Helsinki University of Technology Laboratory for Theoretical Computer Science
Research Reports 74
Teknillisen korkeakoulun tietojenkäsittelyteorian laboratorion tutkimusraportti 74
Espoo 2002 HUT-TCS-A74

ASPECTS OF MODELLING AND SIMULATION OF GENETIC ALGORITHMS: A FORMAL APPROACH

Sam Sandqvist

TEKNILLINEN KORKEAKOULU
TEKNISKA HÖGSKOLAN
HELSINKI UNIVERSITY OF TECHNOLOGY
TECHNISCHE UNIVERSITÄT HELSINKI
UNIVERSITE DE TECHNOLOGIE D'HELSINKI

Helsinki University of Technology Laboratory for Theoretical Computer Science
Research Reports 74
Esbo 2002 HUT-TCS-A74

ASPECTS OF MODELLING AND SIMULATION OF GENETIC ALGORITHMS: A FORMAL APPROACH

Sam Sandqvist

Dissertation for the degree of Doctor of Science in Technology to be presented with due permission of the Department of Computer Science and Engineering, for public examination and debate in Auditorium T2 at Helsinki University of Technology (Esbo, Finland) on the 1st of November, 2002, at 12 o'clock noon.

Helsinki University of Technology
Department of Computer Science and Engineering
Laboratory for Theoretical Computer Science

Tekniska högskolan
Avdelningen för datateknik
Laboratoriet för databehandlingsteori

Distribution:
Helsinki University of Technology
Laboratory for Theoretical Computer Science
P.O.Box 5400
FIN-02015 HUT
Tel. +358-0-451 1
Fax. +358-0-451 3369
E-mail: lab@tcs.hut.fi

ISBN 951-22-6115-4
ISSN 1457-7615

Otamedia Oy
Esbo 2002

Abstract: Genetic algorithms (GAs) are widely used in solving search and optimisation problems involving very large search spaces, or very many variables where closed form solutions are impractical due to the very size of the problems. GAs have been modelled in various ways, from the seminal work by Holland describing an approach based on sets, to works based on Markov chains in the 1990s. This dissertation combines the two salient features of GAs, namely the temporal aspect of the evolutionary approach to solving problems at the heart of the GA, and the stochastic aspect of evolution arising from its reliance on basically random generation of new individuals with stringent selection in determining survival.

The work centres around describing the formal modelling of GAs using a logical approach based on standard first-order logic combined with temporal logic and with probabilistic logic. These logics are combined into a unified logic, temporal-probabilistic logic (TPL) which is formulated in this work.

The GA is then described using TPL as the main tool, and the working of the GA is detailed from its components to the actual processes by formulating a model of the GA. Several important parameters are described and analysed, as is the important mechanism of selection. A simple axiomatisation of the GA using TPL is described as well.

Also presented are simulation of the workings of the genetic algorithm based on high-level Petri nets and experimentation with a genetic algorithm package providing experimental evidence centring on the various selection mechanisms for some of the theoretical results.

Keywords: Genetic algorithm, temporal logic, probabilistic logic, modelling, simulation, Petri net

CONTENTS

List of Figures

List of Tables

PREFACE

The work covered in this thesis has been carried out over a number of years, 1994 — 2001, both at the Laboratory for Theoretical Computer Science (formerly the Digital Systems Laboratory) of the Helsinki University of Technology as well as on my own. I am very grateful to the former head of the laboratory, Professor Leo Ojala, for providing excellent assistance and unwavering support during this long period, without which it would not have been realised.

This work has grown over the years from a simple idea – that of using temporal and probabilistic logic in modelling genetic algorithms – to a full-blown work detailing the intricacies of the logic itself to its application in modelling genetic algorithms, even their axiomatisation. As such it has continued to amaze and humble me; the things that can be done with logic and genetic algorithms are ubiquitous.

I also wish to thank several other people who have been instrumental in making this work a reality by providing helpful comments and assistance. Foremost among these stands Jarmo Alander, my erstwhile mentor who introduced me to the exciting world of genetic algorithms in 1991, and of course my colleagues at the laboratory, among whom especially Tomi Janhunen has been helpful in checking my syntax, and Nisse Husberg for his good humour and cheer.

Finally, my biggest thanks go to my wife, Stina for her support, encouragement and patience over the years. It has not been easy, and do I know it.

Sibbo, August 2002

Sam Sandqvist

1 INTRODUCTION

It is generally accepted that all biological development in nature follows general principles originally laid down so eloquently by Charles Darwin in his seminal work *On the Origin of Species by Natural Selection* in 1859 [Dar59, p. 127]:

> If during the long course of ages and under varying conditions of life, organic beings vary at all in the several parts of their organisation, and I think this cannot be disputed; if there be, owing to the high geometric powers of increase of each species, at some age, season, or year, a severe struggle for life, and this certainly cannot be disputed; then, considering the infinite complexity of the relations of all organic beings to each other and to their conditions of existence, causing an infinite diversity of structure, constitution, and habits, to be advantageous to them, I think it would be a most extraordinary fact if no variation ever had occurred useful to each being's own welfare, in the same way as so many variations have occurred useful to man. But if variations useful to any organic being do occur, assuredly individuals thus characterised will have the best chance of being preserved in the struggle for life; and from the strong principle of inheritance they will tend to produce offspring similarly characterised. This principle of preservation, I have called, for the sake of brevity, Natural Selection.

This, following Dennett, Darwin's dangerous idea [Den95], forms the basis for so-called genetic algorithms as well – suitably substituting organic being with genetic algorithm, natural with artificial, and so forth. The 'struggle' for life within these algorithms takes place inside our computers, and we, the creators of the algorithms, not nature, supply the conditions for survival.

1.1 THE GENETIC ALGORITHM

Following Holland's vivid exposition [Hol92b], we may remark that living organisms are consummate problem solvers. They exhibit a versatility that puts the best computer programs to shame. This observation is especially true for computer scientists, who, often having spent an inordinate amount of time on a thorny problem, discover that nature has bred a solution and organisms come by it through the apparently undirected mechanism of evolution and natural selection.

We see evolution's remarkable power as something to be emulated rather than envied. Natural selection eliminates one of the greatest hurdles of software design: specifying in advance all the features of a problem and the actions a program should take in dealing with them. By harnessing the mechanism of evolution, we may be able to 'breed' programs that solve problems even when no person can fully understand their structure. Indeed, these so-called genetic algorithms have already demonstrated the ability to make breakthroughs in the design of such complex systems as, for example, jet engines [Hol92b].

As pointed out by e.g. Goldberg [Gol89] in addition to Holland [Hol92b], genetic algorithms make it possible to explore a far greater range of potential solutions to a problem than do most conventional algorithms, in the same amount of time. Most organisms evolve by means of two primary processes: natural selection and sexual reproduction. The first determines which members of a population survive to reproduce, and the second ensures mixing and recombination among the genes of their offspring. This mixing allows creatures to evolve much more rapidly than they would if each offspring simply contained a copy of the genes of each parent, occasionally modified by mutation.

Selection is simple: if an organism fails some test of fitness, such as recognising a predator and fleeing, it dies. Similarly, we have little trouble weeding out poorly performing algorithms. For instance, if a program is supposed to sort numbers in ascending sequence one need merely examine whether each entry of the program's output is larger than the previous one; if not, the algorithm is not working, and is discarded.

People have employed a combination of crossbreeding and selection for a long time to breed better crops, dogs and flowers. It is not easy, however, to translate these techniques to computer algorithms. The major problems are finding an equivalent to the 'genetic code' that can represent the structure of different solutions to the problem at hand, and finding a suitable way of measuring the quality of a solution (i.e., its fitness).

Genetic algorithms mimic evolution and natural selection. As such the major problem, as pointed out above, is one of encoding the problem into a form suitable for natural selection. Typically, such an encoding is either a binary string or array of floating point values embodying some aspects of the desired solution.

Recast in the language of genetic algorithms, the search for a good solution to a problem is a search for particular strings, binary or otherwise. The universe of all possible strings can be considered as an imaginary landscape; valleys mark the location of strings that encode poor solutions, and the landscape's highest point corresponds to the best possible string. The landscape metaphor is not new; it was originally used by Sewall Wright in his seminal paper on the roles of mutation, breeding and selection in 1932 ([Wri32], as cited by Dawkins [Daw96]).

Regions in the solution space can also be defined by looking at strings that have 1's and 0's in specified places; a kind of binary equivalent of map co-ordinates [Hol92b]. The set of all strings that start with a 1, for example, constitutes a region in solution space, in the set of all possibilities. According to Jones, this may often be thought of as a *landscape* as well [Jon95].

One conventional technique for exploring such a landscape is hill climbing: start at some random point, and if a slight modification improves the quality of your solution, continue in that direction, otherwise, go in the opposite direction. Complex problems, however, lead to landscapes with many high points. As the number of dimensions of the problem space increases, the size of the search neighbourhood also increases; the countryside may contain tunnels, bridges or even more exotic structures and convoluted topological features. Finding the right hill or even determining which way is up becomes increasingly difficult. In addition, such search spaces are enormous.

Genetic algorithms cast a net over this landscape. The multitude of strings in an evolving population samples many regions in the landscape simultaneously. Interestingly, the rate at which the genetic algorithm samples different regions corresponds directly to the region's average 'elevation'; that is, the probability of finding a good solution in that vicinity [Hol92b].

This ability of genetic algorithms to focus their attention on the most promising parts of a solution space is a direct consequence of their ability to combine strings containing partial solutions. This is because, using crossing operators, high-ranking strings encoding desirable characteristics are mated producing new, possibly even higher-ranking solutions. These offspring do not replace their parents; rather, they replace low-ranking solution candidates.

In this way the population of solution candidates slowly improves. Furthermore, mutations are allowed to occur with some very low probability; this provides insurance against one particular string becoming dominant, replacing every other string and stopping evolution altogether.

1.2 STRUCTURE AND CONTRIBUTIONS OF THIS WORK

This work will study the genetic algorithm, but from two particular viewpoints: that of time, i.e. temporal progression, and that of probability, i.e. stochastic characteristics. The intention is to gain an understanding of these central characteristics wherever applicable to the genetic algorithm, and to provide a formalisation of the genetic algorithm incorporating them.

The particular contributions of this work centre around a new paradigm for describing the workings of the genetic algorithm. The paradigm is that of using logic, specifically temporal and probabilistic logics in conjunction with traditional first order logic combined into a temporal-probabilistic logic in modelling the genetic algorithm, including both its central temporally progressive and probabilistic or statistical aspects into a comprehensive framework. In addition, the work also formulates a foundation for further studies in describing a simple axiomatisation of the genetic algorithm, in line with that of e.g. the formal axiomatisation of the theory of groups, as well as providing two different simulation approaches to genetic algorithms.

The overall structure of the thesis is made up of three major parts: first we have a general part (comprising chapters 1, 2, and 3) providing the basic descriptions of the genetic algorithm as well as of the formulations relating to the logical and mathematical foundations for the work. The second part centres around applying the formulations from the first part to genetic algorithms, and describing the genetic algorithm model itself. This part comprises chapters 4 and 5. The third part, consisting of chapters 6, 7, and 8, are essentially applications of, and excursions from the theory formulated in the previous parts, centring on the simulation of genetic algorithms from a net-theoretic as well as an experimental point of view.

Looking at the thesis from a chapter to chapter viewpoint, it proceeds as follows: chapter 2 provides an overview of the general concept of the genetic algorithm, sufficient for the easy understanding of the remainder of this work. It deals with the standard formulation of the genetic algorithm and briefly touches on the so-called schema theorem and building-block hypothesis which is quite often used to describe the genetic algorithm. Next, chapter 3 provides an overview of the logical foundations used in the rest of the work. It first describes the central tenets of the temporal logic used in the remainder of the formulation of the theory of genetic algorithms. This is a fairly standard exposition with some minor extension to make the logic formulation more suitable for modelling genetic algorithms as presented in this work. For instance, our temporal logic is based on intervals, which makes it particularly suited for describing the common case of generation-based genetic algorithms. We describe a natural extension of normal notation that has been introduced to allow the temporal operator $\bigcirc$ (next) take a superscript denoting future generations (and symmetrically, a negative superscript for past time). This has not been encountered before and represents a new formulation by the author. The chapter also introduces the second major tenet of this work: incorporating the probabilistic element of genetic algorithms. This we accomplish using a second tool, that of probabilistic logic by first providing an overview of probabilistic logic, and continuing in the same chapter, combining temporal and probabilistic logic in a formal framework. This particular formal framework has not been previously formulated, except by the author [San94b, San95], and represents a novel view of the combination of these logics. An overview is provided in that chapter; a fuller treatment is contained in appendix A, which formally describes the syntax and proof theory of the combined logic, and is one of the contributions of this work. The rationale for using interval temporal logic instead of point-based (integer) temporal logic is that it in a more natural way captures the ability of individuals in the genetic algorithm to survive (and thus live longer, or for several base intervals at a stretch), something that is not possible, or at least not as simple to describe, using other alternative formulations.

Continuing, chapter 4 introduces one of the major tenets and is one of the major contributions of this work: how temporal logic may be used to discuss the time dimension in genetic algorithms. It develops a novel description of the genetic algorithm incorporating time. This model, based on interval temporal logic has, as far as the author has been able to ascertain not been previously formulated (except by the author in several conference papers; see [San94b, San95, San96b, San96c, San96a]). Using temporal logic as a tool to explicitly incorporate time in the theory of genetic algorithms we may in a natural manner formulate temporal dependencies that are either not shown by or are obscured in other formalisms.

Furthermore, in chapter 5 we develop a formulation of the genetic algorithm based on the *combined* temporal and probabilistic logic. This, one of the main chapters of this work also includes novel formulations of a temporal selection mechanism. This mechanism is shown to provide abilities to avoid some of the more worrisome problems in practical genetic algorithms, which represents one of the contributions to the perennial problem of ensuring enough diversity in the genetic algorithm and has in this form not been published before. As an example, we may avoid or at least discern the onset of the problem of premature convergence. The novel mechanism employed now described by the author centres around the concept of *equality intervals*: fitness ranges where the differences are considered too small to matter, and where consequently other means should be taken into use to distinguish between the individuals within the range. This is in contrast to the methods usually employed to deal with this problem, as it instead of attempting to artificially enlarge the minute differences between the individuals in order to obtain a clear superiority/inferiority index recognises their equality and treats them in non-genetic ways.

A further contribution never encountered by the author before is represented by the next chapter: in order to show the power and expressibility of the temporal-probabilistic language formulated in this work it includes an exposition on axiomatisations of the genetic algorithm providing an interesting view of this approach. It shows several theorems derived and proved using the axioms and constitutes one of the main contributions of this work.

Furthermore, as a simulation of the workings of the genetic algorithm and as an interesting modelling approach, we also present the genetic algorithm modelled using a high-level Petri net. This is a very promising avenue which we have not seen published before; chapter 7 represents very much a beginning and further research is needed.

In order to test the practical implications of the theoretical framework, and especially the novel temporal selection mechanisms a number of simulations using empirical experiments based on a modified and freely available implementation of genetic algorithms [Hun95b, Hun95a] are described in chapter 8. In addition to the rather theoretical tests described, the well-known travelling salesperson problem (TSP) is solved using both the standard and a modified version of the genetic algorithm (with temporal selection). It is shown that whilst the temporal selection mechanisms may provide advantages in several instances, it is not a priori clear in which instances this would be the case. In particular, it seems in the light of the tests described here, that the TSP problem could benefit from temporal selection.

Finally, chapter 9 draws some general conclusions about the formulation of the model of genetic algorithms and the logics as well as their application to expressing a theoretical framework for genetic algorithms and shows why they are important contributions to the field.

1.3 RELATED WORK

Genetic algorithms may be modelled in many ways, taking different aspects of the algorithm under consideration. Following the seminal work by John Holland [Hol92a], originally published in 1975, the question of models in the field of genetic algorithms has been either been slanted toward a set-based or stochastic point of view.

For the description of genetic algorithms below, I have relied on many sources, e.g. [Gol89, Dav91a] as well as [Hol92b] and [Hol92a]. For a popular account in a mathematical setting, see [Pet90, p. 209-215]. A good introduction to the problems faced by simulation of evolution, and especially the fallacies and mistakes often encountered in these, see [Atm94].

This work builds on our previous work on genetic algorithms, especially in a database setting; see [San93a, San93b, San93c]. Initial work on modelling the genetic algorithm using temporal and probabilistic reasoning appeared in [San94b] and [San95].

During its history the genetic algorithm has been modelled in various ways. As previously cited, we have the work by Holland [Hol92a], a theory based on sets and probabilities; we have the work on deceptive problems by Goldberg [Gol87], continued by Vose and Liepins [VL91]. This work resulted in an exact model for a simple genetic algorithm in the form of a Markov chain, developed by Nix and Vose [NV92]. Davis independently developed a model based on Markov chains as well [Dav91b].

Subsequently, Vose developed the model by Vose and Nix to further tie the genetic algorithm to the infinite-population model [Vos91a].

Some other work, notably Booker [Boo93], Whitley [Whi93], and Vose [Vos93], has also considered different models of genetic algorithms and aspects of genetic algorithms. These have centred on stochastic (or probabilistic) views of the genetic algorithm, and not emphasised the time aspect (apart from the role it naturally plays in e.g. the Markov chain model). Reynolds and Gomatam provide a modern view of stochastic modelling of genetic algorithms in [RG96]. In [HB92, BS93] Bäck and others provide overviews of current evolutionary algorithms and their differences and similarities.

Some very interesting work detailing the theoretical limits of search and optimisation, with especial applicability to genetic algorithms, has been published by Wolpert and Macready [WM95, WM97]; including the so-called "No Free Lunch" theorem.

The landscape metaphor for search has been thoroughly explored by Jones in his dissertation [Jon95].

The Santa Fe Institute of Complexity Studies has published several interesting reports touching on search in general, for instance [WM95]; genetic algorithms and their characteristics, especially compared with hill-climbing techniques, have been covered in [MCH93, FM92, FM93, MH93]; genetic algorithms and artificial life [MF93, MCH93, MHC93], and so forth.

Regarding more introductory works, the well-known Genetic Algorithm Tutorial [Whi94], by Darrel Whitley is a good starting point for understanding and modelling the genetic algorithm. An introductory book by Melanie Mitchell [Mit96] contains much interesting material, as does the overview of the current state of genetic algorithm research by Bäck *et.al.* [BHS97]. Furthermore, a very interesting book by Michalewicz [Mic99] treats the genetic algorithm from both a theoretical and practical viewpoint, with the emphasis on the latter. Fairly recently, a survey of the theory of genetic algorithms also by Bäck *et.al.* [BdGKK97] appeared in the Bulletin of the European Association for Theoretical Computer Science.

2 OVERVIEW OF GENETIC ALGORITHMS

This chapter will introduce the genetic algorithm by providing a standard, simple exposition of the basic underlying concepts. It will introduce the two types of genetic algorithms, and show what distinguishes the two; and it will describe the well-known building-block hypothesis, with its schema theorem that is one of the foundational justifications for using genetic algorithms to solve problems.

2.1 ASPECTS OF GENETIC ALGORITHMS

The genetic algorithm is normally thought of as a *process*, i.e. an ordered progression of well-defined states, whereby solutions to set problems are evolved using a population of solution candidates. These candidates are checked using some fitness measure against the problem (i.e. evaluated) and, using some appropriate convergence or stopping criterion, the process is terminated. The process view has given rise to several models and analyses of genetic algorithms based on Markov processes (for a summary, see e.g. [RG96]).

There are two aspects of the genetic algorithm we wish to consider in this work, with the emphasis on the former:

- the *time* aspect, as exhibited by the population of the genetic algorithm during evolution from generation to generation, and
- the *stochastic* aspect, as exhibited when selecting and forming new members of a population.

An overview of the genetic algorithm is provided in figure 2.1. It shows two generations: a current one to the left, and a subsequent one to the right. A generation consists of three stages: a current (parent) population from which a selection takes place; a child population and the current population from which survival of individuals in the populations is determined; and a new population (together with a discarded population) from which a new generation is born.

Further introductions to genetic algorithms may be found in, for instance, [Dav91a] or [Gol89].

2.2 THE TIME ASPECT

There are many ways to approach the problem of time in a genetic algorithm. In general, time is thought of being a discrete, linear progression, starting from an arbitrary initial time point, and infinite in the positive direction. Formally, nothing prevents us from considering time as infinite in both directions; it is customary, however, to treat time as having a starting

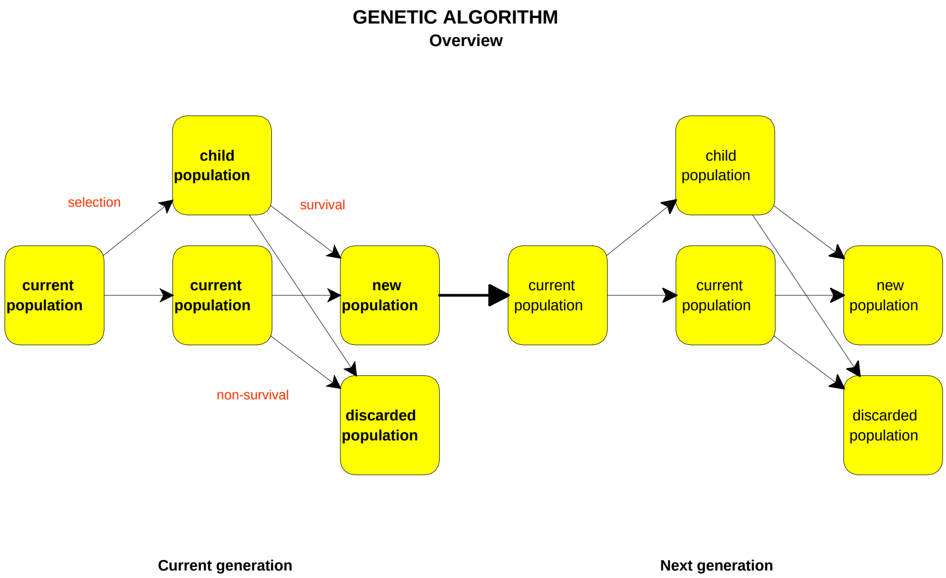

Figure 2.1: Genetic algorithm overview

point and thus bounded in one direction. We will return to this question later in this work, especially when considering the initial population, and the axiomatisation of the genetic algorithm.

In general, there are two ways of considering evolution in a genetic algorithm: the generational and steady-state model. These will be treated separately below.

2.2.1 The generational genetic algorithm

This, the conventional genetic algorithm, may be characterised by the following schematic procedure.

1. all individuals are randomly selected as parents (typically pairwise, but sometimes more),

2. offspring is generated using some genetic operators,

3. in some way the new population is chosen from the combination of the old population and the newly generated offspring so that the population size does not change,

4. the new population (and especially its main characteristic, the fitness of the population) is checked against the stopping criterion; the algorithm either stops or proceeds anew at step 1.

It should be noted that the above is very much a broad outline of the algorithm.

2.2.2 The steady-state genetic algorithm

This non-generational genetic algorithm proceeds schematically as follows.

1. some individuals are randomly selected as parents (again typically pairwise, but sometimes more),
2. offspring is generated using some genetic operators,
3. in some way new members of the population are chosen from the combination of the old parents, the rest of the population, and the newly generated offspring so that the population size does not change,
4. in the same way as in the generational case the new population is checked against the stopping criterion; the algorithm either stops or proceeds anew at step 1.

The critical difference is that the steady-state algorithm generates single (or at least very few, compared with the population size) new members, whereas the generational one renews the whole population every time (even if in some optimised case some members are chosen to survive, so-called elitism [Gol89]).

The time problem can be characterised as dealing with what may be known or determined about subsequent generations given a current generation knowing the overall characteristics of the genetic algorithm. How will fitness change, for instance? What characteristic features of the population will survive? What can be said about the optimum? How will the genetic algorithm converge? We will return to these questions below when we deal with the time model of genetic algorithms.

2.3 THE STOCHASTIC ASPECT

The genetic algorithm relies on random selection (or semi-random, based on some criteria, such as relative fitness, abundance, age, and so forth) of parents when forming a new generation. It also relies on random techniques (again, semi-random is often used as well, or even deterministic) when determining how to combine the parents' genetic material when forming the child individual (or individuals).

In both of these cases there is a certain probability that a certain individual, a certain probability that a certain gene of the individual, will be selected for, and consequently present in a subsequent generation. This gene (or more often, chromosome) will determine a trait that is selected for, and determines a desired characteristic of the solution the algorithm is set out to provide.

The stochastic problem is to determine what these probabilities are, what the probability of the genetic algorithm reaching optimum is, and the probability of any single trait being present in any particular generation and individual. The overwhelmingly most common method to examine the genetic algorithm in stochastic terms is to use Markov chains. Many researchers have used them to determine convergence of simple genetic algorithms, e.g. Mahfoud in [Mah93], or Suzuki in [Suz93]. However, as emphasised by Suzuki, only the very simplest of genetic algorithms may be successfully modelled using Markov chains due to the complexity involved. For a summary of work modelling genetic algorithms with Markov processes see [RG96].

2.4 THE SCHEMA THEOREM

This theorem, which was originally formulated by Holland in 1975 (reprinted as [Hol92a]), reformulated by Goldberg [Gol89], and worked on by many researchers, such as Vose [Vos91b], shows how the fitness evolves from generation to generation. Central to understanding the schema theorem is the notion of a schema, introduced by Holland. The following discussion will endeavour to present the salient points of schemata and the schema theorem without worrying too much that we are using terms and concepts not formally defined yet.

Without loss of generality, assume for the present discussion that a genetic algorithm works with fixed-length chromosomes ξ consisting of elements from {0, 1}. This means that the structures are encoded into simple yes/no (true/false) answers, which is perfectly feasible for any genetic algorithm, although not nearly always the most efficient way of representing the structure [Vos91b]. Anticipating the formal definition (see equation 4.2 on page 32) we denote the set of all possible strings $\mathcal{S}_\xi$. This also means that the powerset $2^{\mathcal{S}_\xi}$ contains all possible sets of strings (i.e. all possible chromosomes). Some interesting points recently formulated regarding the cardinality (of the symbols used) of schemata by Fogel and Ghozeil show that there does not seem to be any advantages to using an alphabet of a specific cardinality [FG97].

In order to proceed we first provide a definition of a schema as used in the genetic algorithm.

Definition 1 *A schema $\mathcal{H}$ is a string notation for a member of $2^{\mathcal{S}_\xi}$ formed from the alphabet {0, 1, #}, where 0 and 1 are binary digits, and # is taken to be a "don't care" situation. Given a chromosome ξ formulated as a string with binary digits it belongs to the schema $\mathcal{H}$ iff the corresponding positions in the schema $\mathcal{H}$ and the string are either identical, or the schema $\mathcal{H}$ has a # in that position.*

In order to fully understand this definition we may study the following example.

Example 1 *An example, taken (and corrected) from [Ala92, p. 25] shows this clearly. Let $\mathcal{S}_\xi$ be all strings of length 8 (i.e. chromosomes have a fixed length of 8 bits), to which belong, among others, the strings $s_1 = 00110011$, $s_2 = 00111100$, $s_3 = 00111011$, and $s_4 = 00101101$.*

Let the schema $\mathcal{H}_1$ be the string representation $0011\#\#11$*. In this case* s_1 *and* s_3 *belong to the schema. Formally,*

$$\{s_1, s_3\} \subset \mathcal{H}_1 = \{00110011, 00110111, 00111011, 00111111\} \in 2^{\mathcal{S}_\xi} \quad (2.1)$$

According to [Vos91b], the main advantage of using schemata is that the expectation value of the improvement from generation to generation in the genetic algorithm may be calculated. Simplifying the discussion (for a more rigorous treatment, see e.g. [San93b]), we can say that the evolution of the population, seen from a schema point of view, depends on the strength of $\mathcal{H}$ or its average fitness with respect to the whole population Π. This implies that in order for the average fitness of the whole population to increase we must ensure that the strength of the schema does not decrease. Since due to the application of genetic operators there are in general two forces acting on a schema (and thus its strength), a strengthening force building up the schema (e.g. recombination), and a destructive force attempting to tear it apart (e.g. mutation), we must thus ensure that destruction does not overcome construction. In short, Vose's schema theorem says that if the strength of a schema $\mathcal{H}$, or the average schema fitness is above the average fitness of all possible strings, then $\mathcal{H}$ will predominate over lower-strength schemata in future generations, provided that the probability of schema break-up is not too large. Especially short schemata (i.e. with many "don't care" bits #) tend to, having been generated, stay around. It is from these that good solutions (structures) are found using genetic operators, like crossover and others.

It is also evident that, whilst the schema theorem does show how the genetic algorithm behaves when examining two consecutive generations, it is dependent on several assumptions. Chief among these is the assumption that the genetic algorithm works by assembling short 'building blocks', i.e. schemata, into larger, by assumption fitter structures, at least from the genetic algorithm's viewpoint, i.e. that viewpoint of the problem that the genetic algorithm was designed to solve in the first place. However, being larger they tend to be destroyed by the action of the crossover or recombination operator (see below). Thus it is not at all clear that the genetic algorithm works, or should work beyond the very first stages, if we only examine it in the light of the schema theorem.

Note that, as the schema theorem is not essential for our temporal and probabilistic treatment of the genetic algorithm we will not use it in our formalisation of the algorithm in what follows in this work; however, we do mention it to contrast it with our approach.

The problem with the Schema Theorem is not that isn't true, but that it is *too* true. Which is to say, it applies not only to schemata but to arbitrary subsets of the search space, regardless of the fitness function.

If one translates it into its physics equivalent, the Schema Theorem says only that, adjusting for particles lost by decay, the centre of mass of a cloud of particles moves at the weighted average velocity of the particles, and the centre of mass of any subset of that cloud moves at the weighted average velocity of that subset.

As Lee Altenberg so succintly says, the schema theorem has nothing at all to say about a GA's performance - there has to be some "knowledge" about the search space embedded in the relationship between the operators, representations, and fitness in order for a GA to perform better than average; see [Alt95] in [WV95]. The need for implicit knowledge was proved in masterful generality by Wolpert and Macready in [WM95] in their famed "No Free Lunch" theorem.

3 LOGICAL FOUNDATIONS

This chapter introduces the logical foundations for this work. It proceeds by first describing temporal logic using standard formulations from e.g. van Benthem [Ben83]. We also describe some shorthand notations useful for our work in describing the genetic algorithm using temporal logic. Next, the chapter describes probabilistic logic. This logic is almost wholly based on that of Bacchus [Bac90]. We briefly summarise his two main branches of probabilistic logic, i.e. propositional and statistical probabilistic logic. We then combine the two (together with standard first order logic) into a unified whole, temporal probabilistic logic, for which we provide an overview in this chapter. It is more formally detailed in appendix A.

3.1 TEMPORAL LOGIC (TL)

As Masini [Mas93] explains, temporal logic may be thought of as a modal logic in which we place further restrictions on modal interpretations. The modal logic employed is expanded from conventional first-order logic with modal operators, giving the accessibility relation between states (or possible worlds).

In this work we will rely on an integer, linear temporal structure, with intervals, as well as temporal operators defined on the structures. In this chapter we will first define and discuss the temporal structures we are going to use. Then we will summarise the temporal operators.

We have chosen to use a temporal logic based on intervals instead of integer points because in our view it is more natural; also, since we use intervals to calculate with the lifetimes of individuals it is much simpler to use intervals. For instance, two individuals in a population may have overlapping lifetimes (intervals), where the beginning and end points may be different (although some generation, or generations, is common to both). In this way it is easy to compare ages and survivability of individuals in a formal setting.

There are many ways of defining a suitable temporal logic on intervals; the classical way, following van Benthem [Ben83], the logic based on the *meeting* operator as defined by Allen and Hayes [AH85, AH87, AH89], and the Tsang approach [Tsa87], roughly similar to van Benthem's. However, as Hajnicz states in [Haj95], the classical approach still offers the best axiomatisation, and richest structures for most applications. Accordingly, our approach closely follows van Benthem.

We will not go into too many details, such as a rigorous, axiomatic model, regarding temporal logic at this stage as it is not essential for appreciating the main thrust of this work. More details can be found in van Benthem, Shoham [Sho88], Manna and Pnueli [MP91], as well as Ostroff [Ost89]. For a proof theory of modal logics, applicable to temporal logic, see e.g. Masini [Mas93]. Modal logic, as generally applicable as well as specifically applicable to temporal settings, may be consulted in Chellas' work [Che80].

3.1.1 Time structure

We need to define the basic temporal structures in genetic algorithms, specifically in terms of the aspects we are emphasising. We begin with the central concepts of *time* and *interval*.

Definition 2 (Time) *In a genetic algorithm, time is seen as a discrete set of points. The algorithm proceeds from a time point t_0 when the population is said to be initial.*

We rely on the standard model of time; see e.g. van Benthem [Ben83, Ben88] given by

$$\langle T, <, V \rangle \tag{3.1}$$

consisting of a "flow of time" $\langle T, < \rangle$, i.e. a set of "moments" ordered by "earlier than" or "before", and a "valuation" V. We defer the discussion of what is actually valuated until a later time.

Note that the internal structure of time is not specified: it may be linear or branching. However, the view of time in this work is one modelled on the integers, $\mathbb{Z}$, and is linear, not branching.

It should be noted that t_0 is in no way special: time may be seen as infinite in both directions as we may simply define the population of the genetic algorithm as static during all time points before t_0.

Following van Benthem [Ben83, p.60], we define an interval.

Definition 3 (Interval) *An interval is a time duration between two time points, the start and end time, respectively. It is defined as follows*

$$I =_{df} [m_1, m_2] =_{df} \{m \in \mathbb{Z} \mid m_1 \leq m \leq m_2\} \tag{3.2}$$

where $m_1, m_2 \in \mathbb{Z}$ and $m_1 < m_2$.

It should be noted that van Benthem first defines these as open intervals of rational numbers, and later mentions the integer case, as described above, as a special case; we also disallow intervals of the form $[m, m]$, which he allows.

We denote the set of all intervals $\mathcal{I}$, i.e. $I \in \mathcal{I}$.

We also define the interval, or period structure INT($\mathbb{Z}$), van Benthem [Ben83], definition I.3.1.2. Note that since van Benthem initially uses the rational numbers, $\mathbb{Q}$ whereas we use integers, some of the definitions below take on a more complex shape than strictly necessary. We have noted where this is the case; however, we wish to stay close to van Benthem and let the more complicated definitions stand.

Definition 4 (Interval structure) *The interval structure INT($\mathbb{Z}$) is the tuple*

$$\langle \mathcal{I}, \subseteq, < \rangle \tag{3.3}$$

where $\mathcal{I}$ consists of all non-empty closed integer intervals $[m_1, m_2]$, $\subseteq$ is set-theoretic inclusion, and $<$ is defined by setting $[m_1, m_2] < [m_3, m_4]$ if $m_2 \leq m_3$.

Note that we, in line with the above definition, intend that

$$[m_1, m_2] > [m_3, m_4] \quad \Leftrightarrow \quad [m_4, m_3] < [m_1, m_2]. \tag{3.4}$$

Furthermore, if $I = [m_1, m_2]$ and $J = [m_3, m_4]$ then

$$I < J \quad \text{if} \quad m_2 \leq m_3 \tag{3.5}$$

$$I \leq J \quad \text{if} \quad I < J \quad \vee \quad I = J \tag{3.6}$$

The intervals are of course equal if $m_1 = m_3$ and $m_2 = m_4$.

We also need to differentiate between intervals that may contain subintervals and those that may not.

Definition 5 (Base interval) *An interval of the form*

$$I = [m, m+1] \tag{3.7}$$

is a base interval.

van Benthem calls these atomic, or minimal intervals, although he does not deal with them further.

As should be clear from the above, the base interval may be stated in another more general way as well:

$$\nexists m, m_1 < m < m_2 \tag{3.8}$$

where $m \neq m_1, m \neq m_2$. This is van Benthem's usage.

The van Benthem definition could be said to be preferable because of its easy generalisation to more complex intervals, such as those based on rational numbers or additional dimensions. However, it is unnecessary in our setting since we do not extend our intervals in this fashion.

All other intervals may be called *composite*. It should also be noted that intervals are *convex*, i.e. uninterrupted, as pointed out by van Benthem. We omit the specifier base or composite whenever the distinction does not matter.

The definition of intervals is similar to Schwartz, *et. al.* [SMSV83], Shoham [Sho88] and Sandewall [San94a]. These characteristics of time are also essentially the same as that of Jensen and Snodgrass in [JS94]. There the authors define *transaction time* as a self-generated timestamp when a relation is added or deleted from a database; in our setting, when an individual is 'born' or 'dies', i.e. the start and end points of the interval, respectively, within which it has a non-zero probability (see below). This corresponds to Jensen's and Snodgrass' *existence interval*, which for an item e is $[tt_e^{\vdash}, tt_e^{\dashv}]$ where tt is the transaction time [JS94, p. 956]. To reason about relations (items) in a temporal setting, the authors also define the concept of *valid time*, i.e. the time when the item is valid. This time is a varying attribute of the relation, and bears no necessary relationship to the transaction times; we will not use valid time, since our individuals only 'exist' within the interval.

In the genetic algorithm we use time intervals to separate generations of parent and child individuals and populations. One generation corresponds to a base interval.

	Future operators		Past operators
$\Box p$	henceforth p	$\overline{\Box} p$	always in the past p
$\Diamond p$	eventually p	$\overline{\Diamond} p$	sometimes in the past p
$p \mathcal{U} q$	p until q	$p \mathcal{S} q$	p since q
$p \mathcal{W} q$	p waiting-for (unless) q	$p \mathcal{B} q$	p back-to q
$\bigcirc p$	next p	$\overline{\bigcirc} p$	previously p
		$\tilde{\overline{\bigcirc}} p$	weak previously p

Table 3.1: Temporal operators. p and q are general propositions.

3.1.2 Temporal operators

Following van Benthem we choose a basic paradigm for the semantics of intervals in order to be able to reason within the new formalism. We first define the binary operations for inclusion and overlap.

Definition 6 (Inclusion) *Given two intervals $I_1 = [m_1, m_2]$ and $I_2 = [m_3, m_4]$ we say that I_1 includes I_2 when the starting point m_3 is not less than m_1 and at the same time the end point m_4 is not greater than m_1. Denoting inclusion by the symbol $\sqsubseteq$ we have formally*

$$I_2 \sqsubseteq I_1 \text{ iff } m_1 \leq m_3 \ \wedge \ m_2 \geq m_4. \tag{3.9}$$

Note that we follow van Benthem strictly here; in our setting we could simply equate $\sqsubseteq$ with $\subseteq$, and thus dispense with this definition altogether, and rely on definition 4. We choose to follow van Benthem's usage here; the reader may treat $\sqsubseteq$ as synonymous for $\subseteq$.

Definition 7 (Overlap) *The binary relation overlap O between two intervals I_1 and I_2 is defined by*

$$I_1 \ O \ I_2 =_{df} \exists I . I \sqsubseteq I_1 \wedge I \sqsubseteq I_2 \tag{3.10}$$

In order to reason within our logic we need to define some basic operators, in addition to the standard first-order logic with its boolean operators $\vee$ (disjunction), $\wedge$ (conjunction), $\neg$ (negation), $\rightarrow$ (implication) and $\leftrightarrow$ (equivalence). We also have the first order operators $\exists$ (existence), and $\forall$ (universality).

Following Manna and Pnueli [MP91, MP93], the temporal operators in table 3.1 are sufficient for our purposes.

In addition, we have the precedence $<$, overlap O and inclusion $\sqsubseteq$ operators for intervals.

Note that some of these operators are 'weak' versions: waiting-for $\mathcal{W}$ is a weak version of the until $\mathcal{U}$ operator which allows for the possibility that q never occurs and, then, p holds at the interval and all intervals beyond. Likewise, back-to $\mathcal{B}$ is a weak form of the since $\mathcal{S}$ operator which allows the possibility that q never occurs and, then, p holds at the interval and all preceding intervals.

Also note that the henceforth $\Box$ and eventually $\Diamond$ operators are reflexive, i.e. p may hold 'now'. Some authors, e.g. Strulo, Gabbay, and Harrison [SGH95] take these as nonreflexive, and define reflexive versions of $\Box$ and $\Diamond$ separately ($\widehat{\Box}$ and $\widehat{\Diamond}$, respectively). Their version of $\Box p$, for instance, is equal to our $\bigcirc\Box p$.

The weak previously operator $\widetilde{\overline{\bigcirc}}$, although defined by Manna and Pnueli, is unnecessary in our setting. It allows for the case where we are at the very 'first' instant of time, where no previous points exist. Our setting is infinite time (i.e. infinite in both past and future directions), where no distinguished start time is used, so weak previous $\widetilde{\overline{\bigcirc}}$ is simply the same as previous $\overline{\bigcirc}$.

We will find a straightforward extension to the next operator $\bigcirc$ and previous operator $\overline{\bigcirc}$ useful. We write $\bigcirc^n$ for n applications of the next operator, i.e.

$$\bigcirc^n =_{df} \bigcirc\bigcirc\ldots\bigcirc \;\; n \text{ times} \tag{3.11}$$

with the analogous definition of $\overline{\bigcirc}^n$ as n times application of $\overline{\bigcirc}$. Given the semantic meaning of these operators, we may also use negative exponents by defining

$$\overline{\bigcirc}^{-n} =_{df} \bigcirc^n \tag{3.12}$$

$$\bigcirc^{-n} =_{df} \overline{\bigcirc}^n \tag{3.13}$$

which is a very useful notation.

However, Manna and Pnueli define the operators for *point* times only. In our work we use intervals, and need to show that the operators may be used on intervals as well.

Let us somewhat informally show this for the future operators; the exact syntax and semantics of these are fully described in Appendix A. Take the henceforth operator $\Box$ first. Given our model INT($\mathbb{Z}$) of intervals (definition 4), we need to show that it gives a meaning to the validation formula

$$(INT(\mathbb{Z}), I) \models \Box p \tag{3.14}$$

where p is a proposition holding in the interval I.

In order for this to be valid, the proposition p must be valid from the *beginning* of the interval I and onwards. Using a construction similar to Manna and Pnueli, and van Benthem [Ben83, p.9] this means that we may formulate the following definition.

Definition 8 (Henceforth) *The henceforth operator $\Box$ may be defined as the operator fulfilling the following condition.*

$$(INT(\mathbb{Z}), I) \models \Box p \iff \forall J \geq I,\ (INT(\mathbb{Z}), J) \models p \tag{3.15}$$

Note that we will actually follow an alternate path for defining operators (and indeed the whole logic); for more details, see Appendix A.

Likewise, for the eventually $\Diamond$ operator:

Definition 9 (Eventually) *The eventually operator* $\Diamond$ *may be defined as the operator fulfilling the following condition.*

$$(INT(\mathbb{Z}), I) \models \Diamond p \Longleftrightarrow \exists J \geq I,\ (INT(\mathbb{Z}), J) \models p \tag{3.16}$$

The until $\mathcal{U}$ operator is slightly more complicated and needs two conditions in its definition.

Definition 10 (Until) *The until operator* $\mathcal{U}$ *may be defined as the operator fulfilling the following condition.*

$$\begin{aligned}(INT(\mathbb{Z}), I) \models p\,\mathcal{U}\,q \quad &\Longleftrightarrow \quad \exists J \geq I,\ (INT(\mathbb{Z}), J) \models q \\ \text{and} \quad & \quad\ \ \forall K, I \leq K < J,\ (INT(\mathbb{Z}), I) \models p\end{aligned} \tag{3.17}$$

In order to simplify the corresponding conditions for the next $\bigcirc$ operator, we define the concept of a *next* interval.

Definition 11 (Next interval) *The next base interval* J *from an interval* I *is the one that satisfies*

$$\begin{aligned} I \quad &< \quad J \\ &\nexists \quad K, I < K < J \end{aligned} \tag{3.18}$$

where K *is an interval.*

Analogously, we define the *previous* interval as follows.

Definition 12 (Previous interval) *The previous base interval* J *from an interval* I *is the one that satisfies*

$$\begin{aligned} J \quad &< \quad I \\ &\nexists \quad K, J < K < I \end{aligned} \tag{3.19}$$

where K *is an interval.*

We denote the next (base) interval from I with a superscript $I^{\bigcirc}$, and the previous with $I^{\overline{\bigcirc}}$. Note that the restriction to base intervals is necessary due to the definition of interval precedence in Definition 4.

Note that the above definitions are *not* dependent on working in $INT(\mathbb{Z})$. If we restrict ourselves to this (and forego the generalisation option) we can instead make use of the interval definitions restricted to integers and write

$$I^{\bigcirc} = [m_1, m_2]^{\bigcirc} = [m_2, m_2 + 1] \tag{3.20}$$

$$I^{\overline{\bigcirc}} = [m_1, m_2]^{\overline{\bigcirc}} = [m_1 - 1, m_1] \tag{3.21}$$

making use of the integer characteristics of the intervals.

Finally, the definition for the next operator $\bigcirc$ now becomes simple.

Definition 13 (Next) *The next operator* $\bigcirc$ *may be defined as the operator fulfilling the following condition.*

$$(INT(\mathbb{Z}), I) \models \bigcirc p \iff (INT(\mathbb{Z}), I^{\bigcirc}) \models p \tag{3.22}$$

Since we will be defining these operators in another way in Appendix A, we are satisfied that the past operators are defined analogously.

3.2 PROBABILISTIC LOGIC (PL)

Traditional first-order logic is incapable of reasoning with uncertain or deficient knowledge, or applications exhibiting stochastic or random behaviour. Probabilistic logic provides a framework for such reasoning, e.g. when modelling the genetic algorithm. The following overview is based on Bacchus [Bac90].

In probabilistic logic two key concepts need to be understood: *statistical* and *propositional probability*.

- Statistical probability is used to reason about sets of individuals; they relate properties, not particular individuals. For example, the probability of a 40-year old smoker contracting lung cancer.
- Propositional probabilities are probabilities attached to propositions about particular individuals. For example, the probability that John, a particular 40-year old smoker, will contract lung cancer.

Consider the statement

$$\text{“}\textit{More than 75\% of all birds fly.}\text{”} \tag{3.23}$$

This is a statistical assertion about the proportion of fliers among birds.

On the other hand,

$$\text{“}\textit{The probability that Tweety flies is} > 0.75\text{”} \tag{3.24}$$

is an assertion about a degree of belief; its truth is determined by the subjective state of the agent who made the statement. This is a propositional probability, relating the degree of belief of a particular individual having a particular property.

What is important is to connect the two interpretations.

Direct inference is a technique for inferring propositional probabilities from statistical information. In other words, by examing the environment, as described by statistical information, we may infer probabilities to properties of particular agents in the environment. Furthermore, a powerful system of defeasible, or non-monotonic, reasoning may be developed.

3.2.1 Propositional Probabilities

Propositional probabilities are probabilities assigned to particular propositions or assertions. It is not *a priori* clear how to assign probabilities to logical formulas in a consistent and useful manner. One way of doing it is sketched below.

- With the standard notion of truth values (*true*,*false*) attached to formulas (e.g. α and β) we can group the formulas into equivalence classes (e.g. $[\alpha]$ and $[\beta]$); if two formulas have the same truth value under all possible valuations they fall in the same class.

- The normal logical connectives act as operators over the equivalence classes forming a Boolean algebra.
- We may define a function over the algebra by observing that the algebra contains a partial order defined as $[\alpha] \leq [\beta]$ iff every truth valuation which assigns $true$ to α also assigns $true$ to β.
- This means that there is a unique smallest element, the equivalence class of all unsatisfiable formulas, and a unique greatest element, the class of tautologies; call these [0] and [1], respectively. Note that sometimes these are denoted $[\perp]$ and $[\top]$, respectively.

A probability function μ may then be defined in the standard way.

Definition 14 *A propositional probability function is a mapping μ satisfying the following criteria*

$$0 \leq \mu([\alpha]) \leq 1 \tag{3.25}$$
$$\mu([0]) = 0 \text{ and } \mu([1]) = 1 \tag{3.26}$$
$$\mu([\alpha \vee \beta]) = \mu([\alpha]) + \mu([\beta]) - \mu([\alpha \wedge \beta]) \tag{3.27}$$

In order to simplify matters, we say that the *probability* of a formula α is $\mu([\alpha])$ which is denoted by $\mathbf{prob}(\alpha)$ in the logical languages being defined.

Note that we introduce the notation **prob** because we will later extend μ; see Appendix A. For more details regarding the function μ see Bacchus [Bac90].

Definition 15 *The* language *generated by the logic is called the language of propositional probability, $\mathcal{L}^{prop}$. This consists of first-order predicate logic with the added* **prob** *operator together with arithmetics to handle the numeric values.*

Note that this section, and only this section, used the standard way of denoting equivalence classes, i.e. the class between '[' and ']' characters. We will be using (following Bacchus) this notation for statistical probablities below. We hope this will not lead to undue confusion.

3.2.2 Probabilities over Possible Worlds

One can also place a probability distribution over a field of sets of interpretations or truth valuations of the language. These different interpretations can be viewed as being different possible worlds. With a probability distribution over possible worlds we can assign a probability to every formula; it is the proportion of the possible worlds satisfying the formula. Conversely, wishing to indicate uncertainty with a probability of a proposition being true, we may view it as being part of a set of possible worlds, giving the indicated probability as indicated above. The way this is accomplished is captured in the definition of μ; it is extended to encompass both worlds (using $\mu_{\mathcal{S}}$) and objects (using $\mu_{\mathcal{O}}$). For more details, see Appendix A where we will give the our logics a possible world semantics, and assign a probability distribution to them.

As indicated above, in order to formalise the notion and use the logic of probability we extend the conventional first-order logic with a probability operator denoting the probability of a formula α by $\mathbf{prob}(\alpha)$.

We also find it convenient to denote certainty as follows.

Definition 16 *A formula α is* certain *when its probability is 1.*

$$\mathbf{cert}(\alpha) =_{df} \mathbf{prob}(\alpha) = 1 \tag{3.28}$$

According to Bacchus [Bac90, p. 69], it can be shown that the modal logic developed for propositional probabilities is a generalisation of ordinary Hintikka-style logics of belief [Hin62].

3.2.3 Statistical Probabilities

Typically statements of statistical probability make assertions about the proportion of individuals from a particular set that are members of some other set; e.g. the proportion of birds that fly. We may also view this as attributing a property to a proportion of individuals in a set with a certain probability.

The difference compared with propositional probabilities is that the statistical probability operator must specify a set of placeholder variables – we are not talking about a particular individual, but about a set of individuals.

We may define the statistical probability formally as follows.

Definition 17 *Let α be a well-formed formula and $\vec{x}$ a vector of n object variables $\langle x_1, x_2, ...x_n \rangle$. Then $[\alpha]_{\vec{x}}$ is the* statistical probability *of α with respect to the variables given.*

Note that the index used is a vector of placeholder variables; however, very often we will have only one variable and will use a non-vector notation for this. In other words, if $\vec{x} = \langle x \rangle$ we will write $[\alpha]_{\vec{x}} = [\alpha]_x$ instead. We hope this will not lead to undue confusion.

We also need a measuring function, known in statistics and probability theory as a random variable [Pap65]. These are used to map individual objects (or properties of objects) to real numbers in order to discuss these objects or properties; e.g. **weight**(x) would give us the weight of x in some convenient unit [Bac90, p.84f] .

Note that we use the standard notation for conditional probabilities below; see definition 65 on page 136.

Example 2 *For example,* "Most birds can fly" *would be*

$$[\mathit{fly}(x) \mid \mathit{bird}(x)]_x > 0.5$$

We can also reason within the formalism.

Example 3 *Given that*

$$\text{“}Most\ birds\ fly\text{”}$$
$$\text{“}Penguins\ do\ not\ fly\text{”}$$

we find that

$$\text{“}Most\ birds\ are\ not\ penguins\text{”}$$

as follows

$$\begin{array}{ll} \vdash & ([\mathit{fly}(x) \mid \mathit{bird}(x)]_x > 0.5 \wedge \forall x.\mathit{penguin}(x) \rightarrow \neg \mathit{fly}(x)) \\ \rightarrow & [\neg \mathit{penguin}(x) \mid \mathit{bird}(x)]_x > 0.5 \end{array}$$

Definition 18 *The* language *generated by the logic is called the language of statistical probability,* $\mathcal{L}^{stat}$.

3.2.4 Combined Probability Logic

The combination of propositional and statistical probabilities is syntactically straightforward: we allow both probability operators in our language.

The formal semantics is more complicated due to existence of rigid and non-rigid terms. We will formulate a semantics for the combined language based on possible world semantics; please consult Appendix A. In short, rigid terms do not change between worlds; non-rigid may do so. For more details, consult Bacchus [Bac90].

Example 4 *For example, "The probability is 0.95 that more than 75% of all birds fly" would be*

$$\mathbf{prob}\left([\mathit{fly}(x) \mid \mathit{bird}(x)]_x > 0.75\right) = 0.95$$

Example 5 *Or, "John is more likely to succeed than the average student" is*

$$\mathbf{prob}\left(\mathit{succeed}(John)\right) > [\mathit{succeed}(x) \mid \mathit{student}(x)]_x$$

When discussing the direct inference principle (below) necessary for reasoning from statistical knowledge to individual knowledge, we need to take into consideration the fact that the combined logic contains non-rigid numeric terms; these denote different real numbers in every possible world.

We also have a probability distribution μ_S over the worlds; with $\mu_S(\mathcal{S}) = 1$, where $\mathcal{S}$ is the set of all possible worlds we attribute probabilities to.

Hence, it makes sense to talk about an expected, or expectation value of non-rigid numeric terms.

Definition 19 *The* expectation value *(or* expected value*) of non-rigid numeric terms is the weighted average of their values over the possible worlds, where* μ_S *determines the weight of influence of each individual world.*

Syntactically, the new expectation operator is denoted by E. Given a formula α, we have the expected value $E(\alpha)$. Finally, we define the language.

Definition 20 *The* language *generated by the combined statistical and propositional logic is called the language of combined probability,* $\mathcal{L}^{comb}$.

3.2.5 The Direct Inference Principle

The language needs one more auxiliary definition in order to be able to present one of the main definitions of this chapter. First we note that the conjunction of all external facts relating to the object application domain (i.e., all fully believed truths in the current world) [1] is denoted **KB**, the knowledge base. As Bacchus puts it [Bac90, p. 145], **KB** corresponds to the set of objective assertations the agent has accepted, the agent's belief base. As we point out in Appendix A, **KB** denotes the conjunction of a finite collection of objective formulas (page 150f); this includes assertions about particular individuals as well as general logical relationships between properties, and statistical information.

We start with the following auxiliary definition.

Definition 21 *Let α be a formula of $\mathcal{L}^{stat}$ and $\mathbf{KB}$ be a finite set of objective formulas from $\mathcal{L}^{stat}$ that are fully believed. If $\langle c_1, ...c_n\rangle$ are the n distinct object constants appearing in $\mathbf{KB}$ and $\langle v_1, ...v_n\rangle$ are n distinct object variables not appearing in $\mathbf{KB}$, then let $\mathbf{KB}^{\vec{v}}$ (and $\alpha^{\vec{v}}$) denote the new formula which results from textually substituting c_i by v_i in $\mathbf{KB}$ (and in α) for all i.*

We may then define the direct inference principle. Note that this is really an axiom (as shown and discussed in Appendix A); since Bacchus always refers to it as a principle, we defer to his usage.

Definition 22 *The* direct inference principle *is given by*

$$\mathbf{prob}(\alpha) = E\left(\left[\alpha^{\vec{v}} \mid \mathbf{KB}^{\vec{v}}\right]_{\vec{v}}\right) \tag{3.29}$$

This is then the agent's belief in α, given that $\mathbf{cert}\left(\left[\mathbf{KB}^{\vec{v}}\right]_{\vec{v}} > 0\right)$, and that the propositional and statistical probabilities are consistent. For more details regarding the direct inference principle, please see Appendix A.

Using the principle we may make inferences from knowledge bases. The following example, taken from Bacchus [Bac90, p. 157] illustrates this.

Example 6 *Suppose that*

$$\mathbf{KB} = bird(Tweety) \wedge [fly(x) \mid bird(x)]_x > c.$$

where c is a rigid (i.e. same in all possible worlds) numeric constant strictly greater than 0.5. In other words, we have accepted the assertion that Tweety is a bird and that the majority of birds fly. Using the direct inference principle we may then infer that

$$\mathbf{prob}\,(fly(Tweety)) > c.$$

[1] Full belief: following Bacchus we equate degrees of belief with probability; i.e. fully believed entails certainty, and vice versa.

We do this in the following way.

(1)	$\mathbf{prob}(fly(Tweety)) =$ $E([fly(v) \vert bird(v) \wedge [fly(x) \vert bird(x)]_x > c]_v)$	$(\mathbf{KB}, dir.)$
(2)	$\mathbf{cert}([fly(v) \vert bird(v) \wedge [fly(x) \vert bird(x)]_x > c]_v =$ $[fly(v) \vert bird(v)]_v)$	$(stat.)$
(3)	$E([fly(v) \vert bird(v) \wedge [fly(x) \vert bird(x)]_x > c]_v) =$ $E([fly(v) \vert bird(v)]_v)$	$(2, expt.)$
(4)	$\mathbf{prob}(fly(Tweety)) = E([fly(v) \vert bird(v)]_v)$	$(1, 3)$
(5)	$\mathbf{cert}([fly(v) \vert bird(v)]_v > c)$	$(\mathbf{KB}, thm.)$
(6)	$E([fly(v) \vert bird(v)]_v) > E(c)$	$(5, expt.)$
(7)	$E(c) = c$	$(rigid)$
(8)	$\mathbf{prob}(fly(Tweety)) > c$	$(4, 6, 7)$

where we have used the following abbreviations

dir. *direct inference principle, Axiom 7. See Appendix A, p. 151.*

stat. *Theorem 23. See Appendix A, p. 148.*

expt. *Expectation operator axiom, in Appendix A* **P16***, p. 146. Specifically, we use the fact that* $\mathbf{cert}(t = t') \rightarrow E(t) = E(t')$

thm. *Theorem 26. See Appendix A, p. 151.*

rigid *by definition, the constants are nonvarying from world to world*

We may extend the formalism to non-monotonic reasoning by noting that

$$\mathbf{cert}\left(\forall \vec{v}.KB^v \rightarrow \beta\right) \rightarrow E\left([\alpha \mid \mathbf{KB}^v]_{\vec{v}}\right) = E\left([\alpha \mid \beta]_{\vec{v}}\right) \tag{3.30}$$

In other words, we allow non-monotonic inheritance of statistical information from superclasses.

For more details, see Bacchus [Bac90].

3.3 TEMPORAL PROBABILISTIC LOGIC (TPL)

Temporal logic is used for reasoning about relationships and questions regarding time. Probabilistic logic, on the other hand, is used when knowledge is not exact; when information is missing or deductions may not be made with perfect knowledge, as informally shown in the previous chapter. This section presents an overview of temporal probabilistic logic (TPL) centring on giving the reader a sufficiently detailed picture of TPL in order to appreciate the present work and the application of TPL to the genetic algorithm that is in the following chapter. A more detailed description of TPL is contained in Appendix A, which formalises a combination of first order logic and temporal logic with the two probabilistic logics into a unified temporal probabilistic logic.

3.3.1 Syntax

The syntax of TPL closely follows that of its parent logics, first order logic and the probabilistic logics as well as temporal logic.

There are symbols from all logics; notably the temporal operators from temporal logic and the two probabilistic symbols **prob** and the '[' and ']' surrounding a term together with a subscripted set of placeholder variables denoting propositional and statistical probabilities, respectively.

Formulas follow standard practice and are very similar to those of standard first order logic with the addition of the operators and symbols from the other logics. The only essential difference is in that it unlike first order logic allows for numeric and temporal terms to be formulated from already existent formulas. For example, given α and β as formulas, we may have $\alpha \wedge \beta$ and other similar formulas as well as $\Box\alpha$ and $\Diamond\beta$ as well-formed temporal formulas. We also note that $\mathbf{prob}(\alpha)$ is valid, as is $[\alpha]_{\vec{x}}$ and $E(t)$, t being a statistical probability term. Numeric terms are also handled in the standard, intuitive way with standard infix symbols like '$+$' and '$-$' and so forth.

We also often use the aforementioned definition for certainty, $\mathbf{cert}(\alpha) =_{df} \mathbf{prob}(\alpha) = 1$.

We also want our language to include conditional probability terms, and formulate the following two axioms, which are duplicated here from the Appendix.

Definition 23 (Axiom of propositional conditional probabilities)

$$\begin{array}{ll} \mathbf{prob}(\beta) \neq 0 & \rightarrow \mathbf{prob}(\alpha|\beta) \times \mathbf{prob}(\beta) = \mathbf{prob}(\alpha \wedge \beta) \\ \mathbf{prob}(\beta) = 0 & \rightarrow \mathbf{prob}(\alpha|\beta) = 0 \end{array} \tag{3.31}$$

Definition 24 (Axiom of statistical conditional probabilities)

$$\begin{array}{ll} [\beta]_{\vec{x}} \neq 0 & \rightarrow [\alpha|\beta]_{\vec{x}} \times [\beta]_{\vec{x}} = [\alpha \wedge \beta]_{\vec{x}} \\ [\beta]_{\vec{x}} = 0 & \rightarrow [\alpha|\beta]_{\vec{x}} = 0 \end{array} \tag{3.32}$$

Note that this definition explicitly defines the case where the conditioning formula has probability zero, contrary to normal practice that leaves it undefined.

3.3.2 Semantics

In order to interpret the formulas in our language we need to define appropriate structures for them, first for temporal and then for probabilistic formulas. We then turn our attention to the interpretation of the formulas.

The concept of time is informally defined using interval structures, and intervals with two basic relations $\sqsubseteq$ 'inclusion' and $<$ 'precedence'. The view of time we use is one modelled on the integers, $\mathbb{Z}$. In order to simplify axioms we introduce the overlap relation O. We also use the notation $x^{\bigcirc}$ for the next base interval relative to x and $x^{\overline{\bigcirc}}$ for the previous; we note that these are used heavily when discussing the genetic algorithm. Also note that as shown earlier we have a smallest interval, the base interval.

Interpretation of the formulas is fairly standard, with some exceptions. Inductively, we assign a truth value **true** to α if our semantic structure or model satisfies $M \models \alpha$. The exceptions centre around the introduction of probabilistic and temporal formulas, which have to be carefully handled in order to obtain sensible results. We have taken the step of as far as possible isolating the logics (and their semantic and syntactical characteristics) from each other. Thus, for instance the probability operator **prob** 'gives' a numeric term, which cannot be handled by any other but probabilistic operators, and so forth. For the details, see Appendix A.

3.3.3 Proof theory

In order to examine proof-theoretic properties of TPL we must present an axiom system for all formulas. We do this separately for the different components of the logic.

First-order logic axioms use an axiomatisation of first-order logic with equality as a basis [Bac90] and is standard. The only thing to look out for is to use compatible terms as specified in the theory.

Regarding numeric axioms, we following Bacchus [Bac90] do not attempt to capture the full behaviour of these real-valued terms; instead we use the axioms of a totally ordered field to capture a large part of their behaviour. For details see appendix A.

The probability axioms from [Bac90] come in three groups: propositional, statistical, and combined probability axioms. Appendix A contains full details of the axioms, together with descriptions of their applicability.

Although there are many temporal axioms (see table A.7 in the Appendix), they do not yield a recursive first-order axiomatisation for the theory of $INT(\mathbb{Z})$. This is no surprise, given the uncertainty regarding the axiomatisability of **FOUND**'s first-order consequences [Ben83, p. 64 and 73].

In order to be able to reason about the future (and the past) we need the frequently used next and future accessibility modalities, $\Box$ and $\Diamond$. Following Masini [Mas93] we have the following intuitive meanings.

$\Box A$ means A is true in every accessible future.

$\Diamond A$ means A is true in some possible accessible future.

We note that $\Box$ may be seen as a 'henceforth' operator, and $\Diamond$ as an 'eventually' operator [Ben83, p. 156] [MP91] [MP93]. We incorporate these into our logic using the axioms in the fourth section in table A.7 [Mas93, p. 17f].

We also need to show explicitly which rules of inference we support in our logic. For completeness' sake they are summarised below as well as in the appendix A [Bac90] [Mas93].

MP. From α and $\alpha \rightarrow \beta$ infer β (Modus ponens)

UG. From α infer $\forall x.\alpha$ (Universal generalisation)

PE. From $\alpha \equiv \beta$ infer $\mathbf{prob}(\alpha) = \mathbf{prob}(\beta)$ (Probability of equivalents)

3.3.4 The direct inference principle revisited

As pointed out by Bacchus [Bac90, p. 139], what we now have is a logic capable of dealing with propositional and statistical probabilities, as well as temporal propositions. However, in this combined logic there is no intrinsic relationship between the two kinds of probabilities. They simply coexist without any necessary interaction.

Bacchus notes that, although the types of probabilities are distinct, there is clearly a connection between them, which is most clearly apparent in actuarial situations, which often equate the two. A good example is an insurance company quoting a rate based on statistical information for an actual person.

This leads us to a very important principle: that of direct inference, from statistical probability to propositional. This we have already treated above.

3.3.5 Soundness, completeness, and consistency

Is this combined logic *sound*, i.e., are all formulas deducible from it valid, and *complete*, i.e., are all valid formulas deducible from it? Furthermore, is it *consistent*, i.e., given a formula α, its negation $\neg\alpha$ must not be deducible.

Briefly, following Bacchus [Bac90, p. 132] we can show that the combined logic of first order, propositional and statistical probability logic is sound. Van Benthem shows that interval-based temporal logic is sound as well; in the appendix detailing our formalism we show that the combination of all of these is both sound and consistent as well.

However, as Bacchus notes [Bac90, p. 62], Abadi and Halpern [Aba88] have shown that propositional probability logic is not complete, which is clear because the set of deducible formulas do not form a recursively enumerable set. As van Benthem also points out, this is true for temporal logic as well. The same is then immediately true for our combined logic.

4 ASPECTS OF TIME IN GENETIC ALGORITHMS

This chapter lays the foundation for the genetic algorithm by first defining the structures to be used and then formalising the algorithm using them. We emphasise the temporal aspects by defining the structures in temporal terms, in a setting based on sets.

Having laid the foundation, we then discuss the basic problems associated with genetic algorithms in the light of the new formalism: convergence, selection, and the existence of optima. Parts of this chapter are based on material first presented in [San95, San96a, San96b, San96c] .

The basic idea is to think of the genetic algorithm being handled by a *logic*, with population individuals being seen as propositions. In other words, an individual's genetic information (encoded in the genes forming the chromosome) is taken as a proposition (in a certain problem-specific language not further detailed) having a defined truth value. Thus a population is formed of individuals, where the fact that they are alive indicate the propositions being true. Furthermore, this logic is also a temporal one, defined over intervals of time: the intervals will separate generations of individuals (and possibly populations). Since individuals are defined over temporal intervals we have an interval-dependent population, one in which propositions can be said to be existing. We may then reason within the logic, trying to solve problems and answer interesting questions regarding the behaviour of the genetic algorithm over time. For instance, we will endeavour to give conditions for the algorithm finding optimal (in some pre-defined sense) solutions, i.e. populations. What really is characteristic of an optimum? Can we tell when the algorithm is 'near' the optimum?

4.1 GENETIC STRUCTURES

We will now turn our attention to defining structures relating more to genetic algorithms.

By necessity we will be using some terms central to the genetic algorithm before a formal definition of them is given; e.g. population, individual, and generation.

Definition 25 (Gene) *A gene γ is a structure embodying a specific item of data encoded in some suitable form.*

A gene forms the inherited information from generation to generation. Note that the gene is defined as the smallest unit in the genetic algorithm, i.e. as far as the further operation of the genetic algorithm is concerned they are atomic. In practical genetic algorithms genes code for problem-specific parameters. Note that natural chromosomes also contain *non-coding* genes, i.e. genes that do not apparently code for any protein. Some studies show that such non-coding genes stabilise the genetic algorithm as well, and allow it to search faster for its objectives [WL95].

Then, seeing the gene structures as bit strings, we denote the set of all possible genes $\mathcal{S}_\gamma = \cup_{i=1}^{k} \mathcal{S}_i$, where $\mathcal{S}_i$ is the set formed from all bit strings of length i (i.e. of size 2^i) and k is any arbitrary length of the bit strings we allow; cf. p. 12.

For completeness' sake we also show the definition of an allele.

Definition 26 (Allele) *An allele is an allowed value of a gene, i.e. the alleles form the range of the genes.*

An allele is thus a specific value of a specific gene within a specific time interval. An important question is that of the allowed range of the values of an allele. Theoretically, the values can be anything – a real number, say – however, they have to be encoded in the gene, as per definition 25. This means that there have to be a finite, or at least denumerable number of possible values. This fits in very well with the characteristics of a string [1]. Since strings are used to encode genes in the genetic algorithm, gene values (i.e. alleles) form $\mathcal{S}_\gamma$, which thus is at least denumerable (and in practice always finite).

Following Bäck and Schwefel [BS93] we define a chromosome as follows.

Definition 27 (Chromosome) *A chromosome ξ is a vector of genes. Formally*

$$\xi = \vec{\gamma} = \langle \gamma_1, \gamma_2 \dots \gamma_n \rangle \tag{4.1}$$

where n is the length (i.e. number of genes) in the chromosome. Note that the genes may be duplicated in the chromosome.

Although genes may be duplicated in chromosomes we may often think of the chromosome as a set of genes instead of a vector. However, note that since genes are positional we, wherever the distinction is important must use the vector representation; such a position in a chromosome is called a *locus* in genetics. This means that, although two genes may look the same, they 'code' for different information if they are at different loci. Every locus (i.e. coded parameter) has its own alleles.

Chromosomes, in contrast to genes, are thus not atomic; they may be split and recombined at gene boundaries.

We denote the set of all possible chromosomes $\mathcal{S}_\xi$. This set has the form

$$\mathcal{S}_\xi = \bigcup_{n \geq 1} \mathcal{L}_1 \times \cdots \times \mathcal{L}_n \tag{4.2}$$

where $\mathcal{L}_i$ is the i:th locus' gene space and n is the number of loci, i.e. the length of the chromosome in number of genes. In the common case, in terms of the genetic algorithm, of a fixed number of genes per chromosome and all genes sharing the same possible values, this degenerates into

$$\mathcal{S}_\xi = \mathcal{S}_\gamma^n. \tag{4.3}$$

[1]According to the IEEE 1484 Learning Technology Standards Committee a *string* is a data type consisting of symbols from standard character sets; properties: unordered, exact, non-numeric, and denumerable.

since then each $\mathcal{L}_i = \mathcal{S}_\gamma$.

Often the concept of a gene is omitted in the literature on genetic algorithms, and chromosomes are used exclusively. In this case the chromosome is often thought of as a string of bits encoding the problem at hand; strictly speaking the bits would then form the genes.

Furthermore, in nature the chromosome is always based on a physical structure. It is important to realise that in nature species may have identical or virtually identical sets of genes, and still differ significantly (see e.g. [HP95]). The reason is that in nature it is strictly speaking not the chromosome that is the most important bearer of information: it is the *genome*, a vector of genes that may or may not encompass a single chromosome. In other words, both the contents and order of the genes within the genome matters. Interestingly, this imposes a second-order structure on the inheritance of information which to our knowledge has not been exploited in genetic algorithms.

Definition 28 (Agent) *An agent α (also called an* individual*) is a tuple*

$$\alpha = \langle \xi, I \rangle \tag{4.4}$$

where ξ is a chromosome and $I \in \mathcal{I}$ is a time interval.

Note that we use the terms agents and individuals interchangeably throughout this work. Sometimes we may also talk about members of a population, especially when we want to emphasise the membership.

Let $\mathcal{S}_\alpha$ be the set of possible agents. This set of all possible agents may be infinite because there is no *a priori* restriction on range of intervals. Because intervals are defined over $\mathrm{INT}(\mathbb{Z})$ the set is countable. We note that the set of all possible agents *within* an interval I (whether base or composite) is

$$\mathcal{S}_\alpha^I = \{\alpha \mid I \sqsubseteq I(\alpha)\} \tag{4.5}$$

where $I(\alpha)$ denotes the interval within which α is defined ('alive').

Example 7 *In order to see what a possible population looks like in this formalism, we can imagine a set of agents $\alpha_i, i = 1 \ldots n$ with the following general form*

$$\alpha_i = \langle \vec{\gamma_i}, I_i \rangle$$

where $\vec{\gamma_i}$ is a vector of genes, say $< 0, 1, 0, 1 >$ making up a chromosome, and $I_i = [m_i, n_i]$ according to the relevant definitions. Then, in particular, the following explicit agents are allowed

$$\begin{aligned}
\alpha_1 &= \langle < 0, 1, 0, 1 >, [0, 1] \rangle \\
\alpha_2 &= \langle < 0, 1, 0, 1 >, [0, 2] \rangle \\
\alpha_3 &= \langle < 0, 1, 0, 1 >, [0, 3] \rangle \\
\alpha_4 &= \langle < 0, 1, 0, 1 >, [1, 2] \rangle \\
&\ldots
\end{aligned}$$

showing that agents may have the same genes (chromosome pattern, as it were), as long as their intervals differ, i.e. they are not alive at the same times.

Central to the genetic algorithm is the concept of the fitness of an agent. Note that since individuals in genetic algorithms only have one chromosome each (in contrast to real life) we could talk about the fitness of the chromosome instead; however, it is customary to use individual (or agent) – assuming, of course, that the fitness is time-independent, and not depending on the interval.

Definition 29 (Probabilistic fitness) *An individual's (i.e., agent's) probabilistic fitness is its probability of occurrence in a population within any given interval:*

$$\phi(\xi, I) = p(\alpha_I) \tag{4.6}$$

where α_I denotes an agent with chromosome ξ within the interval I.

Sometimes we abbreviate $\phi(\xi, I)$ to ϕ_ξ especially in cases where the time interval need not be emphasised. When we want to emphasise the fitness of a certain individual α with chromosome ξ we use the expression $\phi_\xi(\alpha)$. In terms of our formalism the fitness is a number, which may be manipulated using standard numerical operators. This means that we have a two-sorted logic, in line with that of Bacchus' [Bac90] probabilistic logic with its probability operator.

Since ϕ is a probability, this also means that the following hold.

$$\phi : \mathcal{S}_\alpha \rightarrow [0, 1]. \tag{4.7}$$

In other words, ϕ is a function giving a number from 0 to 1, inclusive. Furthermore, assuming I is a base interval,

$$\forall I \in \mathcal{I} : \sum_{\xi \in \mathcal{S}_\xi} \phi(\xi, I) = 1 \tag{4.8}$$

capturing the essential fact that, looking at any single base interval the sum of the probabilities of the possible chromosomes (i.e. the occurring individuals in the population at that interval in time) is one.

Summarising, in this way we have equated the *phenotypic fitness* (the relative ability of an organism to survive in its current environment) with the probability of occurrence (probabilistic fitness); the *genetic fitness* (the relative ability of an organism to propagate its genotype) is used when determining which agents may reproduce [Cas89]. The genetic fitness is what connects the agent to the problem at hand, i.e. the problem-specific connection to the chromosome and its genes with their encoded information.

Strictly speaking, there need be no one-to-one correspondence between the two types of fitness. For example, consider *pleiotropy*, the effect that a single gene may simultaneously affect several phenotypic traits, and *polygeny*, the effect that a single phenotypic trait may be determined by the simultaneous interaction of many genes [Fog94b].

Indeed, as has been pointed out by Nettleton [Net94], citing Lewontin [Lew74], the interaction between genotypic space G and phenotypic space

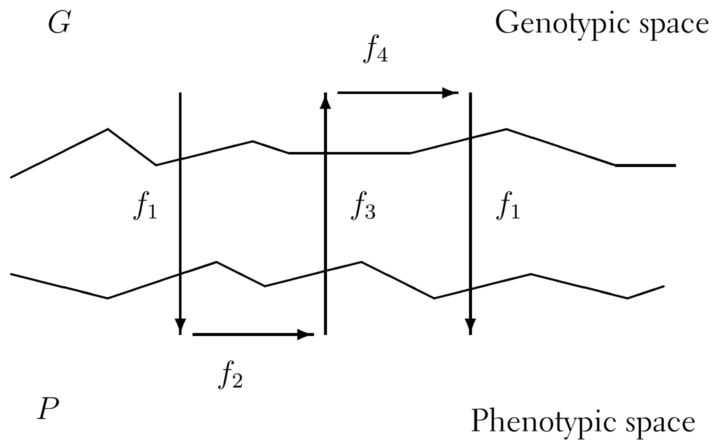

Figure 4.1: Phenotypic and genotypic space mappings suggested by Nettleton and Lewontin. The four minimal mappings to fully describe the relationship between phenotypic and genotypic space are schematically shown.

P is complex, and may be characterised by four functions:

$$f_1 : E \times G \to P \tag{4.9}$$

$$f_2 : P \to P \tag{4.10}$$

$$f_3 : P \to G \tag{4.11}$$

$$f_4 : G \to G \tag{4.12}$$

where E models the addition of environmental symbols. The interaction is shown in figure 4.1.

The mapping f_1 models the development of the phenotype in terms of its genotype and other environmental factors, such as epigenesis (the sensitivity of development to local conditions). For example, some fish change gender depending on the temperature of the sea. f_2 controls the selection, immigration and emigration of individuals within the population; this does not impact on the individual's encoding, but on its behaviour. The phenotypic space contains *traits*, or characteristics of the individual (e.g. speed, dexterity, etc.) as elements, i.e. range for f_2 and f_3 and domain of f_1 and f_2. Then, f_3 maps the effects of f_2 back to the genotype. The genotypic space contains chromosomes (in the sense of the genetic algorithm described above) encoding in some way the aforementioned traits, as range for f_1 and f_4 and domain of f_3 and f_4. Finally, f_4 controls the manipulation of the coding using genetic operators, such as mutation and recombination.

In this model, however, we assume that the genotype and phenotype simply correspond to each other. This leads us to formulate the following assumption.

Axiom 1 (The Central Assumption [CA] of the GA Model.) *There is a one-to-one correspondence between the genetic and phenotypic fitness of an agent.*

To see that it is necessary to formulate this assumption, one need only consider a lethal mutation (on the genotypic level) that makes the individual extremely fecund (on the phenotypic level). Then the mutation would quickly spread, but would make the population very 'unfit', in terms of the genotypic fitness and at the same time very 'fit' in terms of the phenotypic fitness since it would be exhibited by a large proportion of the population. In other words, there is no necessary correlation, either positive or negative, between the phenotypic and genotypic fitness. Of course, this is only true on the individual level; on the population level there must be a positive correlation, otherwise the population would die out. In other words, unless we have this assumption we cannot derive any genotype fitness information from the phenotype.

Using the CA we may omit the qualifier genetic or phenotypic, except where we want to explicitly emphasise which kind of fitness we are referring to.

For an interesting discussion on genotypes vs. phenotypes in a genetic algorithm setting, see Hart, Kammeyer, and Belew [HKB94].

In practice, we use a fitness function (see below) that produces a value that is used to grade individuals. This value is often called fitness as well; strictly speaking, it is an *external* fitness, one that is used by the selection function in determining which individuals shall survive to the next generation. In other words, the probabilistic fitness may be seen as the external fitness normalised to the open interval $(0, 1]$ *after* selection has taken place. The reason for an open lower boundary in the interval is clear from the definition of a population below. This means that the probabilistic fitness cannot be used for selection, as we shall see.

Definition 30 (Population) *The population* Π *of a genetic algorithm is the set of all agents having a non-zero probabilistic fitness within a time interval. Formally*

$$\Pi = \{\alpha \mid \phi(\alpha) > 0\} \subseteq \mathcal{S}_\alpha. \tag{4.13}$$

We also call this the *real* population for reasons that become clear below (theorem 2).

Π is normally very small compared with $\mathcal{S}_\alpha$, i.e. $|\Pi| \ll |\mathcal{S}_\alpha|$. For completeness' sake we denote the set of all possible populations $\mathcal{S}_\Pi$.

Before going on to formally define the genetic algorithm, we assume that the population Π is finite. It is clear that if the population is finite initially, then it will stay finite forever (in practice due to physical limitations). The reason is that all new offspring are generated from a population (either initial or subsequent), and any operation that generates offspring will always generate a finite number of them (see (4.50) and (4.51) below). Of course, we could assume an operator that generates an infinite number of offspring; however, such an operator would be impossible in practice even if conceivable in theory. We need to assume that the *initial* population is finite; from a practical point of view this is obvious (how could we do any calculations in practice on an infinite population?), but from a theoretical point of view the size of the population need not be constrained. Indeed, several researchers have assumed infinite populations in their work (e.g. Goertzel [Goe95] and

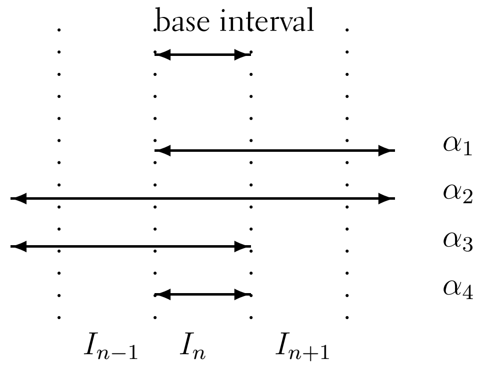

Figure 4.2: Population for an interval

Worden [Wor95]) but none has (as far as the author knows) assumed an operator that from a finite number of parents generates an infinite number of offspring.

The probabilistic fitnesses of the members of the population form a *probability distribution* over $\mathcal{S}_\alpha$. As the population Π is a set of individuals (see Definition 30 above), the probability distribution f on $\mathcal{S}_\alpha$ is a function $f : \mathcal{S}_\alpha \rightarrow [0,1]$ such that $\sum_{m \in \mathcal{S}_\alpha} f(m) = 1$. The *support* of a probability distribution f on $\mathcal{S}_\alpha$ is the set of $m \in \mathcal{S}_\alpha$ such that $f(m) > 0$. For a set of individuals $L \subseteq \mathcal{S}_\alpha$ and a probability distribution f on $\mathcal{S}_\alpha$ we define $f(L) = \sum_{m \in L} f(m)$. For more details see e.g. [SB95].

From the foregoing discussion it is also clear that the population is the support of the probability distribution for the probabilistic fitness.

The population Π_I within a base interval I is given by

$$\Pi_I \;=\; \{\alpha \mid I \sqsubseteq I(\alpha) \wedge \phi(\alpha) > 0\} \tag{4.14}$$

Note that, using the preceding interval relations equation 4.14 may be written as

$$\Pi_I = \{\alpha \mid I \; O \; I(\alpha) \wedge \phi(\alpha) > 0\}, \tag{4.15}$$

using the overlap relation introduced in definition 7, as can readily be seen.

More concisely, this may also be written as

$$\Pi_I \;=\; \{\alpha \in \mathcal{S}_\alpha^I \mid \phi(\alpha) > 0\} \tag{4.16}$$

using the previous definition of the possible set of agents within an interval (equation 4.5).

This is shown in figure 4.2. In the figure we have a base interval I_n with its neighbouring intervals, I_{n-1} and I_{n+1}, and four population members α_1, α_2, α_3, and α_4 depicting the four possibilities:

α_1– extends from the beginning of the interval beyond the end of the interval

α_2– extends from beyond the beginning and beyond the end of the interval

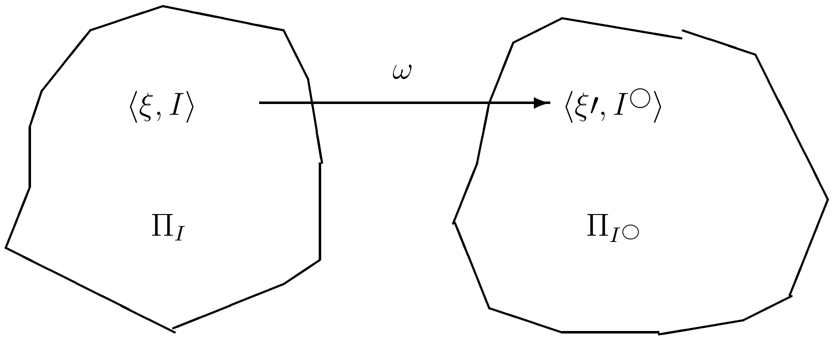

Figure 4.3: Traditional genetic operator

α_3– extends from beyond the beginning of the interval to the end of the interval

α_4– extends exactly over the interval (from the beginning to the end)

With the addition of the case of two totally disjoint intervals, these are the 'interesting' period relations as defined by van Benthem [Ben83, p. 9].

Putting it another way, we may, following Shoham [Sho88, p. 47ff] consider the chromosome as a logical proposition (in the sense of [San94b]), and in his terms say that we are using *liquid* temporal propositions [Sho88, Shoham definition 2.7, p. 49]. Below we will not be using Shoham's formalism since it largely follows van Benthem's.

Traditionally, the genetic algorithm is seen as modifying the genes (e.g. for mutation) or chromosomes (e.g. for crossover and recombination) in a population. This is depicted in figure 4.3 where we use the generic mapping ω denoting the relation between, on the one hand, parent and offspring individuals (strictly speaking chromosomes), and on the other hand subsequent populations (generations).

The problem is that in our setting we do *not* want to be seen modifying the chromosome as it denotes a proposition in our underlying logic; instead we describe the new population in terms of the propositions' fitnesses. This necessitates a change of perspective in such a way that the population so to speak both before *and* after application of the genetic operators consists of the *same* members. This means that a genetic operator that in the traditional manner generates a new individual (the chromosome of which might not have existed in the previous generation) instead modifies the fitness of an (possibly 'virtually') existing one from zero to a finite value. In other words, a new individual gets 'born'. If the individual exists, the operator modifies the fitness according to the composition of the new population. An individual may also 'die', in which case the fitness, i.e. the probability of occurrence after being selected against, goes from a non-zero to zero value.

We may thus think of the population as consisting of two kinds of members: the *real* and the *virtual* members, where the real members have a non-zero probability of occurrence, i.e. fitness, and the virtual ones have zero. This is not the only criterion for a virtual member, however. It is important to realise that the virtual members are not random: they are exactly those that at *some time in the future will exist.* Reversing the picture, any real member has either always been real from the initial moment, or been virtual up to a 'time of birth'. Likewise, an individual may 'die' and become virtual; of course, it may be 'resurrected' at a later time.

Below we define and characterise the concepts; for a real member of a population we have the following definition.

Definition 31 (Real member) *A* real *member α_I of a population Π_I within an interval I is an agent (individual) that exists.*

The corresponding definition of a virtual member is as follows.

Definition 32 (Virtual member) *A* virtual *member $\alpha_{I,v}$ of a population Π_I within an interval I is an agent (individual) that at some interval during the run of the genetic algorithm has existed, or will exist, but that during I does not do so.*

In order to characterise these formally, we define the predicate $alive$.

Definition 33 ($Alive$ predicate) *The predicate $alive$ is defined as*

$$alive(\alpha) : \phi(\alpha) > 0 \tag{4.17}$$

and essentially is **true** *when the fitness of an individual (the argument α) is greater than 0, and* **false** *otherwise.*

This definition is in line with Definition 30 for a population. We may now present the following theorems using the predicate.

Theorem 1 *For a general virtual member $\alpha_{I,v}$ of a population Π_I the following holds*

$$\bigcirc\Diamond alive \;\vee\; \overline{\bigcirc}\overline{\Diamond} alive \tag{4.18}$$

where alive *is the predicate in Definition 33.*

Proof. Separating the two halves dealing with future time ($\bigcirc\Diamond$) and past time ($\overline{\bigcirc}\overline{\Diamond}$), and taking the future case first, we notice that the next operator $\bigcirc$ ensures that we do not deal with the current interval I. Then, since the eventually operator $\Diamond$ ensures that the expression becomes **true** at some, unspecified, time in the future, we have fulfilled the requirement of Definition 32 for future times. It is easy to see that past time is handled similarly, and that our expression embodies the definition of a virtual member. □

What this means is that from the next generation of the individual we are certain ('eventually') that its fitness will some time in the future become non-zero, i.e. it will exist; in other words, it will become a real member of some future generation. Likewise, for the general case, for past times. We will find little use for general virtual populations or their members. We will use the term '1-generation' virtual member for one that is 'alive' in the next generation, and n-generation for one that is in the n:th generation ahead.

For this last case of n generations ahead, we prove the following theorem.

Theorem 2 *For a n-generation virtual member $\alpha_{I,v}$ of a population Π_I the following holds*

$$\bigcirc^n alive \tag{4.19}$$

where alive *is the predicate in Definition 33.*

Proof. n-generation means that the past is no longer taken into account, so following definition 32 and theorem 1 the term with the barred operators is omitted. Furthermore, recalling that $\bigcirc^n$ denotes the abbreviation of n applications of $\bigcirc$ the eventually operator $\diamond$ may be dropped since, temporally speaking, $\diamond$ is equivalent to $\bigcirc$ applied enough times (i.e., $\exists n$ such that $\bigcirc^n\alpha = \diamond\alpha$.) The correctness of the theorem immediately follows. □

Specifically, for a 1-generation virtual member we have the following.

Corollary 1 *For a 1-generation virtual member $\alpha_{I,v}$ of a population Π_I the following holds*

$$\bigcirc alive \tag{4.20}$$

where alive *is the predicate in Definition 33.*

Proof. Directly from Theorem 2 setting $n = 1$. □

In other words, in the next generation the individual will be real, i.e. have a non-zero fitness.

The set of virtual members will be called the *virtual population* $\Pi_{I,v}$. For specific generations we may use superscripts, $\Pi^1_{I,v}, \Pi^2_{I,v} \ldots \Pi^n_{I,v}$. Also note that we mostly omit the superscript if it is 1; and that we frequently use the shorthand $I^{\bigcirc}$ for the next generation's, and $I^{\overline{\bigcirc}}$ for the previous generation's interval. This also means that we use e.g. $\Pi_{I^{\bigcirc},v}$ for the virtual population of the next interval. Furthermore, unless otherwise noted we will always designate the 1-virtual member as a (plain) virtual member, and explicitly use the prefix (e.g n-generation) for others; likewise for virtual populations.

The *total* population $\Pi_{I,t}$ for an interval I is simply the union of the real and virtual members within the interval. Thus we always have that $\Pi_I = \Pi_{I,t} - \Pi_{I,v}$, omitting the superscripts to designate the interval (in the future). The virtual population may also be designated as 1-generation, 2-generation, $\ldots n$-generation, as appropriate. For $n > 1$ we may also have *cumulative* or *direct* virtual populations: as implied by the designations an n-cumulative population is formed by first adding the virtual members for 1-, then 2-, $\ldots n$-generation, whereas an n-direct just adds the members from n-generation to the population of interest.

This leads to the following interesting observation. Note that we use the initial population Π_0 from Definition 40 in advance of its introduction; we trust this will not cause difficulties.

Theorem 3 *The subset of $\mathcal{S}_\xi$ that the genetic algorithm has examined (or checked) up until any interval I is the n-generation cumulative virtual population, n being the ordinal number of the interval in question, in addition to* *the initial population. Formally*

$$\Pi_{I,checked} = \Pi_0 \cup \bigcup_{i=1}^{n} \Pi_{I,v}^{i} \tag{4.21}$$

where Π_0 *is the initial (real) population. This may also be shown as*

$$\Pi_{I,checked} = \{\alpha \mid \overline{\diamond}\, alive(\alpha) = true\} \tag{4.22}$$

where alive *is the predicate in Definition 33.*

Proof. The former equation follows directly from the foregoing discussion and definitions 30, 32, and theorem 2. The latter equation follows from the definition of the temporal previous $\overline{\diamond}$ operator and the definition of a population as having individuals with fitnesses greater than zero (others are never examined); so any that has been checked must have had fitness greater than zero. □

Note that we are *not* implying that all individuals with non-zero fitness have been examined.

We are now ready to discuss how new members of the population are generated.

4.2 GENETIC OPERATORS

We also need to know by what mechanisms an individual changes, i.e. evolves. We call these mechanisms *genetic operators*; strictly speaking there are three different kinds corresponding to operations on the three structures in the genetic algorithm:

- operators affecting genes: $\omega_\gamma : \mathcal{S}_\gamma \rightarrow \mathcal{S}_\gamma$
- operators affecting chromosomes: $\omega_\xi : \mathcal{S}_\xi \times \mathcal{S}_\xi \times \ldots \times \mathcal{S}_\xi \rightarrow \mathcal{S}_\xi$
- operators affecting populations: $\omega_\Pi : \mathcal{S}_\Pi \rightarrow \mathcal{S}_\Pi$

Traditionally, genetic operators are seen as modifying the genes (e.g. for mutation) or chromosomes (e.g. for crossing and recombination) in a population. This is depicted in figure 4.3.

We define three kinds of genetic operators as sketched above. The first one transforms genes to genes. A gene-specific operator ω_γ is a mapping transforming a gene γ_D to another gene γ_R. Mutation serves as a good example: there a gene is transformed by, e.g. randomly flipping a bit (seeing the gene as a bit string along the lines indicated on p. 31) in γ_D altering it to γ_R. Note that in general application of gene-specific operators is optional, i.e. they are applied with a typically very low probability. We can depict the situation as in figure 4.4.

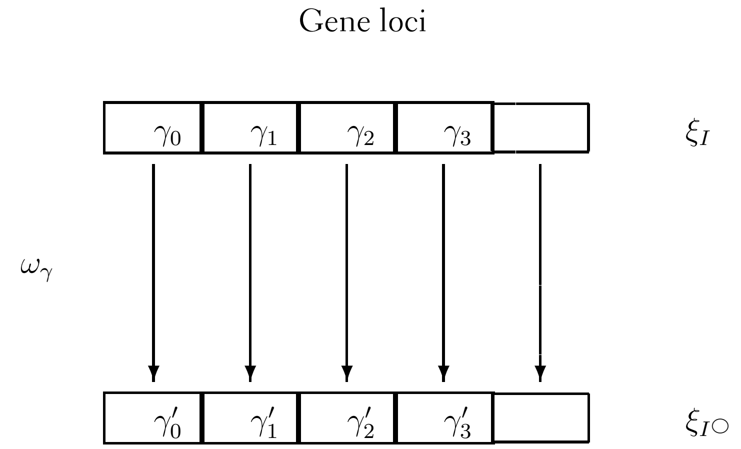

Figure 4.4: Genetic operator

As the figure clearly shows each gene, at its own locus, may be transformed to another *of its own* alleles; in other words, ω_γ is a function.

Thus we may define ω_γ as follows.

Definition 34 (Genetic operator: gene) *A gene-specific genetic operator ω_γ is a deterministic function transforming an allele of a gene γ_D to another γ_R.*

$$\omega_\gamma : \gamma_D \mapsto \gamma_R \tag{4.23}$$

where the subscripts D and R represent domain and range, respectively.

In general, a genetic operator ω_γ is applied with a certain probability δ. Since δ more often than not is comparatively small most genes are not modified during the run of the genetic algorithm. As an example, already mentioned above, consider mutation: a gene may either mutate to another allele, or not at all. In any case one gene may never change into another gene; cf. the discussion on loci above. The end result may from a chromosome point of view be that two chromosomes and hence the individuals contain exactly the same structure. However, these are considered different as the chromosomes (containing the genes) in the population strictly speaking form a multiset, as pointed out above. In other words, the mapping of gene loci from generation to generation is bijective and the operator is a function.

The second kind of genetic operator operates on chromosomes. There we have two or more chromosomes being transformed into one, two or more, probably different chromosomes. The almost universally used crossover (or recombination) operator is a good example of this. There are many ways how crossover may be implemented: one-point, two-point, uniform, and so on [Gol89, Whi94]. In these one new chromosome (individual) is generated from two 'parents' in various different mixing schemes. In the evolutionary strategies variety of evolutionary algorithms we frequently have a number of recombining chromosomes not equal to two [BS95a, SB95].

Before showing the definition, we need to be able to obtain the chromosomes for a population Π at a specific interval I. Recalling Definition 28 for an agent belonging to a population (Definition 30) we see that we need to project out the chromosomes from the population, discarding the interval information in the process. We define a projection operator $\backslash$ as follows.

Definition 35 (Projection) *Given a set $\mathcal{A} = \{A_1, A_2, \ldots A_m\}$ of tuples $A_i = \langle a_{i,1}, a_{i,2}, \ldots a_{i,n}\rangle$ the projection operator $\backslash_{a_j}$ extracts a set B of an indicated element a_j from each of the tuples A_i in $\mathcal{A}$. Formally we have*

$$B = \mathcal{A}\backslash_{a_j} = \{a_{1,j}, a_{2,j}, \ldots a_{m,j}\} \text{ where } 1 \leq j \leq n \tag{4.24}$$

As a straightforward extension to the projection operator $\backslash$ we allow it to project out single elements from tuples; e.g given an agent $\alpha = \langle \xi, I\rangle$ the operation $\alpha\backslash_\xi$ gives just ξ.

Operators for chromosomes may then be characterised by the following general definition.

Definition 36 (Genetic operator: chromosome) *A chromosome-specific genetic operator is a function transforming a set of chromosomes to a chromosome.*

$$\omega_\xi : \{\xi_D \mid \xi_D \in \Pi_I\backslash_\xi\} \mapsto \xi_R \in \Pi_{I^\bigcirc}\backslash_\xi \tag{4.25}$$

where $I^\bigcirc$ refers to the immediately succeeding interval following I.

Strictly speaking, since we often equate chromosomes with individuals we should use the adjective individual-specific, not chromosome specific, or rather specific to a set of individuals; however, it is customary to use the plain chromosome specifier.

The designation function above is warranted because of what we could call the multiset property of the population; even if two individuals share the same genetic information (the chromosomes being equal, and thus the projection $\Pi\backslash_\xi$ strictly speaking forms a multiset, not a set) we consider them distinct because the agent according to Definition 28 includes the interval as well.

Finally, the third type of genetic operator operates on whole populations. There is only one in general use: the *selection* operator determining which individuals will survive to the next generation and which will not. The selection operator uses a special function to grade individuals; this function is called the *fitness* function, as described above. We could easily imagine other population functions as well: in an implementation of a distributed genetic algorithm the actual distribution of the population could be treated as such an operator, to mention one example. The formal definition of the third kind of genetic operator is as follows.

Definition 37 (Genetic operator: population) *A population-specific genetic operator ω_Π is a mapping transforming a population Π_D of individuals into another population Π_R of individuals.*

$$\omega_\Pi : \Pi_D \mapsto \Pi_R \tag{4.26}$$

Of course, this definition may be seen as subsuming all the different cases described above.

Note that due to the mapping $\Pi_D \neq \emptyset$ whereas Π_R may be $\emptyset$ and that $|\Pi_D|$ is not necessarily equal to $|\Pi_R|$. We will treat the population selection and fitness function and their characteristics separately below (section 4.4).

We denote the set of all genetic operators Ω. First setting

$$\begin{aligned}\Omega_\gamma &= \{\omega_{\gamma_1}, \omega_{\gamma_2}, \ldots\} &(4.27)\\ \Omega_\xi &= \{\omega_{\xi_1}, \omega_{\xi_2}, \ldots\} &(4.28)\\ \Omega_\Pi &= \{\omega_{\Pi_1}, \omega_{\Pi_2}, \ldots\} &(4.29)\end{aligned}$$

we have that

$$\Omega = \Omega_\gamma \cup \Omega_\xi \cup \Omega_\Pi. \tag{4.30}$$

It is also clear that the genetic operators described above are composable; recall that a sufficient condition for composability is that they are (at least partial) mappings and that they are compatible. Compatibility in this context can be realised easiest by, so to speak, elevating all operators to operating on populations; in other words, interpreting them all as ω_Π's. Seen from this perspective, ω_ξ will become an ω_Π, namely one where one individual's chromosome has been altered. Clearly, this individual is still an individual of a population, albeit a different one; and we can say that the difference has been caused by the application of an operator ω_Π having exactly this localised effect. In the same way we can argue that ω_γ's can be seen as ω_ξ's and hence as ω_Π's. Summarising, all operators $\omega \in \Omega$ can be said to map from a population to a population (i.e. $\Pi_D \mapsto \Pi_R$ as in Definition 37.) We can formalise this by using a variant of the projection operator $\setminus$ (see definition 35) for operators.

The important thing is to realise that we are dealing with structures embedded in each other in a hierarchical fashion. Recall that

$$\begin{aligned}\Pi_I = \{\langle \xi_1, I\rangle, \ldots \langle \xi_n, I\rangle\} &\quad(4.31)\\ \xi = \langle \gamma_1, \ldots \gamma_m\rangle. &\quad(4.32)\end{aligned}$$

In other words, we have three 'levels': the population level Π, the chromosome level ξ, and the gene level γ, each of which (except for the atomic lowest level γ) contains elements made from a level below. So it is warranted to treat the functions modifying a level as functions also modifying the level immediately above – provided the changes propagated 'upwards' are compatible with the levels they are seen as happening in. We formalise this as follows.

Definition 38 (Operator compatibility) *Given a structure $S = \langle T_1, \ldots T_n\rangle \in \mathcal{S}$ of elements $T_i \in \mathcal{T}$ not further specified (i.e., which can be anything, e.g. other structures) we say that a function $f : x \mapsto \mathcal{F}$ is upwardly compatible if $\mathcal{F} \subseteq \mathcal{T}$, i.e., its domain is part of the 'higher' level's structures' domain. Calling functions in this 'higher' level $g : y \mapsto \mathcal{S}$, we denote such a function*

Level	Function	Function $\upuparrows$	Function $\uparrow$	Function $\downarrow$	Function $\downdownarrows$
Π	ω_Π	$\omega_\gamma \upuparrows^{\omega_\Pi}$	$\omega_\xi \uparrow^{\omega_\Pi}$		
ξ	ω_ξ		$\omega_\gamma \uparrow^{\omega_\xi}$	$\omega_\Pi \downarrow^{\omega_\xi}$	
γ	ω_γ			$\omega_\xi \downarrow^{\omega_\gamma}$	$\omega_\Pi \downdownarrows^{\omega_\gamma}$

Table 4.1: Hierarchical function levels used in genetic algorithms

$f \uparrow^g$. Note that x and y are ranges for the functions f and g, respectively; their actual contents are immaterial (although they more often than not are the same as the domains, i.e. $\mathcal{F}$ and $\mathcal{S}$, respectively). We can easily extend this to even higher levels, denoting them with as many arrows as levels raised (e.g. $\upuparrows^g$); likewise, we can see them from above, so to speak, and extend downwards using $\downarrow^g$ and $\downdownarrows^g$, and so forth. For orders greater than 2, where the number of arrows become unwieldy, we can use $\uparrow^{(n)g}$ and $\downarrow^{(n)g}$ for n levels upwards and downwards compatibility, respectively.

We can visualise the situation as diagrammed in table 4.1, where we show the situation as applied to genetic algorithms with its three levels.

Using the operator compatibility syntax defined above, we can now easily define the projection operator for operators as follows, which will allow us to compose all our genetic operators in the ways we desire.

Definition 39 (Projection for operators) *The projection operator $\backslash_g$ extracts from an operator f those mappings that are compatible (according to definition 38) with the operator g. Formally, given two operators f and g, we define the projection of the operator as one of the following, depending on whether we desire upwards or downwards compatibility:*

$$f\backslash_g = f \uparrow^{(n)g} \tag{4.33}$$
$$f\backslash_g = f \downarrow^{(n)g} \tag{4.34}$$

where n is the number of levels where f can be found in relation to g.

Regarding operator composability, following standard practice, we define composition of functions as $f \circ g(x) = f(g(x))$; most often we omit the argument and say that $f \circ g = f(g(\ldots))$. However, the order does matter: a sequence of genetic operators may for example be in the order

$$\omega_\gamma = \omega_{\gamma_1} \circ \omega_{\gamma_2} \circ \ldots \tag{4.35}$$
$$\omega_\xi = \omega_{\xi_1} \circ \omega_{\xi_2} \circ \ldots \tag{4.36}$$
$$\omega_\Pi = \omega_{\Pi_1} \circ \omega_{\Pi_2} \circ \ldots \tag{4.37}$$

Any order is of course acceptable, but obviously the result may differ. In other words, commutativity between the different genetic operators does not necessarily hold; i.e. in general $\omega_1 \circ \omega_2 \neq \omega_2 \circ \omega_1$.

In order to be able to compose almost arbitrarily, the operators have to compatible in the manner shown above. In other words, if we for instance wish to compose on the population Π level, we must write something similar to the following, using definition 39

$$\omega_{\Pi} = \omega_{\Pi_1} \circ \ldots \circ \omega_{\Pi_n} \circ \omega_{\gamma} \backslash_{\omega_{\Pi}} \circ \omega_{\xi} \backslash_{\omega_{\Pi}} \circ \ldots \tag{4.38}$$

ensuring that we are operating at the same level at all times, considering the effects of the γ and ξ levels on the desired Π level as well.

Analogously with the virtual members of a population discussed above, we may use the concept of *virtual* genetic operators. We use them in order to affect a mapping from fitness value of an individual to fitness value of the *same* individual in the next interval, not from chromosome to chromosome. This is schematically depicted in figure 4.5 where the (real) operator ω is shown together with its dual, the (virtual) operator ω_v.

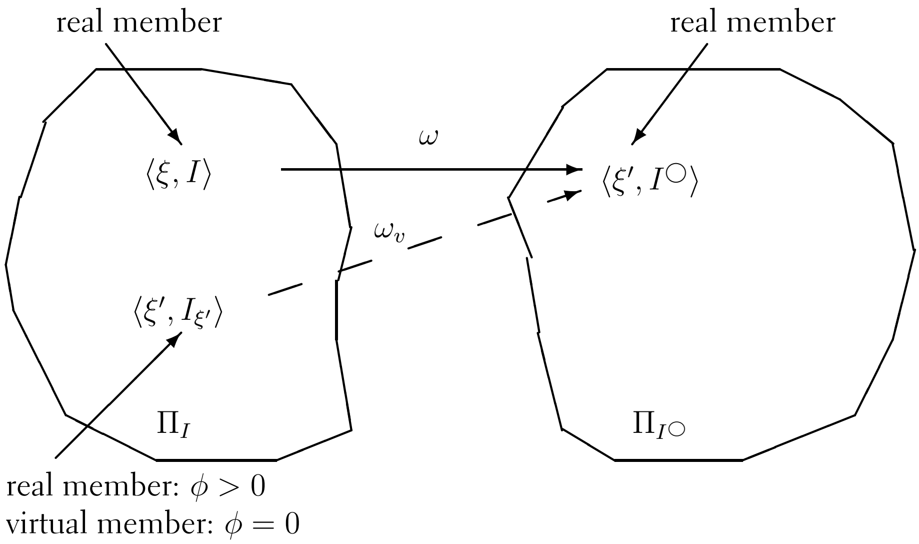

Figure 4.5: Virtual genetic operator

In order to use the virtual operators, we need to formally define morphisms between them and the real ones. In this way we may use the results interchangeably between the standard evolving structures and the virtual evolving fitnesses. Firstly, we need to obtain the fitness $\phi(\alpha)$ for an individual. This is easily accomplished by projecting out ξ from $\langle \xi, I \rangle$ and obtaining $\phi(\xi) = \phi(\alpha)$ from ξ using $\alpha\backslash_{\xi}$. The problem is then that there may be several individuals with the same fitness, i.e. $\phi : \xi \mapsto r \in \mathbb{R}$ is not one-to-one. This would mean that the fitness value $\phi(\alpha)$ for possibly several individuals in an interval could not be mapped uniquely to the next interval. In order to overcome this we need a bijective function from an individual to a value that could serve as a fitness value; i.e. one that may be uniquely mapped to the real fitness value $\phi(\alpha)$. We need to define these morphisms; see figure 4.6.

In a similar fashion, Battle and Vose show [BV91], in an interesting approach to characterising genetic operators using isomorphisms, that transformations of schemata is possible using algebraic formulations, including isomorphic genetic operators.

In our setting figure 4.6 depicts the needed mappings, where the following symbols have been used.

$$
\begin{tikzcd}
\Pi_I \arrow[r, "\omega"] \arrow[d, "f"'] & \Pi_{I\bigcirc} \\
M_I \arrow[r, "\mu"'] & M_{I\bigcirc} \arrow[u, "f^{-1}"]
\end{tikzcd}
$$

Figure 4.6: Necessary commuting transformations and populations

- Π_I and $\Pi_{I\bigcirc}$ are the population of chromosome individuals for the current and the next interval, respectively
- M_I and $M_{I\bigcirc}$ are the population of fitness individuals for the current and the next interval, respectively
- ω is the (canonic) genetic function from chromosome population to population
- μ is the (canonic) genetic function from fitness population to population
- f and its inverse f^{-1} are the transformation functions from/to chromosome and fitness, respectively.

The problem is the function f. It has to be injective, and an inverse must exist. The inverse should be bijective, i.e. injective and surjective. We construct this function in the following manner. Recall that each individual has a fitness $\phi = \phi(\xi) = \phi(\alpha) \in [0, 1]$. We wish to use natural numbers, instead of real numbers. Denoting $\mathbb{N}_0 = \{0\} \cup \mathbb{N}$ it is trivial to define a function $g : \mathbb{R} \to \mathbb{N}_0$ to affect the converse transformation. So we define a function

$$f : \mathcal{S}_\xi \to \mathbb{N}_0 \tag{4.39}$$

where f maps the individual fitness values by first ordering all individuals α in ascending order on $\phi(\xi)$. Assume the number of different values is $N \leq |\mathcal{S}_\xi|$. Note that we use $\mathcal{S}_\xi$ instead of $\mathcal{S}_\alpha$ since the former is finite, whereas the latter is not. Then assign each fitness value a unique natural number following the rules

1. First we divide the natural number axis into increments of $2 \cdot N$, i.e. $0, 2 \cdot N, 4 \cdot N \ldots 2 \cdot N^2$.
2. The first number is 0, corresponding to the first (lowest) real fitness (in practice almost certainly 0, i.e. not 'alive').
3. For each individual
   - if its real fitness value is different (i.e. higher) than the previous one, we assign the individual the natural number of the next increment,

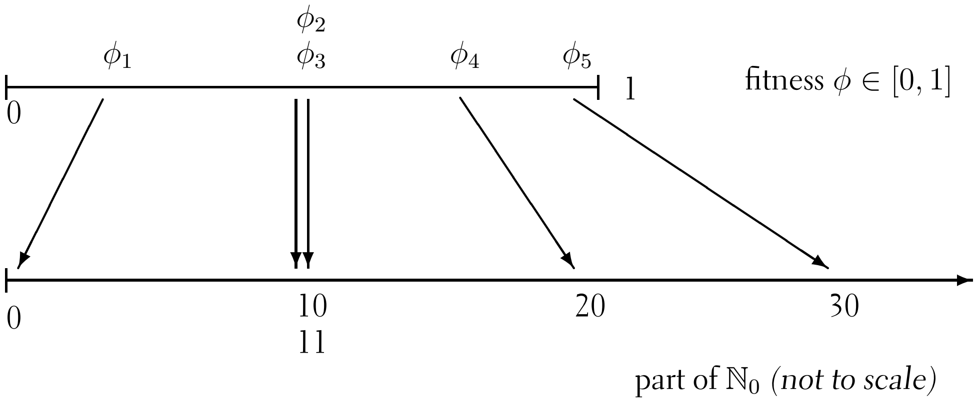

Figure 4.7: Fitness numbers

- if its real fitness value is the same as the previous individual, we assign it the value of the previous individual incremented by one.

4. Continue until all individuals have been assigned a corresponding natural number.

Thus f maps a chromosomal individual to a unique natural number (its *fitness number*), fulfilling the conditions above. The assignment algorithm also ensures that the order is ascending, as well as in the same order as the original fitnesses. See example 8.

Example 8 (Fitness numbers) *Suppose we have a population of five individuals, with varying fitnesses. We wish to assign them unique fitness numbers. See figure 4.7.*

In the figure we have a total of five individuals and their fitnesses, $\phi_1 \ldots \phi_5$, where ϕ_2 and ϕ_3 are equal, the rest all different, making $N = 5$. The assignment proceeds by first assigning ϕ_1 the number 0, ϕ_2, which is different (higher) than ϕ_1 gets the number $1 \cdot 2 \cdot N = 2 \cdot 5 = 10$. Next, since ϕ_3 is equal to $_4$ and ϕ_5, being different from each other and higher than ϕ_3, get the numbers 20 and 30, respectively ($2 \cdot 2 \cdot N$ and $3 \cdot 2 \cdot N$).

The sequence of fitness numbers has several interesting properties. Firstly, it is monotonically ascending, by construction. This means that each possible individual α in $\mathcal{S}_\alpha$ is mapped to a unique fitness number, the value of which depends on its fitness as calculated by $\phi(\alpha)$. f is thus clearly injective, whereas the inverse f^{-1} is bijective, which is exactly according to our demands. In other words, $f^{-1} : f(\mathcal{S}_\xi) \mapsto \mathcal{S}_\xi$. This of course has the consequence that each mapping ω from (chromosomal) individual to individual may be represented in the 'fitness' space $\mathbb{N}_0$ by a unique mapping μ. Secondly, by construction we have defined a range of fitness numbers 'around' each fitness increment ($2 \cdot N$) serving as a range of equality (equivalence interval) for each fitness value ϕ, i.e. a range where we have a set of different individual having the same fitness. We call this the *equivalence interval*. The size of the interval is N, which explains the choice of $2 \cdot N$ above ensuring nonoverlapping equivalence intervals, and thus also ensuring the uniqueness of each fitness number.

$$\begin{tikzcd} \Pi_I \arrow[r, "\omega"] \arrow[d, "f"'] & \Pi_{I\bigcirc} \\ M_I \arrow[r, "\mu"'] \arrow[ur, "v"] & M_{I\bigcirc} \arrow[u, "f^{-1}"'] \end{tikzcd}$$

Figure 4.8: Populations and commuting transformations, normal

This is quite similar to the approach taken by Goldberg in [Gol90] and Mahfoud in [Mah93, Mah95], where the authors effectively partition the (real) fitness space into classes with sets of individuals belonging to each class, formally assigning each class the same fitness.

Note that in the following exposition on commuting transformations we use compositions of functions in the way used in category theory; in other words, we use $f = f_1 \circ f_2$ to denote the composition $f(x) = f_1(f_2(x))$. The reason is that it is much easier to follow the category theory way when describing and using commutative diagrams, cf. figure 4.6 or figure 4.8.

We now have the required equivalences (cf. figure 4.6 and 4.8):

$$\omega = f \circ \mu \circ f^{-1} \tag{4.40}$$

$$\mu = f^{-1} \circ \omega \circ f \tag{4.41}$$

In order to formulate a representation of the virtual genetic operator v we obtain the commutative diagram shown in figure 4.8. It is clear that this diagram commutes without the v arrow; by definition this added arrow must also commute with the rest of the diagram.

Note that since $f = f^{-1} \circ f$ (or $\alpha = f(f^{-1}(\alpha))$) we also have the commutative diagram in figure 4.9.

$$\begin{tikzcd} \Pi_I \arrow[r, "\omega"] & \Pi_{I\bigcirc} \arrow[d, "f"] \\ M_I \arrow[r, "\mu"'] \arrow[u, "f^{-1}"] \arrow[ur, "v"] & M_{I\bigcirc} \end{tikzcd}$$

Figure 4.9: Populations and commuting transformations, reversed

Directly from the diagrams we thus obtain the following for v.

$$v = \mu \circ f^{-1} \tag{4.42}$$

$$= f^{-1} \circ \omega \tag{4.43}$$

	Item	Description
1	Form parents	Typically this is achieved by copying population members in some form and proportion to an intermediate population
2	Select parents	Typically this is done using a *selection scheme* that ensures that fit individuals have a higher survival rate than unfit ones
3	Generate offspring	The selected individuals are further subject to genetic operators that with some probability generate new individuals, thus exploring the search space induced by the genotype
4	Test completion	If an acceptable termination criterion is met we conclude the process, otherwise go back to stage 1

Table 4.2: Main genetic algorithm stages

as well as the trivial $\omega = f \circ v$. This means that v may be defined as a function $v : \mathbb{N}_0 \rightarrow \mathcal{S}_\alpha$, or $v : n \mapsto \alpha_{I\bigcirc}$, where n is a fitness number as described above. Or, casting this into functional form, we may say that $v(n) = \omega(f^{-1}(n))$, where n is a fitness number.

We are now ready to formally define the previously informally presented genetic algorithm.

4.3 FORMAL GENETIC ALGORITHM

This section will use the machinery introduced in the preceding sections to characterise the working of the genetic algorithm from a formal point of view. There are many formal versions based on other characterisations, e.g. overviews and set-based in [Gol89, HB92, Hol92a], as well as algebra-based, notably by Radcliffe and Surry [Rad91, Rad93, Rad94a, Rad94b].

In a way similar to Hoffmeister and Bäck in [HB92] a genetic algorithm GA can be modelled as a 5-tuple

$$GA = \langle \mathcal{S}_\gamma, \mathcal{S}_\xi, \mathcal{S}_\Pi, INT(\mathbb{Z}), \Omega \rangle, \tag{4.44}$$

using the symbols introduced in the preceding section. However, this model is static in nature, and does not take into account the more interesting dynamic behaviour of the genetic algorithm.

As described in the introductory chapter, following the formation of an initial population the genetic algorithm goes through the four stages listed in figure 4.2.

Note that, as has been shown in the previous section the order of selection and recombination does not matter: typically it is the one shown.

We form the initial population by randomly selecting a suitable number of individuals from all possible individuals. In our formalism, using infinite time we may characterise the initial population as follows.

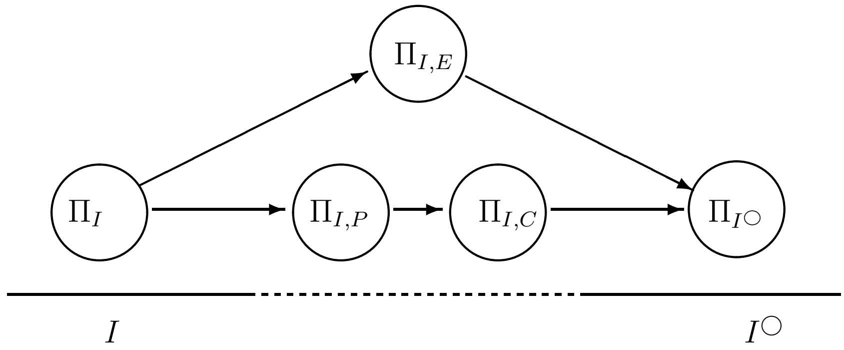

Figure 4.10: Genetic algorithm populations. For symbols see text.

Item	**Description**
Π_I	Original population at interval I
$\Pi_{I,E}$	Elite population saved from interval I to $I^{\bigcirc}$ (does not undergo selection)
$\Pi_{I,P}$	Parent population (undergoes selection)
$\Pi_{I,C}$	Child population selected
$\Pi_{I^{\bigcirc}}$	New population at interval $I^{\bigcirc}$

Table 4.3: Populations in the genetic algorithm

Definition 40 (Initial population.) *A population* Π *for which*

$$\exists I_0, \forall I < I_0, \Pi_I = \Pi \tag{4.45}$$

holds, where Π_I *is the population in interval* I*, is called initial. We denote this population* Π_0*.*

The initial population is thus the one where the population in the next interval differs for the first time, or $\Pi_{I_0^{\bigcirc}} \neq \Pi_0$. We may number the populations with a subscript, $\Pi_0, \Pi_1 \ldots \Pi_n$, but since the initial population is in no way special in terms of the subsequent application of the genetic algorithm, we omit the subscript, and call the population simply Π.

Having formed a population we need to form the set of parents, the size of which may be smaller, the same, or larger than the full population. Thus we continue with selection of those individuals that may participate in procreation - i.e. the parents of the next generation (the offspring).

The whole process may be depicted as in figure 4.10. The various populations, from the original to the new population, are listed in table 4.3.

Of these, the Π_E population constitutes an often used, but optional *elite* population, one that is deemed so fit that it is included without undergoing selection (and thus competition) in the population for the next interval. It may be formally defined as follows.

Definition 41 (Elite population) *An elite is part of a population that is automatically included in the next interval's population because the fitness value of its individuals exceeds an elite threshold,* ϕ_E*. Symbolically*

$$\Pi_E \ = \ \{\alpha \mid \alpha \in \Pi \wedge \phi(\alpha) > \phi_E\} \tag{4.46}$$

Note that, in general ϕ_E is dependent on the interval, i.e.

$$\phi_E = \phi_E(I) \tag{4.47}$$

where I is the interval under consideration. This function might be used to select the (one) best individual for automatic survival, for instance.

It is important to realise that the elite, if any, is bypassing the normal mechanism for individual evolution in the genetic algorithm. Note that this means that the elite does not participate in reproduction; this is of course a choice that the designer of practical genetic algorithm may choose to disregard. However, the effects of elitism may or may not be beneficial; for instance, many practical applications want to retain the best individual found so far, and incorporate elitism. On the other hand, it has also been proven by Rudolph that the canonic genetic algorithm with added elitism does not converge in the general case [Rud94]; cf. the section on convergence starting on page 67.

There are many ways in which the parent population (or *mating pool* [GD91]) may be formed. Note that this stage and the following in figure 4.2 are very often combined into a single *selection* stage [BT95, HB92, Whi94]. We will discuss selection below in section 4.4. The only certain characteristic seems to be that parents must be present in the population; the size of the parent population may be smaller or bigger than the original from which they are chosen (or copied, if an intermediate population is used – in which case there may be multiple copies of a particularly fit individual ensuring a better probability of survival of its genotype).

Thus, in some manner not further specified at this time a parent set $\Pi_{I,P}$ is generated, or chosen from the full population Π_I.

$$\Pi_{I,P} = \{\alpha_i \in \Pi_I \mid i \in K\} \tag{4.48}$$

where K is an index set indexing the population of $\Pi_{I,P}$ with $|\Pi_{I,P}| = |K|$. Since we do not require distinct individuals α_i in $\Pi_{I,P}$, $\Pi_{I,P}$ may formally be seen as a multiset. For instance, in a selection scheme called linear ranking (see table 4.4 on page 55) we purposefully duplicate individuals in order to bias selection according to fitness. Note that this implies that in the general case $\Pi_{I,P} \nsubseteq \Pi_I$, and that we cannot say whether $|\Pi_I| = |\Pi_{I,P}|, |\Pi_I| < |\Pi_{I,P}|$, or $|\Pi_I| > |\Pi_{I,P}|$. This has important consequences, as we shall see below.

A genetic algorithm uses recombination (or *crossover*, short form for *crossing over*) as the basic means of generating new individuals. In order to be able to employ recombination we need to select the individuals whose chromosomes we are going to combine. To this end the parent set $\Pi_{I,P}$ is partitioned into one or more *mating* sets $\Pi_{I,P1}, \Pi_{I,P2} \ldots \Pi_{I,Pn}$. We then select an individual from each mating set forming the *recombination* set, which constitutes all individuals from whom chromosomes will be combined, using a chromosome function as defined in the previous section. We may say that these form *parent tuples* $\langle \alpha_1, \alpha_2, \ldots \alpha_n \rangle$, where each α_i is from the corresponding $\Pi_{I,Pi}$. Following the recombination we have a set of *offspring* *tuples* $\langle \alpha'_1, \alpha'_2, \ldots \alpha'_m \rangle$. Together, all the offspring tuples form a *child* set $\Pi_{I,C}$. In general, we may state that

$$\begin{aligned} \Pi_{I,C} &= f_\Omega(\Pi_{I,P1}, \Pi_{I,P2}, \ldots, \Pi_{I,Pn}) && (4.49) \\ f_\Omega &= \omega_1 \circ \omega_2 \circ \ldots \circ \omega_k && (4.50) \end{aligned}$$

where f_Ω is a composition of all applied genetic operators as described in the preceding section. Note that this is equivalent to the earlier formulation of ω_Π (equation 37). In particular, we note that selection and recombination are included.

The new population $\Pi_{I\bigcirc}$ is formed from the elite and the child populations.

$$\Pi_{I\bigcirc} = \Pi_{I,E} \cup \Pi_{I,C} \qquad (4.51)$$

where $\Pi_{I,E}$ is the optional elite population described above, which determines which individual survives unopposed. Also note that, even if not a formal requirement, we in all practical genetic algorithms have $|\Pi_I| = |\Pi_{I\bigcirc}|$, i.e. we adjust the selection in such a way that our population size does not change. Exceptions to this rule are few, only the socalled Pareto genetic algorithm can use populations whose size change. Also note that the new population's individuals will of course have their intervals suitably adjusted (lengthened for surviving ones, or set for new ones) to include the appropriate interval.

This process continues until the stopping criterion has been met.

We may now characterise the two main types of genetic algorithms.

Definition 42 (Generational GA) *A generational genetic algorithm is one where*

$$\Pi_I \subseteq \Pi_{I,P} \qquad (4.52)$$

i.e. where the whole population participates in offspring creation (and consequently none is left out).

Equivalently, the condition may be written

$$\Pi_{I,P} = \Pi_I \textit{ iff } |\Pi_I| = |\Pi_{I,P}|, \qquad (4.53)$$

which in practice often is the normal case.

Definition 43 (Steady-state GA) *A steady-state genetic algorithm is one where*

$$|\Pi_{I\bigcirc}| > |\Pi_{I,C}| \qquad (4.54)$$

i.e. where only a part of the population generates offspring. The shortfall is made up of individuals coming direct from Π_I*, through* $\Pi_{I,E}$*.*

In a steady-state genetic algorithm we have that typically $|\Pi_{I,P}| \ll |\Pi_I|$. We can also immediately show that an elitist genetic algorithm strictly speaking is not generational.

Theorem 4 *An elitist genetic algorithm is not generational.*

Proof. Since in this case $\Pi_{I,E} \neq \emptyset$ and $\Pi_{I\bigcirc} = \Pi_{I,E} \cup \Pi_{I,C}$ according to equation 4.51 we immediately have that $|\Pi_{I,C}| < |\Pi_{I\bigcirc}|$ and definition 43 is fulfilled. Do note that $\Pi_{I,E} \cap \Pi_{I,C} = \emptyset$ because the individuals in the two populations have different intervals (the children are newer than the elite). □

In other words, looking at the genetic algorithm from a formal point of view the only difference between the two types is in the individuals of the parent set. As has been pointed out by e.g. Reynolds and Gomatam in [RG96] the two kinds may also be seen for the generational genetic algorithm as one without replacement into the population but instead forming a new population, whereas the steady-state genetic algorithm uses replacement and no new temporary populations.

It should be noted, though, that many different variants of genetic algorithms have been described in the literature, and that the sequence of steps, and indeed the steps themselves, are peculiar to each type of genetic algorithm; see e.g. [BS93].

4.4 SELECTION AND THE POPULATION FITNESS FUNCTION

Selection, together with recombination is crucial in the genetic algorithm. They determine which individuals may reproduce and how new offspring are produced. In a temporal and probabilistic setting as outlined above we are interested in the temporal characteristics of parents and offspring: who survives, what are the conditions and possible alternatives, and so on.

The selection function modelled by ω_Π above is most often based on a random selection from the parent set $\Pi_{I,P}$, where the probability of an individual being chosen is directly proportional to its fitness.

Selection is performed in such a way that individuals are selected for in accordance with their fitness, so that the more fitter survive (statistically speaking) and the less fitter do not. The key point here is that, *on the average*, new individuals are fitter than old ones. Mathematically, we could say that

$$a \in \Pi_{I\bigcirc}, b \in \Pi_I : P(\phi(a) \geq \phi(b)) > 0.5 \tag{4.55}$$

or, paraphrasing, on the average $\phi(a) \geq \phi(b)$.

As pointed out above, the formation of the parent set and the subsequent selection process of individuals for offspring creation is most often combined. We have the following common selection strategies, as listed in table 4.4 [BT95, GD91, HB92, MSV95, Sys91] .

There are a number of aspects which characterise the selection, convergence and fitness in standard genetic algorithms [BT95, GD91]. Some of the most common are listed in table 4.5.

From our point of view these parameters need to be redefined to provide meaningful information in a temporal context.

The loss of diversity may be readily transcripted to a temporal context.

Item	Description
Tournament	Selection scheme where a set of individuals of a certain size (*tournament size*, often 2) is randomly chosen, and the best is retained into the child population. Repeat until child population is full.
Proportionate	Selection schemes where we choose individuals with a probability directly dependent on their fitness. Examples include roulette-wheel selection [DJ75] [Gol89], and various stochastic remainder schemes [GD91].
Linear ranking	Selection scheme where we first sort individuals into ascending order on fitness, then assign each individual a number of copies into the intermediate population based on an increasing function on rank.
Truncation	Select a certain percentage of the population for mating, with equal probability [MSV93].
Steady-state	Selection scheme where we work individual by individual, choosing a parent by linear ranking, and replacing the currently worst individual [Sys91]

Table 4.4: Popular selection mechanisms

Item	Description
Loss of diversity	Denotes the narrowing of the current population diversity due to deselection of 'bad' individuals from the population
Reproduction rate	Denotes the ratio of the number of individuals with a certain fitness before and after selection
Selection intensity (pressure)	The expected average fitness of the population after selection. Blickle and Thiele [BT95] point out that since this is dependent on the initial fitness distribution, initially we should have a normalised Gaussian distribution $G(0, 1)$
Takeover time	Denotes the time until the population can no longer be improved; essentially when it consists of only one 'best' individual; cf. loss of diversity above

Table 4.5: Selection parameters

Definition 44 (Loss of diversity) *The proportion of the population of an interval I that was not virtual (i.e. was real) in the previous interval is called the loss of diversity l_I. Recalling that the previous interval is denoted $I^{\overline{\bigcirc}}$ (definition 12), we have*

$$l_I = \frac{|\Pi_I| - |\Pi^1_{I^{\overline{\bigcirc}},v}|}{|\Pi_I|} \tag{4.56}$$

for a certain interval I. Note that we have explicitly indicated that we mean the 1-generation virtual population (see p. 39f) with the superscript 1 for clarity.

Loss of diversity l_I is related to the variance of the fitness in the population. It can be shown [Lei95] that, for proportionate selection (e.g. roulette wheel) the mean fitness $\overline{\phi}$ of a population changes depending on the standard deviation σ^2 (for a definition of statistical parameters see any general text on probability and statistics, e.g. [Pap65]) as follows

$$\overline{\phi}^{\bigcirc} = \overline{\phi}(1 + \frac{\sigma^2}{\overline{\phi}^2}). \tag{4.57}$$

In other words, mean fitness changes less and less as the genetic algorithm converges, *unless* we can ensure that variance σ (or standard deviation σ^2) remains high. Note that equation 4.57 only applies to roulette wheel selection, although the broad statement "improvement is dependent on variance" remains true [Lei95].

In order to better characterise ratios of individual characteristics, especially regarding fitness, we need to be able to lump individuals with almost equal fitnesses together.

In general, a fitness function $\phi(\alpha)$ provides a value for each individual used to grade them; this value is denoted ϕ_ξ (because it solely depends on the chromosome in the individual as previously defined; definition 28). The problem is that this value is often too fine-grained; in other words, too discerning for applying a simple larger-than test. This is especially true when we want to be able to compare individuals in the fitness space, as described above. If this is the case we may 'coarsen' it by defining a range within which two fitness values are considered "equal enough" to be considered the same. Thus we define a equality environment (or equivalence interval) as follows.

Definition 45 (Equality environment) *The equality environment ϵ is a range of fitness values where individuals' fitnesses are considered equal. For two individuals α_1 and α_2 we write*

$$\alpha_1 \cong_\epsilon \alpha_2 \tag{4.58}$$

when the following fitness condition holds

$$|\phi(\alpha_1) - \phi(\alpha_2)| \leq \epsilon \tag{4.59}$$

This is readily expanded to any number of individuals.

Similarly, the set $\tilde{\alpha}$ of individuals in an equality environment ϵ around the given individual $\alpha \in \Pi_I$ is

$$\tilde{\alpha} \;=\; \{\beta \in \Pi_I \mid \alpha \cong_\epsilon \beta\}. \tag{4.60}$$

Varying the value of ϵ we can now obtain any granularity desired. For comparing individuals in fitness space that have identical chromosome space fitnesses, for instance, the value would be $\epsilon = \frac{1}{N}$, where $N = |\Pi_I|$, the number of individuals in that interval (normally equal to the constant population size). This is very similar to the approach taken by Goldberg in [Gol90] and Mahfoud in [Mah93, Mah95].

Note that since both ϕ and ϵ may be seen as free variables, then we should write the above as

$$\tilde{\alpha}(\phi, \epsilon) \;=\; \{\beta \in \Pi_I \mid \alpha \cong_\epsilon \beta\} \tag{4.61}$$

to emphasise that the set of individuals in the equality environment is dependent on these two parameters.

Note that we will drop the subscript ϵ from $\cong_\epsilon$ and write just $\cong$ below since we will not use the $\cong$ symbol for any other purpose.

There are obvious parallels here with fuzzy logic, and fuzzy sets, as first described by Lotfi Zadeh in 1965 [Zad65]. For an excellent introduction to fuzzy modelling, see e.g. [Bez93]. In fuzzy logic we also consider the truth value of a proposition to be dependent on the relative "strengths" of its constituents, so that for instance, two elements may be considered equal if they are "close enough", as determined by the fuzziness of the logic in question.

The reproduction rate may now be characterised for an equality environment. This definition essentially parallels the one in [BT95].

Definition 46 (Reproduction rate) *The reproduction rate r_I for a given fitness ϕ and equality environment ϵ is the ratio of the number of individuals within the equality environment for an interval to the corresponding number in the preceding interval. Formally, given the equality environment parameters ϵ and ϕ and equation 4.61 we have the following definition for r_I.*

$$r_I(\alpha) \;=\; \frac{|\tilde{\alpha}_I|}{|\tilde{\alpha}_{I\overline{\bigcirc}}|} \tag{4.62}$$

From a temporal point of view the selection intensity above is not meaningful. Instead, as the basic idea behind the selection intensity, namely the progress toward fitter and fitter individuals simultaneously implies that the existing individuals will be more and more difficult to replace. This, in turn, implies that the age will increase – and that the proportion of survived individuals from a previous interval will increase. We define the survival rate as follows.

Definition 47 (Survival rate) *The ratio of the number of individuals from the previous interval still surviving in the current interval to the current pop-*

ulation size is called the survival rate R_I. Using equation 4.5 we write

$$R_I = \frac{|\mathcal{S}^I_\alpha \cap \mathcal{S}^{I^{\overline{\bigcirc}}}_\alpha|}{|\mathcal{S}^I_\alpha|} \tag{4.63}$$

where I is the current interval.

Since R_I will be a value from 0 (none survived) to 1 (all survived) we define the selection intensity simply as follows.

Definition 48 (Selection intensity) *The selection intensity S_I is*

$$S_I = 1 - R_I \tag{4.64}$$

where R_I is the survival rate.

Above the takeover time is defined as the time before the population consists of at least $|\Pi| - 1$ copies of the best individuals, measured using the fitness [GD91]. In our context this implies that the corresponding fitness space individuals should belong to the same equality environment since they by definition do *not* have equal fitnesses.

Before continuing we need to define the age concepts relating to our formalism of the genetic algorithm.

Definition 49 (Ages) *The age $a(\alpha)$ of an individual is the number of consecutive base intervals it has belonged to the (real) population. The age $a(\Pi_I)$ of a population is the number of base intervals since the initial population.*

We may calculate the mean age of a population simply as

$$\overline{a(\Pi_I)} = \frac{\sum_{\alpha \in \Pi_I} a(\alpha)}{|\Pi_I|}. \tag{4.65}$$

Looking at the situation from a fitness perspective, takeover has occurred when a sufficient number of individuals are in the same equality environment. At that time, and in following intervals, the fitness profile, i.e. the actual equality environments present in the population will no longer change, but stay constant. In other words, the genotypes of the individuals may change but their fitnesses stay in the same equality environment. What this means is that since the takeover time as defined by Goldberg and others [GD91] is dependent on the selection mechanism (and is indeed different for each one) we cannot define the takeover without specifying the selection used. However, a characterisation based solely on fitness would not be, provided the aim is to maintain or improve the fitness of the population. So we say that takeover has occurred when all individuals in the population belong to the same equality environment. This is formalised in definition 50.

Definition 50 (Takeover) *Takeover has occurred when all individuals belong to the same given equality environment. Formally, given an equality population $\tilde{A}$ with its equality environment takeover has occurred when*

$$\exists \tilde{A} \quad | \quad |\tilde{A}| = |\Pi| \tag{4.66}$$

Unfortunately, this equation does not help us in formulating an estimate for the takeover time (à la Goldberg and Deb in [GD91]). That result depends on the selection mechanism, and assumes a fitness function behaving in a certain way (either linearly or exponentially). No such assumptions are possible here.

We now have the machinery in place for analysing the genetic algorithm from a temporal point of view. However, there is a major problem that may be addressed from this viewpoint: premature convergence, and the corresponding loss of diversity; see e.g. [Gol89]. There are several ways of alleviating this problem. Some have used fitness scaling [Gol89], needing a lot of effort to tune properly. Fitness sharing, also due to Goldberg, as well as niching [Mah95] have also been tried. Methods combining simulated annealing with genetic algorithms have recently been reported [MYT$^+$95].

To deal with the problem in this context a novel selection method, based on temporal parameters, suggests itself. This temporal selection method will be described in the next chapter.

5 THE GENETIC ALGORITHM AND TPL

This chapter will examine the genetic algorithm, its behaviour and characteristics using the formalism described in the previous chapters. We will show that the temporal and probabilistic approach, using TPL, is a viable alternative for the formalisation of the genetic algorithm, as well as that it provides us with a good tool for better understanding the behaviour of the genetic algorithm.

This chapter proceeds as follows. We will first describe temporal selection, with its two main variants. Then we will look at conditions for optimum using the new formalism, and thereafter show the connection between the statistical probability as formulated earlier and the average fitness of the population in a genetic algorithm. Then we will show necessary conditions for achieving optimum in terms of TPL, and prove that they hold in one class of genetic algorithms.

5.1 TEMPORAL SELECTION

Basically, the problem is that the genetic algorithm is intended to find optima (minima or maxima depending on the problem), which it does very well – regardless of whether they are local or global ones. As we are not interested in local optima we must somehow ensure that it doesn't converge *too* fast to one, but instead, having found one continues to explore other regions regardless of the fact that a putative optimum may have been encountered. The phenomenon of premature convergence (to local optima) is prevalent in the standard (canonic) genetic algorithm, and the standard way of alleviating the problem is to scale the fitness function so that small deviations in real fitness give rise to large changes in the function's value - thus discriminating between values with a very fine comb.

We wish to take another approach, superficially similar to Goldberg [Gol90] and Mahfoud [Mah93], where we instead lump values together into an equality environment. This is also similar to Jones' idea of *neighbourhoods* in the fitness landscape [Jon95]. In this work the idea is simply that we will consider the individuals equal within the equality interval, and select on the basis of their ages instead. We also make the assumption that the equality interval $\epsilon \ll 1$, i.e it is but a fraction of the whole fitness range [0,1]. We'll describe the case with just two values, which may readily be generalised.

In order to be able to choose individuals inside the equality interval, we need to consider some other criterion beside their fitnesses. We choose their *ages*, as defined in Definition 49.

So, when selection occurs, if two fitnesses are considered equal, we have to determine which individual survives. In this circumstance we have two cases.

Definition 51 (Time-progressive) *When the newer individual survives to the next generation we call selection time-progressive. We can also call it fitness-conservative, indicating that the fitness is preserved even if the individual is not.*

Definition 52 (Time-conservative) *When the older individual survives to the next generation we call selection time-conservative.*

Formally, we may describe this selection mechanism as follows. Given a population Π with two individuals α and β in the same equality environment, we have that $\alpha, \beta \in \Pi \cong \beta$. In time-progressive selection this implies that $\alpha \in \Pi^{\bigcirc}$ iff $a(\alpha) \leq a(\beta)$. Correspondingly, in time-conservative selection $\alpha \in \Pi^{\bigcirc}$ iff $a(\alpha) > a(\beta)$. In other words, survival of α depends on its age $a(\alpha)$. In order for these concepts to become clear, consider example 9.

Example 9 *We will make a simplified simulation of the first few generations of a simple genetic algorithm, with a population of 6 individuals. The situation is illustrated in figure 5.1.*

The figure shows two parallel scenarios: a time-conservative (above) and a time-progressive (below). We start with an initial population (at left), and generate offspring (shown immediately to the right of the population). We want to keep the population constant, so only six individuals are allowed to survive. Next, new offspring are created (greyed to the right of the population). Number 8 of the old and number 10 of the offspring are sufficiently near in fitness to be considered equal, so we choose one. Here the two parallel populations diverge: the conservative one keeps the older, number 8, whereas the progressive one keeps the younger, number 10. We show one more generation, where numbers 2 and 13 are considered equal, and are chosen to survive accordingly. As can readily be seen, the two populations are starting to show marked differences.

It is also possible to characterise survival in terms of Shoham's interval proposition characteristics [Sho88, p. 48ff], with individuals taking the place of propositions and 'survival' denoting propositions being true over longer and longer, overlapping intervals. However, the formalism developed by Shoham is largely based on van Benthem [Ben83], and would not materially add anything to the discussion above (and below) except some more and somewhat difficult terminology.

However, it should be pointed out that temporal selection as outlined above only comes into play when there is a collision of fitnesses: in all other cases another, traditional selection mechanism, such as one of those listed in table 4.4, should be used. Temporal selection is very much a hybridisation of an existing selection method to take temporal aspects of the individuals into account in order to be able to discern between cases otherwise may be erroneously treated as the same.

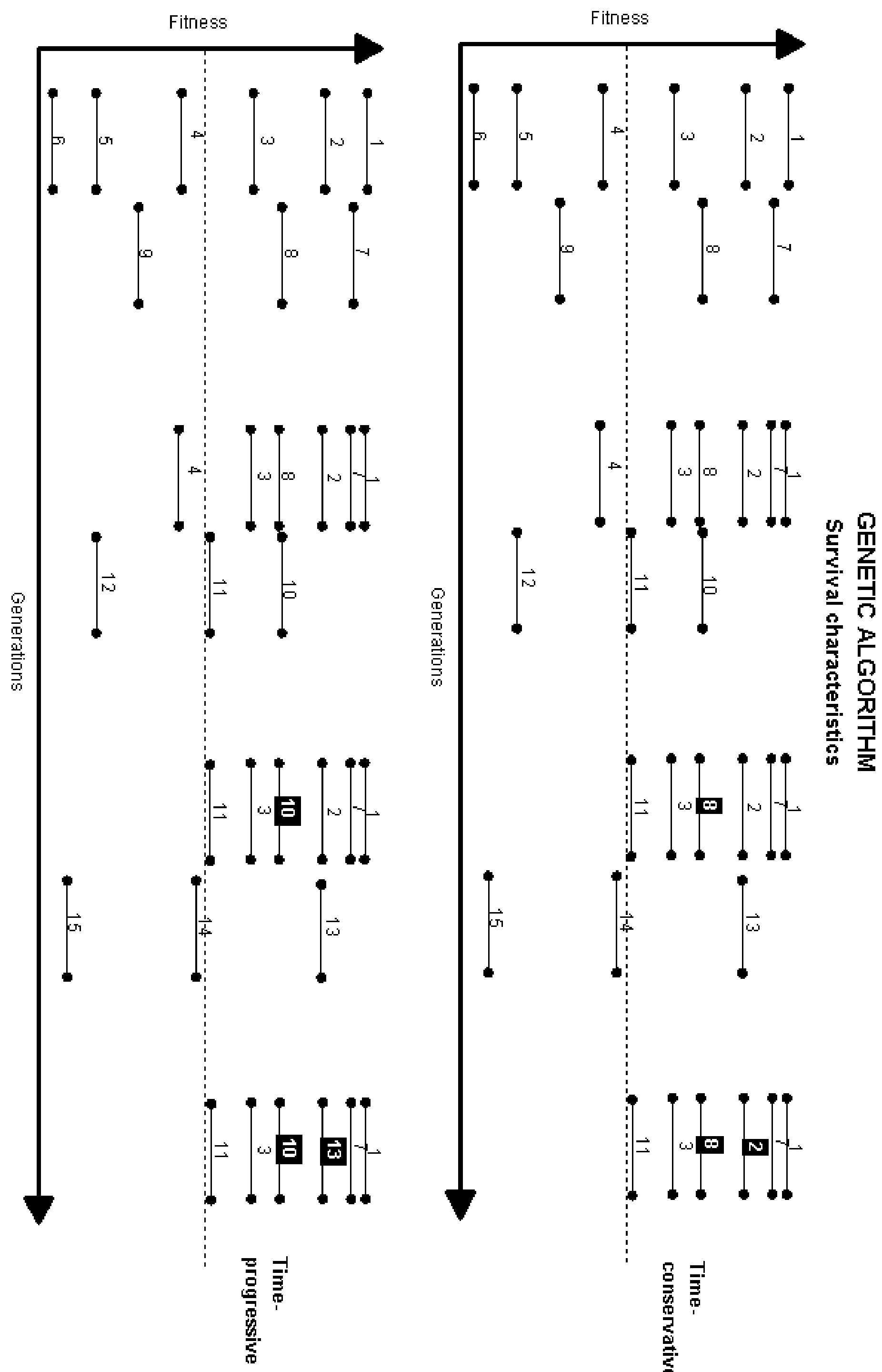

Figure 5.1: Conservative vs. progressive survival

5.2 OPTIMUM

How then does the genetic algorithm reach its goal, namely find an optimum (defined as a maximum or minimum, depending on the actual problem at hand)? In other words, given the initial population Π_0, a selection mechanism, and various genetic operators how do we reach an optimal state, with an optimal population Π_{opt}?

First, let us define fitness optima.

Definition 53 (Local optimum of population) *A local optimum of a population is a state of the population where the average fitness value of the population is at a maximum, i.e. any change to a fitness value of an individual will cause the average to decrease.*

Note that the definition is dependent on the genetic operators used; with different operators we may have many or few optima in fitness space.

Definition 54 (Global optimum of population) *The highest local optimum of a population is called the global optimum of the population.*

Note that there may be several global optima. These definitions mirror those of Haataja [Haa95]. It is clear that they do not constitute very practical definitions; normally, we are interested in finding just *one* individual with an optimal fitness, which then constitutes the solution to the actual problem at hand. However, from a theoretical point of view the meaningful definitions *for populations* are, in our opinion, as stated. This also closely parallels the usage of peaks by Jones in [Jon95], where he defines a ϕ-peak as a vertex in the fitness landscape (under the operator ϕ) where all its neighbours have fitness values that are less. Jones' global maximum is then the highest peak. As stated, the globally optimal population is one where all $n = |\Pi|$ individuals occupy the n highest fitness states; a situation that is both extremely unlikely in practice, and clearly theoretically *the* globally optimal population. In practice, a genetic algorithm has reached its solution when even one of its individuals has reached fitness optimum; however, this can hardly be said to be an optimal population in any theoretical sense even if it in all likelihood terminate the genetic algorithm. For instance, we often in practice operate with the average fitness of a population. The highest average fitness of a population is of course the global optimum given in definition 54.

On the other hand, how can we characterise an optimum population, in temporal terms? Firstly, it should be optimal with respect to something, namely its fitness, or rather, the fitness of its members (individuals). Now, we know that an individual may be replaced by another individual in two circumstances:

- a fitter individual, if such a one exists
- if not, an older (time-conservative) or newer (time-progressive) individual

However, in both cases the fitness does not decrease. Indeed, from a statistical point of view this characterises the genetic algorithm; although, depending on the selection mechanims chosen less fit individuals have a finite chance of being chosen. The reason for this is of course to ensure variability in the population.

Considering the behaviour of the individuals, we treat the time-conservative and time-progressive cases separately below.

5.2.1 Time-conservative optimum

In this case the older individual survives, if a child individual with the same fitness competes for survival. In other words, take two competitors α and β. Without loss of generality, we can assume that α is the older one, and that β is a new child vying for survival (i.e. $a(\alpha) > a(\beta)$). Using definitions 3 and 28 (and for a change explicitly incorporating time t instead of intervals in order to better illustrate the concept) we write

$$\begin{aligned} \alpha &= \langle \xi_\alpha, [t_{n-k}, t_n] \rangle \\ \beta &= \langle \xi_\beta, [t_{n-1}, t_n] \rangle \\ \alpha &\cong \beta \end{aligned} \tag{5.1}$$

Note that the interval end points are the same: t_n, denoting now, and that the start point for β is t_{n-1}, i.e. a new individual that cannot have existed prior to the current interval $[t_{n-1}, t_n]$. The corresponding start point for α is an unknown number of intervals earlier, given by t_{n-k}, and $k \geq 2$. Of course, k must be greater than 1, otherwise both are new child individuals, a case that cannot occur since new individuals do not compete among themselves, only with existing individuals in the population. Also note the constraint $\alpha \cong \beta$, i.e. that they belong to the same equality environment, and consequently their fitness is considered the same; otherwise there would of course be no contest and the fitter one would have a greater probability of survival.

Of course, we may also write the above using intervals, perhaps more concisely but less clearly

$$\begin{aligned} \alpha_I &\cong \beta_J \\ J &\sqsubseteq I \\ J^{\overline{\bigcirc}} &\; O \; I. \end{aligned} \tag{5.2}$$

Or, in words, the two individuals are α and β, in the same equality environment. As indicated by the subscripts they are in the intervals I and J, respectively, where β's interval J is included in α's interval I in such a way that J's predecessor $J^{\overline{\bigcirc}}$ overlaps with I; i.e. β did not exist at that time, and is consequently new. This latter condition may be written

$$J^{\overline{\bigcirc}} \sqsubseteq I, \tag{5.3}$$

relying on the fact the $\overline{\bigcirc}$ returns the previous base interval, which has to be included in I to give us the same conditions as in equation 5.1.

We may then observe that at optimum no individuals get replaced. The reason is simple: if an individual gets replaced, the only permissible cause is that the fitness of the new child is higher, in which case we were not at optimum; if it is lower, there is no contest, and, if it is the same (within the same equality interval) the older survives, per definition. We formulate the following theorem to characterise this situation.

Theorem 5 *Neglecting mutation, given a population* Π*, with individuals* $\alpha \in \Pi$*, in a time-conservative genetic algorithm at optimum the following holds.*

$$\forall \alpha \in \Pi \quad | \quad I^{\overline{\bigcirc}} \sqsubseteq I(\alpha), \tag{5.4}$$

provided $I > I_0$.

Proof. Equation 5.4 says that the intervals for all individuals in the population includes the previous interval. This means that no changes to the intervals have been made since the last interval. This is trivially true, by virtue of definition 52, equation 5.1, and the foregoing discussion. The latter provision takes care of the case where I is at the first, initial interval when $I^{\overline{\bigcirc}}$ doesn't exist. □

Theorem 5 may be paraphrased by saying that the starting point of the intervals of all individuals in the population stay the same. The population is static.

In practice, although this is true, it is a transient situation and is rarely allowed to happen, and certainly not to persist. The reason is that the genetic algorithm may have become stuck in a local optimum, neglecting to seek for a possible global optimum; this is the socalled premature convergence phenomenon [Gol89, Mah95, MYT+95]. We ensure that there always is a way for the genetic algorithm to proceed by incorporating *mutation* as a genetic operator, with a very low probability, as has been described previously. Note that this mutation has to be sufficiently 'big' in order to ensure that the individual in question leaves the local optimum.

5.2.2 Time-progressive optimum

The time-progressive case is more complex. We cannot say, like we did above, that at optimum the population is static since by definition we will replace an older individual by a new one, provided they have the same fitness, i.e. are in the equality environment. We may thus have individuals with lifetimes (intervals) with any starting value in the optimal population, however, all with a 'surviving' fitness value. This is the reason why we called the time-progressive *fitness-conservative* in definition 51 above.

We may thus formulate the following theorem.

Theorem 6 *Given a population* Π*, we first index it on the fitness values giving each individual an index number from 1 to* $|\Pi|$*. Neglecting mutation, in the time-progressive case, at optimum the fitness values of each individual* *with the same index stays constant from one generation to the next. Not only the average fitness value, but also the exact distribution of the fitness values over the individuals stays constant.*

Proof. By the definition of time-progressive (definition 51) we either do not replace an individual, or it is replaced by a new child individual with exactly the same fitness; so the distribution over the indexed individuals (i.e. the individual places) stays constant. □

For the time-conservative optimal population a slightly stronger theorem holds as well.

Theorem 7 *Neglecting mutation, in the time-conservative case, at optimum both the fitness values of each individual as well as the individual itself stay constant.*

Proof. Trivially true, since this is the definition of a static population. □

5.3 CONVERGENCE

For a genetic algorithm to work it must converge towards an optimum. Of course, we must first agree on what an optimum is, as well as what convergence (in terms of the population structure) means. First, let us define convergence.

Definition 55 (Convergence) *With convergence we mean that the average fitness of the population increases; clearly, in order for this to happen the fitness of one or several individuals must change. If only one changes, it must increase; if several, their average fitness must be greater that average of the rest of the population, so that the whole population's average fitness may increase.*

Convergence has been studied fairly extensively, but mostly in terms of the stochastic interpretation of the genetic algorithm; Markov chain analysis gives some insight into convergence [Rud94]. Summarising from [Fog94a] and [Fog95a] we have

1. The canonic genetic algorithm [Hol92a] does not converge [Rud94]
2. The simple genetic algorithm (SGA, see [Gol89]) converges when the population size tends to infinity [Goe95]
3. Elitist versions of the SGA do asymptotically converge to a global optimum [Rud94]
4. Evolutionary strategies and evolutionary programming have asymptotic global convergence properties [BS93]

Rather interestingly, Ankenbrandt [Ank90] has investigated the time complexity of genetic algorithms and found that time until convergence is proportional to $O(ln|\Pi|)$ under several not too stringent assumptions (among others, that such an algorithm is used that *does* converge).

In the following discussion, we assume that some genetic operators generate new individuals, from which the selection operator will choose survivors with a finite probability. If this is not the case, we will clearly not generate new individuals for fitness testing at all, but stay with the same population.

In temporal terms, we must again differentiate between the time-conservative and time-progressive cases.

In both cases, we initially have a situation where there is a high turnaround of individuals: unfit die and are replaced by fitter ones. As we will show, at the end, in the time-conservative case, the situation is static: all individuals survive, and their interval starting points stay the same.

Using previously introduced notation, we may then characterise convergence as follows.

Theorem 8 *In a converging time-conservative genetic algorithm population the number of old individuals will eventually outnumber new. Equivalently, the virtual population of the previous interval will constitute less than half the population. Formally, we have that*

$$\diamond \left(\left| \Pi_{I\overline{\bigcirc},v} \right| \leq |\Pi_I| / 2 \right). \tag{5.5}$$

Proof. Suppose that the reverse is true. Then, since new child individuals always outnumber old, surviving individuals, contrary to what the theorem claims the population will with each interval consist of more new individuals than old ones. Assume that (as a minimum) we get one additional new individual per generation (if we get none the population is not converging, which is the assumption – and we need at least one for convergence, as pointed out in definition 55). Then, given a population Π, after $|\Pi|$ generations all individuals will be new, replaced ones. So at the end we will have a population that is replaced each interval in its entirety. Clearly this is not the case at an optimum, as per theorem 5, so the population cannot converge. Since the assumption was that it was a converging algorithm, the theorem is true. □

Rather interestingly, the situation may be characterised by saying that in a converging time-conservative genetic algorithm, individuals will be slowly migrating towards inclusion in the equivalence interval, and thus be governed totally by the characteristics of the algorithm for the equivalence interval: time-conservative or time-progressive.

We may prove the following stronger theorem as well.

Theorem 9 *In the time-conservative genetic algorithm the size of the virtual population will eventually become zero. Formally, we have that*

$$\diamond \left(|\Pi_v| = 0 \right) \tag{5.6}$$

or equivalently

$$\diamond\left(\Pi_v = \emptyset\right). \tag{5.7}$$

Proof. The statement that the virtual population will become zero means that no individuals will be replaced any longer (recall that the virtual population *is* by definition the set of individuals that will exist - i.e. become 'alive' - in the next generation). Suppose this were not the case, i.e. we would always replace some individuals in the population. Then their fitness would have to be better (not only as good as) than that of the replaced individuals because we have a time-conservative algorithm where individuals are only replaced if their fitness is better, otherwise the older ones survive. Since the possible population space $\mathcal{S}_\xi$ is finite or at least denumerable (recall that a chromosome ξ has a finite length, and thus there are only a finite number of them, and that a chromosome consists of genes, and the allele space is finite or at least denumerable; cf. definition 26 on page 32), we would eventually pick up the $|\Pi|$ best individuals. At that time no more replacements may occur, as per definition 52. This is a contradiction, and the theorem stands. □

In the time-progressive case this is not true because individuals do get replaced, even if their fitness is as good as a new individual's. However, a similar theorem holds, using the fitness distribution instead.

When the population has converged, we may ask how many individuals have been examined in order to find the optimum. In other words, recalling corollary 3, how big did the cumulative virtual population $\Pi_{checked}$ become at convergence?

Theorem 10 *The size of the cumulative virtual population $\Pi_{checked}$ at generation n is*

$$|\Pi_{checked}| \;\leq\; |\Pi_0| + \sum_{i=0}^{n}(S_i \cdot |\Pi_i|) \tag{5.8}$$

where Π_0 is the initial population, Π_i the population at time i, S_i is the selection intensity at time i (see definition 48) and n is the current generation.

Proof. Following theorem 3 we immediately have

$$|\Pi_{checked}| \;\leq\; |\Pi_0| + \sum_{i=1}^{n}|\Pi_{v,i}| \tag{5.9}$$

Now following the definition of the selection intensity S (definition 48) we have the size of the virtual population at time i

$$|\Pi_{v,i}| \;=\; S_i \cdot |\Pi_i| \tag{5.10}$$

which after substitution gives us theorem 10. □

Taking the more common case of a constant population size, we can formulate the following theorem.

Theorem 11 *The size of the cumulative virtual population in a genetic algorithm with constant population size is*

$$|\Pi_{checked}| \leq |\Pi_0| \cdot (1 + \sum_{i=1}^{n} S_i) \qquad (5.11)$$

where Π_0 is the initial population, S_i is the selection intensity at time i (see definition 48) and n is the current generation.

Proof. Follows directly from theorem 10 by setting $|\Pi_i| = |\Pi_0|$. □

If we assume that the selection intensity S_i is constant we obtain the following simple relationship.

$$|\Pi_{checked}| \leq |\Pi_0| \cdot (1 + n \cdot S_i) \qquad (5.12)$$

This of course cannot be valid for the full execution of a genetic algorithm, but it may be valid for an appreciable number of generations at the beginning, when no convergence has yet taken place.

In any case, clearly $|\Pi_{checked}|$ is a finite value if we simply stop the algorithm say, after n generations; however, if we use a time-conservative selection scheme it will also converge toward a finite value since $\Pi_v = \emptyset$ according to theorem 9 at convergence (and thereafter, neglecting possible effects due to mutation).

This implies that at convergence the following holds for the *time-conservative* genetic algorithm

$$\sum_{i=1}^{\infty} S_i \leq \frac{|\Pi_{checked}|}{|\Pi_0|} - 1 = K \qquad (5.13)$$

where K is some constant depending on the actual parameters of the genetic algorithm used.

Furthermore, seeing $\Pi_{checked}$ as the population *that needs to be checked* in order to determine whether we have reached convergence, and the connection with the selection intensity, a measurable quantity, allows us to give a ballpark figure for how long a genetic algorithm should run before we have a reasonable chance of success in finding an optimum. What we need is a figure of the size of $\Pi_{checked}$ in terms of other independent parameters. From Goldberg and Holland we have the following formula.

The population $\Pi_{checked}$ that needs to be checked is

$$n \cdot 2^l < |\Pi_{checked}| < 3^l \qquad (5.14)$$

where n is the number of generations and each chromosome is of size l bits.

This follows directly from Goldberg's and Holland's so-called schema theorem [Gol89, Hol92a] where the authors show how many individuals are being examined provided the building block hypothesis and the schema theorem are considered valid.

This means that the result may be used to give an approximate answer: it defines (just like the authors' formula) an approximate bound. We can also immediately formulate the following formula.

The approximate number of generations needed in a generational genetic algorithm is given by

$$n < \frac{3^l}{|\Pi_0|} \tag{5.15}$$

where n is the number of generations and each chromosome is of size l bits and where we make the assumption that we 'physically' check $|\Pi_0|$ individuals each generation; the realistic number is always less or equal to this.

This is shown by direct substitution of $|\Pi_{checked}| = n \cdot |\Pi_0|$ in formula 5.14.

Note that the above formulas cannot be called theorems since they are only valid given the assumption that the controversial schema theorem and building-block hypothesis are correct.

Example 10 *Suppose we have a genetic algorithm with a fixed population of 100 and a chromosome of length 20. Then the number of generations necessary to ensure convergence is approximately* $3^{20}/100$ *or* $3.5 \cdot 10^7$ *(about 35 million).*

This gives us an approximate upper bound to the number of generations necessary; of course, looking at the example, the bound exceeds the capacity and time available of most current and future systems. However, this does not imply that an optimal individual has not been found long before; it only says that this number is (under the assumptions stated) necessary to ensure a convergent population.

Rather interestingly, this result is in sharp contrast to that of Worden [Wor95]. In his work on the speed limit of evolution, he shows that the rate of increase of the total *genetic information in the phenotype (GIP)* is a property of the population, and is not more that 5 bits per generation for mammals. He also points out that his results are applicable to genetic algorithms as well, although he does not derive a value for them. However, the implication is that, if we assume his formulations are correct for genetic algorithms as well, and for finite populations (as he assumes), we may improve the genotype by at most 5 bits per generation, which in turn gives us a lower bound for the number of generations needed to reach *any desired result* (or bit pattern). This would happen if the change is at its maximum of 5 bits.

Assuming a fixed population size we derive from his discussion the formula for the desired number of generations n

$$\frac{l}{5} < n < \frac{3^l}{|\Pi_0|} \tag{5.16}$$

where l is the length of the chromosome in bits and Π_0 is the initial population.

Example 11 *In the above example 10, having a population size of 100 with chromosome of 20 bits, we get a lower limit for the number of necessary* *generations of 20/5 or 4. This is a very low number considering the upper limit of around 35 million generations!*

However, this number (4 generations) seems to be borne out in practice as well; consider Chapter 8, where we show experimental data. For the standard fitness evolution for the baseline case for the first 10 generations (as shown in figure 8.3) convergence is clearly discernible and has reached almost its final value by the 4th generation. This is not appreciably altered by the choice of selection mechanism, as described in Chapter 8.

A second interesting implication of formula 5.14 is that it gives a upper limit for the size of the necessary initial population as follows.

The size of the initial population Π_0 is given by

$$2^l < |\Pi_0| \tag{5.17}$$

where the size of the chromosome is l bits. This is obtained by direct substitution of $|\Pi_{checked}| = n \cdot |\Pi_0|$ in formula 5.14.

Example 12 *In the above example 10, having a chromosome size of 20, the size of the population need not be larger than 2^{20} or slightly more than 1 million.*

It should be noted that these values (from examples 10–12) should not be taken too seriously as they give extreme values, and are valid only when the block building hypothesis and schema theorem are considered correct.

5.4 STATISTICAL PROBABILITY AND FITNESS

In order to investigate the connection between the statistical probability and population fitness, we recall the definition of an optimum (here taken as a maximum) for a fitness space in a genetic algorithm. Following [RF96] we have that

$$\begin{aligned} \phi(\alpha^*) \text{ is a local maximum} \quad &\Leftrightarrow \\ \exists\delta \quad &> \quad 0 : \forall\alpha \in \mathcal{S}_\alpha : |\phi(\alpha) - \phi(\alpha^*)| < \delta \Rightarrow \\ \phi(\alpha) \quad &\leq \quad \phi(\alpha^*) \end{aligned} \tag{5.18}$$

where $\mathcal{S}_\alpha$ is the possible space of all individuals and α^* is the local maximum. Correspondingly, we have the simpler condition for global maxima

$$\begin{aligned} \phi(\alpha^*) \text{ is a global maximum} \quad &\Leftrightarrow \\ \forall\alpha \quad &\in \quad \mathcal{S}_\alpha : \phi(\alpha) \leq \phi(\alpha^*) \end{aligned} \tag{5.19}$$

Before showing an interesting theorem, we recall that in probabilistic logic a predicate does not necessarily have a value of *true* or *false*, but can be given a probability of being true or false. Of course, once the variables in the predicate have been assigned values and a possible world where the predicate is to be examined has been chosen, the predicate does become a normal proposition in that world, and will have a normal truth value.

In terms of the probabilistic characteristics of the genetic algorithm, we can prove the following. We will use the *alive* predicate from Definition 33 on page 39; recall that $alive(x) : \phi(x) > 0$.

Theorem 12 *The average fitness $\phi_{avg}(\Pi_I)$ of a population Π_I in an interval I is equal to the statistical probability $[alive(\alpha_I)]_{\alpha_I}$. Formally,*

$$\phi_{avg}(\Pi_I) \;=\; \frac{\sum_{\alpha_I \in \Pi_I} \phi(\alpha_I)}{|\Pi_I|} \;=\; [alive(\alpha_I)]_{\alpha_I} \tag{5.20}$$

where we use $|\Pi_I|$ to denote the number of individuals in the population Π_I in interval I and α_I is a place-holder for the variable denoting an individual in the population of Π_I.

Proof. Recalling the direct inference principle (Axiom 7 on page 151) we have that

$$\mathbf{prob}(alive(\alpha_I)) = E([alive(\alpha_I)]_{\alpha_I}) \tag{5.21}$$

where E is the expectation operator. The left-hand side is then the probability of any individual α_I being alive (i.e, its fitness is greater than zero per the definition of the $alive$ predicate) in the population; whereas the right-hand side is a value for the statistical probability for individuals being alive. This value is constant across the worlds under consideration, and per the definition of the expectation value (see **I12** on page 142) it will be the proportion of the occurrence of the individuals in the worlds. Since this is constant we may drop the expectation operator E per **P15** on page 146. Now, by definition (see Definition 29 on page 34) the fitness is the probability of occurrence, then, for all of them the average value applies, which proves the theorem. □

In practice, this means that we may use the average fitness in logical formulas and ask interesting questions about the behaviour of the genetic algorithm.

For instance, will the average fitness increase?

Theorem 13 *The average fitness increases monotonically. Formally,*

$$\phi_{avg}(\Pi_I) \;\leq\; \phi_{avg}(\Pi_{I\bigcirc}) \tag{5.22}$$

where $\phi_{avg}(x)$ is the average fitness of x.

Proof. The monotonicity criterion in our context is

$$I \leq I' \;\Rightarrow\; \phi(\alpha_I) \leq \phi(\alpha_{I'}) \tag{5.23}$$

Using equation 4.57 we obtain the desired result directly. □

Rather interestingly, this implies that (using theorem 12)

$$[alive(\alpha_I)]_{\alpha_I} \;\leq\; [alive(\alpha_{I\bigcirc})]_{\alpha_{I\bigcirc}}. \tag{5.24}$$

This is intuitively plausible as well since, at convergence (i.e. when either all individuals stay the same, or their fitnesses stay the same), the average fitness ceases to increase. Again, note that we are disregarding effects due to mutation.

The problem is, though, is this *necessarily* ($\Box$) true? Or, in temporal terms, henceforth (always in the future) true? Since we cannot use the rule of inference **nec** from section A.3.5 we know that we cannot infer $\Box A$ from A so the conclusion is that we cannot using logic infer that it will always be true once it becomes true. However, disregarding mutation, this should still be the case right from the beginning, *provided* equation 4.57 holds, with its rather stringent prerequisites, and the selection mechanism is proportional.

6 AXIOMATISATION OF THE GENETIC ALGORITHM

This chapter will formulate an axiomatisation of the genetic algorithm using the formalism described in the previous chapters. We will show that, using TPL, it is a relatively simple matter to axiomatise the genetic algorithm in several ways depending on the chosen underlying foundational structures. We will also show that it provides us with a good tool for better understanding the behaviour of the genetic algorithm. For instance, we will obtain a closed system, showing only acceptable executions, and it will be complete.

This chapter proceeds as follows. We will first describe a standard axiomatisation of groups in order to better understand axiomatisation in general, before going on to formalise an axiomatisation based on chromosomes for the genetic algorithm. We will also prove several theorems, some new and some previously proven in other ways, in order to show the power of the axiomatisation.

6.1 ON AXIOMATISATION

Rather interestingly, it is possible to formulate axiomatisations of the genetic algorithm patterned on standard usage as exemplified by e.g. the axiomatisation of groups (see any suitable mathematics text, e.g. [Sto79]).

In order to proceed with the axiomatisation we need a number of primitive notions. For groups, for instance, following [Sto79] these are an unspecified set G, a binary operation on G, for which we use multiplicative notation, i.e., the operation will be symbolised by $\cdot$ and the value at $\langle a, b\rangle$ of this function on $G \times G$ will be designated by $a \cdot b$, and an element e of G. The axioms for groups are the following.

$$\forall a, b, c \in G \mid a \cdot (b \cdot c) = (a \cdot b) \cdot c \tag{6.1}$$

$$\forall a \in G \mid a \cdot e = e \cdot a = a \tag{6.2}$$

$$\forall a \in G\ \exists b \in G \mid a \cdot b = b \cdot a = e \tag{6.3}$$

The above is a standard formulation of group theory. We call $a \cdot b$ the product of a and b, the element e that has the above property in 6.2 an identity element, and finally an element b that satisfies 6.3 the inverse of a (relative to e). We may then proceed to prove interesting theorems using the axioms, e.g. that there exists exactly one identity element, and that every element has exactly one inverse. The question is, can something similar be done for the theory of genetic algorithms?

The answer is yes; we may formulate several distinct axiomatisations of the genetic algorithm depending on what we see as the primitive notions. The one that we show here is based on the population as a set of chromosomes, with an operation to generate new individuals for the next population (i.e., members of the set forming the next population).

Of course, the axiomatisation also assumes the existence of TPL, with its basic constructs and objects, e.g. a temporal setting with intervals commonly designated I. We treat this axiomatisation below.

6.2 CHROMOSOME-BASED AXIOMATISATION

The axiomatisation is based on chromosomes, i.e individuals in the population. The basic set is the population Π, consisting of individuals $a, b, ...$, or chromosomes ξ_i and intervals; we assume all chromosomes are distinct, i.e. Π is a set, not a multiset. There are two distinct operations that may be performed: firstly, one that generates new individuals, and secondly, one that generates the fitness value for an individual. The former operation is ω corresponding to a composition of the three genetic operators treated previously. Recall that the question of composability of the various genetic operators was dealt with starting on page 44, in connection with the full treatment of the genetic operators, ω_γ, ω_ξ, and ω_Π. In other words, we may substitute the three operators as follows in order to 'elevate' them to operate on populations, whilst in reality maintaining the proper (gene, chromosome, and population) perspective.

$$\omega_\gamma \backslash_{\omega_\Pi} =_{sub} \omega_\gamma \tag{6.4}$$
$$\omega_\xi \backslash_{\omega_\Pi} =_{sub} \omega_\xi \tag{6.5}$$
$$\omega_\Pi \backslash_{\omega_\Pi} =_{sub} \omega_\Pi \tag{6.6}$$

The last one is of course an identity (as a function seen from its own level is itself). Then, the general composition of these operators becomes

$$\omega = \omega_\gamma \circ \omega_\xi \circ \omega_\Pi \tag{6.7}$$

defining the genetic operation ω, in this case one that operates on populations.

A moment's reflection allows us to notice that this may be accomplished on all levels, i.e. ξ and γ as well as the Π level as described above. Of course, the actual operations, domains and ranges are totally different on the levels. Since we will be using the ξ, or chromosome, level, we show this below.

The operation which generates new individuals, i.e. forming a new individual not in the current generation, but *in the next generation* (see page 43) would be $\omega\backslash_{\omega_\xi}$. That operation may also be defined as follows (cf. definition 36).

$$\omega : \prod_n \Pi_I \to \Pi_{I\bigcirc}, \tag{6.8}$$

where n is the number of parents that generate the offspring, and $\prod$ is defined in the usual manner as $\times \ldots \times$. In order to simplify matters we assume that the generation of new individuals entails only two parents, and that we thus can use the same kind of multiplicative notation as above. The operation then simplifies to

$$\omega : \Pi_I \times \Pi_I \rightarrow \Pi_{I\bigcirc}, \tag{6.9}$$

which means that we can use the following equivalent notation

$$a = \omega(b, c) \Leftrightarrow a = b \cdot c, \tag{6.10}$$

where $a \in \Pi_{I\bigcirc}$ and $b, c \in \Pi_I$ as well as $\cdot$ is used instead of $\times$. Of course we may extend this to multiple parents by noting that

$$a = \omega(b, c, d \dots) \Leftrightarrow a = b \cdot c \cdot d \dots, \tag{6.11}$$

which clearly shows how this case can be handled. Note that we do not imply that any of the equations for groups, e.g. 6.1 applies, even if this looks superficially the same.

It is important to realise that the three operations above ($\omega_\gamma, \omega_\xi$, and ω_Π) in a very important sense operate on different levels:

1. ω_γ works within the gene (normally same as chromosome), with the bits forming it; i.e. intra-gene,
2. ω_ξ works with the genes (chromosomes); i.e. inter-gene, and
3. ω_Π works with the populations; i.e. intra-population.

The actual mechanism on the levels are, in genetic algorithms theory, the following:

1. for genes: mutation,
2. for chromosomes: recombination (or cross-over), and
3. for populations: selection.

The last one of these is especially important since its 'tool', as it were, is the fitness; the second necessary operation alluded to above is one that generates the fitness value for an individual as follows; cf. definition 29.

$$\phi : \Pi_I \rightarrow [0, 1] \tag{6.12}$$

The corresponding definition of the average population fitness is

$$\phi(\Pi) = \frac{\sum_{\alpha \in \Pi} \phi(\xi)}{|\Pi|} \tag{6.13}$$

or, with the domain emphasised

$$\phi_\Pi : 2^{\mathcal{S}_\alpha} \rightarrow [0, 1] \tag{6.14}$$

or simply the arithmetic mean of the fitnesses of the individuals in the population. We often write $\phi_{avg}(\Pi)$ when we want to show this.

As has been pointed out in the section on selection (see section 4.4 starting on page 54) selection is performed in such a way that individuals are selected for in accordance with their fitness, so that the more fitter survive (statistically speaking) and the less fitter do not. The key point here is that, *on the average*, new individuals are fitter than old ones. Repeating equation 4.55, say that

$$a \in \Pi_{I\bigcirc} \quad b \in \Pi_I \mid P(\phi(a) \geq \phi(b)) > 0.5 \tag{6.15}$$

or, paraphrasing, on the average $\phi(a) \geq \phi(b)$.

The first axiom deals with the generation of new individuals in the genetic algorithm and is the following.

Axiom 2 (GA1 – Axiom of reproduction) *New individuals in a population are produced according to the formula*

$$\Box(\forall a \in \Pi_{I\bigcirc} - \Pi_I \quad \exists b, c \in \Pi_I \mid a = b \cdot c) \tag{6.16}$$

Axiom GA1 captures the fact that for any subsequent (next) generation (i.e. population, or, looking at the situation from a temporal point of view, interval), the individuals are generated from the current generation (population, or interval). We have restricted the axiom to explicitly apply to new individuals only, hence the quantification over $\Pi_{I\bigcirc} - \Pi_I$. Note too that the axiom does cater for the case where individuals survive from generation to generation by the fact that the 'reproductive' function ω_ξ allows for results equal to one of the original values (of course allowing for different intervals). It is important to note that the intended quantification over the individuals is $\forall a \in \Pi_{I\bigcirc}\ \exists b, c \in \Pi_I \ldots$ instead of $\forall b, c\ \exists a \ldots$. The reasons are that the latter is trivial because $\cdot$ is a function and that the former guarantees that new individuals do not appear from nothing. We just have to make sure that new individuals indeed may do so; see axiom GA2 below.

Note that we say that this is valid in *all* possible worlds (in the TPL sense); hence the $\Box$.

Examining the axiom, we note that axiom GA1 may instead be written in a slightly different form, together with a slightly modified assumption for a as follows.

Axiom 3 (GA1′ – Axiom of reproduction) *New individuals in a population are produced according to the formula*

$$\Box(\forall b, c \in \Pi_I \quad \exists a \in \Pi_I \cup \Pi_{I,v} \quad \mid \quad a = b \cdot c) \tag{6.17}$$

In GA1′ we use the property that any individual in the next population belongs either to the current generation's real population or its virtual population; please refer to the discussion on virtual populations on page 38 to see that this must be true. However, this axiom is not really that useful, since it is actually in one sense weaker than GA1: we do not know in which interval the descendant a will become 'alive' since the virtual population $\Pi_{I,v}$ encompasses all generations. Indeed, we are just saying that a reproduction phase in the genetic algorithm generates individuals which may or may not (depending on the fitness, i.e. probability of occurrence) come into existence at some unspecified time in the future. However, GA1′ is stronger in the sense that it does say that all individuals do generate offspring.

One of the things that we have chosen not to take into account in the axioms above is the problem of the initial population. Where does it come from? Previously, as we have formalised the genetic algorithm we require a population Π_0 that is not generated by any genetic mechanism or other functional, but essentially created at random (although randomness as such is not a requirement; the only requirement is that we need not worry about how this generation came about). If we wish to be absolutely complete and take care of this situation we could formulate an axiom for this particular purpose.

The axiom of the inital population could be written as follows, according to definition 40.

Axiom 4 (GA2 - Axiom of the initial population) *The initial population is taken as the first population that changes (or evolves). Symbolically,*

$$\exists I_0, \forall I < I_0 \quad | \quad \Pi_I = \Pi_{I_0} \tag{6.18}$$

This makes it clear that the population may be seen as unchanging up to a certain interval I_0. In this way we avoid the problem of the initial population altogether.

Note that we might be tempted to write it in the following way instead, capitalising on the fact that we usually do have a distinguished starting population Π_0.

Axiom 5 (GA2′ – Axiom of the initial population) *The initial population* Π_0 *does not have a precursor population. In other words,* $\exists I_0$ *such that* $\quad \nexists \Pi_{I_0 \overline{\bigcirc}}$.

The formulation of this axiom is problematic – how can we formalise the non-existence of a population at a certain time? This axiom GA2′ shows that the algorithm starts from an interval where the individuals have been chosen in some other way but the mechanism in axiom GA1; i.e. we are referring to the very first interval I_0, or the initial population Π_{I_0} (or plain Π_0).

However, examining GA2 and GA2′ reveals that they contradict each other; the latter explicitly disallows any populations before the initial one by making it clear that an individual a cannot be produced by two others using $b \cdot c$; whereas the former stipulates that populations do exist before the initial one, but that they are identical to it. Of course, in the case of GA2′ we would have to somehow address how the initial population is created by, say, suitably modifying GA1.

We could also explicitly specify an empty starting population using the following variant.

Axiom 6 (GA2″ – Axiom of the initial population) *The population* Π_I *before the initial population* Π_0 *is empty. In other words,*

$$\exists I_0, \forall I < I_0 \quad | \quad \Pi_I = \emptyset \tag{6.19}$$

This means that we must somehow ensure that Π_0 can be generated from the empty population, which perhaps may be accomplished by allowing the arguments b and c in the product $b \cdot c$ in **GA1** to have null values. In other words, the ω function underlying the product should, for instance, generate a random individual a in this case. Note that this variant is not compatible with GA1 as it now stands; GA1 would need rewriting if we chose GA2″.

With suitable modifications to GA1 to make it commensurate with GA2′ or GA2 ″, it would be a matter of choice which one of the GA2's to use; neither has any bearing for the continued working of the genetic algorithm except for taking care of the problem of the existence of the first generation. Below, we have not used this axiom at all in which case we just have to note that the genetic algorithm starts at some time not further specified.

Let us examine some implications of the axioms. We do this by first proving some theorems, both ones previously shown and new ones, using the axioms as well as TPL as described above.

We start by restating and proving theorem 13 using the axioms.

Theorem 14 *The average fitness increases monotonically. Formally,*

$$\phi_{avg}(\Pi_I) \;\leq\; \phi_{avg}(\Pi_{I\bigcirc}) \tag{6.20}$$

where $\phi_{avg}(x)$ is the average fitness of x.

Proof. Starting from GA1 we note that $\forall a \in \Pi_{I\bigcirc} \quad \exists b, c \in \Pi_I$

$$\Box(a = b \cdot c) \Leftrightarrow \tag{6.21}$$

$$\Box(a = \omega(b, c)) \tag{6.22}$$

using 6.10. Since, according to how our genetic operators compose, cf. discussion starting on page 44, we know that, in this case

$$\omega = \omega_\xi \backslash_{\omega_\xi} \tag{6.23}$$

and consequently that

$$\omega_\xi \backslash_{\omega_\xi}(b, c) = \omega_\Pi \backslash_{\omega_\xi}(b', c'). \tag{6.24}$$

Of course, we cannot simply substitute one function for the other, but we do know that with b' and c' instead of plain b and c , the equivalence certainly exists. It is important to, so to speak 'elevate' our notation to the population Π level, because then we introduce the selection mechanism.

Substituting this in the previous, and rearranging using basic functional composition notation we obtain

$$\Box(a = \omega(b, c)) \Leftrightarrow \tag{6.25}$$

$$\Box(a = \omega_\xi \backslash_{\omega_\xi}(b, c))) \Leftrightarrow \tag{6.26}$$

$$\Box(a = \omega_\Pi \backslash_{\omega_\xi}(b', c')). \tag{6.27}$$

What we have achieved now is to show that although GA1 operates on the chromosome level, it can always be seen as operating on the population level as well, with suitable modifications of the arguments. Furthermore, we can continue and draw the following conclusion

$$\Box(a = \omega_\Pi \backslash_{\omega_\xi}(b', c')) \Rightarrow \Box(\Pi_{I^\bigcirc} = \omega_\Pi(\Pi_I)). \tag{6.28}$$

The implication indicates that what is valid on the chromosome level is also compatible with the population level since the functions may be composed to be compatible, as indicated above.

According to the characteristics of the operator ω_Π, we have that, on the average,

$$\begin{aligned} \phi(b) &\leq \phi(a) \\ \phi(c) &\leq \phi(a). \end{aligned} \tag{6.29}$$

This is of course due to the mechanism underlying that of the function ω_Π, i.e. selection, that ensures that we select individuals for survival according to their fitness.

If this is the case for any individuals in the populations Π_I and $\Pi_{I^\bigcirc}$ then we can certainly say that

$$\phi_{avg}(\Pi_I) \leq \phi_{avg}(\Pi_{I^\bigcirc}). \tag{6.30}$$

Comparing this to Theorem 13, we notice that they are identical, allowing us to conclude that the theorem holds. $\Box$

Rather interestingly, we have the following corollary.

Theorem 15 *For $a \in \Pi_{I^\bigcirc}$ and $b \in \Pi_I$ the following formula holds*

$$\Box([alive(a)]_a \geq [alive(b)]_b). \tag{6.31}$$

where we use the *alive* predicate from Definition 33 on page 39; recall that $alive(x) : \phi(x) > 0$.

Proof. Using Theorem 12 we know that

$$\phi_{avg}(\Pi_I) \;=\; [alive(x)]_x. \tag{6.32}$$

Substituting this in equation 6.30 (same as 6.20 in Theorem 6.20) we obtain

$$\phi_{avg}(\Pi_I) \leq \phi_{avg}(\Pi_{I^\bigcirc}) \Rightarrow \tag{6.33}$$
$$[alive(b)]_b \leq [alive(a)]_a \tag{6.34}$$

where we have used the individuals a and b from the corresponding intervals $I^\bigcirc$ and I, respectively.

Noting that equation 6.28 provides the necessary $\Box$ operator, reversing the order of the terms, and changing $\leq$ to $\geq$ completes the proof. $\Box$

We can also derive another theorem, a new and interesting consequence of the axioms as the following will show.

Theorem 16 *All individuals of a population* Π *will eventually reside in a given equality interval* ϵ.

Proof. According to theorem 15, we have that for $a \in \Pi_{I\bigcirc}$ and $b \in \Pi_I$ $\Box([alive(a)]_a \geq [alive(b)]_b)$. This implies the following sequence

$$[alive(z_{I\bigcirc^n})]_{z_{I\bigcirc^n}} \geq \ldots \geq [alive(a_{I\bigcirc})]_{a_{I\bigcirc}} \geq [alive(b_I)]_{b_I} \geq \ldots, \quad (6.35)$$

where we have included the proper indices to show the interval the various individuals $a, b, \ldots z$ are resident in. n is a number indicating generations forward in time.

Now, according to theorem 12 we can substitute **prob** instead of the statistical probability in the preceding series. These are of course then greater than zero as well. Summarising, we have that

$$\mathbf{prob}(alive(z_{I\bigcirc^n})) \geq \ldots \geq \mathbf{prob}(alive(b_I)) \quad (6.36)$$

So the probability will either rise or stay the same for each generation. Clearly then there is a finite probability that we will come as close to 1 as we desire. Since at 1 all individuals must have a fitness of 1, the equality interval will be zero. Recall that our fitness is equated to the probability of occurrence, i.e. it will certainly be 1 if it is certain that the individual exists. This is contrary to the normal fitness value in a genetic algorithm, which is determined by some external function. The implication is that it must have been greater than zero before this, and in fact that all individuals must have been resident in any equality interval we care to name, from the maximum of 1 to the minimum of 0. Theorem 22 on page 144 gives the link between **prob** and $\Diamond$ proving the theorem. $\Box$

This is a remarkable result, showing that in the genetic algorithm the individuals will eventually all reside in the same equality environment; i.e. the population has converged. Note, though, that this result does hinge on the ω_Π operator actually selecting for better and better individuals, on the average, all the time.

We may rewrite this result as

$$\tilde{\alpha} = \{\beta \quad | \quad \alpha \cong \beta\} = \Pi \quad (6.37)$$

using the notation of definition 45. Of course, here we also assume (quite reasonably as well) that convergence is taken as the same as all individuals staying in the same equality interval, as indeed has been pointed out earlier. Note that inside the equality interval the normal mechanisms for reproduction of individuals is *superseded by* age-based considerations, and hence we may even say that the (at least the canonic) genetic algorithm is no longer operative as such inside the equality interval.

Of course, do note that the preceding, i.e. Theorem 16 only applies if we use time-conservative or time-progressive selection schemes, as described in section 5.1.

As a corollary theorem, we may also immediately show the following.

Theorem 17 *For the time-conservative case no individuals will be replaced at convergence. In other words,*

$$\Diamond(\Pi_v = \emptyset) \tag{6.38}$$

Proof. That this is the case is quickly seen from the following implication, given $\alpha, \beta \in \Pi_I$ and the result from theorem 16

$$\Diamond(\alpha \cong \beta) \quad \Rightarrow \tag{6.39}$$

$$\Diamond(\Pi_{I,v} = \emptyset) \tag{6.40}$$

This is immediately clear from the definition of the time-conservative case; when all individuals are in the equality interval no replacements will take place and the virtual population will consequently stay empty. □

But this is just theorem 9 - which we have thus derived from the axioms. This directly implies theorem 8 as follows.

$$\Diamond(\Pi_{I,v} = \emptyset) \quad \Leftrightarrow \tag{6.41}$$

$$\Diamond(|\Pi_{I,v}| = 0) \quad \Rightarrow \tag{6.42}$$

$$\Diamond(|\Pi_{I,v}| \leq 0) \tag{6.43}$$

Since it is always true that

$$0 \leq |\Pi_{I,v}| \leq |\Pi_I| \tag{6.44}$$

then we know that

$$\Diamond(\left|\Pi_{I\overline{\bigcirc},v}\right| \leq |\Pi_I|/2) \tag{6.45}$$

because it was true either immediately, or became true during some interval I or during the virtual population's continuous shrinking toward 0 (and it will become 0 as shown above). Thus theorem 8 may also be derived from the axioms.

6.3 DISCUSSION

We have shown how genetic algorithms may be axiomatised using fairly simple techniques. In order to gain a fuller understanding of the process, we have also described various options how we can vary the axioms themselves.

However, it must be pointed out that a foundational approach like the one outlined in this chapter can only go so far; the genetic algorithm, being a method for solving problems in real life, needs to be connected to and augmented with problem-specific data (specifically, the fitness function needs to show the fitness in terms of the requirements at hand) largely making the theoretical aspects of lesser importance.

But, nevertheless, the above description of the axiomatisation of the workings of genetic algorithms using a general approach represents a promising avenue for further research.

7 MODELLING GENETIC ALGORITHMS USING HIGH-LEVEL PETRI NETS

This chapter takes advantage of the formalism of high-level Petri nets, which is customarily used to model parallel and distributed systems, to model genetic algorithms. With the help of this formalism the author gives a simulation of the genetic algorithm using Petri nets that is both simple and expressive. We call the new model the Petri net based Genetic Algorithm model, or PGA model for short.

The chapter proceeds by first introducing Petri nets, and showing an informal genetic algorithm formalisation using one; then, a formal presentation follows. Lastly, we will show some comparisons of the standard, or canonic, formulation of genetic algorithms with the Petri net formalism and show some similarities and differences, and discuss the importance of the Petri net model.

The approach to simulating the genetic algorithm in the way described in this chapter has not been encountered elsewhere and represents an original contribution to the field.

7.1 PETRI NETS

The concept of Petri nets has its origin in Carl Adam Petri's dissertation from 1962 [Pet62].

The basic idea behind the concept of a Petri net (the place/transition net [Rei82]) is fairly simple. It is a graphical and mathematical modelling tool consisting of *places*, *transitions*, and *arcs* that connect them. Input arcs connect places with transitions, while output arcs start at a transition and end at a place. In order to characterise the state of the net, as seen from an outside point of view, we imagine ourselves putting *tokens* at each place; the current token distribution of the net is called the *marking* and is given by the number (and identity if the tokens are distinguishable) of tokens in each place. Note that the tokens themselves are an aid in understanding the working and dynamic behaviour of the net; they do not correspond to any physical entities, although this distinction is often ambiguous. Transitions, in their turn, are the active components. They model activities which can occur (i.e. the transition, being enabled can *fire*), thus changing the state of the system (the marking of the Petri net). Transitions are allowed to fire if they are *enabled*, which means that all the preconditions for the activity must be fulfilled (there are enough proper tokens available in the input places – and we are discussing the simple place / transition net; more complex behaviour is exhibited by other classes of Petri nets; e.g. predicate / transition nets below). When the transition fires, it removes tokens from its input places and adds some at all of its output places. In the general case, the number and type of tokens removed / added depends on the cardinality (or arc expression) of each arc. The sequential firing of transitions in each subsequent marking is sometimes called the *token game*.

Petri nets are a promising tool for describing and studying systems that are characterised as being concurrent, asynchronous, distributed, or parallel. As a graphical tool, Petri nets can be used as a visual-communication aid similar to flow charts, block diagrams, and networks. In addition, with the aid these models we find it easier to simulate the dynamic and concurrent activities of systems. As a mathematical analysis tool, it is possible to set up state equations, algebraic equations, and other mathematical constructs governing the behaviour of systems and thus better understand the workings of the net.

For a more detailed description of basic Petri nets and their theory, see e.g. Reisig [Rei82], or Rozenberg [Roz90, RR98a, RR98b].

Note that the original Petri net does not provide a time component. This means that, while correctness properties can be verified, performance analysis is not possible. As a result, Petri nets have been augmented with time. The most widely used of such augmented nets are Time Petri Nets, Timed Petri Nets, and Generalised Stochastic Petri Nets (GSPNs) [AMCB84].

The most important high-level Petri nets are of two basic kinds: Predicate / Transition Nets (Pr-T nets) and Coloured Petri nets (CP nets). They add individuals with changing properties and relations to the previous ('lower-level') Petri nets; the individuals are tokens with structure ('colour') and, for instance enhance the net with arcs furnished with expressions.

For more details on high-level nets, see e.g. [JR91], or [RR98a, RR98b].

7.2 BASIC PETRI NET DEFINITIONS

We will be using these Pr-T nets below in our formalisation of the genetic algorithm as a Petri net. But before going on we will formally define Petri nets and some of their basic characteristics. We will follow [Gen87, Gen90] and [Pys96] (based in part on [Rei82]).

A Petri net is a directed graph with two types of nodes, called places and transitions, respectively. Nodes of one type are connected to nodes of the other type with weighted arcs. Formally, we define the net as a tuple consisting of a set of places, a set of transitions, and a weight function between places and transitions, and transitions and places.

Definition 56 (Petri net) *A* Petri net *is a tuple $N = (P, T; F)$, where*

1. P *is a finite set of* places,
2. T *is a finite set of* transitions $(P \cap T = \emptyset)$, *and*
3. $F : (P \times T) \cup (T \times P) \mapsto \mathbb{N}$ *is a* weight function.

A *marking* of a net is a function that assigns to each place a natural number $M : P \mapsto \mathbb{N}$. At a marking M a place p has $M(p)$ tokens. An *initial marking* M_0 assigns to each place an initial token count. A place p is an *input place* (or *output place*) of a transition t, if $F(p, t) \geq 1$ (or $F(t, p) \geq 1$).

Definition 57 *A transition $t \in T$ is* enabled *at a marking M if for all its input places $M(p) \geq F(p, t)$.*

An enabled transition t at a marking M can fire and yield another marking M' such that $\forall p \in P : M'(p) = M(p) - F(p, t) + F(t, p)$. We denote this by $M \rightarrow_t M'$.

Finally, a marking M' is reachable from a marking M if there is a finite transition sequence (or *firing sequence*) α such that $M \rightarrow_{t_1} M_1 \rightarrow_{t_2} M_2 \ldots \rightarrow_{t_{|\alpha|}} M'$.

Note that we will define Pr-T nets below; see definition 60.

7.3 FORMAL MODEL USING PETRI NETS

From the preceding section it may be possible to understand how we can model genetic algorithms with Petri nets. Since Petri nets are excellent tools for describing parallel and concurrent systems it seems intuitively clear that it should be possible to model genetic algorithms, which exhibit just these traits, using these nets.

A formal model of the genetic algorithm using Pr-T nets (the PGA model alluded to earlier) is formulated in this section. Here we will rely on [JR91] and especially on Genrich in [Gen87]. We first describe the simple net we will use (see figure 7.1), and then its components: the places $P1$, $P2$, $P3$, and $P4$, and their contents and purpose. Then, we will describe the arc expressions, followed by the semantics of the net, and the symbolic transition rules – the whole of which we may interpret as the genetic algorithm, suitably expressed for this purpose.

Why do we use Pr-T nets and not other nets? The reason is simple: the basic net (Petri net above) is not rich enough. The Pr-T net's major difference is its ability to store more than one token at each place, in contrast to the basic net's customary one. This is exactly what we need for the genetic algorithm. Furthermore, the tokens must be indistinguishable, i.e. treated absolutely equally from the Petri net's point of view. In fact, for instance when a token 'moves' from one place to another over a transition, the choice, as it were, of which token is 'moved' is *non-deterministic* (it cannot be otherwise since they cannot be distinguished from each other): precisely what we require in the genetic algorithm when, e.g. 'choosing' parents, or selecting among equally unfit (or fit) individuals. We do not need more elaborate token structures (from a Petri angle) than the ones in the Pr-T net. Of course, we could use seemingly more complicated Petri nets with e.g. coloured tokens, or timed nets; however, for our purposes, the Pr-T net suffices.

7.3.1 The Pr-T net

The objective is to model the genetic algorithm using a Petri net formalism. We do this by constructing a net with two transitions and four places, as depicted in figure 7.1. This is an example of a *predicate/transition* net (Pr-T net) [Gen87].

In this high-level net we use structured tokens α, β, γ, etc. each modelling one individual in the genetic algorithm. According to definition 28, we would have that e.g. $\alpha = \langle \xi_\alpha, I_\alpha \rangle$ where ξ_α is the chromosome for α and I_α the interval, within which its fitness exceeds zero (c.f. definition 29). In this way, we connect the fitness ϕ with the chromosome, and hence the net.

Again, we should be a little careful here: of course, since tokens really are in a sense imaginary (and the transitions do the work) we should construe the above as determining the allowable changes that may occur transitioning from one state to another (i.e. marking) in the form of the token 'movements'.

Place $P1$ contains the set of population tokens Π, $P2$ and $P3$ are used to step the algorithm from phase to phase, and $P4$ contains the set of all possible tokens, denoting the set of all possible individuals $\mathcal{S}_\alpha$ (cf. definition 28 and subsequent text). Transition $T1$ constitutes the recombination phase generating new individuals whereas $T2$ is the selection phase, which selects according to the fitness criteria which individuals are to survive to the next generation.

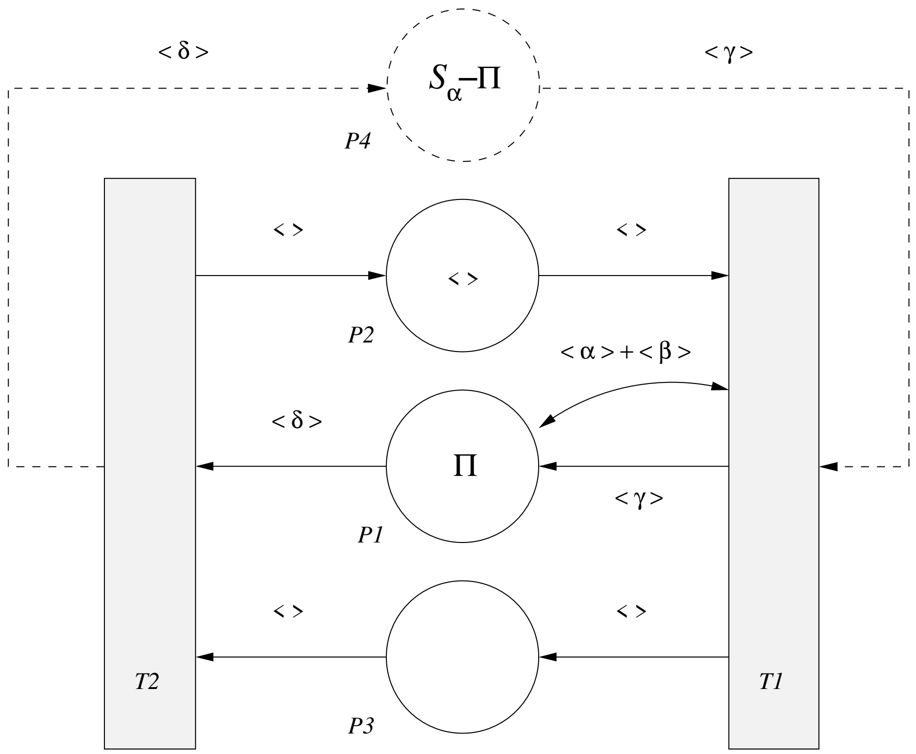

Figure 7.1: Genetic algorithm as a Pr-T net

Note how similar this is to fig. 4.10, which depicts the various populations in the genetic algorithm. The basic flow of the algorithm is as follows (cf. fig.

7.1). We interpret the net in terms of the genetic algorithm, and describe the working of the underlying Petri net as if it were a genetic algorithm.

1. two parents α and β are selected at random from $P1$
2. they are suitably recombined and mutated at $T1$ generating a descendant individual γ, which is taken from $P4$
3. the descendant γ and its original parents α and β are put back into the population at $P1$
4. the 'worst' individual δ is removed from the population at $T2$ and put back into $P4$, and the rest are left at $P1$
5. the cycle begins anew

Note that this exposition assumes that two individuals will generate a third, and that these are put back into the population, and, furthermore, that one is always dropped from the population according to its (inferior) fitness.

Note that, in the description above we imply a sequential sequence; however, this is by no means given. Indeed, we can easily conceive of a system with $T1$ firing continuously, in parallel, generating offspring from all pairs (in the case of fig. 7.1) simultaneously. We can even postulate that $T2$ does the same, removing unfit individuals continuously as well keeping the population size $|\Pi|$ stable.

It is clear that we in this simple way have simulated the workings of a simple genetic algorithm using the Pr-T net in fig. 7.1. Of course, we have to further describe the exact operations of the various components of the net.

7.3.2 The places $P1$, $P2$, $P3$, and $P4$

The places, in a manner of speaking, serve the purpose of storing the tokens in the Petri net. There are two kinds of tokens in this model, as described in the following list of the places.

- always in place $P1$, the high-level tokens denoting the population set $\Pi = \{\alpha, \beta, \gamma, \ldots\}$ consisting of one token per individual
- initially in place $P2$, the synchronisation token, denoted $\langle\,\rangle$ in fig. 7.1,is used to step the net from marking to marking
- the place $P3$, initially empty in fig. 7.1 is used to move the synchronisation token $\langle\,\rangle$, described above back to its initial location in $P1$, thus stepping the net from marking to marking.
- the place $P4$, for possible tokens not in Π at the current interval; see below.

The only crucial place is thus $P1$ and to a lesser extent $P4$; $P2$ and $P3$ are simply used to step the net from marking to marking.

We use the fourth place $P4$ to store all the remaining possible tokens (i.e. $\mathcal{S}_\alpha - \Pi$). This place is used to 'close' the token universe providing a comprehensive ontology for the simulation model acting as a 'source' and 'sink' of newly generated tokens (denoting newborn individuals) and resting place of removed tokens (denoting unfit individuals selected against in the genetic algorithm), respectively. However, it is not really needed for the semantics of the net to work as desired, and as such, it is drawn with dotted lines in figure 7.1. In a realistic model (like the one we analyse in section 7.4 below) we do not include this fourth place. Instead we let the transition $T1$ generate suitable new individuals.

Thus, during the simulation there are normally tokens in $P1$ and in one of $P2$ or $P3$ (but not both). The tokens in $P1$, the one holding the population, are structured tokens, as was described in section 7.3.1 above. The internal structure of the individuals (tokens) play no further role in the exposition of the Petri net model. The token 'universe', as it were, is in $P4$, as explained above.

7.3.3 The arc expressions

The arc expressions basically tell the model what should be transformed (loosely speaking 'moved') from place to transition to place, taking it from a simplistic point of view. In our case we have several expressions, as listed below.

- the empty expressions $\langle\,\rangle$ indicate that no individuals of the genetic algorithm population Π) are being transformed ('moved') in it. It is a high-level token of its own, though, used for synchronisation as described in the section above.
- the expression $\langle\alpha\rangle + \langle\beta\rangle$ is a pair of tokens denoting two individuals from the population Π that have been chosen as parents and whose corresponding tokens are being traced to transition $T1$ to undergo the manipulations necessary to produce new tokens (which will denote offspring individuals; see below). Selection of the tokens is accomplished by transition $T1$ (see below).
- the expression $\langle\gamma\rangle$ is the token (denoting a new individual) being transformed ('moved') back into the token population Π having been generated from α and β. Note that it is taken from $P4$, the size of which is thus decreased (the expression is $\mathcal{S}_\alpha - \Pi$ which is dependent on transition $T1$ removing and $T2$ adding tokens denoting newly generated and removed as unfit individuals, respectively).
- the expression $\langle\delta\rangle$ indicates that the superfluous token (i.e. denoting the individual not selected for; rejected by the genetic algorithm as being unfit according to the problem-specific fitness function) is being handled.

In this way, we have labelled all arcs with an expression indicating what is being transformed ('moved') from place to place when the transitions fires.

Rather interestingly, since the genetic algorithm simulation model created in this way is closed, we also see a natural explanation for the virtual population (1 generation ahead, not general) alluded to earlier (cf. theorem 2 on page 40ff): it is the γ's – recall that we, in order to simplify the discussion assumed only one descendent, not a set – forming the addition to the current population at each generation. In other words, we have accounted for whole ontology of the genetic algorithm by denoting the tokens in this way.

Thinking in Petri net theoretical terms, if we were to construct the reachability graph for the net, we would see the virtual populations emerge very naturally in the graph. Indeed, if we were to trace the graph forwards, we would see the future virtual populations (as well as the real ones), and correspondingly, looking backwards we would see where we have been at each 'generation'. This, of course, provides a very strong indication of the usability of Petri net analysis of genetic algorithms, as described in this work.

7.3.4 The transitions $T1$ and $T2$

The transitions in the PGA model are the components which correspond to action occurrencies. In this case we have two transitions with the following roles:

- transition $T1$ is used for producing new tokens, i.e. denoting individuals in the underlying genetic algorithm. Somewhat loosely speaking from a genetic algorithm (i.e. interpretation) point of view, it will take a number (typically two, as indicated in fig. 7.1 the individuals α and β) of individuals, use them as parents for generating a new individual γ (taken from the 'store' at $P4$), which is subsequently put back together with its parents into the population Π. Note that the generation of new individuals will be in essence non-deterministic since it uses the ω_ξ function, which may use random mutation on the gene γ level, or use random elements in its recombination function at the chromosome ξ level

- transition $T2$ is used for pruning the population according to the given fitness function. It works by accepting the whole population Π (including the offspring and its/their parents from transition $T1$), and removing an equal number of individuals that were produced earlier, thus keeping the population size unchanged. It will then return the remaining individuals to place $P1$. In effect, we will remove the 'unfittest' individual (individuals) from the population Π. The effect is as if (in the example case) one individual δ were removed as indicated by the label on the arc leading from place $P1$. Note that the selection of tokens (individuals) is determined by the type of genetic algorithm; in general, it is some form of randomised selection, e.g. proportional to the fitness; cf. table 4.4.

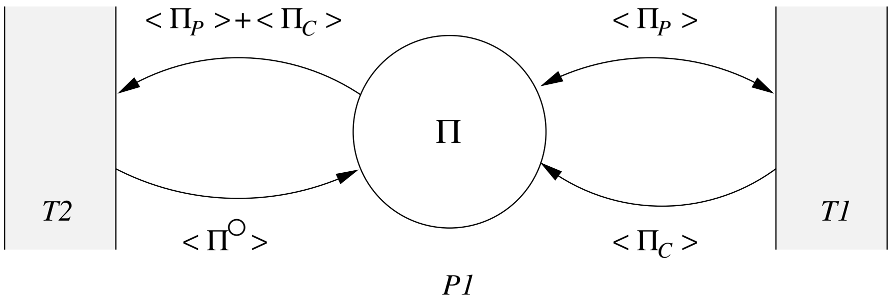

Figure 7.2: General arc expressions in the PGA model

The functions used at the transitions are the already familiar ones governing the dynamic behaviour of the genetic algorithm.

- In $T1$, the transition used for generating new individuals in the genetic algorithm, we use ω_ξ (cf. definition 36). Note that this is a high-level definition; ω_ξ may be any conceivable transformation of the chromosomes combined with the generation of one of several new individuals, cf. the definitions 6.8-6.11 in the discussion on axiomatisation of the genetic algorithm to see how this is done. Effectively, we are saying that $\gamma = \omega_\xi(\alpha, \beta)$.
- In $T2$, the transition used for discarding unfit individuals in the genetic algorithm, we use ω_Π (cf. definition 37). Note that this is a high-level definition; ω_Π may be any conceivable transformation of the population Π to another population $\Pi^{\bigcirc}$ (i.e. a population from the next generation.) Effectively, we are saying that $\exists\delta_{min}\forall\delta(\delta_{min} \leq \delta)$.

Do note that the application of the transitions and functions are deterministic, not random, even if the functions do contain random, non-deterministic elements. E.g. the ω_Π function handles the selection of individuals, which can have random aspects – but *some* individuals are always selected.

Furthermore, it is important to realise that the general PGA model is not restricted to the two parents and single offspring (and single discarded individual) of figure 7.1 but that in fact is wholly free to use any number that makes sense from a genetic algorithm point of view. In order to show the general form of the arcs to / from place $P1$ in figure 7.2 in which we omit places $P2$, $P3$, and $P4$ for clarity.

Note that, compared to table 4.3 and figure 4.10 we have used the notation $\Pi^{\bigcirc}$ instead of $\Pi_{I\bigcirc}$ since we do not want to emphasise the interval in this context. Also note that the above depiction is only true if $|\Pi| = constant$ at all times; this is always the case in our model.

7.3.5 Net semantics

In this section we will develop the formal language and its semantics for the Pr-T nets shown above. For this purpose we have to understand how a Pr-T net is annotated, and what the annotation signifies.

Following [Gen87], our approach here is to model the genetic algorithm in terms of a set of tokens denoting individuals (forming the population) that is structured by functions and relations. The structure is partially static and partially dynamic; in the genetic algorithm the order of the creation of the various subpopulations forming the sequence that defines the next population (possibly a whole generation, but maybe just one individual) is the static part, and the relation *isOccupiedBy* between the populations (and subpopulations) and individuals forming them forms the dynamic part.

It is important to realise, that the Pr-T net always embodies the use of a language (for annotating the net) that is used to model a system, in this case a genetic algorithm. That language is an extension of standard first-order logic, with extensions for handling the formalism of the net itself with its places, transitions and arcs. We rely on Genrich [Gen87, Gen90] and also on Pyssysalo [Pys96, p. 34ff].

In other words, a Pr-T is a Petri net, the places, transitions and arcs of which are annotated with a set of variable predicates, well-formed formulas, and symbolic sums of tuples of a first-order structure for a first-order language $\mathcal{L}$.

We will first formally define the structure and the language used in the annotation of the Pr-T net. The following definition of the structure is based on [Gen87, Gen90].

Definition 58 (Structure) *A* structure *is a tuple of objects, $\mathcal{R}$ defined as*

$$\mathcal{R} = \langle D; f_1, f_2, \ldots, f_k; R_1, R_2, \ldots, R_n \rangle \tag{7.1}$$

where D is a non-empty set of individuals *called the* domain *or* carrier *of $\mathcal{R}$, f_i are* functions *in D and R_j are* relations *in D.*

In our genetic algorithm setting, we have that the domain D is the set of all possible individuals $\mathcal{S}_\alpha$ after the definition of an individual in the genetic algorithm (definition 28). The functions f_i form the set of genetic operators Ω (see page 44 and the formulation of the genetic algorithm, equation 4.44 on page 50). The relations R_j form the set of relationships between places and the populations; in this case there is only one relation, the *isOccupiedBy* relation with obvious meaning.

We are now ready for the definition of the vocabulary of the language in the net; again, we follow [Gen87, Gen90].

Definition 59 (Vocabulary) *For each $n \geq 0$, a finite set of n-ary* operators *(function symbols) and a finite set of n-ary* predicates *(relation symbols) of the structure $\mathcal{R}$ form the* vocabulary *of the first-order language $\mathcal{L}$. $\mathcal{L}$ consists of three kinds of expressions: the set of individual terms $\mathcal{L}_t$, the set of first-order formulas $\mathcal{L}_f$, and the set of symbolic sums $LC^{(n)}$. In addition, there is an infinite set of variable symbols V in $\mathcal{R}$, disjoint from operators and predicates. The three kinds of expressions are formed in the following way.*

The terms $\mathcal{L}_t$, which are formed as follows.

1. *A variable is in $\mathcal{L}_t$.*

2. *If $f^{(n)}$ is an n-ary operator and $v_1, v_2, \ldots, v_n$ are terms, then $f(v_1, v_2, \ldots, v_n)$ is a term.*

3. *No other expression is in $\mathcal{L}_t$.*

The formulas $\mathcal{L}_f$, which are formed as follows.

1. *The constants $\top$ (true) and $\bot$ (false) are atomic formulas in $\mathcal{L}_f$.*

2. *If v_1 and v_2 are terms, then $v_1 = v_2$ is an atomic formula in $\mathcal{L}_f$.*

3. *If $P^{(n)}$ is an n-ary predicate and $v_1, v_2, \ldots, v_n$ are terms, then $Pv_1, v_2, \ldots, v_n$ is an atomic formula in $\mathcal{L}_f$.*

4. *If p_1 and p_2 are in $\mathcal{L}_f$, then $\neg p_1$ and $(p_1 \vee p_2)$ are in $\mathcal{L}_f$.*

5. *If x is a variable and p is in $\mathcal{L}_f$, then $(\exists x)p$ is in $\mathcal{L}_f$.*

6. *No other formula is in $\mathcal{L}_f$.*

The symbolic sums $LC^{(n)}$, which are formed as follows.

1. *The constant 0 is in $LC^{(n)}$.*

2. *If $v_1, v_2, \ldots, v_n$ are terms, then the n-tuple $\langle v_1, v_2, \ldots, v_n \rangle$ is in $LC^{(n)}$.*

3. *If l_1, l_2 are in $LC^{(n)}$, then $(l_1 + l_2)$ is in $LC^{(n)}$.*

4. *If l is in $LC^{(n)}$ and z denotes a non-negative integer, then zl is in $LC^{(n)}$.*

5. *If x is a variable and l is in $LC^{(n)}$, then $\sum_x l$ is in $LC^{(n)}$.*

6. *No other symbolic sum is in $LC^{(n)}$.*

Regarding the symbolic sums, we note that, in order to be able to simplify net structures with multiple arcs between elements of the net, we need a representation of the merger of the arcs. We propose to use the symbolic sums to indicate a linear combination of the constituent expressions; this is shown in fig. 7.3. In this figure we show a net to be simplified in (a); note how we consolidate the places Pa, Pb, and Pc to P (as well as Qa and Qb to Q) and add multiple arrows instead in (b). Also note how we designate the arc expressions, which are further merged into one arc in (c). Finally, we note that $< a > + < c >$ can be replaced by $< x >$ and the net further simplified into (d).

Now we are ready to give a formal definition of the Pr-T net. Let $LC_\Sigma(A)$ denote a set of symbolic sums of a set A and let A^* denote the set of all finite tuples of A.

Definition 60 (Predicate / Transition net) *A* Predicate / Transition net *(Pr-T net) is a 3-tuple $N_L = (N, A, M_0)$, where*

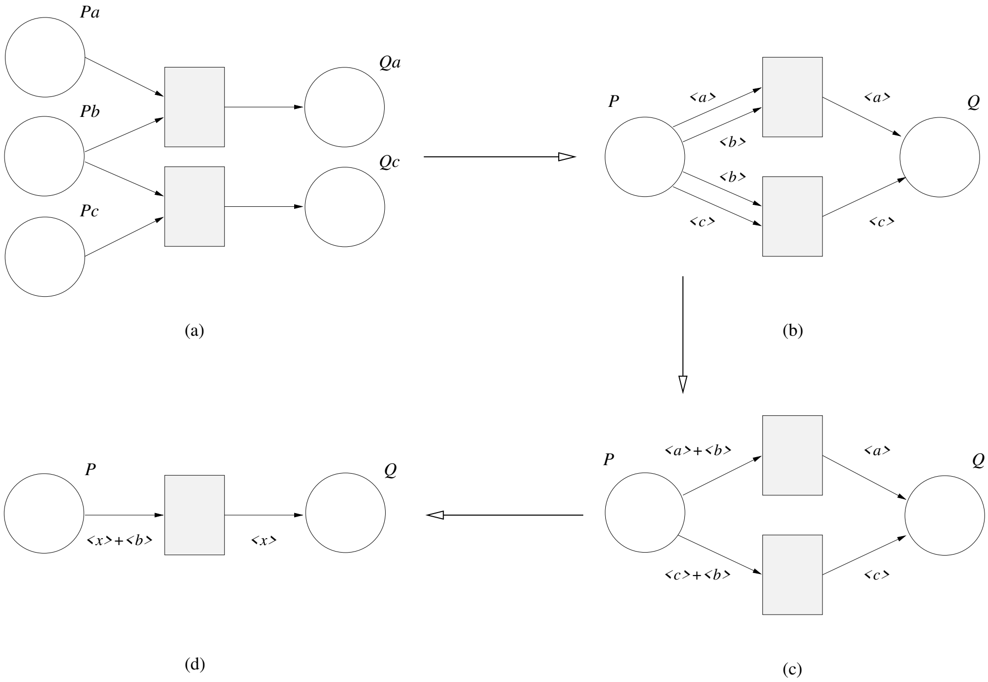

Figure 7.3: Notation for multiple arcs and symbolic sums. Based on [Gen87].

1. *N is an underlying Petri net, $N = (P, T; F)$ according to definition 56,*
2. *A is the* annotation *of N, a 4-tuple $A = (A_N, A_P, A_T, A_F)$, where*
   (a) *$A_N = \mathcal{R}$ is a first-order structure for $\mathcal{L}$, called the* support *of N.*
   (b) *A_P is a bijection between the set of places P and the set of variable predicates that assigns an* arity *to each place. The arity is the size of tuples in that place.*
   (c) *$A_T : T \rightarrow L_f$ assigns a well-formed formula, called a* firing condition *to each transition. All variables that are outside the scope of the quantifiers $\exists$ and $\forall$ must appear in some symbolic sum that A_f assigns to the arc hitting the transition.*
   (d) *A_F assigns to each arc a symbolic sum of $LC^{(n)}$. The arities of all the symbolic sums of tuples in an arc must equal the arity of the place which the arc hits.*
3. *$M_0 : P \rightarrow LC_\Sigma(A_N^*)$ is the* initial marking *of N. It assigns to each place a symbolic sum of tuples of the carrier of A_N. The size of the tuples must equal the arity of the place where they are located.*

A marking of the net consists of tuples of individuals in the places, which denote the interpretation of the relation symbols, i.e. the state of the concurrent system.

As we have seen, when drawing Pr-T nets we put the firing conditions inside the transition squares, arc expressions beside the arcs, and initial markings inside the circles denoting places; as shown in the previous figures (fig. 7.1 and 7.2).

7.3.6 Formal definition of a Petri net modelling a genetic algorithm

When modelling the genetic algorithm, as outlined above, the Pr-T net will have the following components.

Definition 61 (Petri net genetic algorithm) *A genetic algorithm is defined by a Petri net having the following components.*

- N *The Petri net N is as depicted in fig. 7.1. Note that this is an example only; other nets may model genetic algorithms as well.*
- A *The net annotation A is formed from the vocabulary of definition 59 and is as follows.*
  - A_N *The structure A_N is as defined in definition 58.*
  - A_P *The predicates A_P are $P_1 : |\Pi| = m$, where m is a constant integer > 0 denoting the size of the population of the genetic algorithm. Either P_2 or P_3 is empty (i.e. the place is empty), whereas the other one contains the empty token $<>$. Finally, $P4 : |\mathcal{S}_\alpha - \Pi| = n$, where $m + n = |\mathcal{S}_\alpha|$ is the size of the set of possible individuals in the genetic algorithm (i.e. the size of the genetic space)*
  - A_T *The firing conditions A_T are $T_1 : \omega_\xi$ and $T_2 : \omega_\Pi$ for the transitions T_1 and T_2, respectively.*
  - A_F *The arc expressions (i.e. symbolic sums) A_F are as depicted in fig. 7.1.*
- M_0 *Initially, the marking is such that the tokens in P_1 denote the inital population Π_0.*

This fully describes one simple type of genetic algorithm as a Petri net, in particular a Pr-T net.

7.4 ANALYSING THE GA USING A PETRI NET

There are many ways of analysing the Petri net: state analysis, dynamic or static property analysis, and so forth (see e.g. [Pet81]).

In this section we will consider the Petri net given in figure 7.4, which is the simpler version of the model in figure 7.1 with the place $P4$ omitted both for clarity and because its role is not needed in a simulation of the net (recall that its main advantage is that it closes the token space, i.e. individual space from an ontological point of view as described in section 7.3.2 on page 90).

In general, the most promising and fruitful approach is the *reachability graph* used in reachability analysis.

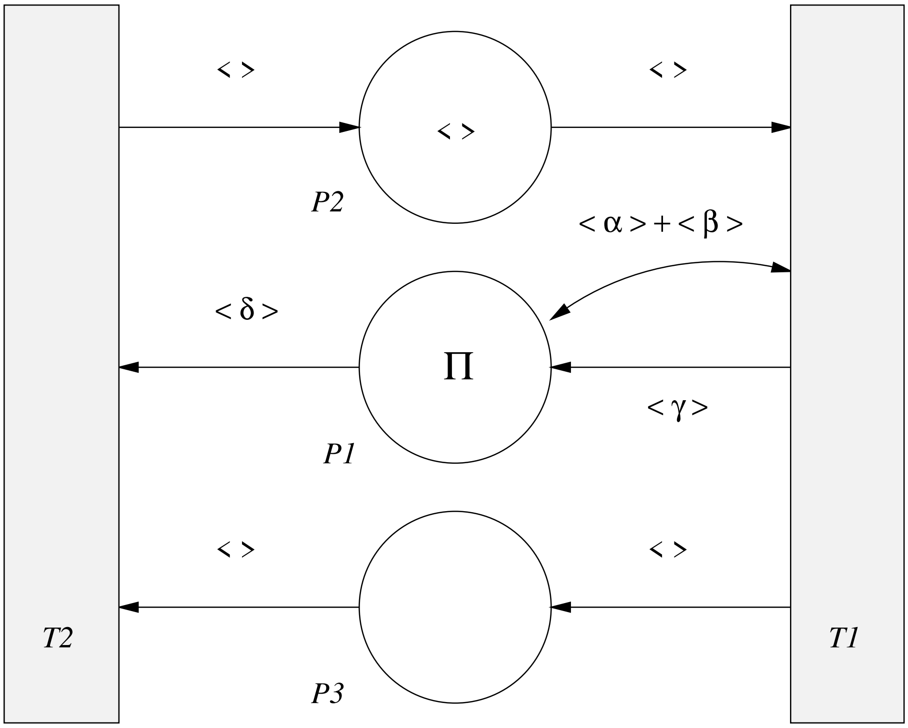

Figure 7.4: Genetic algorithm as a Petri net

Definition 62 (Reachability graph) *A reachability graph is a directed graph $\langle S, T, s; F\rangle$ with a set of nodes S, called states, a set of edges T, called transitions, a distinguished start state $s \in S$ and a set of terminal or final states $F \subseteq S$.*

For Petri net representations the reachability graph corresponds to the transitive closure of the Petri net firing rules over M_0, the initial marking. States of the graph are reachable markings of the Petri net. A transition in the reachability graph corresponds to the firing of a single Petri net transition. The start state corresponds to M_0. Final states of a reachability graph correspond to Petri net markings in which all of the marked places model the termination of a task.

For the net model in figure 7.4, the reachability graph is extremely simple, and is given in figure 7.5. Notice that the uppermost node is s, the starting state node. There are no final state nodes, as the net is cyclic.

To see the components of the graph explicitly, we have the following graph definition.

$$\begin{aligned} \{\langle P1 : \Pi, P2 :<>\rangle, \langle P1 : \Pi, P3 :<>\rangle\}, \\ \{T1, T2\}, \\ \langle P1 : \Pi, P2 :<>\rangle; \\ \{\}\rangle \end{aligned} \tag{7.2}$$

where we have put the components of the graph (which from definition 62 is $\langle S, T, s; F\rangle$) on separate lines for clarity.

The graph is, as pointed out, very simple: there are just two markings: either $P2$ or $P3$ has the empty token, with $P1$ always having the Π tokens.

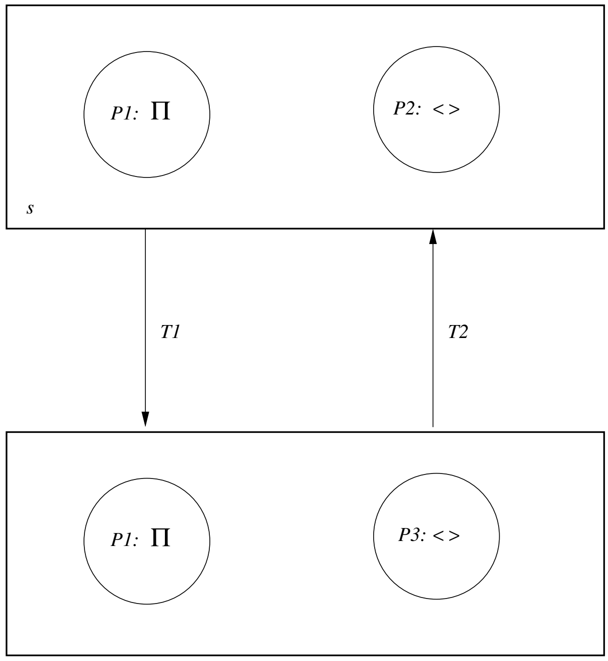

Figure 7.5: Reachability graph

It is perhaps more instructive to consider the internal 'microstates' that $P1$ is involved in in order to gain an understanding of what is happening here. They are shown in figure 7.6.

As can be seen, we have decomposed the $T1$ transition involving $P1$ so that the generation of γ can be clearly distinguished. However, we can still notice that the number of tokens, i.e. the marking of $P1$ does not change, which is the essence of the reachability graph of equation 7.2. Only the markings at $P2$ and $P3$ change.

In order to gain an insight into the simulation and analysis of the net, we have analysed the net model using `PROD`, the Pr-T net reachability analysis tool developed at the Helsinki University of Technology by Varpaaniemi *et. al.* [VHHP95].

The tool we used, `PROD` demands a text file that describes the net. For this model, the description of the net is contained in a file called `ga.net`, which is reproduced in figure 7.7.

The net description is similar to a program in the C language, and the tool is indeed implemented as a C preprocessor generating C code for a reachability tool specific to the net being analysed. The first line defines a macro to indicate the generation of new individuals (an ω function on the individuals α and β). The three places $P1, P2$ and $P3$ are defined; $P1$, the population, is marked with 10 (in this simple example) tokens (i.e. individuals in the genetic algorithm), whereas $P2$ is marked with the empty synchronisation token used for stepping through the net. $P3$ is left empty. Finally, the two transitions $T1$ and $T2$ are defined. In each transition we have defined the arcs inwards (`in`) and outwards (`out`), each with the places where the arc originates and the function label indicating the token 'movement'. In addition, $T1$ has a computing block that computes the new value for γ from its

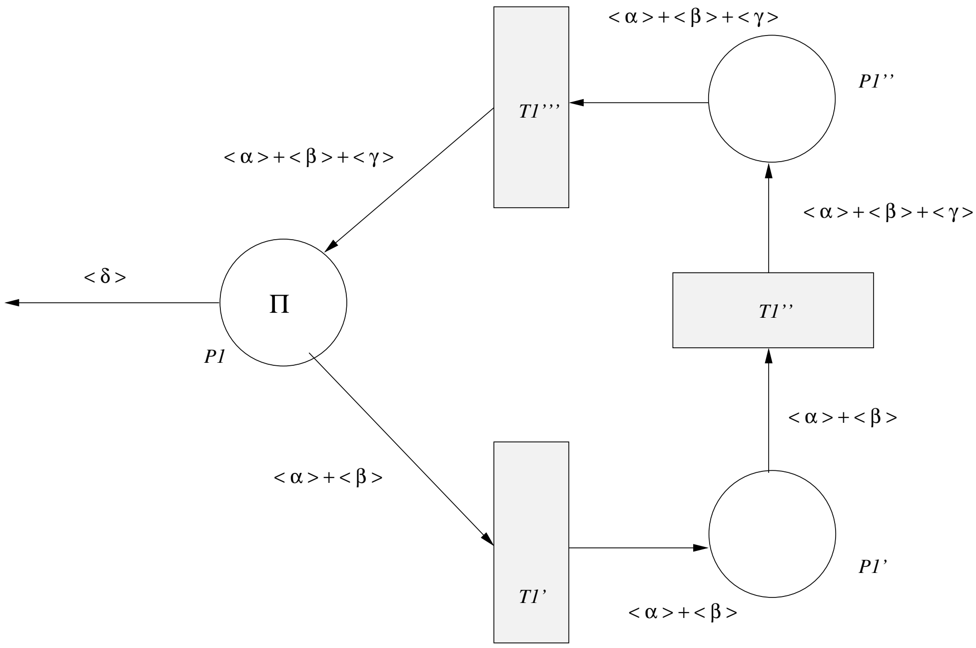

Figure 7.6: Transitions internal to $P1$ and $T1$

parents α and β.

Rather interestingly, there are many ways to enhance the net model to provide a more faithful model of the genetic algorithm. The improvements would provide such things as a more detailed selection phase (compared with figure 7.1 or figure 7.4), a termination clause, either based on generations (a counter) or fitness value (implemented as a gate in PROD, see [VHHP95]). None of these are essential from the modelling point of view, although they do make analysing the Petri net easier from a simulation point of view. The abovementioned improvements are sketched in figure 7.8. Note that we could have included our 'microstates', as well called them, from figure 7.6 but have chosen not to do so.

To explain the improvements further, we list the places and transitions below with descriptions.

- **P1** holds the population Π as before
- **P2** holds the generation counter; initially marked
- **P3** holds the generation counter; initially unmarked
- **PT** holds the termination token (individual), initially unmarked
- **T1** is the generation transition as before. It accepts individuals α and β and produces the offspring γ using the ω functions as described in the previous chapters and as shown in the PROD net description file in figure 7.7
- **T2** removes the unfit individual δ, almost like before as well as increments the generation counter n

```
#define omega(x, y)   ((x>y?x:y) +1)

#place P1 mk(<.1..10.>)
#place P2 mk(<..>)
#place P3

#trans T1
    in   { P1: <.alpha.> + <.beta.>; P2: <..>; }
    out  { P1: <.alpha.> + <.beta.> + <.gamma.>; P3: <..>; }
    comp { gamma = omega(alpha, beta); Accept(); }
#endtr

#trans T2
    in  { P1: <.delta.>; P3: <..>; }
    out { P2: <..>; }
#endtr
```

Figure 7.7: PROD net file `ga.net` for the net in 7.4

- **T3** selects the unfit individual δ from the whole population Π using the selection criteria and functions described in the previous chapters
- **T4** accepts an individual τ from the population Π and checks whether a termination value has been reached; if so, it passes the token τ on to the termination place PT

As we notice, this model is already very close to the canonic genetic algorithm as described in the previous chapters, the main difference being that it is strictly speaking not generational, and that the termination value check takes place continuously ($T4$ fires all the time until a high enough value has been reached). Recalling that Petri nets are widely used in net simulation and analysis, it is expected that this model may be further enhanced for pragmatic reasons to provide various functions deemed necessary for any number of reasons; however, these fall outside the scope of this work and have not been investigated.

In a Petri net, there are a number of characteristics that are formulated in order to better understand the net. These are treated below [Rei82].

- **Liveness.** A net is *live* if every transition can always occur again, or more precisely, if from any reachable marking it is possible to reach some marking that enables the transition. Looking at the net in figure 7.4 it is clear that it is live; the net in figure 7.8 is not since transition $T4$, once occurred cannot occur again. Alternatively, depending on the definition of the place PT, can only occur until PT's capacity is not exhausted. If PT does not have an upper bound, then the net is of course live.
- **Boundedness.** A net is *bounded* if each place has an upper bound, or token capacity. Clearly, the nets in figures 7.4 and 7.8 are bounded.

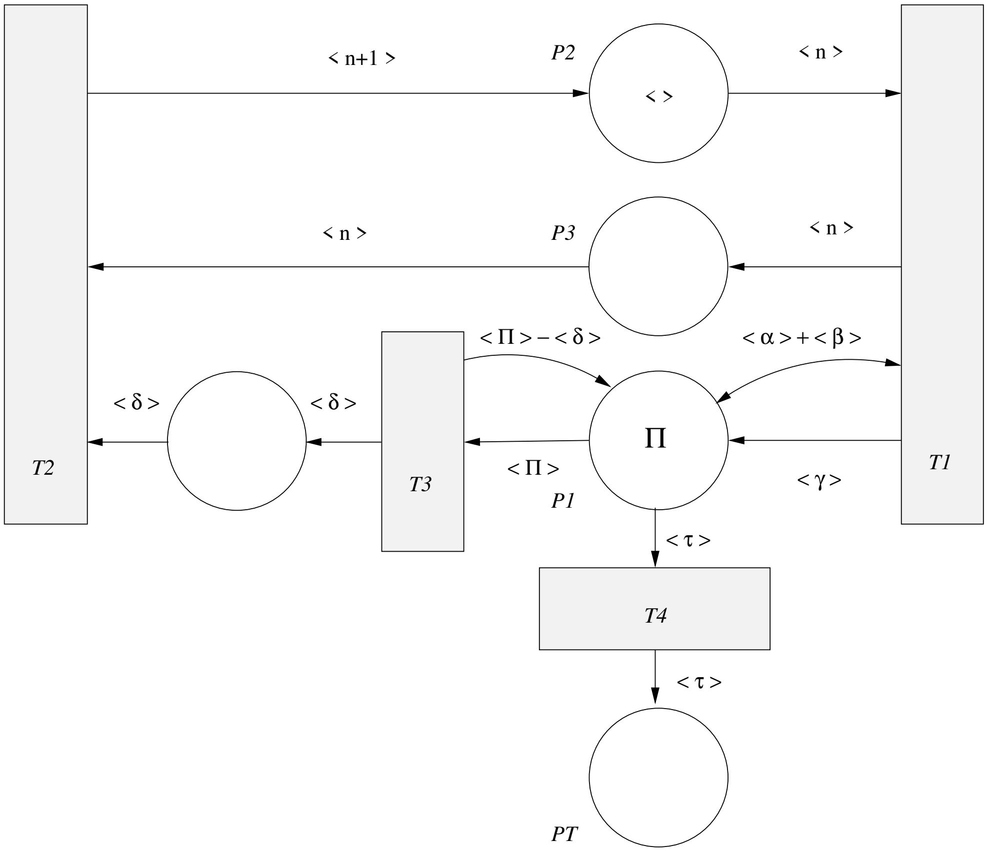

Figure 7.8: More complete genetic algorithm as a Petri net

If the place PT in 7.8 does not have an upper bound then that net is unbounded.

- **Deadlock-freedom.** A net is *deadlock-free* if every reachable marking enables some transition. Again, both our nets are clearly deadlock-free.

- **Reversibility.** A net is *reversible* if it can always reach its initial marking; in our cases the net in figure 7.4 is reversible, whereas the net in figure 7.8 is not. If a token gets to PT, it is never removed, and thus the initial marking cannot be reached again. Execution of a reversible net is *cyclic*.

7.5 GENETIC ALGORITHM COMPARISONS

In order to show the expressiveness of the PGA model we develop comparisons between the most common genetic algorithms and their PGA counterparts.

We may develop an understanding of the various types of genetic algorithms by listing their main characteristics and show the correspondencies in the standard formulation and the Petri net based formulation.

We will examine some of the following characteristics, listed in table 7.1.

Characteristic	Description
Selection mechanism	Various types, e.g. proportionate or linear selection
Elitism	Best (one or several) is saved
Recombination	Various types, e.g. one- and two-point, uniform recombination (crossover)
Mutation	Various types and rates
Fitness	Function determining which survive to the next generation
Convergence	State when population fitness tend to stay unchanged from generation to generation
GA type: Generational	Whole population renewed each generation
GA type: Steady-state	Less than whole generation renewed each generation

Table 7.1: Genetic algorithm characteristics

7.5.1 Selection and elitism

Selection is the mechanism by which parents are chosen in the traditional genetic algorithm; this is handled in various ways, a number of which are listed in table 4.4 on page 55 together with some important selection parameters in table 4.5 on page 55.

In terms of the Petri model of the genetic algorithm, these are handled by the arc expression from $P1$ to $T1$. Note though, that the arc expressions merely tell the transition function which individuals it may act on; in this sense it is really part of the transition function, not the actual arc expression.

Elitism is the pragmatic exception of certain well-fit individuals to forego selection, and go straight to the next generation unchanged, and without any further ado. In one way it defeats the whole genetic algorithm; in another way it ensures that good individuals that have been found by chance early on in the evolution of the population do not disappear.

7.5.2 Recombination and mutation

In the traditional genetic algorithm this is handled within the ω_ξ functions. There are many mechanisms; see e.g. [Gol89] or [BT95] for details on both selection and recombination.

In the Petri net model of the genetic algorithm these are handled by the transition function in $T1$ (refer to fig. 7.1). Also see the paragraph on convergence below.

7.5.3 Fitness

In the traditional genetic algorithm, the fitness function is part of the determination of which individuals survive to the next generation. It is always problem-dependent, and may be quite complicated since it has to determine the sometimes minute differences between individuals. Earlier, see page 56 we introduced one possible way of going forward with the help of equality environments; many more exist.

In the Petri net model the fitness function is built into the transition function at $T2$, as pointed out in the description of the Pr-T net in section 7.3.1 on page 88. This means that it may be applied continuously during the evolution of the population, every time new individuals are generated by $T1$; recall that the condition for its application is that $|\Pi|$ increases, which it must not do.

7.5.4 Convergence

Convergence is the tendency of the population of subsequent generations to stay the same, from a population fitness point of view. In the traditional genetic algorithm it is not a function or mechanism, but rather a problem-dependent state that may or may not be reached during the course of the evolution of the population.

In terms of the Petri net model convergence may be seen in two ways

- transition $T2$ always removes the tokens denoting new individuals generated by transition $T1$ effectively keeping the token set at $P1$ static (indicating that the population they denote is static as well), or
- transition $T2$ always removes tokens denoting individuals with exactly the same fitness value as the ones that were generated by transition $T1$.

It is easily seen that these two cases correspond to the time-conservative and time-progressive selection mechanisms; see definitions 51 and 52 on page 61.

7.5.5 Genetic algorithm type

Traditionally, we have the generational and steady-state types of genetic algorithms; see chapter 2 starting on page 9.

In the Petri net model we strictly speaking can deal with the two models with the help of suitable definitions for the transitions. Since $T2$ fires immediately on changes in the token set size denoting the population Π in place $P1$, we obtain a steady-state model with $T1$ generating one token (denoting a new individual) only; with a creation of a full size population we obtain a generational model. Of course, we may have any number in between; using the definition of generational and steady-state we would call this steady-state as well (cf. definitions 42 and 43 on page 53).

7.6 DISCUSSION

We have shown how genetic algorithms may be modelled and simulated using high-level Petri nets. The Petri net formalism is quite powerful: many techniques exist for analysis of its static and dynamic properties, as well as many easy-to-use tools for the same purposes.

However, it must be pointed out that all our nets in this chapter are very simple from a Petri net analysis point of view; indeed, we can analyse them by hand since the number of places and transitions is so small. A tool like `PROD`, mentioned above, comes into its own only when we reach hundreds or thousands of places and transitions. Clearly, we are very far from this in our models.

But, nevertheless, the above description of the simulation of the workings of genetic algorithms using high-level Petri nets represents a promising avenue for further research.

8 INVESTIGATING THE TEMPORAL GA

This chapter will investigate the temporal aspects of the genetic algorithm, its behaviour and characteristics using the parameters and formalism described in the previous chapters. We will show that the temporal and probabilistic approach, using a modified standard implementation of the genetic algorithm, the SUGAL Genetic Algorithm Package [Hun95b, Hun95a].

This chapter proceeds as follows. We will first briefly describe the standard SUGAL package. Then we will describe the test procedure, and, finally, we will show results both for the standard and temporal selection and discuss the results comparing them with the standard algorithm.

We will then discuss the application of the genetic algorithm (exemplified by SUGAL) on a real, NP-hard problem: the travelling salesperson. This section will describe the behaviour of the genetic algorithm with standard selection mechanisms, and then with temporal ones, ending with discussing the results obtained.

8.1 THE SUGAL PACKAGE

SUGAL is further described by Andrew Hunter in his SUGAL 2.1 User Manual [Hun95b] and SUGAL 2.1 Programming Guide [Hun95a].

It is a software package, written in the C computer language, designed for experimentation with genetic algorithms and related techniques. It is intended to be used in genetic algorithms research; consequently the major emphasis is on providing a large number of options, on configurability and extendibility. Efficiency is considered important but has been sacrificed where it would conflict with the above requirements.

Note that the SUGAL genetic algorithm subsumes most of the GA models which exist as subsets of its functionality, and can be extended to model those it doesn't subsume. Certainly the Holland/Goldberg/DeJong models, Whitley's Genitor, and Fogel's real parameter model are covered (and extended to arbitrary datatypes), along with other more obscure versions. SUGAL breaks the features of the various algorithms into separate parts, so that an extremely extensive range of hybrids of the standard models is also available. This is one of the main reasons why it was chosen for this investigation by the author. Aspects of *evolutionstrategie* are not covered (e.g. where the mutation rates change which are themselves mutated).

The main characteristics of SUGAL in terms of the run of the genetic algorithm and its parameters are as follows [Hun95b].

1. Chromosomes are strings of bits. In our setting we have used a length of 10 bits per chromosome.
2. Chromosomes are randomly initialised, with each bit having a 50/50 chance of being set or cleared.

Parameter	Type	Value
Replacement	method	uniform
	condition	unconditional
	rate	1
	elitism	no
Population	size	20
	generations	100
Crossover	type	two-point
	rate	0.60
Mutation	type	invert bit
	rate	0.01
Chromosome	bits	10

Table 8.1: Standard GA parameters

3. The entire population is replaced by its children on each generation. We have used a population size of 20 and tested for 100 generations.

4. Parents are randomly selected to produce children using the roulette wheel method. Each time a parent is needed, one is randomly chosen from the population. The chance of selection as a parent is proportionate to a chromosome's normalised fitness (see below). This means that better chromosomes will generally produce more children, although the stochastic nature of the process means that sometimes they won't, and sometimes poor solutions will produce children.

5. All children are produced by crossover of two parents; two children are produced simultaneously. Two-point crossover is used; this means that the two parent chromosomes are joined at two randomly selected crossover points somewhere along the length of the chromosome, and swap the sections on either side.

6. Once children have been produced, they may be subjected to mutation. The average number of mutations is 0.5 per child (some children may experience a number of mutations, while others experience none). If a mutation occurs, a randomly selected bit in the chromosome is inverted: that is, its value is flipped.

Fitnesses are normalised internally to the range $[0, 1]$ in order to make them comparable; however, note that this also means that when we introduce our temporal selection methods the equality interval will also have a value between 0 and 1 (inclusive).

The main parameters used when running the genetic algorithm producing the results shown in figures 8.1, 8.2, and 8.3 are listed in table 8.1. We will use the same parameters whenever appropriate in the tests below as well. Especially note that elitism is turned off.

In general, SUGAL uses the following baseline genetic algorithm. Note that, as has been pointed out before SUGAL is extremely versatile and may be used to mimic virtually any of the standard types of genetic algorithms, so clearly it is of paramount importance to choose a set of parameters that are both widely used and meaningful for standard work as well as for our modified algorithm with temporal parameters in addition to the ones in the normal package.

```
Initialise population
Evaluate chromosomes
Calculate normalised fitnesses
Output statistics on population
for each successive generation
do
 Generate and evaluate candidate replacements
 Replace members of the population with candidates
 Reevaluate unchanged members [virtually always ignored]
 Normalise fitnesses
 Output statistics on population
enddo
```

As can be seen, this is very much the canonical generational genetic algorithm, as described by Holland [Hol92a] and Goldberg [Gol89].

8.2 SUGAL TEST RUNS

Now when we have a basic understanding of the package, this section first deals with the test problems used and then provides an overview of the tests performed. We then detail the temporal selection method tests and their results.

8.2.1 Test functions

The modified SUGAL package was run using the widely used De Jong test functions [DJ75]; see table 8.2. It is possible to devise particular problems which are suited to particular search methods, including the various types of genetic algorithm [Hun95b, WM95]. An implication of this is that, if we discover a particular form of algorithm to be good at solving a particular problem, this doesn't necessarily mean that it is good for any other problem. It is thus extremely difficult to make any definitive statements about what versions of the genetic algorithm are 'best'. An obvious approach to this problem is to devise a standard set of test problems (benchmarks), against which various algorithms can be tested. Kenneth De Jong [DJ75] did precisely this, and his five test functions are still widely used for genetic algorithm research. They are very simple functions, designed to exhibit various recognised types of search landscape, including: continuous/discontinuous, convex/nonconvex, unimodal/multimodal, quadratic/nonquadratic, deterministic/stochastic, low and high dimensionality. Although De Jong used binary representation, each of the functions is actually defined on real numbers: De Jong represented

Function	Definition	Features
F1	$\sum_{i=1}^{3} x_i^2$	unimodal quadratic, min at (0,0,0)
F2	$100{\cdot}(x_1^2 - x_2)^2 + (1 - x_1)^2$	multimodal
F3	$\sum_{i=1}^{5} int(x_i)$	discontinuous ‘staircase’ function
F4	$\sum_{i=1}^{30} i \cdot x_i^4 + Gauss$	high dimensional, stochastic
F5	$0.002 + \sum_{j=1}^{5} \frac{1}{(j+\sum_1^2(x_i - a_{ij}))^6}$	extreme multimodality, sharp peaks

Table 8.2: DeJong test functions

each real number parameter as an integer, made up of a number of bits, and divided through by a constant factor. The five functions are described in table 8.2.

Note that $Gauss$ is a random variable with a Gaussian (normal) distribution, mean 0, and standard deviation 1. In F5, the a_{ij}'s are constants, which in SUGAL can be set programmatically. It is worth remembering that the DeJong test functions, F1-F5, are extremely simple problems by genetic algorithm standards, and any conclusions drawn from experiments with these test functions need to be treated with extreme caution. Nevertheless, they do provide a basis from which to demonstrate various genetic algorithm concepts, and are widely used as a common benchmark.

8.2.2 Baseline tests

In order to provide a baseline from which to evaluate the tests, i.e. without employing temporal selection consider figure 8.1 depicting a typical run of function F1 from table 8.2. We show the evolution of fitness over 100 generations, together with the mean $\pm$ the standard deviation. Note that (by definition) we are minimising the function and that is why we start off with a high fitness value and optimise it towards 0. We also show the diversity in figure 8.2, which in this context is defined in a statistically more meaningful way: it is the mean normalised Hamming distance between all pairs of individuals in the population. The Hamming distance between two bit strings is the number of bits that differs between them; normalisation is achieved by dividing by the number of bits. Thus for a random population it is 0.5 because any two randomly chosen bit strings will differ in approximately half their bits. This can be clearly seen as the starting value in the figure. The diversity reaches 0 approximately by the 90th generation indicating a totally converged population. Note that we have chosen to use linear instead of

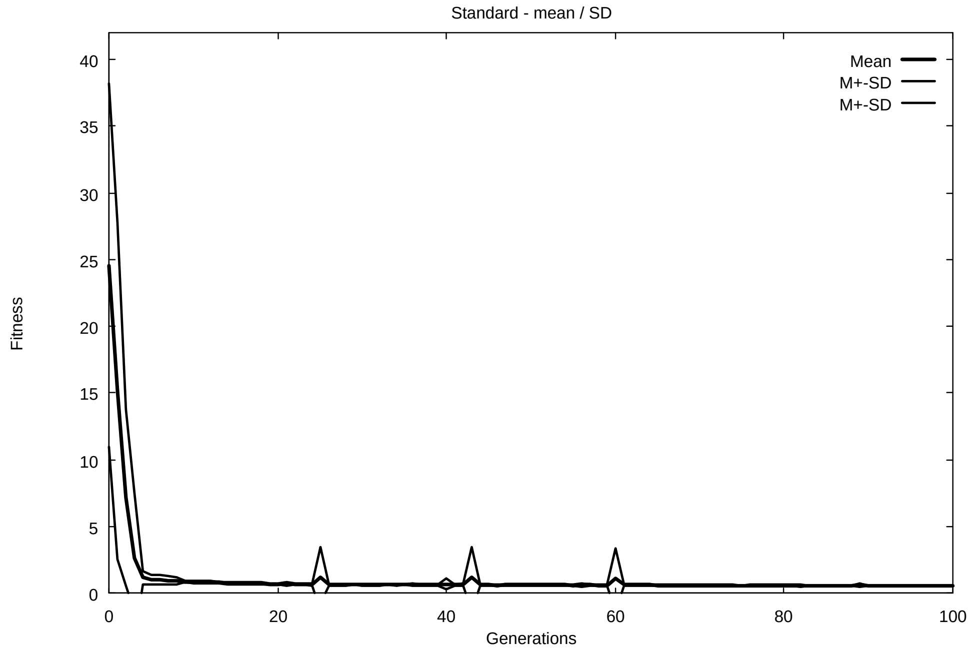

Figure 8.1: Standard fitness evolution for function F1. Baseline case.

logarithmic scale on the vertical axis; even if a logarithmic scale would have shown differences better in later generations of the run, these differences matter little, so in order not to emphasise unimportant variations at later generations we keep the linear scale with its virtually flat curves toward the end of the runs.

Regarding fitness evolution, we can clearly say that for this problem the fitness has been optimised in the first 10 generations of the run; the three blips at the 24th, 42nd, and 59th generations are due to mutation and are immediately deselected away. Redisplaying the data for the first 10 generations of figure 8.1 we obtain figure 8.3. Indeed, fitness has apparently essentially converged by the 4th generation, although we note that there is a small residue (approximately 0.5) left that the algorithm is unable to optimise away.

The standard fitness evolution for the baseline case for the first 10 generations is shown in figure 8.3. The convergence is clearly discernible.

8.2.3 Temporal selection tests

We now consider the same tests as before, but now we add the temporal selection mechanisms described in chapter 4 and 5.

The next figure, 8.4, shows the same but now we use temporal-conservative selection, with an equivalence interval of 0.1. It is interesting to notice how convergence has speeded up as evidenced by the smaller standard deviation and the slightly steeper slope of the mean. Perhaps even more interesting is to examine the evolution of diversity for this selection; it is shown in figure 8.5.

Comparing this with the corresponding standard one in figure 8.2 we notice how quickly the algorithm converges; a small residue of diversity is left,

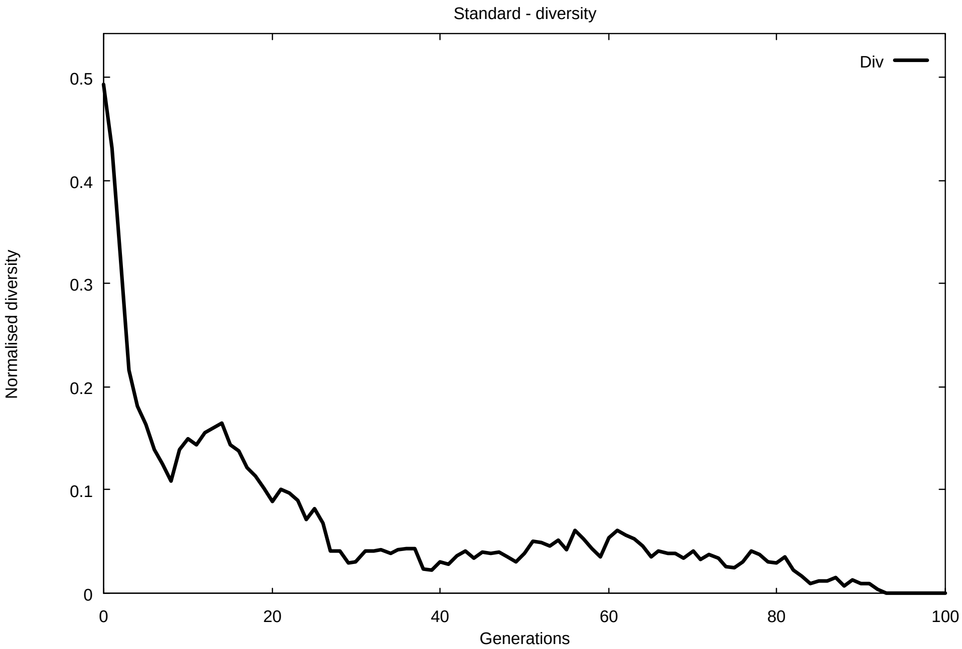

Figure 8.2: Standard diversity evolution for function F1. Baseline case.

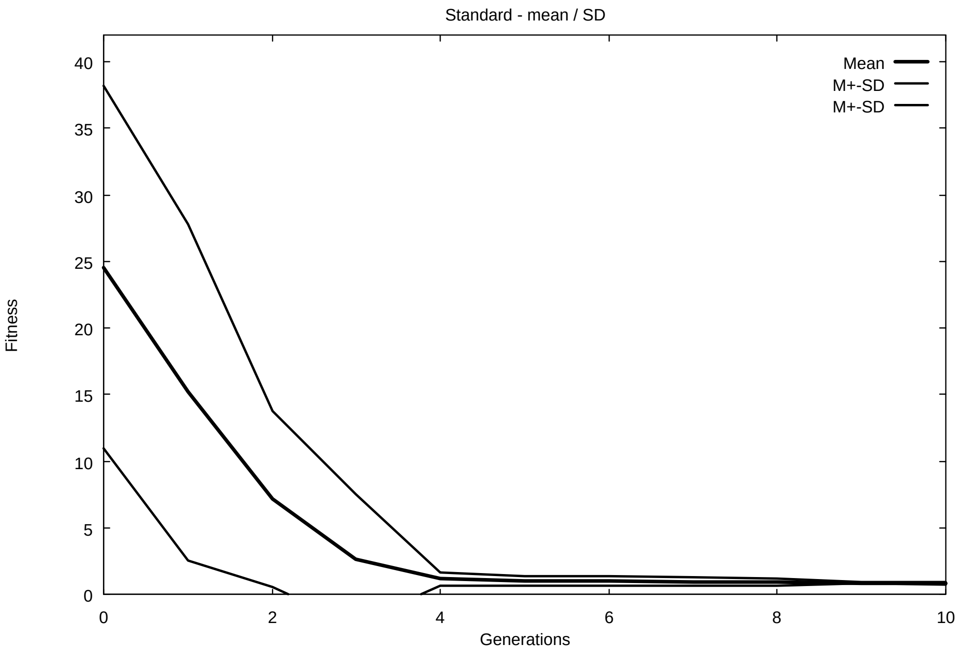

Figure 8.3: The first 10 generations of the standard fitness evolution for function F1.

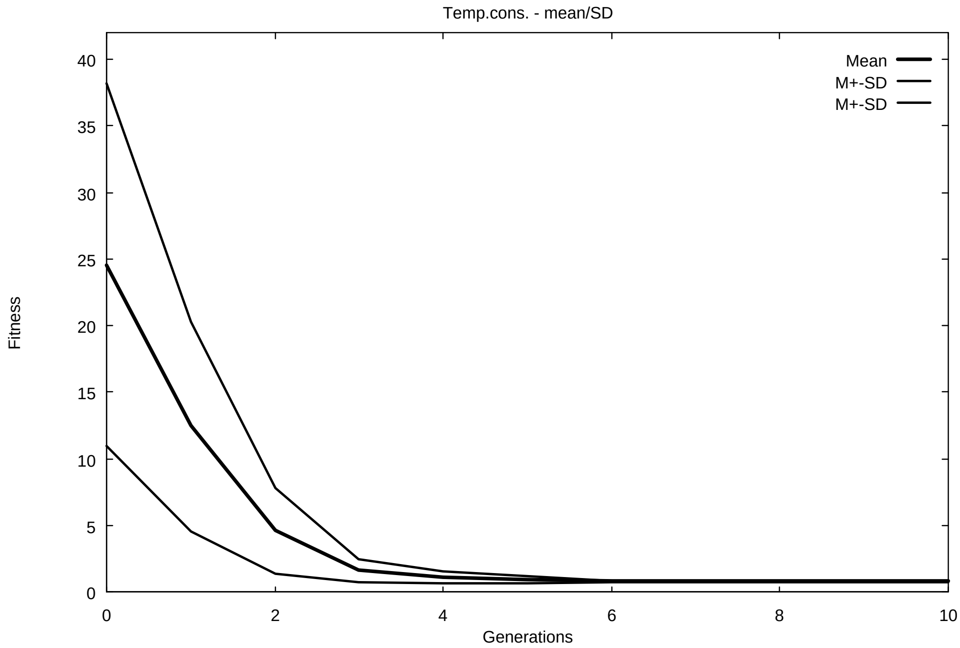

Figure 8.4: Fitness evolution using temporal-conservative selection for function F1.

however.

It is very interesting to see how diversity changes with different values of the equivalence interval. This is shown in figure 8.6. We can clearly discern how diversity decreases with decreasing equivalence interval. The explanation is simple: when fitness is within the equivalence interval the genetic algorithm makes essentially a random choice which individuals survives leading to statistically speaking no change in diversity at all.

Regarding temporal-progressive selection, it is really exactly the same as temporal-conservative until we reach the equivalence interval. We choose to show an expanded view of both kinds of temporal selection in order to show that the difference will only show up when we near the equivalence interval (or when a number of individuals start getting selected temporally instead of unconditionally); see figure 8.7. Using an equivalence interval of 0.1 the curves start to show differences below a fitness value of 1, with completely different behaviour below 0.2.

The corresponding curves for the evolution of diversity are shown in figure 8.8.

As a second example, consider figure 8.9 now depicting runs of problem F4 from table 8.2 shows the difference between the two temporal types of selection as well as a selection where the improved child replaces its parent. Unlike figure 8.8, progressive is better at problem F4 whereas conservative is superior at F1. In order to see how a 'traditional' selection mechanism behaves we have included the simple case of replacement if the child is improved. Notice how all mechanisms (especially outside the scale of the diagram, which has been scaled to show the differences) start off identically but start diverging when more and more individuals reach the equivalence interval, which for this case was set at 10.

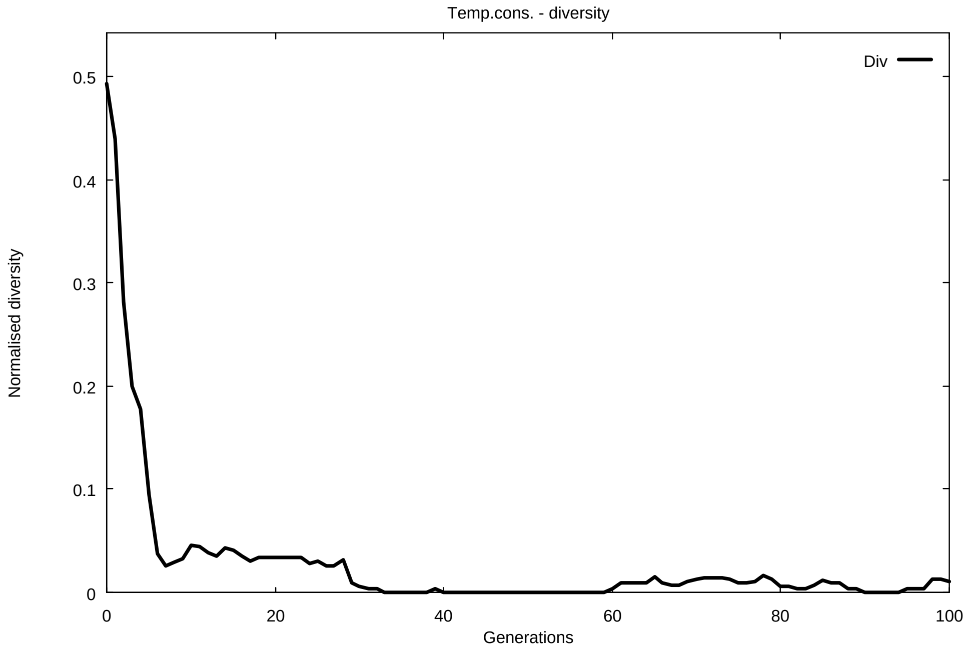

Figure 8.5: Evolution of diversity using temporal-conservative selection for function F1.

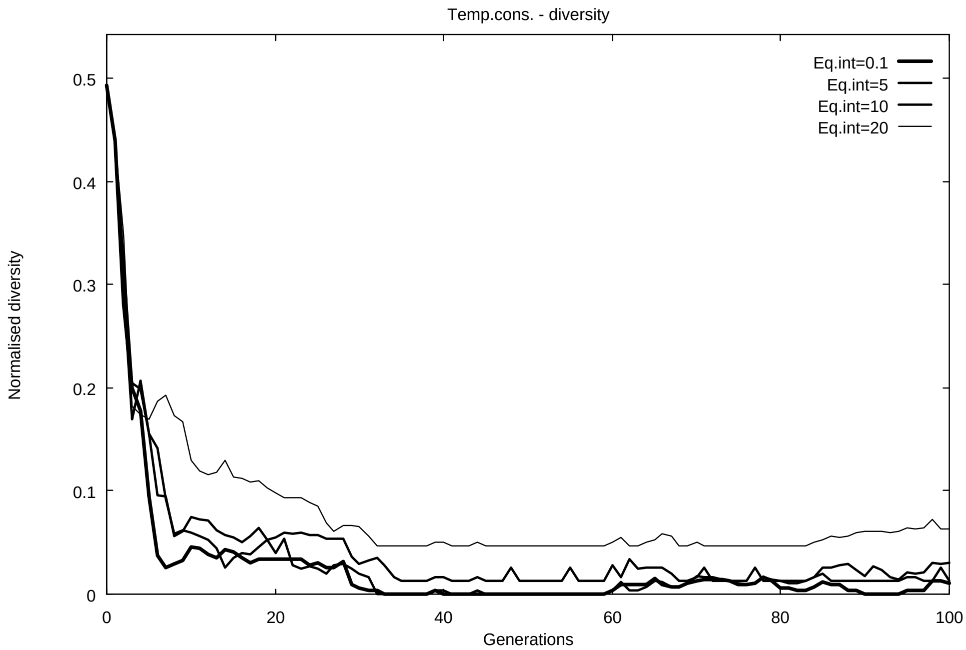

Figure 8.6: Diversity as a function of the equivalence interval for function F1.

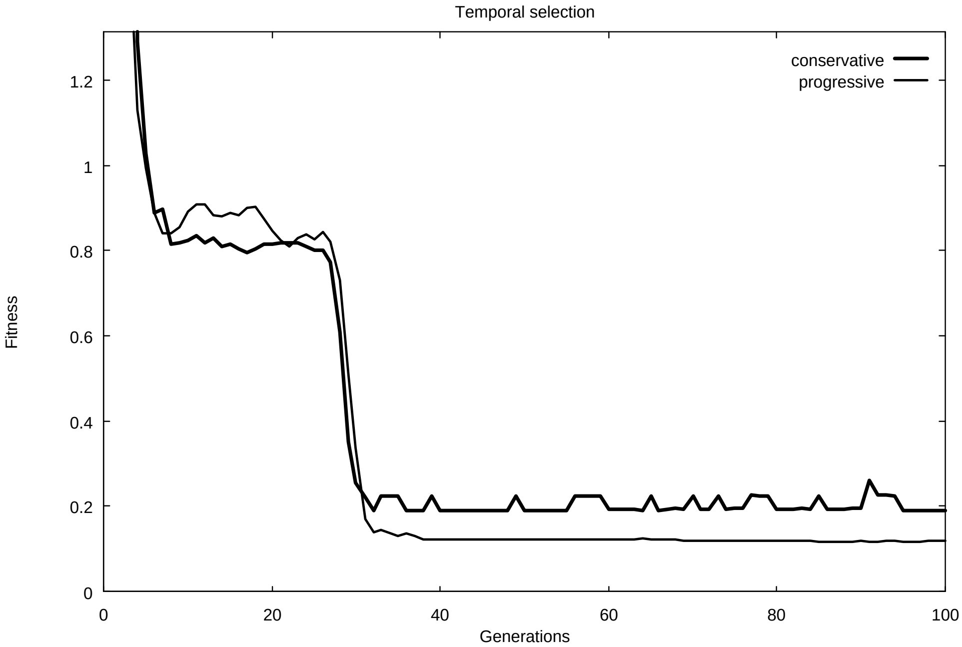

Figure 8.7: Fitness evolution using temporal selection for function F1.

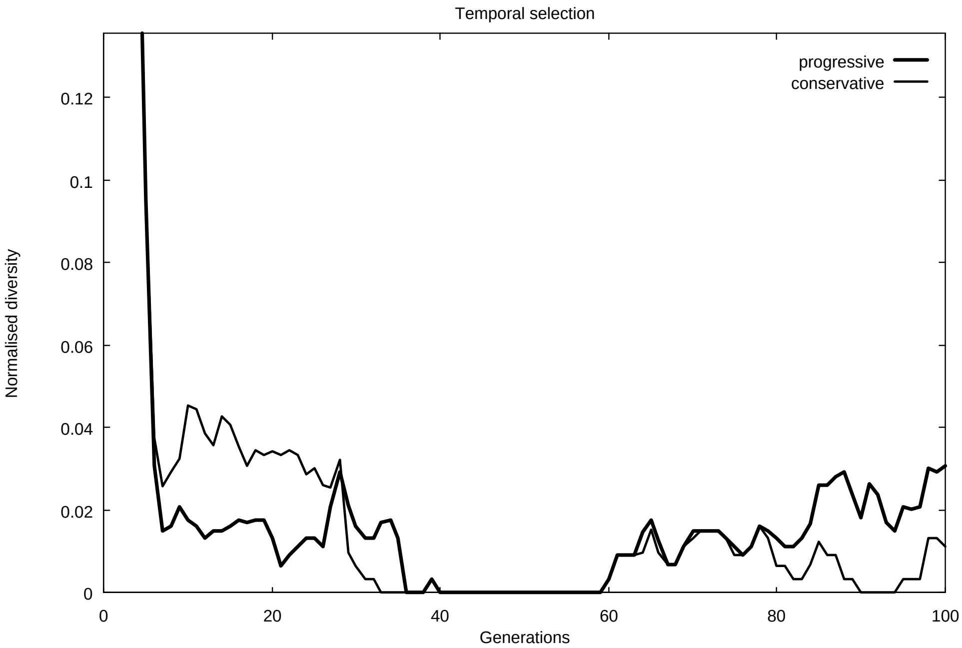

Figure 8.8: Evolution of diversity using temporal selection for function F1.

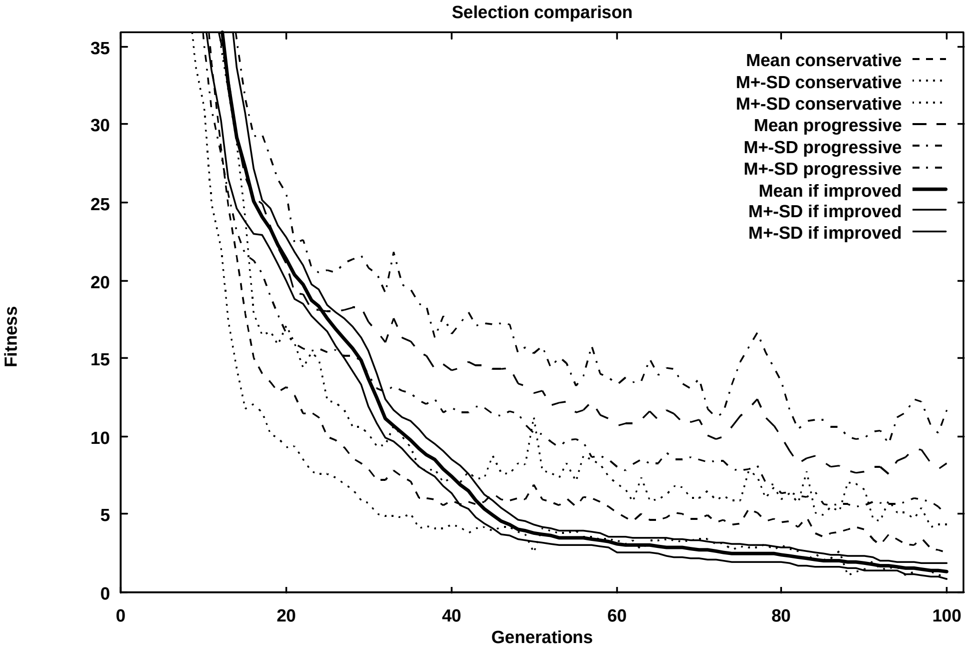

Figure 8.9: Comparison of temporal selection mechanisms for function F4.

We also show the curves for diversity for the same runs, in figure 8.10. As can be seen, no significant differences may be found.

The virtual population may also be examined at various points of the run of the genetic algorithm. We have two basic statistics: the actual number of individuals replaced per generation, and the cumulative number of replaced individuals up to a generation.

We show these two for the preceding De Jong test function F4 in figures 8.11 and 8.12, respectively.

Quite predictably, the population is initially dominated by replacement of its constituent individuals; indeed in the beginning all of the probable replacements take place (recall that we in this scheme generate children that replace their parents only if better; in a random population they would only be better half the time, on the average) and we have a replacement rate of 50%. Quite soon, however, it starts to fall as the population improves, finally reaching a residual level of around 5%. It is instructive to combine the cumulative replacement plots for the two temporal selection types with this 'if improved' selection scheme (used in figure 8.12) in order to compare the diversity evolution and convergence. Refer to figure 8.13.

The conclusions are fairly easy to draw: using the 'if improved' selection scheme we reach (approximately at generation 60) a plateau, where we no longer explore the space of possible solutions. In other words, the population has converged, apart from a small percentage due to mutation. However, this is not the case for temporal selection where the curve of cumulative replacements is straight, indicating a steady replacement rate. This means that we do continue to explore the landscape. Looking at the fitness and diversity data for the same problem, figures 8.9 and 8.10, respectively, we do not see any substantial differences in performance - all seem to find the

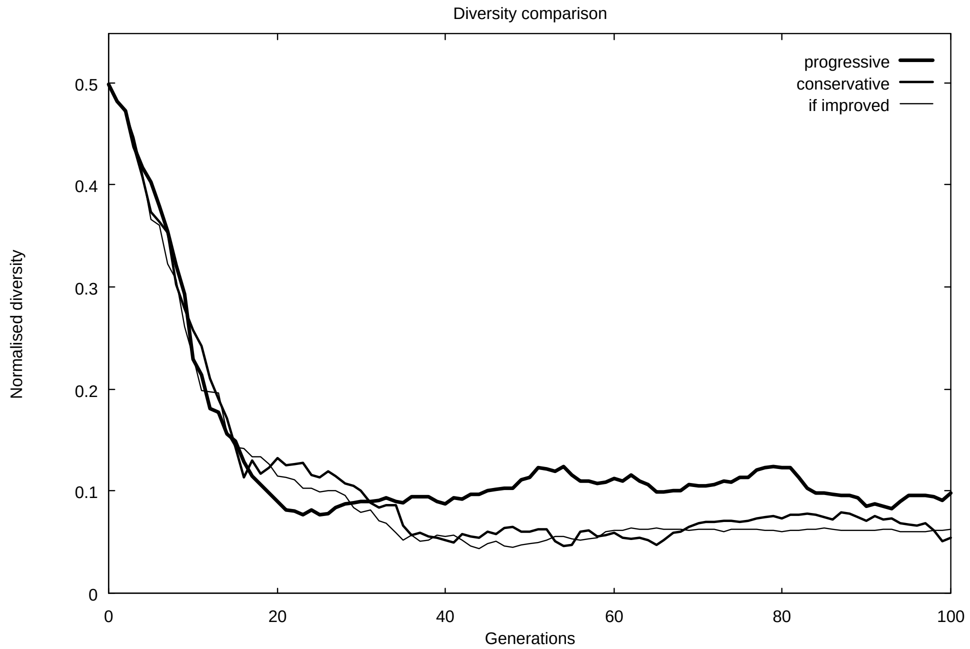

Figure 8.10: Diversity comparisons for different temporal selections for function F4.

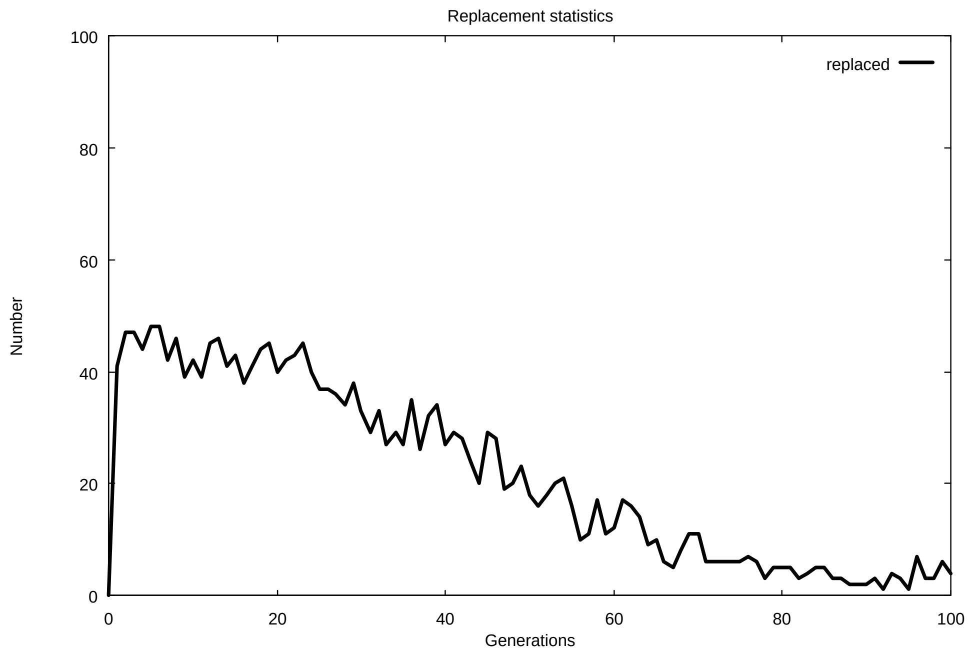

Figure 8.11: Replacement per generation for function F4. Baseline case.

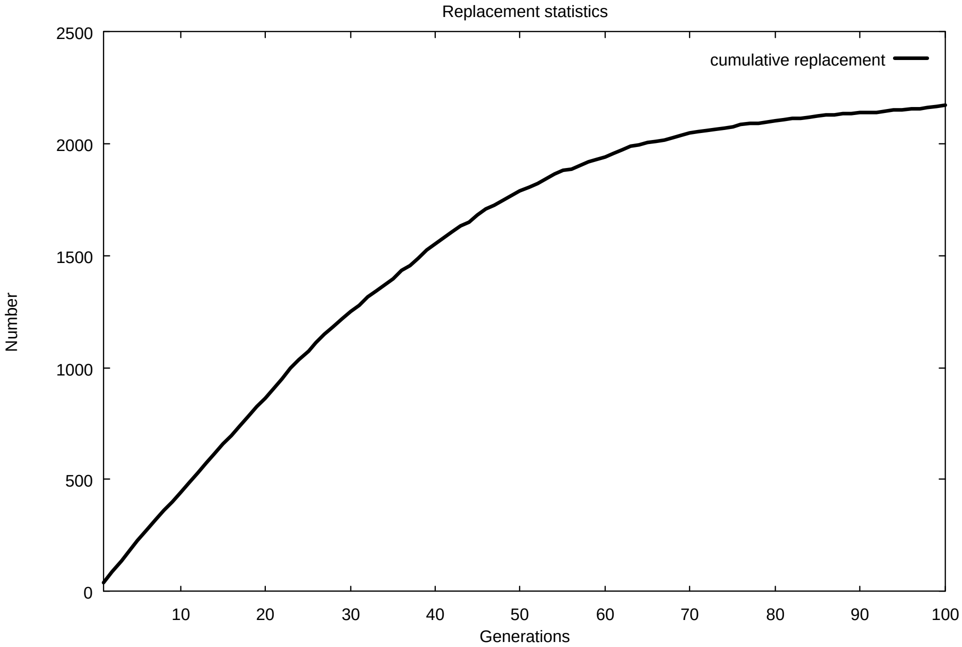

Figure 8.12: Cumulative replacement for function F4. Baseline case.

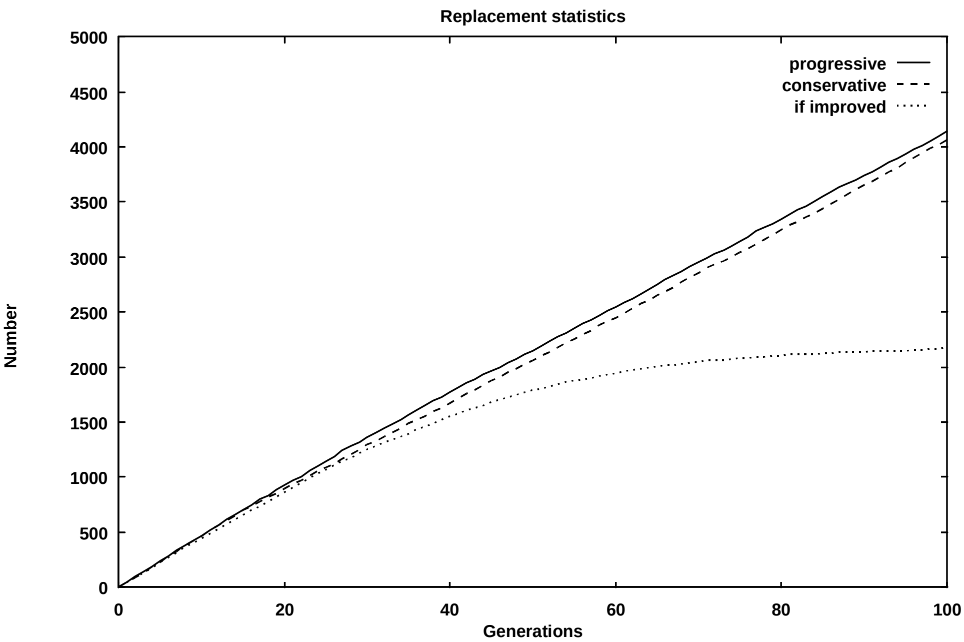

Figure 8.13: Replacement characteristics for function F4.

fittest individuals, and reach the same kind of diversity. The difference is thus that the temporal variants replace the individuals in the population with new ones on a continuing basis, something that is highly desirable in order to avoid the premature convergence phenomenon. Appropriately modifying the equivalence interval we may stay at any explorative level desired.

8.3 THE TRAVELLING SALESPERSON

Lastly, let us investigate the behaviour of the simple genetic algorithm here, with standard selection mechanisms as well as the temporal ones as detailed above on a real, NP-hard problem.

First, we describe the problem setting, its encoding in the genetic algorithm, and the evaluation function determining the fitness value. Then we will look at three cases: the genetic algorithm with standard selection mechanims, with temporal-conservative, and with temporal-progressive. Finally, we will discuss the results.

8.3.1 The problem setting

We will investigate in some depth a travelling salesperson (TSP) problem with 75 cities. Briefly, this entails finding a path through 75 points (cities) such that each point is visited only once, and such that the length of the path is minimised; the actual problem in question is depicted in figure 8.14. It is well-known that the general case is an NP-complete (NP-hard) problem, and as such cannot be solved in polynominal time. Indeed, for a TSP problem with 75 cities, the number of possible paths through them is an astronomical number, namely, for n cities $\frac{(n-1)!}{2}$ which for 75 cities is approximately $1.65 \cdot 10^{107}$.

The generic settings of the genetic algorithm are as follows, see Table 8.3.

As can be seen, the structure of the chromosome is fairly simple: it is a permutation of the coordinate pairs (x, y) for the cities. The fitness function is consequently also simple: it is the total length of the round trip, including returning to the starting point. Since the chromosome consists of pairs of coordinates we will ensure that crossover returns valid routes (i.e. with all cities, no duplicates or omissions) by first doing the crossover, and then scanning and replacing cities that are duplicated, effectively turning a two-point crossover into a multi-point. As a matter of fact, SUGAL offers several variations on the theme, with three different crossover types (for details, the source code for SUGAL is well commented in this regard).

Likewise, mutation must also take the special structure of the chromosome into consideration by always ensuring that it returns a valid route. SUGAL offers two mutation variations: swapping, where two cities are randomly swapped in the chromosome, and inversion, where a part of the chromosome is reordered by inverting its order of cities. Both of course return valid routes.

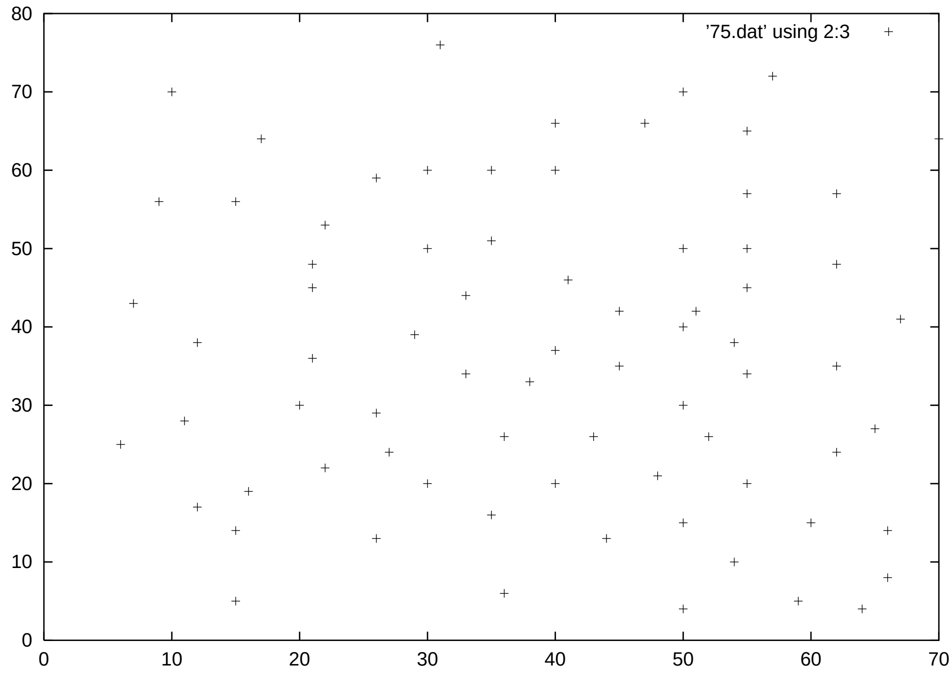

Figure 8.14: A Euclidean TSP problem with 75 cities.

Parameter	**Type**	**Value**
Selection	method	tournament, size 2
	fitness normalisation	reverse scale
Replacement	method	uniform
	condition	if improved
	rate	1
	elitism	yes
Population	size	20
	generations	5000
Crossover	type	multiple
	rate	1 per chromosome
Mutation	type	swap
	rate	1 per chromosome
Chromosome	structure	x and y coordinate of city

Table 8.3: Standard parameters for the TSP problem.

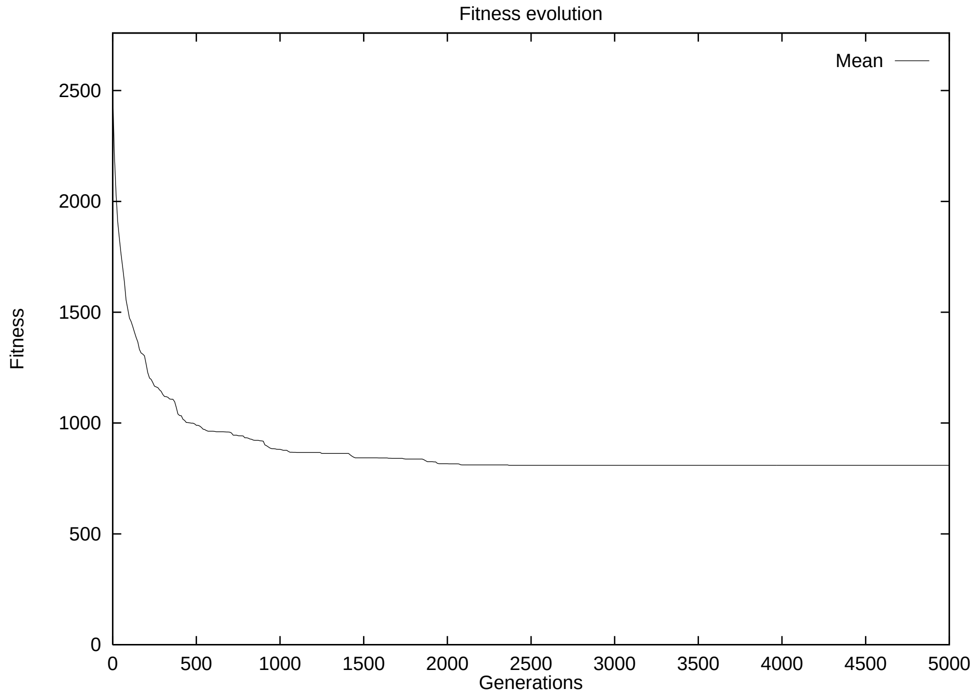

Figure 8.15: TSP with 75 cities, standard selection and other parameters as in table 8.3: Fitness evolution

In these parameters we have endeavoured to find the ones that give a solution that converges quickly; as we shall see, this is a matter of experimentation and tinkering with the settings of the genetic algorithm, since results vary widely with the settings. Also note that, whilst most tests have been performed up to the 5000th generation, the algorithm has been used with other values as well; these cases are noted in the text below.

8.3.2 Standard tests

The standard selection mechanism gives the following results; for the fitness evolution (see Fig. 8.15), and for the diversity evolution (see Fig. 8.16). In this base case the solution found had a length of 809 after 5000 generations.

As can be seen, the diversity rapidly declines to become zero at approximately the 2000th generation. Rather interestingly, this can be significantly enhanced using more frequent mutation. This case is exemplified by the following two corresponding figures for a setting of multiple mutations, namely the previous swap as well as an inversion operation. See Fig. 8.17 and 8.18, for the fitness and diversity plots, respectively.

It is clearly visible how the frequent mutation prevents diversity from ever becoming zero, and, consequently, the genetic algorithm may explore a larger space of possible solutions and reaches a better solution. Incidentally, the found solution had a length of 717.

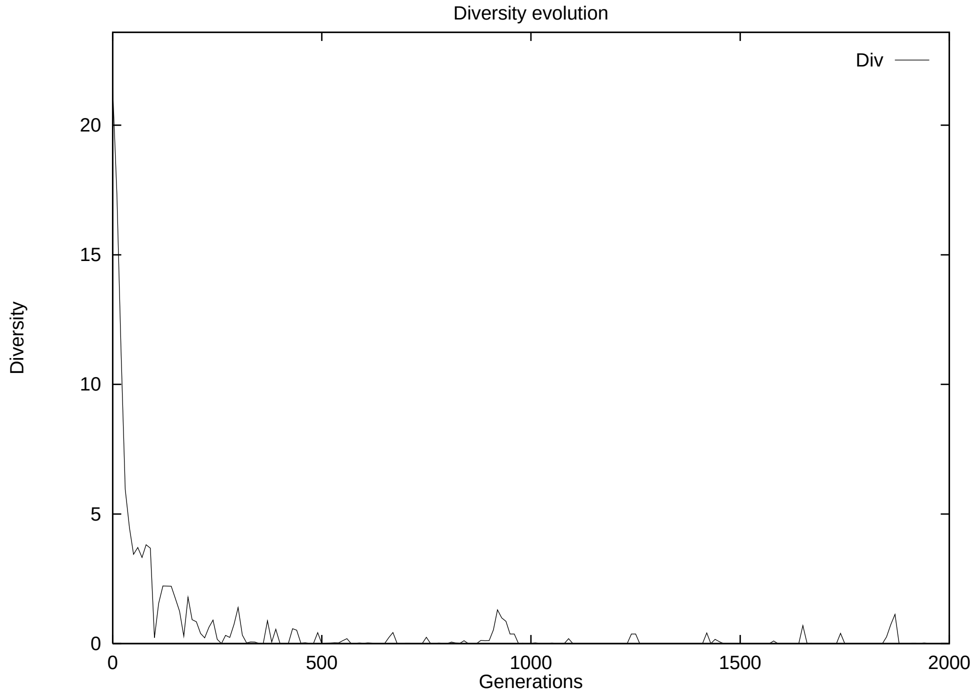

Figure 8.16: TSP with 75 cities. standard selection and other parameters as in table 8.3: Diversity evolution

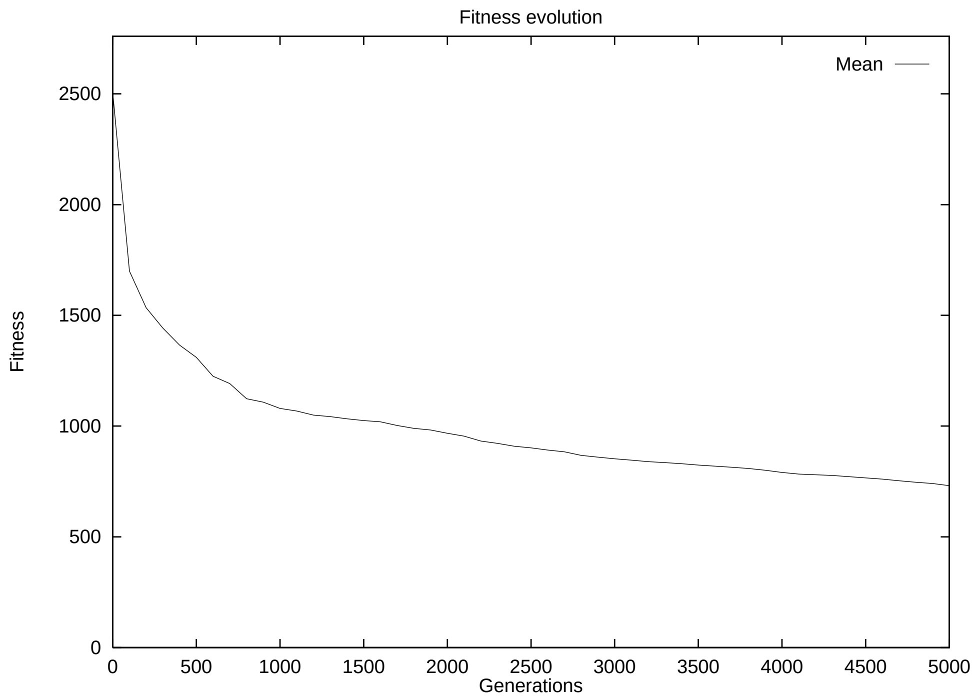

Figure 8.17: TSP with 75 cities, standard selection, parameters as in table 8.3 but with multiple mutation (inversion and swap): Fitness evolution.

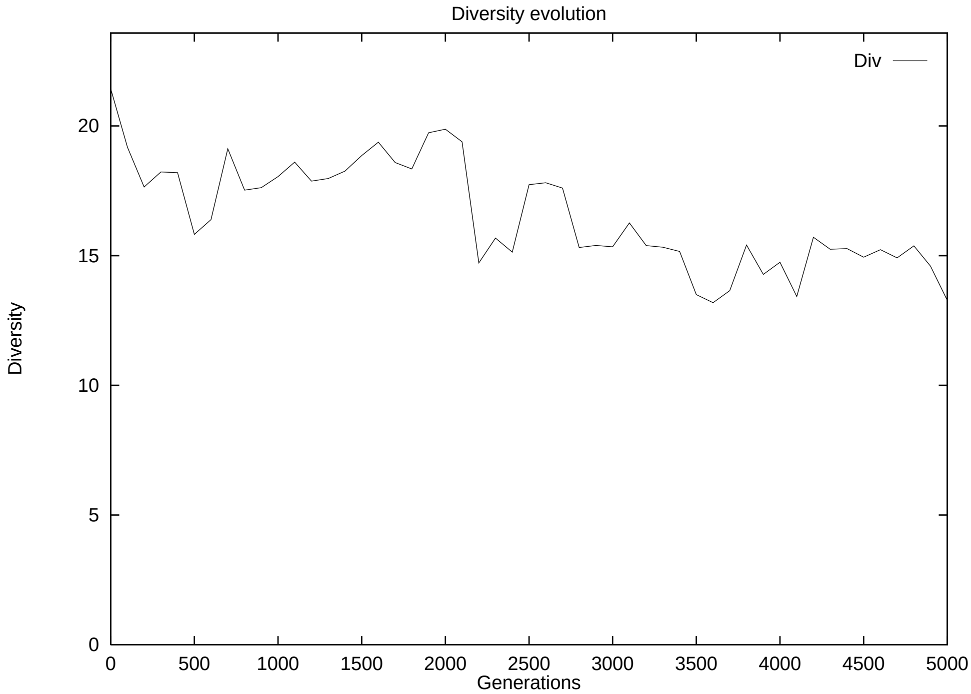

Figure 8.18: TSP with 75 cities, standard selection, as in table 8.3 but with multiple mutation (inversion and swap): Diversity evolution.

8.3.3 Temporal-conservative tests

Having studied the test results from the temporal-conservative selections tests, it must be said that this particular method of selection does not seem to offer any advantages over the more traditional methods, such as roulette wheel selection or tournament selection, which was employed in the standard tests discussed above.

The results can be studied in figure 8.19, for the fitness evolution case, and figure 8.20, for the diversity evolution. As before, the diversity is certainly preserved, but, since no new individuals are being generated and put into the population as a result, the solution is not reached any quicker than in the standard case.

Note that in this case we can also avail ourselves of the increased mutation rate offered by multiple mutations, like the enhanced standard case above. In this case, we obtain the results shown in figures 8.21 and 8.22, for the fitness and diversity, respectively.

Comparing the two sets of curves clearly shows the benefit of increased mutation on the convergence of the algorithm.

8.3.4 Temporal-progressive tests

Studying the results of the temporal-progressive selection tests, the conclusion is clear: this method of selection exhibits the best results of the methods discussed here, in the context of the SUGAL genetic algorithm.

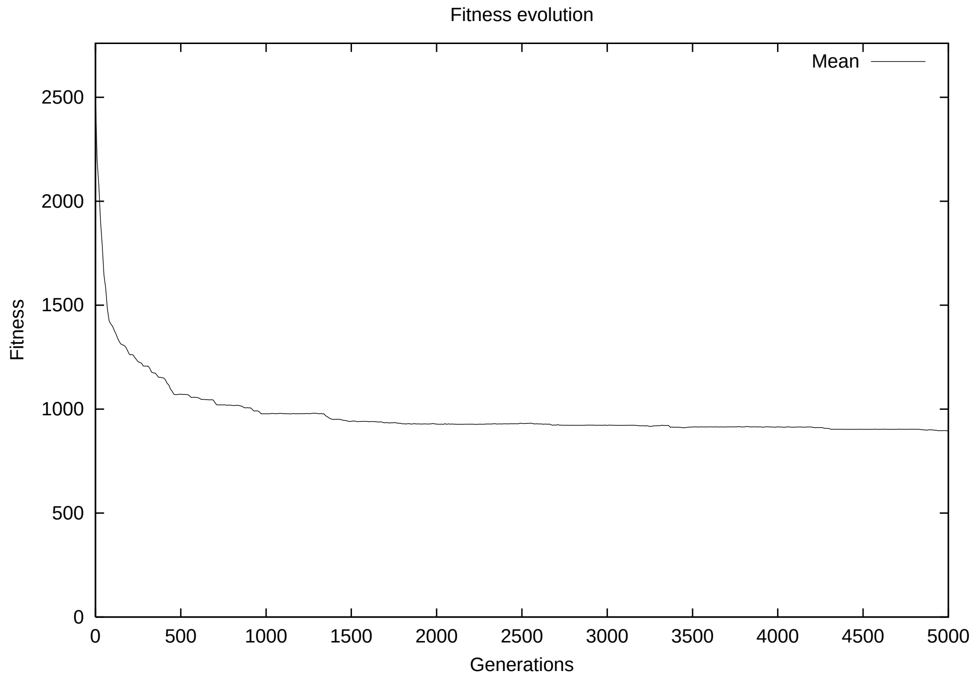

Figure 8.19: TSP with 75 cities, temporal-conservative selection and parameters as in table 8.3: Fitness evolution.

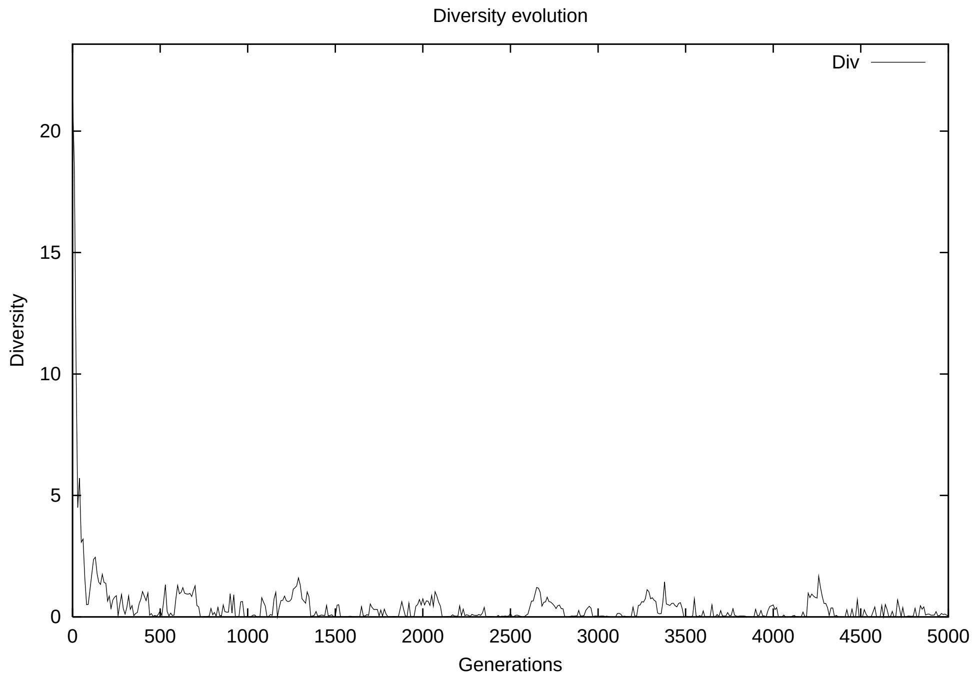

Figure 8.20: TSP with 75 cities, temporal-conservative selection and parameters as in table 8.3: Diversity evolution.

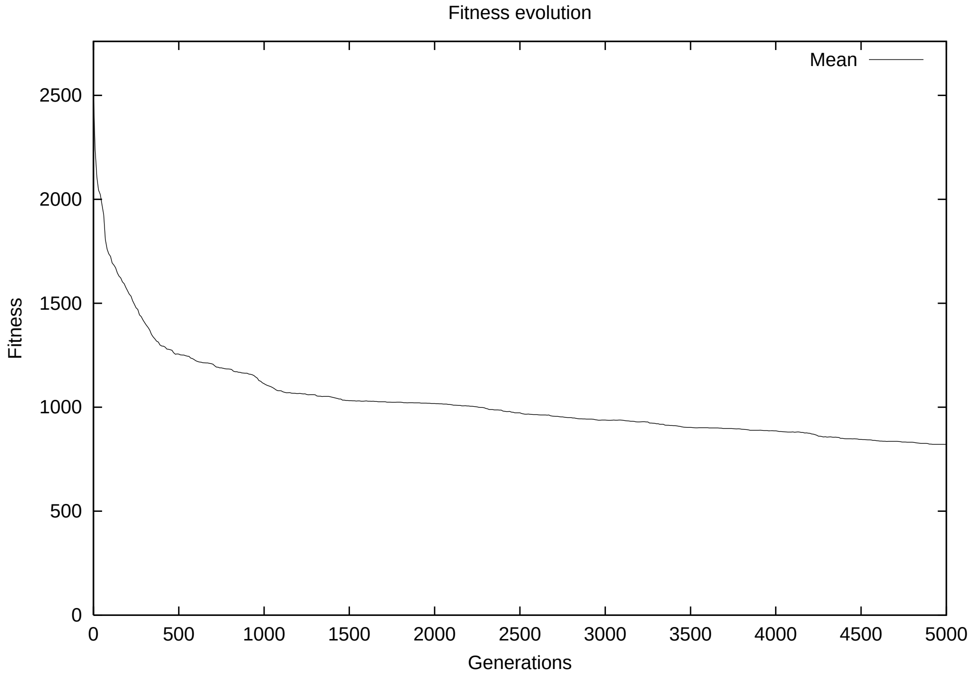

Figure 8.21: TSP with temporal-conservative selection, multiple mutation and parameters as in table 8.3: Fitness evolution

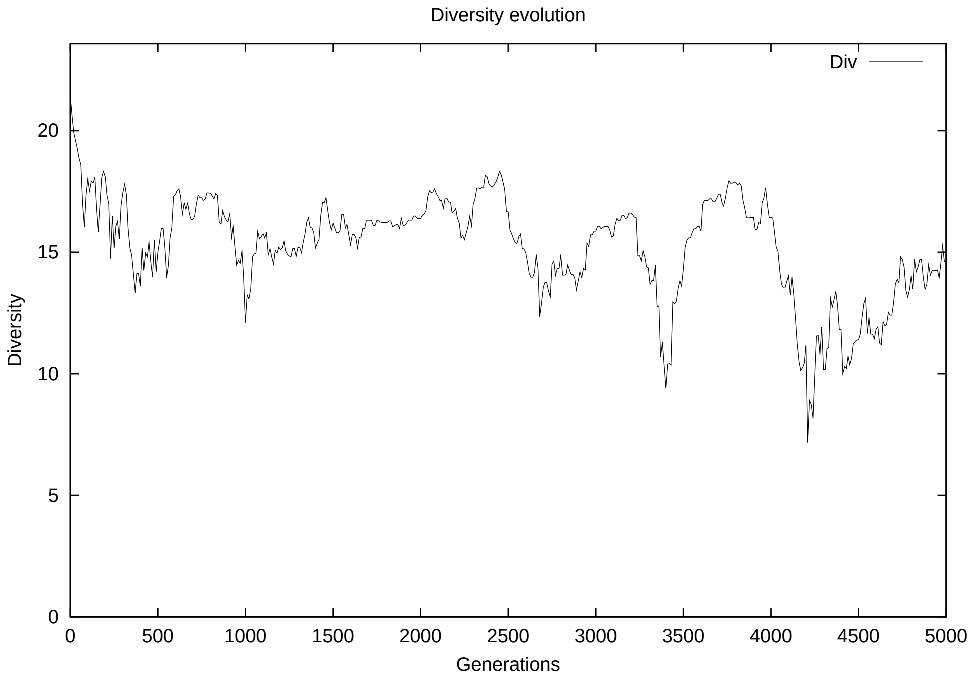

Figure 8.22: TSP with temporal-conservative selection, multiple mutation and parameters as in table 8.3: Diversity evolution

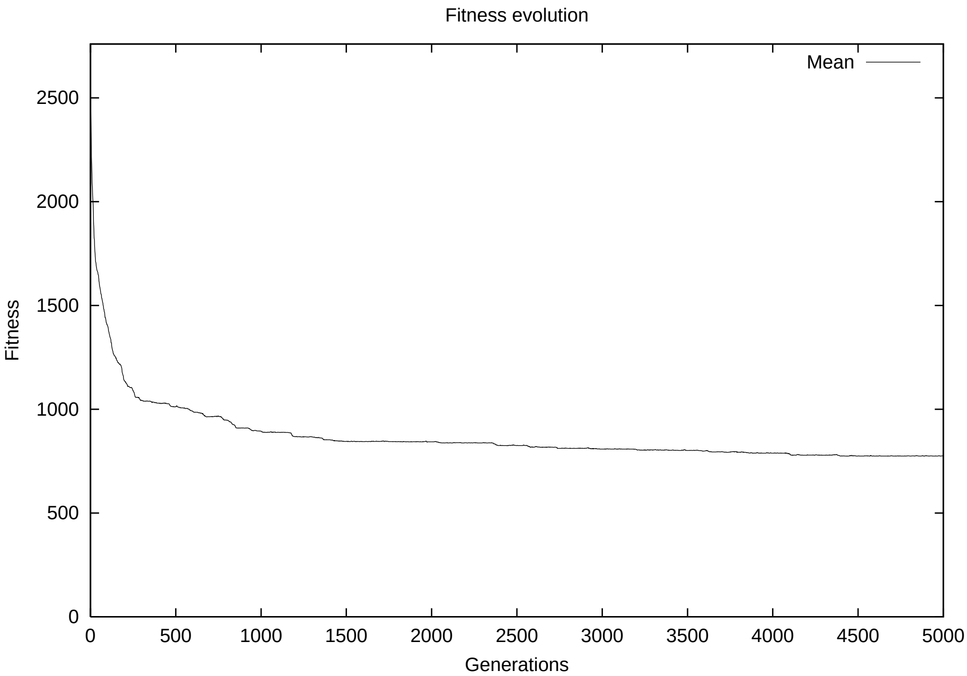

Figure 8.23: TSP with 75 cities, temporal-progressive selection and parameters as in table 8.3: Fitness evolution.

Not only that; the difference is remarkable: compared with the standard selection method algorithm we have gone from a length of 809 to one of 630 (approximately), an improvement of almost 23 %.

The results are summarised in figures 8.23 and 8.24, for the base temporal-progressive case (fitness and diversity, respectively), and for the enhanced mutation case, in figures 8.25 and 8.26, for fitness and diversity, respectively.

Like before, note that in this case we can also avail ourselves of the increased mutation rate offered by multiple mutations, like the enhanced standard case above. Again, in this case, we obtain the results shown in figures 8.25 and 8.26, for the fitness and diversity, respectively.

Again, comparing the two sets of curves clearly shows the benefit of increased mutation on the convergence of the algorithm.

8.3.5 Results

The best found result had a length of approximately 558, after 31,000 generations; see figure 8.27. This solution used temporal-progressive selection, the multiple mutations above, as well as tournament selection of parents. Elitism plays a small, but still positive role, and was turned on. The effect was of the order of 2 % (10 units). We also tried varying the equivalence interval, but a value of 10 seemed fairly optimal as results deteriorated with both larger and smaller values.

Compare this solution with the optimal one, found by the *concorde* program from the W. M. Keck Center for Computational Discrete Optimization at Rice University, Houston, Texas. That solution had a length of 535; see

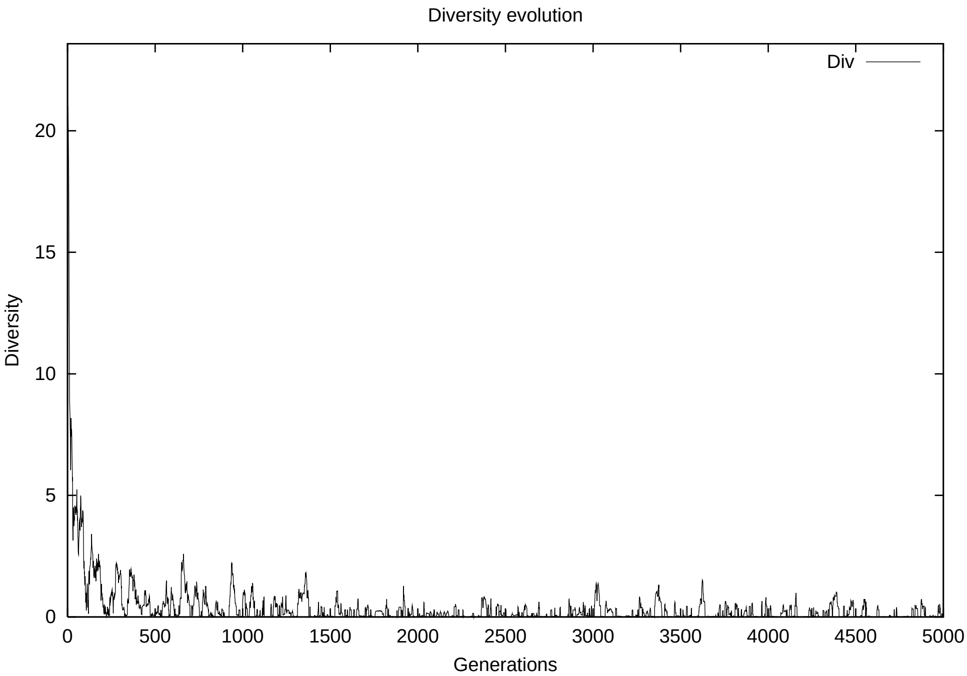

Figure 8.24: TSP with 75 cities, temporal-progressive selection and parameters as in table 8.3: Diversity evolution.

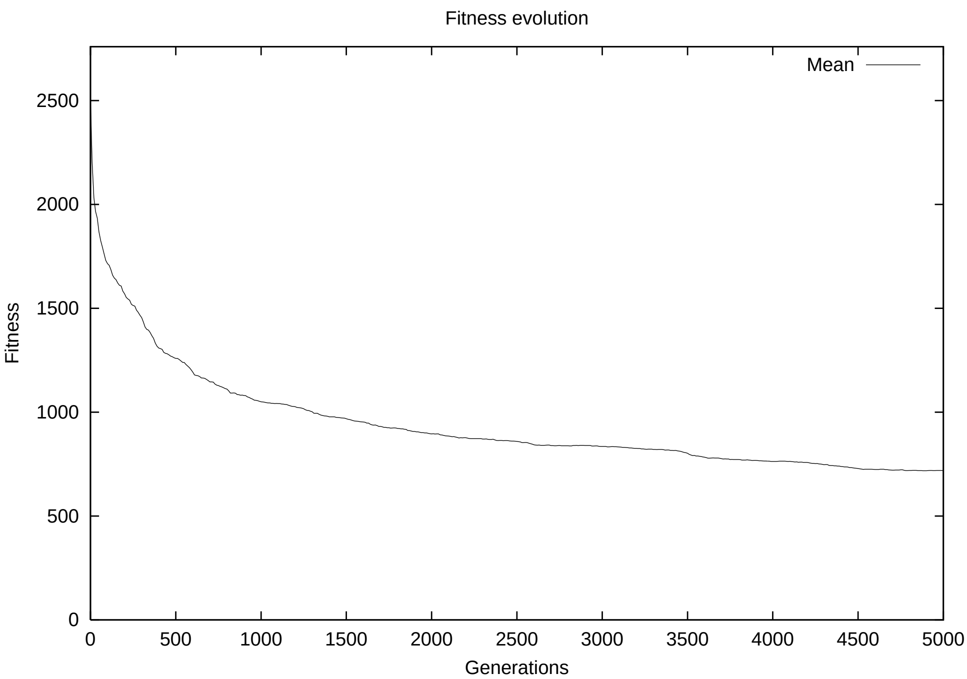

Figure 8.25: TSP with 75 cities, temporal-progressive selection, parameters as in table 8.3, but with multiple mutation (inversion and swap): Fitness evolution.

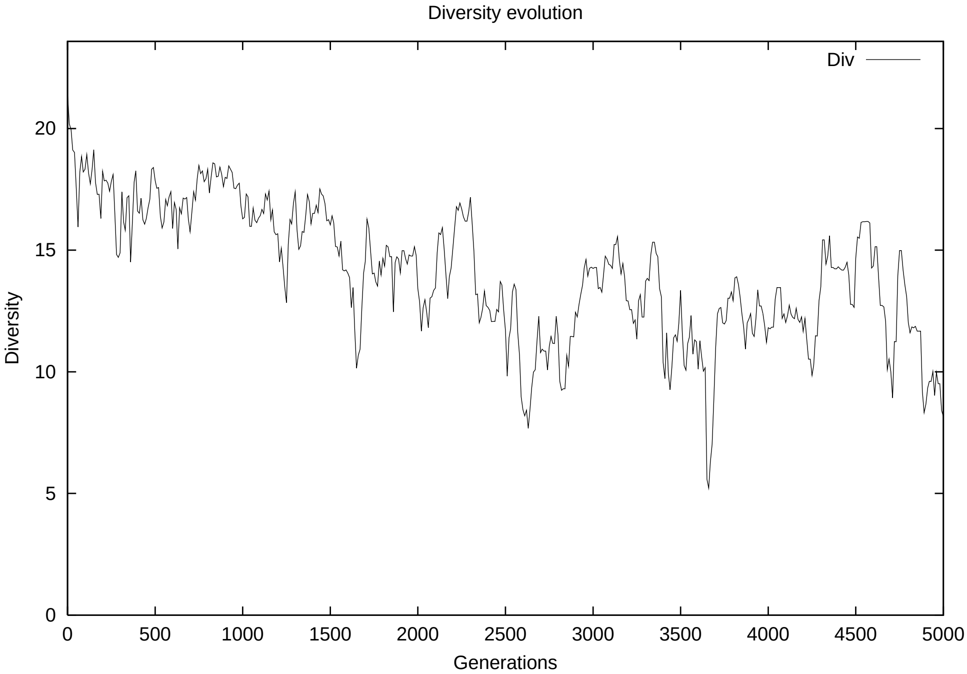

Figure 8.26: TSP with temporal-progressive selection, parameters as in table 8.3, but with multiple mutation (inversion and swap): Diversity evolution.

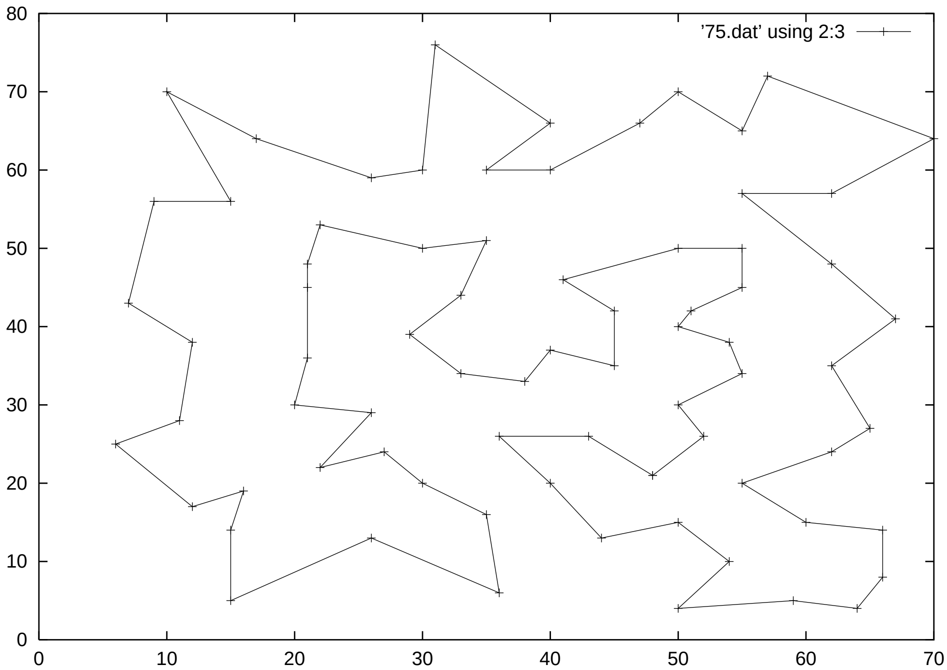

Figure 8.27: TSP with temporal-progressive selection, parameters as in table 8.3, but with multiple mutation (inversion and swap): The GA solution.

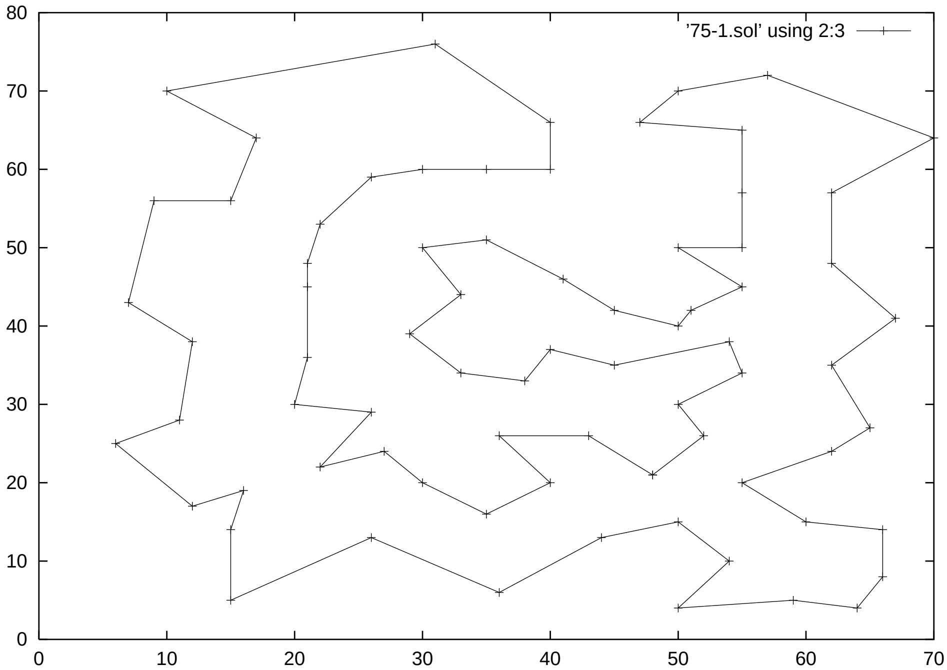

Figure 8.28: TSP problem with 75 cities: The optimal solution

figure 8.28. As can be readily seen, the solutions are fairly close to each other, and the result as found by the genetic algorithm is only about 4 % off, an impressive result by any standards for a non-optimised, general-purpose algorithm on this difficult problem.

The question naturally arises, why does temporal-progressive selection work so well? The answer, we believe, hinges of the fact that it both enhances diversity by preventing premature convergence, and that it makes the genetic algorithm explore new regions of the genetic landscape even if the fitness of the solution is good.

8.4 TEMPORAL SELECTION: A DISCUSSION

Temporal selection mechanisms seem to offer a viable alternative to traditional methods. Not because they supplant them - on the contrary, they cannot be used without a traditional method as a basis. The introduction of an equivalence interval together with temporal selection indicating where the selection should be modified on a temporal basis is a viable tactic to increase exploration of population and individual space, as well as ensuring that the premature convergence problem is avoided.

By examining, ideally in real-time, the actual number of replaced individuals it is fairly easy to determine when the genetic algorithm starts to converge. Even better, we should consider plotting the evolution of the virtual population (both the 1-generation and the cumulative one), although it may necessitate a modification to whatever implementation of the genetic algorithm we use in order to store the two numbers (sizes). There are a number of reasons for doing so:

- we can determine convergence, since the size of the virtual population steadily decreases, or $\lim |\Pi_v^1| = 0$ as we near convergence, depending on temporal selection and the size of the equivalence interval. In particular, note the phenomenon near the border of the equivalence interval: we won't really get nearer convergence than this.
- we can check the size of the cumulative virtual population against theory in order to determine accuracy.

Note that if the equivalence interval $\epsilon = 0$ we have a standard, non-temporal selection, regardless of other parameters. In other words, all temporal selections get arbitrarily close to their non-temporal counterparts as ϵ goes to zero.

In view of the tests above it is also clear that temporal selection is no panacea: in general we cannot in the light of these empirical investigations, even if somewhat simplistic, make the conclusion that it will make the genetic algorithm converge faster, or produce better individuals in the general case. Sometimes it certainly will do so (cf. F1 above), but not always (F4 above). However, its utility is in providing a valuable means for checking the progress of the genetic algorithm and to ensure a healthy balance between exploration and exploitation. If we decide to use temporal selection we have not in the experiments related above really seen any substantial differences in the utility of the two types, conservative and progressive, respectively, making it necessary to choose on a case by case basis which one to use. The real problem of the travelling salesperson proves to be an excellent case where temporal-progressive selection seems particularly beneficial.

9 CONCLUSIONS

This work treats the genetic algorithm from two unusual viewpoints: namely that of time, in the form of a formalisation of time based on temporal intervals, and, to a lesser extent, that of probabilistic logic in the form of statements the truth value of which are probabilistic, i.e. uncertain.

This novel formalisation in modelling the genetic algorithm necessitates a careful redefinition of all the concepts in normal genetic algorithms: genes, individuals, populations and so on, as well as requires a reformulation of the normal genetic algorithm process itself to take into account both the temporal and probabilistic aspects so central to the genetic algorithm.

Following two introductory chapters on the problems with genetic algorithms and a brief exposure to the standard formalisations of genetic algorithms (especially the well-known schema theorem due to Holland and Goldberg) we have included an introductory chapter on the foundations of the logics in order to lay the groundwork and make the understanding of the following chapter easier. That chapter contains one of the central themes of this work: the redefinition of the genetic algorithm in temporal terms.

This redefinition of the genetic algorithm introduces several interesting concepts: that of a virtual population, and that of virtual genetic operators. It is shown how these enhance the understanding of the temporal progression and workings of the genetic algorithm. One of the more interesting concepts is also the probabilistic fitness (definition 29): an individual's probabilistic fitness is its probability of occurrence in a population within any given interval:

$$\phi_\xi = p(\alpha_I) \tag{9.1}$$

where α_I denotes an agent with chromosome ξ within the interval I. We also emphasise the difference between phenotypic and genotypic fitness, and formulate the Central Assumption of the genetic algorithm: that these have a one-to-one relationship.

Regarding the virtual population we prove the following theorem (theorem 2): that for a 1-generation virtual member α_I of a population Π_I the following holds

$$\bigcirc alive(\alpha_I) \tag{9.2}$$

where $alive(\alpha_I) : \phi(\alpha_I) > 0$ is the function defined on page 39.

Having laid the foundations for discussions about the actual genetic algorithm itself, the second central topic of this work explains the genetic algorithm process both in terms of temporal concepts as well as in probabilistic terms. Some of the most important parameters in explaining and examining the genetic algorithm are reformulated using the temporal and probabilistic constructs defined earlier. It is shown how the newly formulated constructs may be used to examine some of the most important characteristics of the genetic algorithm: its convergence, the existence of optima and parameters for testing populations nearing convergence.

As a major contribution, we also show how a new form of selection mechanism may be defined using the temporal aspects of the genetic algorithm; the advantages and disadvantages as well as genetic algorithm characteristics in view of the new temporal selection mechanisms are treated in sufficient detail to ascertain their usefulness. The main temporal selection types are the conservative one, where the older individual survives (should they otherwise be considered equal), and the progressive one, where the newer individual survives to the next generation. The crucial concept of a fitness equivalence interval (or equality interval) underlying the temporal selection mechanisms is explained; individuals considered equal are collected together into equality intervals.

One of the main theorems relates the average fitness to the statistical probability (theorem 12): the average fitness $\phi_{avg}(\Pi_I)$ of a population Π_I in an interval I is equal to the statistical probability $[alive(\alpha_I)]_{\alpha_I}$. Formally,

$$\phi_{avg}(\Pi_I) = \frac{\sum_{i=1}^{|\Pi_I|} \phi(\alpha_{I,i})}{|\Pi_I|} = [alive(\alpha_I)]_{\alpha_I} \tag{9.3}$$

where we use $|\Pi_I|$ to denote the number of inviduals in the population Π_I in interval I.

Furthermore, it is shown that the genetic algorithm may fairly simply be axiomatised. The axiomatisation is based on the standard methodology employed when e.g. the axiomatisation of the theory of groups is conducted. One of the major results of this chapter is the easy derivation of several theorems earlier introduced; also, a new result is presented: it is shown that in the genetic algorithm all individuals end up in the same equality environment. Formally,

$$\diamond(a \cong b) \tag{9.4}$$

where $a \in \Pi_{I\bigcirc}$ and $b \in \Pi_I$. This chapter may be seen as application of the formalistic approach to the genetic algorithm.

As an interesting topic, we have also included a chapter on modelling the genetic algorithm using high-level Petri nets (specifically, predicate / transition nets, Pr-T nets). This should be seen as a simulation of the workings of the genetic algorithm, and represents an original contribution of this work. We are convinced that this simulation approach is but a first step to a more comprehensive exposition of the possibilities of a combination of the theory of genetic algorithm with that of Petri nets, especially in view of the parallel nature of both the genetic algorithm and the Petri net.

Finally, a concluding chapter contains experiments and discussions using an experimental setup with an implementation of the genetic algorithm that has been modified in order to test the preceding concepts in practice. It shows how the newly formulated selection mechanisms work on the standard benchmark problems, the so-called De Jong functions, and shows how some of the parameters may be determined in real genetic algorithms. It is shown how the temporal selection mechanisms may offer advantages in overcoming some of the traditional problems affecting genetic algorithms, chiefly that of premature convergence. This chapter also emphasises the importance of tailoring the genetic algorithm to the problem at hand by showing how the two types of temporal selection, progressive and conservative, are advantageous and disadvantageous on selected problems: no overall rule may be formulated to ascertain in advance which selection mechanism one should prefer. The temporal mechanisms are also compared with traditional mechanisms, and it is underlined how they work together using the equivalence interval. In particular, these mechanisms are shown to bear especial relevance to the well-known problem of the travelling salesperson (TSP), which, as shown in the concluding chapter seems to benefit from the temporal selection mechanism (and more so its temporal-progressive variant) than perhaps *a priori* one would have been led to believe.

It is clear that whilst the formalism in this work may be seen as a fairly detailed exposition of this approach to modelling the genetic algorithm with respect to time, in order to address real problems the model must incorporate less abstract features from real implementations of genetic algorithms.

We are convinced that the formalism outlined in this work, based on temporal probabilistic logic, axiomatisation, and Petri nets is pregnant with exciting possibilities and will prove to be a significant step forward in modelling the genetic algorithm.

A TEMPORAL PROBABILISTIC LOGIC (TPL)

This chapter formalises a combination of first order logic with the two probabilistic logics, namely propositional probabilistic logic and statistical probabilistic logic as well as temporal logic into a unified *temporal probabilistic logic* (which we shall abbreviate TPL).

The TPL described here is based on three logics: standard first order logic, temporal interval logic as outlined in Chapter 3 starting on page 15 and probabilistic logic as outlined by Bacchus [Bac90]. For a similar approach to combining probabilistic and temporal reasoning, see for instance Haddawy [Had94, Had96].

The chapter first introduces the formal syntax of TPL in a conventional way. Then the semantics is dealt with, followed by proof theory. We give some interesting theorems at various places in the text where we feel that it would illuminate the exposition; also, some are needed in the main text when we formalise the structures and workings of the genetic algorithm.

The objective is for the logic to be useful; in our case to enable us to analyse, discuss, and draw conclusions from interesting aspects of the workings of the genetic algorithm.

Note that we restate some definitions and axioms since we feel that we should make the chapter as self-contained as possible.

A.1 SYNTAX

The following presentation of TPL first defines the symbols of the logic. Then rules are given which specify the strings of symbols that form the well-formed formulas. In the main, this presentation closely follows [Bac90], apart from the addition of temporal constructs.

The letters n and m are used as meta-variables denoting natural numbers (i.e. $n, m \in \mathbb{N}$).

A.1.1 Symbols

We start with a set of function and predicate symbols.

a. For every $n \geq 0$ a set of n-ary function symbols $(f, g, h, \ldots)$. Constant symbols are 0-ary function symbols.

b. For every $n > 0$ a set of n-ary predicate symbols $(P, Q, R, \ldots)$.

The function and predicate symbols may be of three types: object symbols, time interval symbols and numeric symbols. The object symbols typically describe some domain of interest, whereas the temporal symbols designate time durations (periods).

We should also include the distinguished symbols of the logic itself, i.e. those symbols that are the same irrespective of the domain being described. The symbols are listed in Table A.1.

S1. The usual parentheses '(' and ')' used for grouping.

S2. The binary object predicate symbol $=_o$.

S3. The numeric constants -1, 0, and 1.

S4. The binary numeric predicate symbols $<_n$ and $=_n$.

S5. The binary numeric function symbols + and $\times$.

S6. The binary time interval predicate symbols $<_t$, $=_t$ and $\sqsubseteq$.

S7. The connectives $\wedge$ and $\neg$.

S8. The quantifier $\forall$.

S9. The temporal quantifiers $\bigcirc$, $\Box$ and $\Diamond$.

S10. The sentential probability operator **prob**.

S11. '[' and ']' surrounding a term together with a subscripted set of placeholder variables.

S12. The letter E used as an operator denoting the expectation value of a statistical probability

S13. A set of numeric variables, a set of time interval variables and a set of object variables.

S14. A set of user-defined function symbols.

S15. A set of user-defined predicate symbols.

Table A.1: Symbols in TPL

We restrict the number of symbols to be at most countably infinite. We also omit the subscript denoting the sort when no confusion exists; e.g. we write = instead of $=_o$, $=_n$, or $=_t$ and likewise for $<$ and $>$.

Note that the user-defined function symbols map object variables to numbers; they may be call *measuring functions* and are known in statistics as random variables. For example, we may wish to discuss the weight of various people using a **weight** function. For details, see [Bac90, p. 84f].

Finally, a **rigid** term is one that stays the same in all possible worlds; e.g. the numbers.

A.1.2 Formulas

The formulas of the language $\mathcal{L}^{TPL}$ are strings of symbols formed by the following recursive rules, given in table A.2. There are no other formulas.

F1. A single object variable or constant is an *o-term*.

F2. A single numeric variable or constant is an *f-term*.

F3. A single temporal variable or constant is a *t-term*.

F4. If f is an n-ary object function symbol and $t_1, \ldots, t_n$ are o-terms, then $ft_1, \ldots t_n$ is an *o-term*.

F5. If g is an n-ary temporal function symbol and $t_1, \ldots, t_n$ are t-terms, then $gt_1, \ldots t_n$ is a *t-term*.

F6. If $\mathbf{f}$ is an n-ary numeric function symbol and t_1,...,t_n are f-terms, then $\mathbf{f}t_1$,...,t_n is an *f-term*.

F7. If P is an n-ary object predicate symbol and $t_1, \ldots, t_n$ are o-terms, then $Pt_1, \ldots, t_n$ is a *formula*.

F8. If Q is an n-ary temporal predicate symbol and $t_1, \ldots, t_n$ are t-terms, then $Qt_1, \ldots, t_n$ is a *formula*.

F9. If $\mathbf{P}$ is an n-ary numeric predicate symbol and t_1,...,t_n are f-terms, then $\mathbf{P}t_1$,...,t_n is a *formula*.

F10. If α is a formula, then so is $\neg\alpha$.

F11. If α is a formula, so are $\bigcirc\alpha, \Box\alpha$ and $\Diamond\alpha$.

F12. If α and β are formulas, then so is $\alpha \wedge \beta$.

F13. If α is a formula and x is a variable of any of the three types, then $\forall x.\alpha$ is a $formula$.

F14. If α is a formula, then $\mathbf{prob}(\alpha)$ is an *f-term*.

F15. If α is a formula and $\vec{x}$ is a vector of n *object* variables, then $[\alpha]_{\vec{x}}$ is an *f-term*. These terms are called *statistical probability* terms.

F16. If t is a statistical probability term or rigid f-term, then $E(t)$ is an *f-term*.

F17. If α and β are formulas, then $\mathbf{prob}(\alpha|\beta)$ is an *f-term*. These terms are called *conditional probability* terms (for propositional probabilities).

F18. If α and β are formulas, then $[\alpha|\beta]_{\vec{x}}$ is an *f-term*. These terms are called *conditional probability* terms (for statistical probabilities).

Table A.2: Formulas in TPL

This definition of formulas is different from the standard first-order logic definition in that it allows for numeric and temporal terms to be formulated from already existent formulas. Note that the expectation operator E acts syntactically as a monadic function. Its semantic interpretation is different from ordinary functions, as we shall see below.

A.1.3 Definitions for numeric and standard extensions

As the intention is not to duplicate well-known work on standard first-order logic, suffice to say that the connectives $\vee$, $\rightarrow$, $\exists$ and $\equiv$ may be defined in the usual way. For instance, $\exists x.\alpha(x)$ is defined as $\neg\forall x.\neg\alpha(x)$.

We also write all numeric binary function symbols, like '+', '−', etc. in the more readable infix form, using standard conventions of precedence and parentheses as required to disambiguate whenever necessary.

Furthermore, we also write multiple quantified variables together, e.g. $\forall xy.\alpha$ will abbreviate $\forall x.\forall y.\alpha$.

It is also convenient to introduce the following abbreviations for numeric terms.

Definition 63 *The binary numeric inequality functions are defined as below.*

a. $x \leq y \;=_{df}\; (x < y) \vee (x = y)$

b. $x \geq y \;=_{df}\; \neg(x < y)$

c. $x > y \;=_{df}\; y < x$

d. $x \in [y, z] \;=_{df}\; y \leq x \wedge x \leq z$

We restate the following useful definition for certainty.

Definition 64 (Certainty) *Certainty, or an abbreviation for probability one, is defined as follows.*

$$\mathbf{cert}(\alpha) \;=_{df}\; \mathbf{prob}(\alpha) = 1 \tag{A.1}$$

The conditional probabilities alluded to in Table A.2 are defined in almost the usual way as follows. These are adapted from Bayes' theorem, (see e.g. [Pap65]).

Definition 65 (Axiom of propositional conditional probabilities)

$$\begin{aligned} \mathbf{prob}(\beta) \neq 0 \;\; &\rightarrow \mathbf{prob}(\alpha|\beta) \times \mathbf{prob}(\beta) = \mathbf{prob}(\alpha \wedge \beta) \\ \mathbf{prob}(\beta) = 0 \;\; &\rightarrow \mathbf{prob}(\alpha|\beta) = 0 \end{aligned} \tag{A.2}$$

Definition 66 (Axiom of statistical conditional probabilities)

$$\begin{aligned} [\beta]_{\vec{x}} \neq 0 \;\; &\rightarrow [\alpha|\beta]_{\vec{x}} \times [\beta]_{\vec{x}} = [\alpha \wedge \beta]_{\vec{x}} \\ [\beta]_{\vec{x}} = 0 \;\; &\rightarrow [\alpha|\beta]_{\vec{x}} = 0 \end{aligned} \tag{A.3}$$

Note that these definitions explicitly defines the case where the conditioning formula has probability zero, contrary to normal practice that leaves it undefined.

Note that these are later restated as two of the probability axioms of TPL.

A.2 SEMANTICS

In order to interpret our formulas in the language we need to define appropriate structures for them, first for temporal and then for probabilistic formulas. We then turn our attention to the interpretation of the formulas.

Note that we repeat some of the definitions for temporal logic from the earlier chapter in the interest of clarity and completeness.

A.2.1 Temporal structures and definitions

The temporal side of TPL is more complex, and needs to be defined carefully from the ground up. We rely on van Benthem [Ben83].

First we define an interval.

Definition 67 (Interval) *An interval is a time duration between two time points, the start and end time, respectively. It is defined as follows*

$$I =_{df} [m_1, m_2] =_{df} \{m \in \mathbb{Z} \mid m_1 \leq m \leq m_2\} \tag{A.4}$$

where $m_1, m_2 \in \mathbb{Z}$ *and* $m_1 < m_2$.

Note that this definition disallows 'null' intervals of the form $[m, m]$.

We also need to differentiate between intervals that may contain subintervals and those that may not.

Definition 68 (Base interval) *An interval* $I = [m_1, m_2]$ *is a base interval iff the following holds:*

$$I = [m, m + 1] \tag{A.5}$$

where $m = m_1$, *and thus* $m_2 = m_1 + 1$.

van Benthem calls these atomic, or minimal, and does not deal with them further. All other intervals may be called *composite*. It should also be noted that intervals are normally *convex*, i.e. uninterrupted, as pointed out by van Benthem, *op. cit.* p. 68. We omit the specifier base or composite whenever the distinction does not matter. The definition above of intervals is similar to Schwartz, *et.al.* [SMSV83], Shoham [Sho88] and Sandewall [San94a].

We denote the set of all intervals $\mathcal{I}$, i.e. $I \in \mathcal{I}$.

The concept of time is informally defined using interval structures. We choose the following general structure.

Definition 69 *An interval or period structure is the triple $\langle \mathcal{I}, \sqsubseteq, < \rangle$ of a non-empty set I carrying two binary relations $\sqsubseteq$ 'inclusion' and $<$ 'precedence'.*

In addition, we can define the basic operation 'inclusion' between two intervals.

Definition 70 (Inclusion) *Given two intervals $I_1 = [m_1, m_2]$ and $I_2 = [m_3, m_4]$ we say that I_1 includes I_2 when the starting point m_2 is not less than that of m_1 and at the same time the end point of m_2 is not greater than that of m_1. Denoting inclusion by the symbol $\sqsubseteq$ we have formally*

$$I_2 \sqsubseteq I_1 \text{ iff } m_1 \leq m_3 \ \wedge \ m_2 \geq m_4. \tag{A.6}$$

As pointed out in the introductory chapter, we follow van Benthem strictly here; in our setting we could simply equate $\sqsubseteq$ with $\subseteq$, and thus dispense with this definition altogether, and rely on definition 4. We choose to follow van Benthem's usage here; the reader may treat $\sqsubseteq$ as synonymous for $\subseteq$.

We then define the interval, or period structure INT($\mathbb{Z}$) we are actually going to use based on the integers as follows.

Definition 71 (Interval structure) *An interval structure INT($\mathbb{Z}$) is the tuple*

$$\langle \mathcal{I}, \subseteq, < \rangle \tag{A.7}$$

where

- $\mathcal{I}$ *consists of all non-empty closed integer intervals* $[m_1, m_2]$
- $\subseteq$ *is set-theoretic inclusion*
- $<$ *is defined by setting* $[m_1, m_2] < [m_3, m_4]$ *if* $m_2 \leq m_3$.

Note that we, in line with the above definition, intend that

$$[m_1, m_2] > [m_3, m_4] \quad \text{if} \quad m_2 > m_3. \tag{A.8}$$

The intervals are of course equal if $m_1 = m_3$ and $m_2 = m_4$, and the $\leq$ and $\geq$ cases are defined accordingly.

Also note that in the definition above the internal structure of time is not specified: it may be linear or branching. However, the view of time we use in this work is one modelled on the integers, $\mathbb{Z}$, and is linear.

Finally, in order to simplify the axioms below we restate the *overlap* relations

Definition 72 (Overlap)

$$zOy \ =_{df} \ \exists u.u \sqsubseteq z \wedge u \sqsubseteq y \tag{A.9}$$

For convenience, we also define a temporal relation giving the next base interval.

Definition 73 (Next interval) *The next base interval y from an interval x is the one that satisfies*

$$\begin{aligned} x &< y \\ \nexists & \; z.x < z < y \end{aligned} \tag{A.10}$$

where z is an interval.

Analogously, we define the *previous* interval as follows.

Definition 74 (Previous interval) *The previous base interval y from an interval x is the one that satisfies*

$$\begin{aligned} y &< x \\ \nexists & \; z.y < z < x \end{aligned} \tag{A.11}$$

where z is an interval.

For temporal objects x we use the syntax $x^{\bigcirc}$ for the next (base) interval relative to x and $x^{\overline{\bigcirc}}$ for the previous. For predicates α, we use $\bigcirc\alpha$ and $\overline{\bigcirc}(\alpha)$, respectively. We can also readily extend this syntax for several intervals in the future (or past) using powers; e.g. for n in the future we could write $\bigcirc^n(\alpha)$ instead of the clumsier $\bigcirc \bigcirc \ldots \bigcirc$, where $\bigcirc$ is repeated n times.

A.2.2 Possible worlds

Below, we assign probabilities to distributions over a field of sets of interpretations or truth valuations of the language. These different interpretations can be viewed as being different possible worlds, where each possible world assigns a truth value to all of the formulas of the language, and the set of formulas which are assigned **true** varies from world to world. If a formula is assigned **true** at a particular possible world, we say that it *satisfies* the formula.

With a probability distribution over a field of sets of possible worlds we can assign a probability to every formula of the language, as we shall see. The probability of a formula becomes the probability of the set of possible worlds that satisfy the formula.

For more details, see Bacchus [Bac90, p. 33ff].

A.2.3 Temporal probability structure

Combining the preceding temporal structure with Bacchus [Bac90] we define the TPL structure as follows.

Definition 75 (Temporal probability structure) *The temporal probability structure M is defined by the tuple*

$$\langle \mathcal{O}, S, \vartheta, \mu_S, \mu_{\mathcal{O}}, INT(\mathbb{Z}) \rangle, \tag{A.12}$$

where

- $\mathcal{O}$ *is a set of individuals representing objects that one wishes to describe in the logic. $\mathcal{O}$ corresponds to the domain of discourse in ordinary first-order logic.*

- S *is a set of states or possible worlds.*

- ϑ *is a function that associates an interpretation of the language with each world. For every $s \in S, \vartheta(s)$ is an interpretation that assigns to every n-ary object predicate a subset of $\mathcal{O}^n$. It also assigns to every n-ary object function symbol function a function from $\mathcal{O}^n$ to $\mathcal{O}$, as well as to every n-ary measuring function symbol a function from $\mathcal{O}^n$ to $\mathbb{R}$, to every n-ary numeric predicate a subset of $\mathbb{R}^n$, and to every n-ary numeric function a function from $\mathbb{R}^n$ to $\mathbb{R}$. In addition, it assigns to every n-ary temporal function a function from $INT(\mathbb{Z})$ to $INT(\mathbb{Z})$, as well as for all quantifier predicates, whether first-order logic or temporal, a truth value $true$ or $false$. For example, it maps the 0-ary object function symbols, the constants, to particular individuals in $\mathcal{O}$. We do not require that the object function symbols be rigid, with the exception of the distinguished symbols (e.g. +, $-$, $\times$ 1, -1, and 0), i.e. on these symbols $\vartheta(s)$ is independent of the world s. Also, the predicate symbols $<$ and $=$ are rigid. Of course, all logic symbols (e.g. $\vee$, $\wedge$ and $\neg$ and derivatives) are rigid as well.*

- μ_S *is a discrete probability function on S. In other words, μ is a function that maps the elements of S to the real interval [0,1] such that $\sum_{s \in S} \mu_S(s) = 1$. This function defines a probability distribution over the subsets of S in the following way: for every $A \subseteq S$ we define $\mu_S(A) = \sum_{s\in} \mu_S(s)$. With a discrete probability function every subset of S will have a probability, but also all except a denumerable number of worlds in S will have zero probability.*

- $\mu_{\mathcal{O}}$ *is a discrete probability function over $\mathcal{O}$, i.e. for $A \subseteq \mathcal{O}$, $\mu_{\mathcal{O}}(A) = \sum_{o \in A} \mu_{\mathcal{O}}(o)$ and $\mu_{\mathcal{O}}(\mathcal{O}) = 1$. In other words, $\mu_{\mathcal{O}}$ is similar to μ_S above.*

- $INT(\mathbb{Z})$ *is the temporal structure from the preceding section.*

Using this definition of the TPL temporal structure, we may formulate the interpretations of the formulas in TPL using the structure and its components.

A.2.4 Interpretation of the formulas

The truth value assigned to a formula in $\mathcal{L}^{TPL}$ will depend on three items: the semantic structure or model M, which determines the probability distributions μ_S and $\mu_{\mathcal{O}}$, the interpretation function ϑ, and the domain of objects $\mathcal{O}$; the current world $s \in S$ where S is the set of all possible worlds; and the variable assignment function v (which maps variables to individual objects) as described in the previous section.

Inductively, we assign a truth value **true** to α if $M \models \alpha$. We write $t^{(M,s,v)}$ for the individual denoted by the term t in the triple.

Table A.3 contains all the basic interpretations we allow in $\mathcal{L}^{TPL}$. This is a fairly standard interpretation of formulas with the possible exceptions of **I10** and **I12** giving an interpretation for the probability and expectation operators, respectively, as well as all of the temporal interpretations **I13–I15**.

It should be noted that, although the table is based on Bacchus [Bac90], it is carefully extended and consolidated from the various tables used by Bacchus quite extensively to take into consideration the temporal objects and operators and their ramifications.

I1. If x is a variable of any type, then $x^{(M,s,v)} = v(x)$. Variable assignment determines interpretation of variables independently of s.

I2. If f is an n-ary function symbol of any type, and $t_1, \ldots, t_n$ are terms of the same type, then

$$f(t_1, \ldots, t_n)^{(M,s,v)} = f^{\vartheta(s)}(t_1^{(M,s,v)}, \ldots, t_n^{(M,s,v)})$$

I3. If P is an n-ary predicate symbol of any type, and $t_1, \ldots, t_n$ are terms of the same type, then

$$(M, s, v) \models P(t_1, \ldots, t_n) \text{ iff } \langle t_1^{(M,s,v)}, \ldots, t_n^{(M,s,v)} \rangle \in P^{\vartheta(s)}$$

In addition, if P is a numeric predicate symbol, then $P^{\vartheta(s)} = P^{\vartheta(s\prime)}$ for all $s, s\prime \in S$. That is, the numeric predicates are rigid.

I4. If t and t' are terms of the same type, then

$$(M, s, v) \models (t = t') \text{ iff } t^{(M,s,v)} = t'^{(M,s,v)}$$

I5. For every formula α

$$(M, s, v) \models (\neg\alpha) \text{ iff } (M, s, v) \not\models \alpha$$

I6. For every pair of formulas α and β

$$(M, s, v) \models (\alpha \wedge \beta) \text{ iff } (M, s, v) \models \alpha \text{ and } (M, s, v) \models \beta$$

I7. For every formula α and object variable x

$$(M, s, v) \models \forall x.\alpha \text{ iff } (M, s, v[x/o]) \models \alpha \text{ for all } o \in \mathcal{O},$$

where $v[x/o]$ is the variable assignment function identical to v except that it maps the variable x to the individual o.

I8. For every formula α and numeric variable x

$$(M, s, v) \models \forall x.\alpha \text{ iff } (M, s, v[x/r]) \models \alpha \text{ for all } r \in \mathbb{R},$$

where $v[x/r]$ is the variable assignment function identical to v except that it maps the variable x to the real number r.

I9. For every formula α and temporal variable x

$$(M, s, v) \models \forall x.\alpha \text{ iff } (M, s, v[x/t]) \models \alpha \text{ for all } t \in INT(\mathbb{Z}),$$

where $v[x/r]$ is the variable assignment function identical to v except that it maps the variable x to the temporal interval t.

I10. For every formula α, the f-term created by the probability operator $\mathbf{prob}(\alpha)$ is given the interpretation

$$(\mathbf{prob}(\alpha))^{(M,s,v)} = \mu_S\{s' \in S : (M, s', v) \models \alpha\}$$

I11. For every formula α, the f-term created by the statistical probability operator $[\alpha]_{\vec{x}}$ is given the interpretation

$$([\alpha]_{\vec{x}})^{(M,s,v)} = \mu^n_{\mathcal{O}}\{\vec{a} : (M, s, v[\vec{x}/\vec{a}]) \models \alpha\},$$

where $\vec{x}$ and $\vec{a}$ are n-ary vectors of object variables and individual objects, respectively.

I12. Every f-term $E(t)$ created by the expectation operator E is given the interpretation

$$E(t)^{(M,s,v)} = \sum_{s' \in S} \mu_S(s') \times t^{(M,s',v)}$$

i.e., the weighted average of the operand across the different possible worlds.

I13. For every formula α and temporal objects x and y (i.e. intervals)

$$(M, s, v) \models \Box\alpha \text{ iff } \exists s' \in S \text{ so that } (M, s', v) \models \alpha \text{ for } \forall y^{(M,s',v)} \geq x^{(M,s',v)},$$

where $x^{(M,s',v)}$ fixes α to a world where it is valid in x.

I14. For every formula α and temporal objects x and y

$$(M, s, v) \models \Diamond\alpha \text{ iff } \exists s' \in S \text{ so that } (M, s', v) \models \alpha \text{ for } \exists y^{(M,s',v)} \geq x^{(M,s',v)},$$

where $x^{(M,s',v)}$ fixes α to a world where it is valid in x

I15. For every formula α and temporal object x

$$(M, s, v) \models \bigcirc\alpha \text{ iff } \exists s' \in S \text{ so that } (M, s', v) \models \alpha \text{ for } x^{\bigcirc,(M,s',v)},$$

where $x^{\bigcirc}$ is the next interval from x and the rest of the superscript fixes the object in the correct world s'.

Table A.3: Interpretations in TPL

Considering the interpretations in table A.3, we can formulate the following interesting theorem connecting the, so to say, probabilistic and temporal sides.

Theorem 18 *Given a formula α and $M \models \alpha$ in TPL, the following holds*

$$\models \Diamond\alpha \Rightarrow \models \mathbf{prob}(\alpha) > 0. \tag{A.13}$$

Proof. Recalling the right-hand side of the formula defining the interpretation of the probability operator **prob** (see **I10** in table A.3), reproduced below,

$$\mu_S\{s' \in S : (M, s', v) \models \alpha\} \tag{A.14}$$

the subset of worlds from S modelling α will then form (with the help of μ_S) a probability distribution that will give **prob** its value.

Now, comparing this to the definition of eventuality, **I14** in table A.3

$$\exists s' \in S \text{ so that } (M, s', v) \models \alpha \text{ for } \exists y^{(M,s',v)} \geq x^{(M,s',v)}, \tag{A.15}$$

where x and y are temporal objects, we notice that when y (recall that y is a *temporal* object, so *when* is the correct word), then we have, in fact, found a world (or set of worlds) where M is modelling α, and where **prob** will have a value, and thus be valid. □

Theorem 19 *Given a formula α and $M \models \alpha$ in TPL, the following holds*

$$\Diamond\alpha \Rightarrow \mathbf{prob}(\alpha) > 0 \tag{A.16}$$

$$\neg\Diamond\alpha \Rightarrow \mathbf{prob}(\alpha) = 0 \tag{A.17}$$

Proof. Since we have a world (or set of worlds) M modelling α, then **prob** will have a value according to theorem 18. If $\Diamond\alpha$ is $true$ that subset obviously is nonempty, **prob** must have a nonzero value as demanded. Conversely, when it is empty, no worlds exist and $\Diamond\alpha$ is **false** and the value of **prob**(α) is zero. □

Looking at the situation from an intuitive perspective, it comes as no surprise that $\Diamond\alpha \Rightarrow \mathbf{prob}(\alpha) > 0$ – if something is eventually true, it is clearly true in a possible world; then of course **prob** will be nonzero according to its definition see **I10** in table A.3.

We have the following straightforward corollary theorem.

Theorem 20 *Given a formula α and $M \models \alpha$ in TPL, the following holds*

$$\Box\alpha \Rightarrow \mathbf{cert}(\alpha). \tag{A.18}$$

Proof. Since $\Box\alpha$ means (see **I13** in table A.3) that α is true in all possible worlds, clearly it is valid in all possible worlds, and recalling the interpretation of **prob** (see **I10** in table A.3), then $\mathbf{prob}(\alpha) = 1$, which according to definition 64 is equal to **cert**(α). □

An interesting question is, can theorem 18 be reversed? In other words, what can be said about $\models \Diamond\alpha \Leftrightarrow \models \mathbf{prob}(\alpha)$? It turns out that this is possible under some stringent restrictions.

Theorem 21 *Assuming the non-rigid objects in $\mathcal{O}$ defining $s \in S$ (the current world set) are all temporal then the following holds*

$$\models \Diamond\alpha \Leftrightarrow \models \mathbf{prob}(\alpha) > 0. \tag{A.19}$$

Proof. The implication $\Rightarrow$ case is already covered by theorem 18, so it remains to show that $\Leftarrow$ holds. This can be seen by considering the definitions of the two cases, **I10** and **I14** in table A.3. Since the assumption is that all non-temporal objects are rigid, i.e. the same in all possible worlds, this means that we only have temporal objects that may change. Then clearly the implication stands. □

As an immediate consequence we can state the following.

Theorem 22 *Given a formula α and $M \models \alpha$ in TPL, and assuming the non-rigid objects in $\mathcal{O}$ defining $s \in S$ (the current world set) are all temporal the following holds*

$$\begin{aligned} \Diamond\alpha &\Leftrightarrow \mathbf{prob}(\alpha) > 0 && \text{(A.20)} \\ \neg\Diamond\alpha &\Leftrightarrow \mathbf{prob}(\alpha) = 0 && \text{(A.21)} \\ \Box\alpha &\Leftrightarrow \mathbf{cert}(\alpha). && \text{(A.22)} \end{aligned}$$

Proof. An immediate consequence of theorems 19, 18, and 21 and their proofs. □

A.3 PROOF THEORY

In order to examine proof-theoretic properties of TPL we must present an axiom system for all formulas. We do this separately for the different components of the logic.

A.3.1 First-order logic axioms

Here we may use any axiomatisation of first-order logic with equality as a basis [Bac90]. In the following list α, β and γ are formulas. The axioms are listed in table A.4.

PC1. $\alpha \rightarrow \beta \rightarrow \alpha$

PC2. $(\alpha \rightarrow \beta \rightarrow \gamma) \rightarrow (\alpha \rightarrow \beta) \rightarrow \alpha \rightarrow \gamma$

PC3. $(\neg\alpha \rightarrow \beta) \rightarrow (\neg\alpha \rightarrow \neg\beta) \rightarrow \alpha$

PC4. $\forall x.(\alpha \rightarrow \beta) \rightarrow (\forall x.\alpha \rightarrow \forall x.\beta)$

PC5. $\exists y.\mathbf{prob}(y = t) = 1 \rightarrow (\forall x.\alpha \rightarrow \alpha(x/t))$, where t is any term of the same type as x, free for x in α, and $\alpha(x/t)$ is the result of replacing all free occurrences of x in α by t

RGV $\exists y.\mathbf{prob}(y = t) = 1$ for every rigid term t. Among these are e.g. -1, 0, 1, i.e. numeric variables, as well as object and temporal variables

EQ1. $t = t$, where t is any term

EQ2. $t_1 = t_{n+1} \to \ldots \to t_n = t_{2n} \to ft_1 \ldots t_n = ft_{n+1} \ldots t_{2n}$, where f is any n-ary function symbol, and $t_1, \ldots, t_{2n}$ are terms of compatible type

EQ3. $t_1 = t_{n+1} \to \ldots \to t_n = t_{2n} \to Pt_1 \ldots t_n = Pt_{n+1} \ldots t_{2n}$, where P is any n-ary predicate symbol, and $t_1, \ldots, t_{2n}$ are terms of the same type

Table A.4: First-order axioms in TPL

The first three are axioms that generate all propositional tautologies, and the last three are axioms for reasoning with equality.

A.3.2 Numeric axioms

Following Bacchus [Bac90] we do not attempt to capture the full behaviour of these real-valued terms; instead we use the axioms of a totally ordered field to capture a large part of it.

Here all variables are numeric and universally quantified, unless the existential quantifier is explicitly used. The list of numeric axioms is in table A.5.

N1. $x + (y + z) = (x + y) + z$

N2. $x + 0 = x$

N3. $x + (-1 \times x) = 0$

N4. $x + y = y + x$

N5. $x \times (y \times z) = (x \times y) \times z$

N6. $x \times 1 = x$

N7. $x \neq 0 \to \exists y.(y \times x = 1)$

N8. $x \times y = y \times x$

N9. $x \times (y + z) = (x \times y) + (x \times z)$

N10. $\neg(1 = 0)$

N11. $\neg(x < x)$

N12. $x < y \to (y < z \to x < z)$

N13. $x < y \vee x = y \vee y < x$

N14. $x < y \to x + z < y + z$

N15. $0 < x \to (0 < y \to 0 < x \times y)$

Table A.5: Numeric axioms in TPL

A.3.3 Probability axioms

The probability axioms from [Bac90] are summarised in below, in table A.6. They come in three groups: propositional, statistical, and combined probability axioms.

P1. $\mathbf{prob}(\alpha) \geq 0$

P2. $\mathbf{prob}(\alpha) + \mathbf{prob}(\neg\alpha) = 1$

P3. $\mathbf{prob}(\alpha \wedge \beta) + \mathbf{prob}(\alpha \wedge \neg\beta) = \mathbf{prob}(\alpha)$

P4. $\alpha \to \mathbf{prob}(\alpha) = 1$ if all non-rigid function and predicate symbols in α occur inside of a propositional probability operator

P5. $\forall x.\mathbf{prob}(\alpha) = 1 \to \mathbf{prob}(\forall x.\alpha) = 1$

P6. $\forall x_1 \ldots \forall x_n \alpha \to [\alpha]_{\vec{x}} = 1$ where $\vec{x} = \langle x_1, \ldots, x_n \rangle$ is a vector of object variables.

P7. $[\alpha]_{\vec{x}} \geq 0$

P8. $[\alpha]_{\vec{x}} + [\neg\alpha]_{\vec{x}} = 1$

P9. $[\alpha \wedge \beta]_{\vec{x}} + [\alpha \wedge \neg\beta]_{\vec{x}} = [\alpha]_{\vec{x}}$

P10. $[\alpha]_{\vec{x}} = [\alpha(x_i/z)]_{\vec{x}(x_i/z)}$ where z is an object variable that does occur in α, and $\vec{x}(x_i/z)$ is the new vector of object variables $\langle x_1, \ldots, x_{i-1}, z, x_{i+1}, \ldots, x_n \rangle$

P11. $[\alpha \wedge \beta]_{\langle \vec{x}, \vec{y} \rangle} = [\alpha]_{\vec{x}} \times [\beta]_{\vec{y}}$ if none of the free variables of α are in $\vec{y}$, none of the free variables of β are in $\vec{x}$ and $\vec{x}$ and $\vec{y}$ are disjoint. This enforces the product measurement constraint.

P12. $[\alpha]_{\vec{x}} = [\alpha]_{\pi(\vec{x})}$ where π is any permutation of $\{1, \ldots, n\}$ and $\pi(\vec{x})$ is the permuted vector $\vec{x}$, i.e. $\pi(\vec{x}) = \langle x_{\pi(1)}, \ldots x_{\pi(n)} \rangle$

P13. $\forall x.\exists r.\mathbf{prob}([y = x]_y = r) = 1$

P14. $\exists r.\mathbf{prob}(r = E(t)) = 1$

P15. If t is rigid, then $E(t) = t$

P16. $\mathbf{cert}(t \circ t') \to E(t) \circ E(t')$ where

$\circ \in \{$ "$=$", "$<$", "$>$", "$\leq$", "$\geq$"$\}$

P17. $E(t) = 0 \equiv \mathbf{cert}(t = 0)$

P18. $E(t) = 1 \equiv \mathbf{cert}(t = 1)$

P19. $E(t + t') = E(t) + E(t')$

P20. If r is rigid, then $E(r \times t) = r \times E(t)$

P21. If r is rigid and non-zero, then $E(t/r) = E(t)/r$

P22. Axiom of propositional conditional probabilities
$$\mathbf{prob}(\beta) \neq 0 \quad \rightarrow \mathbf{prob}(\alpha|\beta) \times \mathbf{prob}(\beta) = \mathbf{prob}(\alpha \wedge \beta)$$
$$\mathbf{prob}(\beta) = 0 \quad \rightarrow \mathbf{prob}(\alpha|\beta) = 0$$

P23. Axiom of statistical conditional probabilities
$$[\beta]_{\vec{x}} \neq 0 \quad \rightarrow [\alpha|\beta]_{\vec{x}} \times [\beta]_{\vec{x}} = [\alpha \wedge \beta]_{\vec{x}}$$
$$[\beta]_{\vec{x}} = 0 \quad \rightarrow [\alpha|\beta]_{\vec{x}} = 0$$

Table A.6: Probability axioms in TPL

Axiom **P5** is the propositional probabilistic analogue of the Barcan formula; **P13** is the one for statistical cases. The former captures the fact that the set of possible objects $\mathcal{O}$ did not vary across possible worlds; the latter the fact that the domain probability function $\mu_{\mathcal{O}}$ is invariant across possible worlds.

Axiom **P6** says that if a set of satisfying instantiations of a formula contains all the vectors of $\mathcal{O}^n$ then the probability of this set is 1 thus capturing the relationship between the universal quantifier and the statistical probability terms.

Axioms **P7–P9** express the usual behaviour of probabilities.

Axiom **P10** captures the fact that variant probability terms are equivalent.

Axiom **P11** enforces the product measure constraint. For details see Bacchus' soundness proof [Bac90, p. 100].

Axiom **P12** captures the property that product measures are invariant under permutations.

Axioms **P14–P21** capture properties of the expectation operator E. Note that all of these may be derived from the semantics of the operator; however, as Bacchus notes [Bac90, p. 136], by adding them as axioms we may reason with the expectation operator.

Axioms **P22–P23** dealing with conditional probabilities explicitly defines the case where the conditioning formula has probability zero, contrary to normal practice that leaves it undefined.

As shown by Bacchus [Bac90, p. 103ff], there are many interesting tautologies and reasoning possible with the probability axioms. Some lemmas, e.g. Bacchus' Lemma 34 [Bac90, p. 107] show that certain complex probability terms may be simplified.

Theorem 23 (Bacchus' Lemma 34) *If no $x_i \in \vec{x}$ which appears in $\alpha \wedge \beta$ is free in λ then*

$$\models [\beta \wedge \lambda]_{\vec{x}} \neq 0 \rightarrow [\alpha|\beta \wedge \lambda]_{\vec{x}} = [\alpha|\beta]_{\vec{x}}. \tag{A.23}$$

Proof. The variables in $\vec{x}$ can be divided into two disjoint subsets $\vec{z}$ and $\vec{y}$. The subset $\vec{z}$ contains the variables that appear free in $\alpha \wedge \beta$, while $\vec{y}$ contain the variables that appear free in λ. By axiom **P11**, $[\alpha \wedge \beta \wedge \lambda]_{\vec{x}} = [\alpha \wedge \beta]_{\vec{z}} \times [\lambda]_{\vec{y}}$, and $[\beta \wedge \lambda]_{\vec{x}} = [\beta]_{\vec{z}} \times [\lambda]_{\vec{y}}$. Hence, the conditional probability $[\alpha|\beta \wedge \lambda]_{\vec{x}} = [\alpha \wedge \beta]_{\vec{x}}/[\beta]_{\vec{x}}$. Furthermore, since the variables $\vec{y}$ do not appear in $\alpha \wedge \beta$ we can add these extra random designators obtaining $[\alpha \wedge \beta]_{\vec{x}}/[\beta]_{\vec{x}} = [\alpha|\beta]_{\vec{x}}$ as required.

The fact that $[\beta \wedge \lambda]_{\vec{x}} > 0 \rightarrow [\lambda]_{\vec{y}} > 0$ allows us to cancel this term in the conditional probability. □

A.3.4 Temporal axioms

We base our logic on the following axioms using the relations mentioned above. We summarise van Benthem's theorem I.3.1.6, [Ben83, p. 72ff].

Theorem 24 *Up to isomorphism, $INT(\mathbb{Z})$ is defined by the first three groups of axioms in table A.7.*

Note that in **MOND** in table A.7 we use the union operator $\cup$ which for intervals is problematic (since intervals are not necessarily following each other). Following van Benthem we take the stand that the union of x and y implies that all intermediate durations (if any) are part of the union.

Proof. See van Benthem [Ben83, p. 72f] and Masini [Mas93, p. 17f]. □

As noted by van Benthem, though, that this theorem and its proof do not yield a recursive first-order axiomatisation for the theory of $INT(\mathbb{Z})$ the reason being axiom **FOUND**, which is not recursive, because a consequence of **FOUND** is that it implies the existence of base intervals (definition 68 above). In other words, it is not possible to have an infinite chain of inclusive intervals; at some time we will find the base interval with no inclusive intervals.

In order to be able to reason about the future (and the past) we need the frequently used next and future accessibility modalities. There are four of these: $\bigcirc_\Box, \bigcirc_\Diamond, \Box$ and $\Diamond$. Following Masini [Mas93] we have the following intuitive meanings, as shown in table A.8. We will omit the index from $\bigcirc$ whenever we do not need to emphasize the possibility of branching time (which we do not use in this work); thus $\bigcirc_\Box$ is the more common interpretation and the one we denote when omitting the subscript. We also note that $\Box$ may be seen as a 'henceforth' operator, and $\Diamond$ as an 'eventually' operator [Ben83, p. 156] [MP91, MP93]. This behaviour is captured in the following two definitions which are another versions of axioms **I13** and **I14**.

	For $<$:	
TRANS$(<)$	$\forall xyz.x < z < y \rightarrow x < y)$	transitivity
IRREF$(<)$	$\forall x.\neg x < x$	irreflexivity
SUCC	$\forall x \exists y.y < x, \quad \forall x \exists y.x < y$	succession
NEIGH	$\forall xy.x < y \rightarrow \exists y.x < y \wedge \neg \exists z.x < z < y$	
	$\forall xy.y < x \rightarrow \exists y.y < x \wedge \neg \exists z.y < z < x$	neighbourhood
	For $\sqsubseteq$:	
TRANS$(\sqsubseteq)$	$\forall xyz.x \sqsubseteq y \sqsubseteq z \rightarrow x \sqsubseteq z$	transitivity
REF$(\sqsubseteq)$	$\forall x.x \sqsubseteq x$	reflexivity
ANTIS	$\forall xy.x \sqsubseteq y \sqsubseteq x \rightarrow x = y$	anti-symmetry
CONJ	$\forall xy.xOy \rightarrow \exists z \sqsubseteq x.z \sqsubseteq y \wedge \forall u \sqsubseteq x.u \sqsubseteq y \rightarrow u \sqsubseteq z$	conjunction
DISJ	$\forall xy.\exists z \sqsupseteq x.z \sqsupseteq y \wedge \forall u \sqsupseteq x.u \sqsupseteq y \rightarrow u \sqsupseteq z$	disjunction
FREE	$\forall xy.\forall z \sqsubseteq x\ zOy \rightarrow x \sqsubseteq y$	freedom
DIR	$\forall xy \exists u.x \sqsubseteq u \wedge y \sqsubseteq u$	upward directedness
	For $<, \sqsubseteq$:	
MON	$\forall xy.x < y \rightarrow \forall u \sqsubseteq x.u < y$	left monotonicity
	$\forall xy.x < y \rightarrow \forall u \sqsubseteq y.x < u$	right monotonicity
MOND	$\forall xy.x < y \rightarrow \forall z.z < y \rightarrow x \cup z < y$	
	$\forall xy.y < x \rightarrow \forall z.y < z \rightarrow y < x \cup z$	
CONV	$\forall xyz.x < y < z \rightarrow \forall u.(x \sqsubseteq u \wedge z \sqsubseteq u) \rightarrow y \sqsubseteq u$	convexity
LIN$*$	$\forall xy.x < y \vee y < x \vee xOy$	linearity
FOUND	For no x is there a descending sequence $x \sqsupseteq y_1 \sqsupseteq y_2 \sqsupseteq \ldots$ in which every two successive periods are distinct.	well-foundedness
	For the modal temporal operators:	
K1	$\bigcirc(\alpha \rightarrow \beta) \rightarrow (\bigcirc\alpha \rightarrow \bigcirc\beta)$	
K2	$\Box(\alpha \rightarrow \beta) \rightarrow (\Box\alpha \rightarrow \Box\beta)$	
t1	$\Box(\alpha \rightarrow \bigcirc\alpha) \rightarrow (\alpha \rightarrow \Box\alpha)$	
t2	$\Box\alpha \rightarrow \bigcirc\Box\alpha$	
t3	$\Box\alpha \rightarrow \bigcirc\alpha$	
t4	$\Box\alpha \rightarrow \Diamond\alpha$	
4	$\Box\alpha \rightarrow \Box\Box\alpha$	
T	$\Box\alpha \rightarrow \alpha$	

Table A.7: Temporal axioms

$\bigcirc_\Box A$ means A is true in each possible next time.

$\bigcirc_\Diamond A$ means A is true in some possible next time.

$\Box A$ means A is true in each possible accessible future.

$\Diamond A$ means A is true in some possible accessible future.

Table A.8: Accessibility modalities in TPL

Definition 76 (Henceforth) *We say that $\Box A$ is true at a moment t iff A is true at $\forall t'$ such that $t \leq t'$. In other words, t' is later than t.*

Definition 77 (Eventually) *We say that $\Diamond A$ is true at a moment t iff A is true at some t' (or $\exists t'$) such that $t \leq t'$.*

It is easy to show that $\Box A$ implies $\Diamond A$.

Theorem 25 *$\Box A$ implies $\Diamond A$.*

Proof. Since $\Box A$ means that A at time t is true for all $t' > t$. Since we clearly have for all formulas B

$$\forall t.B \to \exists t.B \tag{A.24}$$

then we have that at time t there exists a future time t' where A is true (namely all of them because $\Box A$). This is the definition as stated above for $\Diamond A$ and we have that $\Box A \to \Diamond A$ as demanded. □

We incorporate these into our logic using the axioms in the fourth section in table A.7 [Mas93, p. 17f].

A.3.5 Rules of inference

We also need to show explicitly which rules of inference we support in our logic. They are listed below in table A.9 [Bac90, Mas93].

MP. From α and $\alpha \to \beta$ infer β (Modus ponens)

UG. From α infer $\forall x.\alpha$ (Universal generalisation)

PE. From $\alpha \equiv \beta$ infer $\mathbf{prob}(\alpha) = \mathbf{prob}(\beta)$ (Probability of equivalents)

Table A.9: Rules of inference in TPL

However, we cannot add the common necessity rule of inference **nec** to the table of rules above since its meaning would be that a true statement remains true for all future times; this is something that we cannot allow since statements are time-dependent. In other words, from A we cannot infer $\Box A$. Vice versa is allowed by axiom **T** in table A.7; i.e. from $\Box A$ we can infer A as well as $\Diamond A$ using theorem **t3** and **t4**.

A.4 THE DIRECT INFERENCE PRINCIPLE

As pointed out by Bacchus [Bac90, p. 139], what we now have is a logic capable of dealing with propositional and statistical probabilities, as well as temporal propositions. However, in this combined logic there is no intrinsic relationship between the two kinds of probabilities. They simply coexist without any necessary interaction.

As already pointed out, Bacchus notes that, although the types of probabilities are distinct, there is clearly a connection between them, which is most clearly apparent in actuarial situations, which often equate the two. A good example is an insurance company quoting a rate based on statistical information for an actual person.

This leads us to a very important principle: that of direct inference, from statistical probability to propositional.

Given a fixed statistical probability language $\mathcal{L}^{stat}$ extended with the propositional probability operator **prob** and the expectation operator E in order to be able to express agents' beliefs in propositions, we obtain a combined language $\mathcal{L}^{comb}$, as developed above. Now, given a finite knowledge base **KB** expressed as a conjunction of formulas, we can see **KB** as the belief base of the agent: everything the agent has accepted. This includes assertions about particular individuals as well as general logical relationships between properties, and statistical information. In order to formulate the direct inference principle, we restate definition 21. Note that, following Bacchus we equate degrees of belief with probability; i.e. fully believed entails certainty.

Definition 78 *Let α be a formula of $\mathcal{L}^{stat}$ and $\mathbf{KB}$ be a finite set of objective formulas from $\mathcal{L}^{stat}$ that are fully believed. If $\langle c_1, ...c_n\rangle$ are the n distinct object constants appearing in $\mathbf{KB}$ and $\langle v_1, ...v_n\rangle$ are n distinct object variables not appearing in $\mathbf{KB}$, then let $\mathbf{KB}^{\vec{v}}$ (and $\alpha^{\vec{v}}$) denote the new formula, which results from textually substituting c_i by v_i in $\mathbf{KB}$ (and in α) for all i.*

The direct inference principle is then as follows (restated from definition 22). Note that we give it the status of an axiom, although Bacchus always refers to it as a principle.

Axiom 7 *The* direct inference principle *is given by*

$$\mathbf{prob}(\alpha) = E\left(\left[\alpha^{\vec{v}} \mid \mathbf{KB}^{\vec{v}}\right]_{\vec{v}}\right). \tag{A.25}$$

This is then the agent's belief in α, given that $\mathbf{cert}\left([\mathbf{KB}^{v}]_{\vec{v}} > 0\right)$, and that the propositional and statistical probabilities are consistent.

As shown by Bacchus, [Bac90, p.152ff], the following important theorem is true.

Theorem 26 (Bacchus' Theorem 51) $\mathbf{prob}(\alpha) = 1$ *is a logical consequence of the direct inference principle iff (if and only if)* $\mathbf{KB} \models \alpha$.

Proof. Say that $\mathbf{KB} \models \alpha$. This means $\mathbf{KB} \rightarrow \alpha$ is valid; it is satisfied by every world in every combined probability structure. Hence, if $< c_1, \ldots, c_n >$ are the the constraints that appear in $\alpha \wedge \mathbf{KB}$, then no matter what their denotation is in a particular world, $\mathbf{KB} \rightarrow \alpha$ will be satisfied by that world. Therefore, $\forall\vec{v}.(\mathbf{KB} \rightarrow \alpha)^{\vec{v}}$ will also be satisfied by every world: it also will be valid. Then from the properties of the statistical terms we obtain the validity of $[(\mathbf{KB} \rightarrow \alpha)^{\vec{v}}]_{\vec{v}} = 1$, and $[\alpha^{\vec{v}}|\mathbf{KB}]_{\vec{v}} = 1$. Since this last is valid it is satisfied by every world, i.e., $\mathbf{cert}([\alpha^{\vec{v}}|\mathbf{KB}]_{\vec{v}} = 1)$, and hence we have that $E([\alpha^{\vec{v}}|\mathbf{KB}]_{\vec{v}}) = 1$. Thus we obtain $\mathbf{prob}(\alpha) = 1$ as a logical consequence of the direct inference principle.

In the opposite direction if $\mathbf{KB} \not\models \alpha$, then $\exists\vec{v}.(\mathbf{KB} \rightarrow \alpha)^{\vec{v}}$ is satisfiable. Hence, $[\alpha^{\vec{v}}|\mathbf{KB}]_{\vec{v}} < 1$ and so is $E([\alpha^{\vec{v}}|\mathbf{KB}]_{\vec{v}}) < 1$. Therefore, $\mathbf{prob}(\alpha) = 1$ is not a logical consequence. $\square$

A.5 SOUNDNESS, COMPLETENESS, AND CONSISTENCY

We have presented a formalisation of a logic, temporal probabilistic logic. As stated, this logic is a combination of three logics: standard first order logic, probabilistic logic, and temporal logic.

Is this combined logic *sound*, i.e., are all formulas deducible from it valid, and *complete*, i.e., are all valid formulas deducible from it? Furthermore, is it *consistent*, i.e., given a (deducible) formula α, its negation $\neg\alpha$ must not be deducible.

Briefly, following Bacchus [Bac90, p. 132] we can show that the combined logic of first order, propositional and statistical probability logic is sound. Van Benthem shows that interval-based temporal logic is sound as well. In order to show that these indeed hold for the combination of all three we formulate the following theorems for soundness and consistency.

Theorem 27 *TPL is sound.*

Proof. Since the only problem area is the constructs in the combined language that cross the border from temporal logic to Bacchus' logic (and vice versa) we have to show that these do not introduce formulas that are not valid into TPL. Following Epstein [Eps90] we rely on the properties of the Syntactic Consequence Relation, and show that, given a formula α, if there is a sequence $\alpha_1, ..., \alpha_n$ where each α_i is either an axiom, in TPL, or a consequence of some of the previous α_n 's using a rule of the system or a derived rule, and $\alpha_n = \alpha$, then TPL$\vdash \alpha$. Since this has been shown by Bacchus for the combination of first order, propositional and statistical probability and Bacchus' extensions to first-order logic (the constructs **prob**, [ ] and E) all generate f-terms (refer to table A.2), which are handled by his logic, Bacchus' logic cannot generate formulas containing temporal constructs, and is thus sound (i.e. no nonvalid formulas can be deduced from it).

The same is true going from temporal formulas, since, referring to the same table, we cannot deduce anything but formulas valid in first-order logic, as shown by van Benthem. The combination is thus also sound. $\square$

What we have shown above is that the combination of sound logics is itself sound, a result that is intuitively true.

Theorem 28 *TPL is consistent.*

Proof. As pointed out above, Bacchus has shown that the combined logic of first order, propositional and statistical probability logic is consistent; similarly, van Benthem has shown that interval temporal logic is consistent. Since non-consistency only then can be introduced in the combination of the two by constructs spanning the logics in question, we have a situation exactly analogous to the one for soundness. It is thus clear that given a formula α, its negation $\neg\alpha$ cannot be deduced; the combination logic is thus consistent. □

However, as Bacchus notes [Bac90, p. 62], Abadi and Halpern have shown that propositional probability logic is not complete, which is clear because the set of valid formulas do not form a recursively enumerable set. As van Benthem also points out, this is true for temporal logic as well. The same is then immediately true for our combined logic.

Bibliography

[Aba88] J. Y. Abadi, M. amd Halpern. Decidabilty and expressiveness of first-order logics of probability. Technical Report RJ. 7220 (67987), IBM Research, Almaden Research Center, San Jose, 1988.

[AH85] J. F. Allen and P.J Hayes. A common-sense theory of time. In *Proc. 9th Int. Conf. on AI*, pages 528–31, Los Altos, 1985. Morgan Kaufmann.

[AH87] J. F. Allen and P. J. Hayes. Short time periods. In *Proc. 10th Int. Joint Conf. on AI*, pages 981–83. Morgan Kaufmann, Los Altos, CA, 1987.

[AH89] J. F. Allen and P. J. Hayes. Moments and points in interval-based temporal logic. *Computational Intelligence*, 5:225–38, 1989.

[Ala92] Jarmo T. Alander. Geneettiset algoritmit – genetic algorithms. Technical Report TKO-C53, Helsinki University of Technology, Department of Computer Science, 1992.

[Ala96] Jarmo T. Alander, editor. *Proc. 2nd Nordic Workshop on Genetic Algorithms and their Applications (1996)*, Vaasa, 1996. Proceedings of the University of Vaasa, Number 11, University of Vaasa.

[Alt95] Lee Altenberg. The Schema Theorem and Price's theorem. In *Foundations of Genetic Algorithms 3*. Morgan Kaufmann, 1995.

[AMCB84] M. Ajmone-Marsan, G. Conti, and G. Balbo. A class of Generalised Stochastic Petri Nets for the performance evaluation of multiprocessor systems. *ACM Transactions on Computer Systems*, 2:93–122, 1984.

[Ank90] Carol Ann Ankenbrandt. *The Time Complexity of Genetic Algorithms and the Theory of Recombination Operators*. PhD thesis, Tulane University, 1990.

[ARS93] R. F Albrecht, C. R. Reeves, and N. C. Steele, editors. *Proc. Int. Conf. on Artifical Neural Networks and Genetic Algorithms ICANNGA-93*, Wien, 1993. Springer-Verlag.

[Atm94] Wirt Atmar. Notes on the simulation of evolution. *IEEE Transactions on Neural Networks*, 5(1):130–147, 1994.

[Bac90] Fahiem Bacchus. *Representing and Reasoning with Probabilistic Knowledge*. MIT Press, Cambridge, 1990.

[BdGKK97] Thomas Bäck, Jeannette de Graaf, Joost N. Kok, and Walter A. Kosters. Theory of genetic algorithms. *Bulletin of the European Association for Theoretical Computer Science*, (63):161–192, October 1997.

[BE95] Wolfgang Banzhaf and Frank H. Eckman. *Evolution as a Computational Process*, volume 899 of *Lecture Notes in Computer Science*. Springer Verlag, Berlin, 1995.

[Ben83] J. F. A. K. van Benthem. *The Logic of Time*. D. Reidel Publishing Company, Dordrecht, 1983.

[Ben88] J. F. A. K. van Benthem. *A Manual of Intensional Logic*. CSLI, Stanford University, Stanford, 1988.

[Bez93] James Bezdek. Fuzzy models — what are they, and why? *IEEE Transactions on Fuzzy Systems*, 1:1:1–6, 1993.

[BHS97] Thomas Bäck, Ulrich Hammel, and Hans-Paul Schwefel. Evolutionary computation: Comments on the history and current state. *IEEE Transactions on Evolutionary Computation*, 1 (1):3–17, 1997.

[Boo93] Lashon B. Booker. Recombination distributions for genetic algorithms. In *Foundations of Genetic Algorithms 2*, pages 29–44. Morgan Kaufmann, 1993.

[BS93] Thomas Bäck and Hans-Paul Schwefel. An overview of evolutionary algorithms for parameter optimization. *Evolutionary Computation*, pages 1–23, Vol. 1, No. 1, 1993.

[BS95a] Thomas Bäck and Hans-Paul Schwefel. Evolution strategies i: Variants and their computational implementation. In G. Winter, J. Périaux, M. Galán, and P. Cuesta, editors, *Genetic Algorithms in Engineering and Computer Science*, chapter 6. John Wiley & Sons, 1995.

[BS95b] Leonard Bolc and Andrzej Szalas. *Time and Logic: A Computation Approach*. University College Press, London, 1995.

[BT95] Tobias Blickle and Lothar Thiele. A comparison of selection schemes used in genetic algorithms. Technical Report TIK-Report 11, Swiss Federal Institute of Technology (ETH), Zürich, May 1995. Version 1.1.

[BV91] David L. Battle and Michael D. Vose. Isomorphisms of genetic algorithms. In *Foundations of Genetic Algorithms*. Morgan Kaufmann Publishers, San Mateo, 1991.

[Cas89] John L. Casti. *Paradigms Lost*. Avon Books, New York, 1989.

[Che80] Brian F. Chellas. *Modal Logic: An Introduction*. Cambridge University Press, Cambridge, 1980.

[Dar59] Charles Darwin. *On the Origin of Species by Means of Natural Selection.* Murray, London, 1859.

[Dav91a] Lawrence Davis, editor. *Handbook of Genetic Algorithms.* van Nostrand Reinhold, New York, 1991.

[Dav91b] T. Davis. *Toward an Extrapolation of the Simulated Annealing Convergence Theory onto the Simple Genetic Algorithm.* PhD thesis, University of Florida, 1991.

[Daw96] Richard Dawkins. *Climbing Mount Improbable.* Viking, London, 1996.

[Den95] Daniel C. Dennett. *Darwin's Dangerous Idea.* Simon and Schuster, New York, 1995.

[DJ75] Kenneth A. De Jong. *An Analysis of the Behaviour of a Class of Genetic Adaptive Systems.* PhD thesis, University of Michigan, 1975. Dissertation Abstracts International 36(1), 5140B.

[Eps90] Richard L. Epstein. *The Semantic Foundations of Logic Volume 1: Propositional Logics.* Kluwer Academic Publishers, Dordrecht, the Netherlands, 1990.

[FG97] David B. Fogel and Adam Ghozeil. A note on representation and variation operators. *IEEE Transactions on Evolutionary Computation*, 1(2):159–161, July 1997.

[FM92] Stephanie Forrest and Melanie Mitchell. Relative building-block fitness and the building-block hypothesis. In *FOGA-92. Proc. Of Workshop on Foundations of Genetic Algorithms and Classifier Systems*, pages 109–126, 1992.

[FM93] Stephanie Forrest and Melanie Mitchell. What makes a problem hard for genetic algorithms? *Machine Learning*, pages 285–319, Vol. 13, No. 2-3 November-December 1993.

[Fog94a] David B. Fogel. Global convergence. Communication in newsgroup comp.ai.genetic, November 1994.

[Fog94b] David B. Fogel. An introduction to simulated evolutionary optimization. *IEEE Transactions on Neural Networks*, 5(1):3–14, 1994.

[Fog95a] David B. Fogel. *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence.* IEEE Press, Piscataway, N.J., 1995.

[Fog95b] David B. Fogel, editor. *IEEE Proc. Int. Conf. on Evolutionary Computation (ICEC-95)*, Perth, Australia, 1995.

[For93] Stephanie Forrest, editor. *Proc. 5th Int. Conf. on Genetic Algorithms*, San Mateo, 1993. Morgan Kaufmann.

[GD91] David E. Goldberg and Kalyanmoy Deb. A comparative study of selection schemes used in genetic algorithms. In Gregory J.E. Rawlins, editor, *Foundations of Genetic Algorithms*, pages 69–93. Morgan Kaufmann, San Mateo, 1991.

[Gen87] Hartmann J. Genrich. Predicate / transition nets. In W. Brauer, W. Reisig, and G. Rozenberg, editors, *Petri nets: Central problems and their properties*, LNCS, no. 254, pages 207–247. Springer-Verlag, Berlin, 1987.

[Gen90] Hartmann J. Genrich. Equivalence transformations of PrT-Nets. In Grzegorz Rozenberg, editor, *Advances in Petri Nets 1989*, LNCS, no. 424, pages 179–208. Springer-Verlag, Berlin, 1990.

[Goe95] Ben Goertzel. A convergence theorem for the simple GA with population size tending to infinity. In *Proc. IEEE ICEC'95*. IEEE Press, Perth, Australia, 1995.

[Gol87] David E. Goldberg. Simple genetic algorithms and the minimal deceptive problem. In L. Davis, editor, *Genetic Algorithms and Simulated Annealing*, pages 74–88. Pitman, London, 1987.

[Gol89] David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, Reading, 1989.

[Gol90] David E. Goldberg. A note on Boltzmann tournament selection for genetic algorithms and population-oriented simulated annealing. *Complex Systems*, 4:445–460, 1990.

[Haa95] Juha Haataja. Comparing global optimization methods. In *Proc. 1st Nordic (3rd Finnish) Workshop on Genetic Algorithms and their Applications*, pages 189–204, Vaasa, 1995. Vaasan Yliopisto.

[Had94] Peter Haddawy. *Representing Plans under Uncertainty*, volume 770 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, Berlin, 1994.

[Had96] Peter Haddawy. A logic of time, chance, and action for representing plans. *Artificial Intelligence*, 80(2):243–308, February 1996.

[Haj95] E. Hajnicz. An analysis of structure of time in the first order predicate calculus. In *Time and Logic: A Computational Approach*, chapter 7. University College Press, 1995.

[HB92] Frank Hoffmeister and Thomas Bäck. Genetic algorithms and evolution strategies: Similarities and differences. Technical Report SYS-1/92, University of Dortmund, Dortmund, 1992.

[Hin62] Jaakko Hintikka. *Knowledge and Belief: An Introduction to the Logic of the Two Notions.* Cornell University Press, Ithaca, 1962.

[HKB94] William E. Hart, Thomas E. Kammeyer, and Richard K. Belew. The role of development in genetic algorithms. Technical Report CS94-394, also in [WV95], U.C.S.D., San Diego, 1994.

[Hol92a] John H. Holland. *Adaptation in Natural and Artificial Systems.* MIT Press, Cambridge, 1992.

[Hol92b] John H. Holland. Genetic algorithms. *Scientific American,* pages 44–51, July 1992.

[HP95] Sridhar Hannenhalli and Pavel A. Pevzner. Towards a computational theory of genome rearrangements. In Jan van Leeuwen, editor, *Computer Science Today,* volume 1000 of *LNCS*, pages 184–202. Springer-Verlag, 1995.

[Hun95a] Andrew Hunter. *SUGAL Programming Manual V2.1.* University of Sunderland, Sunderland, England, 1995.

[Hun95b] Andrew Hunter. *SUGAL User Manual V2.1.* University of Sunderland, Sunderland, England, 1995.

[Jon95] Terry Jones. *Evolutionary Algorithms, Fitness Landscapes and Search.* PhD thesis, University of New Mexico, 1995.

[JR91] Kurt Jensen and Grzegorz Rozenberg. *High-Level Petri nets: Theory and Applications.* Springer-Verlag, Berlin, 1991.

[JS94] Christian S. Jensen and Richard Snodgrass. Temporal specialization and generalisation. *IEEE Transactions on Knowledge and Data Engineering,* pages 954–974, Vol. 6, No. 6, December 1994.

[Lei95] Donald Dewar Leitch. *A New Genetic Algorithm for the Evolution of Fuzzy Systems.* PhD thesis, University of Oxford, 1995.

[Lew74] R. C. Lewontin. *The Genetic Basis of Evolutionary Change.* Columbia University Press, 1974.

[Mah93] Samir W. Mahfoud. Finite Markov chain models of an alternative selection strategy for the genetic algorithm. Technical Report IlliGAL 91007, University of Illinois at Urbana-Champaign, 1991, rev. 1993.

[Mah95] Samir W. Mahfoud. *Niching Methods for Genetic Algorithms.* PhD thesis, University of Illinois at Urbana-Champaign, 1995. Also published as IlliGAL Report 95001.

[Mas93] Andrea Masini. *A Proof Theory of Modalities for Computer Science.* PhD thesis, University of Pisa, 1993.

[MCH93] Melanie Mitchell, James P. Crutchfield, and Peter T. Hraber. Evolving cellular automata to perform computations: Mechanisms and impediments. Technical Report 93-11-071, Santa Fe Institute, Santa Fe, 1993.

[MF93] Melanie Mitchell and Stephanie Forrest. Genetic algorithms and artificial life. Technical Report 93-11-072, Santa Fe Institute, Santa Fe, 1993.

[MH93] Melanie Mitchell and John H. Holland. When will a genetic algorithm outperform hill-climbing? In *Proc. 5th Int. Conf. On Genetic Algorithms*, page 647, 1993.

[MHC93] Melanie Mitchell, Peter T. Hraber, and James P. Crutchfield. Revisiting the edge of chaos: Evolving cellular automata to perform computations. Technical Report Report WP-93-03-014, The Santa Fe Institute, Santa Fe, 1993.

[Mic99] Zbigniew Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs.* Springer Verlag, New York, 1999. 3rd revised and extended edition.

[Mit96] Melanie Mitchell. *An Introduction to Genetic Algorithms.* MIT Press, Cambridge, 1996.

[MP91] Z. Manna and A. Pnueli. *The Temporal Logic of Concurrent and Reactive Systems: Specification.* Springer-Verlag, New York, 1991.

[MP93] Z. Manna and A. Pnueli. Models for reactivity. *Acta Informatica*, 30:609–678, 1993.

[MSV93] Heinz Mühlenbein and Dirk Sclierkamp-Voosen. Predictive models for the breeder genetic algorithm. *Evolutionary Computing*, 1(1), 1993.

[MSV95] Heinz Mühlenbein and Dirk Sclierkamp-Voosen. Analysis of selection, mutation and recombination in genetic algorithms. In Wolfgang Banzhaf and Frank H. Eckman, editors, *Evolution as a Computational Process*, volume 899 of *Lecture Notes in Computer Science*, pages 142–168. Springer-Verlag, Berlin, 1995.

[MYT+95] Naoki Mori, Junji Yoshida, Hisashi Tamaki, Hajime Kita, and Yoshikazu Nishikawa. A thermodynamical selection rule for the genetic algorithm. In David T. Fogel, editor, *IEEE Proc. Int. Conf. on Evolutionary Computation (ICEC-95).* IEEE Press, Perth, 1995.

[Net94] David John Nettleton. *Evolutionary Algorithms in Artificial Intelligence: A Comparative Study Through Applications.* PhD thesis, University of Durham, United Kingdom, 1994.

[NV92] A. Nix and M. D. Vose. Modeling genetic algorithms with Markov chains. In *Annals of Mathematics and Artificial Intelligence 5*, pages 79–88. 1992.

[Ost89] Jonathan Ostroff. *Temporal Logic for Real-Time Systems.* John Wiley, New York, 1989.

[Pap65] Athanasios Papoulis. *Probability, Random Variables, and Stochastic Processes.* McGraw-Hill, New York, 1965.

[Pet62] Carl Adam Petri. *Kommunikation mit Automaten.* PhD thesis, Technische Universität Darmstadt, 1962. Fakultät Matematik und Physik.

[Pet81] James L. Peterson. *Petri Net Theory and the Modelling of Systems.* Prentice Hall, Englewood Cliffs, New Jersey, 1981.

[Pet90] Ivars Peterson. *Islands of Truth – A Mathematical Mystery Cruise.* W. H. Freeman and Company, New York, 1990.

[Pys96] Tino Pyssysalo. An induction theorem for ring protocols of processes described with predicate / transition nets. Technical Report Series A, no. 37, Helsinki University of Technology, Digital Systems Laboratory, Otaniemi, 1996.

[Rad91] Nicholas J. Radcliffe. Equivalence class analysis of genetic algorithms. *Complex Systems*, pages 183–205, Vol. 5, No. 2, 1991.

[Rad93] Nicholas J. Radcliffe. Genetic set recombination. In *Foundations of Genetic Algorithms 2.* Morgan Kaufmann, 1993.

[Rad94a] Nicholas J. Radcliffe. The algebra of genetic algorithms. *Annals of Mathematics and Artificial Intelligence*, (10), 1994.

[Rad94b] Nicholas J. Radcliffe. Equivalence class analysis of genetic algorithms. In *Foundations of Genetic Algorithms 3.* Morgan Kaufmann, 1994.

[Rei82] Wolfgang Reisig. *Petri Nets: An Introduction.* Springer-Verlag, Berlin, 1982.

[RF96] Jean-Michel Renders and Stéphane P. Flasse. Hybrid methods using genetic algorithms for global optimization. *IEEE Transactions on Systems, Man and Cybernetics*, 26(2):243–258, April 1996.

[RG96] David Reynolds and Jagannathan Gomatam. Stochastic modelling of genetic algorithms. *Artificial Intelligence*, 82(1–2):303–330, April 1996.

[Roz90] Grzegorz Rozenberg, editor. *Advances in Petri Nets 1989*, volume 424 of *LNCS.* Springer-Verlag, Berlin, 1990.

[RR98a] Wolfgang Reisig and Grzegorz Rozenberg. *Lectures on Petri Nets I: Basic Models*, volume 1491 of *LNCS*. Springer-Verlag, Berlin, 1998.

[RR98b] Wolfgang Reisig and Grzegorz Rozenberg. *Lectures on Petri Nets II: Applications*, volume 1492 of *LNCS*. Springer-Verlag, Berlin, 1998.

[Rud94] Günter Rudolph. Convergence analysis of canonical genetic algorithms. *IEEE Transactions on Neural Networks*, 5(1):96–101, 1994.

[San93a] Sam Sandqvist. Genetic algorithms and database queries. In Jarmo T. Alander, editor, *Proc. 1st Finnish Workshop on Genetic Algorithms and their Applications*, pages 123–132, Otaniemi, 1993. Helsinki University of Technology.

[San93b] Sam Sandqvist. *On Finding Optimal Potential Customers from a Large Marketing Database - A Genetic Algorithm Approach*. Helsinki University of Technology, Otaniemi, 1993. Licentiate Thesis.

[San93c] Sam Sandqvist. On finding optimal potential customers from a large marketing database - a genetic algorithm approach. In R. F. Albrecht, C. R. Reeves, and N. C. Steele, editors, *Proc. ICANNGA-93 Int. Conf. on Artifical Neural Networks and Genetic Algorithms*, pages 528–535, Wien, 1993. Springer-Verlag.

[San94a] Erik Sandewall. Features and fluents. Technical Report LiTH-IDA-R-94-15, University of Linköping, Linköping, 1994.

[San94b] Sam Sandqvist. A temporal probabilistic model of the genetic algorithm. In Jarmo T. Alander, editor, *Proc. 2nd Finnish Workshop on Genetic Algorithms and their Applications*, pages 138–150, Vaasa, 1994. University of Vaasa.

[San95] Sam Sandqvist. Modelling the time behaviour of genetic algorithms. In Jarmo T. Alander, editor, *Proc. 1st Nordic (3rd Finnish) Workshop on Genetic Algorithms and their Applications*, pages 205–214, Vaasa, 1995. University of Vaasa.

[San96a] Sam Sandqvist. Some aspects of time in genetic algorithms. In Jarmo T. Alander, editor, *Proc. 2nd Nordic (4th Finnish) Workshop on Genetic Algorithms and their Applications*, pages 267–276, Vaasa, 1996. University of Vaasa.

[San96b] Sam Sandqvist. Temporal populations. In Takeshi Furuhashi, editor, *Proc. 2nd Online Workshop on Evolutionary Computation*. Nagoya University, Japan, 1996.

[San96c] Sam Sandqvist. A temporal view of selection and populations. In Terence C. Fogarty, editor, *Evolutionary Computing*, volume 1143 of *LNCS*, pages 126–136. Springer-Verlag, Berlin, 1996.

[SB95] Hans-Paul Schwefel and Thomas Bäck. Evolution strategies ii: Theoretical aspects. In G. Winter, J. Périaux, M. Galán, and P. Cuesta, editors, *Genetic Algorithms in Engineering and Computer Science*, chapter 7. John Wiley & Sons, 1995.

[SGH95] B. Strulo, D. Gabbay, and P. H. Harrison. Temporal logic in a stochastic environment. In *Time and Logic: A Computational Approach*, chapter 5. University College Press, 1995.

[Sho88] Yoav Shoham. *Reasoning About Change*. MIT Press, Cambridge, 1988.

[SMSV83] Richard L. Schwartz, P. M. Melliar-Smith, and Friedrich H. Vogt. Interval logic: A higher-level temporal logic for protocol specification. pages 3–18. Elsevier Science Publishers, 1983.

[Sto79] Robert R. Stoll. *Set Theory and Logic*. Dover, New York, 1979. First published by W. H. Freeman, 1963.

[Suz93] Joe Suzuki. A Markov chain analysis on a genetic algorithm. In Stephanie Forrest, editor, *Proc. Fifth International Conference on Genetic Algorithms*, San Mateo, 1993. Morgan Kaufmann.

[Sys91] Gilbert Syswerda. A study of reproduction in generational and steady-state genetic algorithms. In Gregory J. E. Rawlins, editor, *Foundations of Genetic Algorithms*, pages 94–101. Morgan Kaufmann, San Mateo, 1991.

[Tsa87] E. P. K. Tsang. Time structures for AI. In *Proc. 10th Int. Joint Conf. on AI*, pages 456–61. Morgan Kaufmann, Los Altos, 1987.

[VHHP95] Kimmo Varpaaniemi, Jaakko Halme, Kari Hiekkanen, and Tino Pyssysalo. PROD reference manual. Technical Report B13, Helsinki University of Technology, Digital Systems Laboratory, Otaniemi, 1995.

[VL91] Michael D. Vose and G. E. Liepins. Punctuated equilibria in genetic search. *Complex Systems*, 5:31–44, 1991.

[vL95] Jan van Leeuwen. *Computer Science Today: Recent Trends and Developments*, volume 1000 of *Lecture Notes in Computer Science*. Springer-Verlag, Heidelberg, 1995.

[Vos91a] Michael D. Vose. Formalizing genetic algorithms. Technical Report CS-91-127, University of Tennessee, 1991.

[Vos91b] Michael D. Vose. Generalizing the notion of schema in genetic algorithms. *Artificial Intelligence*, 50(3):385–396, 1991.

[Vos93] Michael D. Vose. Modelling simple genetic algorithms. In *Foundations of Genetic Algorithms*, pages 63–74. Morgan Kaufmann, 1993.

[Whi92] L. Darrell Whitley, editor. *FOGA-92. Proc. Workshop on the Foundations of Genetic Algorithms and Classifier Systems*, San Mateo, 1992. Morgan Kaufmann.

[Whi93] L. Darrell Whitley. An executable model of a simple genetic algorithm. In *Foundations of Genetic Algorithms 2*, pages 45–62. Morgan Kaufmann, 1993.

[Whi94] L. Darrell Whitley. A genetic algorithm tutorial. *Statistics and Computing*, 4:65–85, 1994.

[WL95] Annie S. Wu and Robert K. Lindsay. Empirical studies of the genetic algorithm with non-coding segments. *Evolutionary Computation*, 3(2), 1995.

[WM95] David H. Wolpert and William G. Macready. No free lunch theorems for search. Technical Report SFI-TR-95-02-010, The Santa fe Institute, Santa Fe, 1995.

[WM97] David H. Wolpert and William G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1 (1):67–82, 1997.

[Wor95] Robert P. Worden. A speed limit for evolution. *Journal of Theoretical Biology*, 176 (137), 1995.

[WPGC95] G. Winter, J. Périaux, M. Galán, and P. Cuesta. *Genetic Algorithms in Engineering and Computer Science*. John Wiley & Sons, Chichester, 1995.

[Wri32] Sewall Wright. The roles of mutation, inbreeding, crossbreeding, and selection in evolution. In *Proc. 6th Intl. Congress of Genetics*, pages 356 – 366, 1932.

[WV95] L. Darrell Whitley and Michael D. Vose. *Foundations of Genetic Algorithms 3*. Morgan Kaufmann Publishers, San Fransisco, 1995.

[Zad65] Lotfi Zadeh. Fuzzy sets. *Information and Control*, 8:338–353, 1965.

HELSINKI UNIVERSITY OF TECHNOLOGY LABORATORY FOR THEORETICAL COMPUTER SCIENCE
RESEARCH REPORTS

HUT-TCS-A61	Tuomas Aura, Carl Ellison Privacy and accountability in certificate systems. April 2000.
HUT-TCS-A62	Kari J. Nurmela, Patric R. J. Östergård Covering a Square with up to 30 Equal Circles. June 2000.
HUT-TCS-A63	Nisse Husberg, Tomi Janhunen, Ilkka Niemelä (Eds.) Leksa Notes in Computer Science. October 2000.
HUT-TCS-A64	Tuomas Aura Authorization and availability - aspects of open network security. November 2000.
HUT-TCS-A65	Harri Haanpää Computational Methods for Ramsey Numbers. November 2000.
HUT-TCS-A66	Heikki Tauriainen Automated Testing of Büchi Automata Translators for Linear Temporal Logic. December 2000.
HUT-TCS-A67	Timo Latvala Model Checking Linear Temporal Logic Properties of Petri Nets with Fairness Constraints. January 2001.
HUT-TCS-A68	Javier Esparza, Keijo Heljanko Implementing LTL Model Checking with Net Unfoldings. March 2001.
HUT-TCS-A69	Marko M<E4>kel<E4> A Reachability Analyser for Algebraic System Nets. June 2001.
HUT-TCS-A70	Petteri Kaski Isomorph-Free Exhaustive Generation of Combinatorial Designs. December 2001.
HUT-TCS-A71	Keijo Heljanko Combining Symbolic and Partial Order Methods for Model Checking 1-Safe Petri Nets. February 2002.
HUT-TCS-A72	Tommi Junttila Symmetry Reduction Algorithms for Data Symmetries. May 2002.
HUT-TCS-A73	Toni Jussila Bounded Model Checking for Verifying Concurrent Programs. August 2002.
HUT-TCS-A74	Sam Sandqvist Aspects of Modelling and Simulation of Genetic Algorithms: A Formal Approach. September 2002.

ISBN 951-22-6115-4
ISSN 1457-7615