to be a maximum of some function, if

$$f(x^*) \geq f(x) \tag{1.2}$$

for all $x \in S$.

A value x^* is defined to be a minimum of f , if

$$f(x^*) \leq f(x) \tag{1.3}$$

for all $x \in S$.

Let us define optimisation in terms of affinity. Affinity comes from the Latin *affins*, meaning connected with, or similar to. Affinity is related to the difference between the maximum or minimum value of a function $f(x^*)$ and the current value $f(x')$. The affinity is maximised when the difference between $f(x^*)$ and $f(x')$ is minimised; when $\|f(x^*) - f(x')\| = 0$. Hence, optimisation attempts to maximise the affinity of the function; to maximise the similarity between the candidate solution $f(x')$ and the optima $f(x^*)$ be it a maximum or a minimum.

1.1.1 Optima

Grammatically, optimum (number of optima = 1) is specific to the singular and optima (number of optima > 1) is specific to the plural. In order to make general statements, it is sensible to discuss a general number of optima, where the number of optima ≥ 1 . This point may seem trivial, but it is not. The grammatical constraint imposes an unnecessary and unwanted assumption on any statements made.

Theory developed for an arbitrary number of optima can be collapsed to the case where there is a single optimum. Theory developed under the assumption that functions have a unique optimum will not necessarily generalise. In order to make statements about an unknown function with an unknown number of optima, we require a word that means one or more optima, in order to maintain the required generality. Optima shall be used in all cases where it is not specifically known that there is a single optimum, but the use of the word optima should not restrict statements to functions which have more than one optimum. Consequently we define

Optima: A point in the search space with the highest affinity.

Note this does not exclude the possibility that other points in the search space with equally high affinity exist, nor does it strictly state that other points with equally high affinity exist.

Under the definition of optima given in this thesis, all functions must contain one or more points that qualify as optima, even if the function is a constant, which is a case that is excluded in some definitions, as in Rudolph (Rudolph, 1994a).

1.2 The nature of space in computers

In order to solve a theoretical computational optimisation problem, a well posed optimisation problem is needed. We quote Rudolph (Rudolph, 1994a) for a concise example of how this is achieved "... static optimisation problems of the type $\max\{f(b) | b \in B^l\}$ assuming that $0 < f(b) < \infty$ for all $b \in B^l = \{0,1\}^l$ " and similar definitions are made in (Nix and Vose, 1992, Clark et al., 2005, Villalobos-Arias et al., 2004, Häggström, 2002). None of the authors make a discussion about the choice of representation used and how that can affect the optimisation problem. If we look at common benchmark functions such as the twelve benchmark functions defined in (Salomon, 1996), all twelve functions could be considered to be of the type $f : \mathbb{R}^l \rightarrow \mathbb{R}$. Each problem is further specified by pre-defined limits; the limits imply differing levels of scaling, some being coarser search spaces than others. The importance of the choice of search space grid should not be discarded lightly. This section shall demonstrate how the number of optima, location of the optima and values of the function at the optima of the discretised function may differ from the optima of the continuous function. Moreover, how two different representations applied to the same function may produce different optimisation problems; demonstrating that function alone does not define the optimisation problem, but that the search space grid used is of equal importance.

We must first examine the nature of the substrate on which the computational optimisation algorithm is based. Let us assume that the computational device that will be used as a platform for the optimisation problem is the common digital computer (as opposed to an analogue computer). This assumption leads directly to the knowledge that we are working with a finite number of discrete states (as opposed to an infinite number of continuous states). A continuous mathematical function is defined by the mapping

$$f : S \rightarrow \mathbb{R}, \quad (1.4)$$

where the set S is infinite and continuous on a finite state machine forces a requirement for S to be discretised to some subset S' which is both finite and discrete. So we must define a new mapping

$$g : S' \rightarrow \mathbb{R}, \quad (1.5)$$

where both S' and \mathbb{R} are both finite discrete sets.

Let us now introduce the idea of "input-output" pairs. We have some search space grid that defines the set S' . The "input" defines the point on the search space grid $x \in S'$. The "output" is the value of the mathematical function at that point $g(x)$ where $x \in S'$. A search space grid and a mathematical function f can therefore be thought of as a discrete and finite set of input-output pairs $g : S' \rightarrow \mathbb{R}$, where $\mathbb{R} \subset \mathbb{R}$ as the output must also be discretised to exist on a finite state machine. Consequently, for a finite state machine, there are finitely many input-output pairs; finitely many functions.

There are three parts to be considered in calculating the complete set of input-output pairs of a discretised mathematical function (let us assume the

function is one dimensional for conceptual simplicity, although the outcomes will also hold for higher dimensional functions):

- Representation of the input domain.
- Representation of the function.
- Representation of the output domain.

Only a finite number of points can be mapped to, as computers only have a finite number of states. Potentially this number could be huge, as modern finite state machines contain many states. For practical purposes let us limit the discussion to typical binary strings of arbitrary finite length l .

Whether the string maps to a fixed point representation (constant granulation), a floating point representation (variable granulation), a more obscure number system, or even some completely abstract representation, the total number of states that can be represented remains the same,

$$|S^l| = 2^l. \quad (1.6)$$

It is worth acknowledging that floating point numbers have an implicit bit of mantissa accuracy; it should also be noted however, that this does not provide any additional states of the machine, or points in the search space.

We argue that it is unreasonable to assume that in general the global optimum of an arbitrary mathematical function maps exactly onto any of the input-output pairs. There are many examples in which it does, these occur either because the search space grid has been chosen to fit the function, or because the function has been chosen to fit the search space grid. In fact, the input value that corresponds to the true mathematical optimum, will *almost surely not* (a mathematical term meaning a probability of 0) be exactly represented, if the location of the optima is not known when choosing the search space grid. Hence if $f(x) = (x - \pi)^2$ where $x \in S$ and $g(x) = (x - \pi)^2$ where $x \in S'$, there is no guarantee that

$\min(f(x)) = \min(g(x))$, as f and g are not the same mapping. Whether the minima are equal is dependent on the values $x \in S'$ containing a value of x that matches up with the minimum of $f(x)$.

To highlight this, we present a basic thought experiment concerning how a continuous mathematical function is represented as a set of input output pairs. The input output pairs defined by g are all that the algorithm can "see", the path of the mathematical function between states cannot be known by the optimisation algorithm. In Figure 1 the grey dots represent the search space grid. In A, the black line represents the continuous mathematical function f and the black dot on the line is the optimum of the function. In B, the dots with a diagonal black line through them are non-optimal input-output pairs, the dots with the crosses through them are optimal input-output pairs.

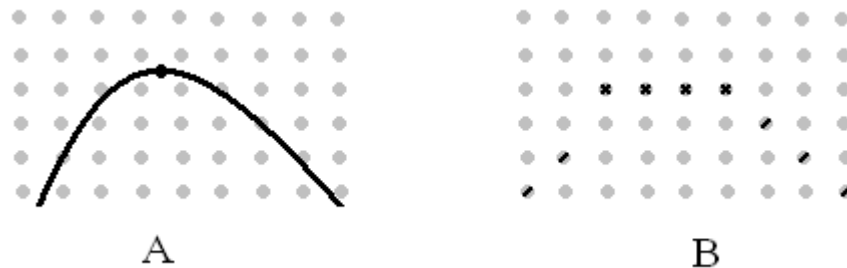


Figure 1. Mapping a continuous function onto a discrete search space grid. The grey dots show the search space grid

When we examine the representation of the function in Figure 1 B, we see that there are four points of equal affinity. These points all have crosses through them to show that they are all equally valid optima of this representation of the function. So in this example, not only is the optimum of the mathematical function not represented exactly, but the number of optima is not the same as the mathematical function. This demonstrates that the representation of a continuous function on a discrete space is far from trivial.

The issues raised thus far have been theoretically important, in terms of noting the issues of converting a continuous mathematical function onto a finite set of points. In practical terms the matter of the optima being incorrectly located by a matter of a few grid points is negligible for most practical applications. There is however another aspect of converting a continuous mathematical function into a finite set of points, which has a potentially large impact for a range of applications. When an optimisation algorithm is applied to a real world problem, often little or nothing is known about the search space. It is therefore the case that there is no way of determining that the true global optima exist in the range covered by the optimisation algorithm.

This issue is that the true global optima may exist outside the range of the search space. Without additional knowledge of the function, it is not possible to make statements about the number or location of the global optima based on the outcome of computational optimisation algorithm.

In some practical cases there may be sufficient domain knowledge to have a high degree of confidence that the global optima of the function are included within the search space grid. Or the value of the optima may be known: this is often the case in benchmark functions where the distance from the optima is used as a performance measure. In some real world problem domains, parameters may relate to control settings that have a physically limited range. Therefore this issue is not always of critical importance, but certainly should not be disregarded.

The problems that arise from representing an infinite continuous set by a discrete finite set are non-trivial. This half of the problem of computational optimisation presents at least an equal challenge as the behaviour of the algorithm. In fact, as we separate out the issues of representation and algorithm behaviour, the problem of algorithm behaviour becomes well posed.

We now extricate the behaviour of the algorithm from the issues of picking an appropriate representation. This is done by essentially discarding the mathematical function and considering the behaviour and performance of the algorithm purely on the set of input output states. The objective of optimisation algorithms is to find the “search space global optima” which we define here.

Search space global optima (SSGO): A point in the search space is considered to be a global optimum if there is no other point with higher affinity (points with equal affinity are equally valid global optima). All functions have at least one SSGO as long as the search space is not the empty set.

The objective of a computational optimisation is to find the SSGO, not the global optima of the function. Having made this statement we are now in line with the situation quoted from (Rudolph, 1994a) in this section. It is important to be fully aware of the issues associated with representing a continuous function on a discrete search space grid. A function and a search space grid define a discrete set of input-output pairs which contain at least one SSGO. A function, with the choice of search space grid left undefined, may be used to define an infinite number of different sets of input-output pairs.

1.3 All representations map to binary strings

In this section we shall consider various discrete representations and how they relate to each other, also how they relate to the continuous domains, such as \mathbb{R} used to define f in equation (1.4).

Floating point numbers being discrete and finite, are not real numbers in the mathematical sense. Perhaps in an engineering context they can be treated as such, but from a theoretical viewpoint floating point numbers

cannot be considered to be real numbers. From a theoretical viewpoint, they are in fact not particularly different from a standard integer or fixed point binary string when considered in terms of their mathematical properties. It is fairly easy to demonstrate that the floating point numbers can be mapped exactly to a standard integer binary string of the same length and vice versa: Consider that two classes of binary strings of the same length have the same number of states. These states are ordered, or can be assigned an order (as they are countable). State 1 of the floating point string can be mapped to state 1 of the fixed point string and so on. In this way all representations (that can be represented on a digital computer) can be mapped to each other via an ordered set of states and consequently to a standard binary string. More simply: all digital computers run on binary, hence all representations (that can be represented on a digital computer) must map to a binary state space.

Essentially if the behaviour of an algorithm can be determined for an arbitrary representation then its behaviour can be determined in all other representation by a simple mapping. In general we will work with an abstract "state" referring to the ordering of the points in the space rather than their explicit locations in \mathbb{R} . For conceptual convenience, we shall let this state be mapped to a fixed point binary string, which could in principle map to any representation and hence is general. For examples we will use the integer representation as it has conceptual simplicity. The simplicity being that the state number is the same as the number given by the representation: state 000 = 0, state 001 = 1 and so on.

Having said that binary representations and floating point representations are not fundamentally different in their mathematical properties, it is important to say that the choice of representation may have a large impact on the performance of the algorithm on a specific problem. A basic observation can be made that different representations can cover different ranges, so they have an inherent difference. Extending this observation, projecting a mathematical function onto two different representations will, in the cases where the two representations define different search space

grids, give two very different sets of input-output pairs. As the input-output pairs are all the optimisation algorithm can see, these are effectively two different optimisation problems g_1 and g_2 , despite both representing the same mathematical function f . So, it is only to be expected that empirical observations show the choice of representation has a potentially large effect on the performance of an algorithm, given that the representation can in effect radically change the set of input-output pairs to be optimised, making the optimisation problem easier or harder, regardless of the fact that both optimisation problems are derived from the same function f .

1.4 Black box optimisation

Having defined search on a digital platform, we essentially have a set of known input states that map to a set of unknown output states by some function g . The objective of the search is to find the input value(s) that are mapped to the optimal output value(s). Picking the appropriate representation and algorithm for an optimisation problem, can only really be done with "domain knowledge". There are three levels of domain knowledge, defined as:

1. A white box problem: The positions of the optima are known and hence the value of the optima is also known (at the cost of one function evaluation), either because the problem has previously been solved exhaustively, or the function itself is analytic; either way all the information required from the search is known *a priori*.
2. A grey box problem: The value of the optimal output is known (and is known to lie on a search space grid point), but the positions of the optima are not known. Some of the information required from the search is known *a priori*.
3. A black box problem. The value and position of the optima are unknown.

Searching white box problems is only useful for the purposes of benchmarking algorithms. Obviously some information needs to be discarded (the location of the optima for example). The amount of *a priori* information discarded varies from one set of experiments to another.

Algorithms that use the "distance" to the optima as part of the fitness criteria or stopping criteria are tackling a grey box problem.

Benchmark problems are often treated as black box problems, although even when intending to treat a problem as a black box some information may inadvertently be retained, it is often known that the optimum of the function $f, x^* \in S$ is exactly represented in the input domain of the discretised function $g, x^* \in S'$. This usually arises not because the practitioner has taken knowledge from the problem, but because the original choice of the problem took knowledge of a common representation. Treating a white box or grey box optimisation problem, as a black box is more difficult than it might appear. There can be *a priori* information unintentionally built into the algorithm's design, such as the representation used in the mutation operator, e.g. the practitioner may well have a simple initial benchmark problem in mind which they intend to use to test the algorithm that they are about to design. If this test problem that has been considered is the ones counting problem (Kallel et al., 2001) the likely outcome is that the practitioner will choose to use hamming mutation operators when designing their algorithm.

Although they can then run this algorithm on the "ones counting" (a standard optimisation problem where the fitness score relates directly to the number of 1s in the binary string) problem (Kallel et al., 2001) as if it were a black box problem, they may well discard the obvious information that makes this a white box problem, but if the design of the algorithm has a particular application domain in mind, then it is more than possible that some knowledge of the application domain will be used in the design of the algorithm. It is even possible to inherit such assumptions by using operators developed in other algorithms.

1.5 Operators

Operators used in optimisation algorithms can be split into two categories: "search operators" and "state operators":

Search operators are defined here to be operators that affect which points in the search space should be evaluated during the iteration based on the current state of the algorithm.

State operators are defined here to be operators that affect how some internal state of the optimisation algorithm is updated based on the result of the current iterations function evaluations.

1.5.1 Search operators

The role of a search operator is to generate a point (or set of points) in the search space which the algorithm will evaluate, based on the information available to the algorithm at the time. In the case of stochastic searches the search operator defines a probability distribution over all of the points in the search space.

A typical search operator is a mutation operator, which takes in a position in the search space and mutates it to give a new state to be searched. Population based operators such as a crossover operator (used in GAs) are also search operators as they influence which states in the state space will be searched (Nix and Vose, 1992). Search operators can be combined consecutively provided they meet each other's relevant input and output requirements. Combining of search operators makes no fundamental difference, as no matter how many search operators are combined they still will produce a probability distribution over the states, or points in the search space.

1.5.2 State operators

State operators are used to guide the search, by controlling how the state of the algorithm is updated, or to keep track of the “best so far” solution. A “memory set” can be applied passively to any algorithm, it simply keeps a copy of the “best so far” candidate solution (or candidate population). An elitist operator also maintains the best so far candidate solution, but does so within the set of candidate solutions which affects the state of the algorithm. Memory sets and elitist operators are distinguished by whether or not they affect the dynamics of the algorithm.

Operators such as elitism are not search operators, as after applying the elitism operator, a point (or set of points) to be evaluated is not produced. Elitism does however affect how the state of the algorithm is updated.

Algorithms that do not have an elitist operator require a memory set in order to avoid losing the “best so far” solution. This has been shown to be the case for canonical GAs (Rudolph, 1994a). Algorithms without elitism form an indiscriminate search and are able to “drift” through the search space. The state of the algorithm is determined entirely by the search operator; the algorithm does not “discriminate” between searching a state that is better than the current state and a state that is worse than the current state.

In the case of an indiscriminate search algorithm with a memory set, the appropriate thing to examine is the transition between states in the memory set. In order to determine the probability between transitions in the memory set, the associated indiscriminate search algorithm must be modelled including the memory as part of the definition of the state of the algorithm.

An elitist algorithm will have very different dynamics compared to an algorithm using the same search operator (without elitism) and a memory set. The indiscriminate algorithm can drift through the search space regardless of whether it is finding states of higher affinity or not. This makes escaping local optima trivial. However, the algorithm is equally likely to drift away from the global optima in the same way. In contrast an elitist algorithm will have to jump from a local optimum to a state with higher affinity in one iteration; the probability of doing this may be very small, so elitist algorithms may have more difficulty escaping local optima. However, once an elitist algorithm is close to a global optimum, the probability of jumping away from that global optimum may also be very small.

1.6 Convergence

For optimisation algorithms the rate of convergence is critical. Empirical studies on the rate of convergence cannot provide general evidence that the algorithm will always converge to the search space global optima.

To converge, in a mathematical sense, is to approach a limit. In terms of function optimisation the "limit" should be one of the global optima. The meaning of the word converge has been diluted due in part to the uncertainty about whether the algorithm under discussion will always find the optima. The meaning of the word has been further diluted by the use of the term "premature convergence". In function optimisation, the word converge is used both to describe approaching the optima and simply to have a population that is all the same. The problem of "premature convergence" is often commented on in the GA literature (Andre et al., 2001, Schraudolph and Belew, 1992). However, Vose has shown that a simple genetic algorithm will "transit through all states infinitely often" (Vose, 1995), similarly Rudolph has shown that the canonical genetic algorithm will not converge "even in infinite time" (Rudolph, 1994a). Quoting from Eiben (Eiben and Rudolph, 1999) "The term 'convergence' in GAs mostly denotes the phenomenon that the population approaches a

state where it consists of multiple copies of the same bit string. This differs from the traditional use of the word, standing for the approximation of a solution". Hence, the common usage of "convergence" in the GA community does not comply with the conventional mathematical meaning of convergence, thus introducing an ambiguity.

We define the term "Convergent Absolute" enabling strong mathematical statements about convergence by tailoring the strict mathematical definition of convergence to the specifics of computational functional optimisation.

Convergent Absolute: An algorithm can be said to be convergent absolute if and only if it can be shown to converge on the search space global optima with probability 1 from all points in the search space, for all functions.

Recall that search space global optima (SSGO) was defined in section 1.2. One of the most obvious things to do with an exact model of an optimisation algorithm is to look at its long term behaviour. Specifically, to find out if the algorithm is convergent absolute or not.

Given that the algorithms are stochastic, it can be easily deduced that an algorithm can only be shown to be convergent absolute in an arbitrarily large amount of time; the proofs of convergence presented by Villalobos-Arias (Villalobos-Arias et al., 2004), Häggström (Häggström, 2002) and section 3.5 are in infinite time. One might argue that attempting such a proof is of little practical value. There are however some potential benefits to be gained. If a proof could be kept sufficiently general, it may provide conditions that can be used to test if a large variety of algorithms are convergent absolute or not. If only some algorithms are convergent absolute, algorithms that do not satisfy the conditions to be convergent absolute would have to have significant short term advantages in order to be considered advantageous over an algorithm that is convergent absolute.

Proving that an algorithm is convergent absolute (or not) in infinite time has some value in itself, even if it is purely as an academic exercise.

Construction of such a proof will require the development of theoretical tools, many of which may be useful in the wider analysis of the optimisation algorithm. Once the theoretical tools are in place, a more practical goal would be to prove that the algorithm can find an optimum of a function with some probability less than 1. It would be possible to prove the algorithm finds an optimum in a finite number of function evaluations if the probability with which it finds an optimum is less than 1. Such a proof may have practical value, allowing different algorithms to be theoretically benchmarked.

1.7 Publications

Publications in which I am the first author include one journal paper (Clark et al., 2006) and one conference paper (Clark et al., 2005). I have also been a contributing author on a further four journal papers, a research note and three conference papers with a further conference paper and journal paper under review. In total I have published a research note, 5 journal papers, 3 conference papers.

Publications that relate directly to the theoretical core of this thesis are (Clark et al., 2005) in which a Markov chain model of an immune system algorithm is presented and a proof of convergence for that algorithm is given. The content of this publication is covered in sections 3.4, 3.5 and 3.6. A review of theoretical advances in the field of AIS (Timmis et al., 2008c), containing a summary and review of the Markov chain models in the field of AIS as well as other theoretical approaches within the field.

As well as my publications which are of direct relevance to this thesis, I have publications in the wider context of AIS, publications include (Timmis et al., 2008a, Timmis et al., 2008b, Secker et al., 2008) that are pertinent in a peripheral sense. I have further publications in the areas of bio-informatics (Davies et al., 2008b, Davies et al., 2008c, Davies et al., 2008a) and in condensed matter (Clark et al., 2006).

Awaiting review are an invited journal paper that is an extension of (Secker et al., 2008) and a conference paper on the simulation of a bacterial matabolome (Hickinbotham et al., 2009).

1.8 Order of material to come

Chapter 2 presents background on Markov chains and an overview of the Markov chain models in the literature of the fields of Simulated Annealing, Evolutionary Algorithms and Artificial Immune Systems. A discussion of the literature is given from which the thesis draws motivation for following a novel modular modelling approach.

Chapter 3 presents a case study on constructing mathematical models of the B-Cell Algorithm (BCA) and several of its sub-algorithms. The modular modelling methodology is developed, by building on the simpler sub-algorithms to produce models of the more complicated algorithms and finally to produce a complete and exact mathematical representation of the BCA. Based on this case study of modular modelling of an optimisation algorithm we develop a detailed theoretical understanding of this algorithm including a proof of convergence presented in chapters 3 and bounds on the rate of convergence of the BCA presented in chapter 4.

Chapter 4 investigates approximate methods of calculating the rate of convergence of elitist algorithms and as a result produces bounds on the rate of convergence for an unknown function. The bounds obtained here are compared with the bounds obtained in the GA literature and are found to be equivalent. A discussion of bounding on unknown functions is given and a lower limit to the upper bound is obtained that demonstrates that bounding the rate of the convergence on an *unknown function* is not of practical interest.

Chapter 5 presents the framework for modelling stochastic optimisation algorithms. The framework is based on the case study in chapter 3, but also considers operators from EA, SA and AIS. The modular approach motivated in chapter 2 and investigated in chapter 3 is continued. Operators are subdivided into search operators and state operators, each of which is modelled by a matrix. A general method of combining these matrices to give the transition matrix of a Markov chain is obtained. A practical guide for constructing novel models with the framework is given.

Chapter 6 applies the framework developed in chapter 5 and presents three Markov chain models with worked examples obtaining the numerical transition matrix for each model. The models include a model of SA given in terms of the framework which is exactly equivalent to a model in the literature, demonstrating the ability of the framework to include existing Markov chain models. A model of hill climber is given to demonstrate the flexibility of the framework, modelling all classes of optimisation algorithm. A novel model of the $(1+\lambda)$ ES with fitness proportional mutation is constructed demonstrating the use of the framework to construct novel models.

Chapter 7 presents a discussion of the thesis, the key results of the thesis are summarised and discussed and the conclusions of the thesis are given. Discussion of future directions focuses on how the framework can be utilised in such a way that the results are useful to practitioners.

A chapter dependency diagram for this thesis is given in Figure 2

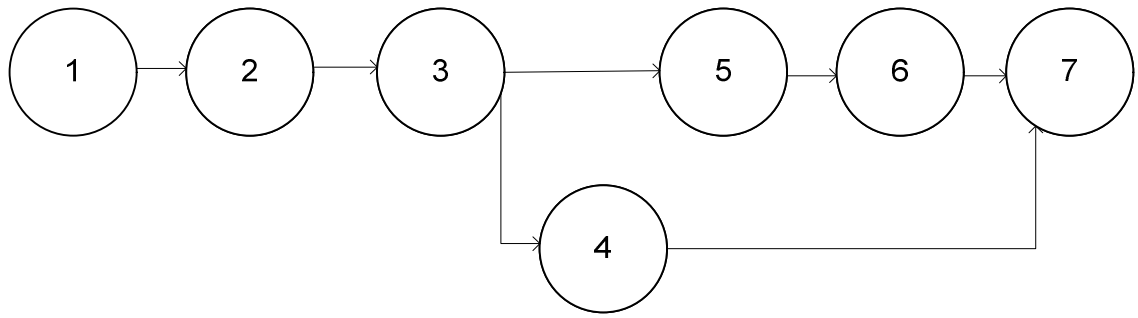


Figure 2. Chapter dependency diagram

Chapter 2 Background

Models presented in this thesis take the form of a Markov chain. The decision to form the models into a Markov chain is made in section 3.3.6 where it is demonstrated to be the natural representation in the context of the case study. A Markov chain is a commonly adopted form for a model to take in cases where algorithms can be shown to satisfy the Markov property. In this chapter we present material pertaining to the use of Markov chains to model optimisation algorithms, providing a good background for the remaining thesis.

In order to contextualise the models in chapter 3 and subsequent chapters, a review of Markov chains as applied to the modelling of algorithms from the fields of Simulated Annealing (SA), Evolutionary Algorithms, which incorporates Evolutionary Strategies (ES) and Genetic Algorithms (GA), and Artificial Immune Systems (AIS). A brief overview of Markov chains precedes the review of the application of Markov chains to model optimisation algorithms. The fields of optimisation algorithms are reviewed in the order in which Markov chain models first appear.

The chapter concludes with a discussion of key themes that are identified in the literature across all fields and we draw motivation for the direction of the research undertaken in this thesis.

2.1 Markov chains

This section presents a general background on Markov chains. The reader is referred to chapter 2 of (Häggström, 2002), which gives a formal yet hands on introduction to Markov chains. For a more traditional introduction to Markov chains see Seneta (Seneta, 1981).

2.1.1 The Markov property

Formally, a Markov chain is a sequence of random variables X_0, X_1, X_2, \dots that satisfies the Markov property: The probability of the system being in any given state at the next time step depends only on the state of the system at the current time step and is independent of the previous states of the system. This is sometimes also known as the “memoryless property”

$$p(X_{n+1} = x | X_n = x_n, \dots, X_0 = x_0) = p(X_{n+1} = x | X_n = x_n). \quad (2.1)$$

The possible values of the random variable X_i form a countable set referred to as the “state space of the chain”.

2.1.2 Matrix properties

Definitions adapted from Rudolph (Rudolph, 1994a)

A square matrix $\mathbf{P} = (p_{ij})$: $N \times N$ is said to be

(a) nonnegative ($\mathbf{P} \geq 0$), if $p_{ij} \geq 0$ for all $i, j \in \{1, \dots, N\}$

(b) positive ($\mathbf{P} > 0$) if $p_{ij} > 0$ for all $i, j \in \{1, \dots, N\}$

A nonnegative square matrix $\mathbf{P} = (p_{ij})$: $N \times N$ is said to be

(c) primitive, if there exists a $n \in \mathbb{N}$ such that \mathbf{P}^n is positive,

(d) reducible, if \mathbf{P} can be brought into the form (with square matrices \mathbf{C} and \mathbf{T})

$$\begin{pmatrix} \mathbf{C} & \mathbf{0} \\ \mathbf{R} & \mathbf{T} \end{pmatrix}$$

by applying the same permutations to rows and columns,

(e) irreducible, if it is not reducible

(f) stochastic, if $\sum_{j=1}^N p_{ij} = 1$ for all $i \in \{1, \dots, N\}$

A stochastic matrix $\mathbf{P} = (p_{ij})$: $N \times N$ is said to be

(g) stable, if it has identical rows

(h) column allowable if it has at least one positive entry in each column

Matrices that describe the transition probabilities in a Markov chain are stochastic matrices and will be referred to as a “transition matrix”. The matrices formed by modelling algorithms can be either reducible as in chapter 3, published (Clark et al., 2005), or irreducible as in (Villalobos-Arias et al., 2004). It is important to note which form the transition matrix takes; as whether the matrix is reducible or not is important for determining the properties of the algorithm.

2.1.3 Optimisation algorithms and the Markov property

In section 2.1.1 we formally introduced the Markov property, now we will see if that definition is satisfied by just some, or all, optimisation algorithms. We shall demonstrate here by a simple thought experiment that many optimisation algorithms satisfy the Markov property: consider an optimisation algorithm where the position in the search space is the “state of the algorithm”. A search operator is applied to this *state* to generate the next point in the search space to be evaluated and thus the next state of the algorithm. The search operator defines some probability distribution over the possible points in the search space. If the search operator acts only on the current state of the algorithm and not the previous states, then the algorithm satisfies the Markov property.

Based on the above thought experiment we can conclude that as long as the search operator does not act on previous states of the algorithm; only the current state, then it will satisfy the Markov property. Now let us consider a search operator that acts on the current position in the search

space and the previous position in the search space to define a probability distribution of being in any given position in the search space at the next time step. Note the shift in language: we are no longer referring to the position in the search space as the state of the algorithm. This is because if the current position in the search space was the state of the algorithm, then the algorithm would not satisfy the Markov property. However, no restrictions have been made as to what the *state* of the algorithm is. For an algorithm that has a search operator that is dependent on both the current position and the previous position of the algorithm in the search space we can define the state of the algorithm to be the current and one previous position of the algorithm in the search space. Thus the search operator and hence the algorithm now satisfy the Markov property.

There is unfortunately a price to pay for defining the state of the algorithm in this way and that price is the size of the state space. If the number of states of the algorithm that only requires the current position in the search space to define a state is S , then the number of states of the algorithm that requires both the current and one previous position of the algorithm would (all other factors being kept the same) have a state space of S^2 .

Based on our thought experiment it appeared that we would be able to design an algorithm based on the definition of the Markov property, such that the algorithm would not satisfy the Markov property. However, *by redefining what the state of the algorithm is, we were able to make the algorithm satisfy the Markov property.* Hence we have failed to find an algorithm that does not satisfy the Markov property.

Let us change the level of detail of which we consider whether all optimisation algorithms satisfy the Markov property. Let us consider an algorithm implemented on a finite state machine. We propose that finite state machines naturally satisfy the Markov property, although they have extremely large (but finite) state spaces. If one accepts that a finite state machine satisfies the Markov property, and that all "algorithms" can be implemented on a finite state machine, it follows that all algorithms must

also satisfy the Markov property. There is however a non-trivial problem of determining what can we define as the state of the algorithm such that the algorithm satisfies the Markov property.

2.1.4 Tractability of numerical Markov chain calculations

Let us consider how the size of the state space affects the tractability of numerical calculations. If the number of states of an algorithm is given by “the minimal state space”

$$|S| = 2^l \tag{2.2}$$

Where 2^l is the number of points in a search space defined by a binary string of length l , then the number of elements in the transition matrix of the algorithm is given by $|S|^2 = 2^{2l}$; the number of elements in the matrix increases faster than exponentially (e^l). A non-minimal state space will increase the size of the matrix even faster as a function of l . There is a serious issue of tractability for any numerical work. It is important when constructing Markov chain models of algorithms to keep the definition of the state of the algorithm to the bare minimum required to allow the algorithm to conform to the Markov property.

2.2 Simulated Annealing

Simulated Annealing (SA), originally published by Kirkpatrick (Kirkpatrick et al., 1983) and independently achieved by (Černý, 1985), is an optimisation method that takes its inspiration from annealing in metallurgy. Materials heated and then cooled on a specified temperature schedule, allow atoms to find lower energy states, which at the macroscopic level means that the material can be formed into larger crystals with fewer defects. In the

natural situation, the energy of the system is reduced. The process is sensitive to the temperature schedule used, fast schedules cause poor energy minimisation within the material, while slow schedules result in better energy minimisation.

Simulated Annealing algorithms have two characteristic features: firstly, the search operator is confined to points in the search space that are close to the current position of the algorithm. This is analogous to atoms moving within a material. Secondly: the probability of accepting a newly searched point that has a higher "energy" than the current point is given by an equation of the form

$$p = e^{-\left(\frac{g(x_j) - g(x_i)}{T}\right)} \quad (2.3)$$

Where x_i is the current state of the algorithm and x_j is the proposed next state of the algorithm. $g(x_i)$ and $g(x_j)$ are the values of the objective functions for their respective states, specified to be $g(x_j) > g(x_i)$ and T is the temperature.

In equation (2.3) $g(x_i)$ is analogous to the energy of a configuration of atoms in the natural system. For large values of T , changes that increase the energy of the system can occur frequently, allowing the algorithm to search the space. As T is reduced, such changes are rejected more frequently; and as $T \rightarrow 0$, $p \rightarrow 0$ and such changes are forbidden.

Defining an appropriate temperature schedule is critical for simulated annealing algorithms: if the temperature is reduced too quickly (in too few a number of iterations), then the algorithm may become trapped in a local minimum, which may be a poor quality solution. If, on the other hand, the schedule reduces the temperature too slowly, then an intractable number of iterations will be required before the algorithm terminates.

2.2.1 Simulated annealing Markov chain models

Markov chain models of simulated annealing began appearing in the literature very shortly after the field originated in 1983. A complete and exact time-inhomogeneous Markov chain is presented by (Mitra et al., 1985). The authors show the chain to be strongly ergodic, but go on to show that the algorithm converges to the global optimum if a sufficiently slow temperature schedule is used. The authors also obtain a bound on the rate of convergence and postulate how this may lead to the development of an optimal annealing schedule.

Gidas (Gidas, 1985) examines the asymptotic behaviour of their Markov chain as $t \rightarrow \infty$ and uses their analysis to provide a procedure for obtaining an "optimal annealing schedule" (Gidas, 1985). Gidas goes on to explain the trade off the "adiabatic" effect, whereby if the temperature is lowered too quickly the system may end up in a local optimum, and the "super-cooling" effect, whereby if the temperature is lowered too slowly the system will converge on the global optimum, but extremely slowly. Tsitsiklis presents some approximate methods and expand some of these discrete time Markov chain results to continuous time Markov chains (Tsitsiklis, 1985).

In 1987 Laarhoven concluded "that the theoretical basis of the algorithm had reached a certain level of saturation" (Laarhoven and Aarts, 1987). Laarhoven provides a wide ranging, but well presented review of the theory and applications of simulated annealing (Laarhoven and Aarts, 1987). Given that the SA community had proven the convergence of the algorithm, assuming a bounded annealing schedule (Mitra et al., 1985, Gidas, 1985) and derived conditions for which an optimal annealing schedule can be obtained (Mitra et al., 1985, Gidas, 1985), the views of Laarhoven seem justified. Indeed we present no further fundamental advancements, we do however review the work of (Häggström, 2002). Although the work presented by Häggström is not novel, it is extremely well presented and representative of Markov chain models of SA.

Häggström presents a Markov chain model of simulated annealing (Häggström, 2002). The book, being primarily about Markov chains and their applications, approaches the problem by constructing a simulated annealing algorithm from a Markov chain, as opposed to attempting to construct a Markov chain model based on the algorithm. As such, there is at no point in the chapter the pseudo-code for the simulated annealing algorithm, as the model itself serves as the algorithm specification. As the requirements of the model call for far more specific statements about the operators of the algorithm, this is actually a highly effective way of describing an algorithm.

Häggström also presents a good notation for construction of an inhomogeneous Markov chain: Quoting from:

“If the first Markov chain has transition matrix P' and is run for time N_1 , the second Markov chain has transition matrix P'' and is run for time N_2 , and so on, then the whole algorithm can be viewed as an inhomogeneous Markov chain with transition matrices

$$P^{(n)} = \begin{cases} P' & \text{for } n = 1, \dots, N_1 \\ P'' & \text{for } n = N_1 + 1, \dots, N_1 + N_2 \\ \vdots & \vdots \end{cases} \quad (2.4)$$

(Häggström, 2002)

Häggström goes on to show that for a sufficiently slow cooling schedule, the probability of being in the optimum at time n does tend to 1 as $n \rightarrow \infty$. Häggström includes a numerical example, going from the algebraic transition matrix/algorithm he has constructed to a numerical transition matrix for a particular example toy problem. This tangible example of a numerical transition matrix makes it that much easier to resolve any

uncertainties the reader may have. It is an excellent device to open up the chapter to a wider audience.

We adopt several ideas from (Häggström, 2002): The use of numerical examples to add clarity is taken up in general and is used throughout the thesis, but is particularly evident in chapter 6. The method of construction of time inhomogeneous chains Häggström has influenced the method used in the framework in chapter 5. The model of simulated annealing presented by Häggström is re-instantiated in chapter 6 in terms of the framework we develop in chapter 5.

2.3 Evolutionary Algorithms

The term Evolution Algorithm (EA) is a superclass of algorithms encompassing several streams of research: Genetic Algorithms (GA), Evolutionary Strategies (ES), Genetic Programming (GP) and Evolutionary Programming (EP) (Eiben and Rudolph, 1999). The four streams are linked because they are all based on the principles of biological evolution; of this superclass we consider the algorithms applied to function optimisation: GA and ES.

2.3.1 Genetic Algorithms

Genetic Algorithms (GAs) (Holland, 1975), are a form of what has become known as Evolutionary Algorithm (EA), or Evolutionary Computation (EC). A good review of the roots of the area is presented by Fogel (Fogel, 1998). GAs are based on evolutionary ideas such as selection and inheritance, with particular focus on how genes are transferred from parents (via "sexual reproduction") to children, giving rise to ideas of crossover and mutation of the genetic code.

The operator that particularly characterises GAs is the cross-over operator; two "parent" strings are selected from the current population, one or more points is selected on what is their analogy to a string of genetic code (commonly a binary string) and the strings are recombined such that a child contains elements of both parents. The child's string is also subject to mutation allowing for a global search of the space.

2.3.2 Evolutionary Strategies

The field of Evolutionary Strategies (ES) is derived from Rechenberg's seminal thesis (Rechenberg, 1971). Note: Rechenberg's thesis is in German, we have not read it, but its content is reported by many sources, for example (Rudolph, 1994b, Weicker and Weicker, 2003). ES is another branch of EC/EA and is based on evolutionary ideas. A good review of the area is presented by Beyer (Beyer and Schwefel, 2002). The core idea behind ES is Darwinian evolution, viewed at an abstract level. A number of copies are made of the current best candidate solution, each of these is mutated and then compared with the parent. The best of the parent and all the mutants will be the parent for the next generation.

ES have a succinct common notation of the form $(1+\lambda)$, which in this case specifies that there is one parent and that parent spawns λ mutants at each generation. More generally there is the $(\mu+\lambda)$ ES where μ parents spawn λ mutants.

2.3.3 Theory of Evolutionary Algorithms

The theory results in evolutionary algorithms are obtained via a variety of methods including Markov chains. An overview of the methods is given by (Eiben and Rudolph, 1999) who highlight key references in a range of analytical techniques including: Dimensional analysis, order statistics,

quantitative genetics, orthogonal functions analysis, Quadratical dynamical systems, statistical physics, Schema theory and Markov chains. We discuss Schema Theorem in section 2.3.4. In section 2.3.5 we review Markov chain models of evolutionary algorithms. For a high level review of the other areas see (Eiben and Rudolph, 1999).

2.3.4 Schema Theorem

This literature review is targeted towards Markov chains; however, it is not possible to review the GA theory literature without commenting on Schema theorem. Holland's schema theorem (Holland, 1975) claims to explain the power of Genetic Algorithms and was the dominant theory in the area until around 1990 (Eiben and Rudolph, 1999). At that time, various critics began to point out fundamental problems with schema theorem: "Schema Theorem is almost a tautology only describing proportional selection" (Mühlenbein, 1991) and that "the question of why the genetic algorithm build better and better substrings is ignored" by Schema Theorem (Mühlenbein, 1991). These views are echoed and stated more formally in (Vose, 1991, Grefenstette and Baker, 1989, Radcliffe, 1991). A good summary of these objections to Schema Theorem is given in (Altenberg, 1995) who also gives a well phrased explanation of how such fundamental problems could have been overlooked:

"What the Schema Theorem says is that schemata with above-average fitness (especially short, low order schemata), increase their frequency in the population each generation at an exponential rate when rare. The mistake is to conclude that this growth of schemata has any implications for the quality of the search carried out by the GA. The Schema Theorem's implication, as many have put it, is that the genetic algorithm is focusing its search on promising regions of the search space, and thus increasing the likelihood that new samples of the search space will have higher fitness. But the phrase 'promising regions of the search space' is a construct through which hidden assumptions are introduced which are not implied by Schema

Theorem. What is a 'region' and what makes it 'promising'?" (Altenberg, 1995)

It can be seen from this, that although the mathematics of Schema Theorem are not erroneous, not every statement of the theorem is supported by the mathematics. The construct through which hidden assumptions are introduced was sufficiently subtle to avoid major opposition for over a decade.

Similar points to those made by (Altenberg, 1995) are made in (Eiben and Rudolph, 1999) who add: "it is implicitly assumed that the problem is separable to some extent. The ignorance of this assumption has led to the 'block building hypothesis' which allegedly explains the working mechanism of GAs." We have quoted from (Altenberg, 1995) that the "schemata with above-average fitness increase their frequency in the population each generation at an exponential rate when rare". In order for this to lead to a high probability of a global optimum via crossover, there also needs to be a high frequency of the complementary schemata. This is what is meant by the claim made by (Eiben and Rudolph, 1999) that the problem is assumed to be separable to some extent.

This debunking of schema theorem by the GA community coincides with the early Markov chain work in the area (Vose and Liepins, 1991) who present a Markov chain model of a GA with an infinite population size. This paper lays the foundations for the seminal Nix and Vose model of a simple GA (Nix and Vose, 1992).

2.3.5 Markov chain models of Evolutionary Algorithms

There are a number of Markov chain models of evolutionary algorithms, both of genetic algorithms (Nix and Vose, 1992, Davis, 1992, Rudolph, 1994a) and evolutionary strategies (Rudolph, 1994b). Many of the models and Markov chain results obtained are covered in Rudolph's seminal review

paper: finite Markov chain results in Evolutionary computation a tour d'horizon (Rudolph, 1998).

It is noticeable that with the exception of Davis (Davis, 1992), the papers in which the Markov chain models of EAs are derived (Nix and Vose, 1992, Rudolph, 1994a, Rudolph, 1994b) make no mention of the related work in the field of simulated annealing which pre-dates all the EA models. Markov chain models of optimisation algorithms are necessarily constructed from first principles, so knowledge of Markov chains and an algorithm specification is all that is required.

In terms of Markov chain models in the field of genetic algorithms, the Nix and Vose model of a simple genetic algorithm (Nix and Vose, 1992) is a seminal piece of work. Although similar results were obtained independently by (Davis, 1992), the Nix and Vose model is the earliest complete and exact Markov chain model of a GA with no simplifications or limiting assumptions. The algorithm includes crossover, a population based operator which leads to a non-minimal state (state of the algorithm) space. The authors show the number of states the algorithm can be in is given by

$$N = \binom{n+r-1}{r-1} \quad (2.5)$$

Where n is the population size and $r = 2^l$, where l is the length of a binary string. This is in contrast to a minimal state space, which using Nix's notation, would be $N = r$.

The authors go on to find the transition matrix for this non-minimal state space that can be translated into a list of strings (the population) by the use of an "incidence vector", which is essentially a very large key relating the state of the algorithm to a population of strings, or positions in the search space.

The Nix and Vose model has one minor limiting factor: The mutation operator and crossover operator are not fully separable. The mutation rate cannot be set to 1 or 0 as it would cause a division by zero due to the form of the equation; this limitation is acknowledged (Nix and Vose, 1992). The importance of this restriction is that the model cannot be used to examine the effects of crossover in the absence of mutation. An instantiation of the algorithm could have the mutation rate set to 0 and run with just crossover, but the model of the algorithm cannot. There is no such restriction placed on the probability of crossover, thus the vast majority of the parameter space is still open to investigation.

The Nix and Vose model spawned two research tracks, a theoretical track, where Markov chain analysis techniques are applied to the transition matrix in order to determine general properties. There is also a more empirical track, undertaking numerical work, applying the model to example search spaces and explicitly observing the changes in probability mass.

The (Nix and Vose, 1992) model of a simple genetic algorithm has provided a firm foundation for rigorous mathematical analysis. The analysis of the model is reviewed and summarised (Vose, 1995) which demonstrates the kind of results that can be obtained through mathematical scrutiny.

The Nix and Vose model of a simple GA takes the form of an irreducible Markov chain, meaning that all states can be revisited or as Vose puts it: "When the mutation rate is greater than zero, a finite-population GA typically forms an ergodic Markov chain, visiting every state infinitely often. Hence, after some period of time a GA will escape *every* local basin." (Vose, 1995). Rudolph's model of a canonical GA also forms an ergodic Markov chain and similar results are obtained (Rudolph, 1994a). In the case of an irreducible or ergodic Markov chain it is appropriate to consider the steady-state distribution from an asymptotic analysis. Having stated that the GA will visit every state "infinitely often", the question is: what is its probable distribution over the states of the algorithm. Analysis of the steady-state distribution leads Vose to conclude that long term behaviour of a GA is to be

at the optimum of the largest basin of attraction (Vose, 1995). This is an interesting result because the largest basin of attraction of a multi-modal function does not necessarily belong to the global optimum. This essentially proves the views expressed by (De Jong, 1992, Rudolph, 1998) that genetic algorithms are not function optimisers as they do not converge to the global optimum. "In optimisation theory an algorithm is said to *converge* to the global optimum if it generates a sequence of solutions or function values in which the global optimum is a limit value" (Rudolph, 1994a). As the model of a GA takes the form of an ergodic Markov chain, we know that this is not the case with the GA as modelled by (Nix and Vose, 1992). With the inclusion of elitism, proofs of the convergence of EAs have been obtained (Eiben et al., 1991, Fogel, 1992). Once it has been established that a EA will transit through all points in the search space and thus the global optimum, one can see that it will be possible to construct a proof of convergence by retaining the best so far solution (via some form of elitism). Such proofs can be constructed even without explicit knowledge of the exact transition probabilities as long as the form of the Markov chain can be established (Eiben et al., 1991, Fogel, 1992). The sequence of best so far solutions will be a sequence in which the global optimum is a limit value; achieving convergence (in the technical sense of the word) on the global optimum.

Rudolph makes a key point regarding proofs of convergence: "Although convergence to the global optimum as the time approaches infinity should be the minimal requirement for a stochastic optimisation algorithm, the knowledge of this property is not of practical interest because the solutions of a finite search space can be enumerated in finite time" (Rudolph, 1994a). This point can be extended to all theoretical methods that require a number of function calls greater than the number of points in the search in order to give a result.

There are several approaches to obtaining results of practical interest in the literature. Bounding the rate of convergence of the algorithm on an unknown problem, early work bounding the rate of convergence of a GA

was undertaken by (Aytug and Koehler, 1996), these bounds were later improved by (Aytug and Koehler, 2000) and obtained independently by (Greenhalgh and Marshall, 2001). These results are discussed and compared in detail with the bounds obtained in thesis in chapter 4.

Other approaches include theoretical investigation of specific classes of functions in finite time, these investigations are restricted to certain problem classes and simple EAs and their results are of course specific to the problem class (Rudolph, 1998). There is also the possibility of numerical investigation of Markov chain models.

A large scale empirical investigation of the Nix and Vose model was conducted by De Jong (De Jong et al., 1995). Due to the rate of increase of the state space described in equation (2.5), the numerical work is limited to toy problems. The numerical work is conducted with parameters such as $l=2, n=5$, $l=3, n=3$ or $l=5, n=2$, where l is the binary string length and n is the population size.

The experiments aim to gain insight into how the mutation rate and crossover rate change the probability of finding the optimum. There is also a study of how the algorithm performs on "deceptive" problems compared with "non-deceptive" problems.

Optimal mutation rates were determined for string lengths of $l=2,3,4,5$ with $n=2$ by a relatively fine grained plot of the mutation rate from $\mu=0.01$ to $\mu=0.99$ in steps of 0.01 (note that the boundary values of $\mu=0, \mu=1$ are excluded from this investigation). The optimal mutation rate found for each string length on the given objective function was used to compare the simple GA with an optimal mutation rate with random search. The model of random search is achieved by setting the mutation rate to $\mu=0.5$ in the Nix and Vose model. Unsurprisingly, the GA with the optimal mutation rate tuned for this particular objective function was found to have a lower Expected Waiting Time (EWT) than random search (De Jong et al.,

1995). EWT is essentially the expected number of iterations required before the string representing the optima first appears in the population.

The effects of crossover on the EWT were also looked at in a coarse parameter search, as well as a coarse investigation into the interaction between mutation rate and crossover rate.

The course of experiments laid out was interesting, although the ability of the results of the experiments to form a solid basis for conclusions about the experiments is significantly undermined by the population size and string length. If the numerical work, originally published in 1995, were extended to push the limits of current computational capabilities, the range of parameters that could be investigated may be large enough to provide theoretically optimal parameter settings for any class of problem investigated.

Initial visualisation work was also undertaken in (De Jong et al., 1995). Visualisation in this context is achieved by considering the transition matrix as an image, where the image is created by mapping the probability of each element in the matrix to a colour (on the grey scale from 0 to 225) based on its probability. This process can be applied to any transition matrix, hence the process can be used to observe the flow of probability as the one step transition matrix is raised to a variety of powers. This visualisation process made it apparent to the authors that the ordering of states used by (Nix and Vose, 1992) was "not intuitive".

These ideas were extended in (Spears and De Jong, 1997) where visualisations of transition matrices with states ordered by a variety of methods including: average fitness of the population, average hamming distance between all pairs in the population and by probability mass. While many of the orderings explored shifted the high probability states to the left edge of the matrix, all the orderings are dependent on knowledge of the fitness function. In order to construct an explicit Markov chain, explicit knowledge of the fitness function is required, so in principle the approach is

appropriate. However, including knowledge of the fitness function in the ordering of the states means that for different problems, the states may be ordered differently. Although the authors point out features of interest for each of the orderings, none of the orderings presented in (Spears and De Jong, 1997) showed a definitive improvement over the lexicographic ordering of (Nix and Vose, 1992).

Spears and De Jong acknowledge that full state space Markov models are not likely to scale to “real word configurations”; they present preliminary ideas on semantically lumped models, where similar states are combined to reduce the size of the state space. A semantically lumped model is an approximate method. The accuracy of the approximation is related to the number of states in the semantically lumped model, compared to that of the full model.

If one wished to explore further semantically lumped state models of algorithms, then the semantic ordering of states may be a useful way to automate the lumping process. Approximate methods such as the semantic lumping of states could in principle allow numerical work on non-toy sized problems.

2.4 Artificial Immune Systems

The field of Artificial Immune Systems (AIS) grew from early works by Farmer (Farmer et al., 1986) and Bersini (Bersini, 1991, Bersini and Varela, 1991a, Bersini and Varela, 1991b). AIS encompasses several types of immune inspired algorithms: Negative Selection (Dasgupta and Gonzalez, 2002), Immune Networks (Bersini, 1991), Clonal Selection (De Castro and Von Zuben, 2000) and Danger Theory (Greensmith et al., 2004).

We look in detail at two clonal selection algorithms: the B-Cell Algorithm (BCA) proposed by Kelsey et al (Kelsey et al., 2003, Kelsey and Timmis, 2003) which is examined and modelled in chapter 3. We also examine the

Villalobos-Arias model (Villalobos-Arias et al., 2004) of MISA (Coello and Cortes, 2002) here.

The key element of a clonal selection algorithm is the clonal selection operator. The clonal selection operator can be thought of in an algorithmic sense as having three stages (De Castro and Von Zuben, 2000):

- Cloning – where copies of the original are made
- Mutation – where variation is introduced
- Selection – where inferior clones are discarded

It has been proposed that the clonal selection theory and its associated operator can be interpreted as a microcosm of Darwin's law of evolution, with the major principles of diversity, variation and natural selection (Cziko, 1995). The central processes involved in the production of antibodies, genetic recombination and mutation, are the same ones responsible for the biological evolution of sexually reproducing species (Holland, 1995). These assertions that clonal selection algorithms are essentially the same as evolutionary algorithms are evidenced by work undertaken in this thesis. We construct a model of a clonal selection algorithm, Algorithm 4 in chapter 3 and a model of the $(1+\lambda)$ evolutionary strategy, Algorithm 7 in chapter 6. Comparison of these models demonstrates that the differences between the algorithms are indeed semantic, the mathematical difference between the two algorithms being only the choice of mutation operator used.

2.4.1 Markov chain models of clonal selection algorithms

An overview of Markov chains for clonal selection is given in (Timmis et al., 2008c) giving the key references for the area. A model of MISA and proof of convergence is presented by Villalobos-Aris et al (Villalobos-Arias et al., 2004) and is discussed here. The authors make an incomplete (disregarding the clonal stage), but exact model of the clonal selection based, Multi-

objective Artificial Immune System Algorithm, MISA . They explicitly give the transition probabilities for a non-elitist version of MISA. The authors also show, via a Markov chain construction, that the elitist version of MISA is convergent on all problem spaces, in infinite time. However the authors unnecessarily restrict the range of the two mutation rates to $0 < p_m, \rho_m \leq \frac{1}{2}$. It can be deduced by considering the generalised proof of convergence given in section 3.5.1 that the proof will in fact continue to hold over the region $0 < p_m, \rho_m < 1$. It would appear the upper bound that the authors have given is determined by “conventional wisdom” about mutation rates, instead of being derived from the mathematical constraints imposed by the analysis of the model.

The model of MISA (Villalobos-Arias et al., 2004) is the first such instance of a proof of convergence of an artificial immune system optimisation algorithm. An explicit equation for the mutation probabilities of the elitist MISA is not given, but the proof of convergence can be made convincingly without the explicit probabilities.

Results presented in chapter 3 and published (Clark et al., 2005) demonstrate a novel Markov chain construction method for modelling elitist algorithms exactly and can confirm the results for MISA. Application of this novel modelling methodology to MISA would give the transition matrix for the elitist version of MISA explicitly. The methodology presented here, could go on to extend the model of MISA to include the clonal selection stage, making the model both complete and exact.

Cutello et al (Cutello et al., 2007) present two conditions, that if satisfied by a general Immune Algorithm (IA) which they define, prove its convergence. The proof was adapted from a similar approach on qualitative models of EAs presented by Rudolph (Rudolph, 1998) who adopted the approach (Eiben et al., 1991). Conditions directly equivalent to those obtained (Cutello et al., 2007) are obtained independently in chapter 3, based directly on the proof of convergence of the BCA first published in (Clark et al., 2005) where the

conditions of the proof as published are already broad enough to be considered a general proof, although no claims of generality were made in (Clark et al., 2005).

Function independent upper bounds on the rate of convergence are also obtained by (Cutello et al., 2007). The authors acknowledge that the bounds obtained are “not of practical interest” and suggest that the bounds obtained for particular problem classes may be practical, in line with the view taken in the EA community (Rudolph, 1998).

Cutello et al also suggest that “The examination of simple AIS inspired operators could give better insight of how IAs actually work” (Cutello et al., 2007), putting forward the view that more operator based investigation may be more informative, in the context of the current literature where algorithms have been shown to converge to global optimum.

2.5 Discussion of the literature

We have described several groups of optimisation algorithms: Evolutionary algorithms (ES and GA), Simulated annealing and artificial immune systems and presented their associated Markov chain modelling literature. We now look at themes that exist in all Markov chain models of optimisation and discuss how these issues can be used to guide the research undertaken in this thesis. The two critical themes identified for guiding novel research are:

- How the “state of an algorithm” is defined across different models and why different models have chosen their definition of a state of the algorithm.
- The inherent modularity of optimisation algorithms and to what extent this modularity is exploited for construction of Markov chain models.

2.5.1 Defining the state of an algorithm

In section 1.2 we raised the issues associated with defining the state of an optimisation algorithm and quoted a concise example of a definition of a state of an algorithm (Rudolph, 1994a) from a Markov chain model. We now consider a contrasting method of defining a state of an algorithm from the No Free Lunch (NFL) theorem (Wolpert and Macready, 1997). The NFL theorem for optimisation states:

“The no free lunch theorem for search and optimization applies to finite spaces and algorithms that *do not* resample points. All algorithms that search for an extremum of a cost function perform exactly the same when averaged over all possible cost functions. So, for any search/optimization algorithm, any elevated performance over one class of problems is exactly paid for in performance over another class.” (Wolpert and Macready, 1997)

In this thesis, we exclusively consider algorithms that *can* resample points (to resample a point is to evaluate the objective function at a point that has already been evaluated.) and as such, the results of NFL theorem are of peripheral importance. However, NFL theorem requires a same level of formalism to define the state of an optimisation algorithm as the formalisms required by a Markov chain model. We describe the formalism used by Schumacher (Schumacher et al., 2001). The authors define a “trace” T_m of length m , to be a sequence of pairs, points in a search space and their associated fitness, as determined by some objective function.

$$T_m \equiv \langle (x_1, y_1), (x_2, y_2), \dots, (x_m, y_m) \rangle. \quad (2.6)$$

(Schumacher et al., 2001)

The pairs recorded in the trace are essentially the same as the input-output pairs discussed in section 1.2. A trace as used by (Schumacher et al., 2001) is a time-ordered record of points in the search space that the algorithm has visited. The authors go on to consider a search operator which given a

trace T_m , returns the next point in the search space to be evaluated and thus generates T_{m+1} . Hence the state of the algorithm at any given time is explicitly defined by the trace at that time.

In the formalism used in (Schumacher et al., 2001) we can see the implicit segregation of the algorithms state from the search operator (we discuss this distinction in section 1.5). The authors do not constrain how much of the information contained in the trace, if any, is utilised by the search operator and thus do not constrain the form of the search operator. We note also that with the information in the trace at time T_m it is possible to mimic potentially any state operator.

Now let us consider if the definition of the algorithm described in (Schumacher et al., 2001) satisfies the Markov property described mathematically by equation (2.1). The question is more subtle than it appears; as we have stated above, the trace is a time-ordered list of points in the search space. In order to satisfy the Markov property, the probability of the algorithm being in any given *state* at the next time step depends only on the *state* of the system at the current time step and is independent of the previous *states* of the system. Using the trace to define the state of the algorithm can be said to satisfy the Markov property. Talking in general terms, this follows from the reasoning given in section 2.2.3 that finite state machines satisfy the Markov property, therefore all computer algorithms satisfy the Markov property for some meaning of the word “*state*”. Specifically, it satisfies the Markov property because T_{m+1} depends only on T_m (and the search operator), it is independent of $T_{m-1}, T_{m-2}, \dots, T_1$. In this case T_m is only independent from T_{m-1} because the information in T_{m-1} is a subset of T_m ; so T_{m-1} can provide no additional information to the search operator.

The definition of the state of an algorithm used by (Schumacher et al., 2001) differs significantly from the more traditional state of an algorithm used in examples of Markov chain models of algorithms which either use a

single position in the search space (Villalobos-Arias et al., 2004, Häggström, 2002, Clark et al., 2005), or the current position of a population of points in the search space (Nix and Vose, 1992) as a definition of the state of an algorithm. The trace used by (Schumacher et al., 2001) is sufficiently general that it could be used to define the algorithms in all aforementioned models, so why is it that Markov chain models do not make use of such a definition of state and why do some Markov chain models use a different definition of state to others?

Equation (2.5) shows the number of states of the algorithm for the (Nix and Vose, 1992) model of a simple GA; the authors comment that it is far larger than a minimal state space. It is however the smallest possible state space for the algorithm that has been modelled; we attribute this increase in state space to the crossover operator. It should be noted that when the population size is set to 1, the number of states in the Nix and Vose model collapses to the minimal state space.

Despite the non-minimal state space of the (Nix and Vose, 1992) model, some useful numerical work was presented by (De Jong et al., 1995) although only for toy problems. Given the increase in computational power since 1995 it is quite plausible that some non-toy problems could be investigated with more numerical Markov chain work, even for the state space of the Nix and Vose model. For an algorithm that is modelled with a minimal state space, the chance for numerical investigation of the model for moderate size search spaces ($\sim 2^{16}$ points) of unlimited complexity is a valuable prospect; one that should not be discarded lightly. If we were to adopt the definition of an algorithm used by (Schumacher et al., 2001) the combinatorial explosion of the state space would make numerical investigation intractable for all but trivial search spaces. It would also complicate formulation of the Markov chain for all algorithms that do not actually use the entire trace to determine the next position in the search space to evaluate.

2.5.2 Motivation for a new approach to Markov chain modelling

Let us consider the Nix and Vose Markov chain model of a simple genetic algorithm (Nix and Vose, 1992) as an example modelling methodology of the traditional Markov chain approach and look at the limitations of this approach. The results obtained from this model (Vose, 1995) link the long term behavioural properties of the Markov chain to the *algorithm as a whole*. In section 2.3.1 we noted that the limiting values of the mutation rate would cause a division by zero and were therefore excluded. This was noted again in section 2.3.3 where the numerical work by (De Jong et al., 1995) was attempting to probe the properties of the model. In this numerical investigation optimal mutation rates were explored as well as an investigation into how crossover and mutation interact with each other. Due to the limitation of the model it is not possible to have a mutation rate of 0 and look at the effects of crossover in the absence of mutation. We see in the numerical work the first concerted attempt to move away from properties of the algorithm (of the model) as a whole and towards connecting properties to specific operators.

We have seen that the operators in the Nix and Vose model are not entirely separable, which makes the model hard to break down and analyse on an operator level. It also makes it cumbersome (although not impossible) to alter the model to contain variations of operators.

Nix and Vose implicitly identify that operators can be split into two categories: "the basic mechanism of a GA consists of:

- Evaluation of individual fitness and selection of a gene pool
- Mutation and crossover"

(Nix and Vose, 1992)

There is a tacit notion in GAs that there are search operators and other operators: "In GAs the emphasis traditionally lies on the search operators:

mutation and recombination (crossover)” (Eiben and Rudolph, 1999) the selection operator is excluded, indicating the tacitly assumed distinction. The potential advantages of taking this tacit knowledge and making it explicit are yet to be explored. Using the terminology set out in chapter 1, both mutation and crossover are search operators while selection of a gene pool is a state operator.

We propose that there are potential advantages to maintaining the modular nature of the algorithms in the models of the algorithms. It is a modelling guideline that the mathematical representation of operators in the model of an algorithm should be separable, to reflect that they are separable in the algorithm. Even in the case of two search operators applied consecutively, models should keep the models of the operators separate to make the resulting models easier to analyse; allowing properties of the algorithm as a whole to be linked to a specific operator or combination of operators.

Examination of operators and how they combine to make optimisation algorithms “actually work” was called for in (Cutello et al., 2007). This brings the point that it is not simply a matter of modelling the operators in isolation that is important (although that would no doubt give some insight), but modelling how the operators interact with each other in order to give the overall properties of the algorithm. We propose that the way to achieve this is to make models of algorithms in which the operators remain fully separable and interchangeable as they are in the algorithm, allowing insight into not just the operators, but how the operators interact to give the algorithm its properties.

2.6 Summary

We have presented a variety of nature inspired function optimisation methods: Genetic Algorithms, Evolutionary Strategies, Simulated Annealing and Artificial Immune systems, detailing their origins and broadly describing their characteristic operators. We have focused our review of these areas on

their theoretical analysis, with specific attention to existing Markov chain models in each area.

We have identified the potential advantages of an explicitly modular modelling approach, where operators are separable and easily substituted for comparable operators with different characteristics. A modular system would allow properties of an algorithm to be clearly linked to a particular operator, as well as allowing investigation into how operators interact. With a modelling framework that provides for these more flexible models, it would even be possible to have theoretical work propose and test novel adaptations to existing algorithms from a firm theoretical basis.

We propose the way to derive such a framework is to perform a case study on an existing algorithm, but instead of constructing a single model, construct multiple models: The initial model being a highly simplified version of the algorithm, where a "simplification" is the removal of an operator. Subsequent models should be made of versions of the algorithm with one additional operator restored in each modelling cycle, until a complete and exact model of the existing algorithm is achieved.

We have also identified the importance of maintaining a minimal definition of the "state of an algorithm"; meaning that for each algorithm we should determine the minimum definition required. There are theoretical and practical motivations for maintaining minimal state spaces: from a theoretical perspective a non-minimal definition would add unnecessary complication to the formulation of a model and from a practical perspective it is important to maintain a minimal state space in order to not preclude the possibility of numerical work based on Markov chain models.

Based on the case study of modular modelling of an optimisation algorithm we develop a detailed theoretical understanding of this algorithm (chapters 3 and 4), but more importantly, we provide the basis for a framework that will allow modular models to be constructed (chapter 5). This framework will greatly aid the construction of novel Markov chain models of existing

algorithms. The framework will also aid conversion of existing non-modular Markov chain models into modular models (chapter 6). Due to the models' modular nature, properties of the models of the algorithms will be associated with the effects of individual operators.

Chapter 3 The B Cell Algorithm

This chapter provides the case study for the thesis hypothesis (stated in section 1), laying the foundation for the modelling framework. We shall also undertake both a theoretical and numerical analysis of one of the models proposed in this chapter, demonstrating that the models produced can be used to further the understanding of optimisation algorithms. The theoretical work takes the form of a proof of convergence, presented in section 3.5 and a brief numerical analysis presented in section 3.6. The model constructed in section 3.4 is further investigated in chapter 4 where theoretical bounds are obtained and a comparison with bounds found in the literature is undertaken.

In order to aid the development of theoretical tools, a case study of the B Cell Algorithm (BCA) (Kelsey and Timmis, 2003) will be undertaken. The aim of the study is to produce a complete and exact model of the B cell Algorithm and use this model to determine the properties of the algorithm. As the tools developed in this case study should be flexible and adapt easily to algorithms that perform the same function, the model will be purposefully constructed in a highly modular manner, starting with a simple base model and subsequently adding features such as elitism and the clonal pool. By taking the time to explore the case study algorithm thoroughly, investigating any potentially interesting properties as they arise from the modelling process, a greater understanding of the modelling methodology is developed. The tools developed in this chapter will provide the basis for the general framework for modelling stochastic optimisation algorithms in chapter 5, which will allow the development of additional models in chapter 6, without the need to start from scratch on each new algorithm.

The chapter looks at the BCA pseudo code and algorithm description in its original form (Kelsey and Timmis, 2003, Kelsey et al., 2003) in section 3.1; then the modelling process begins, starting *from first principles*. The

concept of a mutation mask is described in section 3.2 and then used to construct a highly simplified model of the BCA algorithm in section 3.3, based entirely on the model of the hypermutation operator and mutation masks. In section 3.4 a second model is constructed that explicitly includes elitism. The elitist model is analysed theoretically and a proof of convergence is presented in section 3.5. In section 3.6 we present a numerical study of the effect of the mutation rate on the rate of convergence of the elitist algorithm modelled in section 3.4. A third model of the BCA is constructed in 3.7 including the clonal pool.

3.1 The B Cell Algorithm

In this section, we present the B-Cell Algorithm (BCA) as originally published (Kelsey and Timmis, 2003, Kelsey et al., 2003). In sections 3.3, 3.4 and 3.7 we present pseudo code of various simplified versions of this algorithm. We have denominated the algorithms numerically, so that the algorithms' pseudo code can be referred to without potential for ambiguity.

Algorithm 1: The B-Cell Algorithm adapted from (Kelsey and Timmis, 2003)

Step 1: Initialise a population of P binary strings x each
 of length l . Each element in the string has a 0.5
 probability of having the value 1

Step 2: evaluate $g(x)$ for all $x \in P$

Step 3: While stopping criterion not met:

Step 4: for all $x \in P$

Step 5: Create C clones by making C copies of x and mutating them using the contiguous region hypermutation operator

Step 6: Evaluate $g(x_c)$ each of the C clones

Step 7: x' is defined as the clone with the greatest $g(x_c)$. In the case where k clones have the same highest $g(x_c)$ value, each of the k clones has a probability of k^{-1} of being selected to become x'

Step 8: If $g(x') > g(x)$ then replace x by x'

Step 9: end for

Step 10: Return to step 3

The BCA in (Kelsey and Timmis, 2003) is minimising, but could just as easily be maximising. We refer to it as minimising/maximising where appropriate for a given example and optimising in general.

An important feature of the BCA is its use of a novel mutation operator, known as contiguous somatic hypermutation, also known as the Contiguous Region Hypermutation Operator (CRHO). Evidence of an analogous biological mechanism in the immunological literature is sparse (Rosin-Arbesfeld et al., 2000), however, the authors argue that mutation occurs in clusters of regions within cells: this is broadly analogous to contiguous regions. The representation employed in the actual implementation of the BCA in (Kelsey and Timmis, 2003) takes an N -dimensional vector of bit strings of length l . These vectors are considered to be the B-cells within the system. Each B-cell within the population is evaluated by the objective

function, $g(x)$. More formally, the B-cells are defined as a vector $x \in P$ of bit strings of length l where P is the population of vectors x (B-cells).

Empirical work presented in (Kelsey and Timmis, 2003, Kelsey et al., 2003, Hone and Kelsey, 2004) indicates that an efficient population size for many functions is low, a typical size would be $|P| = [3..5]$. For the purposes of our work, in order to simplify the complexity of the mathematical model of the BCA, we consider a population size of 1. This is not a limiting simplification because population members do not interact, thus allowing us to calculate probabilities independently for each member of the population.

After evaluation by the objective function, a B-cell is cloned to produce a clonal pool, C . It should be noted that there exists a clonal pool C for each B-cell within the population and also that all the adaptation (mutation) takes place within C . The size of C is typically the same size as the population P (but this does not have to be the case). In order to maintain diversity within the search, one clone is selected at random and each element in the vector is randomized. Each B-cell clone $x' \in C$ is then subjected to a novel contiguous somatic hypermutation mechanism (described below). The BCA typically uses a distance function as its stopping criterion: when it is within a certain prescribed distance from the optimum, the authors consider the algorithm to have converged.

The unusual feature of the BCA is the form of the mutation operator. This operates by subjecting bits contained within a contiguous region of x' to mutation with probability r such that $0 \leq r \leq 1$. As shown in Figure 3, the CRHO selects a random site (or hotspot) within x' , a length L is also determined randomly and may be up to the full length of the vector. The vector is then subjected to mutation from the hotspot onwards for the length of the contiguous region.

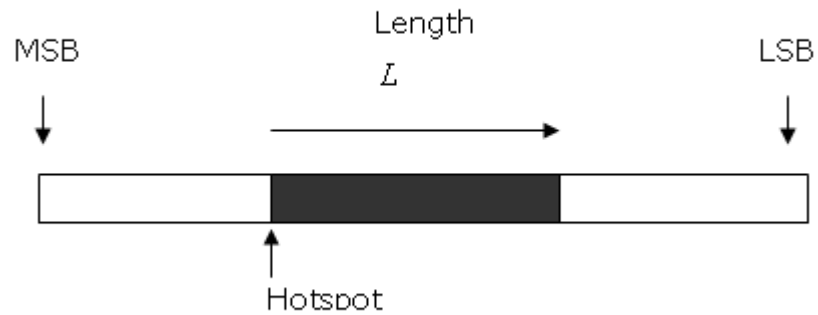


Figure 3. The Contiguous Region Hypermutation Operator. Figure adapted from (Clark et al., 2005)

It is important to note that the mutation region shown in Figure 3 does not wrap around the vector in the case where the hotspot plus the length (of the mutation region) L , exceeds the length of the binary string l . It is also important to note that the direction contiguous region extends from the hotspot: towards the Least Significant Bit (LSB). The direction, combined with the fact that the region does not wrap around causes a positional bias making small mutations more probable than large mutations as can be seen. Accounting for these properties makes modelling the CRHO far more cumbersome than might be expected.

3.1.1 The BCA from a modelling perspective

The first point of interest to be noted from the definition of the algorithm, is that members of the population do not interact in any way; members of the population are *independent*. That is to say that the probability of a given member of the population moving from its current position in the search space to its next position in the search space, is unaffected by the position (past, present or future) of any other member of the population. This means we can focus our efforts on modelling a simplified version of the algorithm

with a population of 1 and generalise results to a population of $|P|$ using basic probability theory, see (Häggström, 2002) chapter 1.

Conversely, the members of a clonal pool are not independent; the clonal pool must be modelled explicitly. The mutation of each $x' \in C$ are independent, the interaction within the clonal pool is in the form of ordering and occurs at the clonal selection stage. Only the clone with the highest affinity will be selected to be included in the next generation. Hence there is an ordering within the clonal pool; the clonal selection process requires knowledge of all clones in order to determine which has the highest affinity. The algorithm can be simplified by setting the clonal pool size to be 1 for the purposes of the preliminary investigations. For a complete model of the BCA the clonal pool of an arbitrary size must be modelled explicitly, see section 3.7.

3.2 Mutation masks

The first aspect of the BCA that we will model is the mutation operator. Modelling the mutation operator will provide us with the probabilities for the mutation of any binary input string to any binary output string. Because the mutation operator is a binary based mutation operator it makes sense to consider the probability of generating mutation masks instead of explicit position-to-position mutations, as there are only 2^l mutation masks but 2^{2^l} position-to-position mutations.

The crux of the mutation masks concept is the \oplus (the exclusive or operator, "xor") operator for binary strings. We denote the mutation mask T , the input string j and the output state k . These are related by

$$j \oplus T = k \tag{3.1}$$

Where the input string, output string and mutation mask are all binary strings of length l .

A key point to observe is that these three strings form a triple, where if you take any two strings and perform an \oplus operation you get the third string. We shall use this simple fact in the way in which we approach the modelling of the mutation operator.

Consider: For a given fixed input string and a complete set of output strings we must use a complete set of mutation masks. So, we can choose any input string without loss of generality, as long as we model the transitions to all possible output strings. The obvious choice is to fix the input string as zero, for conceptual simplicity. With an input string of zero, the mutation mask and output strings are the same.

Now that we have introduced the concept of a mutation mask we can progress to constructing a model of Algorithm 2, a highly simplified version of the BCA, in section 3.3. We will make use of the concept of mutation masks when constructing the model of the CRHO in section 3.3.1.

3.3 BCA with clonal pool of 1, population of 1 and no elitism

This section will construct an exact mathematical model of a highly simplified version of the B-Cell Algorithm. This simplified algorithm is defined here:

Algorithm 2: The B-Cell algorithm with a clonal pool of 1, population of 1 and no elitism

Step 1: Initialise a binary string x of length l . Each element in the string has a 0.5 probability of having the value 1

- Step 2: evaluate $g(x)$
- Step 3: While stopping criterion not met:
- Step 4: Create x_c , a clone of x mutated using the
contiguous region hypermutation operator
- Step 5: Evaluate $g(x_c)$
- Step 7: replace x by x_c
- Step 8: Return to step 3

The objective $g(x)$ is defined as a search space with 2^l input output pairs, where $l \in \mathbb{Z}^+$. All input values x , can be represented by the configurations of a binary string of length l and all output values, $g(x)$, are numerical values that occur on a floating point representation.

3.3.1 The Contiguous Region Hypermutation Operator (CRHO)

Let us assume that the initial vector of length l is a binary vector containing only zero elements. This arbitrary starting condition can be applied without loss of generality provided we consider an exhaustive set of mutation masks.

Consider the probability that the vector will be mutated by the operator to some given number T , or "target vector". We define that the target vector contains one or more non-zero elements (the special case of a target vector of all zero elements $T=0$ will be formulated separately). Starting from an initial vector of all zero elements, it is clear that the target vector is the same as the mutation mask required to reach the target vector.

The probability of generating a given mutation mask by the CRHO is the product of three independent probabilities:

- The probability that the non-zero element(s) of the target vector are contained within the contiguous region.
- The probability that the target element(s) are mutated, given that they are in the region. The probability of an individual element being mutated given that it is in the region of mutation is denoted r .
- The probability that any "incidental" elements contained within the contiguous region are not mutated. The probability of an individual element not being mutated given that it is in the region is therefore denoted $1-r$.

To find the general formula to generate these probability values, we consider the specific cases for $l=1,2,3,4$ and write out the algebraic transition probabilities in section 3.3.3 using the above proposition on how the probabilities can be calculated. We must consider an exhaustive set of ways each transition can occur. The above proposition notably does not cover a null transition. In a null transition, all the elements in the contiguous region are incidental, so the proposition for a null transition is merely a sum over all possible regions, where the probability of each region occurring is multiplied by the probability that all the elements contained within the region are not mutated.

As we will be looking at an exhaustive set of algebraic transition probabilities in section 3.3.3 we can expect that the sum over all probabilities (for a given value of l) will be equal to one algebraically. It is good practice to check that this is the case, as it guards against incorrect methodology.

The probability of any particular hotspot being selected is $\frac{1}{l}$ similarly the probability of any particular region length L being selected is also $\frac{1}{l}$, therefore the probability of any given regions occurring is $\frac{1}{l^2}$. The hotspot plus the region length may exceed the end of the vector, this property will, of course, be taken into account. We must be precise in our specification of the hotspot and length of the contiguous region, there are in fact multiple equally valid ways the regions can be defined and counted.

3.3.2 Counting states in the Contiguous Region Hypermutation Operator (CRHO)

In order to represent the mutation operator correctly we must consider all the possible regions and how many ways each region can be formed. Some regions can be formed in more than one way, due to the contiguous region not wrapping around the string. The minimum size of a region formed by the mutation operator is one element. Let us define the convention that when a hot spot is selected, it has an intrinsic length of one. Hence the minimum length that can be added to this hot spot to define the region should be zero, not one (the reverse of this convention could be applied and would yield identical results). Let us define that:

The start point of a hot spot, a , can be any integer from 1 to l

$$a \in [1, \dots, l] \quad (3.2)$$

The physical endpoint, b , is specified from 1 to l

$$b \in [1, \dots, l] \quad (3.3)$$

The additional length, c , is specified from 0 to $l-1$

$$c \in [0, \dots, l-1] \quad (3.4)$$

In the case where $a+c \geq l \Rightarrow b=l$ and in the case $a+c < l \Rightarrow b=a+c$.

The total number of bits to be flipped, defined as k , is a positive integer.

$$a \leq k \leq b+1-a \quad (3.5)$$

Let us consider the number of distinct regions that can be defined and the number of ways each of these regions can occur. There are l possible hot spot start points and l possible lengths that can be added to any given hot spot, therefore there are l^2 possible ways of defining a region. Although some of these physical regions are degenerate; they can be defined by more than one hotspot - region length pair.

3.3.3 Contiguous Region Hypermutation Operator for a bit string of length one

Let f_T be the probability that the starting vector will be mutated to a target vector T .

In the trivial case of $l=1$, the complexities of counting states are not apparent, the equations can be formed trivially.

For $T=0$, mutating from an initial string of 0, to a target string of 0 we have $f_0 \Rightarrow a=1, c=0 \Rightarrow b=1$ and no bits within the region are to be flipped $k=0$

$$f_0 = \frac{1}{l^2}(1-r) \quad (3.6)$$

For $T=1$, mutating from an initial string of 0, to a target string of 1 we have $f_1 \Rightarrow a=1, c=0 \Rightarrow b=1$ and one bit within the region needs to be flipped $k=1$

$$f_1 = \frac{1}{l^2} r \quad (3.7)$$

Trivially, the sum of the probabilities can be shown to be one

$$\sum_{T=0}^{2^l-1} f_T = f_0 + f_1 = \frac{1}{l^2} = 1 \quad (3.8)$$

3.3.4 Contiguous Region Hypermutation Operator for a bit string of length two

Now let us consider the first non-trivial case, $l=2$. The null transition $T=0$ contains all possible regions. Let us take the convention that we start counting states by selecting the left most bit, then all the possible region lengths, before moving on to the second left most bit and so on.

There are 3 distinct physical regions. In general the number of physical regions, R_{phys} , is given by the formula

$$R_{phys} = \sum_1^l w \quad (3.9)$$

However, the total number of ways of defining a region, R_{def} , is given by the formula

$$R_{def} = l^2 \quad (3.10)$$

Hence for $l=2$ there are only 3 distinct physical regions as defined in equation (3.9), but there are 4 possible ways to define a region as both a and c have two possible values $a \in [1,2], c \in [0,1]$.

The regions where the hot spot plus the additional length can be greater than the total length of the vector l , are degenerate (as the same physical region can be defined multiple ways). The order of counting regions is defined by the bit position. The first position in the bit string is defined to be 1. The regions are defined as $1 \rightarrow 1, 1 \rightarrow 2, 2 \rightarrow 2, 2 \rightarrow 3$

f_0 , starting vector $[0,0]$, Target vector $[0,0]$

In this case there are only two bits, however, when the "hotspot" selected is "2", there are two possible length regions to consider both of which are actually the same physical regions, as the region does not "wrap around" the string. The position "3" on the string (of length 2) represents an imaginary end point to the region. The physical region represented by $2 \rightarrow 3$ is physically the same as the region $2 \rightarrow 2$.

$$f_0 = \frac{1}{l^2} [(1-r) + (1-r)^2 + (1-r) + (1-r)] = \frac{1}{l^2} [(1-r)^2 + 3(1-r)] \quad (3.11)$$

f_1 , starting vector $[0,0]$, Target vector $[0,1]$

The region $1 \rightarrow 1$ will have a zero contribution in this case, so there are 3 (defined) regions to be considered (but only 2 physical regions).

$$f_1 = \frac{1}{l^2} [(1-r)r + r + r] = \frac{1}{l^2} [3r - r^2] \quad (3.12)$$

f_2 , starting vector $[0,0]$, Target vector $[1,0]$

From the target vector, it is clear that strings starting from "2" will have a zero contribution to the probability of transition between the initial vector [0,0] and the target vector [1,0], so only the first two regions give a non-zero contribution.

$$f_2 = \frac{1}{l^2} [r + r(1-r)] = \frac{1}{l^2} [2r - r^2] \quad (3.13)$$

f_3 , starting vector [0,0], Target vector [1,1]

There is only one region that can contribute: $1 \rightarrow 2$, the other 3 regions do not contain both of the non-zero elements in the target vector.

$$f_3 = \frac{1}{l^2} [r^2] \quad (3.14)$$

This completes the set of transitions for $l=2$, we can see that the sum of the terms is equal to one:

$$\begin{aligned} \sum_{T=0}^{2^l-1} f_T &= f_0 + f_1 + f_2 + f_3 \\ &= \frac{1}{l^2} \left([1 - 2r + r^2 + 3(1-r)] + [2r - r^2] + [3r - r^2] + [r^2] \right) \\ &= \frac{1}{l^2} \left([1+3] + [-2-3+2+3]r + [1-1-1+1]r^2 \right) \\ &= \frac{4}{2^2} = 1 \end{aligned} \quad (3.15)$$

This process has been repeated by hand for $l=3,4$ but for brevity these results are not presented here. The process of counting states and determining probabilities is a very mechanical process, it becomes apparent that the quantities needed for the calculation are the position of the first "on" bit in the target vector, the last "on" bit in the target vector and the total number of bits to be flipped. Notably the position of bits between the first and last on bit is not important, this is the case because these bits will always be included in any region that will contain both the first and last "on"

bits in the target vector. The pattern that emerges from studying the cases for $l=1,2,3,4$ is sufficient to determine the general formula for all non-zero transitions. For the special case of a zero transition all regions contribute a non-zero probability of the transition occurring. The formula is a sum over all regions of $(1-r)$ raised to the power of the number of elements in each region.

Extrapolating from the examples, it is possible to obtain general formulae to calculate the probabilities of all possible mutation masks.

$$f_T = \frac{1}{l^2} \left[\sum_{n=1}^a \sum_{m=b}^{l-1} (1-r)^{m+1-n-k} r^k + \sum_{n=1}^a n (1-r)^{l+1-n-k} r^k \right] \quad (3.16)$$

Where: f_T is the probability of transition from zero to some number T ; l is the length of the binary string; a is the position of the first "on" bit counting from the most significant bit; b is the position of the last "on" bit counting from the most significant bit; k is the total number of bits that must be flipped to mutate from zero to T . The probability of a bit being mutated, given it is within the contiguous region is r . Some basic properties of the terms given in equation (3.16) are defined to be

$$\{l, a, b, k, T\} \in \mathbb{Z}^+; 0 < T \leq 2^l - 1; a \leq b \leq l; \{f_T, r\} \in \mathbb{R} 0 \leq r, f_T \leq 1. \quad (3.17)$$

Equation (3.16) can generate all the probabilities of all non-zero mutation masks. The zero mutation mask is also required and is given by

$$f_0 = \frac{1}{l^2} \left[\sum_{n=1}^l \sum_{m=n}^{l-1} (1-r)^{m+1-n} + \sum_{n=1}^l n (1-r)^{l+1-n} \right] \quad (3.18)$$

With this formulation only the probability of the mutation mask occurring is generated, so the formula is general (for a clonal pool of one). Modelling the

mutation masks instead of state-to-state probabilities allows these results to be applied to standard binary, Gray code or any other binary encoding.

The mutation operator is significantly more complicated than might have been anticipated, it is worth testing that the derivation of these mutation mask probabilities agrees with experimental data obtained from an implementation of the mutation operator. After formulation of equations (3.16) and (3.18) the CRHO was implemented to compare theory, Figure 4 A, with experiment Figure 4 B.

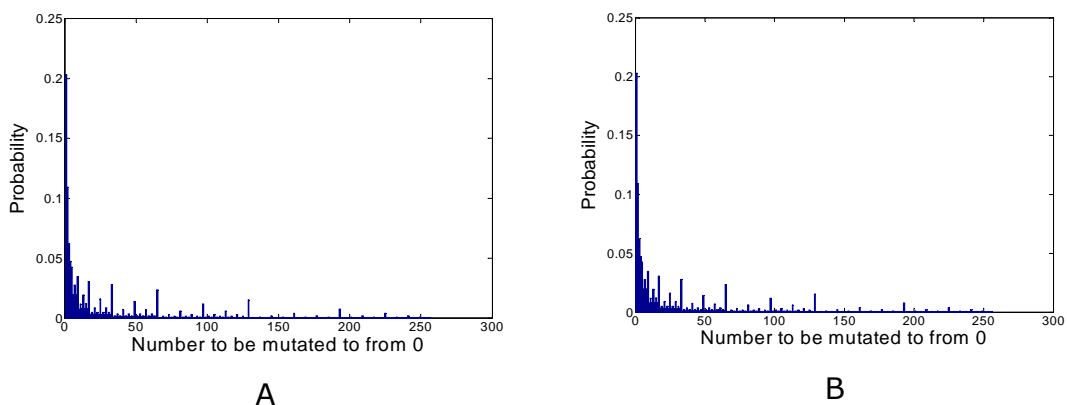


Figure 4. The probability distribution of the Contiguous Region Hypermutation Operator: String Length $l = 8$, Probability of mutation $r = 0.5$. The theoretically determined probability distribution is shown in A. The probability distribution in B was determined experimentally by averaging one million results from the implemented CRHO. Figure adapted from (Clark et al., 2005).

The positional bias of the CRHO, which we alluded to when discussing Figure 3 in section 3.1, is plain to see in Figure 4. The probability distribution is far from uniform; there is far greater chance of generating a small mutation than a large mutation. The spikes that appear in the distribution Figure 4 occur when the mutation from 0 is obtained by flipping a single bit.

In this section we have derived equations (3.16) and (3.18) which describe the probability of generating the mutation mask required to mutate from 0

to some target T . In section 3.3.5 we will demonstrate how to use mutation masks to extend these results so that all state-to-state transitions are given explicitly.

3.3.5 From mutation masks to state tables

We have demonstrated in section 3.3.4 that we can calculate the probability of sampling any state, given that the starting state is zero, by calculating the probability of the required mutation mask. So in order to calculate the probability of sampling any state, given any starting state, all we need know is the mutation mask between these states.

	00	01	10	11
00	00	01	10	11

Table 1. Mutation masks from state 00 to all states.

The edges of Table 1 represent, in binary, the starting state and end states, with the body of the table showing the mutation mask (exclusive or of the corresponding states) required to move from one state to another. Replacing the mutation mask with the probability that the mutation mask will be generated we obtain Table 2.

	00	01	10	11
00	f_0	f_1	f_2	f_3

Table 2. Probabilities of mutation masks occurring from state 00 to all states.

The edges of the Table 2 represent, in binary, the starting state on the row and end states as columns, with the body of the table showing the probability of the required mutation mask occurring.

As the formulation of the probabilities involved is for the mutation mask and is not limited by the starting state, we can calculate the probability of moving to any state from any given initial state by determining which mutation mask is needed. If we consider starting in state one instead of state zero we can calculate the required mutation masks to sample any end state, this is shown in Table 3.

	00	01	10	11
01	01	00	11	10

Table 3. Mutation masks from state 01 to all states.

As we can calculate a full set of mutation masks, we need only substitute the mutation masks shown in the body of Table 3 with the relevant probabilities of each of those mutation masks occurring; as shown in Table 4.

	00	01	10	11
01	f_1	f_0	f_3	f_2

Table 4. Probabilities of mutation masks occurring from 01 to all states.

We can repeat this process for all starting states and construct a state table showing the mutation masks from all states to all states, as shown in Table 5.

	00	01	10	11
00	00	01	10	11
01	01	00	11	10
10	10	11	00	01
11	11	10	01	00

Table 5. Mutation masks from state all states to all states.

Substituting the mutation masks in Table 5 with the probability that the mutation mask will be generated by the mutation operator we obtain Table 6

	00	01	10	11
00	f_0	f_1	f_2	f_3
01	f_1	f_0	f_3	f_2
10	f_2	f_3	f_0	f_1
11	f_3	f_2	f_1	f_0

Table 6. Probabilities of mutation masks occurring from all states to all states.

Table 6 shows the mutation mask required to mutate from any state to any other state. Equations (3.16) and (3.18) can, for a given value of the mutation rate r , calculate the probability of each of these mutation masks occurring via the mutation CRHO. This describes the probability of mutating from any state to any other state in a single "step" (iteration/application of the mutation operator).

3.3.6 Choosing to use the Markov chain representation

Working with this highly simplified version of the BCA, described in Algorithm 2 in section 3.1, we have been able to determine the probabilities of mutation from any one state to any other state in one step, shown in Table 6. In this section we consider how to obtain the two-step and n step probabilities. To get the two-step probability of moving from state 00 to state 01, we would need to sum the probabilities of each of the four possible routes between the two states:

- Remaining in state 00 in the first step and then moving to state 01 in the second step $f_0 f_1$.

- Moving to state 01 in the first step and then remaining in state 01 in the second step f_1f_0 .
- Moving to state 10 in the first step and then moving to state 01 in the second step f_2f_3 .
- Moving to state 11 in the first step and then moving to state 01 in the second step f_3f_2 .

Therefore the probability of being in state 01 after two steps, given that the starting state was 00 is given by $f_0f_1 + f_1f_0 + f_2f_3 + f_3f_2$. We can repeat this process for any initial state moving to any end state in two steps. If we take the body of Table 6 and define it as a matrix

$$\mathbf{M}_1 = \begin{pmatrix} f_0 & f_1 & f_2 & f_3 \\ f_1 & f_0 & f_3 & f_2 \\ f_2 & f_3 & f_0 & f_1 \\ f_3 & f_2 & f_1 & f_0 \end{pmatrix} \quad (3.19)$$

We can tackle the problem of moving from state 00 to state 01 in two steps in a simple and compact way. It is the multiplication of the first row on the second column, which gives $f_0f_1 + f_1f_0 + f_2f_3 + f_3f_2$ as above.

We find that all these summations of the probabilities of moving from any one state to any other state in two steps can be expressed simply by raising the one step transition matrix to the power two: \mathbf{M}_1^2 .

A matrix notation is not only an efficient method of storing the probabilities for one step, state to state moves; it also allows for the n step probabilities to be calculated simply by raising the matrix to the power n . This is by no means a novel discovery, this is standard in Markov chains, see (Häggström, 2002) chapter 2. The Markov chain representation provides all the functionality required to represent the simplified BCA and does not

provide any unnecessary features or impose any assumptions that we would not otherwise have made.

For these reasons, we will henceforth adopt the matrix notation of Markov chains and its associated terminology given in section 2.1. The relevant parts of Markov chains, will be covered conceptually as and when they are required in this chapter; indeed, to demonstrate that they are required.

The matrix defined in equation (3.19) gives the “transition matrix” for Algorithm 2, the highly simplified BCA with a clonal pool of 1, population of 1 and no elitism. The matrix construction presented in this section is also a useful component for the following modelling iterations, these more complete versions of BCA are described as Algorithm 3 in section 3.4 and Algorithm 4 in section 3.7. In the context of these remaining models we will refer to a matrix formed in this section as a “sample matrix” which we shall summarise in section 3.4.1 and investigate its properties.

This concludes section 3.3 as the model of the highly simplified version of the BCA Algorithm 2 defined in pseudo code at the beginning of the section has been completed. The concepts and tools developed in this section, along with the adoption of Markov chains provide a strong foundation for the development of subsequent models.

3.4 The BCA with population of 1 and a clonal pool of 1

This section will construct an exact mathematical model of a simplified version of the B-Cell Algorithm. This simplified algorithm is defined here

Algorithm 3: The B-Cell Algorithm with a population of 1, a clonal pool of 1 and elitism

- Step 1: Initialise a binary string x of length l . Each element in the string has a 0.5 probability of having the value 1
- Step 2: evaluate $g(x)$
- Step 3: While stopping criterion not met:
- Step 4: Create x_c , a clone of x mutated using the contiguous region hypermutation operator
- Step 5: Evaluate $g(x_c)$
- Step 7: If $g(x_c) > g(x)$ then replace x by x_c
- Step 8: Return to step 3

The objective function $g(x)$ is defined as a search space with 2^l input output pairs, where $l \in \mathbb{Z}^+$. All input values, x , can be represented by the configurations of a binary string of length l and all output values, $g(x)$, are numerical values that occur on a standard floating point representation.

The difference between Algorithm 3 and Algorithm 2 modelled in section 3.3 is that this algorithm includes elitism; elitism is included in Algorithm 3 by the "if" statement in step 7. The model constructed in section 3.3 is labelled the "sample matrix". We will investigate the properties of the sample matrix in section 3.4.1 and we shall describe the "possible transit matrix" in section 3.4.2. The transition matrix for Algorithm 3 is constructed by combining the modular components of the model 3.4.3. A worked example is given in

section 3.4.4 to aid understanding of this novel method of construction of a transition matrix.

3.4.1 The Sample Matrix

The sample matrix first described in section 3.3.6 represents the method of determining the next point in the search space to be evaluated by the objective function, but contains no information on the outcome of the evaluation, or what to do with that outcome. The sample matrix is dependent on the search space grid, the representation and the mutation operator, but it is independent of the function.

If the probabilities of all the mutation masks are calculable, it is possible to construct the sample matrix numerically. The sample matrix contains the probabilities that the algorithm will “sample” a given point within the search space. Sampling a point is equivalent to generating a new potential solution via the CRHO and it being evaluated by the objective function.

To construct the BCA sample matrix from equations (3.16) and (3.18) the binary representation being used must be employed as an interpreter between the search space grid values of states and the mutation mask required to transit between the states. This is done by converting the initial and final states into binary strings and using the exclusive or operator to determine which mutation mask is required to transit between these states. We can determine which mutation mask is required for each transition simply as shown in Table 5 and Table 6. Equations (3.16) and (3.18) can be utilised to replace the mutation mask with the algebraic sampling probability, dependant on the mutation rate r (or a numeric answer if the value of r is specified).

This is an efficient method of constructing what is, potentially, a very large $2^l \times 2^l$ matrix. There are 2^l possible functions that can be defined by equations (3.16) and (3.18). As no two end states (of any initial state) are

the same and each row has 2^l elements, it is clear that mutation masks only occur once per row, on all rows. Once one line of the sample matrix has been calculated, it is possible to “unpack” the rest of the matrix using just the exclusive or operator and copying the relevant entry from the completed row.

It is conceptually helpful to define the first row as $i=0$ instead of $i=1$, as this brings the row and column indices into alignment with the states they represent, when considering an integer space. In order to calculate the first row of the matrix $i=0$ we require use of the formula to calculate the probability of each mutation mask occurring. A complete set of mutation masks from $T=0$ to $T=2^l-1$ make up the elements of the first row of the matrix. Subsequent rows can be calculated without calling upon the functions (3.16) and (3.18) as the probabilities of an exhaustive set of mutation masks has already been calculated, it is simply a matter of copying the value from the correct element in the first row of the matrix. This can be achieved by converting the row index and column index into binary strings and calculating the exclusive or of these strings. The integer version of the resultant binary string gives the column index of the element in row 0 that has the same mutation mask (and hence the same probability value).

A further optimisation of the method of constructing the sample matrix is available as the exclusive or operator forms a triple between three strings, hence the exclusive or of ij will be the same as the exclusive or of ji . This makes the matrix (3.19) symmetric about the diagonal, which means when calculating which value should be copied into the upper triangular region ij you can copy the same value into the symmetric opposed element ji .

We know that the sum of the probabilities in any row is equal to 1 shown in equation (3.20), this means the sample matrix is by definition a stochastic matrix, as defined in section 2.1.2.

$$\sum_{T=0}^{2^l-1} f_T = 1 \quad (3.20)$$

This is a simple property, but an important one that will be utilised many times throughout the remainder of this chapter.

3.4.2 The Possible Transit Matrix

We know that we can construct the sample matrix if we restrict the clonal pool size of the BCA to one. Under this condition and with a population of one, *the position in the search space can be considered to be the state of the algorithm*. The sample matrix gives the probability distribution of where in the search space to search next, for any given current state.

After the algorithm samples a point via the mutation operator, the algorithm must decide if it will “transit” to this new state, or if it will remain at the current state. The elitist mechanism in the algorithm gives rules on whether the newly proposed state should be transited to or not. These rules given in step 7 of Algorithm 3 are already a transparent model; all that is required to allow the rule to be obtained, is for the objective function (and the associated search space grid) to be defined.

The elitist operator is defined such that if the new state is of higher affinity than the current state, then the algorithm transits to that state with probability 1. If the new state is of lower or equal affinity than the current state, then the algorithm transits to that state with probability 0 (the algorithm remains in its current state with probability 1).

The possible transit matrix is dependent on the search space and the function space but is independent of the representation. We can use this to define a possible transit matrix: For a given initial state affinity, consider the affinity of all the possible end states. If the affinity of the end state is greater than the affinity of the initial state then the matrix element

representing that transition is assigned the value 1. If the affinity is equal or less than that of the initial state then the matrix element is assigned the value 0.

3.4.3 Constructing the Transition Matrix

The sample matrix and possible transit matrix can be combined by means of an element-to-element multiplication to form the transition matrix for Algorithm 3 with elitism that is correct, except for the values of diagonal elements. During the element-to-element multiplication, some of the values in the sample matrix will be multiplied by 0, hence some of the rows will no longer sum to 1.

A transition matrix of a Markov chain must be a stochastic matrix; all rows must sum to one. In Algorithm 3, a rejected move results in a null move (remaining in the current state); therefore the probabilities that were multiplied by 0 must be added to the diagonal element of the appropriate row. Thus the fully correct transition matrix is produced. This process is illustrated via a worked example given in section 3.4.4.

3.4.4 Worked Example

An example construction of the transition matrix follows. There is a convention used in this example that must be kept in mind in order to understand how each matrix is constructed: each row of a matrix represents a state at the start of an iteration and each column represents a state at the end of an iteration. The order of states is the same for both rows and columns, the states are the same as the search space starting from the top and left respectively. The search space will be $[0,1,2,3]$ represented by a binary string of length $l=2$. For the sake of familiarity, standard binary shall be used as the representation. The objective function to be maximized in this example is

$$g(x) = -x^2 + 4x \quad (3.21)$$

where $x \in [0,1,2,3]$.

This function and search space grid have been chosen as they contain a pair of points that have the same affinity as well as a maximum and a minimum. Hence all possible cases of the possible transit matrix are therefore demonstrated in this example.

We shall obtain the sample matrix and possible transit matrix and then combine them to form the transition matrix that represents the algorithm in full when applied to $g(x)$ on this search space grid. The search space global optima (SSGO) is the same as the maximum of the function $x = 2$.

The sample matrix for this problem has been obtained in section 3.3 defined in equation (3.19), but is repeated here in equation (3.22) for ease of use in this example. Note that the objective function (3.21) was not required to define the sample matrix.

$$\mathbf{M}_1 = \begin{pmatrix} f_0 & f_1 & f_2 & f_3 \\ f_1 & f_0 & f_3 & f_2 \\ f_2 & f_3 & f_0 & f_1 \\ f_3 & f_2 & f_1 & f_0 \end{pmatrix} \quad (3.22)$$

The values for these probabilities for a given value of the mutation rate r , can be calculated explicitly from equations (3.16) and (3.18).

The possible transit matrix is obtained from the objective function equation (3.21). If the value of $g(x')$ of the end state is greater than the value of $g(x)$ for the initial state then the matrix element is assigned the value 1. If it is less than or of equal value then it is assigned the value 0.

$$\mathbf{M}_2 = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \quad (3.23)$$

Combining the sample matrix equation (3.22) and possible transit matrix equation (3.23) using an element-to-element multiplication an incomplete transition matrix can be formed. All positive transitions are represented in this mid stage, but the null transitions that occur on the diagonal of the matrix are not represented.

$$\mathbf{M}_3 = \begin{pmatrix} 0 & f_1 & f_2 & f_3 \\ 0 & 0 & f_3 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & f_1 & 0 \end{pmatrix} \quad (3.24)$$

When the transition matrix is complete, the sum of each row must equal 1, by definition. As each row is only missing the diagonal entry it is possible to determine this missing element cheaply by finding the difference between 1 and the sum of the rows in equation (3.24); this difference can then be added to the relative diagonal element.

$$\mathbf{P} = \begin{pmatrix} f_0 & f_1 & f_2 & f_3 \\ 0 & 1-f_3 & f_3 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & f_1 & 1-f_1 \end{pmatrix} \quad (3.25)$$

Equation (3.25) is the full algebraic transition matrix for Algorithm 3 applied to the example problem (3.21) given that all the elements can be calculated explicitly from equations (3.16) and (3.18).

Choosing an arbitrary value of mutation rate between 0 and 1 it is possible to obtain the transition matrix in its numerical form. Let $r = 0.5$, substituting $l = 2$ and $r = 0.5$ into equations (3.16) and (3.18) we obtain

$$\begin{aligned}
f_0 &= \frac{3(1-r) + (1-r)^2}{l^2} = 0.4375 \\
f_1 &= \frac{(1-r)r + 2r}{l^2} = 0.3125 \\
f_2 &= \frac{r + (1-r)r}{l^2} = 0.1875 \\
f_3 &= \frac{r^2}{l^2} = 0.0625
\end{aligned} \tag{3.26}$$

Substituting the results from (3.26) into (3.25) we obtain the numerical transition matrix

$$\mathbf{P} = \begin{pmatrix} 0.4375 & 0.3125 & 0.1875 & 0.0625 \\ 0 & 0.9375 & 0.0625 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0.3125 & 0.6875 \end{pmatrix} \tag{3.27}$$

This ends the worked example, which has constructed a model of the simplified BCA with elitism Algorithm 3. The next section presents the proof of convergence of the simplified BCA with elitism.

3.5 Proof of convergence

The formalism (not including equations specific to the proof of convergence of the BCA) described in this section by equations (3.28)-(3.36) and surrounding text was primarily contributed by Hone, first appearing in the literature in (Stepney et al., 2005). This section, including the parts that appear in (Stepney et al., 2005), appears in full in (Clark et al., 2005) and is revisited in (Timmis et al., 2008c).

An important property of the B-cell algorithm, from a modelling perspective, is that it has a non-interacting population. A model for a population of one is completely general, as it can be scaled to a population of any size by

using rules for independent probabilities. This greatly simplifies the modelling process.

For a maximization problem, we can represent the state of our system by a random variable X_t , which changes with the time t , and then the objective will be to maximize a given objective function $g(x)$. The state X_t corresponds to the value of the bit string corresponding to a cell. The aim of the algorithm is to find the state value x that maximizes the function $g(x)$, by making many iterations in time. At each time t , a clone is taken from the cell, hypermutation is applied to the clone, and if it happens that $g(C_t) > g(X_t)$ then the next state value is $X_{t+1} = C_t$, otherwise the original cell is kept and $X_{t+1} = X_t$; see (Kelsey and Timmis, 2003) for more details. At each stage there is a probability for transition to a new state (bit string) value for X_{t+1} , and the BCA is purely elitist in the sense that only mutations that result in improvement are kept (so that the value of $g(x)$ is non-decreasing with t). To describe the evolution of a cell in the BCA in terms of a Markov chain, we label the possible states by an index j running from 0 to $N-1$, where $N = 2^l$ is the number of possible states (i.e. the number of possible strings of length l). Then we choose the value of the bit string at time t to be given by the probability distribution $\mathbf{v}_t = (v_{t,1}, v_{t,2}, \dots, v_{t,N})$, with the j th component of the row vector \mathbf{v}_t being just

$$v_{t,j} := P(X_t = j) \quad (3.28)$$

that is the probability of being in state j at time t . Furthermore, the probability of transition between state j and another state k is independent of the time t , and so can be represented by the $N \times N$ transition matrix $\mathbf{P} := (P_{jk})$ with entries

$$P_{jk} = P(X_{t+1} = k | X_t = j) \quad (3.29)$$

To work out the probability distribution at time $t+1$, the standard rules of conditional probability imply that

$$v_{t+1,k} = \sum_{j=1}^N v_{tj} P_{jk} \quad (3.30)$$

Therefore, rewriting the equation (3.30) in matrix notation we have

$$\mathbf{v}_{t+1} = \mathbf{v}_t \mathbf{P} \quad (3.31)$$

which describes the time evolution of the row vector \mathbf{v}_t . The row vector \mathbf{v}_t is the probability distribution of the state X_t at time t . Note that \mathbf{P} is a stochastic matrix: all of its entries lie between 0 and 1, and the row sums satisfy

$$\sum_{k=1}^N P_{jk} = 1 \quad (3.32)$$

i.e. from state j , the cell must make a transition somewhere (including null transitions) with probability 1.

Because the transition matrix \mathbf{P} is time-independent, equation (3.31) means that the probability distribution vector \mathbf{v}_t at time t can be written immediately in terms of the initial distribution, as

$$\mathbf{v}_t = \mathbf{v}_0 \mathbf{P}^t \quad (3.33)$$

It is evident from the form of (3.33) that if we wish to understand the long-term behavior of the algorithm, we need to understand what happens to the powers of the transition matrix \mathbf{P}^t as $t \rightarrow \infty$. In fact, for the BCA it is further possible to prove that where there is a unique optimum state, it is reached with probability one in the limit $t \rightarrow \infty$; in the terminology of Markov chains (Grimmett and Stirzaker, 1982), a unique optimum is an absorbing state. Similarly, if there are several optima (i.e. different values of x for which $g(x)$ takes the same optimal value) then it turns out that the overall probability of lying in at least one of these optima also tends to one as $t \rightarrow \infty$.

For our detailed analysis of the Markov chain model of Algorithm 3 defined in section 3.4, we will rely on some standard properties and results concerning Markov chains. It is helpful to introduce the notation

$$P_{jk}(n) = P(X_{t+n} = k \mid X_t = j) \quad (3.34)$$

for the entries of the matrix \mathbf{P}^n (the n th powers of the transition matrix). A state j can be said to be persistent if the sum

$$\sum_{n=0}^{\infty} P_{jj}(n) = \infty \quad (3.35)$$

Otherwise if the sum converges, so that

$$\sum_{n=0}^{\infty} P_{jj}(n) < \infty \quad (3.36)$$

then the state j is transient. By the Decomposition Theorem (see (Grimmett and Stirzaker, 1982) page 123) any Markov chain can be decomposed into the set of transient states together with some irreducible closed sets of persistent states. In this section we show that for the BCA, all of the non-optimal states are transient, while all the optima are persistent. Furthermore, the optima turn out to be absorbing states of the chain, in the sense that once an optimum is reached it is never left (due to the elitist nature of the BCA). Since the BCA corresponds to a Markov chain with a finite number of states, we can calculate the transition matrix \mathbf{P} explicitly for a given objective function.

In order to prove that the BCA is convergent absolute, we shall show that the BCA model will always take the form of an absorbing Markov chain with a non-zero one step transition probability from all points in the space to a global optima. In the terminology of Markov chains, all non-optimal states of the BCA are transient. Moreover, all the optima are absorbing states.

Proposition 1: All non-optimal states are transient provided $0 < r < 1$

Proof: Under the condition $0 < r < 1$, the sample matrix contains only non-zero elements. By inspection of equation (3.16) it is clear that $f_T > 0$ if $(1-r) > 0$ and $r > 0$. Thus, we impose the condition $1 > r > 0$, this condition also makes $f_0 > 0$ as can be seen in equation (3.18). (The formulae (3.16), (3.18) would need to be modified in the case when the clonal pool C has more than one member but the condition for $f_T > 0$ is the same.) Hence it is possible to reach the absorbing state in one step from any initial state. This is irrespective of the function due to the fact that by definition of the possible transit matrix, transition from a non-optimal state to an optimum is allowed with probability 1. Hence, for a non-optimal state j the probability of remaining in that state for one time step is $p = P_{jj} < 1$. Once a state has been left for a state of higher affinity, it can never return to a previously

occupied state, since the possible transit matrix forbids transitions to states of lower affinity, hence

$$P_{jj}(n) = p^n; \quad (3.37)$$

it follows that

$$\sum_{n=0}^{\infty} P_{jj}(n) = \sum_{n=0}^{\infty} p^n = \frac{1}{(1-p)} < \infty \quad (3.38)$$

and hence state j is transient.

The general theory of Markov chains is particularly effective in the case of irreducible chains. The chain corresponding to the BCA is reducible, however: by the Decomposition Theorem (Grimmett and Stirzaker, 1982) it can be partitioned into the set of transient non-optimal states together with the (disjoint) closed sets of absorbing states corresponding to the optima. Adopting the terminology of (Villalobos-Arias et al., 2004), we can say that the non-optimal states of the BCA are inessential, in the sense that for all such non-optimal states j there exists another state k such that j can make a transition to k but not vice-versa: it is sufficient to choose any state k with a larger value of the objective function. Then if there are M optima we can partition the transition matrix \mathbf{P} as follows

$$\mathbf{P} = \begin{pmatrix} \mathbf{1} & \mathbf{0} \\ \mathbf{R} & \mathbf{Q} \end{pmatrix} \quad (3.39)$$

where $\mathbf{1}$ denotes the $M \times M$ identity matrix, \mathbf{Q} corresponds to the transitions between the inessential states and \mathbf{R} corresponds to transitions from inessential to optimal states. It follows from standard properties of stochastic matrices (see e.g. (Seneta, 1981)) that the powers $\mathbf{Q}^t \rightarrow 0$ as $t \rightarrow \infty$. Using this partitioning, we arrive at the following:

Proposition 2: All optima are absorbing states

Proof: By definition, the possible transit matrix prohibits transition from a global optimum to any other state, even another global optima. Therefore once the algorithm enters a global optimum it satisfies the condition for persistence given in equation (3.35). Clearly for an optimum state j we have $p = P_{jj} = 1$ (i.e. once an optimum is reached then one remains there with probability one), so the corresponding sum (3.38) diverges and the optima are all persistent states; since they do not communicate with any other state, they are also absorbing.

Theorem 1: The BCA is convergent absolute provided $0 < r < 1$, where r is the probability of mutation for bits contained within the contiguous region.

Proof: We have the state vector calculated from powers of the transition matrix according to

$$\mathbf{v}_t = \mathbf{v}_0 \mathbf{P}^t . \quad (3.40)$$

Now

$$\mathbf{P}^t = \begin{pmatrix} \mathbf{1} & \mathbf{0} \\ \mathbf{R}_t & \mathbf{Q}^t \end{pmatrix} \quad (3.41)$$

for some matrix \mathbf{R}_t constructed from sums of powers of \mathbf{Q} acting on \mathbf{R} . Also, it follows from standard properties of stochastic matrices (see e.g. (Seneta, 1981)) that the powers $\mathbf{Q}^t \rightarrow 0$ as $t \rightarrow \infty$. So in the limit $t \rightarrow \infty$, the powers of \mathbf{P} take the form

$$\mathbf{P}^\infty = \begin{pmatrix} \mathbf{1} & \mathbf{0} \\ \mathbf{R}_\infty & \mathbf{0} \end{pmatrix} \quad (3.42)$$

Acting with this matrix on the initial state vector \mathbf{v}_0 we see that the probability of being in any of the non-optimal transient states tends to zero as $t \rightarrow \infty$, and hence the probability of ending up in an optimum tends to one, as required.

3.5.1 General proof of convergence

When Theorem 1 was originally published in (Clark et al., 2005), it was not stated that it was applicable beyond the BCA. In this section we state that the proof of Theorem 1 is in fact general and we give the general conditions that an algorithm must satisfy in order to be covered by the proof.

If we examine the constraint $0 < r < 1$ from proposition 1, we can translate this constraint on the value of the mutation rate to a constraint on the form of the sample matrix. When $0 < r < 1$ all mutation masks occur with some probability greater than zero, therefore the sample matrix is positive. The proof of proposition 1 also proves the *equivalent proposition: All non-optimal states are transient provided the sample matrix is positive*. The definition of a positive matrix is given in section 2.1.2. Proposition 2 is already in terms of conditions on the possible transit matrix.

When the possible transit matrix of an elitist algorithm is combined with a sample matrix it takes the form (3.39), a reducible Markov chain and the result shown in theorem 1 holds. Therefore any optimisation algorithm can be shown to be convergent absolute by the method used in theorem 1 if it satisfies these two conditions:

- The sample matrix is positive
- The algorithm is elitist

A similar observation has been made independently by Cutello et al (Cutello et al., 2007), whose conditions are exactly equivalent to the conditions

given here. The observation made by Cutello et al are based on the proofs given by Rudolph in (Rudolph, 1998). Rudolph presents these arguments for EAs, Cutello uses similar formalism to extend Rudolph's arguments to Immune Algorithms in general.

In some ways the proof we have presented is easier to understand without the Markov chain formalism and can be summed up by the argument: If an optimisation algorithm has a non-zero probability of finding an optima in one step from any point in the search space, zero probability of losing an optimum once it has been found and infinite chances to find the optima, then it will find it with probability one. This is in essence all that proofs of convergence (that are based on infinite iterations) are saying.

It should be noted that while an algorithm that satisfies the two conditions is proven to be convergent, not satisfying the conditions does not prove the algorithm does not converge. Simulated Annealing for example does not satisfy the first condition and for a temperature $T > 0$ does not satisfy the second condition either. A model SA is presented in section 6.2, an example sample matrix is given by equation (6.12). The model presented in section 6.2 is based on, and exactly equivalent to, the model given in (Häggström, 2002) which is shown to converge for an appropriate annealing schedule (Häggström, 2002).

3.6 Demonstration of numerical application of the model

Other complete and exact Markov chain models, for example in the GA literature (Nix and Vose, 1992) provide good analytical follow up work (Vose, 1995) but numerical application of the model (De Jong et al., 1995) is constrained by the complexity of the model to unrealistically small variables, e.g. string length 2. The BCA produces a model with a state space small enough to do numerical work with realistic values of the models variables, string lengths as large as 16 may be possible on current hardware.

Potentially, the BCA model can be utilised to gain insight into all facets of the BCA, the most obvious of these is the effect of the mutation rate on the convergence rate of the algorithm. We have probed this application of the model on a simple one-dimensional quadratic function with a search space of size $|S|=2^8$. In this example all values are accurate within the limits of double precision (IEEE standard 754, 64-bit floating point). Figure 5 presents a coarse sweep of the range of mutation rates, and produces two features of particular interest.

First, a mutation rate of 1 (using the CRHO) is shown to be the optimal mutation rate for the problem, this is interesting because $r=1$ lies outside the conditions for which the BCA has been proven to be convergent absolute given in section 3.5. $r=1$ is not included under the conditions of the proof because it has the potential to produce persistent non-optimal states in the BCA as some of the one step transition probabilities to the SSGO are zero. However, the function space in this example is easy enough for all states to have a non-zero multi-step probability of reaching the SSGO allowing the BCA to converge on the SSGO with probability 1. Note that the positional bias imposed by the CRHO, as described in 3.1 and 3.3.4, means that the mutation rate in the CRHO produces a significantly different behaviour than, for example, a mutation rate applied independently to each bit in the string. The results obtained here should be considered only in the context of the CRHO. Comparing what are effective values of the mutation rate for the CRHO with effective values of the mutation rate for other operators may not be valid.

Second, the line for $r=0.1$ on Figure 5 looks flat, which is somewhat unexpected considering that the BCA has been shown to be convergent absolute given $0 < r < 1$. This raised the question when does the probability of being in the optimum become 1? Further numerical investigation showed that at 2^{20} iterations, the probability of being in the optimum had risen to ~ 0.85 , at 2^{40} iterations the probability had risen to ~ 1 . This result was

obtained by squaring the transition matrix at each step; hence only 40 matrix multiplications were required. This example demonstrates the power of the model to look at the extreme long-term behaviour of the model numerically as well as analytically. It also illustrates the fact that just because an optimisation algorithm is shown to converge, it does not mean that the optimisation algorithm is necessarily an efficient method of optimisation.

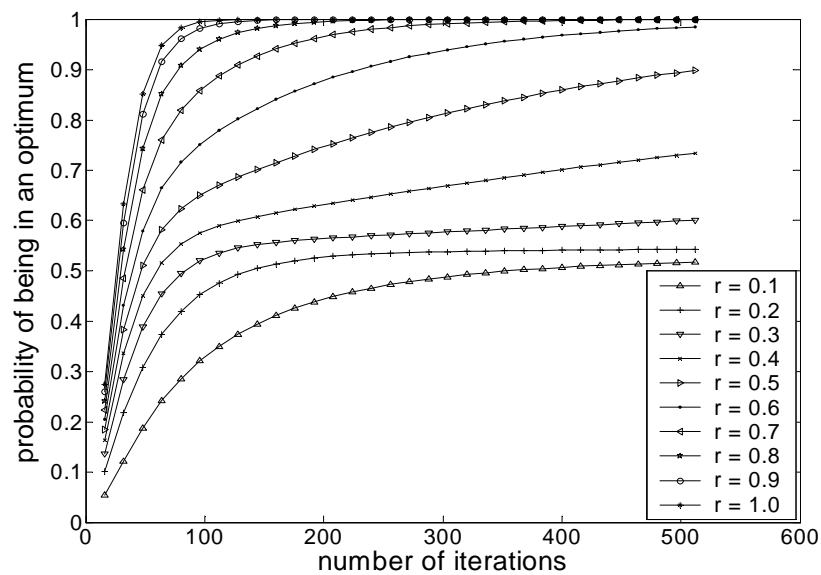


Figure 5. Examination of the effect of mutation rate, r , on the B cell algorithm convergence rates for a one-dimensional quadratic function. String length $l = 8$. Figure taken from (Clark et al., 2005).

This numerical work presented in Figure 5 shows some promising results, both for the BCA and for numerical analysis of Markov chain models of optimisation algorithms. The size of the search space for this particular problem is $|S| = 2^8 = 256$, yet Figure 5 shows the algorithm to converge on the SSGO with probability ~ 1 using less than 256 function calls for the best performing mutation rates $r = 1, 0.9, 0.8$. This means that the BCA is sure to converge on the SSGO of *this particular function* without having searched the whole space. Obviously, this result will not generalise to functions of arbitrary complexity, but it is encouraging to see such high probabilities of

convergence using a number of function evaluations less than half the size of the search space. It should also be noted that for the worst performing mutation rates the BCA took many orders of magnitude more function calls than there are points in the search space in order to find the SSGO with probability 1 (within the limits of double precision).

A point made well by Häggström (Häggström, 2002) is that the case we should have in mind when considering optimisation algorithms is that they are searching a space that is so huge, plotting the space becomes intractable. So presenting results that make as many or more function calls than points in the space is of little practical value. The theoretical results presented by Häggström later in the same chapter in (Häggström, 2002) are in infinite iterations. We also present results in infinite iterations; we, like Häggström also attempt to stress considering optimisation problems that make plotting intractable and hence the need for results in less function calls than there are points in the search space. We accept that these results are not going to be proofs of convergence with probability 1, but suggest that numerical work on Markov chain models provides the most promising source for results of practical interest. We favour numerical analysis over derivation of theoretical convergence bounds for particular problem classes because numerical analysis can be applied to investigate search spaces of arbitrary complexity, allowing numerical analysis to be applied to problems from any class.

3.7 BCA with clonal pool of C , population of 1 and elitism

This section will construct an exact mathematical model of the version of the B-Cell Algorithm that includes the clonal pool and elitism. This simplified algorithm is defined here as Algorithm 4.

Algorithm 4: The B-Cell Algorithm with a clonal pool of C , population of 1 and elitism

- Step 1: Initialise a binary string x of length l . Each element in the string has a 0.5 probability of having the value 1
- Step 2: evaluate $g(x)$
- Step 3: While stopping criterion not met:
- Step 4: Create C clones by making C copies of x and mutating them using the contiguous region hypermutation operator
- Step 5: Evaluate $g(x_c)$ each of the C clones
- Step 6: x' is defined as the clone with the greatest $g(x_c)$. In the case where k clones have the same highest $g(x_c)$ value, each of the k clones has a probability of k^{-1} of being selected to become x'
- Step 7: If $g(x') > g(x)$ then replace x by x'
- Step 8: Return to step 3

The optimisation problem is defined as a search space with 2^l input output pairs, where $l \in \mathbb{Z}^+$. All input values, x , can be represented by the configurations of a binary string of length L and all output values, $g(x)$, are numerical values that occur on a standard floating point representation.

3.7.1 How cloning affects the state space of the BCA

Let us consider the mechanics of the clonal selection operator in detail and consider what the algorithm's minimal state space is. An input string is presented from the current generation, the string is cloned C times and each clone is mutated by the CRHO. The best of the clones is selected and the other clones are killed. There are two possible ways of defining the state: either from the current pool of clones, to the next pool of clones, or from the best clone to the best clone in the next generation. The methods would look at $C \rightarrow C$ or $1 \rightarrow C \rightarrow 1$ respectively, with the middle number being a hidden mid-state. The size of the state space would depend on C for the $C \rightarrow C$ approach. The size of the state space for the $1 \rightarrow C \rightarrow 1$ approach is independent of C (and minimal).

Although both methods are valid, it is far more desirable to consider state transitions from 1 to 1 instead of C to C . The step from a single input string to a pool of C clones is obtainable, as we already have the probabilities of sampling any string from a given input string. This is however only half an iteration; in order to complete the iteration we must obtain a method for determining the probability that a given end state will be the best end state, of all the end states in the clonal pool. There is also the issue of it being better than the current best state, but that will be taken care of by the possible transit matrix, so it should not be explicitly included.

3.7.2 Exploring a modification of the sample matrix

It is clear that to include the clonal pool, knowledge of the search space is required, so it is not a trivial modification of the sample matrix. Let us retain the same search space grid and function (3.21). If we rank the states in terms of affinity from highest to lowest we have: $x=2$ with the highest affinity. $x=1$ and $x=3$ with joint second highest affinity and $x=0$ with the lowest affinity.

We shall consider the simplest non-trivial case, a clonal pool of size 2. Given that we already have a method for applying elitism we do not need to consider if the clones are superior to the current state, just the probabilities that a given next state will be the state with the highest affinity within the clonal pool. Even though we do not need to compare with the current state at this stage we do need to declare a starting state in order to know what mutation masks are required for sampling each possible position in the search space. As before, for conceptual simplicity, we shall start in state zero so the mutation mask is the same as the end state. Let us consider the probability of each possible end state being the clone with highest affinity in the clonal pool.

Given that the algorithm is in state zero and has a clonal pool size of 2: What is the probability that the member of the clonal pool with highest affinity is state zero?

Having ranked the state space we know that state zero is the state of lowest affinity, so the only way it could be the state of highest affinity in the clonal pool is if both clones mutate to state zero. The clones are mutated independently. The mutation masks required are [00],[00] which occurs with probability f_0f_0 so the probability of this being the case is simply

$$p_{0,0} = f_0^2 \tag{3.43}$$

The notation $p_{i,j}$ is the probability of transition from state i to state j .

Let us now consider the probability that state one will be the member of the clonal pool with highest affinity. The obvious combinations are [01],[00] or [00],[01] or [01],[01] which will occur with probability $f_1f_0 + f_0f_1 + f_1^2$ as state one has a higher affinity than state zero. State one and state three have equal affinity, which makes things marginally more complicated. Recall from

the pseudo code that if two or more members of the clonal pool are of equal affinity, then the end state will be randomly selected between these members. So in the cases of [01],[11] or [11],[01] the probabilities will be shared equally between states one and three, so the probabilities for state one being selected are $\frac{1}{2}f_1f_3 + \frac{1}{2}f_3f_1$. So the total probability of state one being the highest affinity state in the clonal pool is

$$p_{0,1} = f_1^2 + 2f_0f_1 + f_1f_3 \quad (3.44)$$

State two is the optimum of the function, so all combinations that include state two will contribute to the total probability of state two being the highest affinity state in the clonal pool. Possible combinations are [10],[00] or [00],[10] or [10],[01] or [01],[10] or [10],[10] or [10],[11] or [11],[10]. So the total probability of state two being the highest affinity state in the clonal pool is

$$p_{0,2} = 2f_2f_0 + 2f_2f_1 + f_2^2 + 2f_2f_3 \quad (3.45)$$

State three is of equal affinity as state one, so we must remember to share the probability equally where both states appear together. Possible combinations are [11],[00] or [00],[11] or [11],[01] or [01],[11] or [11],[11]. So the total probability of state two being the highest affinity state in the clonal pool is

$$p_{0,3} = f_3^2 + 2f_0f_3 + f_1f_3 \quad (3.46)$$

Having the complete set of probabilities for transition from state zero to all states we can now sum these probabilities and check that they sum to one algebraically

$$\sum_{j=0}^{2^l-1} p_{0,j} = 1 \quad (3.47)$$

We require the mutation operator to satisfy the condition specified in equation (3.20), the sum of the mutation masks must equal one. Thus

$$\begin{aligned} \sum_{j=0}^3 p_{0,j} &= p_{0,0} + p_{0,1} + p_{0,2} + p_{0,3} \\ &= (f_0^2) + (f_1^2 + 2f_0f_1 + f_1f_3) + (2f_2f_0 + 2f_2f_1 + f_2^2 + 2f_2f_3) + (f_3^2 + 2f_0f_3 + f_1f_3) \\ &= f_0^2 + f_1^2 + f_2^2 + f_3^2 + 2f_0f_1 + 2f_0f_2 + 2f_0f_3 + 2f_1f_2 + 2f_1f_3 + 2f_2f_3 \quad (3.48) \\ &= (f_0 + f_1 + f_2 + f_3)(f_0 + f_1 + f_2 + f_3) \\ &= 1 \end{aligned}$$

In checking that the probabilities sum to one, a point of interest has been uncovered: All possible states and the number of ways each state can be generated is given by raising the sum of the probabilities of generating all mutation masks to the power of the number of clones.

So far we have looked at the probabilities of mutating from state zero to all other states with a clonal pool of size two.

	00	01	10	11
00	$p_{0,0}$	$p_{0,1}$	$p_{0,2}$	$p_{0,3}$

Table 7. Generic probabilities of sampling all states from a starting state of 00, for a clonal pool of arbitrary size.

Substituting equations (3.43),(3.44),(3.45) and (3.46) into Table 7 gives Table 8

	00	01	10	11
00	f_0^2	$f_1^2 + 2f_0f_1 + f_1f_3$	$2f_2f_0 + 2f_2f_1 + f_2^2 + 2f_2f_3$	$f_3^2 + 2f_0f_3 + f_1f_3$

Table 8. Example algebraic probabilities of sampling all states from a starting state of 00, assuming a clonal pool size of two

We should now go on to consider how to get all the required entries for all states to all states

	00	01	10	11
00	$p_{0,0}$	$p_{0,1}$	$p_{0,2}$	$p_{0,3}$
01	$p_{1,0}$	$p_{1,1}$	$p_{1,2}$	$p_{1,3}$
10	$p_{2,0}$	$p_{2,1}$	$p_{2,2}$	$p_{2,3}$
11	$p_{3,0}$	$p_{3,1}$	$p_{3,2}$	$p_{3,3}$

Table 9. Generic probabilities of sampling all states from a starting from all states, for a clonal pool of arbitrary size

Having completed the first line of Table 9 in Table 8, let us now consider more generally: what is the probability that the member of the clonal pool with highest affinity is state zero? State zero has the lowest affinity so the only way it can be the clone with highest affinity is if both clones mutate to state zero. This is general for all starting states, it is simply a matter of knowing the probability of the occurrence of the mutation mask required to mutate from the current state to the state zero. So, from state one, we would require [01],[01] which will occur with probability f_1^2 . From state two, we would require [10],[10] which will occur with probability f_2^2 . From state three, we would require [11],[11] which will occur with probability f_3^2 .

This mechanical replacement of mutation masks depending on starting state generalises to the whole of the rest of the table. The combinations of states that can occur to give each possible output remain the same; hence the form of the equation remains the same in each column of the table, it is

only the mutation masks that change depending on the starting state as shown in Table 10.

	00	01	10	11
00	f_0^2	$f_1^2 + 2f_0f_1 + f_1f_3$	$2f_2f_0 + 2f_2f_1 + f_2^2 + 2f_2f_3$	$f_3^2 + 2f_0f_3 + f_1f_3$
01	f_1^2	$f_0^2 + 2f_1f_0 + f_0f_2$	$2f_3f_1 + 2f_3f_0 + f_3^2 + 2f_3f_2$	$f_2^2 + 2f_1f_2 + f_0f_2$
10	f_2^2	$f_3^2 + 2f_2f_3 + f_3f_1$	$2f_0f_2 + 2f_0f_3 + f_0^2 + 2f_0f_1$	$f_1^2 + 2f_2f_1 + f_3f_1$
11	f_3^2	$f_2^2 + 2f_3f_2 + f_2f_0$	$2f_1f_3 + 2f_1f_2 + f_1^2 + 2f_1f_0$	$f_0^2 + 2f_3f_0 + f_2f_0$

Table 10. Example algebraic probabilities of sampling any states from a any starting state, assuming a clonal pool size of two

Once we have a completed row for a given clonal pool size and problem we can fill in the rest of the rows by making the appropriate substitution of mutation mask probabilities.

3.7.3 General construction of a clonal matrix

We previously noted that all possible states and the number of ways each state can be generated is given by raising the sum of the probabilities of generating all mutation masks to the power of the number of clones.

We also define the set of states in the search space to be

$$S = [s_0, s_1, \dots, s_{2^l-1}] \quad (3.49)$$

And that for all $s_i \in S$ there exists $g(s_i)$

$$\chi_h = [g(s_0), g(s_1), \dots, g(s_{2^l-1})] \quad (3.50)$$

We define h to be the number of different values $g(s)$, in χ and that $h \leq |S|$ as some state may be mapped to the same value $g(s)$ by the objective function g .

We define the set ξ_h and state that $s_i \in \xi_h$ if and only if $g(s_i) = \max(g(s))$ where $g(s) \in \chi_h$ and that

$$\chi_{h-1} = \chi_h - \xi_h \quad (3.51)$$

and we define the set I_h such that $i \in I_h$ if and only if $s_i \in \xi_h$.

Let us define a set of terms that occur due to the clonal pool

$$\Psi_h = \left(\sum_{T=0}^{2^{l-1}} f_T \right)^{|C|} \quad (3.52)$$

Where each term in Ψ_h is a product of $|C|$ instances of f_T . We also define the set ω_h as the set of all terms such that $T=i$ for all $i \in I_h$.

The probability of mutating from state 0 to a state $i \in I_h$ is given by $p_{0,i}$ which is the sum of all terms in the set ω_h which contain $T=i$ $i \in I_h$; in the event a term has κ instances of $T=i$ $i \in I_h$, κ^{-1} of that term is added to all $p_{0,i}$ $i \in I_h$.

This process is repeated for $h-1, h-2, \dots, 1$ which provides the $p_{0,j}$ for all j . In general for $p_{i,j}$ the form of the equation $p_{0,j}$ and the mutation masks f_T are permuted as required.

3.8 Summary

We considered 3 models of the B-Cell Algorithm (BCA): from a highly simplified version of the algorithm defined as Algorithm 2 in section 3.3, an elitist version of the highly simplified algorithm defined as Algorithm 3 in section 3.4 through to a complete and exact model of the BCA (disregarding the population) defined as Algorithm 4 in section 3.7. The three models taken together form a case study for modular modelling of optimisation algorithms.

For the model of Algorithm 2 we only needed to know the probability distribution of the BCA's mutation operator the Contiguous Region Hypermutation Operator (CRHO) which is described by equations (3.16) and (3.18), subsequently the model took the form of a Markov chain in section 3.3.6.

In section 3.4 we construct a model of Algorithm 3, which differs from Algorithm 2 by the inclusion of an elitist operator. The introduction of elitism requires careful thought about what the internal state of Algorithm 3 is and how it is updated. Elitism can be encapsulated in a matrix which maps out transitions that are allowed 1, and transitions that are not allowed 0. This "possible transit matrix" is not however a Markov chain of Algorithm 3. The possible transit matrix for Algorithm 3 was combined with the "sample matrix" (defined by the model of Algorithm 2) to produce a Markov chain model of Algorithm 3.

In section 3.5 the form of the transition matrix that represents Algorithm 3 was used as the basis to construct a proof, showing Algorithm 3 to be convergent absolute; meaning that Algorithm 3 will always find one of the set of optima of any function if given sufficient time. The proof was generalised in section 3.5.1 and shows that the proof of convergence is inclusive of any algorithm that can satisfies these two conditions:

- The sample matrix is positive
- The algorithm is elitist

The model of Algorithm 3 and the proof of convergence appear in (Clark et al., 2005) along with a brief numerical survey of how the algorithm's rate of convergence is affected by the mutation rate.

The brief numerical survey presented in section 3.6 demonstrates that the BCA can locate the optimum of a simple function with a probability approaching 1 using less than half the number of function evaluations that would be required to enumerate the space. The variation in performance (number of function evaluations required to reach a probability approaching 1 of having located the optimum) of the BCA on this simple function spanned many orders of magnitude. This demonstrates the importance of the mutation rate to the effectiveness of the algorithm and that optimal rates can be obtained via numerical analysis of Markov chain models. It is noted that there is potential for further numerical investigation of Markov chain models to provide results from theoretical analysis of optimisation algorithms that are of practical interest.

Algorithm 4, modelled in section 3.7, includes a clonal selection operator as well as the mutation operator and elitism that appear in Algorithm 3. Consideration was given to what the minimal state of the algorithm is in the case of an algorithm with a clonal pool. It was found that the content of the clonal pool is not required to define the state of the algorithm; it is sufficient to have the clonal pool explicitly included in the state transition probabilities. Maintaining a minimal state space makes numerical work on algorithms with clonal pools possible for non-trivial search spaces.

Chapter 4: Bounding the rate of convergence and approximate calculations

This chapter contributes to the thesis hypothesis by demonstrating that the models produced in this thesis can be used to further the understanding of optimisation algorithms through theoretical investigation. Based on the model of the BCA constructed in section 3.4, we present bounds on the rate of convergence of an unknown function mapped onto a search space grid of a known size. A description of bounding is given in this section. We present our approximate calculation methods in section 4.1, from these methods we obtain upper and lower bounds to the rate of convergence. In section 4.4 we compare the bounds we obtain here with those obtained by the GA community (Aytug and Koehler, 2000) which requires us to re-arrange our set of bounds to have the same LHS of the equation as the bounds obtained in the literature. Algebraic comparison of the bounds obtained here shows them to be approximately equivalent to the bounds obtained for GAs. A discussion of bounding unknown functions is given in section 4.5 and it is concluded that while it is interesting that bounds obtained here are equivalent to bounds obtained via different methodology applied to a different class of algorithm; bounds on the rate of convergence of an *unknown* function will never be of practical interest.

Before we can discuss bounding the rate of convergence we must define how the rate of convergence or performance an algorithm is measured. The performance of an algorithm can be expressed in two ways: $\delta(t)$ which is the probability δ of having found the optima, given as a function of the number of iterations t . Alternatively we have $t(\delta)$, which is the number of iterations required t , to have found the optima with the probability δ . Both of these performance measures can be bounded $\delta_2(t) < \delta(t) < \delta_1(t)$, or

$t_1(\delta) < t(\delta) < t_2(\delta)$. Note that the numbering is reversed between the two methods of bounding. The reason for the reversal is best explained in Figure 6.

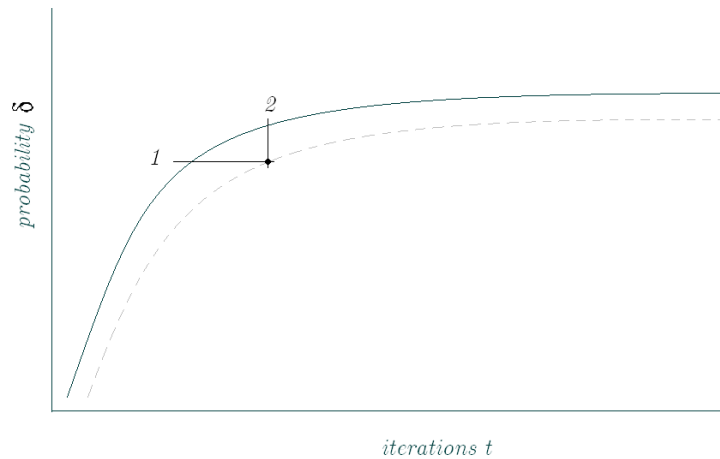


Figure 6. Bounding the rate of convergence. The curved black line is the true rate of convergence on the optima, while the dashed line is a line that bounds it. Whether the dashed line is an upper or lower bound is subjective, depending on the direction you project onto the black line. Projection 1: for a given value of probability of being in the optima δ , the dashed line will be an upper bound on the number of iterations required $t_1(\delta)$. Projection 2: for a given number of iterations t , the dashed line will be a lower bound of the probability of being in the optima $\delta_1(t)$.

Due to the size of the search spaces that are required by real world applications of optimisation algorithms, the prospect of numerical investigation of the Markov chain model at the required gradation cannot be undertaken; the principle reason for this is that it would require a plot of the space, in a real world application that is not obtainable. However something can still be said about the rate of convergence; the “worst case” rate of convergence is intuitively the most important for an unknown function, as in general little will be known about the characteristics of a real

world problem. For practicality, calculation of a bounding behaviour must be computationally far cheaper than a full Markov chain calculation.

The intent behind bounding the rate of convergence is that practitioners could in principle use the bound as a guide to how well the optimisation algorithm is going to perform on their unknown function. However, bounds produced by the existing methods (Aytug and Koehler, 2000, Greenhalgh and Marshall, 2001) give a number of iterations required to have found the optima with any significant probability far in excess of the number of points in the search space.

The results obtained here for bounding the rate of convergence are derived from an unconventional starting point: an investigation of algorithm performance on a “needle in a haystack” function. A needle in a haystack function, is any function that has one input state that corresponds to an optimal output value, with all other input states corresponding to a uniform lower affinity value. A formal example of a needle in a haystack is adapted from (Jansen, 1999) $\{g_a \mid a \in \{0,1\}^n\}$ with $g_a(a) = 1$ and $g_a(x) = 0, \forall x \neq a$.

The function itself is fairly uninteresting, but in conjunction with an algorithm that utilises an elitist mechanism, it greatly simplifies the resulting Markov chain of elitist algorithms. Specifically we are considering the model of Algorithm 3 presented in chapter 3. Analysing the resulting simple transition matrix brings to light some interesting function specific properties, but it also allows some more general statements to be made, providing the basis for an upper and lower bounding method for the rate of convergence.

Considering that transition to states of equal or lower fitness is forbidden by the elitist mechanism as shown in section 3.5, we can immediately determine that the only non-zero elements of the transition lie on the diagonal and the column corresponding to the optimum. Considering that each column in the sample matrix contains a complete set of mutation

masks ($\sum_{T=0}^{2^l} f_T = 1$) and that the location of f_0 (the null mutation) is on the diagonal we can deduce that the probability of reaching the optimum (averaged over all possible starting positions) in one step is given by

$$1 + \frac{\sum_{T=1}^{2^l} f_T}{2^l}$$

without knowing the location of the optimum. The 1 occurs where the column corresponding to the optimum intersects the diagonal, as there are no valid state transitions away from this state the probability of remaining there is 1. A typical transition matrix is represented in Figure 7

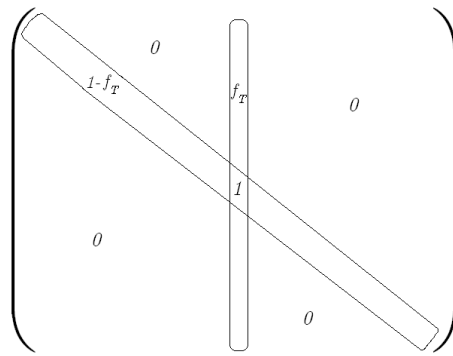


Figure 7: The transition matrix for an elitist algorithm optimising a needle in a haystack function. The column of the optimum contains all the values f_T for $T > 0$ exactly once. $f_{T=0}$ is not explicitly represented in the matrix as it occurs where the diagonal intersects the column of the optima where there is a 1, as all transitions from an optima are forbidden. All other diagonal elements take the complementary value $1 - f_T$ for a given value of T for the same reason. It should be noted that while the positioning of the column is placed arbitrarily to reflect the unknown location of the optima, all transition matrices that represent needle in a haystack functions can be permuted to be lower triangular.

This property of the independence of the location of the optima produces an interesting property regarding the probability distribution used by the

algorithm for finding the next state to search. The probability distribution given by the values f_T can be permuted without affecting the average performance of the algorithm on this particular function (provided the value of f_0 remains fixed) as both the sum $\sum_{T=1}^{2^l} f_T$ remains constant and the multiplication of the matrix to some power n also produces the same terms as any other permutation of the values of f_T for $T = [1, \dots, 2^l]$. The actual values of f_T for $T = [1, \dots, 2^l]$ do however make a difference even if the sum $\sum_{T=1}^{2^l} f_T$ remains constant, as they affect the terms produced when the transition matrix is raised to some power n .

4.1 Calculation methods

For any row that does not correspond to starting in the optimum, we know that the probability of transition to the optimum is given by f_T for $T = [1, 2^l]$ and that all other non-diagonal elements are zero and that all rows must sum to 1, hence the diagonal element of the one step transition matrix has the value $P_{jj}(1) = 1 - f_T$. We have already determined that the value of the diagonal elements at time n can be calculated simply by $P_{jj}(n) = (p_{jj})^n$, first shown in equation (3.37). Under the highly simplifying conditions imposed by the needle in a haystack function, we can also devise a simplified method of calculating the average rate of convergence, as for each row whatever probability is lost from the diagonal element must be on the column representing the optimum. Therefore $P_{jk^*} = 1 - (1 - f_T)^n$ where j is an arbitrary row and k^* is the column corresponding to the optimum. This result can also be explicitly determined by multiplying the matrix. Let us sum the column k^* of the transition matrix after it has been raised to the power n and divide by the number of states to get probability of being in the optimum after n iterations averaged over all possible starting states

$$\bar{p}_n(k^*) = \frac{1 + \sum_{T=1}^{2^l-1} (1-f_T)^n}{2^l} \quad (4.1)$$

This is a numerically useful result, as it requires no matrix operations to be performed; it doesn't suffer the problems of scale to the same extent as most numerical results that can be obtained from Markov chain models.

Having established the worst case function and seeing that the performance on the worst case function has a dependence on the distribution $f(T)$, the obvious question is what distribution of $f(T)$ gives the best performance on this problem. For many classes of problems the method to tackle such a question is not apparent, but as the needle in a haystack function greatly simplifies the possible transit matrix, and by extension the transition matrix, the best distribution of $f(T)$ can be determined directly by examining (4.1) and noting the value f_0 is not included in the equation.

We wish to maximise the probability at a given number of iterations n , $\max(\bar{p}_n(k^*))$. In order to do this we need to maximise the summation in equation (4.1) as all other terms are fixed. Recall from equation (3.20) that the condition $\sum_{T=0}^{2^l-1} f_T = 1$ must be satisfied by the distribution of values of

$f(T)$. Hence $\sum_{T=1}^{2^l-1} f_T = 1 - f_0$ and we are free to choose $f_0 = 0$ if we wish.

The reason f_0 is not included in the summation in equation (4.1) is that it corresponds to a null mutation, the diagonal elements of the sample matrix are always f_0 , consequently the intersection of the column of the optima with the diagonal occurs where there would be an f_0 element in the sample matrix, but as transitions away from the optima are prohibited by the

possible transit matrix, the element at the intersection will contain the full sum $\sum_{T=0}^{2^l-1} f_T = 1$, that is why the value of f_0 can be chosen freely. So in order to maximise equation (4.1), the summation must be maximised $\max\left(\sum_{T=1}^{2^l-1} 1 - (1 - f_T)^n\right)$ which requires $\min\left(\sum_{T=1}^{2^l-1} (1 - f_T)^n\right)$. This minimisation occurs when the distribution of values of $f(T) \forall T > 0$ is uniform as a decrease in the value $f(T)$ for a particular value of T will result in an increase in one or more other values of $f(T)$. Hence a uniform distribution of probability or 'random search' (excluding the possibility of null mutations) is optimal when performance is averaged over every possible starting state.

4.2 The Lower Bound

This section will make the case for a needle in a haystack function being the hardest possible function to be optimised (performance averaged over all possible starting states) by an algorithm that uses an elitist mechanism. This could be true of optimisation algorithms in general, although we do not claim this to be the case, as the work presented in this section is specific to elitist mechanisms; it cannot provide a ubiquitous result for all optimisation algorithms.

By demonstrating that the needle in a haystack function is the hardest possible function for an elitist algorithm to optimise, we will be able to utilise the efficient method of calculating the exact performance of an elitist mechanism algorithm on a needle in a haystack function, to provide a worst possible performance bound for all unknown functions that have a search spaces of equal size.

Let us consider a set of input-output pairs that correspond to some function and search space grid. Let us, for the moment, restrict this set of input-output pairs to containing a single global optimum. Consider the transition

matrix that corresponds to the set of input-output pairs being optimised by a simple algorithm with an elitist search mechanism, for example Algorithm 3 examined in chapter 3. Let us also apply the standard condition that all mutation masks are non-zero. There is little that can be said about the transition matrix with the information we have so far. However, we can say with certainty what the sum of k^* column will contain, as there is a single optimum we know that the element located where the k^* column intersects the diagonal will have the value 1. Also, because there is a single optimum, we know that all states will be permitted to transit to the optimum by the possible transit matrix, hence:

$$\bar{p}_1(k^*) = \frac{1 + \sum_{T=1}^{2^l-1} f_T}{2^l}. \quad (4.2)$$

Without knowing more about the function we cannot determine $\bar{p}_n(k^*)$, nor can we directly determine the rate at which the elements in the column k^* accumulate probability mass. However we can make a limiting statement about the values of the elements on the diagonal. For some arbitrary row j , ($j \neq k^*$) we know $P_{jk^*} = f_T$ for some $T = [1, \dots, 2^l]$ hence the value on the diagonal of that row can be limited $P_{jj} \leq 1 - f_T$. In the case of the needle in a haystack function of course $P_{jj} = 1 - f_T \quad \forall j \neq k^*$. All other classes of function must have $P_{jj} < 1 - f_T$ for at least some value(s) of j .

Intuitively, a needle in a haystack function has one route from each possible starting state to the optimum this route allows a set amount of probability mass to flow from the starting state to the optimum at each iteration. Having one or more intermediate steps to the optima would allow for an additional flow of probability out of the state in which it starts and into the optimum (over some number of iterations). However, these additional routes do not have any effect on the one step route to the optimum. Hence

we can deduce any non-uniform affinity values in the search space can only increase the probability of the optimum being found for $2 \leq n$.

More formally: because the future values of any diagonal element depends only its own initial value $P_{jj}(n) = (p_{jj})^n$, as shown in section 3.5, we can deduce that the rate at which the diagonal elements $P_{jj \neq k^*}(n)$ fall to zero with n for any function that is not a needle in a haystack function, must be faster for at least some values of j ; and that for no values of j can it be slower than a needle in a haystack function. Consequently the rate at which the block of transient states falls to zero must be slower for needle in a haystack functions than for any other functions (with the same size search space). Hence the rate at which the elitist algorithm converges on any non-needle in a haystack function must be faster than the performance of the algorithm on a needle in a haystack function with the same size search space.

The performance of the algorithm on the needle in a haystack function can therefore provide a lower bound of performance for an unknown function based only on the number states in the search space grid. Hence the values of the lower bound are given by equation (4.1).

4.3 The Upper Bound

Although the lower bound of the rate of convergence came from a specific function, the bound that is produced is general for all functions with the same number of states in the search space. The upper bound by contrast is function specific. A new upper bound must be calculated for each function as the method of calculation incorporates information from the function, but instead of doing a full Markov chain calculation an approximation is made based on the methods uncovered by analysing the needle in a haystack function.

The upper bound is essentially an approximate method of calculating the rate of convergence of an elitist mechanism algorithm on any specific search space, it just so happens that it can be shown to give a greater or equal rate, to the rate of convergence that would be produced by doing a full Markov chain calculation.

The method of calculating the upper bound consists of obtaining the true values for the diagonal elements of the one step transition matrix $P_{jj}(1) = (p_{jj})^1 \quad \forall j$, the values of the diagonal elements at some number of iterations n , will be as accurate as the full Markov chain calculation as $P_{jj}(n) = (p_{jj})^n$. We can then simply declare that for each row, the probability mass that is not on the diagonal is in the column of the optimum, with all other entries being zero (in the same way as the needle in a haystack function).

As the diagonal elements are accurate $\forall n$ we know that these are the true maximum values that could be in the column of the optimum. For small values, e.g. $n < 10$, the assertion that the probability mass that is not on the diagonal is all in the column of the optimum is poor. However, the off diagonal elements are initially small compared with the diagonal elements and raising the transition matrix to modest values makes the assertion quite reasonable. The values of the upper bound are given by

$$\bar{p}_n(k^*) = \frac{1 + \sum_{j \neq k^*} 1 - (p_{jj})^n}{2^l} \quad (4.3)$$

The formulation of the performance bound that has been derived in equation (4.1) is in terms of the probability of finding the optima given a known number of iterations. Figure 6 Demonstrates that this formulation is equivalent to a bound stated in terms of the number of iterations required

to have located the optima with a known probability. Existing bounds for GAs by Aytug et al (Aytug et al., 1996, Aytug and Koehler, 1996, Aytug and Koehler, 2000) are given in the later form. In order to compare the bounds directly, it is useful to change the notation used in equation (4.1) into the notation commonly found in GA bounding:

$$\delta_1(t) = \frac{1 + \sum_{T=1}^{2^l-1} 1 - (1 - f_T)^t}{2^l} \quad (4.4)$$

Where $\bar{p}_n(k^*)$ has been replaced by $\delta_1(t)$ and n has been replaced by t .

In order to re-arrange equation (4.4) to have $t_1(\delta)$ on the LHS it is useful to apply some simplifying conditions. Lets assume that all values of f_T , $\forall T \neq 0$, are equal and that $f_0 = 0$, which implies $f_T = \frac{1}{2^l - 1}$, $\forall T \neq 0$. This is the optimal probability distribution for solving a needle in a haystack problem, as described in section 4.1, which allows the expansion of the summation for calculating $\delta_1(t)$

$$\delta_1(t) = \frac{1 + \sum_{T=1}^{2^l-1} 1 - (1 - f_T)^t}{2^l} = \frac{1}{2^l} + \frac{(2^l - 1)}{2^l} \left(1 - \left(1 - \frac{1}{2^l - 1} \right)^t \right). \quad (4.5)$$

For large values of l , $2^l \gg 1$ and hence $\frac{1}{2^l}$ is small and $\frac{(2^l - 1)}{2^l} \approx 1$. Hence we obtain the approximation:

$$\delta_1(t) \approx 1 - \left(1 - \frac{1}{2^l} \right)^t. \quad (4.6)$$

This approximation can be re-arranged to give

$$t_1(\delta) \approx \frac{\ln(1-\delta)}{\ln\left(1-\frac{1}{2^l}\right)} \quad (4.7)$$

This can now be directly compared with bounds found in the GA literature.

4.4 Comparison with current bounding results

There are essentially two attempts at producing a bound the initial work by Aytug et al (Aytug et al., 1996, Aytug and Koehler, 1996) and the improvement on this by Aytug et al (Aytug and Koehler, 2000) and the same result achieved independently by Greenhalgh et al (Greenhalgh and Marshall, 2001).

The first bound proposed by Aytug (Aytug and Koehler, 1996) in the notation it appears in (Greenhalgh and Marshall, 2001) is:

$$t_1(\delta) \approx -\frac{(K-1)^{\gamma n} \ln(1-\delta)}{\mu^{\gamma n}} \quad (4.8)$$

and the improved bound proposed by Greenhalgh (Greenhalgh and Marshall, 2001) is:

$$t_1(\delta) \approx -\frac{(K-1)^\gamma \ln(1-\delta)}{n\mu^\gamma} \quad (4.9)$$

Where K is the cardinality of the space, δ is the required probability of having found the optimum, $t_1(\delta)$ is the number of iterations required to have found the optimum with probability at least δ , γ is the length of the string and n is the size of the population.

The Greenhalgh formulation is for the probability of having seen all points in the search space with probability δ . Greenhalgh also provides the optimal value of the mutation rate to minimise $t_1(\delta)$ by setting $\mu = \frac{K-1}{K}$ which for binary cardinality is $\mu = \frac{1}{2}$ which corresponds to a random search and negates any effects from the crossover operator.

Greenhalgh (Greenhalgh and Marshall, 2001) also provides some numerical calculations of $t_1(\delta)$, in particular comparing how $t_1(\delta)$ varies according to the population size. The population sizes for which $t_1(\delta)$ is calculated are $n=15$ and $n=30$. Increasing the population size decreases the number of iterations required, which contrasts with the original bound obtained by Aytug (Aytug and Koehler, 1996) which increases significantly with the increase in population size, as it requires seeing all possible populations with probability δ , not just all points in the search space. Greenhalgh (Greenhalgh and Marshall, 2001) goes on to claim that the reduction in the $t_1(\delta)$ with increasing population size, demonstrates that larger populations are more efficient. While prima facie, this statement follows from the reduction in iteration number as a result of increasing population; if the number of iterations is converted into function evaluations by multiplying by the population, the number of function evaluations required by both population sizes $n=15$ and $n=30$ is exactly the same. The numerical results presented by Greenhalgh (Greenhalgh and Marshall, 2001) simply do not support the claim that an increase in population size increases efficiency.

Furthermore, it can be seen directly from equation (4.9) that the effect of the population in this simplified calculation of $t_1(\delta)$ is simply the granulation of the number of function calls that are made. Multiplying equation (4.9) through by n to get the number of function calls, $nt_1(\delta)$, on the LHS instead of iteration number

$$nt_1(\delta) \approx -\frac{(K-1)^\gamma \ln(1-\delta)}{\mu^\gamma} \quad (4.10)$$

We can see that the effect of the population size is purely granular: Setting $n=1$ gives the minimal integer number of function calls required. Hence, any value of n that is not a factor of this integer will cause an excess of up to $n-1$ function calls to be made. This result in fact supports a statement that a population size of $n=1$ to be optimal for all cases as it produces the minimal number of function calls, however it must be viewed in its proper context. Equation (4.9) has been optimised and simplified for this general bounding problem. Use of the optimal mutation rate $\mu = \frac{K-1}{K}$ negates the effect of the population based, crossover operator. It is not at all surprising therefore that a large population size is of no value in this particular case. However, general claims for or against the dependence of efficiency of the algorithm on population size, should not be made based on a formula describing a worst case bound.

Aytug's improved bound (Aytug and Koehler, 2000), achieved independently from Greenhalgh's result, is

$$\min \bar{t} = \frac{\ln(1-\alpha)}{n \ln(1-\mu^l)} \quad (4.11)$$

Where $\min \bar{t}$ Is the number of iterations required to have found the optimum with probability at least α , l is the length of the string, μ is the rate of mutation (uniform rate for each element in the string) and n is the size of the population.

Equation (4.11) is Aytug's improved bound (Aytug and Koehler, 2000), of the many formulae in the paper this is the most relevant for the purposes of comparison with the original work in this section as well as the result from Greenhalgh (Greenhalgh and Marshall, 2001). Equations (4.11) and (4.9)

differ in the denominator (given $K=2$ in equation (4.9)), but the denominators are related by the identity $\ln(1-x) \approx -x$ (valid for small values of x). It is not particularly surprising that these two formulae are so closely related as they both stem from Aytug's paper (Aytug and Koehler, 1996).

If we compare Aytug's bound given by equation (4.11) with the equation we derived here (4.7), both equations have been simplified by applying assumptions that minimise the number of iterations required given the constraints inherent in their respective models. Aytug finds the optimal value for the mutation rate is $\mu = \frac{1}{2}$, therefore the two equations differ only by the population term, as population has been omitted in the derivation of (4.7) where the LHS refers specifically to the number of function evaluations instead of iteration number.

Multiplying equation (4.11) through by n to put Aytug's LHS into function evaluations as well, we find the RHS of both equations are the same. This is somewhat surprising given the difference in the starting models and techniques used. However, as both bounds are optimised under simplifying conditions to give the minimal number of function calls, the differences between the algorithms for which the minimal bounds are calculated gets negated by the application of simplifying conditions. Both models essentially become random search, so it is reasonable that the same bound could be obtained from both of them.

We have produced an upper bound t_1 for the number of function calls needed in order to find the optimum with probability δ , based on a Markov chain model of the BCA. We have compared this upper bound with the tightest upper bounds found in the GA literature (Aytug and Koehler, 2000, Greenhalgh and Marshall, 2001) and have found that the results we have obtained are as good as the state of the art.

4.5 Chapter discussion

By investigating the mechanics of an approximate method of calculation of the rate of convergence given in section 4.1, we have developed a set of theoretical bounds given in sections 4.2 and 4.3. Comparing the bounds we have developed with the state of the art in the GA literature (Aytug and Koehler, 2000, Greenhalgh and Marshall, 2001) we have shown the bounds developed in this thesis to be equivalent.

This result is particularly interesting because the result obtained in this thesis is for an elitist AIS, whereas the results obtained by (Aytug and Koehler, 2000, Greenhalgh and Marshall, 2001) are for a non-elitist GA. The bounds derived in this thesis arise via a method that is very different from the methods employed in (Aytug and Koehler, 2000, Greenhalgh and Marshall, 2001). With all these differences in starting point and approach, it seems somewhat more than a coincidence that we arrive at an equivalent bound and indeed it is not a coincidence.

Using the results of this chapter we can determine the reason that the bounds developed here are equivalent to those of (Aytug and Koehler, 2000, Greenhalgh and Marshall, 2001); this will in turn lead to an important result about bounding the rate of convergence of optimisation algorithms in general.

4.5.1 The Isomorphism between bounding and plotting

We obtained our bounding result by demonstrating the needle in a haystack function is the hardest function for an elitist algorithm to optimise and subsequently demonstrating that the location of the optimum in the search space did not affect the rate of convergence. Given that the optimum could be anywhere in the search space, the probability of finding the optimum with some probability $0 \leq \delta \leq 1$ presents a problem that is isomorphic to

having searched $\delta|S|$ points in the search space of size $|S|$. If we set $\delta = 1$ then the probability of having found the optimum using an optimisation algorithm is equivalent to having completed a plot of the search space. Similarly, finding the optimum with some arbitrary probability δ is equivalent to having completed a plot of $\delta|S|$ points in the search space.

This equivalence between bounding and plotting can be shown by a very simple thought experiment: Consider a needle in a haystack function, a single optimum in an otherwise flat landscape. The location of the optimum is unknown, no information about the location of the optimum exists in the rest of the landscape. The probability of locating the optimum at any given point in the search space is $|S|^{-1}$, where $|S|$ is the size of the search space. Consequently, if we evaluate $\delta|S|$ points in the search space and each one has an independent probability $|S|^{-1}$ of being the optimum then the probability that we have found the optimum is $\delta = \delta|S| \cdot |S|^{-1}$. Hence the probability of having found the optimum with a probability δ is equivalent to having searched (or "plotted") $\delta|S|$ points in the search space.

Having established the equivalence relationship between bounding and plotting it is now easy to see why the bounding results obtained here are equivalent to the results obtained in (Aytug and Koehler, 2000, Greenhalgh and Marshall, 2001). Inspecting the basis of the bounding calculations we see that: (Aytug and Koehler, 1996) calculate the probability of every possible population of a GA having been seen and consequently the bounds are extremely loose for populations of more than one as there are combinatorially more populations than there are points in the search space. The results obtained in (Aytug and Koehler, 2000, Greenhalgh and Marshall, 2001) are based on "...all population members (not all populations) are seen at least once with a specified probability", quoted from (Greenhalgh and Marshall, 2001). This relates directly to the thought experiment above and hence relates directly to the calculations made in this thesis.

4.5.2 On the usefulness of bounding results in stochastic search

We have demonstrated in section 4.5.1 the probability of having found the optimum of an arbitrary function with some probability δ , is equivalent to having searched $\delta|S|$ points in the search space of size $|S|$. If we now consider the number of function calls required to have located the optimum, the lower limit of the upper bound $t_1(\delta)$ is given by

$$\delta|S| \leq t_1(\delta). \quad (4.12)$$

Consequently the best possible bound that is achievable under this set of assumptions is evaluating some fraction of the search space equal to the desired probability of having found the optimum δ . The stochastic search techniques considered in this thesis are not prohibited from evaluating points in the search space that they have already evaluated, which gives rise to the “or greater than” part of equation (4.12). Thus it has been demonstrated that bounding the probability of locating the optimum by stochastic search algorithms will never be useful for real world problems as the lower limit of the upper bound is given by plotting the required proportion of the search space; and in any real world problem, plotting any significant fraction of the search space is not a feasible proposition.

Equation (4.12) demonstrates that the current approach to bounding, while producing results that are both true and general, will not produce a *practical* result; a result that practitioners can make use of. This does not however mean that there will never be practical bounding results. The potential for practical bounding results exists, if some amount of generality is sacrificed. Some suggestions from (Aytug and Koehler, 2000) about where generality can be sacrificed are to consider finding the average time rather than a bound on the time required to obtain a specified confidence, which is an

interesting proposition, but it is not immediately apparent that times produced in this way will be sufficiently small to be of practical use.

The most promising suggestion made by (Greenhalgh and Marshall, 2001) was to give results for a restricted class of objective functions, the restriction being that the number of optima are known, (Greenhalgh and Marshall, 2001) go on to say that this information is not normally known. Results which are only valid for a restricted set of objective functions have the potential to give practical times, but of course for an unknown function it is not possible to say which restricted set of objective functions it belongs to.

4.6 Summary

This chapter describes the problem of bounding the rate of convergence of an optimisation algorithm on an unknown function. A set of upper and lower bounds are derived from Algorithm 3 (a simplified model of the BCA developed in chapter 3) in conjunction with the needle in a haystack function. The upper bound derived here is shown to be equivalent to the upper bound derived by (Aytug and Koehler, 2000) and achieved independently by (Greenhalgh and Marshall, 2001). An extensive comparison between the three sets of bounds uncovers why the bounds have all given the same result despite differences in the algorithm that is being bounded and the methods used.

Analysis of why all the bounding results have been shown to be equivalent has revealed that all the approaches are isomorphic to plotting a given fraction of the search space, the relationship between the problems is characterised by equation (4.12) which also gives the lower limit to the upper bound. The conclusion of this analysis is that in order to have a confidence of δ of having found the optimum of *an unknown function*, $\delta|S|$ points in the search space must be evaluated, where $|S|$ is the total number

of points in the search space. Put simply, the analysis reveals that if you wish to have a confidence of $\delta = 0.95$ of having found the optima of an unknown function, plot 95% of the search space.

The outcome of the chapter is that it appears that upper bounds for the rate of convergence of *an unknown function* are not of any real practical use. Bounds developed for particular function classes may give results of practical interest. Due to the stochastic nature of the algorithms and the fact that the algorithms allow "retracing" (evaluating a point in the search space that has previously been evaluated) the bounds that can be obtained require more function evaluations than simply plotting $\delta|S|$ points of the search space. In the case of values of δ that are close to 1, the number of function evaluations given by the bound is many times the number of points that are in the search space, so if the number of function evaluations required by the bound can actually be achieved, the search space could simply be plotted instead of using a stochastic optimisation algorithm.

Chapter 5: Modelling Framework

This chapter contributes to the thesis hypothesis by building on the modular modelling case study in chapter 3 and using it as a basis to derive the general framework for modular modelling of optimisation algorithms.

In chapter 3, a complete and exact model of the BCA was constructed under the assumption of a static objective function. Although the case study of the BCA used a modular modelling structure, it was focused on a single algorithm. We have seen the flexibility of the modular modelling style having produced several models of BCA sub-algorithms in sections 3.3, 3.4 and 3.7. In this chapter we shall examine how this modelling methodology can be expanded into a more general framework for constructing models of a wide variety of optimisation algorithms: Hill Climbing (section 6.1), Simulated Annealing (section 6.2), Evolutionary Algorithms (section 6.3) and Artificial Immune Systems (chapter 3). To begin with, we retain the assumption of a static objective function and expand each section of the modelling process to a more general level, by considering how a variety of operators can be incorporated into either the sample matrix or possible transit matrix. Then we shall go on to develop the framework to ensure it can include dynamic algorithms (on static objective functions), dynamic objective functions (optimised by static algorithms) and finally dynamic algorithms optimising dynamic objective functions.

Once we have considered operators from a wide variety of algorithms and solidified the framework, we will be able to give a practical guide, given in section 5.7, for how to use the framework to construct novel models of optimisation algorithms.

The chapter is organised as follows. In section 5.1 we lay out the general path on which the framework will be developed, namely the sample matrix and possible transit matrix approach to constructing the transition matrix.

In section 5.2 we begin to address the definition of the state of an algorithm. We identify three contributing factors that affect what the minimal definition of state of the algorithm is; these factors are: *population based operators*, *memory sets* and *dynamic operators*. In section 5.2.1 we address *population based operators*, where we make the distinction between algorithms with populations and algorithms with population based operators. In section 5.2.2 we consider *memory sets* which we argue are essentially passively stored and how they differ from elitist operators, where the dynamics of the search is actively affected by maintaining the best so far solution.

From these discussions, we generalise the sample matrix (for static algorithms) in section 5.3 and the possible transit matrix in section 5.4. In section 5.5 we present the general method to combine the sample matrix and possible transit matrix to form the transition matrix; the key result being equations (5.11) and (5.12), which establish the framework for static algorithms.

From this basis we consider dynamic algorithms and functions including *dynamic operators* extending the framework to inhomogeneous Markov chains section 5.6. We then give a practical guide to modelling algorithms using the framework in section 5.7 where we present eight classes of algorithm and their minimal definitions in Table 11. The chapter is summarised in section 5.8. The framework and practical guide developed in this chapter will be applied in chapter 6 to produce a novel model of the $(1+\lambda)$ Evolutionary Strategy in section 6.3 and two other models in order to demonstrate the utility of the framework.

5.1 Method of construction

We shall continue with the two-matrix approach to constructing a transition matrix, first developed in 3.4. The two-matrix approach is far more

transparent than a direct mathematical formulation, keeping the operators explicitly separable, making it far more flexible. The two matrices have been named the “sample matrix” and the “possible transit matrix”, see sections 3.4.1 and 3.4.2 respectively.

- The sample matrix gives the probability distributions of selecting the next point(s) in the search space to be evaluated given a current state of the algorithm.
- The possible transit matrix determines how each evaluation of a new point in the search space will change the state of the algorithm.

When these two matrices can be determined, they can be combined to form the transition matrix for the algorithm. We will consider operators used in simulated annealing, hill climbing, evolutionary algorithms, artificial immune systems, while the modelling framework is developed, but there may be algorithms with operators that are outside this broad search structure that may not be included by the framework.

5.2 Defining the state of an algorithm

The importance of the definition of the state of an algorithm is highlighted in chapter 2 where we discussed in detail the various definitions that appear in the literature, in Markov chain models (Nix and Vose, 1992, Häggström, 2002) and in other formal descriptions of algorithms such as in No Free Lunch theorem (Schumacher et al., 2001). It was concluded in section 2.5.1 that it is important to keep the definition of the state of an algorithm minimal, which means that for each algorithm to be modelled one must determine what is the minimal definition of the state of that algorithm.

As in the Markov chain models that appear in the literature (Nix and Vose, 1992, Häggström, 2002), the state of an algorithm is defined a set of one or more points in the search space and the value of the algorithms internal

parameters, such that the probabilities of any potential set of points in the search space occurring in the next iteration can be unambiguously defined.

In this section we shall examine some of the factors that determine the minimal definition of a state of an algorithm. The different properties of the algorithms being modelled allow for different criteria to be excluded from the definition of a state of the algorithm. We find that the definition of the state of the algorithm depends on whether or not the algorithm uses *population based operators* (section 5.2.1) – any operator that requires global knowledge, if the algorithm has a separate *memory set* (section 5.2.2) and whether or not the algorithm uses *dynamic operators* (section 5.6). As the minimal definition of the state of an algorithm depends on three factors, each of which can either be included or not included in an algorithm we will define $2^3 = 8$ classes of algorithm. These distinctions will be made based on the properties of the algorithms, not the origins of the algorithms.

5.2.1 Algorithms with populations and algorithms with population based operators

In section 3.1.1 we noted that one of the properties of the BCA (defined in Algorithm 1 1) is that the members of the population do not interact in any way; members of the population are independent. That is to say that the probability of a given member of the population moving from its current position in the search space to its next position in the search space, is unaffected by the position (past, present or future) of any other member of the population.

There is an intuitive example that illustrates how members of a population of algorithms can be treated separately as long as no population based operators are used. In this example we consider a hill climber algorithm, see Algorithm 5 in section 6.1:

In the first case we consider a single hill climber, we initialise the hill climber and run it for a fixed number of iterations. We do $|P|$ independent runs of this experiment.

In the second case we consider a population of P independent hill climbers as a single algorithm. We initialise each member of the population and run the algorithm for a fixed number of iterations.

As the hill climber is a simple deterministic algorithm and we state that the hill climbers are independent, it is clear that the members of the population do not affect each other in any way. Therefore the overall probability of finding the global optima of the objective function in these two situations is indistinguishable. This example demonstrates that while the inclusion of a population in an algorithm's definition enables the potential to use population based operators, the inclusion of a population in an algorithm's definition does not enforce the use of population based operators.

In the same way, all algorithms that have populations without population based operators, can be modelled as an algorithm with a population of one and the result of that model scaled up to $|P|$ independent runs using the basic probability theory for independent events, see (Häggström, 2002) chapter 1.

- If an algorithm uses no population based operators, the state of the algorithm can be represented by a single point on the search space of the function. Each member of the "population" is *independent* of the rest of the "population" can be disregarded as it does not affect the dynamics of the algorithm.
- If an algorithm makes use of population based operators, operators that utilise knowledge of the population as a whole, such as ranking or relative fitness, then the states of the algorithm must be the contents of the population. Each member of the population is *dependent* on the

fitness of the other members of the population. The members of the population interact in some way that affects the dynamics of the algorithm.

In a population based algorithm, the minimal definition of the algorithm must include the content of the population. An algorithm that has a “population”, but no population operators, can be treated as $|P|$ independent algorithms, each of which have a minimal definition of the state of the algorithm of a single point in the search space.

In this section we have highlighted the difference to the minimal definition of a state of the algorithm between: algorithms with population based operators and algorithms that merely have a population. We have not however made a case for there being no practical value in having algorithms that have populations, but no population based operators. By considering the curvature of the rates of convergence in Figure 5, we see that the steepest part of the graphs is in the early iterations and the least difference between the best performing and worst performing mutation rates. We suggest that there may be an advantage to be gained from $|P|$ independent runs (an algorithm with a population, but no population based operators) compared to other stochastic algorithms using the same number of function calls, although we do not investigate this conjecture in this thesis.

5.2.2 Memory Sets and elitist operators

In this section we discuss the difference between: a “memory set” which preserves the best so far solution in a separate set, and “elitist selection” which preserves the best so far solution in the current population via an elitist selection operator. We consider how these operators affect the dynamics of the algorithm they are in and what conditions they impose on the minimal definition of a state of an algorithm.

An algorithm with a memory set stores the best so far solution, but the dynamics of the algorithm's search are *independent* of what is stored in the memory set. An elitist selection operator maintains the best solution in the current population and so the dynamics of the algorithm are in some way *dependent* on the best so far solution. These two situations are distinct; this must be accounted for in an exact model of an algorithm.

In terms of defining the state of an algorithm and consequently the number of states in the Markov chain, the difference between these operators is vast.

Consider a simple algorithm that does not maintain the best so far solution, such as Algorithm 1 2. If we now modify this algorithm to have an elitist selection operator we obtain Algorithm 3, we have shown this modification causes the state of the algorithm to be defined by a single point in the search space in chapter 3. The number of states of the algorithm is given by $S = 2^l$.

Let us now consider Algorithm 2 and modify it with a memory set. We define that the memory set stores one point in the search space and that the memory set must have the potential to store any point in the search space. The individual (population of one) of the algorithm must be able to search the space independent of what is stored in the memory set. For each possible point stored in the memory set, the individual can be in any point in the search space. The state of the algorithm requires knowledge of memory set as well as the individual on which determines the probability distribution of searching the other points in the search space. As both the individual and memory set can both be in any position in the search space, the number of states of the algorithm is given by $S = 2^{2l}$, which is significantly larger than in the case with just an elitist operator.

5.3 The Generalised Sample Matrix

In chapter 3 we had a single mutation operator and were working with an algorithm with a minimal state space. In order to generalise this approach we need to be sure that the definition of the sample matrix is appropriate. Working with just the Contiguous Region Hypermutation Operator (CRHO) in chapter 3 we defined a sample matrix to be: A matrix containing the probability distribution from each possible state to all other states. We require that each row of the sample matrix sums to one. At this point we drop the condition that the sample matrix is independent of the objective function.

In the case of the CRHO, the probability distribution could be derived from analysis of the binary operator, see section 3.3. Thinking more generally, the probability distribution could be given by any kind of operator provided the distribution sums to 1. The probability distribution could even be defined explicitly for each state-to-state transition. Indeed, when the states and operator are both known we can think of the sample matrix as the explicitly defined state-to-state probability distribution (provided all the values of operator variables are also known).

Given the lack of constraining factors on the ways in which the probability distribution from one state to the others can be defined, it would not be possible to give a completely general set of guidelines for the modelling of mutation operators. The remainder of this section consists of a selection of examples of models of mutation operators that have special characteristics or points of interest.

5.3.1 Special case – random search

Random search is frequently found to be at least as good as other algorithms in theoretical explorations of optimisation algorithms, this can be seen in No Free Lunch Theorem (Wolpert and Macready, 1997) and

bounding results: (Aytug and Koehler, 2000) and in section 4.1. Random search is a uniform probability distribution, each state having an equal probability of being sampled. The distribution can be stated generically: If there are in total $|S|=2^l$ possible target strings to mutate to, each of the possible target strings will have a probability of being sampled given by

$$f_T = \frac{1}{2^l} \quad \forall T. \quad (5.1)$$

Having a uniform distribution means that the probability of finding the optima in one step is independent of the current state of the algorithm, this property is the key to why the operator is often found to be optimal in general theoretical investigations where nothing is known about the space. While random search gains no benefit from starting close to the optima, there is no starting point from where it performs especially poorly.

Proposition 3

The probability of obtaining the optimum k^* using an elitist random search at some arbitrary number of iterations t is independent of the state of the algorithm X_t at that time: $p(X_{t+1} = k^* | X_t = j) = p(X_{t+1} = k^* | X_t = j')$.

Proof:

By definition: $p(X_{t+1} = k^* | X_t = j) = f_T$, where $T = k^* \oplus j$ and

$$p(X_{t+1} = k^* | X_t = j') = f_{T'}, \text{ where } T' = k^* \oplus j'. \text{ For random search } f_T = \frac{1}{2^l} \quad \forall T,$$

therefore $f_T = f_{T'}$ and $p(X_{t+1} = k^* | X_t = j) = p(X_{t+1} = k^* | X_t = j')$ \square

The independence of the current position of the elitist algorithm leads to the following conjecture: An elitist algorithm employing a random search strategy will have exactly the same performance as a non-elitist algorithm (with an elitist memory set) that is also employing a random search

strategy on the same objective function. We do not go on to present a proof of this conjecture here, although we propose that it can be shown to be true using models constructed from the generalised framework.

5.3.2 Special case – composite operators

Let us define a “composite operator” to be two or more methods of mutation that are applied to the members of a population, with some fixed probability distribution over the choice of operator. The most obvious approach to including such operators in the framework is to separately model each of the operators and then combine the probability distributions of the operators according to how many members of the population each operator is applied to.

Let us define the final probability distribution to be f_T and let the probability distributions of all contributing mutation operators be numbered sequentially $f_{1,T}, f_{2,T} \dots$. If we consider the specific case where two operators are combined to form a composite operator we can give the distribution of the composite operator f_T to be the weighted sum of the two operators

$$f_T = \frac{P_1 f_{1,T} + P_2 f_{2,T}}{P_1 + P_2} \quad (5.2)$$

where P_1 and P_2 are the number of members of the population P mutated by the operators associated with the distributions $f_{1,T}$ and $f_{2,T}$ respectively. We impose the condition that P_1 and P_2 account for all members of the population $P = P_1 \cup P_2$ for completeness.

This formulation can be generalised to cover k operators, where k is a positive integer.

$$f_T = \frac{\sum_k P_k f_{k,T}}{\sum_k P_k} \quad (5.3)$$

5.3.3 Special case – forbidding null mutations

The elements of the sample matrix that lie on the diagonal are special as they give the probability of a null sample: the probability that the operator will return the current state as the next state to be sampled. While having a non-zero probability on the diagonal is wasteful (as it does not search a new point), the vast majority of algorithm operators used to select the next state to be searched will produce non-zero probabilities on the diagonal elements of the sample matrix. Algorithms that are used on real, computationally expensive problems should re-distribute the probability from the diagonal entries in some manner. The simplest way to implement this is with a line of code that checks to see that the state to be searched that has been produced by the operator is not the same as the current state and if it is the same that the operator should not be applied again until a state other than the current state occurs. We can take this simple method into account and similarly adjust the elements of the sample matrix to compensate.

Consider a transition matrix with elements p_{ij} $i, j \in \{0, 1, \dots, 2^l - 1\}$, by definition the matrix must also be stochastic. For the purpose of generality let us assume that not all the diagonal elements have the same value. In this case, under this assumption we require a row by row adjustment. The required adjustment is given by

$$p_{ij}^* = \frac{1}{(1 - p_{ii})} p_{ij}, \quad \forall j \neq i \quad (5.4)$$

Where p_{ij} is the value of the element before the adjustment, p_{ij}^* is the value of the element after the adjustment and p_{ii} is the diagonal element. We can check that the adjusted element values on this will always sum to one provided the initial values also satisfied this condition (which they do by definition, because the matrix is stochastic). Let us use the condition to make the replacement $1 - p_{ii} = \sum_{j \neq i} p_{ij}$ in equation (5.4) and sum all values of p_{ij}^*

$$p_{ii}^* + \sum_{j \neq i} p_{ij}^* = p_{ii}^* + \frac{1}{\left(\sum_{j \neq i} p_{ij} \right)} \sum_{j \neq i} p_{ij} = p_{ii}^* + 1 \quad (5.5)$$

$p_{ii}^* = 0$ by definition, hence the summation condition will be satisfied.

5.3.4 Special case - The clonal matrix

The clonal selection operator is described in section 2.4. In section 3.7 we demonstrated a method for the construction of a matrix that represented the clonal selection operator, which we named the clonal matrix. The clonal matrix also represents a special case of the sample matrix in as much as within the context of the framework to construct a transition matrix it can be treated in the same way. A formal method of construction of the clonal matrix has been given in section 3.7.3. The distinct difference between a sample matrix and a clonal matrix is that a sample matrix describes the probability of making a single function evaluation, whereas the clonal matrix accounts for $|C|$ function evaluations where $|C|$ is the size of the clonal pool. The clonal matrix encapsulates multiple function evaluations, so the resulting transition matrix is not a one step transition matrix in terms of function evaluations.

5.4 General Possible Transit Matrix

The possible transit matrix is dependent on the objective function and the “selection” system used by the algorithm. In this section we shall detail the most common of these selection systems and how each of them affect the possible transit matrix. In general, the elements of the possible transit matrix are determined by some selection operator, which we shall denote as G , operating on the values returned by the objective function acting on the two states being determined. Let us denote the general possible transit matrix by M and two particular states of the algorithm s_i, s_j . Then

$$M_{ij} = G(g(s_i), g(s_j)) \quad (5.6)$$

The values $g(s_i), g(s_j)$ are independent of the search mechanism, depending only on the objective function (and sample space), so it is the operator G , that we are interested in defining for various algorithms.

5.4.1 Indiscriminate search

Algorithms such as Algorithm 2 rely on the probability distribution of the sample matrix alone; in this case, *the sample matrix is the transition matrix*. The operator G is always implicit in this case as $M_{ij} = 1, \forall ij$ there is no point in actually applying an operator:

$$G \begin{cases} g(s_i) > g(s_j), M_{ij} = 1 \\ g(s_i) = g(s_j), M_{ij} = 1 \\ g(s_i) < g(s_j), M_{ij} = 1 \end{cases} \quad (5.7)$$

The sectioning used in equation (5.7) seems somewhat arbitrary as the probability assigned to all three regions is the same. This is an example of a

“non-elitist” operator as the algorithm accepts all sampled points in the search space as state changes. Algorithms that have this sort of possible transit matrix take the form of an irreducible chain. Although this is the least interesting operator from an algorithmic perspective, irreducible chains have a vast body of mathematical literature, as they are the most interesting type of Markov chain viewed from a mathematical perspective. Consequently, there is a library of analytical techniques available to analyse these models (Seneta, 1981, Grimmett and Stirzaker, 1982).

5.4.2 Elitism

Algorithm 3, modelled in section 3.4, is an example of an elitist algorithm that uses an elitist method of determining the probability of transit to a state proposed by sampling a new point in the search space. The rules for the possible transit matrix are described in section 3.4.2. We can define the elitist operator:

$$G \begin{cases} g(s_i) > g(s_j), M_{ij} = 0 \\ g(s_i) = g(s_j), M_{ij} = 0 \\ g(s_i) < g(s_j), M_{ij} = 1 \end{cases} \quad (5.8)$$

This is an elitist selection operator (assuming maximising affinity). As we have seen in chapter 3, this type of operator rejects some state-to-state moves and this probability must be accounted for. This matter of accounting does not affect the possible transit matrix. We will deal with this issue fully when we discuss the general transition matrix in section 5.5.

5.4.3 Simulated Annealing

Simulated Annealing (SA) is introduced and described in section 2.2. The state operator used in SA differs from the others we have seen so far, in as much that it does not use fixed probability values for every outcome. The

normal SA terminology is in terms of energy of a state $E(s)$, instead of affinity of a state, $g(s)$. In general we consider algorithms that attempt to increase affinity; in SA this is synonymous with decreasing energy. SA also has a user defined parameter temperature, T , which is used in calculating the probability that a move to a “worse” (lower affinity, higher energy) state is allowed. This probability depends explicitly on the values of the states in question $g(s_i), g(s_j)$.

In terms of affinity we can define the operator:

$$G \begin{cases} g(s_i) > g(s_j), M_{ij} = e^{-\left(\frac{g(s_i)-g(s_j)}{T}\right)} \\ g(s_i) = g(s_j), M_{ij} = 1 \\ g(s_i) < g(s_j), M_{ij} = 1 \end{cases} \quad (5.9)$$

This is an example of a “partial elitist” operator as transitions are allowed but with a probability $0 < e^{-\left(\frac{g(s_i)-g(s_j)}{T}\right)} < 1$. This is a special case of partial elitism as the probability of a transition to a state of lower affinity is *variable*, depending on the affinity values of the two states, as opposed to a constant numerical value.

Defining the same operator in terms of energy (minimising the energy):

$$G \begin{cases} E_i > E_j, M_{ij} = 1 \\ E_i = E_j, M_{ij} = 1 \\ E_i < E_j, M_{ij} = e^{-\left(\frac{E_j-E_i}{T}\right)} \end{cases} \quad (5.10)$$

The parameter T , has its value altered during the running of the algorithm according to an annealing schedule, which causes models of the algorithm to take the form of an inhomogeneous Markov chain as discussed in section

2.2.1. We shall return to SA when we look at dynamic algorithms in section 5.6.

5.5 General Transition Matrix

In order to have a general transition matrix we need to formally define a method of combining the information from the sample matrix with the possible transit matrix. This method must correctly account for any probability mass that corresponds to sample states that have been *rejected* with some probability $0 < p \leq 1$ by the possible transit matrix. We can succinctly define the construction of the general transition matrix as:

$$T_{ij} = S_{ij}M_{ij}, \quad \forall i \neq j, \quad (5.11)$$

$$T_{jj} = M_{jj}S_{jj} + \sum_{i=0}^{2^l-1} (1 - M_{ij})S_{ij}, \quad (5.12)$$

where T is the transition matrix, S is the sample matrix and M is the possible transit matrix, i is the row index and j is the column index.

Equations (5.11) and (5.12) are the key to the framework, understanding them is critical to understanding the framework. Equation (5.11) means that for all non-diagonal elements the value of the transition matrix element is the multiple of the corresponding elements in the sample and possible transit matrices. Equation (5.12) means that the diagonal elements contain all the probability that is not accounted for by the other elements on the same row (as each row must sum to one in a transition matrix).

Upon examining equation (5.12) it can be seen why it is difficult to define a general transition matrix with a single equation, as *the diagonal elements are potentially dependent on all the information in the row*. In the simplest

case where all transitions are allowed ($M_{ij} = 1 \quad \forall i, j$) equations (5.11) and (5.12) reduce to $T = S$. This simplest case occurs in models of non-elitist algorithms such as the non-elitist model of MISA (Villalobos-Arias et al., 2004) and in the model of Algorithm 2 modelled in section 3.3 which both take the form of an irreducible chain.

Having established a general method of Markov chain construction, we can consider what classes of Markov chain can be produced under this framework and the associated long term behaviour of each class of the Markov chain.

5.6 Dynamic algorithms and landscapes

Up until now we have been dealing exclusively with static algorithms optimising static functions. Based on this premise, the framework already covers many optimisation algorithms. A class of algorithms that has not yet been included in the framework are dynamic algorithms. The simplest change that can be made to a static algorithm to cause it to become a dynamic algorithm is to change one of the algorithm parameters as some function of the iteration number. For example, making the mutation rate a function of the best so far solution would make the algorithm dynamic. Therefore the inclusion of dynamic algorithms into the framework is an important step towards making a useful contribution to the community.

Let us now consider how to describe a dynamic algorithm in terms of the framework. Let us consider an algorithm for this thought experiment: The algorithm evaluates a single potential solution (binary string) per iteration, the algorithm mutates each element in the string with some constant probability (we shall make the value vary from one iteration to the next to make the algorithm dynamic). The algorithm then evaluates the objective function and then applies elitism to the result to determine if the mutation should be accepted or rejected. In order to keep things as simple as

possible let us define that the algorithm will run for a fixed number of iterations and that there is a schedule of changes to mutation rate during this time.

Thinking about how the algorithm will progress: The initial scheduled mutation rate is defined, so we can calculate a sample matrix S_1 for the algorithm. We have stated that the algorithm is elitist and we assume knowledge of the function, so the possible transit matrix P_1 can also be calculated. So we have all the parts we need to produce a one step transition matrix T_1 using the equations (5.11) and (5.12) from the framework. The algorithm will progress just as a static algorithm would up until the first scheduled change in the mutation rate, which will occur after n_1 iterations.

The mutation rate is changed after the algorithm has been running for n_1 iterations at a fixed value. We know the current state of the Markov chain, which could be described in a number of ways, but let us describe it by the n_1 step transition matrix given by $T_1^{n_1}$. We now require a method of making the next step. This can be achieved by multiplying $T_1^{n_1}$ by a new transition matrix that is the correct one step transition matrix for the phase two of the scheduled change in the mutation rates. Let us call this transition matrix T_2 .

It is important to note that T_2 is not dependent on T_1 or the value n_1 . Fortunately the construction of T_2 is relatively simple: The possible transit matrix from the construction of T_1 is still the same, as the function is static and the elitism operator remains the same. Therefore we set $M_2 = M_1$, but the sample matrix S_2 must be calculated explicitly. However, as only the mutation rate has been changed the formula describing the distribution as a function of the mutation rate is already known from the calculation of the values of S_1 . The recalculation of the sample matrix is entirely numerical,

requiring no additional theoretical work. The one step transition matrix T_2 is then constructed from S_2 and M_2 as above.

Having expressed the current state of the algorithm as an n_1 step transition matrix given by $T_1^{n_1}$ we can now describe an $n_1 + 1$ step transition matrix as $T_2 T_1^{n_1}$. The $n_1 + 2$ step transition matrix (assuming $n_2 \geq 2$) would then be given by $T_2^2 T_1^{n_1}$. We can now describe a dynamic optimisation algorithm as an inhomogeneous Markov chain. At this point we introduce the notation

$$T_k^{n_k} \dots T_2^{n_2} T_1^{n_1} = T_{k \dots 1}^{n_k \dots n_1} \quad (5.13)$$

Where k is an integer. This notation is based on similar notation used in (Häggström, 2002) quoted in section 2.2.1.

In this notation, $T_{k \dots 1}^{n_k \dots n_1}$ does not denote a one step transition matrix that has been raised to a power, it is simply a shorthand for the process. In fact, given that an alteration of the mutation rate can cause a significant change in the behaviour of the algorithm, for example see Figure 5, the idea that a single one step transition matrix could completely represent a dynamic algorithm for a large finite number of iterations is, while possible in principle (see section 2.1.3), highly impractical.

Having highly constrained the initial example, we can see from the resulting method described by equation (5.13) that many of these constraints can be easily lifted. When only the mutation rate was being changed, only the sample matrix required recalculation in order to produce the new transition matrix. However, it is just as valid to change the possible transit matrix, as a result of either the function being changed (dynamic function) or the state operator changing.

Dynamic function landscapes will require recalculation of the possible transit matrix whenever the function changes. If the algorithm is static then the

sample matrix is independent of the function landscape and will not need to be recalculated.

More generally dynamic algorithms applied to optimising dynamic function landscapes will require recalculation of both the sample matrix and the possible transit matrix whenever either the algorithm or function change.

While the sample matrix can only be affected by the function landscape if the mutation operator is a function of the landscape, the possible transit matrix can be affected by a dynamic algorithm even on a static function. One example of this is simulated annealing, see section 6.2, where a scheduled change in the algorithm variable representing temperature causes the probability of accepting a change of state that negatively affects the affinity (increases the energy) to vary. The probability of allowing a transition is accounted for in the possible transit matrix, hence a change in temperature is reflected by a recalculation of the possible transit matrix.

5.7 A practical guide to modelling with the framework

This is a practical guide for using the framework developed in this chapter. It is intended for use as a modelling checklist and as such covers the key points of the framework, but assumes the reader is familiar with sections 5.1-5.6.

There are four tasks to be completed in order to model an optimisation algorithm using the framework

- Determining the minimal definition of a state of the algorithm
- Given a starting state, determining the probability that the algorithm will sample each of the possible states; model the search operator(s)
- Given a state has been sampled, determining the probability that the algorithm will update its state to the sampled state; model the state operator(s)

- Combining the above into a transition matrix

5.7.1 Definition of a state of the algorithm

In general, the state of an algorithm is defined as a set of points in the search space and the value of the algorithm’s internal parameters, such that the probabilities of any potential set of points in the search space occurring in the next iteration can be unambiguously defined.

In order to minimise the size of the transition matrix it is prudent to tailor the definition of a state of the algorithm to include the very minimum:

Dynamic (run time) Algorithm	No	No	No	No	Yes	Yes	Yes	Yes
Population Based Operator(s)	No	Yes	No	Yes	No	Yes	No	Yes
Memory Set	No	No	Yes	Yes	No	No	Yes	Yes
	1	2	3	4	5	6	7	8

Table 11. Minimal state of the algorithm guide

Use Table 11 in conjunction with the key given here to determine what needs to be included in the minimal definition of the state of the algorithm being modelled. The key for Table 11:

1. A single member population (one point in the search space)
2. The contents of the population
3. A single member population and the contents of the memory set
4. The contents of the population and the contents of the memory set
5. The operators, internal parameters and a single member population
6. The operators, internal parameters and the contents of the population
7. The operators, internal parameters, a single member population and contents of the memory set

8. The operators, internal parameters, contents of the population and contents of the memory set.

These definitions are based on material presented in sections 5.2.1, 5.2.2 and 5.6.

5.7.2 Constructing the sample matrix

The sample matrix: Calculate an exact probability distribution of how the algorithm chooses which points in the search space are to be sampled in the current iteration given any current state of the algorithm.

Based on the current state of the algorithm (having determined what a "state" of the algorithm is), decide which point(s) in the search space will be searched next. This will take the form of a probability distribution over the points in the search space. Once this probability distribution can be determined for all possible starting states, this information is used to construct the "sample matrix". This can be thought of as the row number defining a starting state and the column number defining an end state (after one iteration). Each complete row defines the probability distribution from a given starting state to all possible end states. For an example see section 3.4.3.

This probability distribution can, for algorithms with simple mutation operators, be determined simply in terms of probabilities of mutation masks occurring, by modelling the mutation operator. In algorithms which make use of population based operators, it could require the modelling of several different operators i.e. in a simple GA you would need to model both the mutation operator and the crossover operator and determine an overall probability distribution, see (Nix and Vose, 1992).

5.7.3 Constructing the possible transit matrix

The possible transit matrix: For any given current state of the algorithm, calculate the probability that a newly sampled state will result in a change of state of the algorithm.

The possible transit matrix is based on comparison of the affinity $g(s)$ of the "sampled state" s_j with that of the initial state s_i . The comparison has three natural mathematical regions. The affinity of the sampled state is less than the affinity of the initial state $g(s_j) < g(s_i)$, equal to the initial state $g(s_j) = g(s_i)$ or greater than the initial state $g(s_j) > g(s_i)$. The rules for these three subdivisions must be determined. The rule can then be applied for any proposed state to state transition, which is all that is required from the possible transit matrix. The rules for a change of state of an algorithm falls into three categories: accept the sampled state with probability one $M_{ij} = 1$, accept the sampled state with some probability $0 < M_{ij} < 1$, determined by some less trivial function of the affinity values of the two states $M_{ij} = G(g(s_i), g(s_j))$, or reject the sampled state (accept with probability zero) $M_{ij} = 0$.

$$G \begin{cases} g(s_i) > g(s_j), M_{ij} = ? \\ g(s_i) = g(s_j), M_{ij} = ? \\ g(s_i) < g(s_j), M_{ij} = ? \end{cases}$$

Each subdivision may have a different rule, or they may all have the same rule. Typically the case of equal affinities is given the same rule as one of the other rules, as in practical terms it makes little difference. However, it can be of theoretical importance as it can affect the type of Markov chain that is produced. See section 5.4 for details and examples.

5.7.4 Constructing the transition matrix

In this section we shall describe how to combine the sample matrix and possible transit matrix to form a standard Markov chain transition matrix. Essentially all non-diagonal elements of the transition matrix are the product of the corresponding elements in the sample matrix and possible transit matrix. The diagonal elements contain whatever probability is required to satisfy the condition that the sum of each row is 1. More formally combine the sample matrix and possible transit matrix using equations (5.11) and (5.12) given in section 5.5.

5.8 Summary

In this chapter we expanded on the case study of the BCA constructed in chapter 3 and generalised from the modular models of the BCA to a framework for modelling stochastic optimisation algorithms with Markov chains. This has been achieved by considering the differences between the models in chapter 3, given in sections 3.3, 3.4 and 3.7; as well as considering operators from EA, SA and AIS while formulating the framework.

We have demonstrated that optimisation algorithms can conceptually be split into two parts: search operators and state operators. Search operators are represented by the sample matrix described in section 5.3. State operators are represented by the possible transit matrix described in section 5.4. These matrices can be combined by equations (5.11) and (5.12) to form the transition matrix of a Markov chain model of an algorithm. These equations allow the construction of the transition matrix from the sample matrix and possible transit matrix; they are the key to the framework:

$$T_{ij} = S_{ij}M_{ij}, \quad \forall i \neq j \quad (5.11)$$

$$T_{jj} = M_{jj}S_{jj} + \sum_{i=0}^{2^l-1} (1 - M_{ij})S_{ij} \quad (5.12)$$

Where T is the transition matrix, S is the sample matrix and M is possible transit matrix, i is the row index and j is the column index.

Equation (5.12) gives the diagonal elements of the transition matrix, from the form of equation (5.12) one can see that the diagonal elements potentially depend on the value of every other element in the row, which could make obtaining a single formula of the form " $T_{ij} =$ " extremely awkward, depending on the forms of S and M . Using the framework to avoid this problem greatly simplifies the construction of exact models of elitist algorithms.

Another advantage of the framework is that the sample matrix and possible transit matrix can be obtained independently from each other. As a consequence, one can modify a model constructed in this manner to reflect modifications to an algorithm, such as a change to a mutation operator.

In section 5.7 we gave a practical guide for modelling with the framework which can be viewed as a summary of the technical aspects of the chapter and is the quick reference for applying the framework to construction of novel models of stochastic optimisation algorithms. In section 5.7, the results of the discussions of the minimal definition of a state of the algorithm that occur in sections 2.5.1, 5.2 and 5.6 culminate in Table 11 and the associated key in section 5.7.1. We define 8 categories of algorithm, based on the inclusion of:

- population based operators
- a memory set
- dynamic operators

Each of the 2^3 algorithm categories is given a minimal definition of the state of an algorithm. We do not claim that this list covers all algorithms, but it is a useful guide for considering what needs to be included in the definition of the state of an algorithm when constructing novel models. We find in the literature only three of the categories given here: Exact models of SA (Häggström, 2002) take the form of an inhomogeneous chain, they are dynamic, but do not contain population based operators, or memory sets. Exact models of a simple GA (Nix and Vose, 1992) have population based operators, no memory set or dynamic operators. There are also models which do not include any of the three classes of operator listed above (Villalobos-Arias et al., 2004).

In chapter 6 we will apply the framework that has been developed in this chapter to demonstrate how existing models in the literature can be incorporated by the framework and how the framework can be used to construct novel models of existing algorithms.

Chapter 6: Applying the framework

This chapter contributes to the thesis hypothesis by demonstrating how to apply the framework developed in chapter 5 through a series of examples. The framework is applied to model several algorithms of varying degrees of complexity and from a range of computational fields. The algorithms modelled in this chapter are: hill climber, simulated annealing (SA) and the $(1+\lambda)$ Evolutionary Strategy (ES). In each case we present pseudo code for the algorithm, construct the model using the framework and produce an example transition matrix for a toy size problem.

The algorithms that are modelled in this chapter have been selected to demonstrate the range of algorithms that can be modelled by the general modelling framework and also to show how the framework can incorporate pre-existing Markov chain models. The model of SA is adapted from a pre-existing model already available in the literature (Häggström, 2002); this model is re-formulated in terms of the modelling framework. The hill climber is essentially a deterministic optimisation algorithm (apart from initialisation); it was selected to be modelled to demonstrate that the framework can incorporate algorithms from a wide variety of research areas. The only truly novel model in this chapter is of the $(1+\lambda)$ ES. The model of $(1+\lambda)$ ES demonstrates that with the aid of the modelling framework, novel models of optimisation algorithms can be constructed relatively easily.

In the context of the framework a model of an algorithm requires no more than the definition of the sample matrix and possible transit matrix. However, in order to make the models more open to the reader, each model will conclude with the practical construction of a transition matrix for an example function and search space grid. For simplicity and for ease of

comparison between algorithms the same function will be used to demonstrate the transition matrix of each of the algorithms

$$g(x) = -x^2 + 4x \quad (6.1)$$

Recall from chapter 1, that a function alone does not define the optimisation problem. We must also specify a search space grid: $x = [0,1,2,3]$, hence the set of values for the function are $g(x) = [0,3,4,3]$. The set of input output pairs are $[(0,0),(1,3),(2,4),(3,3)]$. The search space grid contains a unique global maximum, it also contains two points in the grid that have the same affinity; this property has been deliberately included so that all outcomes (greater than, less than and equal to) of the possible transit matrix are required to instantiate the transition matrix for the example.

Having outlined the general framework in chapter 4, let us now go on to demonstrate the use of this framework

6.1 Algorithm model – hill climber

Being a deterministic search algorithm (excluding initialisation), the hill climber is already intuitively and explicitly well understood. Based on the initial position in the search space, the hill climber will follow the basin of attraction to the nearest optima (traditionally maxima) where the algorithm will stay (or terminate). The random initialisation of the algorithm does make it in some sense stochastic, but a Markov chain model could be considered excessive. The purpose of modelling the hill climber algorithm here is to demonstrate the wide applicability of the framework, the model itself is not intended to reveal any points of interest which are not already well known and understood.

Let us explicitly define the algorithm to be modelled

Algorithm 5: Hill Climber

- Step 1: Initialise a binary string x of length l . Each element in the string has a 0.5 probability of having the value 1
- Step 2: evaluate $g(x)$
- Step 3: While stopping criterion not met:
- Step 4: Evaluate the points adjacent to x , $g(x_{+1})$ and $g(x_{-1})$
- Step 5: x' is defined as adjacent point with highest corresponding $g(x_{+1})$. In the case where $g(x_{+1}) = g(x_{-1})$, x_{+1} and x_{-1} have a probability of 0.5 of being selected to become x'
- Step 7: If $g(x') > g(x)$ then replace x by x'
- Step 8: Return to step 3

6.1.1 Determine the minimal definition of a state of the algorithm:

The hill climber algorithm, determines which points in the search space will be searched based on the current point in the search space. Hence the position in the search space is the state of the algorithm. This remains the case even if there is a "population" of hill climbers. Each member of the population operates independently, so we can consider the population to be multiple independent runs of a hill climber without a population, as described in section 5.2.1.

As each row of the transition matrix T represents each possible starting state and the columns represent the end state, T^∞ will give a map of the end state for each of the possible starting states which may be of some value in the case of multi-modal functions.

6.1.2 The Hill Climber Possible Transit matrix

The Possible Transit matrix for a hill climber is determined by the function:

$$G \begin{cases} g(s_i) > g(s_j), M_{ij} = 0 \\ g(s_i) = g(s_j), M_{ij} = 0 \\ g(s_i) < g(s_j), M_{ij} = \frac{1}{k_i} \end{cases} \quad (6.2)$$

And we define k such that the sum of any row of the possible transit matrix is 1. More formally, for a fixed value of i we impose the condition

$$\sum_{j=0}^3 M_{ij} = 1 \quad (6.3)$$

We define that (for a maximizing hill climber) states of equal or lower affinity than the current state are rejected and that states of higher affinity than the current state are accepted with equal probability, such that the sum of the probability is one.

6.1.3 The Hill Climber Sample matrix

At this stage it is helpful to include the knowledge that the example function (6.1) is one dimensional. Consequently we can say

$$S_i \begin{cases} j = [i+1, i-1], S_{ij} = 1 \\ j \neq [i+1, i-1], S_{ij} = 0 \end{cases} \quad (6.4)$$

Which simply means the algorithm searches neighbouring states (with probability 1)

6.1.4 Hill climber example

Using the set of input output pairs $[(0,0),(1,3),(2,4),(3,3)]$ defined by the function (6.1) and the G function given in equation (6.2), the hill climber Possible transit matrix is given by

$$\mathbf{M}_{HC} = \begin{pmatrix} 0 & \frac{1}{k_i} & 0 & \frac{1}{k_i} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \quad (6.5)$$

Using equation (6.4) the sample matrix is given by

$$S_{HC} = \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{pmatrix} \quad (6.6)$$

This is a highly irregular sample matrix, there are two points to note about it: firstly, the rows sum to 2 not to 1 and secondly we have encountered boundary effects which occur when starting in state 0 and state 3. We must actively choose to either wrap the space around, which leads to "1" in the diagonally opposed corners (as shown). Or we could choose to not wrap around, the pseudo code does not specify either choice.

6.1.5 The hill climber transition matrix

After combining the sample matrix and possible transit matrix we obtain

$$T_{HC} = \begin{pmatrix} 0 & \frac{1}{k_i} & 0 & \frac{1}{k_i} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \quad (6.7)$$

and must solve for k using the constraint given by equation (6.3), which in this case is $0 + \frac{1}{k_i} + 0 + \frac{1}{k_i} = 1$, hence $k_i = 2$ for the row relating to starting in the 0 state, $i=0$. In this example, the point at which we solve for k is unimportant. However, in more complicated cases some entries in the possible transit matrix will be multiplied by 0 when combined by the sample matrix, so solving for k before forming the transition matrix will lead to an error. This step is required because the algorithm is comparing two states within the same iteration. In algorithms where the transition matrix is a "one step" matrix the process with k is not required, the sample matrix arbitrates the final transition probabilities.

$$T_{HC} = \begin{pmatrix} 0 & \frac{1}{2} & 0 & \frac{1}{2} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \quad (6.8)$$

It should be noted that an iteration of the hill climber as defined by Algorithm 5, requires two function evaluations per iteration; this is evident as the sum of the rows in the sample matrix is 2 instead of one. This is particularly important when comparing the hill climber to other algorithms in this chapter which make only one call to the function (6.1) per iteration.

This is because the objective measure of performance is the probability of having found the optima in a given number of function calls, not a given number of iterations.

6.2 Algorithm model – Simulated annealing

We present a Markov chain model of simulated annealing based on the model presented in (Häggström, 2002). The purpose of presenting this model is to demonstrate how existing models from the literature can be incorporated by the framework. Häggström does not present a pseudo code for the simulated annealing algorithm, an exact mathematical model being suitable as a definition of an algorithm. For the purposes of following a fixed format, we back-engineer pseudo code from the model presented (Häggström, 2002) and re-model the algorithm using the framework.

Algorithm 6: Simulated Annealing

- Step 1: Initialise a binary string x of length l . Each element in the string has a 0.5 probability of having the value 1
- Step 2: evaluate $g(x)$
- Step 3: While stopping criterion not met:
- Step 4: Update T from the temperature schedule
- Step 4: x_{+1} and x_{-1} have a probability of 0.5 of being selected to become x'
- Step 5: Evaluate $g(x')$

Step 7: If $g(x') \geq g(x)$ then replace x by x' with probability 1. If $g(x') < g(x)$ then replace x by x' with probability $e^{-\left(\frac{g(x')-g(x)}{T}\right)}$

Step 8: Return to step 3

6.2.1 Determine the minimal definition of a state of the algorithm

Algorithm 6 has no factors that complicate the definition of the state of the algorithm (such as a population or memory set). The state is given by the current position in the search space. Simulated annealing has a minimal state space.

6.2.2 The Simulated Annealing possible transit matrix

Traditionally SA is a minimiser, in order to fit in with the example problem of maximising the G function has been given "upside-down" so that transitions to states with a higher or equal affinity are allowed with probability 1 and transitions to states are allowed with some probability less than one controlled by some parameter T

$$G \begin{cases} g(s_i) > g(s_j), M_{ij} = 1 \\ g(s_i) = g(s_j), M_{ij} = 1 \\ g(s_i) < g(s_j), M_{ij} = e^{-\left(\frac{g(s_j)-g(s_i)}{T}\right)} \end{cases} \quad (6.9)$$

6.2.3 The Simulated Annealing Sample matrix

$$s_i \begin{cases} j = [i+1, i-1], S_{ij} = \frac{1}{2} \\ j \neq [i+1, i-1], S_{ij} = 0 \end{cases} \quad (6.10)$$

The sample matrix has not been formed using mutation masks, but is given by explicit rules. Note the similarity between the definition given here and the definition used in Hill climbing, given in section 6.1.3.

6.2.4 Simulated Annealing example

Using the set of input output pairs $[(0,0),(1,3),(2,4),(3,3)]$ defined by the function (6.1) and the G function given in equation (6.9), the simulated annealing possible transit matrix is given by

$$M_{SA} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ e^{-\left(\frac{3}{T}\right)} & 1 & 1 & 1 \\ e^{-\left(\frac{4}{T}\right)} & e^{-\left(\frac{1}{T}\right)} & 1 & e^{-\left(\frac{1}{T}\right)} \\ e^{-\left(\frac{3}{T}\right)} & 1 & 1 & 1 \end{pmatrix} \quad (6.11)$$

The sample matrix has been formed by a rule that assumes there is a state either side of the current state. This leads to the complication of having to choose a special adaptation of the rule to apply at the boundaries of the search space. The choices are either to: have a wrap around space so that there is always a state before and after each state as shown in equation (6.12). Or to not allow wrap around and have boundary effects which is shown in equation (6.13).

$$\begin{pmatrix} 0 & \frac{1}{2} & 0 & \frac{1}{2} \\ \frac{1}{2} & 0 & \frac{1}{2} & 0 \\ 0 & \frac{1}{2} & 0 & \frac{1}{2} \\ \frac{1}{2} & 0 & \frac{1}{2} & 0 \end{pmatrix} \quad (6.12)$$

If the search space did not wrap around we would have

$$S_{SA} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ \frac{1}{2} & 0 & \frac{1}{2} & 0 \\ 0 & \frac{1}{2} & 0 & \frac{1}{2} \\ 0 & 0 & 1 & 0 \end{pmatrix} \quad (6.13)$$

In this instance when the algorithm is against a boundary of the search space it can only search in one direction, therefore the search of the adjacent state takes place with probability 1. The choice to wrap around or not wrap with this kind of search operator is fairly arbitrary when considering realistic size state spaces. For the purposes of this example we shall choose to allow wrap around space to be consistent with the example given in (Häggström, 2002).

6.2.5 The simulated annealing transition matrix

As we have not yet picked a “temperature”, a value for the parameter T the transition matrix is not particularly elegant. Additional algebraic terms are required on the diagonal entries in some states as they can not be resolved without a known value for temperature.

$$T_{SA} = \begin{pmatrix} 0 & \frac{1}{2} & 0 & \frac{1}{2} \\ \frac{1}{2}e^{-\left(\frac{3}{T}\right)} & \frac{1}{2}\left(1-e^{-\left(\frac{3}{T}\right)}\right) & \frac{1}{2} & 0 \\ 0 & \frac{1}{2}e^{-\left(\frac{1}{T}\right)} & 1-e^{-\left(\frac{1}{T}\right)} & \frac{1}{2}e^{-\left(\frac{1}{T}\right)} \\ \frac{1}{2}e^{-\left(\frac{3}{T}\right)} & 0 & \frac{1}{2} & \frac{1}{2}\left(1-e^{-\left(\frac{3}{T}\right)}\right) \end{pmatrix} \quad (6.14)$$

Having given a general, in terms of T , transition matrix for SA, for the sake of comparison let us put in a value of $T = 1$ for the example.

$$T_{SA} = \begin{pmatrix} 0 & 0.5 & 0 & 0.5 \\ 0.025 & 0.475 & 0.5 & 0 \\ 0 & 0.184 & 0.632 & 0.184 \\ 0.025 & 0 & 0.5 & 0.475 \end{pmatrix} \quad (6.15)$$

$T = 1$ may not be a very suitable value for temperature by conventional standards, although on a problem of this size a conventional temperature may not be suitable. Equation (6.14) could be used to explore the parameter space for T and how it affects the probability of locating the optimum in a given number of iterations. Note that simulated annealing does not have an explicit memory mechanism when the temperature is high, however as $T \rightarrow 0$ the simulated annealing algorithm becomes elitist, and shares many similarities with a hill climbing algorithm.

6.3 Algorithm model – (1+ λ) Evolutionary Strategy

We shall create a model of the so called (1+ λ) ES. We start by defining the algorithm to be modelled.

Algorithm 7: $(1+\lambda)$ Evolutionary Strategy

- Step 1: Initialise a binary string x of length l , each element of the string has a 0.5 probability of having the value 1
- Step 2: Evaluate $g(x)$
- Step 3: While stopping criterion not met:
- Step 4: Create λ mutants by making λ copies of x and mutating them using the affinity proportional mutation operator (defined in section 6.3.3)
- Step 5: Evaluate $g(x_\lambda)$ each of the λ mutants
- Step 6: x' is defined as the mutant with the highest corresponding $g(x_\lambda)$. In the case where k mutants have the same highest $g(x_\lambda)$ value, each of the k mutants has a probability of k^{-1} of being selected to become x'
- Step 7: If $g(x') > g(x)$ then replace x by x'
- Step 8: Return to step 3

The pseudo code includes affinity proportional mutation, elitism, both of which are readily understood in the light of the framework. The affinity proportional mutation is yet to be defined and modelled, but this is a formality which will be tackled in the sample matrix section. The only real issue is of the λ mutants making the algorithm an unknown form. We shall tackle this problem directly in the definition of the state of the algorithm.

6.3.1 Definition of state of the algorithm

The definition of the state of an algorithm for the $(1+\lambda)$ ES is non-trivial, as at each generation λ mutants are produced to compete with the parent. However given the work undertaken in chapter 3 on the BCA we can compare λ mutants competing with the parent to spawn the next generation with C clones competing with the parent to spawn the next generation. Considered in the light of this comparison it becomes clear that there is no significant mathematical difference between a clonal selection algorithm from AIS and an ES. Having made the comparison, we can see that the difference between C clones and λ mutants is a semantic one and that the same mathematical approach can be applied to both cases.

As in the case of the BCA we can make the definition of the state of the algorithm minimal if we monitor the transitions from parent to parent instead of from group of mutants to group of mutants. The approach used to generate the clonal matrix described in section 3.7.3 can be re-applied here and the resulting one step transition matrix will consequently represent λ function evaluations.

6.3.2 Possible transit matrix

The $(1+\lambda)$ ES uses the same elitist operator as the B-Cell Algorithm where states with lower or equal affinity are rejected and states with higher affinity are accepted. The G function for this is

$$G \begin{cases} g(s_i) > g(s_j), M_{ij} = 0 \\ g(s_i) = g(s_j), M_{ij} = 0 \\ g(s_i) < g(s_j), M_{ij} = 1 \end{cases} \quad (6.16)$$

6.3.3 Sample matrix

We must construct the sample matrix for a given number of clones and a given set of input output pairs. In most cases the sample matrix is independent of the objective function, in the case of the clonal matrix, or sample matrix that incorporates the affects of the λ mutants, the sample matrix does depend on the objective function as the states need to be ranked. The approach presented in chapter 3 will be reused.

We also need to account for the affinity proportional mutation. One would intuitively expect that affinity proportional mutation would be a dynamic operator, however as the states of the algorithm and their associated affinities are known the operator can be treated as static, if the mutation rate is calculated for each state.

Let us define the mutation operator to mutate each bit in the string independently (standard bit flip mutation). We start by defining how the mutation rates will be determined for each state of the algorithm.

The mutation rate for a given state s_i is defined by

$$m_i = \exp\left(-\rho \frac{g(s_i)}{\max(g)}\right), \quad (6.17)$$

where ρ is a mutation rate multiplier, example values are $\rho = 5, 10, 20$ (Castro and Von Zuben, 2002) $g(s_i)$ is the value of the objective function g , at the state s_i .

Continuing with the mutation masking method, as in chapter 3, we require the function f_T which denotes the probability of forming the mutation mask

required to get to some string T , from the string corresponding to 0. Recall that the mutation mask, T and 0 form a triple linked by the exclusive or operator.

For the operator that mutates each bit in the length of a string l with a probability m_i

$$f_T = (m_i)^k (1 - m_i)^{(l-k)} \quad (6.18)$$

Where k is the total number of bits that must be flipped to mutate from 0 to T .

As the value of the probability of mutation m changes from row to row we introduce the notation

$$f_{T,i} := f_T(m_i) \quad (6.19)$$

where the T defines the form of the equation and i denotes the row number, which in turn defines the mutation rate for that row m_i .

6.3.4 (1+ λ) Evolutionary Strategy example

Using the set of input output pairs $[(0,0),(1,3),(2,4),(3,3)]$ defined by the function (6.1) and the G function given in equation (6.16), the (1+ λ) ES possible transit matrix is

$$M_{ES} = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \quad (6.20)$$

As the mutation operator in this case is a mutation mask, we can apply the method described in section 3.7.3, where the sum of the mutation masks are raised to the power of the number of clones to produce the correct set terms. These terms are then associated with a state j based on the ranking of the search space. For conceptual ease we shall use the mutation masks from $i=0$, hence the mutation mask subscript f_T is determined by $f_{T=j}$.

$$\left(\sum_{T=0}^{2^L-1} f_T \right)^\lambda = 1 \quad (6.21)$$

We must at this point specify a value for λ the number of mutants. Let us say that $\lambda = 2$.

$$\begin{aligned} \left(\sum_{T=0}^3 f_T \right)^2 &= 1 = (f_0 + f_1 + f_2 + f_3)^2 \\ &= f_0^2 + f_1^2 + f_2^2 + f_3^2 + 2f_0f_1 + 2f_0f_2 + 2f_0f_3 + 2f_1f_2 + 2f_1f_3 + 2f_2f_3 \end{aligned} \quad (6.22)$$

Ranking the input output pairs $[(0,0),(1,3),(2,4),(3,3)]$, given by equation (6.1), we can form the top row of the ES sample matrix S_{0j} . Any clonal pool that contains the optimum will cause the algorithm transit to the optimum irrespective of the other members of the clonal pool, hence all terms that contain f_2 belong to S_{02} . After these terms have been removed, the remaining terms are claimed in order of the rank of the state S_{0j} giving.

$$\begin{aligned} p_{0,0} &= f_0^2(m_0) \\ p_{0,1} &= f_1^2(m_0) + 2f_0(m_0)f_1(m_0) + f_1(m_0)f_3(m_0) \\ p_{0,2} &= 2f_2(m_0)f_0(m_0) + 2f_2(m_0)f_1(m_0) + f_2(m_0)^2 \\ p_{0,3} &= f_3(m_0)^2 + 2f_0(m_0)f_3(m_0) + f_1(m_0)f_3(m_0) \end{aligned} \quad (6.23)$$

The mutation mask methodology can then fill out the sample matrix

$$S_{ES} = \begin{pmatrix} p_{0,0} & p_{0,1} & p_{0,2} & p_{0,3} \\ p_{1,0} & p_{1,1} & p_{1,2} & p_{1,3} \\ p_{2,0} & p_{2,1} & p_{2,2} & p_{2,3} \\ p_{3,0} & p_{3,1} & p_{3,2} & p_{3,3} \end{pmatrix} \quad (6.24)$$

Where the elements of the matrix S_{ES} are given by

$$\begin{aligned} p_{0,0} &= f_{0,0}^2 & ; & & p_{0,1} &= f_{1,0}^2 + 2f_{0,0}f_{1,0} + f_{1,0}f_{3,0} \\ p_{1,0} &= f_{1,1}^2 & ; & & p_{1,1} &= f_{0,1}^2 + 2f_{1,1}f_{0,1} + f_{0,1}f_{2,1} \\ p_{2,0} &= f_{2,2}^2 & ; & & p_{2,1} &= f_{3,2}^2 + 2f_{2,2}f_{3,2} + f_{3,2}f_{1,2} \\ p_{3,0} &= f_{3,3}^2 & ; & & p_{3,1} &= f_{2,3}^2 + 2f_{3,3}f_{2,3} + f_{2,3}f_{0,3} \end{aligned} \quad (6.25)$$

$$\begin{aligned} p_{0,2} &= 2f_{2,0}f_{0,0} + 2f_{2,0}f_{1,0} + f_{2,0}^2 + 2f_{2,0}f_{3,0} & ; & & p_{0,3} &= f_{3,0}^2 + 2f_{0,0}f_{3,0} + f_{1,0}f_{3,0} \\ p_{1,2} &= 2f_{3,1}f_{1,1} + 2f_{3,1}f_{0,1} + f_{3,1}^2 + 2f_{3,1}f_{2,1} & ; & & p_{1,3} &= f_{2,1}^2 + 2f_{1,1}f_{2,1} + f_{0,1}f_{2,1} \\ p_{2,2} &= 2f_{0,2}f_{2,2} + 2f_{0,2}f_{3,2} + f_{0,2}^2 + 2f_{0,2}f_{1,2} & ; & & p_{2,3} &= f_{1,2}^2 + 2f_{2,2}f_{1,2} + f_{3,2}f_{1,2} \\ p_{3,2} &= 2f_{1,3}f_{3,3} + 2f_{1,3}f_{2,3} + f_{1,3}^2 + 2f_{1,3}f_{0,3} & ; & & p_{3,3} &= f_{0,3}^2 + 2f_{3,3}f_{0,3} + f_{2,3}f_{0,3} \end{aligned}$$

Where f_T is given in terms of m_i , k and l by equation (6.18) and m_i is given by equation (6.17) and must be calculated for each row.

For the example we must move from the algebraic version of the sample matrix to the numerical which requires us to calculate the forms of the equations for f_0, f_1, f_2, f_3 . We have $l=2$ and must determine k (the total number of bits that must be flipped to make the transition) for each value of T

$$\begin{aligned} T=0, k=0, f_0 &= (1-m_i)^2 \\ T=1, k=1, f_1 &= (m_i)(1-m_i) \\ T=2, k=1, f_2 &= (m_i)(1-m_i) \\ T=3, k=2, f_3 &= (m_i)^2 \end{aligned} \quad (6.26)$$

We must also calculate a value for m_i for $i = 0, 1, 2, 3$ using equation (6.17). Let us set $\max(g) = 4$ and set $\rho = 5$ which gives us

$$\begin{aligned} g(s_0) &= 0, m_0 = 1 \\ g(s_1) &= 3, m_1 = 0.0235 \\ g(s_2) &= 4, m_2 = 0.0067 \\ g(s_3) &= 3, m_3 = 0.0235 \end{aligned} \tag{6.27}$$

6.3.5 $(1+\lambda)$ Evolutionary Strategy transition matrix

Substituting the definitions of f_T from equation (6.26) into equation (6.25) and using the values from (6.27) we have the numerical sample matrix for the $(1+\lambda)$ ES

$$S_{ES} = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 5.2738 \times 10^{-4} & 0.9749 & 0.0011 & 0.0235 \\ 5.2738 \times 10^{-4} & 3.8410 \times 10^{-5} & 0.9978 & 0.0016 \\ 2.0612 \times 10^{-9} & 0.0066 & 0.0133 & 0.9800 \end{pmatrix} \tag{6.28}$$

Using equations (6.28) and (6.20) we can form the transition matrix for this example

$$T_{ES} = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0.9989 & 0.0011 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0.0133 & 0.9867 \end{pmatrix} \tag{6.29}$$

6.4 Summary

We presented Markov chain models and numerical examples of three optimisation algorithms: Hill climber, Simulated Annealing and $(1+\lambda)$ Evolutionary strategy in terms of the framework presented in chapter 5. These instantiations demonstrate the framework's power and flexibility, to simplify the modelling process and to present models of multiple algorithms under a single set of assumptions, allowing for direct comparison between the models of the algorithms.

The model of the hill climber demonstrates the flexibility of the framework. The crude version of the algorithm defined by Algorithm 5 makes two function evaluations for each change of state. In all other algorithms presented in this thesis the sample matrix is a stochastic matrix (each row sums to 1), but in the case of the hill climber the sum of each row is two. The framework is able to incorporate this property without modification.

The model of the simulated annealing algorithm is a re-instantiation of a Markov chain model presented by (Häggström, 2002). This model was reformulated in terms of the framework in order to demonstrate how existing complete and exact models in the literature can be included in the framework. Häggström's model was chosen as the example as it has an exceptionally clear and concise traditional instantiation and numerical example in (Häggström, 2002) allowing for a fairly easy comparison of the more traditional modelling method and models produced via the framework. We find the model of simulated annealing presented here to be no more complicated than the model presented by Häggström.

Unlike the hill climber and simulated annealing models which are included primarily to demonstrate the framework, the model of the $(1+\lambda)$ Evolutionary Strategy is a novel model. The model is particularly significant because it maintains a minimal state space which allows a realistic possibility of meaningful numerical work in the future. This model also

includes a fitness proportional mutation operator defined by equation (6.17).

The model of the $(1+\lambda)$ ES as defined by Algorithm 7 is almost identical to the model of the clonal selection algorithm, the B-Cell Algorithm (BCA) defined by Algorithm 4 presented in chapter 3, the only mathematical difference being the choice of mutation operator.

These three models together demonstrate the power and flexibility of the framework to incorporate a diverse range of algorithms, existing Markov chain models and to aid the development of novel models of existing algorithms by simplifying the modelling process.

Chapter 7: Discussion and Conclusion

In this chapter we provide a discussion of the framework for modelling stochastic optimisation algorithms, its development and associated theoretical results. In section 7.1 we discuss the modelling framework. In section 7.2 we discuss the potential for using Markov chain models of an algorithm as the specification for that algorithm and the associated advantages and disadvantages of this approach. General observations about bio-inspired algorithms are then given in section 7.3 based on the six Markov chain models presented in the thesis. The algorithms modelled are three variants of the B Cell Algorithm presented in chapter 3, see sections 3.3, 3.4, 3.7 for models of Algorithm 2, 3 and 4 respectively. As well as a model of a Hill Climber, Algorithm 5 in section 6.1; Simulated Annealing, Algorithm 6 in section 6.2; and a model of the $(1+\lambda)$ Evolutionary Strategy with fitness proportional mutation, Algorithm 7 in section 6.3. In section 7.4 we provide an overview of the theoretical analysis that has been performed in the thesis. We give an analysis of proofs of convergence in section 7.4.1, bounds on the rate of convergence in section 7.4.2 and numerical analysis of Markov chain models in 7.4.3. We present the conclusions of the thesis in section 7.5 and outline the future work in 7.6

7.1 The framework

We have developed a novel modelling methodology, whereby the transition matrix of the Markov chain is constructed in two parts: the sample matrix which represents the search operator and the possible transit matrix which represents the state operator.

The sample matrix determines the probability distribution over the points in the search space to be searched in the iteration, see section 5.3. The possible transit matrix determines how to update the state of the algorithm

based on the information obtained by sampling the search space, see section 5.4. Section 5.5 describes how these matrices are then combined to form the transition matrix of a Markov chain model using equations (5.11) and (5.12).

The two-matrix construction allows the model to be easily updated to reflect variants of the algorithm. It also allows a transparent view of which properties of the transition matrix are associated with which operators. Apart from the conceptual advantage of this novel modelling methodology, there is a clear technical advantage to approaching the problem in this way. The combination of the off diagonal elements of the sample matrix and possible transit matrix via equation (5.11) is simply an element to element multiplication which could easily be obtained under a traditional approach. However, forming the diagonal elements via equation (5.12) demonstrates that in cases of full or partial elitism, knowledge of the entire row is required in order to give the value explicitly.

7.2 Markov chain models as algorithm specifications

Markov chain models of optimisation algorithms are a completely explicit specification of the algorithm and leave no room for interpretation. Markov chain models also require an equation characterising the probability distribution of the mutation operator, which can be used to check that an implementation of the operator is actually correct. Stochastic operators give an inherently stochastic output, checking that the operator has been correctly implemented is a non-trivial problem.

There are some clear advantages to using the Markov chain model as the specification for implementations of optimisation algorithms. There are however some potential pit-falls. While some aspects of the model of the algorithm are easily modified, which can correspond to large changes to the algorithm; there are also trivial coding changes that can be extremely difficult to incorporate in an existing model. For example, an algorithm that

has a population, but no population based operators, has a minimal set of states, see section 5.2.1. If the population is in any way ordered, ranked or otherwise interacting, the states of the algorithm change from minimal to non-minimal.

Consider the example: each member of a population of ten has a 0.2 chance of being mutated with operator A, and a 0.8 chance of being mutated with operator B. In this case the state space remains minimal and a composite sample matrix is derived by a weighted sum of the probability distributions of each of the operators, as in section 5.3.2. Compare this with: the 'fittest' two members of the population, out of a population of ten (implies the population is ranked by a population based operator) are mutated by one operator and the rest of the population is mutated by a different operator. Moving from the first example to the second is only a small coding change, but the model of the algorithm must be re-worked to incorporate the change in the definition of a state of the algorithm. The differences in the code for these two situations may be subtle, but from a theoretical perspective the difference is fundamental.

There are some advantages to using the Markov chain model of the algorithm as the specification for the algorithm instead of pseudo code; we should however consider the potential disadvantages of this very formalised method of specifying optimisation algorithms: If algorithms were specified by the Markov chain model it may well discourage the use of population based operators, because such models are more difficult to update and not nearly as transparent due to the lexicographic ordering of states, as proposed by Nix and Vose (Nix and Vose, 1992). The lexicographic ordering of states defined by populations is not at all intuitive, other orderings have been proposed (Spears and De Jong, 1997), such as ordering based on the average hamming distance within the population (homogeneity), but such orderings have no definitive advantages over the lexicographic ordering.

With full awareness of the potential drawbacks of using the Markov chain model of an optimisation algorithm as the specification for the algorithm, we

propose that this is an approach worthy of investigation and development. We shall return to this point in section 7.6 where the discussion of this potential future use of the framework is given in detail.

7.3 General observations about bio-inspired algorithms

We have presented several versions of the B Cell Algorithm, see sections 3.3, 3.4 and 3.7 during the case study on which the framework is based. We have also applied the framework to produce models of a Hill Climbing algorithm in section 6.1, Simulated Annealing in section 6.2 and the $(1+\lambda)$ evolutionary strategy in 6.3. We have found that the algorithms contain many similarities and indeed that algorithm operators are interchangeable. This result is in agreement with the results of a practical framework for population based algorithms proposed by Newborough and Stepney (Newborough and Stepney, 2005) who found "...properties previously thought particular to one class of algorithms to be applied uniformly across all the algorithms."

We also found by comparing the Markov chain models of the BCA and $(1+\lambda)$ ES that although the algorithms come from different bio-inspired fields and do not use a common vocabulary, that the corresponding models take the same form, see sections 3.7 and 6.3. The only observed difference between the models of the BCA and the $(1+\lambda)$ ES is the mutation operator. This can be seen by comparison of the pseudo codes of Algorithm 4 and Algorithm 7 which have been written in such a way as to accurately reflect the Markov chain models.

While developing the framework, we have found a number of key differences in the minimal definition to the state of an algorithm, summarised in Table 11; as well as a key difference between elitist algorithms, non-elitist algorithms and partially elitist algorithms, see section 5.4. All of the algorithmic distinctions discussed in this thesis can be

incorporated by the framework developed in chapter 5. We have observed that all stochastic optimisation algorithms discussed in this thesis, irrespective of their inspiration, attempt to make iterative improvements based on some assumptions about the type of function they are optimising. These assumptions are implicitly made by the forms of the search and state operators.

7.4 Convergence proofs and theoretical analysis

We have presented a proof of convergence for the elitist BCA (Algorithm 3) in section 3.5, we demonstrate in section 3.5.1 that this proof is a general proof of convergence. We also conduct a brief numerical analysis of the Markov chain model of the BCA (Algorithm 3) in section 3.6. In chapter 4 we obtain bounds on the rate of convergence of elitist optimisation algorithms. We review these results here and focus our discussion of these results in terms of their potential value to practitioners.

7.4.1 Proof of convergence

In section 3.5 we presented a proof of convergence for the elitist BCA (Algorithm 3) by proving that the model takes the form of a reducible Markov chain where all non-optimal states are transient and all optimal states are absorbing. In section 3.5.1 we demonstrated that in order to be covered by the proof an algorithm need satisfy only two criteria:

- The sample matrix is positive
- The algorithm is elitist

If an algorithm is capable of satisfying these requirements with any parameter settings, then it will generally do so for almost all values of the parameters (excluding the bounds of the parameters where probabilities can become zero). In contrast to this theoretical result, parameter tuning is

often an important part of practical optimisation. This contrast comes about because of the vast difference in timescales between the practical and theoretical observation of the algorithm's behaviour.

The number of function evaluations in which proof of behaviour is relevant to practitioners is far less than the number of function evaluations in which theoretical results are obtained. The very longest timescale that should be considered, other than for pure mathematical interest, is the number of points in the search space. Even this is very long in practical terms as, optimisation algorithms are applied to problems where plotting the search space is an intractable task (excluding benchmark functions).

Having constructed a proof of convergence for the BCA and extended it to be a general set of conditions for a proof in section 3.5.1, it has become apparent that the constraints that need to be met in order for an algorithm to be convergent absolute are not tight enough to be of practical interest, over and above proving the algorithm is an optimisation algorithm.

A proof of convergence does little more than prove that the algorithm is an optimisation algorithm; meaning that it generates a sequence of solutions or function values in which a global optimum is a limit value. This in itself, is actually of questionable importance given that the canonical GA has been shown never to converge (has no limit value in its sequence of solutions) and as such is not a function optimiser (De Jong, 1992, Vose, 1995, Rudolph, 1994a), as discussed in section 2.3.5. Supporting this view is the example in section 3.6, where we find that the mutation rate $r=1$ gives the best performance for Algorithm 3 on a simple quadratic function is outside the conditions of the proof of convergence. It is not just that the value of the mutation rate causes it to lie outside the proof of convergence, as is the case with SA which can be shown to be convergent under a different proof (Häggström, 2002). Algorithm 3 with $r=1$ can in fact be shown *not* to be convergent absolute by analysis of the behaviour of the algorithm on the needle in a haystack function. The formal proof of this has not been given, but is available by inspection of Figure 7. Despite the fact that Algorithm 3

with $r=1$ is not a general function optimisation algorithm in the formal sense, it performs better than Algorithm 3 with $0 < r < 1$ which is a general function optimisation algorithm.

7.4.2 Bounds on the rate of convergence

In chapter 4 we obtained bounds on the rate of convergence of elitist optimisation algorithms. The upper bound is shown to be equivalent to the upper bound derived for the rate of convergence of GAs (Aytug and Koehler, 2000) and achieved independently by (Greenhalgh and Marshall, 2001). Analysis of why all the bounding results have been shown to be equivalent has revealed that all the approaches are isomorphic to plotting a given fraction of the search space, the relationship between the problems is characterised by equation (4.12) which also gives the lower limit to the upper bound. The conclusion of this analysis is that in order to have a confidence of δ of having found the optimum of *an unknown function*, $\delta|S|$ points in the search space must be evaluated, where $|S|$ is the total number of points in the search space. Put simply, the analysis reveals that if you wish to have a confidence of $\delta = 0.95$ of having found the optima of an unknown function, plot 95% of the search space.

Due to the stochastic nature of the algorithms and the fact that the algorithms allow "retracing" (evaluating a point in the search space that has previously been evaluated) the bounds that can be obtained require more function evaluations than simply plotting $\delta|S|$ points of the search space. In the case of values of δ that are close to 1, the number of function evaluations given by the bound is many times the number of points that are in the search space, so if the number of function evaluations required by the bound can actually be achieved, the search space could simply be plotted instead of using a stochastic optimisation algorithm.

Our observations on bounding the rate of convergence of an algorithm on an *unknown function* echo those in the EA literature (Rudolph, 1998) and AIS literature (Cutello et al., 2007), who go on to advocate obtaining bounds on certain problem classes. Obtaining bounds for specific function classes *may* lead to upper bounds that are sufficiently low to be of practical interest. However, they will only be applicable in cases where practitioners already know that their problem corresponds to a problem class for which a practical bound has been obtained.

7.4.3 Numerical analysis

We conducted a brief numerical analysis of the Markov chain model of the BCA (Algorithm 3) in section 3.6, the results of which are presented in Figure 5. We have demonstrated that numerical work with Markov chains can take place over a broad range of timescales relatively cheaply as the one step transition matrix can be squared, and then the resulting matrix squared again and so on, allowing examination of a t step transition matrix, where $t = 2^\zeta$, in ζ matrix multiplications.

Numerical work is not without its challenges however, the amount of memory required to 'write down' the transition matrix is at best proportional to 2^{2l} . Problems on the order of $l=10$ can be run on a high specification desktop, but each increase by 1 in l requires a factor of 4 increase in memory, so numerical work quickly becomes expensive in terms of memory. So although it is by no means a complete solution to bridging the gap between theoretical work and practitioners, it is certainly an area that could be explored more fully.

The numerical survey on the effects of mutation rate r on the rate of convergence of Algorithm 3 presented in Figure 5 in section 3.6 was brief, but the results obtained were both unexpected and significant. As discussed in section 7.4.1, the value of the mutation rate that gave the best rate of

convergence was not only outside the proof of convergence given in section 3.5, but can be shown to not be convergent absolute. Also of interest, was the extremely long time required for $r = 0.1$, which is inside the conditions of the proof of convergence, to obtain a probability of ~ 1 of having located the optimum, approximately 2^{40} iterations required, which is a vast number considering the search space only had 2^8 points in it.

Perhaps the most important observation about the results summarised in Figure 5 is that very high probabilities of convergence were obtained, for multiple values of r , in a number of function evaluations far less than 2^8 . This result is of course function specific and unlikely to hold for all functions, it demonstrates that a theoretical approach can obtain results in a number of function evaluations that makes the result relevant to practitioners.

We have discussed in this section the computational burden of numerical analysis of Markov chain models. The promising thing about the computational burden is that it is dependent on the size of the search space, but the function that is mapped onto that search space grid can be of arbitrary complexity. A numerical analysis approach can be applied to any function and obtain results for a practical number of function evaluations.

In our investigation in section 3.6, we considered the effects of mutation rate r on the rate of convergence of Algorithm 3. For some function of arbitrary complexity, the optimal mutation rate could be obtained for that particular function by numerical analysis. Having obtained an optimal mutation we could subsequently investigate the optimal clonal size for Algorithm 4, described in section 3.7, which is a modification of Algorithm 3. We conjecture that the optimal mutation rate is independent of the size of the clonal pool and as such could be determined and then used to determine the optimal clonal number.

7.5 Thesis conclusions

We have presented a framework for modelling stochastic optimisation algorithms with Markov chains. The framework has been derived from a case study of modular modelling of the B Cell Algorithm (BCA). Based on the Markov chain models of the BCA we have obtained a proof of convergence of the algorithm, bounds for the rate of convergence of the algorithm and a brief numerical analysis of the Markov chain model. While the results of the convergence proof and bounds on the rate of convergence are both true and general, such results are obtained for a number of function evaluations greater than the number of points in the search space. We propose that the most promising method of obtaining results that are of practical interest is numerical analysis of exact Markov chain models.

We have applied the framework to create models of a Hill Climber, Simulated Annealing and a novel model of the $(1+\lambda)$ Evolutionary Strategy with fitness proportional mutation. Along with the model of the BCA that provided the basis for the framework, we have demonstrated that the framework is not specific to any particular field of optimisation.

The framework demonstrates that optimisation algorithms can conceptually be split into two parts: search operators and state operators. Search operators are represented by the sample matrix described in section 5.3. State operators are represented by the possible transit matrix described in section 5.4. These matrices can be combined by equations (5.11) and (5.12) to form the transition matrix of a Markov chain model of an algorithm. These equations allow the construction of the transition matrix from the sample matrix and possible transit matrix; they are the key to the framework:

$$T_{ij} = S_{ij}M_{ij}, \quad \forall i \neq j \quad (5.11)$$

$$T_{jj} = M_{jj}S_{jj} + \sum_{i=0}^{2^l-1} (1 - M_{ij})S_{ij} \quad (5.12)$$

Where T is the transition matrix, S is the sample matrix and M is the possible transit matrix, i is the row index and j is the column index.

Equation (5.12) gives the diagonal elements of the transition matrix, from the form of equation (5.12) one can see that the diagonal elements potentially depend on the value of every other element in the row, which could make obtaining a single formula of the form " $T_{ij} =$ " extremely awkward, depending on the form of the matrices S and M . Using the framework to avoid this problem, greatly simplifies the construction of exact models of elitist algorithms.

7.6 Future work

This thesis has provided a framework for modelling stochastic optimisation algorithms with Markov chains, greatly simplifying the task of generating exact models of optimisation algorithms. We propose that the best way to make use of the framework is to explore the ideas raised in section 7.2, where the model of an algorithm acts as the specification. We propose that with a library of theoretically modelled operators, a toolkit could be constructed based on the framework, where the specification of an algorithm generates both the code for the algorithm and an exact Markov chain model of the algorithm; encompassing both the theoretical and practical aspects of algorithm development.

The toolkit would allow operators from all the fields of optimisation to be hybridised and investigated via the automatic generation of exact Markov chain models and executable code. The Markov chain models would be available for further theoretical scrutiny, as well as allowing the potential for numerical analysis of the models.

Each operator added to the library of modelled operators would provide the potential for an ever increasing number of optimisation algorithms; as within the framework, any search operator can be combined with any state operator. As a consequence, the toolkit would be ideal for the construction and analysis of hybrid algorithms.

The toolkit would be of considerable value to theorists; automatically producing models of the algorithms that can be constructed from the library of operators and even providing a certain level of automatic classification and proof of convergence of the algorithms that can be constructed from the library of operators. However, we propose that the primary focus of the toolkit should be the numerical analysis of the exact Markov chain models. This would require significant computational resources, but would provide results that are of practical interest. We propose the toolkit could be used to conduct a theoretical benchmarking of the algorithms on an appropriate set of test functions, for all the algorithms that can be constructed from the library of operators. As well as the automated aspects, the toolkit could be used to aid development of novel algorithms for specific application areas.

References

- ALTENBERG, L. (1995) The Schema Theorem and Price's Theorem. *Foundations of Genetic Algorithms 3*, 23-49.
- ANDRE, J., SIARRY, P. & DOGNON, T. (2001) An improvement of the standard genetic algorithm fighting premature convergence in continuous optimization. *Advances in Engineering Software*, 32, 49-60.
- AYTUG, H., BHATTACHACHARYYA, S. & KOEHLER, G. J. (1996) A Markov chain analysis of general cardinality genetic algorithms with power of cardinality alphabets. *European Journal of Operations Research*, 96, 195-201.
- AYTUG, H. & KOEHLER, G. J. (1996) Stopping Criteria for Finite Length Genetic Algorithms. *INFORMS journal on computing*, 8, 183-191.
- AYTUG, H. & KOEHLER, G. J. (2000) New stopping criteria for genetic algorithms. *European journal of operational research*, 126, 662-647.
- BERSINI, H. (1991) Immune Network and Adaptive Control. Proceedings of the 1st European Conference on Artificial Life (ECAL), 217-226.
- BERSINI, H. & VARELA, F. J. (1991a) Hints for Adaptive Problem-Solving Gleaned from Immune Networks. *Parallel Problem Solving from Nature*, 496, 343-354.
- BERSINI, H. & VARELA, F. J. (1991b) The immune recruitment mechanism: A selective evolutionary strategy. *Proceedings of the Fourth International Conference on Genetic Algorithms*, 520-526.
- BEYER, H. G. & SCHWEFEL, H. P. (2002) Evolution strategies – A comprehensive introduction. *Natural Computation*, 1, 3-52.
- CASTRO, L. N. D. & VON ZUBEN, F. J. (2002) Learning and optimization using the clonal selection principle. *IEEE Transactions on Evolutionary Computation*, 6, 239-251.
- ČERNÝ, V. (1985) Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *Journal of Optimization Theory and Applications*, 45, 41-51.

- CLARK, E., HONE, A. & TIMMIS, J. I. (2005) A Markov Chain Model of the B-Cell Algorithm. *International Conference on Artificial Immune Systems, ICARIS, LNCS 3627*, 318-330.
- CLARK, E., MEAD, R. & MOUNTJOY, G. (2006) A molecular dynamics model of the atomic structure of Tb metaphosphate glass Tb_2O_3 0.25(P_2O_5)0.75. *Journal of Physics: Condensed Matter*, 18, 6815-6826.
- COELLO, C. A. C. & CORTES, N. C. (2002) An approach to solve multiobjective optimization problems based on an artificial immune system. *International Conference on Artificial Immune Systems, ICARIS*, 212-221.
- CUTELLO, V., NICOSIA, G., ROMEO, M. & OLIVETO, P. S. (2007) On the Convergence of Immune Algorithms. *Foundations of Computational Intelligence*, 409-415.
- CZIKO, G. (1995) *The Immune System: Selection by the Enemy*, A Bradford Book, The MIT Press.
- DASGUPTA, D. & GONZALEZ, F. (2002) An immunity-based technique to characterize intrusions in computer networks. *Evolutionary Computation, IEEE Transactions on*, 6, 281-291.
- DAVIES, M., SECKER, A., FREITAS, A., TIMMIS, J., CLARK, E. & FLOWER, D. (2008a) Alignment-Independent Techniques for Protein Classification. *Current Proteomics*, 5, (in press).
- DAVIES, M., SECKER, A., FREITAS, A., TIMMIS, J., CLARK, E. & FLOWER, D. (2008b) Optimising amino acid groupings for GPCR classification. *Bioinformatics*, 24, 1980-1986.
- DAVIES, M., SECKER, A., HALLING-BROWN, M., MOSS, D., FREITAS, A., TIMMIS, J., CLARK, E. & FLOWER, D. (2008c) GPCRTree: Online Hierarchical Classification of GPCR Function. *BMC Research Notes*, 1.
- DAVIS, T. (1992) Toward an extrapolation of the simulated annealing convergence theory onto the simple genetic algorithm. PhD. Thesis, University of Florida.
- DE CASTRO, L. N. & VON ZUBEN, F. J. (2000) The clonal selection algorithm with engineering applications. *Genetic and Evolutionary Computation Conference; GECCO*, 36-37.
- DE JONG, K. (1992) Genetic Algorithms are NOT Function Optimizers. *Foundations of Genetic Algorithms*, 2, 5-17.

- DE JONG, K. A., SPEARS, W. M. & GORDON, D. F. (1995) Using Markov chains to analyze GAFOs. *Foundations of Genetic Algorithms*, 3, 115-137.
- EIBEN, A. E., AARTS, E. H. L. & VAN HEE, K. M. (1991) Global convergence of genetic algorithms: A Markov chain analysis. *Parallel Problem Solving from Nature*, 1, 4-12.
- EIBEN, A. E. & RUDOLPH, G. (1999) Theory of evolutionary algorithms: a bird's eye view. *Theoretical Computer Science*, 229, 3-9.
- FARMER, J. D., PACKARD, N. H. & S., P. A. (1986) The Immune System, Adaptation, and Machine Learning. *Physica D*, 22, 187-204.
- FOGEL, D. (1992) Evolving artificial intelligence. PhD thesis, University of California.
- FOGEL, D. (1998) Evolutionary Computation: The Fossil Record., IEEE PRESS.
- GIDAS, B. (1985) Nonstationary Markov chains and convergence of the annealing algorithm. *Journal of Statistical Physics*, 39, 73-131.
- GREENHALGH, D. & MARSHALL, S. (2001) Convergence criteria for Genetic Algorithms. *SIAM journal on computing*, 30, 269 - 282.
- GREENSMITH, J., AICKELIN, U. & TWYLCROSS, J. (2004) Detecting Danger: Applying a Novel Immunological Concept to Intrusion Detection Systems. *6th International Conference in Adaptive Computing in Design and Manufacture*. Bristol, UK.
- GREFENSTETTE, J. J. & BAKER, J. E. (1989) How genetic algorithms work: a critical look at implicit parallelism. *Proceedings of the third international conference on Genetic algorithms 93136*, 20-27.
- GRIMMETT, G. R. & STIRZAKER, D. R. (1982) *Probability and random processes*, Oxford, Oxford University Press.
- HÄGGSTRÖM, O. (2002) *Finite Markov Chains and Algorithmic Applications*, Cambridge University Press.
- HICKINBOTHAM, S., CLARK, E., STEPNEY, S., CLARKE, T. & YOUNG, P. (2009) Gene Regulation in a Particle Metabolome. submitted to Congress on Evolutionary Computation 2009 (under review).
- HOLLAND (1975) *Adaptation in Natural and Artificial Systems*, University of Michigan Press.

- HOLLAND, J. H. (1995) *Adaptation in Natural and Artificial Systems*, MIT press.
- HONE, A. & KELSEY, J. (2004) Optima, Extrema, and Artificial Immune Systems. *International Conference on Artificial Immune Systems, ICARIS*, 80-90.
- JANSEN, T. (1999) On Classifications of Fitness Functions. *Technical report*, 1-18.
- KALLEL, L., NAUDTS, B. & ROGERS, A. (2001) *Theoretical Aspects of Evolutionary Computing*, Springer-Verlag Berlin and Heidelberg GmbH & Co. K
- KELSEY, J. (2004) An Immune Inspired Algorithm for Function Optimisation. Msc Thesis, University of Kent at Canterbury.
- KELSEY, J. & TIMMIS, J. (2003) Immune inspired somatic contiguous hypermutation for function optimisation. *Genetic and Evolutionary Computation Conference; GECCO*, 2723, 207-218.
- KELSEY, J., TIMMIS, J. & HONE, A. (2003) Chasing chaos. *IEEE Congress on Evolutionary Computation; CEC*, 413-419.
- KIRKPATRICK, S., GELATT, C. D., JR. & VECCHI, M. P. (1983) Optimization by Simulated Annealing. *Science*, 220, 671-680.
- LAARHOVEN, P. J. M. & AARTS, E. H. L. (1987) *Simulated Annealing: Theory and Applications*, Springer.
- MITRA, D., ROMEO, F. & SANGIOVANNI-VINCENTEILI, A. (1985) Convergence and finite time behavior of simulated annealing. *24th Conference on decision and control*, 761-767.
- MÛHLENBEIN, H. (1991) Evolution in time and space - the parallel genetic algorithm. *Foundations of Genetic Algorithms*, 316-337.
- NEWBOROUGH, J. & STEPNEY, S. (2005) A generic framework for population-based algorithms, implemented on multiple FPGAs. *ICARIS*, 3627, 43-55.
- NIX, A. E. & VOSE, M. D. (1992) Modeling genetic algorithms with Markov chains. *Annals of Mathematics and Artificial Intelligence*, 5, 79-88.
- RADCLIFFE, N. J. (1991) Equivalence class analysis of genetic algorithms. *Complex Systems*, 5, 183-205.

- RECHENBERG, I. (1971) Evolutionsstrategie - Optimierung technischer Systeme nach Prinzipien der biologischen Evolution. PhD thesis, Technical University of Berlin.
- ROSIN-ARBESFELD, R., TOWNSLEY, F. & BIENZ, M. (2000) The APC tumour suppressor has a nuclear export function. *Letters to Nature*, 406, 1009-1012.
- RUDOLPH, G. (1994a) Convergence analysis of canonical genetic algorithms. *IEEE Transactions on Neural Networks*, 5, 96-101.
- RUDOLPH, G. (1994b) Convergence of non-elitist strategies. Proceedings of the First IEEE International Conference on Evolutionary Computation, 1, 63-66.
- RUDOLPH, G. (1998) Finite Markov Chain Results in Evolutionary Computation: A Tour d'Horizon. *Fundamenta Informaticae*, 35, 67-89.
- SALOMON, R. (1996) Re-evaluating genetic algorithm performance under coordinate rotation of benchmark functions. A survey of some theoretical and practical aspects of genetic algorithms. *BioSystems*, 39, 263-278.
- SCHRAUDOLPH, N. N. & BELEW, R. K. (1992) Dynamic Parameter Encoding for Genetic Algorithms. *Machine Learning*, 9, 9-21.
- SCHUMACHER, C., VOSE, M. D. & WHITLEY, L. D. (2001) The no free lunch and problem description length. *Proceedings of the Genetic and Evolutionary Computation Conference*, 565-570.
- SECKER, A., DAVIES, M., FREITAS, A., TIMMIS, J., CLARK, E. & FLOWER, D. (2008) An Artificial Immune System for Evolving Amino Acid Clusters Tailored to Protein Function Prediction. *ICARIS, LNCS 5132*, 242-253.
- SENETA, E. (1981) *Non-Negative Matrices and Markov Chains.*, New York, Springer-Verlag.
- SPEARS, W. M. & DE JONG, K. A. (1997) Analyzing GAs using Markov models with semantically ordered and lumped states. *Foundations of Genetic Algorithms*, 4, 85-100.
- STEPNEY, S., SMITH, R., TIMMIS, J., TYRRELL, A., NEAL, M. & HONE, A. (2005) Conceptual Frameworks for Artificial Immune Systems. *International Journal of Unconventional Computing*, 1, 315-338.
- TIMMIS, J. & ANDREWS, P. (2007) An Introduction To Artificial Immune Systems: A Beginner's Guide. *In Silico Immunology*, 47-62.

- TIMMIS, J., ANDREWS, P., OWENS, N. & CLARK, E. (2008a) Immune Systems and Computation: An Interdisciplinary Adventure. *Unconventional Computation*, LNCS 5204, 8-18.
- TIMMIS, J., ANDREWS, P., OWENS, N. & CLARK, E. (2008b) An Interdisciplinary Perspective on Artificial Immune Systems. *Evolutionary Intelligence*, 1, 5-26.
- TIMMIS, J., HONE, A., STIBOR, T. & CLARK, E. (2008c) Theoretical advances in artificial immune systems. *Theoretical Computer Science*, 403, 11-32.
- TSITSIKLIS, J. N. (1985) Markov chains with rare transitions and simulated annealing. *LIDS-P-1497*, 1-23.
- VILLALOBOS-ARIAS, M., COELLO COELLO, C. A. & HERNÁNDEZ-LERMA, O. (2004) Convergence Analysis of a Multiobjective Artificial Immune System Algorithm. *Proceedings of ICARIS 2004*, LNCS 3239, 226-235.
- VOSE, M. D. (1991) Generalizing the notation of schema in genetic algorithms. *Artificial Intelligence*, 50, 385-396.
- VOSE, M. D. (1995) Modeling Simple Genetic Algorithms. *Evolutionary Computation*, 3, 453 - 472.
- VOSE, M. D. & LIEPINS, G. E. (1991) Punctuated equilibria in genetic search. *Complex Systems*, 5, 31-44.
- WEICKER, K. & WEICKER, N. (2003) Basic principles for understanding evolutionary algorithms. *Fundamenta Informaticae*, 55, 387-403.
- WOLPERT, D. H. & MACREADY, W. G. (1997) No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1, 67-82.