TEDA: A TARGETED ESTIMATION OF DISTRIBUTION ALGORITHM

GEOFFREY NEUMANN

Doctor of Philosophy

Division of Computing Science and Mathematics
University of Stirling

February 2014

[16th May 2014 at 0:34]

DECLARATION

I, Geoffrey Kenneth Neumann, hereby declare that all material in this thesis is my own original work except where reference has been given to other authors and that it has not been submitted for the award of any other degree at the University of Stirling or at another Institution.

Stirling, February 2014

Geoffrey Neumann

ABSTRACT

This thesis discusses the development and performance of a novel evolutionary algorithm, the Targeted Estimation of Distribution Algorithm (TEDA). TEDA takes the concept of *targeting*, an idea that has previously been shown to be effective as part of a Genetic Algorithm (GA) called Fitness Directed Crossover (FDC), and introduces it into a novel hybrid algorithm that transitions from a GA to an Estimation of Distribution Algorithm (EDA).

Targeting is a process for solving optimisation problems where there is a concept of *control points*, genes that can be said to be *active*, and where the total number of control points found within a solution is as important as where they are located. When generating a new solution an algorithm that uses targeting must first of all choose the number of control points to set in the new solution before choosing which to set.

The hybrid approach is designed to take advantage of the ability of EDAs to exploit patterns within the population to effectively locate the global optimum while avoiding the tendency of EDAs to prematurely converge. This is achieved by initially using a GA to effectively explore the search space before transitioning into an EDA as the population converges on the region of the global optimum. As targeting places an extra restriction on the solutions produced by specifying their size, combining it with the hybrid approach allows TEDA to produce solutions that are of an optimal size and of a higher quality than would be found using a GA alone without risking a loss of diversity.

TEDA is tested on three different problem domains. These are optimal control of cancer chemotherapy, network routing and Feature Subset Selection (FSS). Of these problems, TEDA showed consistent advantage over standard EAs in the routing problem and demonstrated that it is able to find good solutions faster than untargeted EAs and non evolutionary approaches at the FSS problem. It did not demonstrate any advantage over other approaches when applied to chemotherapy.

The FSS domain demonstrated that in large and noisy problems TEDA's targeting derived ability to reduce the size of the search space significantly increased the speed with which good solutions could be found. The routing domain demonstrated that, where the ideal number of control points is deceptive, both targeting and the exploitative capabilities of an EDA are needed, making TEDA a more effective approach than both untargeted approaches and FDC. Additionally, in none of the problems was TEDA seen to perform significantly worse than any alternative approaches.

To my parents

ACKNOWLEDGMENTS

I saved this section to the very end and it feels amazing to be writing it at last. It's impossible to include everyone on this list. If you've ever given me advice, given me a chance to take my mind off my thesis or just sat there and listened to me complain about the frustrations of research when I've needed it then thank you so much, whether I've remembered to include you in this section or not.

First and foremost, I would like to thank my supervisor, David Cairns. You've patiently reassured me for every doubt that I've had, answered my every question and meticulously gone through every word I've written. I would not have known where to begin without you. I would also like to thank Paul Godley, whose work made the topic of my thesis possible. I'd like to thank John McCall, I only spoke to you a few times but the advice you gave at a CEC conference in Brisbane ultimately enabled me to complete some of the most successful work in this thesis. I'd like to thank Jerry Swan and the DAASE team for giving me a chance to expand my mind and work on things quite different to my thesis towards the end of my PhD, work which ultimately lead to a job. I'd like to thank Leslie Smith as well for sharing some of your incredible wisdom with me.

I would like to thank the Division of Computing Science and the University of Stirling for giving me the opportunity to undertake this PhD and for all the support networks you have in place that has made this journey so much smoother and more enjoyable. I would like to thank Horizon studentships for funding me. I would like to Savi Maharaj anyone else in the department responsible for giving me the opportunity to teach. Teaching has kept my computing knowledge fresh but has also taught me so much more beyond computer science.

I'd like to thank all of the PhD's who have gone before me and shared their wisdom, Claire, Jamie, James, Jesse and most of all, Andy.

But beyond computer science there are many others who have been equally invaluable. Most importantly, I'd like to thank my parents, your reassurances have kept me going no matter what. I'd like to thank my best friends back in Sussex; Ajit, Rich, Ross and Laura. You've had to listen to your fair share of my thesis woes when times have been bad and yet you've always been patient, always advised as much as you could and few things have been more successful at cheering me up in the harder phases than being over competitive at air hockey on Brighton pier or exploring Ditchling Common or the Downs.

I would also like to thank Noelle, our travels together really brightened up what was otherwise the difficult second year of my PhD. And your reassuring words beyond those trips have also been invaluable.

There are two organisations I'd like to thank (and the people in them of course). Neither of which have anything to do with computer science or academic research but that's part of the reason why they've been so invaluable.

Firstly, there's the Renshinkan Shorinji-Ryu karate school and Shihan, my teacher. For 18 years karate has been a constant in my life and an absolute release no matter what else has been going on and for that I will forever be grateful. With my parents, karate and my three best buddies its no wonder I spent so many nights on busses going down to Sussex. I'm not going to go quite as far as thanking Megabus though!

Secondly, there's the Stirling University Drama Society. Thanks for being entertaining and funny and awesome and for giving me the chance to sometimes take a break from writing about computer science to write about dinosaurs and superheroes instead, in the form of comedy sketches!

Often people end acknowledgements on the person or people who they wish to thank the most but I couldn't possibly choose! Thankyou everyone!

LIST OF PUBLICATIONS

- G. K. Neumann and D. E. Cairns. Introducing Intervention Targeting into Estimation of Distribution Algorithms. In *Proceedings of the 27th ACM Symposium on Applied Computing*, Trento Italy, pages 220-225, 2012.
- G. K. Neumann and D. E. Cairns. Targeted EDA adapted for a routing problem with variable length chromosomes. In *IEEE Congress on Evolutionary Computation (CEC)*, Brisbane Australia, pages 334-341, 2012.
- G. K. Neumann and D. E. Cairns. Applying a Hybrid Targeted Estimation of Distribution Algorithm to Feature Selection Problems. In *5th International Conference on Evolutionary Computation Theory and Applications*, Villamoura Portugal, pages 136-143, 2013.
- G. K. Neumann and D. E. Cairns. A Targeted Estimation of Distribution Algorithm Compared to Traditional Methods in Feature Selection. In *Studies in Computational Intelligence, LNCS, Springer, 2013 (In Press)*.

CONTENTS

LIST OF TABLES

LIST OF ACRONYMS

BER Balanced Error Rate

BIC Bayesian Information Criterion

BS Backward Selection

CALEB Calculated Expanding Bin

CFS Correlated Feature Selection

CGA Compact Genetic Algorithm

DAG Directed Acyclic Graph

DUC Directed Uniform Crossover

EA Evolutionary Algorithm

EC Evolutionary Computing

EBNA Estimation of Bayesian Network Algorithm

ECGA Extended Compact Genetic Algorithm

EDA Estimation of Distribution Algorithm

EP Evolutionary Programming

ES Evolutionary Strategies

FCBF Fast Correlatation Based Filter

FDA Factorised Distribution Algorithm

FDC Fitness Directed Crossover

FDC_2 Fitness Directed Crossover 2

FS Forward Selection

FSS Feature Subset Selection

GA Genetic Algorithm

GP Genetic Programming

K-NN K-Nearest Neighbour

KW Kruskal Wallis

MIMIC Mutual Information Maximisation for Input Clustering

NSGA Non Dominated Sorting Genetic Algorithm

PBIL Population Based Incremental Learning

PLS Probabilistic Logic Sampling

SVM Support Vector Machine

TEDA Targeted Estimation of Distribution Algorithm

TEDA_a Targeted Estimation of Distribution Algorithm - Prototype 'a'

TINSSEL Targeted Intervention with Stochastic Selection

UC Uniform Crossover

UMDA Univariate Marginal Distribution Algorithm

1

INTRODUCTION

In this thesis we discuss the application of Evolutionary Algorithms (EAs) to optimization problems where a key concern is the total number of control points within a solution. A *control point* is defined as a gene which can be said to be *active*, often through having a positive value. There may be a cost associated with each control point and so the overall objective is to solve a given problem while using as few control points as possible. We propose a general solution to many different problems that share this characteristic. Problems such as:

- Optimal control problems where control points take the form of interventions which may have a cost associated with them.
- Routing problems where the objective is to find an effective route through a network that passes through as few routers as possible.
- FSS problems where we wish to identify feature sets that contain only the features necessary for the classification task at hand.

EAs are metaheuristic optimization techniques which use the principles of selection as found in natural evolution to generate new solutions. In EAs a population of possible solutions is generated at random and then scored. Those that score well are selected and used to produce a new population of potential solutions. Each solution is represented as a *chromosome*, a structure containing *genes* which represent problem variables.

The three main areas of Evolutionary Computing are [112] Genetic Algorithms (GA) [66], Evolutionary Programming (EP) [106] and Evolutionary Strategies (ES) [42]. Of special interest in this work are GAs and a collection of techniques falling within the broader ES category known as Estimation of Distribution Algorithm (EDA) [83].

In GAs, two solutions are selected via some competitive process and new solutions are produced by mixing the genes found in these two *parent* solutions. This mixing process is known as *crossover*. The newly created individuals, the *children*, are then modified slightly using a technique known as *mutation* in order to maintain diversity within the population.

GAs have been used to produce optimal solutions to control problems where there is a penalty associated with using too many control points. Previous work by Godley et al resulted in Fitness Directed Crossover (FDC) [48], a GA crossover technique designed to find optima with appropriate intervention usage. FDC outperformed more traditional approaches, such as Uniform Crossover (UC), when applied to a bio-control agent scheduling problem in mushroom farming and to a cancer chemotherapy problem [48].

Instead of selecting two parent solutions for crossover, EDAs build a probability distribution from a pool of highly fit solutions and produce new solutions by sampling from this distribution. As they are sensitive to patterns within the population, EDAs have been proven to be powerful optimization techniques [112][97][100][113], although this can be very dependent on how well they initially sample the population [112].

1.1 THE TARGETED ESTIMATION OF DISTRIBUTION ALGORITHM

Targetted EDA (TEDA) [94] is the algorithm proposed in this thesis and is constructed as a GA/EDA hybrid. It uses the targeting principle from FDC to decide on an appropriate number of control points, the intervention target, and then generates new solutions with that number of control points set by sampling from a distribution using an EDA. TEDA also changes over time, behaving purely like FDC early on and later transitioning to a more EDA like behaviour after FDC has driven down the number of interventions used and caused the population to begin to converge.

This transitioning process is used to enable TEDA to exploit the strengths of both EDAs and GAs. The tendency of EDAs to prematurely converge is especially significant in the case of TEDA as choosing a number of control points to set in advance places an extra restriction on the range of solutions that can be created. On the other hand, the superior ability of EDAs to exploit patterns in the population should enable TEDA to ultimately find fitter solutions than the purely GA based FDC. To perform well an EDA should be applied when the population is sufficiently close to the global optimum that it does not risk prematurely converging before it reaches it and so TEDA should use FDC to explore the search space and find an area suitably close to the global optimum and should then apply an EDA to more precisely locate the optimum.

Combining the ability of EDAs to use statistical information taken from the whole population with a GA's ability to create new solutions similar to existing fit solutions is an area that has already produced useful algorithms [143]. Hauschild lists hybridisation with other algorithms as one of the main techniques for improving the efficiency of EDAs [64]. Chen demonstrates that alternating GAs with EDAs and using each to generate a proportion of the population can ensure that diversity is maintained and that premature convergence is avoided [30]. Yang suggests using genetic algorithms to locate the area of the global optimum before other techniques are used to accurately locate the optimum. This exploits the fact that GAs are effective at initial exploration of the search space but less effective at honing the search towards the global optimum [138].

In much of this work the way in which the different techniques are combined is intelligent, so that the algorithm responds to the nature of the population when deciding which technique to use. This is because the rate at which the population changes from one most suitably

optimized by a GA and one most suitably optimized by an EDA cannot be assumed to be fixed or to be easily determined by a human expert based on their knowledge of the domain. This is the theory behind self adaptive algorithms [39] and hyperheuristics [108][22]. TEDA draws on this research and a technique for transitioning dynamically from a GA to an EDA based on the diversity within the population has been developed.

1.2 OUTLINE

This work sets out the development and testing of TEDA in chronological order. It is organised so that there is one chapter for each of the problems to which TEDA was applied. Each of these chapters describes how TEDA performed on the problem at hand and details how TEDA was adapted to this problem and any improvements that were made to TEDA in the process. These problems were chosen because they each introduce different challenges. As TEDA falls into the category of hybrid algorithms, a domain which focusses on devising algorithms that are suitable for a wide range of problems [96], it is essential that TEDA be tested on problems covering several different areas and characteristics. A brief description of these four classes of problem is as follows:

1.2.1 *Cancer Chemotherapy Scheduling, an Optimal Control Problem*

In the chemotherapy problem [88] the objective is to decide on a drug treatment schedule for a patient undergoing chemotherapy. This schedule should ensure that the cancerous tumour is minimized while the patient receives doses of potentially harmful drugs as infrequently as possible. Using this problem the initial development of TEDA is discussed. This process was carried out in two steps. First, a version of FDC was adapted that uses a probability distribution built from multiple parents instead of just crossover and mutation from the information in two parents. Next, the process of transitioning from true FDC to this EDA based technique was developed. Through this problem we also explore variants of TEDA where the EDA distribution is learnt incrementally, using techniques based on the EDAs, Population Based Incremental Learning (PBIL) and the Compact Genetic Algorithm (CGA), rather than being built afresh each generation.

1.2.2 *Network Routing*

Finding minimal cost routes across a landscape is an abstraction of problems that are important in several fields. For example, in mobile and internet routing it is important to find the optimum route through a network that will enable a packet to pass between points in as little

time as possible. In this type of problem each link in a network has a cost associated with it in terms of how long a packet is expected to take to travel through it [1]. As a route may pass through any number of nodes, it is natural to represent this problem as a variable length integer string. In applying TEDA to this problem we adapt it from an algorithm designed for chromosomes encoded as fixed length binary strings to one able to work with variable length strings containing a more complicated alphabet of genes. Encodings that are different to the traditional binary string are now popular in evolutionary computing as, despite being more difficult to implement, there is often a performance benefit when the encoding better matches the nature of the problem [71]. For this reason, adapting evolutionary algorithms such as TEDA to different types of encoding is important if they are to be useable on a wider range of problems.

1.2.3 *Feature Subset Selection*

In this problem TEDA is tasked with finding out which features from an available set are most effective at classifying samples. Through this problem the scalability of TEDA is demonstrated. Furthermore, as features are frequently related, we explore how it might be extended into a multivariate EDA that models these relationships. We test TEDA with three different datasets which all have large and noisy feature sets, initially containing between 500 and 20,000 features, most of which are redundant or misleading [56]. This problem therefore allows us to demonstrate the usefulness of TEDA when applied to much larger chromosomes than those used in previous problems, which have had a genome size of just 100 in the case of chemotherapy and 50 in the case of routing.

Another area that is explored through both the feature selection problem and through the k-trap problem is how TEDA might be developed into a multivariate algorithm. As extremely large chromosomes have previously been considered impractical for complex multivariate EDAs [69], by developing TEDA into an algorithm that transitions into a multivariate EDA after it has identified good, small, feature subsets we ask whether TEDA will enable multivariate EDAs to be used for problems where previously they have been unsuitable.

1.2.4 *Overall Structure*

This thesis proceeds with a general background chapter which introduces EAs and specifically discusses GAs, EDAs and previous works on hybridising these techniques. It also elaborates on key concepts such as selection, crossover, the importance of maintaining diversity and multivariate optimization. Chapter 3 describes the problems to which TEDA has been applied. Following this, Chapter 4 discusses the methodology that we have used. In this chapter TEDA is

explained in detail and we also describe the conditions under which TEDA, and the algorithms which it was compared against, were tested.

The remaining chapters detail the development of TEDA step by step and demonstrate how it performs on the four problems listed above. Chapter 5 describes how TEDA was developed from FDC through the chemotherapy problem. Chapter 6 describes how TEDA was applied to variable length chromosomes through the routing problem. Chapter 7 demonstrates how well TEDA performs on the much larger feature subset selection problem. Chapter 8 demonstrates how we have developed TEDA into a multivariate algorithm through the feature subset selection problem and the K trap problem. The thesis concludes with a review of the areas where TEDA was effective and those where it was less effective and discusses future work.

2

BACKGROUND

Evolutionary algorithms are a category of metaheuristics and so it is worth, first of all, explaining what is meant by a metaheuristic. A general definition is that a metaheuristic is a technique for choosing heuristics. A heuristic is a process that is used to solve a particular problem whereas metaheuristics operate at a higher level, choosing the appropriate heuristic for any problem to which they are applied [125]. They stochastically explore the set of possible solutions to a problem, with each solution being a heuristic. Because of this they are said to explore a *search space* of heuristics. These search spaces are often large and so exhaustive search, visiting every point in the search space, is often not practical. Additionally, there is not usually a mathematical method that guarantees that an ideal solution will be found. These problems are often *NP-Hard*, a term that describes problems for which a deterministic method of finding a solution in polynomial time is not known. For this reason, metaheuristics use randomness to explore the search space. An additional difficulty comes from the fact that these problems may have *local optima*, these are solutions which are preferable to all solutions which are close to them in the search space but are not close to being the best solution available, the *global optimum*. There is a danger that when a local optimum is discovered a metaheuristic technique will be unable to leave the vicinity of this local optimum to find the global optimum as any move away from it will appear to produce worse results. In general, metaheuristics do not guarantee that an ideal solution will be found but they are often the only means of finding solutions that provide satisfactory results. It is said that for many practical problems the objective is not to find the ideal solution but simply to find a solution that is "good enough" in a fast and cost effective manner [126].

2.1 EVOLUTIONARY ALGORITHMS

The term *Evolutionary Algorithm* refers to a set of techniques in metaheuristics in which problems are solved by randomly generating a population of possible solutions, combining the most effective ones, and repeating this process until a satisfactory solution is reached or until a time limit has expired. The inspiration for these techniques comes from the natural world where selection, breeding and mutation drive the evolution of a species. Each solution is encoded as a *chromosome* and each element within this chromosome is referred to as a *gene*. Each gene has an associated value, usually from a pre specified range. A possible value is referred to as an *allele*. An encoded solution is considered to be a *genotype* and the actual

physical realisation of the genotype is the *phenotype*. The encoding is essentially a method of mapping from phenotype to genotype.

The main steps of an evolutionary algorithm are described below [83].

1. Initialization: An initial population of chromosomes is generated randomly, satisfying certain constraints. This population need not be completely random as it may be seeded with known fit solutions.

2. Scoring: Potential solutions are applied to a problem specific task and then awarded a fitness score- a measure of quality describing how well they completed this task. This process is referred to as *fitness function evaluation*.

3. Selection: A number of fit solutions are selected in some way from the population. These selected solutions can be described as *parent* solutions.

4. Recombination: New solutions, *child* solutions, are generated from information available from the parent solutions.

5. Replacement: The new solutions are used to create a new population, the child population. This population may be made up entirely of the new solutions generated in step 4 or it may also contain some solutions that are copied directly from the previous population without any modification.

6. Steps 2 to 5 repeat until a stopping criteria is met, this may be after a fixed amount of time or a fixed amount of fitness function evaluations have been carried out or until a solution of a certain level of fitness has been found. How fit the fittest solution in the population is at each point in the process and how quickly a solution of a given level of fitness is found are the main criteria used to assess evolutionary algorithms.

EAs have several advantages over traditional optimization approaches [83]. A key advantage is that they can be applied to a new problem with minimal effort as little information about the search space is required. This also means that they can be applied to problems which are not fully defined. They are especially useful for problems that are said to be difficult problems, for example by having a highly multimodal landscape.

They do, however, also have several disadvantages. Firstly, they do not guarantee that the global optimum will be reached. They also often require the tuning of several parameters in order to work well. This necessity somewhat reduces the natural ability that EAs have of being easy to apply. This is because a human will need to either tune these parameters through trial and error or to use any knowledge that they may have of factors such as whether the fitness landscape for this problem is likely to be deceptive or multimodal. EAs can also be computationally expensive. Despite these disadvantages, EAs have shown good results on real problems, allowing the field to grow exponentially [83].

The three main varieties of EA are Genetic Algorithm (GA), Evolutionary Strategies (ES) and Evolutionary Programming (EP). Developed by Holland in 1975 [66], GAs were the original EA and the closest to evolution as it occurs in nature as new solutions are bred from two parents and mutated slightly. EAs, developed by Rechenberg [106], are similar to GAs except that the process of generating new solutions may be more removed from natural evolution and may involve more than two parent solutions. EP was pioneered by Fogel [42]. In EP chromosomes consist of a string of symbols that act as instructions. This allows for the evolution of programs [83].

Of these only GAs and ES (specifically EDAs) are relevent to this thesis. In the remainder of this chapter we will discuss key considerations in Evolutionary Computing, such as Schema Theory and Diversity, and will then move on to discussing GAs and EDAs in more detail.

2.2 SCHEMA THEORY

Central to the study of EAs is the concept of combining good substructures to produce high quality solutions. In 1975 John Holland [66] carried out research into the ways in which good substructures survive and spread across the population during the execution of an EA. A high quality substructure is referred to as a *building block* and can be described through a *schema*.

Assuming fixed length binary strings of length 8, an example of a typical schema would be **00**1*. The "*" symbols denote wildcards, these are values within the string which are not important for this schema. A schema S is said to have an order that is denoted as $o(S)$ and a defining length, denoted as $\delta(S)$. The order is the number of fixed (non wildcard) indexes within the schema, in this case 3, and the defining length is the distance from the first fixed index to the last, in this case 5. A schema also has a fitness. This is the average fitness of all solutions which contain this schema.

Evolutionary algorithms have the ability to process a large number of schemata simultaneously. An individual solution contains a large number of schemata, a maximum of 2^l where l is the length of the solution, therefore every time a solution is selected for breeding 2^l schemata are also selected. As the length of solution increases the number of schemata within it increases exponentially. This phenomenon has lead to the development of the *Schema theory*. As explained by Michalewicz in [91], this theory predicts that good, low order, schemata will receive progressively more trials from one generation to the next as the solutions that contain them proliferate throughout the population. This is a strength of Evolutionary Computing (EC) as the length of time that it would take to individually assess the fitness of these schemata is naturally a lot longer than it takes to assess them in this implicitly parallel manner. Holland describes how the large number of schemata give an EC a "broad base", enabling it to escape local optima.

It is therefore an important consideration in the development of any EC that the spread of schemata is not disrupted. This is a concern that has been much discussed in evolutionary computing. Whitley, for example, explains how one and two point crossover have a high probability of disrupting schemata with a long defining length but that Uniform Crossover (UC) is relatively more disruptive for schemata of all lengths [134].

2.3 MAINTAINING DIVERSITY

One of the most important requirements of any EA is maintaining population diversity to avoid premature convergence. Premature convergence happens when most solutions within a population are close to a local optimum and so the diversity needed to move away from this optimum is not available in the population. The complex np-hard problems to which evolutionary algorithms are usually applied are frequently multi-modal. They have a large number of optima scattered across a complex landscape [6]. A search process will tend to be drawn to whichever optimum happens to be closest, regardless of whether or not this is close to the global optimum. In order to allow the population to escape from a local optimum it is necessary to ensure that it maintains a diverse spread of different solutions [130]. The aim of any EA is that it should be able to converge on a globally optimal solution without prematurely converging on local optima.

Most explanations for why diversity is lost within a population concern either selection or gene flow. A selection method that nearly always selects the fittest individuals in the population will quickly fill the population with clones of these individuals. The way that genes flow within a population is connected to crossover and other recombination operators and may lead to a loss of diversity as, in most situations, each solution may breed with any other solution. This means that a high quality schema may spread until it is present in every solution in the population, allowing for no rival schemata to exist [130].

Solutions have been suggested to both of these possible causes of a loss of diversity. One of the main methods asssociated with maintaining diversity in selection, crowding, is discussed in the section of this chapter on selection (Section 2.5). Other approaches include the Diversity-Control-Oriented Genetic Algorithm developed by Shimodaira [114]. This is a method that ensures diversity by penalizing solutions that are too close to other fit solutions in the population. Solutions are given a survival probability based on their Hamming distance to the fittest solution in the population. Ursem [130] developed an algorithm that measures diversity through the "distance to average point" measure and uses a threshold to classify it as either high or low. A technique designed to increase diversity, mutation, is carried out when diversity is low whereas techniques that decrease diversity, selection and crossover, are carried out when diversity is high.

2.3.1 *Exploration and Exploitation*

Related to the concept of population diversity is the concept of *exploration* and *exploitation*. An explorative algorithm will move around the solution space relatively freely whereas an exploitative algorithm will only investigate solutions that are likely to cause an improvement. An algorithm should be capable of both of these approaches. In order to fully explore the search space, instead of just discovering the local optimum closest to the initial state of the population, it needs to be explorative. This means that when many solutions in a population are close to an optimum there should still be other solutions and the option of generating other solutions that explore parts of the search space some distance away from this optimum and so potentially find better optima. On the other hand, an algorithm should be exploitative enough so that when a solution is close to an optimum, it is able to follow the gradient of the fitness landscape to come as close as possible to this optimum.

It is often necessary that early on in the evolutionary process an algorithm is explorative as this is the point in time when it is least likely to be near the global optimum and most likely to be at risk of converging on any local optima that are near. Later on it is often necessary to use an exploitative approach as, by this stage, it is more likely to be approaching the global optimum [86].

2.4 GENE INTERDEPENDENCY

Another key concern in evolutionary algorithms is the concept of linkage, or gene interdependency [112]. These terms refer to the fact that, in many problems, genes cannot be assumed to be independent. Each gene is likely to be influenced by other genes within the chromosome. A building block, as introduced in the on schema theory (Section 2.2), may consist of a set of closely related genes and competent evolutionary algorithms should be able to discover these highly correlated building blocks [32]. Sets of related genes are often referred to as *cliques*. Muhlenbein described an example of this problem in [93]. In a problem that he referred to as the *Plateau problem* the population is divided into chunks and each gene's usefulness is dependent on the values of the other genes within its chunk. If we assume that each chunk has a length of 3 then the fitness of solution x is calculated through Equation 2.1.

$$f(x) = \sum_{i=1}^{m} g(x_{3i-2}, x_{3i-1}, x_{3i}) \tag{2.1}$$

In this equation $g(x_1, x_2, x_3)$ is equal to 1 only if x_1,x_2 and x_3 have a value of 1, otherwise it is equal to 0. Each gene is therefore only useful when its two neighbours are set.

This problem is made more difficult still by introducing a deceptive element to it, so that a gene may, with the same value, be either detrimental or beneficial to the fitness of a solution

depending on the values of other genes. An example of such a problem is found in the work of Harik [62]. The problem that he describes, *k-trap* problem, is similar to the plateau problem in that the chromosome is divided into sections and each section only achieves optimal fitness if all of its constituent genes have a given value.

To understand the k-trap problem, let us assume binary encoded chromosomes where every gene has a value of 0 or 1 and where an individual section achieves optimal fitness if all of its genes have a value of 1. The deception arises because, in cases where solutions have some but not all of the genes set to 1 within a given section, the fitness of that section will actually become worse as more of the genes within it are set to 1. If all but one of the genes within a section are set to 1 the fitness of this section will be 0 (assuming a maximisation problem where 0 is the lowest possible fitness). Each section is referred to as a "trap". The complete problem is a concatenation of m traps each of length k. For each individual subproblem $trap_i$, the fitness of a binary vector x, $f(x, trap_i)$, is calculated as detailed in Equation 2.2.

$$f(x, trap_i) = \begin{cases} k, & \text{if } \sum x_i = k. \\ k - \sum x_i - 1, & \text{otherwise.} \end{cases} \tag{2.2}$$

For a problem such as this an algorithm that does not take into account linkage information is likely to drive towards setting all genes to 0. In the global optimum for this problem all genes have a value of 1 and so such an algorithm would drive in the opposite direction of the global optimum. Harik proved this to be the case, and demonstrated that a GA using uniform crossover will actually lead to a decrease in fitness over time as cliques are disrupted [62].

In real word problems the situation is even more complicated as it is not known in advance whether a given problem features a high degree of interdependency or not and it is not known which genes are related or whether a given clique occupies a contiguous section or contains genes scattered across the whole chromosome [112]. For this reason much research has been done on how the linkage learning capabilities of evolutionary algorithms may be improved. Linkage learning techniques in EAs generally fall into two categories [112]. Some methods, such as the messy genetic algorithm [51] order the genes within a chromosome so that closely related genes are close to each other to reduce the probability of cliques being disrupted. Other methods attempt to explicitly model linkage and incorporate linkage information into a probability model. These methods include multivariate Estimation of Distribution Algorithms and shall be discussed in more detail later.

2.5 SELECTION

Selection determines which solutions from one generation are likely to be used to produce the following generation and how many times each solution is likely to be used [58]. Selection is

important in relation to Holland's schema theorem as it is one the main determinants of how many copies of each building block survive from one generation to the next [66] [50]. Each selection technique is associated with a given *selection pressure*. If the fittest solutions are nearly always selected then the selection pressure is high. Methods that select solutions relatively randomly are described as having a low selection pressure. Selection is critical in determining how the algorithm will behave [58] as an algorithm with overly high selection pressure will tend to repeatedly select the same small set of very fit solutions and so ultimately fill up the population with clones. Conversely, if the selection pressure is too low then this may slow down the rate of convergence [130]. A high selection pressure is associated with exploitative methods whereas a low selection pressure is associated with explorative methods. Several selection techniques are discussed below [10] [58]:

- *Fitness Proportionate Selection* involves selecting individuals probabilistically in proportion to their fitness relative to the rest of the population. There exist several different ways that this may be accomplished. One well known and relatively simple approach is roulette wheel selection, where a virtual roulette wheel is spun in which each segment corresponds to one solution and has a width which is determined by the relative fitness of that solution. This method has a relatively low selection pressure as the proportion of the roulette wheel taken up by a single fit solution is likely to be small.

  For example, assume that in a population of size 100 the fittest solution (x_f) has a fitness of 2 and all other solutions have a fitness of 1. The probability of x_f being selected will be $2/101 = 0.0198$. The probability of the fittest solution in the population being selected in this situation is very low, even though this solution has double the fitness of any other solution. In fact, Hancock [58] goes as far as to suggest that the level of noise associated with this method and the high probability of sampling errors whereby unfit solutions are selected makes it unsatisfactory. Alternative approaches to fitness proportionate selection such as windowing, sigma scaling and linear scaling work by calculating the number of offspring that each solution is expected to generate based on its fitness and use this to control how many to actually generate.

  In general, fitness proportionate selection is the original method used in Genetic Algorithms and it has the advantage that it is relatively simple [58]. It does, however, have the disadvantage that it can be sensitive to how the fitnesses have been scaled. For example, suppose that there are two individuals in the population with fitnesses of 1 and 2. In this situation the fitter individual has twice the probability of being selected as the less fit individual. If, however, the fitness function is such that these individuals have fitnesses of 11 and 12 then the fitter individual is only slightly more likely to be selected than the less fit individual even though the latter situation may simply be the result of adding 10 to every fitness score. In general, selection pressure in these

approaches can be high or low depending on how fitness is distributed. If there is a small number of individuals in the population that are much fitter than the rest of the population then these will be heavily selected and the population will converge towards these individuals. If, however, the population is relatively evenly distributed in terms of fitness, then fitness proportionate selection will be a much more explorative process [58].

- *Ranking* was an early approach, developed by Baker [7], to overcome the sensitivity that fitness proportionate selection has to the scaling of the fitness function. Individuals are selected according to where they rank in a population that is ordered by fitness rather than according to their actual fitness scores. After the population has been ranked according to fitness each individual is given a dummy fitness score that is inversely proportional to its rank. In one variant of this approach, linear ranking, the fittest individual in the population is given a fitness score s, a value between 1 and 2, the least fit individual is given a fitness score of $2 - s$ and all other individuals are given fitness scores between these two values on a linear scale.

  In order to allow for higher selection pressures, something which cannot be done with linear ranking without reducing the probability of the least fit solutions being selected to 0, a second form of ranking, exponential ranking, is sometimes used. In this method the fittest individual is given a fitness of 1, the second fittest individual is given a fitness equal to the tunable parameter s, usually 0.99, and all remaining individuals are given a fitness of s^i where i is the solution's ranking value. This will be 2 for the third solution, 3 for the fourth solution and so on. Ranking methods have been shown by Hancock [58] to allow for careful control of selection pressure although the actual fitness of a solution may be useful information which is lost in this approach.

- In *Tournament Selection* a group of individuals are picked at random from the population and the fittest individual in this group is used for breeding. The size of the group is referred to as the tournament size. The selection pressure can be tuned by changing the tournament size. As large tournaments are more likely to contain the fittest individuals in the population, the selection pressure is higher when the tournament size is higher. Tournament selection has the advantage that each tournament can be processed in parallel and so the method is well suited to take advantage of the parallel capabilities of evolutionary computing. In some applications, such as games, the fitness of solutions is determined by making them compete against rival solutions and so tournament selection is a natural choice. However, Hancock showed that tournament selection suffers from a similar drawback to roulette wheel selection as the process is noisy and there is no guarantee that the solutions selected will not be primarily unfit solutions [58]. This will be the case if only unfit individuals are selected to form the tournament.

- *Best Selection*, also known as *Top-n* or *Truncation Selection*, involves sorting the population in order of fitness and simply selecting the fittest $n\%$ of the population. This method is the most exploitative selection method as it guarantees that only the fittest solutions will be selected. It risks reducing diversity yet can cause locally optimal solutions to be found rapidly. This method, referred to as (μ, λ), is one of the most popular methods used in evolutionary strategies and evolutionary programming. A slight variation, denoted as $(\mu + \lambda)$, is also used. In both cases μ is the number of parents selected and λ is the number of children generated. The only difference is that in (μ, λ) the new children generated completely form the new population whereas in $(\mu + \lambda)$ both the children and the parents together form the new population. These techniques are discussed by Hancock in his survey of selection methods [58] where he predicts that if (μ, λ) has the value of $(100, 200)$ then the fittest solution in the population will completely take over the population in only $\log_2(100) = 7$ generations on average. This underlies how high the selection pressure is with this approach.

It should also be mentioned that there exist more explicit techniques for maintaining diversity through selection regardless of the selection pressure. A primary example of these techniques is crowding [85]. Crowding is based on the observation that in nature similar individuals are more likely to compete than dissimiliar individuals. In algorithms that use crowding a proportion of the population, G, is selected each generation and used to generate new solutions. These new individuals will be added to the population and will replace a set of less fit individuals G'. For each individual in G, individual G_i, a set of existing individuals is selected and the one nearest to G_i is chosen as the one to be replaced. This prevents the population from filling up with similar individuals.

2.6 REPLACEMENT

When new solutions have been produced they are used to build a new population. How they are used is referred to as the replacement strategy. In classic genetic algorithms this is simply a matter of generating n new solutions, where n is the size of the population, and using all of these new individuals and none of the old solutions to form a new population. This is referred to as *generational replacement*. In some cases this may be modified so that a small part of the previous generation is copied into the new generation unaltered. An alternative to generational replacement is used in *steady state* genetic algorithms [134]. In these algorithms, for each new solution that is generated a solution is selected from the population to be deleted. The new solution then replaces the deleted solution in the population. Next time a new individual is generated any solution in the population may be used as a parent solution, including the individual that has just been added.

Steady state style replacement may, in some situations, lead to faster convergence than generational replacement. This is because, as soon as a new solution is generated, it is available to be used as parent. This means that newly discovered building blocks can quickly be integrated into the population [132]. Steady state replacement does have the disadvantage that it may lead to a loss of diversity as the same individual may be selected repeatedly in a short space of time. In general it can be said that steady state GAs have a higher selection pressure than generational GAs because individuals are forced to compete for selection with new individuals that are likely to be fitter than they are.

When steady state replacement is used there are a number of different ways that the individual to be replaced may be chosen. A popular strategy is to simply replace the worst individual in the population [134]. Alternatively, a random individual may be selected for replacement. Replacing the worst individual creates a higher selection pressure than replacing a random individual [134] [50] and so this strategy may lead to fast convergence but may also lead to a premature loss of diversity. There are methods that explicitly ensure diversity such as the crowding method mentioned in the previous section where individuals that are similar to the new individual are chosen to be replaced regardless of their fitness [84].

When generational replacement is used it is not always the case that the entire population is replaced with new individuals. It was mentioned in the previous section that the $(\mu + \lambda)$ strategy is popular in genetic programming where the new population is formed from μ parents copied directly from the old population and λ newly created children. This strategy has a very high selection pressure as good individuals can quickly dominate the population. Alternatively, just one individual, the best individual in the population, may be copied directly into the new population. The rest of the new population will consist of new individiuals. This strategy is referred to as *elitism*. This strategy ensures that the best individual is not lost [131].

2.7 GENETIC ALGORITHMS

Genetic Algorithms (GAs) are the original evolutionary algorithm that Holland developed in 1975 [66]. GAs follow the same process as other evolutionary algorithms, described in Section 2.1, but are distinguished from these other techniques primarily by the recombination operator and by the fact that the *mutation* operator is usually used. The GA recombination operator is referred to as *crossover* and it consists of producing new solutions that are usually a combination of 2 parent solutions. Mutation is the process of very slightly changing new solutions to ensure that diversity is maintained within the population. Mutation ensures that genetic algorithms are able to search the complete search space as otherwise only genes that already exist within the population may appear in new solutions. The GA process is as follows:

1. Initialization of the population.

2. Scoring the population.

3. Selection of parents. Two are normally used but more than two can be used.

4. From these two parents two new children solutions are generated through a two-step process:

   Crossover.

   Mutation.

5. Replacement: In most cases new children are generated until a new population has been formed.

As with other evolutionary algorithms steps 2 to 5 repeat until a stopping criteria is met.

2.7.1 *Crossover*

Crossover is the process in GAs that allows information to be combined from fit solutions. Two or more parent solutions are selected and children solutions are generated that are a combination of these two parents [49][123]. Choosing the most appropriate crossover settings for a particular problem is, however, a non trivial task. Crossover is the process that makes GAs so effective as it allows fit building blocks to spread throughout the population and so the main criteria for any crossover technique is that it should effectively spread building blocks without disrupting them.

There are two main decisions that need to be made. A crossover rate needs to be decided on and a crossover technique needs to be chosen. The crossover rate, P_c, is the probability that crossover should take place. For each new individual a decision is made whether it should simply be copied from the previous population or whether it should be produced through crossover. For each solution the probability of crossover taking place is P_c. Higher values of P_c are common as this allows good solutions to be produced at a faster rate. However, if P_c is too high it may cause good solutions to be disrupted by crossover at a faster rate than they can be selected. P_c is normally set at between 0.5 and 1 [119]. The crossover technique used is dependant on issues such as the system of encoding used, the need to maintain diversity and the presence of gene interdependencies. A discussion of the most prominant techniques will now follow.

2.7.1.1 *One Point Crossover*

In this technique a gene is chosen at random from somewhere in the chromosome. The index of this gene is used as the point at which the chromosome is split into two sections. Two children are produced, one child will have the front section of one parent and the rear section of the other and in the other child this will be reversed.

The following example illustrates this process:

$\mathtt{Parent}_A$: 01001110

$\mathtt{Parent}_B$: 00010001

split the parents at gene 3:

$\mathtt{Parent}_A$: 010 and 01110

$\mathtt{Parent}_B$: 000 and 10001

recombine to produce two new solutions:

$\mathtt{Child}_{AB}$: 01010001

$\mathtt{Child}_{BA}$: 00001110 (2.3)

The problem with this method is that genes next to each other will only be separated if the crossover point falls between them whereas genes at opposite ends of the chromosome are almost guaranteed to be separated. In terms of schema theory it can be said that one point crossover has a high probability of disrupting schemata with a long defining length.

2.7.1.2 *Two Point Crossover*

This is similar to one point crossover except that two crossover points are used instead of one. The section swapped is the section between these two crossover points. This reduces the probability that genes at the edges of the chromosome will be separated but there is still a bias towards disrupting longer sequences and keeping shorter sequences intact. It can be said that there is still a bias towards disrupting schemata with a longer defining length although this is less the case than for one point crossover.

The following example illustrates this process:

$\mathtt{Parent}_A$: 01001110

$\mathtt{Parent}_B$: 00010001

these two parents split at genes 2 and 5:

$\mathtt{Parent}_A$: 01 and 001 and 110

$\mathtt{Parent}_B$: 00 and 010 and 001

Recombine to produce two new solutions:

$\mathtt{Child}_{ABA}$: 01010110

$\mathtt{Child}_{BAB}$: 00010001 (2.4)

2.7.1.3 *Uniform Crossover*

In this technique each gene is considered individually and a random decision is made, with equal probability, whether to use the gene from one parent or the other in the child solution.

A second child can be generated which, for every gene, uses the gene from the parent that was not selected. This technique is therefore similar to one or two point crossover except that any number of crossover points might be used, up to a maximum equal to the length of the chromosome. This ensures that there is no bias when it comes to dividing up pairs of genes, unlike with the other techniques.

The following example illustrates this process:

$$
\begin{aligned}
&\mathtt{Parent}_A : 01001110 \\
&\mathtt{Parent}_B : 00010001 \\
&\text{Use the gene from parent : } A\ B\ A\ B\ A\ A\ A\ B : \\
&\mathtt{Child}_A : 00001111 \\
&\mathtt{Child}_B : 01000000
\end{aligned} \tag{2.5}
$$

2.7.1.4 *Building Block Disruption*

It has already been mentioned that one point crossover has an inherant bias towards breaking schemata with a long defining length, that this is less of a problem with two point crossover and less still with uniform crossover, where the probability of disrupting building blocks is independent of their defining length. In general, where less crossover points are used the probability of disrupting each schema becomes more even and less biased based on the length of the schema.

To understand why this is the case, suppose that we have an 8 bit chromosome 11000101 and that different crossover techniques are used to combine it with the chromosome 00111010, a chromosome with which it shares no genes. This means that in every situation where crossover takes place within a building block that building block will be disrupted. If one point crossover is used then the building block defined by the schema 1******1 will be disrupted regardless of where the crossover point is placed. We can say that any building bock with a defining length of 8, equal to the length of the chromosome, is guaranteed to be disrupted. The building block defined by the schema 1*****0, including genes 1-7 but not gene 8, will be spared disruption only if the crossover point lies between gene 7 and gene 8. In contrast, any building block defined by a schema with a defining length of just 2 will only be disrupted if the crossover point lies between the two genes that define it. When the number of crossover points increases, the relationship between defining length and the probability of schema disruption is complicated by the fact that if two defined genes within the schema are separated by two crossover points then both parts will be taken from the same chromosome. If these are the only two defined genes in the schema then it will therefore survive undisrupted. In two point crossover, for example, 1******1 is likely to survive whereas 1***0***1 is likely to be disrupted.

De Jong formalised this relationship as follows [72] [118]. Suppose that H_k is a schema of length k with only two defined genes, formally a schema of order 2. The probability that

H_k will be disrupted when crossover takes place at n points can be obtained mathematically through the following equations. The general rule is that if the number of crossover points between the first defined gene in a schema and the last is even then the schema will not be disrupted. Because of this, the probability of disrupting H_k using n crossover points is equal to the probability that the number of crossover points between the first and last genes in H_k is even, as stated in Equation 2.6. This probability is calculated through Equation 2.7.

$$P(n, H_k) \leqslant P_{2,even} \tag{2.6}$$

$$P2, even(n, L, L_1) = \sum_{i=0}^{\lfloor \frac{n}{2} \rfloor} \left[\frac{L_1}{L}\right]^{2i} \left[\frac{L - L_1}{L}\right]^{n-2i} \tag{2.7}$$

In Equation 2.7 L and L_1 define the first and last genes in schema H_k respectively. $\lfloor \frac{n}{2} \rfloor$ represents the number of different combinations of n points that exist when only those combinations containing an even number of points are considered. The second term, $\sum_{i=0}^{\lfloor \frac{n}{2} \rfloor} \left[\frac{L_1}{L}\right]^{2i}$ gives the probability of placing an even number of crossover points between L and L_1 and the final term is the probability that the remaining points do not lie between L and L_1.

The outcome of this equation for different defining lengths and different values of n was plotted by Spears to produce the graph presented in Figure 2.1 [118]. In this graph it can be seen that, with one point crossover, the probability that an even number of points lie within the bounds of a given schema is high only when the defining length is so low that there is a high probability that no crossover points lie within the schema, as the only other possibility is that 1 point falls within the schema. For higher, odd numbers of crossover points the probability of an even number of points falling within a schema and the schema remaining intact remains comparitively steady, but still decreases dramatically as the schema length approaches the length of the chromosome. Overall, it appears that the least disruptive crossover techniques use a low and an even numbers of crossover points. Two point crossover was shown through this study to be the least disruptive to building blocks. Although uniform crossover avoids bias it was found to be relatively disruptive. Syswerda [124] argues that uniform crossover compensates for this with its ability to recombine schemata that it has disrupted.

2.7.2 *Fitness Directed Crossover*

FDC is a GA crossover technique developed by Godley et al [48] [47] in which the control point targeting technique that forms the basis of TEDA was initially developed. In FDC two parents are used to derive a target number of control points and new solutions are then produced with that number of control points set.

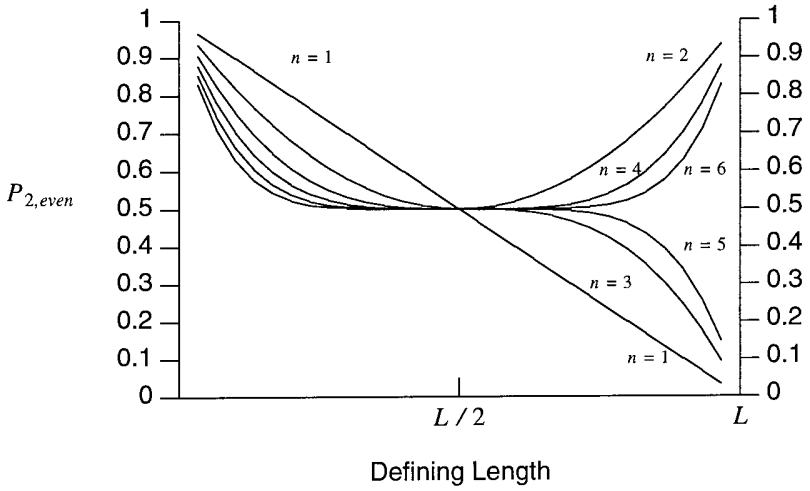

Figure 2.1: n point crossover schema disruption [118]

FDC arose from a study of optimal control problems. In these problems a process takes place over a fixed period of time, for example 100 days. A chromosome represents this period of time with each gene corresponding to a single day. The decision that needs to be made is on which days should an action, or *intervention* take place. On days when an intervention does take place the corresponding gene will have a value of 1 or above, whereas on all other days it will have a value of 0. *Intervention* is the term used in optimal control literature for what is referred to as *control points* in this thesis and so, for consistency, the term control point will be used from now on. The question that arises from these problems is that, given a 10 gene problem for example, with the chromosomes detailed below, is useful information contained in the fact that Solution A has 5 control points whereas solution B has only 1 and that Solution A is also fitter than Solution B (assuming a maximising fitness function)? Can this information be used to produce effective new solutions?

$$\begin{aligned} \text{Solution}_A &: 1011110000, \text{Fitness} = 0.3 \\ \text{Solution}_B &: 0000100000, \text{Fitness} = 0.1 \end{aligned} \tag{2.8}$$

Godley [47] investigated various possible answers to this question. The three techniques that he explored before arriving at FDC were Directed Uniform Crossover (DUC), Targeted Intervention with Stochastic Selection (TInSSel) and Calculated Expanding Bin (CalEB). All of these approaches are based on a genetic algorithm and begin with two parents being selected from the population. Once these have been selected a target number of control points, I_T needs to be obtained. The original and simplest method of doing this, used in DUC, is to just create new solutions with the same number of control points as in the fitter of the two parents.

This has the drawback that there is a danger of producing a lot of solutions with the same number of control points and so losing diversity in the population. TInSSel and CalEB attempted to rectify this by setting I_T to a random value near to the number in the fitter solution through Equation 2.9. D_i is the gap in the number of control points between the fittest and the least fit individual in the population, $D_i = |I_1 - I_2|$ given that I_1 and I_2 are the number of control points in the fittest and least fit individuals in the population respectively. In all cases I_T is confined to a minimum of I_{min} and a maximum of I_{max}.

$$I_T = I_F - \frac{D_i}{2} + rand(D_i) \tag{2.9}$$

Once a method of calculating I_T has been decided on it is then necessary to decide in which order genes should be selected to be set. The approach used by all of these techniques is similar. In all cases the control points that appear in at least one of the parents are divided up into two sets:

- S_{dup} : The control points found in both parents.
- S_{single}: The control points found in only one parent.

This is motivated by the assumption that control points found in both parents are likely to be of a higher quality than those found in just one parent. To create a new solution with I_T control points, first of all control points from S_{dup} are selected at random and set in the new solution until either I_T has been reached or all control points in S_{dup} have been set. If I_T is larger than S_{dup} then control points are then chosen at random from S_{single} until I_T has been reached or all control points in S_{single} have been set. When a control point in S_{single} is set its value is picked from the parent which has a non zero value at that index. When a control point in S_{dup} is set then one of the two parents is selected at random to be the source of the value that this control point is then set to.

CalEB differs from the other two approaches in that control points are not selected completely at random from S_{single} but are selected in such a way as to be evenly distributed across the time series. The use of S_{dup} is a natural decision as it is close to the way that UC behaves. In UC any gene that has the same value in both parents is guaranteed to be set in the child solution. This is almost but not quite true when S_{dup} is used as only some of the control points in S_{dup} will be used if I_T is smaller than the size of S_{dup}. The alternative, selecting control points in a random order, would not be satisfactory as the relatively fit control points that are found in both parents might be ignored completely if I_T is reached before they have been selected.

Of these approaches TInSSel was demonstrated to be the most successful, performing better than CalEB and DUC as well as UC. The implication of this is that there is indeed useful information in the number of control points found in fit solutions that can be exploited and

that the best approach involves generating new solutions with a similar but not identical number of control points to the fitter parent solution so that diversity is maintained. These results provided support for the concept of targeting but Godley had a further question regarding how exactly the number of control points should be chosen. Would it not produce better solutions if this decision took into account the fitness of the parent solutions? The idea was that the number of control points should be driven further in the appropriate direction if the gap in fitness between the two parents was greater. This seems logical as the difference in control point numbers between two solutions of a very similar fitness is likely to not be important and may represent random fluctuation whereas the difference in control point number between solutions that have very different fitness values is more likely to represent significant relationships between fitness and the number of control points.

With this motivation FDC was developed. FDC uses the same manner of selecting control points as TInSSel but the method of calculating I_T was replaced with the FDC rule, as detailed in Equation 2.10.

$$I_T = I_F + (2T - 1)(I_1 - I_2)(F_1 - F_2) \tag{2.10}$$

In this equation the number of control points set in parent 1 and parent 2 is denoted as I_1 and I_2 respectively.F_1 and F_2 are the normalised fitnesses of the two parent solutions and I_F is the number of control points in the fitter solution. T is 0 for a minimisation problem and 1 for a maximisation problem. The fitnesses used are normalised against the population through Equation 2.11. In this equation the original fitness F is converted into a normalised fitness F_{norm} using the range of fitnesses in the population with F_{max} corresponding to the highest fitness in the population and F_{min} corresponding to the lowest.

$$F_{norm} = \frac{F - F_{min}}{F_{max} - F_{min}} \tag{2.11}$$

The effect of this rule is that I_T will be beyond the number of control points in the fitter of the two parents and that the gap in fitness between these two parents will determine how far beyond this value it will be.

To provide an example of how these techniques work, Table 2.1 shows the value that I_T will be calculated at using each of the techniques described here, given that the two parents used are those given as examples at the start of this section. I_1 and I_2 correspond to the number of control points in the fitter and the less fit parent respectively. F_1 and F_2 correspond to the fitness of the fitter and the less fit parent respectively. For FDC we assume that the fitnesses 0.1 and 0.3 have already been normalised.

In Godley's experiments [47], FDC, TInSSel, CalEB, DUC and UC were tested on two optimal control problems. A cancer chemotherapy problem, and a bio control mushroom farming

Algorithm	I_1	I_2	F_1	F_2	I_T	Explanation
DUC	5	1	0.3	0.1	5	$I_T = I_1$
TInSSel/CalEB	5	1	0.3	0.1	3-7	$I_1 - 2 + (0 \text{ to } 4)$
FDC	5	1	0.3	0.1	5.8 (rounded to 6)	$I_1 + (I_1 - I_2 = 4)(F_1 - F_2 = 0.2)$

Table 2.1: Examples of FDC and Other Targeting Methods

problem. FDC was shown to perform better than the other approaches in the chemotherapy problem and, in the bio control problem, it was shown to perform better than the other approaches when the mutation rate was high but was only partially successful using a lower mutation rate. As a hugh mutation rate introduces a lot of genetic material into solutions at random, it adds a significant amount of noise into the population. The method of choosing I_T used in FDC, which takes into account of solution fitness, appeared to be better able to cope with populations with a high degree of noise than the more random method used by TInSSel. This is because, with FDC, where two solutions that have a similar fitness contain different quantities of control points this will only have a significant effect on new solutions generated if they also have significantly different fitness values. Targeting should therefore only respond to true relationships between control point number and solution fitness. Random variation in the length of solutions, a more frequent occurrence in noisy environments such as a population that has been highly mutated, will not have a significant effect on the number of control points in new solutions. Godley pointed out that many real world problems feature a high level of genetic noise and so this suggests that FDC is the most effective method of carrying out targeting of the methods proposed in his thesis.

2.8 estimation of distribution algorithms

Estimation of Distribution Algorithms are Evolutionary Strategies which generate new solutions from a probability model which approximates the fitness function. EDAs arose from Genetic Algorithms but, unlike GAs, they replace the crossover and mutation operators with an averaging process that takes place from a set of fit solutions [112]. These parents are used to build the probability model and new solutions are sampled from this model with the intention that they should therefore be similar to the fittest solutions in the population [112]. In this thesis the set of parent solutions is referred to as the *breeding pool*. *Univariate EDAs* assume the independence of each gene whilst more sophisticated techniques, *Multivariate EDAs*, model relationships between genes.

The motivation behind EDAs is based on the fact that in a standard GA it is possible to predict the likely proportion of genes with a particular value in the population at a given time [8]. Given a generation g and assuming a population of fixed length strings, fitness proportionate selection and a simple pairwise crossover operator such as one point or uniform crossover, it is possible to predict the number of solutions in $g+1$ in which gene i has a value of j through Equation 2.12 [8].

$$\text{Proportion}(x_i = j) = \frac{\sum_{v \ni v_i = j \in P_g} \text{Fitness}(v)}{\sum_{v \in P_g} \text{Fitness}(v)} \tag{2.12}$$

In this equation $\text{Proportion}(x_i = j)$ is the proportion of individuals in the new population in which gene i has the value j. P_g is the old population, the population at generation g, $v \in P_g$ is a solution in that population and $v \ni v_i = j$ indicates that in solution v gene v_i has a value of j. This equation simply gives the proportion of solutions containing $v_i = j$ in the current population weighted according to solution fitness. The implication is that if it is possible to predict this without needing to carry out the selection, crossover and mutation operators associated with GAs then disposing of these operators and simply generating a population from this prediction would appear to be more direct way of generating a new population, without risking the inevitable sampling errors associated with the GA process. Although generating a new population directly through this equation is not likely to be useful (without selection there is no way of moving towards good solutions) the principle does form the basis for the many EDA techniques that exist.

There are several advantages that EDAs offer over GAs and these are explained below.

1. There are a number of advantages associated with the fact that in an EDA there is an explicit probabilistic model. The model can be seeded with whatever information already exists of the problem domain. Even if this information is limited and only part of a model can be constructed this way it can reduce computation time significantly. Furthermore, previously unknown information about the nature of the problem can be extracted from the model. Detailed information can be extracted regarding the process and rate at which an algorithm is able to solve a problem. This in turn can inform further exploration of which algorithms may work well on a problem [5].

2. Multivariate EDAs have the ability to take into account the interdependencies between genes [83]. As mentioned, a lot of the motivation behind the development of EDAs is that they are one of the two main approaches to linkage learning alongside GAs that have a reordering operator to ensure that related genes are close to each other. Linkage learning in this manner through GAs has proved difficult [61] and so EDAs offer advantages. Part of the difficulty with using permutation based GAs for linkage learning is that there is essentially a “race" between allele selection and linkage learning. If a building block

has converged then it is often no longer possible to learn linkage concerning the genes in that building block. We may, for example, have a binary problem with allele values 0 and 1 where the value of gene x_i depends on gene x_j in such a way that x_i should be 1 when $x_j = 1$ and 0 when $x_j = 0$. If this relationship is not spotted early on then a situation may arise in which almost every solution in the population has $x_i = 0$. Allele selection has essentially won the race as there are not enough solutions where $x_i = 1$ for the algorithm to discover that, when $x_j = 1$ this setting is actually preferable. Another problem is that new crossover techniques need to be developed in order to cross over chromosomes that may be in different orders [61]

3. They have fewer parameters that need tuning than GAs [113]. In GAs there are several parameters that need to be selected, such as mutation and crossover rates, that are not relevant in EDAs. As choosing the best values for these parameters can become an optimization task in its own right [53] this can make EDAs easier to use [112].

4. They can be more rigorously theoretically analysed than genetic algorithms [83] [112]. There is a long history of theoretical research into GAs [66]. However, predicting their rate of convergence remains a challenge and the upper bound in how much the actual behaviour of a GA differs from any model constructed remains elusive [142]. By comparison, the theoretical study of EDAs has a much shorter history and yet there have been a number of papers published which predict the rate of convergence for various types of EDA, including the studies of Zhang [142], Grahl [52] and Muhlenbeim [92]. EDAs are therefore sometimes used for their potential to be theoretically analysed [112].

There are, however, several disadvantages associated with EDAs as follows:

1. EDAs can become very complex and computationally expensive. Producing a probability model that accurately describes the population would involve analysing every individual in the population. This would be inhibitively expensive and so any technique used will inevitably be an approximation of the true fitness function [83].

2. EDAs also become complicated when used to solve real valued problems. This situation complicates the question of what the probability distribution should look like as a single probability value for each gene will not suffice [16]. We also have to carefully consider how the genome should be encoded. If several genes are used to encode each continuous value then interdependency becomes a complicated issue as these genes are now dependent on each other in addition to the possibility of being dependent on variables from another part of the chromosome, themselves encoded as a cluster of genes [15].

3. A further disadvantage in EDAs is their tendency to prematurely converge [102] [140]. Grahl [52] proved that the area of the search space that can be visited by a standard EDA,

specifically UMDA using truncation selection, is bounded. The bounds is a function of n, the number of individuals selected each generation, and if n is chosen incorrectly then there will be feasible solutions that can never be visited, as the variance will reduce to 0 and so the population will completely converge before the area of the search space in which these solutions are found has been reached. If the global optimum lies within this region then it cannot be found. Because of this, whether or not an EDA converges before it has found the global optimum is dependent both on how far the initial population is from this optimum and on n. In general, an EDA that attempts to build a model from an unreliable population may learn spurious dependencies [20] which may lead to premature convergence.

The main workflow in EDAs is as follows:

1. Initialize a population.
2. Evaluate the population.
3. Select a breeding pool, B.
4. Build a Probability model ρ from B.
5. Sample from ρ to produce a new population.
6. Repeat steps 2 to 5 until a stopping criteria is met.

EDAs derive their distinct identity from steps 4 to 5 of this process. The remaining steps are the same as in other EAs. Most research into EDAs concerns the development of different methods of building and sampling from a probability distribution.

2.8.1 *Univariate EDAs*

The simplest and most computationally efficient EDAs assume that all genes are independent. In this section three specific examples of univariate EDAs will be discussed in detail. In all of these examples we assume a population of fixed length binary chromosomes.

2.8.1.1 *The Univariate Marginal Distribution Algorithm*

The simplest EDA, the Univariate Marginal Distribution Algorithm (UMDA), uses a probability model that consists of a vector containing the marginal probabilities of each gene. UMDA is directly based on the rule for predicting probabilities described above (see Equation 2.12). The only differences are that, instead of using the whole population, a selection of fit individuals constituting the breeding pool are selected and the probability of a given gene is obtained simply by counting the number of individuals containing that gene rather than by taking

into account fitness [8]. If we assume a population of fixed length binary strings then, from these individuals, the probability ρ_i that the value of gene x_i is 1 is calculated through Equation 5.2 [112][83]:

$$\rho(x_i = 1) = \frac{1}{n} \sum_{x \epsilon D, x_i = 1} 1 \tag{2.13}$$

Here D is the set of fit solutions in the breeding pool and n is the number of solutions in set D. This will produce a value between 0 and 1 which can be seen as the proportion of solutions within the set D where x_i is 1. For example if n = 10 and x_i = 1 is true in 6 members of D then $\rho(x_i = 1)$ will be calculated as 0.6.

The pseudocode in Algorithm 1 describes the UMDA process in detail.

2.8.1.2 *Population Based Incremental Learning*

In Population Based Incremental Learning (PBIL) [8] a model is built that takes into account both information from the current generation and previous generations. Like with UMDA, this model is a vector of probabilities. To distinguish it from the UMDA vector ρ, this vector shall be denoted P. As with UMDA, each probability P_i corresponds to an individual gene at index i. Unlike in UMDA, P is changed gradually over time through the use of a tunable learning rate Δ. P is initialised so that every element has a value of 0.5. Each generation it is moved towards a promising probability vector S through Equation 2.14.

$$P'_x = ((1 - \Delta)P_x) + (\Delta S_x) \tag{2.14}$$

Here, Δ has a value between 0 and 1. P_x is the probability of gene x that was used to generate new solutions last generation. P'_x is the probability of gene x that will be used to generate new solutions this generation.

The effect of this is that over time P will move from a vector of values close to 0.5 to a vector of values close to 1 or 0. The theory behind PBIL is that this corresponds to how a GA population changes [8]. Early on in a GA, for a given gene each allele is likely to have an equal probability as there is no reason why a greater proportion of solutions should have a particular allele instead of the alternatives and so the chances of each allele appearing in a new solution through crossover alone is relatively equal.

In a binary problem the chance of each gene being set to 1 is likely to be close to 0.5. Selection from the first population slightly moves these probabilities in a certain direction in the same way that the first update of P in generation 1 slightly moves this vector in a certain direction. Later on, the GA population is likely to contain a large number of similar or identical solutions and so the probability of a particular gene being set to 1 through crossover alone is likely to be close to 0 or 1. For this reason PBIL should maintain diversity in a population early

Algorithm 1 UMDA

function EVOLVE
  $P_0 \leftarrow$ InitialisePopulation()
  for $g \leftarrow 0 \rightarrow generations$ **do**
    for all $P_{g_i} \in P_g$ **do**
      AssessFitness(P_{g_i})
    end for
    $B \leftarrow$ TruncationSelection(b, P_g)
    $\rho \leftarrow$ BuildUMDAProbabilityModel(B)
    $P_{g+1} \leftarrow \emptyset$
    while $|P_{g+1}| < popSize$ **do**
      $I \leftarrow$ SampleFromProbabilityModel($\vec{\rho}$)
      $P_{g+1} \leftarrow P_{g+1} \cup I$
    end while
  end for
end function

function BUILDUMDAPROBABILITYMODEL(B)
  $n \leftarrow solutionLength$
  $\rho \leftarrow (\rho_0 = 0, \rho_1 = 0 ... \rho_n = 0)$
  for $i \leftarrow 0 \rightarrow n$ **do**
    for all $B_j \in B$ **do**
      if B_{j_i} **then**
        $\rho_i \leftarrow \rho_i + \frac{1}{|B|}$
      end if
    end for
  end for
  return ρ
end function

function SAMPLEFROMPROBABILITYMODEL(ρ)
  $I \leftarrow (I_0, I_1 ... I_n)$
  for $i \leftarrow 0 \rightarrow n$ **do**
    $I_i \leftarrow$ Random $< \rho_i$
  end for
  return I
end function

on in the same way as a GA but it should also be able to converge on the global optimum later on. Although PBIL may prematurely converge when the values in P approach 0 or 1, it has the advantage that its rate of convergence can be controlled by changing the value of Δ [8]. With a low learning rate the probability vector remains close to its initial setting in which every index is equal to 0.5 and so new solutions are generated almost at random whereas with a high learning rate new solutions are generated in a much more exploitative manner. UMDA is equivalent to PBIL with a learning rate of 1, and so it is more exploitative than PBIL and more prone to premature convergence [83].

To strengthen the correspondance between PBIL and a GA and to ensure that premature convergence is avoided, mutation may form part of this algorithm. Mutation can either operate on the probability vector P or on new solutions once they have been generated [8].

One question that arises when PBIL is used is how the vector that P is moved towards, S, should be obtained. The initial strategy was that S should be the fittest solution in the population. The disadvantage of this approach is that a wealth of information that exists within the rest of the population is essentially thrown away. P may be moved towards a set of individuals instead of a single individual.

Early empirical evidence suggested that using multiple solutions may produce better performance [8]. This reduces the probability of becoming trapped in a local optimum that is, by chance, close to whichever good solution was selected to be S and utilises information from the rest of the population. One disadvantage is that using multiple vectors introduces an extra parameter that needs tuning, the number of solutions used.

In one variant of the multiple solution strategy P may simply be moved in the direction of a set of fit solutions. There are a number of different ways of doing this. The process previously used for a single fit solution may be repeated separately for each solution. An alternative is to only move P for genes where all of the solutions agree on a value. As an alternative to just using fit solutions, the fact that certain solutions perform poorly may also be useful information and so in some versions of PBIL the least fit solution in the population may be used. This may simply involve moving P in the direction of the fittest solution and then reversing the process for the least fit solution. The problem with this is that if both solutions are similar any movement towards the fitter solution is likely to be undone as P moves away from the less fit solution. A preferred method is to move away from the worst solution only for genes where the two solutions disagree [8]. In this situation, each element in P is obtained through Equation 2.15. In this equation S_1 is the best individual in the population and S_2 is the worst. Two learning rates are used, a positive learning rate (Δ_1) and a negative learning rate (Δ_2). These may be different.

$$P'_x = ((1-\Delta_1)P_x) + (\Delta_1 S_{1_x}) \tag{2.15}$$
$$\text{if } S_{1_x} \neq S_{2_x} \text{then } P'_x = ((1-\Delta_2)P_x) + (\Delta_2 S_{1_x})$$

Table 2.2 provides an example of how P is generated in PBIL as compared to UMDA. In this example a population of 10 chromosomes is given, each with 4 genes. These constitute the initial population, the population at generation 0. P_i is the vector that is used to produce generation i. In the case of P_0 this is simply the initial state of P in which each value is equal to 0.5. Two versions of PBIL are included. Type A is the original version of PBIL, defined in Equation 2.14, in which only the fittest individual in the population is used. Type B uses the fittest and the least fit individual in the population and is the variation defined in Equation 2.15, as this variation was shown to produce good results during the initial development of PBIL [8]. We assume that Δ has a value of 0.1. For simplicity's sake this ignores any selection operator and assumes that in UMDA that whole population is used to produce the probability model.

2.8.2 *The Compact Genetic Algorithm*

The Compact Genetic Algoirthm (CGA) [60] differs from conventional EDAs in that it does not use a population. Like PBIL it uses a probability model, P that learns and improves over time. Unlike PBIL, this is used to generate just two solutions instead of a population. These two solutions are used to produce two more solutions that immediately replace them and so on. The learning rule used also differs from PBIL. Fixed size increments are used to update P and so the update is not proportional to the existing values in P as is the case in PBIL.

For a binary encoded problem the CGA process proceeds as follows:

1. P is initialised with all values set to 0.5.
2. Two solutions are generated from P. For binary problems these solutions are encoded as $x_1, x_2 ... x_n$ where $x_i \in [-1, 1]$.
3. P is updated in a similar manner to PBIL from these two solutions. For genes where the two solutions differ, P will move towards S_1 (see Equation 2.16).

$$\text{if } S_{1_x} \neq S_{2_x} \tag{2.16}$$
$$P'_x = P_x + \Delta S_{1_x}$$

4. Steps 2 to 3 are repated until a stopping criteria is met.

CGA has the advantage that there is no need to store a population and so less space is required. When it was developed, CGA drew on PBIL but introduced the notion that the

Table 2.2: Example of Generating the Sampling Vector in UMDA and PBIL

	Genes			
	0	1	0	0
	0	0	0	1
	0	0	0	0
	0	1	0	0
	0	1	0	1
(Fittest individual - S_1)	0	1	0	1
	0	1	1	0
	0	1	0	0
(Least fit individual - S_2)	0	1	1	0
	0	1	0	0
Total	0	8	2	3
UMDA				
ρ	0	0.8	0.2	0.3
PBIL - Type A				
P_0	0.5	0.5	0.5	0.5
P_1	(0.5×0.9)	(0.5×0.9)	(0.5×0.9)	(0.5×0.9)
	$+(0 \times 0.1)$	$+(1 \times 0.1)$	$+(0 \times 0.1)$	$+(1 \times 0.1)$
P_1	0.45	0.55	0.45	0.55
PBIL - Type B				
P_0	0.5	0.5	0.5	0.5
P_1 (towards S_1)	0.45	0.55	0.45	0.55
P_1 (towards S_2)	no	no	(0.45×0.9)	(0.55×0.9)
	change	change	$+(0 \times 0.1)$	$+(1 \times 0.1)$
P_1 (towards S_2)	0.45	0.55	0.405	0.595

population is not necessary. A good EA should lead to good genes spreading throughout the population while bad genes are lost. In a standard GA this may or may not be the case due to the fact that whole solutions are selected rather than individual genes. When poor genes are selected due to their presence in good solutions it is a form of sampling error and the reason for the population is to act as a buffer against this sampling error. Even when a number of bad decisions are made for gene i resulting in solutions that have a bad value for this gene it would be expected that the population will contain other solutions with a good value for gene i and will continue to contain at least some solutions with a good value for gene i in subsequent generations. This means that the overall probability of selecting a good value for gene i is unlikely to drop to 0. CGA dispenses with the population as its probability model fulfils this function. As each pair of solutions only changes the probability model by a small amount the probability of each possible allele is unlikely to drop to 0. In this way CGA is able to eliminate the population and reduce the memory requirements of a traditional GA while still ensuring that diversity is maintained and that good genes are not lost.

The manner in which the update rule differs from that used by PBIL is also motivated by the objective of reducing the amount of memory needed. As the vector is updated in fixed steps there is only a limited range of values that it can hold and so it can be compressed in a manner not possible in PBIL.

Table 2.3 provides a worked example of CGA. In this example a Δ of 0.01 is used.

2.8.3 *Multivariate and Bivariate EDAs*

In this section we discuss EDAs that generate new solutions by taking into account gene interdependencies, a process that is sometimes called *linkage learning*. Their potential ability to learn linkage information is one of the key motivations behind EDAs. As Chen [32] points out, in GAs sophisticated linkage learning techniques must be employed whereas in EDAs interdependencies may be automatically incorporated into the probability model.

On the other hand, univariate EDAs may perform worse than simple GAs in multivariate environments as the probability distribution may be misleading [64]. Take, for example, an instance of the k-trap problem introduced in Section 2.4 in which an individual trap of length k has a fitness of k if all the genes are set but a fitness of $k - l - 1$ if only a subset of the genes of size l is set. This creates a deceptive situation where the fittest suboptimal solutions have few genes set but the global optimum is a solution with all genes set. If UMDA is used to solve this problem then, for each gene x_i within this trap, there will be more highly fit solutions where $x_i = 0$ than where $x_i = 1$ and so the probability that $x_i = 1$ as calculated from the set of good solutions in the breeding pool is likely to be low. This is because more highly fit solutions follow the deceptive slope towards a solution of all zeros than towards an optimal solution of all ones. In GAs this is less of an issue as even though crossover is not

Table 2.3: Example of Generating the Sampling Vector in CGA

P_0	0.5	0.5	0.5	0.5
Pair of Solutions 1				
(Fittest - S_{1_1})	0	1	1	0
(Least Fit - S_{2_1})	0	1	0	1
P_1	0.5	0.5	0.5	0.5
	$+(0.01 \times 0)$	$+(0.01 \times 0)$	$+(0.01 \times 1)$	$+(0.01 \times -1)$
P_1	0.5	0.5	0.51	0.49
Pair of Solutions 2				
(Fittest - S_{1_2})	1	0	1	0
(Least Fit - S_{2_2})	0	0	1	1
P_2	0.5	0.5	0.51	0.49
	$+(0.01 \times 1)$	$+(0.01 \times 0)$	$+(0.01 \times 0)$	$+(0.01 \times -1)$
P_2	0.51	0.5	0.51	0.48
And so on...				

an ideal process as it disrupts building blocks randomly [70], GAs normally breed from just two highly fit solutions. New solutions will therefore, on average, contain half the genes from each parent, making the chance of building blocks surviving higher than in univariate EDAs where the proportion of genes surviving from each parent is likely to be lower.

For these reasons many multivariate EDAs have been developed. These can be broadly classified into bivariate EDAs, which only model pairs of interdependent genes and not higher order interdependencies, and multivariate EDAs, which model interdependencies between an arbitrary numbers of genes [112].

This section will focus on multivariate EDAs as bivariate EDAs are insufficient for some problems [98]. In the k-trap problem, for example, only dependencies of order k are significant and so, if k is greater than 2, a bivariate algorithm would be unable to detect dependencies. In this section we shall discuss in detail two multivariate EDAs, the Extended Compact Genetic Algorithm (ECGA) and the EBNA. These are popular EDAs that have the advantage that they don't require any prior knowledge of where interdependencies are likely to exist in the problem, in contrast to some other multivariate EDAs such as the Factorised Distribution Algorithm (FDA) [69]. Before discussing these multivariate techniques, we shall briefly describe one of the most popular bivariate EDAs, Mutual Information Maximisation for Input Clustering (MIMIC).

MIMIC is a bivariate EDA developed by De Bonet [13]. It makes multiple attempts at choosing an order into which to sort the genes and gives each order a score for how accurately it models the problem. For each ordering MIMIC will obtain the marginal probability for the final gene and then work backwards through the genes and for each gene calculate its conditional probably given the gene before it. For ordering π and with chromosomes of length n the associated probability vector $p_l^\pi(x)$ can be obtained through this equation:

$$p_l^\pi(x) = p_l(x_1|x_2) \cdot p_l(x_2|x_3)...p_l(x_{n-1}|x_n) \cdot p_l(n) \tag{2.17}$$

The Kullback-Leibler divergence [79] is then used to calculate how far $p_l^\pi(x)$ diverges from $p(x)$, the vector of marginal probabilities, thereby providing a score for how well the model predicts the actual probabilities of each gene.

One of the drawbacks to multivariate EDAs is the issue of scalability. As related genes may not be adjacent the first task of a multivariate EA is to discover which genes are related. As the length of the chromsome increases the number of possible combinations becomes exponentially large. Even if we only consider pairwise interactions the number of possible directed dependencies to examine for a problem of size l is $l(l-1)$. The number of directed dependencies of arbitrary order for a problem of size l is $l!$. Because of these issues it is sometimes found that, for problems that might otherwise benefit from a multivariate EDA, the computational effort needed to find good solutions is actually less for univariate EDAs. Inza

et al in Larranaga et al [83] demonstrated that this was the case for large feature selection problems, where multivariate algorithms such as MIMIC did not find good solutions faster than standard GAs when CPU time was taken into account even though they did when CPU time was not taken into account.

2.8.3.1 *The Extended Compact Genetic Algorithm*

The Extended Compact Genetic Algorithm (ECGA), developed by Harik [62], is a well known example of a multivariate EDA. The defining philosophy behind ECGA is that the model generated should be the simplest possible model that still accurately describes the population, with simplicitly defined in terms of the number of bits required to store the model. Harik explains that unbiased search for a model is futile and that he has therefore chosen to use a bias towards producing a simple model in the development ECGA [62].

ECGA, like univariate EDAs such as UMDA and PBIL, is based around the marginal probability distribution. The only difference is that the marginal probabilities are calculated over combinations of genes instead of over individual genes. These are used because of their simplicitly. In ECGA the problem is divided into sets of interrelated genes, or *partitions*. For each partition the marginal probability associated with each configuration of that partition is stored. For example, suppose we have a breeding pool of fit chromosomes as detailed in Table 2.5. They have each been divided into 2 non contiguous partitions, Partition A and Partion B (see Table 2.4). In Table 2.6 each configuration of each partition has been given a probability obtained by dividing how often it occurs in the breeding pool by the breeding pool size.

To understand why some models require less bits to store them than others despite being obtained from the same number of solutions each containing the same number of genes, consider that for partition A in the in the example in Table 2.4 the configurations 000, 100 and 111 have a positive probability and all other configurations have a probability of 0. A probability of either 1 or 0 requires only 1 bit to store it. There are therefore 5 configurations in this example that only require 1 bit to store them. Clearly if every configuration of partition A had a probability between 1 and 0 this would not be the case and the probability distribution would require more bits to store it. Likewise, if all configurations have a probability of 0 except for a single configuration that has a probability of 1 then the marginal probability distribution could be stored using just 1 bit for each configuration.

By using this property to build models that are as simple as possible ECGA is able to store its models in fewer bits than univariate EDAs such as UMDA. As algorithms such as UMDA make no attempt to compress the model, for each gene a number of bits equal to the number of chromosomes (n) is required. This means that, for 4 genes, $4n$ bits are required. In the ideal situation, in which every configuration can be represented by one bit, ECGA may only require

Table 2.4: ECGA Example - Partitions

Partition	Gene Indexes
A	1,3,4
B	0,2

Table 2.5: ECGA Example - Chromosomes

Chromosome
11000
10100
01111
10000

Table 2.6: ECGA Example - Marginal Probabilities

Configurations of A	Number of Occurences	Probability
*0*00	2	0.5
*0*01	0	0
*0*10	0	0
*0*11	0	0
*1*00	1	0.25
*1*01	0	0
*1*10	0	0
*1*11	1	0.25
Configurations of B	Number of Occurences	Probability
0*0**	0	0
0*1**	1	0.25
1*0**	2	0.5
1*1**	1	0.25

8 bits. This is because n will equal 2 in this situation and $4n$ will therefore be equal to 8. This is less than $4n$ for a breeding pool containing more than 2 individuals [61].

The complexity of a model is scored using the *Combined Model Complexity*. This is the product of two scores that are calculated through Equations 2.18 and 2.19, the Model Complexity and the Compressed Population Complexity [62]. In these equations, N is the population size, S_I is the size of partition I and M_I is the marginal distribution for partition I. The model complexity is calculated as the product of the total number of marginal probabilities that need to be modelled within the population, $\sum_I (2^{S_I} - 1)$, and the amount of space needed to store each of these marginal probabilities uncompressed, $\log_2(N+1)$. The compressed population complexity score allows us to then incorporate how far the model may be compressed.

$$\text{Model Complexity} = \log_2(N+1) \sum_I (2^{S_I} - 1) \tag{2.18}$$

$$\text{Compressed Population Complexity} = N \sum_I \text{Entropy}(M_I) \tag{2.19}$$

ECGA WORKFLOW To choose how the chromosome should be divided up, ECGA follows the following steps:

1. Divide the chromosome into partitions such that there is a partition for each gene, containing just that gene.
2. Measure the combined model complexity.
3. For every possible pair of partitions, measure what the combined complexity would be if these two partitions were to be merged.
4. Merge the two partitions that were most successful at reducing the combined complexity.
5. Repeat steps 2 to 5 until it is no longer possible to reduce the combined complexity.

This process is repeated afresh every generation.

To produce new solutions, the marginal probability vector can be sampled in the same way as in UMDA. A configuration is chosen for the genes corresponding to each partition in the new solution by applying roulette wheel selection over the set of configurations and their calculated probabilities.

2.8.3.2 *The Estimation of Bayesian Networks Algorithm*

Another Multivariate EDA, and one that has already been proved to show promising results on feature selection problems is the Estimation of Bayesian Network Algorithm (EBNA) [70]. EBNA, developed by Etxeberria et al [40], uses the individuals in the breeding pool to build a

Bayesian network which it then samples from to produce new individuals. Bayesian networks are directed networks that are able to store any probability distribution and so are often used for probability calculations [70]. EBNA defines a method for building a Bayesian network and also defines how this network is then sampled to produce a new population.

BAYESIAN NETWORKS A Bayesian Network is a structure from which the conditional probability of a variable can be calculated. It consists of two parts. One part is a Directed Acyclic Graph (DAG), generally denoted as M, where each node corresponds to a variable and where an arc between two nodes indicates a relationship between them. The second part is a set of conditional probabilities, generally denoted as θ.

Using a Bayesian Network we can say that the probability p that variable X_i has value x_i is determined by the parent nodes of X_i within M. The probability factorisation of a given solution x, made up of variables $x_i ... x_n$, is given in Equation 2.20.

$$p(x) = \prod_{i=1}^{n} p(x_i|\pi_i) \tag{2.20}$$

In this equation π_i is the current instantiation of the parent nodes of X_i. The probability of x_i given its parent nodes, $p(x_i|\pi_i)$, is obtained from the probability matrix θ. We represent the probably that X_i is in its kth state when its parents π_i are in their jth state as θ_{ijk}.

BUILDING A NETWORK One of the most important questions that needs to be asked when Bayesian networks are used is how to build the most appropriate DAG to use as M.

In an ideal situation the probability vector will be calculated as an average of all possible DAGs. This is referred to as *Bayesian Model Averaging* [65] and is as carried out as shown in Equation 2.21. In this equation $\mathbf{m}$ refers to the set of all possible DAGs.

$$p(x|D) = \sum_{M \in \mathbf{m}} p(x|M, D)p(M, D) \tag{2.21}$$

This is, however, not usually practical because, as the number of variables increases the number of possible networks increases at an exponential rate [17]. The two practical alternatives are *selective model averaging* and *model selection*. In selective model averaging the average probability distribution is obtained from a selection of good potential models rather than from every possible model. In this process the probability distribution is calculated through Equation 2.22, where $\mathbf{m}^s$ is the set of good potential models. In model selection only one network is used, the network with the highest posterior likelihood. Equation 2.23 describes this process where $\hat{M}$ is the network with the highest likelihood. Although selective model averaging can yield better results, model selection is done more frequently because it is relatively low cost. In the case of the EBNA model, selection is preferred [69] as it is necessary

that the modelling process is fast in an evolutionary algorithm as it will have to be repeated many times.

$$p(x|D) \approx \sum_{M \in \mathbf{m}^s} p(x|M,D)p(M,D) \quad \text{where} \sum_{M \in \mathbf{m}^s} p(M,D) \approx 1 \tag{2.22}$$

$$p(x|D) \approx p(x|M,D)\hat{M} \quad \text{where } p(\hat{M},D) \approx 1 \tag{2.23}$$

EBNA attempts to identify an M such that $M \approx \hat{M}$. To do this it requires both a search method which defines in which order it attempts to add or remove arcs from the network and a method for scoring each potential network. A greedy search technique developed by Buntine [21] was used. This is a simple and efficient technique that starts with a network in which there are no arcs and in which every variable is assumed to be independant. It then iteratively adds arcs until no further arc offers any improvment. This technique was implemented using an efficient implementation developed by Bouckaert in [17], where he identified it as *Algorithm B*. In Algorithm B, initially every possible arc $a[i,j]$ is joined in turn, given that j is the parent node and i is the child node. The network is scored with $a[i,j]$ joined and then $a(i,j)$ is separated again. Once every arc has been assessed, the arc associated with the highest scoring network, $a[I,J]$ is joined permanently and the process is repeated. In the second iteration, however, only arcs that would potentially connect the recently connected child node, node I, with a new parent node need to be reassessed. This is described schematically in Algorithm 2 for a problem of n variables [17]. $pred_i$ refers to all the ancestors of i and $desc_i$ refers to all of the descendants of i, including i. A is a vector containing the score of each potential arc.

The Bayesian Information Criterion (BIC) was used to score potential networks [40] [70]. BIC [111] is a popular model scoring system and has the advantage that, by penalising larger networks, it encourages simple representations and is computationally inexpensive. The BIC score is calculated as follows:

$$\text{BIC}(M,D) = \sum_{i=1}^{d}\sum_{j=1}^{q_i}\sum_{k=1}^{r_i} N_{ijk}\log\frac{N_{ijk}}{N_{ij}} - \frac{\log N}{2}\sum_{i=1}^{d}(r_i-1)q_i \tag{2.24}$$

In this equation, d is the number of variables, r_i is the number of states that variable i may be in and q_i is the number of states that its parents may be in. N_{ijk} is the number of instances within the sample where variable i is in its jth state and where its parents are in their kth state. N_{ij} is the number of instances where variable i is in its jth state. N is the size of the sample.

Algorithm 2 Algorithm B

```
function ALGORITHM B                                   ▷ initially assess all arcs
    M ← arcless network
    for all i ∈ M, j ∈ M do
        if i ≠ j then
            A[i, j] ← score(M ∪ a[i, j]) − score(ø)
        else
            A[i, j] ← −∞                               ▷ prevent self loops
        end if
    end for
                                                       ▷ main loop
    finished ← false
    while !finished do
        select i, j that maximizes A[i, j]
        if A[i, j] > 0 then
            M ← M ∪ a[i, j]
            oldscore ← score(M)
            for all a ∈ Pred_i, b ∈ Desc_i do
                A[a, b] ← −∞                           ▷ prevent loops
            end for
            for k ← 1 → n do
                if A[i, k] > −∞ then
                    A[i, k] ← score(M ∪ a[i, k]) − oldscore
                end if
            end for
        else
            finished ← true
        end if
    end while
end function
```

EBNA WORKFLOW Overall, the EBNA process can be divided into three steps:

1. Generate M by initially assuming that all variables are independent and then iteratively adding the arc which offers the greatest improvement to the network's BIC score. This is repeated until no arc offers any improvement.
2. Calculate θ.
3. Generate a new population.

The new population is formed through Probabilistic Logic Sampling (PLS). In PLS nodes are ordered according to ancestral ordering. This means that for every pair X_i and X_j where X_i is an ancestor of X_j, X_i should be placed before X_j in the ordered list of nodes. The list is then iterated through and nodes are set based on their conditional probability. The first nodes in the list will be those nodes which do not have parents and these are set according to their univariate marginal probability in the same manner as in univariate EDAs.

2.9 HYBRID ALGORITHMS

The algorithm proposed in this thesis falls into the category of algorithms that contain elements of both GAs and EDAs. These approaches are useful as neither EDAs nor GAs perform better than the other approach on all problems. On some problems EDAs become trapped in local optima while on other problems they produce faster convergence than GAs. It can be difficult to predict whether an EDA or a GA will perform better for a particular problem [99]. There has been some discussion of a concept referred to as the "No Free Lunch" theorem which states that, when tested across all possible problems, the average performance of every algorithm will be roughly the same [108]. If this is the case then, for every problem in which algorithm B outperforms algorithm A we would expect there to be a problem in which algorithm A outperforms algorithm B. In many domains this may not be an issue as time can be spent finding and tuning the perfect algorithm for the problem. However, in some domains there exist multiple, rapidly changing variations on a basic problem. Sometimes a small change can have a dramatic effect on the solution needed to solve a problem and the metaheuristic needed to find this solution as well. Burke gives an example of a nurse rostering problem in Belgium and explains that a solution was found using a metaheuristic that was tailor made to the rules and regulations applying specifically to Belgium such that it would need to be significantly changed were it to be applied to a problem in another country. Often users do not want to fine tune algorithms for every problem and simply want to find a solution that is, as described by Burke, "good enough, soon enough, cheap enough" [23]. The result of this is that there is a need for techniques that are adaptable to different problems and hybridisation may, in some cases, be the only way to achieve this. Several problems have also been investigated in which non hybridized evolutionary algorithms fail to obtain optimal solutions [54].

Hybridisation may be integrated at a number of points in the evolutionary process [54].

- When good solutions are selected or new solutions are bred they may be improved upon by local search techniques. These algorithms are referred to as *Memetic Algorithms*
- The initial population may be generated by a search technique which is known to produce good solutions for the problem at hand and these may then be improved upon further by an evolutionary algorithm.
- Some algorithms may produce solutions that have an encoding that does not directly map to a phenotype. The exact nature of the mapping from genotype to phenotype may be decided upon by another metaheuristic method.
- Some algorithms may operate in two or more phases, with each phase producing a population that is then used as input for the next phase.

One example that falls into the category of memetic algorithm is the Cooperative Local Search developed by Landa-Silva and Burke in 2007 [82]. In this technique each individual is improved through a local search process until no further improvement is possible. At this point information from the rest of the population is used to move it to a different part of the search space where it may be improved upon further. This has the advantage that at any one time the population will consist of a mixture of solutions that have recently been modified using information from the population as well as older solutions which have been following their own path using local search for a much longer period of time and so this helps to ensure diversity within the population [63].

Another example, this time falling into the category of algorithms that adopt different approaches at different phases in the evolutionary process, EvoC, was developed by Tan et al [127] to solve a medical classification problem. In these problems a large amount of information is collected from patients and trying to obtain knowledge, such as a diagnosis, from this information can be a difficult and tedious task for human professionals. To obtain this knowledge, rules are used that normally have an "IF-THEN" structure and take as input a piece of information, that may be ordinal or nominal, and produce as output some derived knowledge. The tree structure commonly used in Genetic Programming (GP) techniques is a natural a way of representing these "IF-THEN" rules. However, ordinal information cannot be placed straight into this structure without first being converted into nominal classes, a process that can be done by hand but this is time consuming and prevents an algorithm from being quickly applied to any problem. Rules for ordinal attributes have been shown to be best evolved through GAs instead of GP. This is, therefore, an example of a problem where a particular form of hybridization, combining GAs with GP techniques, may offer an advantage.

EvoC solves this problem with a two phase process. In the first phase an initial population of good rule sets is evolved through a combination of GA and GP. In the second phase the rule

sets are further refined through a second evolutionary process to produce an optimal rule set. The complete process is outlined below. This process assumes a two class problem though it can be extended to multiple classes simply by repeating the process.

1. Phase 1
    a) Two populations are generated, one contains rule sets consisting of only the nominal features and one contains rule sets consisting of only ordinal features.
    b) Tournament Selection is used to select individuals from both populations.
    c) A GP is used to breed from the nominal rule sets.
    d) A GA is used to breed from the ordinal rule sets.
    e) Once a new population of both kinds has been produced a token competition [137] is applied to remove redundant rule sets. This ensures diversity by reducing the number of similar rule sets.
    f) Once a fixed number of generations has passed both final populations of rule sets are used as input for phase 2
2. Phase 2
    a) The rule sets are grouped so that for each number of rules there exists a group containing all rule sets with that number of rules.
    b) Each group is evolved through a standard GA to obtain optimal rule sets for each number of rules.
    c) The optimal rule set from each group is combined into a final population.
    d) This final population is evolved to find the optimal rule set. Optimising different size rule sets independantly enables both the number of rules and which rules to select to be simultaneously optimized.

Examples of techniques that combine GAs with EDAs include a technique developed by Pena called GA-EDA [99]. This approach generates two populations, one through an EDA and one through a GA. The number of individuals that are generated through each technique is determined by a function referred to as the *Participation Function*. A number of variations of the participation function have been explored:

- A fixed percentage of solutions is generated from the GA and the EDA.
- The number of solutions generated by each approach is incremented according to Equation 2.25, in which gen is the current generation and M is the generation at which both the EDA and the GA should represent 50% of the population. Two versions of this technique were tested. In one version r is the ratio of solutions generated by the EDA, in which case this algorithm will start out generating most of the solutions through a

GA and will over time increase the proportion of solutions generated through an EDA. Alternatively, r may be the proportion of solutions generated by the GA.

$$r = \frac{gen}{M + gen} \tag{2.25}$$

- The algorithm alternates between generating all solutions through a GA and generating all solutions through an EDA.
- A dynamic process is used to calculate the ratio of solutions generated by each approach according to how well solutions generated by each approach performed in the previous generation. In this approach two average fitness scores, $avg(\text{EDA})$ and $avg(\text{GA})$ are calculated as the average fitness of the top 25% of solutions generated by each approach in the previous generation. The algorithm starts with both sets having a ratio of 50% and then each generation the ratios are adjusted through the following equation:

$$\begin{aligned} diff &= |avg(\text{EDA}) - avg(\text{GA})| - base \\ ratio(\text{EDA}) &= ratio(\text{EDA}) + ((avg(\text{EDA}) - age(\text{GA})) \times adjust \times diff) \\ ratio(\text{GA}) &= 1 - ratio(\text{EDA}) \end{aligned} \tag{2.26}$$

GA-EDA was tested on several different problems. On some of these problems the best performance was from a GA and on others the best performance was from an EDA but in each case using a mixture of the two approaches performed better than the worse performing of the two pure approaches, demonstrating that, although hybrid approaches are not necessarily the best approach for every problem they at least ensure that a reasonably good solution can be found without needing to know whether an EDA or a GA should be used for a given problem to achieve optimal performance.

The dynamic method of choosing ratios was interesting as it demonstrates another advantage of hybrids, their usefulness in learning about the problem. Pena was able to monitor the way that the ratios changed over time and was able to learn through this that on 3 out of the the 4 problems tested initially a GA performed well but later on an EDA performed better.

Zhang proposed an approach called Guided Mutation which produces new solutions that contain a mixture of genes copied from a promising individual and genes sampled from an EDA probability distribution [143]. This is because there is an assumption in evolutionary computing, referred to as the optimality principle [44], that good solutions will have a similar selection of genes and a similar structure to other good solutions. For this reason, EAs that use mutation and crossover and so essentially generate new solutions from just 1 or 2 good solutions are effective because they generate new solutions close to these good solutions. However, when they generate solutions close to an existing solution that is in a local optimum, they lack the global knowledge needed to move away from that local optimum and find the global optimum. EDAs, by contrast, produce the global knowledge needed to do this but lack

a method of generating new solutions close to a single good existing solution. Assuming fixed length binary encoded chromsomes, guided mutation will produce a new individual y as follows:

1. Select a good individual x from the population.
2. For every gene x_i in x:
   a) Choose a random value r between 0 and 1.
   b) If $r < \beta$ then $y_i = x_i$ where β is a predefined probability.
   c) Otherwise set y_i with probability p_i, where p is a univariate marginal probability distribution, as in UMDA.

This is particularly relevant to the development of the hybrid algorithm put forward in this thesis where the selection and tuning techniques are automated using statistics from the population. The remaining subsections within this chapter will discuss different ways in which algorithms adapt to the nature of the population, either by behaving like a different algorithm entirely or simply by operating with different parameters.

2.9.1 *Hyperheuristics*

There are some algorithms in which any number of different techniques are used and when and which techniques to use is chosen dynamically. These algorithms are referred to as *hyperheuristics*. Hyperheuristics can be thought of as search techniques that search through the space of metaheuristics and choose the metaheuristic best suited to the problem [108][22]. They are designed to be problem independent and to require minimal tuning [96]. Because of this, they eliminate the need for a human to design or tune the ideal metaheuristic for their problem, considering, for example, factors such as the shape of the fitness landscape or the level of noise in the population. This is a process that is time consuming and that requires a human expert who has knowledge both of the current problem and of a variety metaheuristic techniques.

There are a number of forms that hyperheuristics may take. They may take the form of a metaheuristic in which the individuals that make up the population are themselves metaheuristics or combinations of metaheuristics. They may simply choose metaheuristics according to a rota or randomly. More sophisticated hyperheuristics integrate a feedback loop in which measures such as the quality of solution that is produced by each metaheuristic or the level of diversity achieved is used to decide on how often each metaheuristic will be used in subsequent generations [96].

In one hyperheuristic, proposed by Burke in [23], a tabu search based approach is used in which different heuristics are ranked according to how well they solve the problem. Initially

all heuristics have a rank of 0. When a heuristic is applied its rank is increased if it finds a solution fitter than any previously found and decreased if it does not. The amount that it is increased or decreased by may be dependant on the increase in quality of solution that it has produced or on the length of time taken to find this solution. The idea behind this approach is that it should be possible to choose and apply any heuristic at all. It is not necessary to know anything about the internal structure of each heuristic as they are judged by simply allowing them to compete with rival heuristics. In addition, a tabu list of heuristics is maintained to ensure that heuristics that have not been successful are not tried again. Heuristics are removed from the tabu list after a certain period of time.

Burke also proposed an alternative approach in which a solution consists not of a single heuristic but of a list of heuristics that are carried out in a particular order to generate new solutions [26]. New solutions are generated from existing solutions by changing 2 of the heuristics in the list. Once again, a tabu list is used and each new solution is placed in the tabu list so as not to be visited again until a certain amount of time has passed. In [25] a hyperheuristic method based on case based reasoning is proposed. In this method different variations of a timetabling problem are stored as cases, represented by a weighted list of features. The system consists of a *case base*, a collection of cases, each mapped to a heuristics that is known to be able to solve it. The features that are used and the weights are chosen through hill climbing and tabu search. The case base is then trimmed to ensure that harmful or redundant cases are not included through a "leave one out" process. Cases are removed from the case base and they are only returned if removing leads to a reduction in system performance. Once a relatively optimal case base and a weighted list of features has been produced the system uses the nearest neighbour algorithm to choose a heuristic for any new case that it is presented. This involves measuring which cases are closest to the new case and using a weighted average to choose the heuristic most likely to solve it.

2.9.2 *Self Adaptive Algorithms*

While hyperheuristics automate the process of choosing which metaheuristic to use, evolutionary algorithms are also defined by the parameters with which they run. Choosing a good value for parameters such as mutation rate or crossover is important for an algorithm to perform well. The two terms that are relevant here are *parameter tuning* and *parameter control*.

Parameter tuning is the process whereby parameters are chosen before the algorithm is used to best fit the needs of the problem. Parameter control involves dynamically altering parameters during the evolutionary process. Parameter tuning is traditionally carried out by hand. With multiple parameters optimising them manually can be time consuming. This problem is made more significant as parameters are often interrelated and so ideally one would need to examine every possible combination of parameters. For just 4 features and 5

values each, this would amount to $5^4 = 625$ combinations. As explained by Eibin in [39], if each parameter combination were to be tested with 100 runs a total of 62500 runs would need to be completed to establish the optimal combination. Parameter control, by contrast, is by definition an automatic process that takes place while the algorithm is running. There is some argument that it is necessary in any effective evolutionary algorithm to alter the evolutionary parameters during the course of a run as different values are optimal in different stages of the evolutionary process. Mutation rate is a good example as early on a high mutation rate is needed to effectively explore the search space but later a lower mutation rate is needed to effectively hone in on an optimal solution. It is possible to alter the parameters during a run at a fixed rate over time. This, however, relies on choosing a rate of change and it can be described as a "blind" process as it does not take into account the actual state of the population at a given time [39].

There are algorithms that obtain some measure from the population or from individuals within the population, perhaps connected to diversity or fitness spread, and set parameters as a function of that measure. One example of a genetic algorithm that uses parameter control is the Adaptive Genetic Algorithm (AGA) developed by Srinivas [119]. In AGA the average fitness of the population $\bar{f}$ and the fitness of the fittest individual in the population f_{max} are obtained. It was observed that as the population converged on a local optimum $f_{max} - \bar{f}$ decreases to 0 and so this was used as a measure of convergence. This is used to control two parameters, the mutation rate p_m and the crossover rate p_c through the following equations:

$$p_c = k_1/(f_{max} - \bar{f}) \tag{2.27}$$
$$p_m = k_2/(f_{max} - \bar{f})$$

In this equation k_1 and k_2 are different for each solution and are dependant on the fitness of that solution. Both values are smaller when the solution is fitter, making the probability of crossover and mutation lower. The effect of this algorithm is that fitter solutions survive from one generation to the next as they are less likely to be disrupted by crossover and mutation than less fit solutions. When the population is starting to converge more fit solutions will avoid disruption, enabling the solution to converge on the optimum. However, this technique does not eliminate all manual selection of parameters. Even though k_1 and k_2 are parameters that change dynamically, they are calculated through functions that use pre-specified values.

Another approach to parameter control is *Self Adaptive Algorithms*. In these algorithms parameters such as the mutation rate are included in the chromosome and evolve as the solutions that contain them evolve [39]. The rationale behind this method is that optimal evolutionary parameters will be selected and evolved naturally without needing to explicitly choose them. The term "mutation operators are evolved for free" has been used [39] to reflect this fact. This may be the only way to adequately choose these parameters. This is because the information about which parameters work best may not be available, the range of

parameters with which good optimization is possible may be narrow and the ideal parameters may change during the course of an evolutionary process [59]. An adaptation of this basic approach, developed by Hansen, is the Covariate Matrix Adaptation Evolutationary Strategy (CMA-ES). In this approach the mutation and crossover parameters are adjusted according to the path that these parameters have followed across time. The basic principle is that, if a parameter has repeatedly moved in the same direction then these multiple small steps can be seen as equivalent to one large step, and so, in future, a larger change in this direction is encouraged. If, on the other hand, a small step in one direction was followed by a step in the reverse direction that effectively cancelled it out then neither step was necessary and so these steps are ignored. In an ideal case every change in the parameters will be perpendicular to previous changes.

2.10 GENE REPRESENTATION

Evolutionary Algorithms can be classified based on the type of chromosome that they work with. The original research into GAs and EDAs focussed on chromosomes that consisted of fixed length binary strings [15][6] in which each gene may have a value of 0 or 1.

Many problems require chromosomes with continuous valued genes. Though it is possible to approximate continuous domains through binary chromosomes this approach has its weaknesses. Several binary genes may correspond to just one value in the phenotype, meaning patterns and relationships that exist may be obscured [15].

As well as the nature of the genes, chromosomes also vary in their structure. Beyond the classical fixed length string, variable length chromosomes allow for individuals of varying complexity [141]. In the context of the problems discussed here a variable length structure may allow the number of control points to be physically represented by the length of the chromosomes, thus creating a more natural representation and potentially saving storage space for extremely large and sparse chromosomes.

Variable length chromosomes pose extra challenges for evolutionary algorithms. Originally GA crossover techniques were designed with the assumption that both parents had the same number of genes in the same order. Techniques such as one and two point crossover are easily adaptable to variable length chromosomes [77]. However, uniform crossover for example, relies on both chromosomes being the same length in its standard form [118].

2.11 MULTIPLE OBJECTIVE OPTIMIZATION

In many real world problems, there is no single overall objective but rather a range of possibly contradictory objectives [68]. The task of an EA now becomes not to find a single optimal solution but to find a range of solutions that between them are optimal for all objectives.

EAs are especially well suited to simultaneously assessing multiple objectives due to their implicit parallelism [144][129]. By developing EAs specifically designed for multiple objective problems we are able to find a range of solutions satisfying different requirements that it would otherwise take several runs to find. This is important for many problems as a solution that is acceptable to one designer and in one situation might be not be acceptable in a slightly changed situation. If a set of solutions are provided a human expert may choose between them to decide on the solution that is right for the current situation in a way that would not be possible for an automatic optimizer [120].

For example, when choosing a car there are a number of different requirements such as cost, comfort, performance and reliability. The relative importance of these requirements depends on the preferences of each buyer. Simply discovering an optimal car is therefore not possible and so it is better to present a range of cars between them covering all requirements and let each buyer choose based on the requirements that are important to them [38].

2.11.1 *The Pareto Front*

Multiple Objective optimisation involves searching for solutions that are *Pareto optimal* or *non-dominated*. A Pareto optimal solution is a solution that is not dominated by any other solution in the population. Solution A is said to dominate Solution B if, for every objective, Solution A performs better than or to the same standard as Solution B and there exists at least one objective where Solution A outperforms Solution B. For example, consider a problem with two objectives, objective A and objective B, and a population of solutions as given in Table 2.7. For each solution their fitness in both objectives and the solutions that dominate them is given. Solution 1 is, for example, dominated by solution 2 because 2 is a better fitness than 1 and 5 is a better fitness than 4. In this case the Pareto front consists of solutions 2,3,6 and 7.

These solutions can be plotted on a graph, as shown in 2.2, in which the Pareto optimal solutions can be clearly seen forming an arc across the top and the right of the graph. This arc can be described as the *Pareto Front*. The challenge of multiple objective optimisation is to find a set of solutions that approximate this arc.

2.11.2 *Fitness in Multiple Objective Optimisation*

The difficulty with these problems is knowing how to score solutions as using a single fitness score is no longer sufficient. There are two main requirements that any method used must fulfil. Firstly, that solutions must be found that are as close to the true Pareto front as possible and secondly, that solutions should be well spread out across the Pareto front. This second requirement is referred to as finding a set of solutions that have good *Sparsity*. It is important

Table 2.7: Pareto Front Example

Solution	Fitness A	Fitness B	Dominated by
1	3	4	2,7
2	3	5	none
3	2	6	none
4	1	6	3
5	4	1	6,7
6	5	1	none
7	4	4	none
8	2	5	2,3
9	1	2	1,2,3,4,7,8,10
10	2	4	1,2,3,7,8

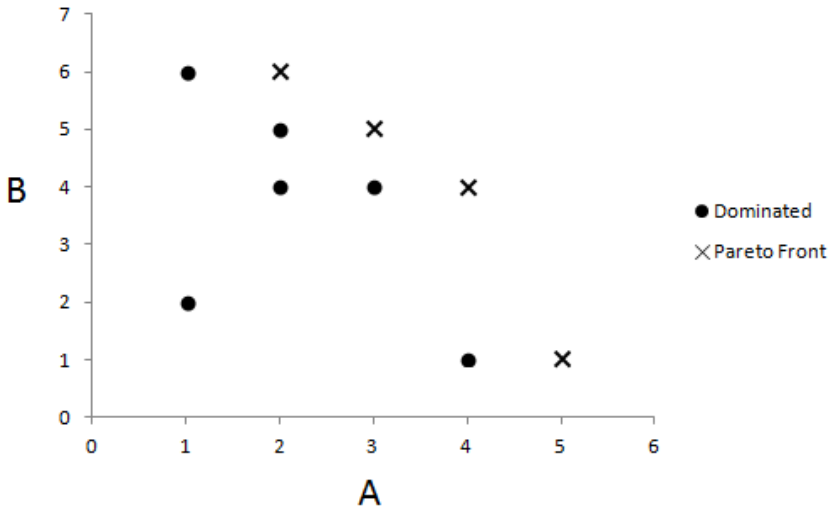

Figure 2.2: Pareto Optimal Solutions

that sparsity is maintained every generation as it ensures that all objectives are equally well covered. For this reason, a Pareto scoring system should take into account of a solution's sparsity as well as whether it is Pareto optimal [120]. Because of the sparsity requirement maintaining diversity and avoiding premature convergence is especially important in these problems [129].

2.11.2.1 *Converting to a Scaler*

A crude method is to simply add together the fitnesses for different objectives, multiplying each fitness by a weight factor. This converts a multi-objective fitness function into a scaler and therefore allows us to use any single objective optimization technique. Deb [38] suggests that this approach corresponds to how we naturally think of objectives. If we wish to find a car that satisfies two objectives, for example fuel efficiency and reliability, then it is natural to simply combine the two, weighted according to which is the most important to us. As the objectives might not necessarily be on the same scale they should each be normalised before hand [38]. The overall procedure is as given in Equation 2.28. This describes how the combined fitness of a solution, F is obtained from a vector of fitnesses f and a vector of weights w. Each element of f and w corresponds to one objective in a problem with a total of n objectives. The vector f is normalised to f' using the highest fitness score, $max(f)$, and the lowest fitness score, $min(f)$, for each objective.

$$F = \sum_{i=0}^{i=n} f'_i \times w_i \tag{2.28}$$

$$f'_i = (f_i - min(f_i))/(max(f_i) - min(f_i))$$

There are a number of issues with this approach. Firstly, it does not take into account of sparsity and so care must be taken to ensure that any algorithm used maintains diversity in the population. Even if uniform weights are used it is unlikely that this approach will produce a set of solutions spread out across the Pareto front. Uni-objective algorithms are designed to converge on one solution and so are not well suited to produce a range of solutions. In particular, solutions in a non convex portion of the Pareto front are unlikely to be discovered [38]. Which solutions are discovered is sensitive to which weights are used [68][120] and these can themselves be hard to choose.

2.11.2.2 *Objective Switching*

Due to the drawbacks of the scaler based method it was necessary to develop approaches more suitable to multiple objective optimisation. One option is to sometimes use one objective and other times use another. A simple approach is to randomly switch between objectives each time a solution is selected [80]. The Vector-Evaluated Genetic Algorithm (VEGA) has

a different approach. In VEGA [110] the population is randomly divided into a number of equally sized sub populations. Each sub population has an associated objective so that, for each individual in sub population S_i, the fitness value that will be used will be its fitness for objective O_i. These subpopulations are then combined and selection and crossover with these fitness values is used to produce the new population. The drawback of these approaches is that they are good at finding solutions that are optimal for one objective but less able to find solutions which are good at several different objectives [76].

2.11.2.3 *Pareto Fitness*

There are some algorithms that use Pareto dominance to derive a fitness score for each solution. The first algorithm of this type was proposed by Goldberg in [49]. In this technique solutions in the Pareto front are given a fitness of 1, assuming that scores are being minimised. These solutions are then removed and solutions in the new Pareto front are given a fitness of 2. These solutions are also removed and in the next Pareto front solutions are given a fitness of 3. This process is repeated until all solutions have been assigned a fitness. In general, solutions that become the new Pareto front when solutions in Pareto front n are removed are given the score $n+1$.

The Non Dominated Sorting Genetic Algorithm (NSGA) [120] is a similar approach except that it introduces a sharing operator to satisfy the sparsity requirement. First of all, a large dummy fitness value is chosen and every solution in the Pareto front is given this fitness. Each solution is then required to share its fitness with solutions close to it. In this way solutions that are in sparse regions of the front end up with a higher fitness than those in dense regions. The algorithm then follows the same approach as used in Goldberg's technique, in which successive Pareto fronts are removed, creating a heirarchy of Pareto fronts. Each front is given a dummy fitness and for each front the sharing process is applied, always ensuring that the highest fitness in front f_i is lower than the lowest fitness in front f_{i-1}, the front immediately infront of it.

2.11.3 *Non Dominated Sorting Genetic Algorithm-II*

NSGA-II was proposed by Deb in [37] based on NSGA. NSGA-II offered two main improvements over NSGA. The first improvement was that the the sharing operator was replaced by an operator described as "crowding" that had the advantage that there was no need to choose a sharing parameter. The second improvement was the introduction of elitism to ensure that good solutions were not lost. In NSGA-II each new population generated is merged with the old population. The Pareto optimal solutions as well as some elite non optimal solutions from this combined population are then bred from.

In NSGA-II an individual has a Pareto rank score and a sparsity score. The Pareto rank score is 0 for an individual in the first Pareto front, 1 for an individual in the second Pareto front and so on. A solution's sparsity score is calculated by ordering the Pareto front that it is a member of by each objective in turn and calculating the sum of the difference in fitnesses between the individuals either side of it. Equation 2.29 describes how the NSGA2 fitness F_x of a solution x is calculated. Here J is the set of all objectives and the function $f_j(x)$ returns the fitness of solution x in terms of objective j. $next_j(x)$ is the solution immediate ahead (fitter than) solution x when solutions are ordered by objective j and $previous_j(x)$ is the solution immediately behind (less fit that) solution x when solutions are ordered by objective j. $max(f_j)$ and $min(f_j)$ are the fitnesses of the fittest and the least fit solutions respectively in terms of objective j.

$$F_x = \sum_{j=0}^{|J|} \frac{f_j(next_j(x)) - f_j(previous_j(x))}{max(f_j) - min(f_j)} \tag{2.29}$$

The workflow of NSGA-II is as follows:

1. The population is initialized and evaluated.
2. Every solution in the population is given a Pareto score as described above.
3. This score is adjusted through crowding so that individuals in sparser regions of the front have a better score than individuals in denser regions.
4. Individuals are selected, bred and mutated based on their score to produce a new population. The selection, breeding and mutation techniques are not specified by this algorithm.
5. The new population and old population are merged.
6. The Pareto front from this merged population is added to an elite pool of equal size to the initial population.
7. If there is still room in the elite pool then the next Pareto front is added, and so on until there is no more room in the elite pool.
8. The elite pool is merged with the new population and the algorithm repeats from step 2.

2.11.4 *Co-Evolution*

In addition to having multiple objectives, some problems can be thought of as an amalgamation of different but related problems, each of which require a different solution that depends on a different set of variables. These problems may be so highly interrelated that they could not

be effectively optimized independently [103]. In many problems, there is a co-operative or competitive relationship between two or more components. For example, in some software engineering problems there is a need to produce both a test suite of test cases able to detect any errors within a program and a program which should, ideally, not contain errors. These can be optimized in a competitive environment in which programs attempt to produce as few errors as possible when tested using test suites which are simultaneously trying to detect as many errors as possible [107].

These problems are difficult to solve with conventional EAs, which are designed to produce single solutions and which lack an inbuilt method for producing solutions that fall into different categories. Co-evolutionary algorithms are those which divide the population into two or more subpopulations. These subpopulations may have different objectives characterised by different fitness functions or even use solutions with different structures or encoding systems and so they are therefore a logical choice for evolving solutions that fall into different categories. Although designed for problems of this nature, co-evolutionary algorithms have a number of advantages that make them useful for a wide range of problems. They are useful for multi-objective problems, as well as problems where diversity is important or where solutions could be decomposed into separate parts, even if this decomposition is not immediately apparent.

Typically, in co-evolutionary algorithms the evolutionary operators of mutation, crossover and selection take place separately within each subpopulation. These function in the same manner as in standard EAs. Co-evolution places no special requirement on which operators are used, it is not necessary that different populations use the same operators and it is not even necessary that all or any of the populations use evolutionary algorithms. Any optimisation technique could, in theory, be used. Once a new generation of solutions has been generated the process of evaluating the fitness of each individual is often one where multiple subpopulations need to be involved. In competitive situations an individual will need to be made to compete against an individual from another subpopulation in order to assess its fitness. In situations where subpopulations produce different components of a larger solution each individual will need to be combined with an individual from at least one other subpopulation. Typically, while the fitness of individuals in one population, population A, are being assessed, the individual representing the collaborating or competing population, population B, is kept the same and not assessed. This process is then reversed so that the individuals in population B can be assessed. This allows the fitness of individuals within each subpopulation to be fairly assessed, without the fitness of the opponent or collaborator having an effect. Deciding on the individual that should represent a subpopulation in these collaborations may simply be a matter of selecting the fittest individual in the last population to be assessed. Alternatively, an individual may be selected randomly or stochastically based on its fitness [103].

Aside from being the natural choice for some problems, co-evolutionary approaches have a number of advantages. Firstly, they help to ensure diversity is maintained. Although each subpopulation is expected to ultimately converge on a set of good solutions and to lose diversity, as with any EA, there is no logical reason why different subpopulations should converge to the same solution even if there is no difference between subpopulations with regards to the structure of solutions or the fitness function. The fact that breeding is restricted to a single subpopulation also helps to prevent the kind of destructive breeding that occurs when two individuals, specialised in different areas, are bred together. In evolutionary computing, as in nature, the breeding of two very different individuals is likely to produce non viable offspring [103]. This is a problem when a single population is used, especially where there are two or more objectives, as any two individuals, including those highly specialised for different objectives, may be selected to be bred together. Co-evolution also has the advantage of being inherently parallelizable as separate subpopulations have minimal interaction and so can be run as separate processes.

2.12 SUMMARY

This chapter has provided an overview of evolutionary algorithms, the field of metaheuristics in which the algorithm that is the focus of this thesis falls. It has introduced the main subcategories of Evolutionary Algorithm that are explored in this thesis, Genetic Algorithms and Estimation of Distribution Algorithms. As the algorithm within this thesis is a hybrid of these two approaches other forms of hybrid algorithms have also been discussed.

It has been explained that evolutionary algorithms are algorithms in which new solutions to a problem are generated by selecting promising solutions from a population and exchanging information between these solutions in a stochastic manner. Key areas of consideration when building such an algorithm have been considered. This has included how diversity is maintained within the population, how information about interdependencies between problem parameters is obtained and used to produce new individuals and the basics of schema theory and how to avoid disrupting useful patterns, known as building blocks.

GAs are EAs where new solutions are generated by selecting two parents, carrying out crossover whereby new solutions generated are a mixture of the two parents and changing them slightly through mutation. EDAs, by contrast, select a set of fit solutions from the population and build a probability model from these which is then used to produce new solutions. Although GAs were the initial and simplest form of evolutionary algorithms EDAs have several advantages over them, including their ability to model independencies between variables, their potential for theoretical analysis and the absence of parameters such as crossover and mutation rate that need tuning. The discussion of hybridisation covered several different meth-

ods including self adaptive and hyperheuristic techniques where the hybridisation process is, in some ways, automated.

3

APPLICATIONS

In this thesis TEDA has been applied to several problems from different domains to determine how well its performance generalises. This is important as a lot of existing research in metaheuristics has suffered from the limitation of focussing on developing bespoke solutions to problems which are expensive to adapt whenever the problem changes. In contrast, TEDA draws from the field of hyperheuristics which has sought to redress this by developing algorithms that can adapt themselves to a range of different problems [22]. It is desirable that TEDA has this quality and is able to generalise to different problems without being greatly modified.

The problems described here were chosen as they all share the common feature that solutions contain a certain amount of control points which have an associated cost. These are therefore problems where there exists an optimal number of control points, or an optimal range within which the number of control points should fall, as the objective is to solve the problem while minimising the cost. These problems are therefore suitable for TEDA as the targeting process should be capable of identifying this optimal number of control points.

The first type of problem discussed in this thesis for which this is true is optimal control. Optimal control problems are well suited to this area of research as they involve selecting control points with the aim of achieving two objectives, satisfying the physical constraints of a problem while minimizing or maximising some performance criterion. A performance criterion may involve minimising time taken to solve a problem or effort [74], objectives that may be directly related to the total amount of actions taken.

The second domain looked at is routing. Although the basic routing problem [9] is simply a matter of finding a route across a landscape involving links that have a certain cost associated with them this is extended in this thesis as a penalty is added in proportion to the total number of links. This is used to reflect the fact that passing from one link to another may itself be an expensive process [33]. This is an interesting topic for this thesis as TEDA can be tested on versions both with and without this penalty to demonstrate the difference a problem that explicitly rewards a lower number of control points has on TEDA's performance.

Finally, FSS was investigated. In this domain the equivalent of control points are features and targeting is important as there is a need to discover minimal sets of features. This problem is useful as some feature set problems may involve 1000s of features and so these problems are larger and potentially more noisy than the other problems in this thesis and so test TEDA's scalability.

In addition to being problems where targeting may be needed these problems each raise different challenges for TEDA. In the optimal control problem explored, whenever a control point is selected a magnitude must be chosen for it whereas the other problems are binary problems in which a control point may simply be on or off. This extra decision may affect TEDA's performance. In the routing problem solutions are defined both by which routers are found in them and by the order of these routers and so the process of simply deciding whether a control point should be on or off is complicated in a different manner. This problem is also encoded through a variable length encoding system and so reveals whether TEDA can be successfully applied to such an encoding system.

3.1 PROBLEM 1: CHEMOTHERAPY SCHEDULING

This section discusses optimal control problems and the specific optimal control problem that was used in this thesis. This was a cancer chemotherapy treatment scheduling problem that was formulated by McCall and Petrovski in 1999 [89].

3.1.1 *Optimization of Control Problems*

When optimizing control problems the aim is to choose a set of actions that best solve an optimal control problem, that is to say they are the most effective at controlling some process. These problems consists of three parts, a mathematical model that describes the process, a set of constraints and a performance criterion [74]. These actions are referred to as *interventions* in optimal control literature although for consistency with the other problems in this thesis they will be referred to here as *control points*. As an example, we consider the optimal control problem that is used to introduce these concepts by Kirk in [74], the problem of driving a car from point A to B. The mathematical model describing this problem takes the form of a differential equation describing the car's movement. A constraint that is likely to be relevent is the upper limit on the acceleration that the car may achieve. A performance criterion may be the time which the car takes to travel from A to B.

Optimizing these problems is a matter of deciding which control points should take place when in order to maximise some score whilst minimizing a cost. Constraints reduce the range of allowable candidate solutions. In the class of problem that we are interested in, the cost is related to how many control points are used, therefore the objective is to maximise fitness while using as few control points as possible. Solutions will often be encoded as a vector in which every element represents a potential control point. The fitness function evaluation may take the form of applying candidate solutions to a mathematical model of the system or they may be tested on a real or simulated system [4].

With a large range of requirements and constraints these problems are often quite difficult to solve through many existing optimization methods (such as gradient based methods) or through mathematical programming. Local optima may be scattered apparently randomly across a complex fitness landscape and the global optimum can be difficult to find [100]. These problems are also challenging due to the presence of a feedback loop where the state of the system at a given time determines what effect a control point will have at that time. The state of the system has, in turn, been affected by previous control points and so earlier control points can have a crucial effect on the outcome of later control points [4].

3.1.2 *Overview of Problem*

Chemotherapy is often used to combat cancer and due to the toxicity of the drugs used there is a need to minimise the frequency with which the drugs are applied while still reducing the tumour size enough to cure the patient [48]. The problem is classed as an optimal control problem as it takes the form of a period of days and the objective of an optimizer is to decide on which days an action, the adminstration of drugs, should take place.

The primary objective of the chemotherapy problem is to obtain a drug treatment schedule that minimizes the final size of a cancerous tumour after the period of treatment has finished. In addition, it is necessary to do this while limiting the total amount of drugs used. Chemotherapy drugs are delivered directly to the bloodstream and they affect all body tissues. They are toxic and so it is necessary to limit both the amount of drugs applied at any one time and the total cumulative dose over the entire treatment period so that they do not damage sensitive tissue. It is also necessary to ensure that, throughout the trial, the size of the tumour stays below a lethal level [101].

This problem was chosen for several reasons. It is the problem used to develop FDC, the algorithm which gave rise to TEDA [48]. It is also a popular problem for evolutionary algorithms as it is a challenging problem. This is because it features a large number of variables and constraints and finding the optimal solution is a delicate balancing act between applying enough drugs to cure the patient while minimizing any toxic side effects [19].

It should be noted that the version of the chemotherapy problem used in this paper is the simpler of two main variants found in literature. A variant is found in the work of Petrovski et al [101] which considers multiple different types of drug whereas the variant used in this paper is the single drug variant used in the thesis of Paul Godley [47]. In the multi-drug variant, an integer value will need to chosen at each time slot for each drug. This introduces an additional challenge as it is accompanied by an additional requirement. While in the single drug variant, drug toxicity is incorporated purely through a limit on the cumulative dosage permitted, in the multi drug variant each drug has its own maximum cumulative dosage and

there is an additional toxic side effects requirement incorporated into the equation which considers all of the drugs.

3.1.3 *Problem Formulation*

Chemotherapy treatment schedules are encoded using a fixed length genome where each gene represents a day in the treatment schedule. For these studies a treatment schedule of 100 days is used. Each gene holds an integer value between 0 and 15 where 0 indicates that no drugs are applied and 15 indicates that the maximum possible dosage should be applied.

In order to obtain a fitness for the primary objective of reducing the tumour size, first of all the change in size without considering the effect of drug treatment is obtained through the Gompetz model as detailed in Equation 3.1 [46]. This defines how the size of the population of tumour cells N grows over time, limited by θ, a constant providing an upper limit on the size of the tumour. $N(t)$ is the total number of tumour cells at time t, λ is the rate of tumour growth and θ is an absolute limit on the tumour size.

$$\frac{dN}{dt} = N(t)\lambda ln\frac{\theta}{N(t)} \tag{3.1}$$

The ability of anti cancer drugs to kill these cells is then calculated as a function of the concentration of drugs at time t, denoted as $C(t)$, the toxicity of these drugs (κ) and the number of tumour cells. This function is given in Equation 3.2.

$$\frac{dN}{dt} = -\kappa C(t)N(t) \tag{3.2}$$

These models are combined in Equation 3.3 to predict how the number of tumour cells changes over time when subject to the treatment schedule C:

$$\frac{dN}{dt} = N(t)\lambda ln\left(\frac{\theta}{N(t)}\right) - \kappa C(t)N(t) \tag{3.3}$$

κ is the toxicity of the drug. Each solution is passed through this model gene by gene and the final value for N at the end of the treatment schedule, $N(T_{final})$ is used to obtain a final fitness score f through Equation 3.4 [47].

$$f = ln\left(\frac{\theta}{N(T_{final})}\right) \tag{3.4}$$

The additional requirements relating to drug toxicity and to keeping the tumour below a lethal size at all times are implemented through constraints.

The first constraint is detailed in Equation 3.5. This ensures that the drug dosage given in any one day is always below a pre specified maximum, C_{max}. This is implemented by simply limiting the value of each gene to 15.

$$g_1(c) = C_{max} - C_i \geqslant 0 \tag{3.5}$$

The second constraint is detailed in Equation 3.6 and ensures that the cumulative dose stays below a pre specified limit, C_{cum}. If g_2 is less than 0 then it is added to the final fitness (f in Equation 3.4) in order to reduce the final fitness and so penalize treatment schedules that involve the excessive use of drugs.

$$g_2(c) = C_{cum} - \sum_{i=1}^{n} C_i \geqslant 0 \tag{3.6}$$

The final constraint, detailed in Equation 3.7, ensures that the number of cells in the tumour at no point exceeds the lethal level N_{max}. As with the second constraint, this is added to f if it is less than 0.

$$g_3(c) = N_{max} - N(t_i) \geqslant 0 \tag{3.7}$$

3.2 PROBLEM TWO: ROUTING

The routing problem simulates the task of finding a route across a cost landscape where the cost is the change of height experienced while traversing the landscape combined with the number of route changes made. The objective is to find a route involving a number of straight line links where the total height change across each link is minimal. This is analogous to finding a route across a network where a link is the connection between two routers. We consider height change to be equivalent to factors such as link transmission capacity and network congestion. In this context the overall objective of finding a route with the minimal height changes is equivalent to finding the fastest route across a network [1].

In addition we penalise the number of links in a route. There is a complex system of queues within a router and packets may get lost or delayed [33] and so it is desirable to discover routes that pass through as few routers and contain as few links as possible. Because of the additional requirement of passing through as few routers as possible this problem differs from more traditional routing problems [9], justified by the real life cost of passing through routers in many networks. The objective of finding shorter routes may alternatively be achieved through constraints limiting the length of routes. The disadvantage of using constraints is that it requires an assumption to be made before hand as to how long an ideal route should be, or

at least what the upper limit should be. There are also no constraints on how routers may be visited. Each router may connect to any other router or to itself and each router may be visited at any stage of the route.

Aside from these factors this problem is essentially a traditional routing problem. Some similar problems may use a representation in which the cost of traversing a link would be represented through its length instead of through its change in gradient. This representation could be achieving by flattening out the landscape that we have used so that the hills and valleys become distances across a flat plain in proportion to their original height. This is simply an issue of ease of representation and it does not affect the underlying problem.

3.2.1 *Encoding*

Each chromosome represents a route and takes the form of an ordered list of points that are used in that route. All routes start at the same fixed starting point s and finish at a fixed finishing point e and so these points are not included in the chromosome. Optimization involves choosing between 0 and $max|C|$ points from a set of available points K and choosing the order in which these points should be visited. K is a set of randomly chosen 2D coordinates representing the location of available routers positioned in the space $(x_{min} .. x_{max}, y_{min} .. y_{max})$. Each gene identifies the point used as an index in the array of available points K. Therefore, genes are integer values with 1 as their minimum and $|K|$ as their maximum.

Unlike in the chemotherapy problem, there are no zero valued genes and unused routers are not represented in any way and so chromosomes are of a variable length and contain just the routers that are used. This problem therefore, unlike chemotherapy, cannot be treated as a problem of fixed length binary strings. Encodings that are different to the traditional fixed length binary string are now popular in evolutionary computing as, despite being more difficult to implement, there is often a performance benefit when the encoding better matches the nature of the problem [71]. For this reason adapting evolutionary algorithms such as TEDA to different types of encoding is important if they are to be useable on a wider range of problems.

3.2.2 *The Fitness Function*

Equation 3.8 defines the fitness f of a route. $t(g, g+1)$ denotes the absolute change in height that occurs traversing from the router identified by gene g to the router identified by gene $g+1$, sampled at a set interval z. Note that although the start and end points are not explicitly encoded in the chromosome C, they are added to each end of it before calculation of the fitness cost for the route. The landscape used is defined by function 3.9. $h(x, y)$ refers to the height of the landscape at coordinate (x, y). w is a scaling constant that controls how high the

hills in the landscape are. For all tests in this thesis, w has a value of 10. Figure 3.1 shows this landscape as a 3D plot.

$$f(C) = \sum_{g=0}^{g=|C|-1} t(g, g+1) + |C| \times penalty \tag{3.8}$$

$$h(x, y) = w \cdot \sin(2(\frac{2x}{\cos(y)+2})) \tag{3.9}$$

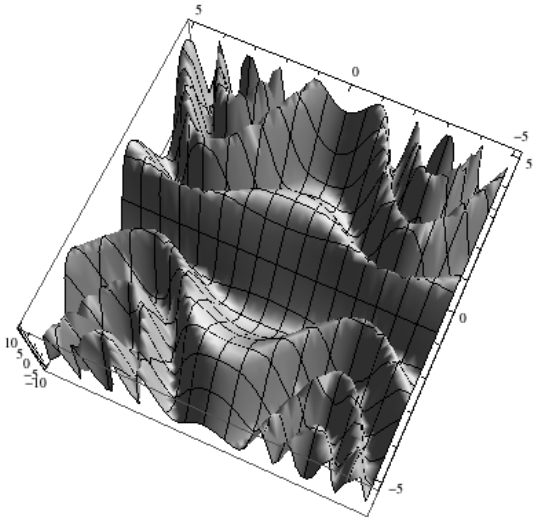

Figure 3.1: Landscape for Routing Problem

3.3 PROBLEM THREE: FEATURE SUBSET SELECTION

Many machine learning problems involve classifying data samples defined by a set of variables or *features*, into two or more classes. These are classification problems. Initially these problems are presented with a database of samples in which each sample is a representation of a particular instance or case study about which certain facts are known, referred to as the features of that sample. A learning algorithm uses this data set to produce a set of rules or *discriminants* which specify how features can be used to predict which class samples should by placed into. These rules may take the form of *IF THEN ELSE* rules or they may be more complex. Once this is done, when a new sample is introduced it is possible to use these rules to classify it into a particular class. This is based on the assumption that if two samples contain similar features then they are likely to fall into the same class. As this process relies on real and possibly noisy data, systems usually produce a probability that a new sample may fall into a particular class rather than a definite statement that it will fall into this class [3]. In one example, taken from [3], a bank uses a classification system to assess customers applying

for credit. Features describing a customer may include their income, savings, collaterals, profession, age and financial history. Customers may be classified as high risk or low risk. The probability which is produced can therefore be considered a numerical risk factor, which can then be used to decide whether the customer should be granted credit or not.

FSS is the process by which redundant or unecessary features are removed from consideration. Reducing the number of features used is vital as it may improve classification accuracy, allow for faster classification and enable a human expert to focus on the most important features [109]. A technique that is tasked with solving FSS problems can therefore be said to have two objectives. It should be able to find feature subsets that are as small as possible while also enabling samples to be classified with as great an accuracy as possible.

FSS has long been known as an NP hard problem in machine learning, as there are 2^n possible subsets of a feature set of length n and an exhaustive search is not possible. Various search heuristics have been developed [35].

Techniques can be divided into filter and wrapper methods [81]. Filters build feature sets by calculating the capacity of features to separate classes whereas in wrappers [75] each candidate feature set in turn is used to classify the final sample set with the classification technique that will ultimately be used. The key difference is that in filters features are assessed independantly or in smaller groups whereas in wrappers an entire candidate feature set is assessed at once. Wrapper methods can be more powerful than filter methods because they consider multiple features at once and so are able to take account of interdependencies and yet they tend to be more computationally expensive [56].

Among the most popular FSS methods are the sequental selection methods. These are Sequential Forward Selection (FS), introduced by Whitney in 1971 [135], and Sequential Backward Selection (BS), introduced by Marill and Green in 1963 [87][104]. In FS the most informative feature is selected to begin with. This is obtained by using each feature on its own to classify the data set using the final classifier and selecting the feature which is able to do this with the greatest accuracy. Once this feature has been added to the feature set a greedy search is carried out and the second most informative feature is added. This process is repeated until a feature set of size L, a pre-specified limit, is reached or no further improvement can be achieved by adding any of the remaining features. BS is the reverse of this process. In BS initially all features are part of the network and, at each iteration, the feature which improves the classification accuracy the most when it is removed is selected for removal from the network. Both of these techniques suffer from a similar disadvantage. In FS, a selected feature cannot later be eliminated and in BS an eliminated feature cannot later be selected [104]. This prevents the techniques from carrying out further exploration once a potential solution has been discovered.

EAs have often been applied to FSS problems. As features are often correlated [12] the implicit parallelism of EAs is of benefit as it means that they are able to simultaneously process

multiple feature sets. Cantu-Paz [27] demonstrated that both GAs and EDAs were equally capable of solving FSS problems but that a simple GA was able to find good solutions faster than EDAs. He concluded that, although in the cases that he tested a GA performed better, it was unclear in what situations the model building capabilities of EDAs would be of benefit and so this area should be investigated further. Inza [70] compared an EDA to both sequential approaches and GAs and found that it was able to find significantly more effective feature sets than any of the techniques that it was compared against.

3.3.1 *Scalability*

One of the main difficulties in this domain is the number of features available to the classifier. In a 1997 paper Blum noted that most classification problems involve less than 40 features [11]. Since then problems have expanded to often involve thousands or tens of thousands of features [56]. This change reflects the fact that in many real world domains such as text processing and bio-informatics feature sets are often this large. This thesis discusses problems that contain between 500 and 20000 features.

For these problems the maximum number of features is so large that the high order multivariate EDAs that are needed to handle the multivariate nature of these problems are impractically expensive [69]. Many of these problems are extremely noisy and only a small proportion of the features are useful [56]. For problems which are so noisy that only a tiny proportion of features are useful, driving down the size of the feature set is an important part of FSS. To achieve this objective techniques such as constraining the number of features and then iteratively removing features to fit within this constraint have been explored [109]. In this approach a decision needs to be made as to what value to set the constraint to. This is problematic as it requires previous knowledge of the problem.

3.3.2 *Classification Teachniques*

Once a feature set has been chosen there are a number of different classification techniques that may be used to classify the data. The performance of a feature set using a given classifier may be assessed in several different ways, but typically the data set is divided into a *training set*, a set of samples for which the class is already known, and a *test set* contains samples for which we wish to discover the class. Popular classification techniques include K-Nearest Neighbour (K-NN) [73] and Support Vector Machines (SVMs). In K-NN the k individuals in the training set that are most similar to the new sample are used to determine the new sample's class. In SVMs [14] two classes are distinguished by drawing between them the hyperplane that separates the instances of each class by the greatest possible margin. New samples are then

classified based on which side of the hyperplane they fall on. LIBSVM is an implementation of a SVM produced by [28] that is used in this thesis.

4

METHODOLOGY

As explained in the introductory chapter, Chapter 1, the purpose of this thesis is to introduce a novel evolutionary algorithm, the Targeted Estimation of Distribution Algorithm (TEDA). An important requirement is that this algorithm should be easily applied to different problems without needing to be modified, potentially a costly and time consuming process. It is therefore important that the conclusion of this thesis is a single, definitive version of TEDA. The remaining chapters of this thesis discuss each problem to which TEDA was applied and how it was applied to these problems and in doing so various different versions of TEDA are explored. For this reason, this chapter is included before we begin this discussion to provide a complete description of the final, definitive version of TEDA. This version is described in sufficient detail for it to be replicated in order to be tested on a new problem.

It is also important that for every problem TEDA is, as much as possible, tested using the same experimental setup and so this chapter describes this experimental setup. This includes the evolutionary parameters used to test both TEDA and the algorithms against which it has been compared as well a description of the scientific method used and how statistical significance has been ensured.

Section 4.1 describes the final version of TEDA in detail and Section 4.2 describes the experimental parameters used to test TEDA.

Most of the steps involved in developing this final version of TEDA were performed through the chemotherapy problem and Chapter 5 describes this process in detail. TEDA was further refined when it was applied to the routing problem, Chapter 6. The main change made in this chapter was the decision that a particular method of transitioning from a GA to an EDA, the probabilistic transitioning process, should form part of the final version of TEDA instead of the transitioning process proposed in Chapter 5.

4.1 TEDA

The main principle of TEDA is that it should begin the evolutionary process by operating exactly the same as FDC and should then, after the population has started to converge, transition to using a process that combines EDA style probability sampling with FDC style control point targeting to further converge the population. This process, referred to as *transitioning*, is designed so that, when the level of diversity within the population is high early on, TEDA will breed from 2 parents selected using a low selection pressure and when the population

diversity is low, TEDA will breed from a large number of parents selected with a high selection pressure. The TEDA process can be summarised in the steps below and the full TEDA process is detailed in Algorithm 3.

1. The population is initialized.
2. The population is evaluated against the fitness function.
3. The Breeding Pool, B, is selected in three steps. This is the part of the process that transitioning controls and it is described through pseudocode 4:
   a) 2 individuals are selected to form the start of the breeding pool. They are selected from the top s individuals in the breeding pool through tournament selection.
   b) If these individuals are similar then another individual is added to the breeding pool and the selection pressure is increased by decrementing s.
   c) Step b is repeated until a breeding pool of a size that is consistent with the level of convergence within the population has been obtained.
4. The EDA probability distribution is generated from the Breeding Pool.
5. From The Breeding Pool new solutions are generated through the following breeding process, detailed in Algorithm 5:
   a) The target number of control points I_T is calculated from the fittest and the least fit individual in B.
   b) All control points found in every member of B are selected at random and set in the new solution until I_T control points have been set.
   c) If I_T control points have still not been set then all control points found in some but not all members of B are selected at random and set with their marginal probability until I_T control points have been set.
6. Every new solution is mutated and placed into the new population.
7. Steps 2 to 6 are repeated until a stopping criterion is reached.

4.2 METHODOLOGY

To ensure consistency, all tests were run under the general conditions described in this section. A description of the conditions specific to each problem can be found at the start of each chapter. These sections describe how mutation and initialisation are carried out for their respective problem as there are some necessary differences between the different problems in this area. They also provide a table containing every parameter with which tests for that

Algorithm 3 TEDA Pseudocode

```
function EVOLVE
    P_0 ← InitialisePopulation()
    s ← s_max ← popSize
    for g = 0 → generations do
        for all P_{g_i} ∈ P_g do
            AssessFitness(P_{g_i})
        end for
        P_{g+1} ← Elite(P_g)
        while |P_{g+1}| < popSize do
            B ← GetBreedingPool(s, P_g)
            I_T ← GetTargetNumOfControlPoints(fittest(B),leastFit(B))
            ρ⃗ ← BuildUMDAProbabilityModel(B)
            S_all ← ∀i ∈ ρ where ρ_i = 1
            S_some ← ∀i ∈ ρ where 0 < ρ_i < 1
            for |B| times do
                I ← Mutate(Breed(S_all, S_some, ρ⃗, I_T))
                P_{g+1} ← P_{g+1} ∪ I
            end for
        end while
    end for
end function
```

Algorithm 4 TEDA Pseudocode - Transitioning and Selection

```
function GetBreedingPool(s, P)
    S ← bestSelection(s)
    b ← b_min ← 2
    B_1, B_2 ← tournamentSelectionFromSet(S)
    p ← getOverlap(B_b,B_{b−1})
    while random(1) < p do
        b ← b + 1
        s ← s - 1
        S ← bestSelection(s)
        B_b ← tournamentSelectionFromSet(S)
        if b = b_max then p ← 0
        else p ← getOverlap(B_b,B_{b−1})
        end if
    end whilereturn B
end function

function getOverlap(B_1, B_2)
    f̄_1 ← all control points in B_1
    f̄_2 ← all control points in B_2
    return size(f_1 ∩ f_2) / size(f_1 ∪ f_2)
end function
```

Algorithm 5 TEDA Pseudocode - Breeding

function GETTARGETNUMOFFEATS(B, t)

$Q \leftarrow$ Fittest(B) $\cup$ LeastFit(B)

$I_f \leftarrow$ NumberOfControlPointsIn(Fittest(B))

I_1 = NumberOfControlPointsIn(Q_1)

F_1 = Fitness(Q_1)

I_2 = NumberOfControlPointsIn(Q_2)

F_2 = Fitness(Q_2)

$I_T \leftarrow I_f + (2t - 1)(I_1 - I_2)(F_1 - F_2)$

return I_T

end function

function BREED($S_{all}, S_{some}, \vec{\rho}, I_T$)

$A \leftarrow \{x_1 = 0, x_2 = 0 ... x_n = 0\}$ ▷ new individual

while $I_T > 0$ and $S_{all} \neq \{\}$ **do**

$r \leftarrow$ random control point $\in S_{all}$

$A \leftarrow$ set r in A

$I_T \leftarrow I_T - 1$

remove S_{all_r} from S_{all}

end while

while $I_T > 0$ and $S_{some} \neq \{\}$ **do**

$r \leftarrow$ random control point $\in S_{some}$

if $\rho_r >$ random(1.0) **then**

$A \leftarrow$ set r in A

$I_T \leftarrow I_T - 1$

remove S_{some_r} from S_{some}

end if

end whilereturn A

end function

problem were carried out. General parameters, with which all tests were run unless otherwise specified, are given in Table 4.1.

In all cases a generational method of replacement was used to generate the new population. Generational replacement allows us to compare our approach with standard EDAs which usually breed from a section of a generation. FDC was originally proposed by Godley using steady state replacement [48] but by comparing the performance of FDC in section 5.2.1 with the results obtained by Godley it appears as though the replacement method does not have a significant effect on the performance of FDC.

To ensure validity, all tests are run multiple times. Final tests on which statistical analysis was carried out were run 200 times whereas intermediate tests were run 50 times. All graphs show the fitness score of the fittest individual in the population on the y axis. This is taken as a median of all runs. The x axis shows computational effort. As the evaluation of the fitness function is normally the most computationally expensive part of the evolutionary process [100], this is measured as the number of fitness functions evaluated. As generational replacement is used and the population size is 100, every 100 on the x axis is equivelant to one generation. To add further reliability to every graph, variance is indicated through the interquartile range at intervals of 50 evaluations.

The median is used due to the fact that we cannot assume that the fitness scores follow a normal distribution. Figures 4.1, 4.2 and 4.3 are histrogram graphs showing sample fitness distributions. They are taken at generations 250, 375 and 500 during a run of the chemotherapy problem using TEDA. As can be seen the distribution ranges from being roughly normal at 250 generations to being very skewed towards higher fitnesses at 500 generations. For this reason the mean does not give a good indication of results and so the median has been used instead.

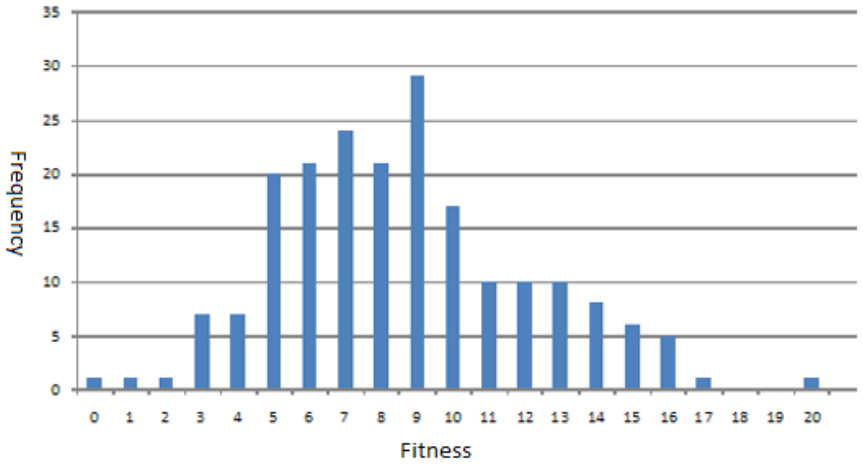

Figure 4.1: Fitness Distribution at 250 Generations

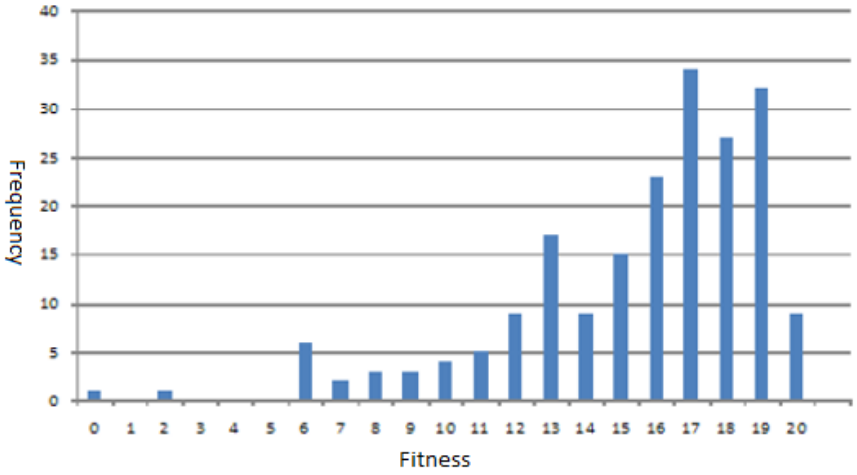

Figure 4.2: Fitness Distribution at 375 Generations

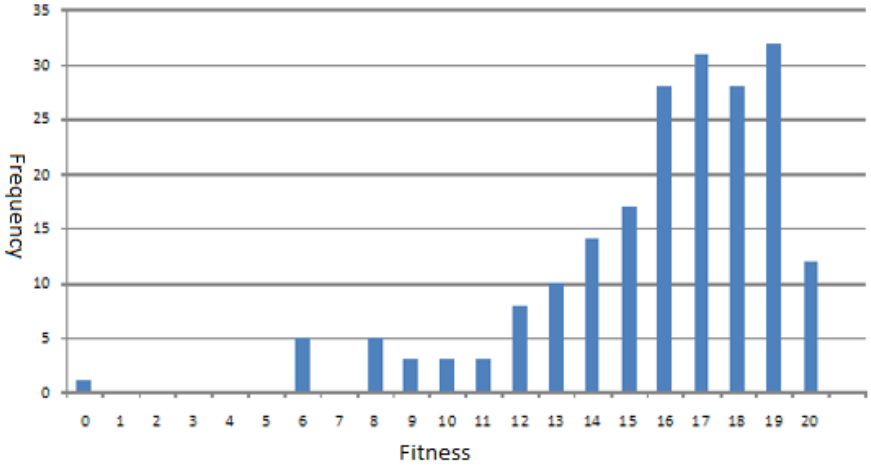

Figure 4.3: Fitness Distribution at 500 Generations

Paremeter	Value
Crossover Probability	1
Mutation Rate	0.05
Population Size	100

Table 4.1: General Experimental Parameters

4.2.1 *Kruskal Wallis Statistical Analysis*

In order to determine if there is a statistically significant difference between the performance of different algorithms, the Kruskal Wallis (KW) one-way analysis of variance by ranks test is used [115]. Kruskal Wallis was developed by William Kruskal and Allen Wallis [78] as a way of measuring statistical difference that uses the ranks of samples rather than their absolute value. This gives it the advantage that few assumptions need to made about the nature of the data, such as that the data is normally distributed or follows a linear scale. In order to compare two sets, set A and set B for example, the Kruskal Wallis test proceeds as follows:

1. The two sets A and B are combined to form set AB
2. Each data sample in set AB is given a rank, such that the highest valued sample has a rank of 1, the next highest value has a rank of 2 and so on. Where there is a tie each tied sample is given a rank half way between i and $i + n + 1$ where i is the rank of the sample immediately above the tied group and n is the size of the tied group. The next sample after the tied group is given the rank $i + n + 1$.
3. The mean is calculated from the ranks of all the samples in set A and from the ranks of all the samples in set B.
4. The difference between these ranks is used to obtain a difference score between the two data sets. This is written as $\left|\bar{R}_A - \bar{R}_A\right|$ where R_i is mean of all ranks in data set i.
5. This difference score is compared to a threshold. If it is greater or equal to this threshold then it is judged to be statistically significant. This is represented through Equation 4.1.

$$Z_{\alpha/(k(k-1)/2)}\sqrt{\frac{N(N+1)}{12}\frac{2}{n_A + n_B}} \tag{4.1}$$

   Here k is the number of data sets, n_i is the number of samples in data set i, N is the total number of samples across all data sets and α is the required confidence level. As each data set is compared against every other dataset, for k data sets there will be $k(k-1)/2$ comparisons. Z is obtained from a look up table that maps the value of α used to the number of comparisons. This look up table is provided in [115].

   For all tests in this thesis a confidence level of 0.05 is used. This means that any difference score which exceeds the result of the right hand side of this equation is statistically significant with a confidence level of at least 0.05.

Table 4.2 provides an example of a Kruskal Wallis test carried out on two data sets, set A and set B.

The difference between these two data sets is 0.25 as $4.625 - 4.375 = 0.25$. In this example, in order to be statistically significant with a confidence level of 0.05 the difference should

Values		Ranks	
A	B	A	B
3.8	1.5	2	8
2.4	5.0	5.5	1
1.6	2.4	7	5.5
3.0	3.2	4	3
Total Rank		18.5	17.5
Mean Rank		4.625	4.375

Table 4.2: Kruskal Wallis Example

be above the threshold obtained through Equation 4.2. With two data sets there is only one comparison and the look up tables tell us that $Z_{0.05/1} = 1.960$. The difference between these two sets is therefore not statistically significant. As this equation evaluates to approximately 4.94, a much larger value than the difference score 0.25, these data sets do not differ to a statistically significant extent.

$$1.960\sqrt{\frac{8(8+1)}{12}\frac{1}{8}} \approx 4.94 \tag{4.2}$$

5

CHEMOTHERAPY AND THE DEVELOPMENT OF TEDA

This chapter describes the initial development of the Targeted Estimation of Distribution Algorithm (TEDA) from FDC. FDC is described in detail in Section 2.7.2. Essentially it is a two phase Genetic Algorithm crossover process in which an optimal number of control points is predicted from the number in each of the parents and then a new solution is produced with that number of control points set. FDC was developed to solve optimal control problems and was primarily tested on the cancer chemotherapy problem described in Section 3.1. This problem was therefore chosen to be the problem through which TEDA was initially developed and tested against. This section describes how TEDA was developed through two main steps:

1. Control point targeting was introduced into an EDA with the objective of creating a version of FDC that used more than two parents.

2. A transitioning process was introduced through which TEDA should transition from FDC to the approach that was developed in step 1.

The cancer chemotherapy problem is an optimal control problem which models a patient undergoing chemotherapy treatment for a fixed period of time. It is explained in detail in Section 3.1.1. Each gene contains an integer value that represents a day in the treatment schedule and may either have a value of 0 if no treatment is to be carried out on that day or may have a value of between 1 and 15 representing a level of drug dosage if treatment is to be carried out. Control points, in the context of this problem, are genes corresponding to days in which drug treatment is applied. The task of an optimizer is to decide on which days treatment should take place and what dosage should be applied on those days.

5.1 EXPERIMENTAL METHOD

Table 5.1 shows the problem parameters with which all testing on the chemotherapy problem was done. κ is the toxicity of the drugs used, λ is the rate of tumour growth and θ is an absolute limit on the tumour size. C_{cum} is the maximum cumulative quantity of drugs permissable in the body at any one time, a requirement that exists due to the toxic effect of these drugs. See Section 3.1.3 for a more detailed explanation of what these parameters control.

These are the same parameters as used in previous work on this problem, including in the development of FDC [48].

Paremeter	Value
Min. Gene Value	0
Max. Gene Value	15
λ	0.7
C_{cum}	120
κ	0.045
θ	100.484

Table 5.1: Chemotherapy Model Parameters

Although our primary concern is when control points should take place and not at what value they should be set to, this problem remains an integer based problem, not a binary problem, and the methods of mutation and initialization reflect this.

5.1.1 *Mutation*

A standard mutation technique was used for all algorithms that use mutation. This mutation technique was the same method as that used by Godley [48]. Mutation is attempted a number of times equal to the chromosome's length and each time it is carried out with a probability equal to the mutation rate. Unless otherwise specified this mutation rate is fixed at 0.05. If mutation is to be carried out then a random decision will be made, with an equal probability, to either set a gene that is currently not set or to unset a gene that is currently set. If it is decided that an unset gene should be set then an unset gene will be chosen at random and will be set to a value that is randomly chosen between 1 and the maximum permissable value, in this case 15. If, on the other hand, it is decided to unset a gene then a gene will chosen to simply have its value set to 0. This ensures that the total number of genes set is not affected by mutation.

5.1.2 *Initialization*

Each solution in the initial population is produced by simply iterating through its length and choosing at random a value between 0 and the maximum possible value (15). This is used as a simple commonly used method of initialisation for fixed length problems with an integer encoding.

5.1.3 *Choosing an Allele*

The primary concern of this research is how to decide whether control points should be set or not and so all the approaches compared are binary algorithms. In UMDA marginal probabilities are used which simply define the probability of a control point being on or not and TEDA is an extension of this model. For the chemotherapy problem, however, genes take an integer value. For this reason a second step needs to be used to decide which value to set a gene to once a decision has been made to set a control point at that point. For consistency an identical approach is used across all algorithms. The simplest solution is to just select an allele a_i for gene i from the set A_i which contains all alleles found in all parents in the breeding pool at gene i where $a_i \in A_i$ and $a_i > 0$. This allele is chosen randomly with all possibilities having an equal probability. EDAs designed for non binary domains could, of course, be considered as an alternative basis for TEDA for this problem and as alternative algorithms to compare TEDA against but this simpler method was used as all other problems in this thesis are in the binary domain and the focus is producing a consistent version of TEDA for different domains and testing it in a consistent manner across these different domains.

It should be noted that, as this is an integer based problem, the fitness function considers the level of dosage rather than simply considering whether a dose is applied or not on a given day. Optimizing it as a binary problem could, therefore, potentially effect the result of these experiments. The same approach is used for all algorithms and so this research does give an indication for how well TEDA performs on this problem when compared to other algorithms, but perhaps the integer based aspect of this problem should be revisited in future work. A future project could involve the adaptation of TEDA into an algorithm able to intelligently optimize integer values.

5.2 FDC

This section explores FDC, described in Section 2.7.2, and its performance on the chemotherapy problem. First, FDC will be tested on this problem with a range of parameters, as understanding FDC's performance on this problem is the first step towards developing TEDA, an algorithm that is based on FDC. Secondly, we will begin the process of adapting FDC into TEDA by introducing a probabilistic method of selecting genes.

5.2.1 *Tuning FDC*

First, we shall optimize the performance of FDC by fine tuning fundamental evolutionary parameters such as the selection method and the mutation rate. We do this for two reasons.

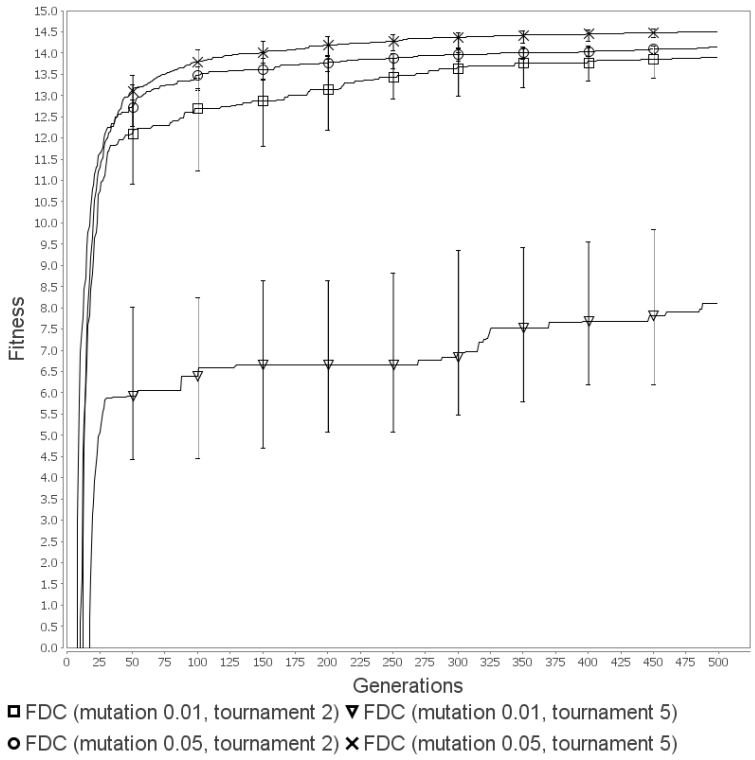

Figure 5.1: Tuning FDC for Chemotherapy Problem

Firstly, it ensures that we are comparing TEDA against the best possible implementation of FDC to establish whether it offers genuine improvement. Secondly, due to the similarities between FDC and TEDA, this provides us with a starting point for establishing with which configurations TEDA performs well.

5.2.1.1 *Mutation and Tournament Size*

The effect different mutation rates and tournament sizes have on the performance of FDC is shown in Figure 5.1. The results include tests with parameters used by Godley [45] in the development of FDC, a mutation rate of 0.01 and a tournament size of 2, and tests that feature a higher mutation rate of 0.05 and a tournament size of 5. As a higher tournament size increases how exploitative an algorithm is and a higher mutation rate makes an algorithm more explorative these tests cover both exploitative and explorative situations.

As we can see, the best performance is seen with a combination of a high mutation rate and a high tournament size. The most successful test uses a tournament size of 5 and a mutation rate of 0.05. However, in no situation does tournament size and mutation rate have a large effect on performance except for in the case where a combination of low mutation rate and large tournament size is used. The test *mutation 0.01, tournament 5* early on performs in a similar manner to the other approaches but quickly appears to become trapped at a low level of fitness. This suggests that any implementation of FDC works well unless it is overly exploitative. An exploitative EA that uses targeting is likely to lose diversity too quickly to

perform well as this aspect adds another limitation to the solutions generated due to the fact that, given two parents, the size of solution generated is now chosen deterministically and not stochastically. One should therefore ensure that any algorithm that uses targeting does, at least early on, achieve a similar level of exploration to FDC.

Based on these observations we can say which parameter configuration should be considered the default configuration for TEDA. Unless otherwise stated TEDA will use:

- A tournament size of 5.
- A mutation rate of 0.05.

To ensure a fair test FDC and all other algorithms for which tournament size and mutation rate is relevant will also use these parameters. We can also say that, when developing TEDA, we should ensure that it maintains a high degree of exploration in the early stages of the evolutionary process. In these early stages a key design requirement is that it should be equivalent to FDC and not to a standard EDA.

5.2.2 *FDC2 - proportional FDC*

Fitness Directed Crossover 2 (FDC_2) is a new variation of FDC that has been developed for this thesis which takes into account the fitness of parent solutions when deciding which control points should be set. FDC_2 follows the same basic sequence as FDC:

1. Two parents are selected from the population.
2. These are used to obtain a target number of control points, I_T using the same equation as in standard FDC, as described in Section 2.7.2.
3. Control points found in both parents are placed into the set S_{dup} and these are selected until I_T control points have been set.
4. Control points in the set S_{single}, those found in only one parent are selected at random.
5. Each Control Point c_i selected from S_{single} is set with probability $p(c_i)$. This is calculated through Equation 5.1 in which $f(x_{c_i})$ is the normalised fitness of solution x_{c_i}, the solution in which control point c_i is set and $f(x_{\not c_i})$ is the normalised fitness of the parent solution in which c_i is not set.

$$p(c_i) = \frac{f(x_{c_i})}{f(x_{c_i}) + f(x_{\not c_i})} \tag{5.1}$$

6. Steps 4 and 5 are repeated until the new solution has I_T control points set.

The only part of the process in which FDC_2 differs from FDC is step 5. This change was made as it ensures that control points in the fitter parent are more likely to be set than those in the

less fit parent, based on the assumption that the control points in fitter individuals are more likely to be of a higher quality and to form part of the optimal solution. This changes the process from one in which control points found in just one parent are selected randomly to one in which control points are selected in a stochastic manner based on parent fitness, and so makes FDC_2 more exploitative than FDC. To illustrate the FDC_2 process suppose that the parents selected are the two solutions given below. These solutions have 4 genes which, from left to right, are numbered 0, 1, 2 and 3. The normalised fitness of these two solutions is also given.

- Solution 1: 1100 (fitness = 0.4)
- Solution 2: 0111 (fitness = 0.5)

For these two solutions I_T will be equal to 3 and so 3 control points need to be chosen and set. With FDC_2 these control points will be chosen as follows:

1. Gene 1 will be set with probability 1 as it is in set S_{dup}.
2. All genes set in solution 1 only will be assigned a probability p where $p \approx 0.44$. This is calculated from Equation 5.1. Likewise all genes in solution 2 will be assigned a probability p where $p \approx 0.55$.
3. A random gene will be chosen. Suppose that gene 2 is chosen.
4. Gene 2 will be set with $p \approx 0.55$ as it is in solution 2. Suppose that it is set.
5. A random gene will be chosen. Suppose that gene 0 is chosen.
6. Gene 0 will be set with $p \approx 0.44$ as it is in solution 1. Suppose that it is set.
7. 3 control points have now been set and crossover is complete.

This change is also useful as it can be seen as a stepping stone to combining FDC with EDAs. As EDAs use a probability distribution to choose the value of genes they also use a stochastic process that takes into account the quality of each potential control point before setting it. In standard UMDA the probably of setting a control point is based on its frequency in the breeding pool whereas in FDC_2 it is based on the fitness of the solution in which it is found as only two parents are used. FDC_2 is therefore, in some ways, the equivalent of a targeted EDA using just two parents.

The results in Figure 5.2 show the performance of FDC_2 on this problem when compared to standard FDC. It appears as though proportional selection of control points has no effect on performance. Obviously with only two parents building the accurate probability model necessary to create an effective EDA is not possible although the fact that performance is not degraded by this approach suggests that with more parents it might be possible to produce a more effective algorithm.

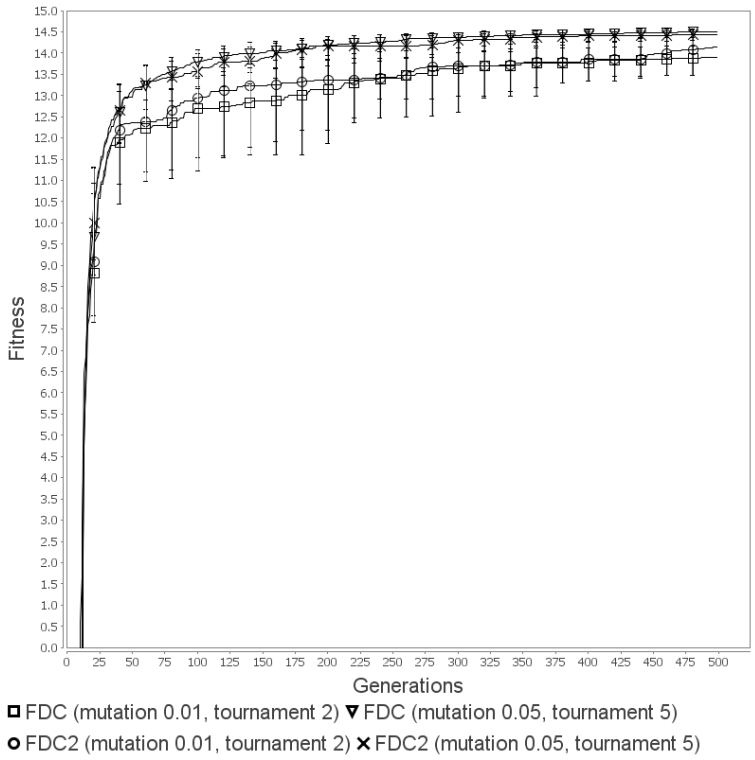

Figure 5.2: Performance of FDC_2 on Chemotherapy Problem

5.3 APPLYING CONTROL POINT TARGETING TO AN EDA

In this section we describe how control point targeting was first introduced into an EDA framework. In order to do this we need to consider the three steps that such an EDA should go through:

1. Building a probability distribution ρ.
2. Calculating I_T, the number of control points to use.
3. Choosing control points and setting them with their probability ρ until I_T control points have been set.

For step 1 it is possible to simply adopt the approach used by an established EDA. This is briefly discussed in Section 5.3.1.

In step 2, the targeting method adopted by FDC is used unaltered. As this method is designed for just two parents there is a question of which parents should be used. This issue is addressed in Section 5.3.4.

The most important issue to consider is how step 3 should be carried out. As only a subset of control points are even considered before I_T is reached clearly the order in which control points are considered is of key importance. This is discussed in Section 5.3.2.

This section can be thought of as detailing the development of an early version of TEDA which does not incorporate the transitioning process that defines the final version of TEDA. In this section the early version of TEDA being developed shall be referred to as TEDA_a.

5.3.1 *The Probability Model*

The probability model used by UMDA, as detailed in Section 2.8 is the one used in TEDA. This was used as it is a simple and popular EDA that has been shown to be effective at solving linear problems [92]. The choice of probability model was not explored in great detail at this stage as the probability model used, rather than being an integral part of TEDA, can simply be viewed as an exchangable part of the algorithm that may well be replaced with more complex multivariate models at a future date. UMDA is simply used in the initial development of TEDA as it is simple, well established and easy to implement.

A slight modification is applied to standard UMDA such that the probability of each gene is calculated as the sum of the normalised fitnesses of each member of the breeding pool which contains it instead of simply from the frequency of that gene within the breeding pool. The following two equations are provided to clarify this distinction. Equation 5.2 is the standard, frequency based, method of obtaining the probability of gene x_i, $\rho(x_i = 1)$. This is simply obtained by dividing the number of individuals in breeding pool B by the number of individuals with x_i set to 1 or true. In Equation 5.3 the cardinality of B is replaced by the sum of the marginal fitnesses f_x of all solutions within B and likewise the number of individuals within B where x_i is 1 is replaced by sum of the marginal fitnesses of these individuals. This modification is motivated by the size of the breeding pool in TEDA and the fact that it transitions from a GA. Although using frequency alone to calculate these probabilities is an effective technique for when the breeding pool is large, for example 50% of the population as is common in EDAs, TEDA is a process that transitions from initially using only 2 parents and so the use of normalized fitnesses ensures that genes are weighted according to some measure of quality throughout the evolutionary process.

This is not without precedent. In Section 2.8 the theoretical background of EDAs is discussed and it is explained that they arose from work on predicting from one generation how many copies of a gene is likely to exist in the following generation [8]. These predictions were calculated from the sum of fitnesses within the population in a manner similar to the modified version of UMDA used here.

$$\rho(x_i = 1) = \frac{1}{|B|} \sum_{x \in B, x_i = 1} 1 \tag{5.2}$$

$$\rho(x_i = 1) = \frac{1}{\sum_{x \epsilon B} f_x} \sum_{x \epsilon B, x_i = 1} f_x \tag{5.3}$$

5.3.2 *Control Point Selection*

In this section we discuss 4 different methods of choosing control points. These are each illustrated through the example of choosing control points from the solutions in Table 5.2. Each solution contains 10 genes numbered 1 to 10.

Table 5.2: Control Point Selection Example - Solutions

Solution 1	0000010001
Solution 2	1110010001
Solution 3	0101000101
Solution 4	0111000101
Solution 5	0101000011

For the sake of simplicity we assume that these solutions are of equal fitness and so the probability of each control point is simply $n/5$ where n is the number of solutions in which it occurs. Based on this assumption, Table 5.3 gives the probability of each gene.

We also assume that 4 is the target number of control points (I_T).

5.3.2.1 *Two Set*

All control points appearing in 100% of the breeding pool, and so with a probability of 1, are chosen at random first. These are placed in the set S_{all}. If this set is smaller than I_T then control points in less than 100% but more than 0% of the breeding pool, control points in the set S_{some}, will be selected at random until the control point target is reached or all control points in S_{some} have been considered. This approach is designed to be equivalent to FDC. FDC also first uses control points found in both parent solutions and only when these are exhausted uses control points found in only one of the two parents (see Section 5.2). This method therefore helps to fulfil our objective that, with a breeding pool size of two, TEDA should behave exactly like FDC.

In the above example the only gene in S_{all} will be the final gene, gene 10. This will be set first. Following this, the remaining 3 genes that need to be set to reach I_T will be selected at random from the set (1,2,3,4,6,8,9) and set with their respective probability. Genes 5 and 7 are excluded as they appear in no solutions and so have a probability of 0.

Table 5.3: Control Point Selection Example - Probabilities

Gene	Probability
1	0.2
2	0.8
3	0.4
4	0.6
5	0
6	0.4
7	0
8	0.4
9	0.2
10	1

5.3.2.2 *Roulette*

This method is similar to the fitness proportionate roulette wheel method for selecting parent solutions (see Section 2.5). Control points are selected probabilistically with a probability of ρ_i/t whereρ_i is the marginal probability of control point i, defined as the total fitness of all solutions in which i is found and t is the sum of the marginal probabilities of all the control points. As with the other approaches, when a control point is selected it is either set or rejected based on its marginal probability and we continue to select control points, without replacement, until I_T control points have been set.

To apply this technique in the case of the solutions given above, first of all which proportion of the roulette wheel is to be allocated to each gene has to be determined. The necessary calculations are carried out below. In this description the section of the roulette wheel allocated to gene i is denoted as $section_i$ and is expressed in terms of the two bounding values between which this section falls.

$$
\begin{aligned}
&\text{Roulette wheel total size} = 0.2 + 0.8 + 0.4 + 0.6 + 0 + 0.4 + 0 + 0.4 + 0.2 + 1 = 4 \\
&0.2/4 = 0.05 \therefore 0 \leqslant \mathtt{section}_1 < 0.05 \\
&(0.8/4) + 0.05 = 0.25 \therefore 0.05 \leqslant \mathtt{section}_2 < 0.25 \\
&(0.4/4) + 0.25 = 0.35 \therefore 0.25 \leqslant \mathtt{section}_3 < 0.35 \\
&(0.6/4) + 0.35 = 0.5 \therefore 0.35 \leqslant \mathtt{section}_4 < 0.5 \\
&(0/4) + 0.5 = 0.5 \therefore \mathtt{section}_5 \text{ doesn't exist} \\
&(0.4/4) + 0.5 = 0.6 \therefore 0.5 \leqslant \mathtt{section}_6 < 0.6 \\
&(0/4) + 0.6 = 0.6 \therefore \mathtt{section}_7 \text{ doesn't exist} \\
&(0.2/4) + 0.6 = 0.65 \therefore 0.6 \leqslant \mathtt{section}_8 < 0.65 \\
&(0.4/4) + 0.6 = 0.75 \therefore 0.65 \leqslant \mathtt{section}_9 < 0.75 \\
&(1/4) + 0.75 = 1 \therefore 0.75 \leqslant \mathtt{section}_1 0 < 1
\end{aligned} \tag{5.4}
$$

After this roulette wheel has been built a random value between 0 and 1 will be chosen. If this value is 0.52, for example, $\mathtt{section}_6$ will be chosen and so gene 6 will be set. If it is 0.19 then $\mathtt{section}_2$ will be chosen and gene 2 will be set.

5.3.2.3 *Unordered*

This is the most explorative of the 4 approaches discussed and is the closest to a standard EDA. Control points are selected completely at random and set with their corresponding probability until I_T is reached.

If this technique is used on the above solutions then first of all a random number will be chosen between 1 and 10. If 4 is chosen then gene 4 will be selected and set with probability 0.6. If gene 4 is set then another random number will be chosen between 1 and 10, excluding 4, and the process is repeated.

5.3.2.4 *Ordered List*

This is the most exploitative of the 4 techniques. control points are ordered based on their probability and this ordered list is iterated through. If there are several control points with the same probability then they will be placed in this list in a random order. We iterate through this list and either set or reject each control point that we come to with its marginal probability before moving onto the next control point. This process continues until I_T control points have been set.

In the above example the genes with a non zero probability will be placed in the following order: $10, 2, 4, (6, 8, 3), (1, 9)$. As genes 6,8 and 3 have an identical probability of 0.4 their ordering will be chosen at random. The same applies to genes 1 and 9 which both have a

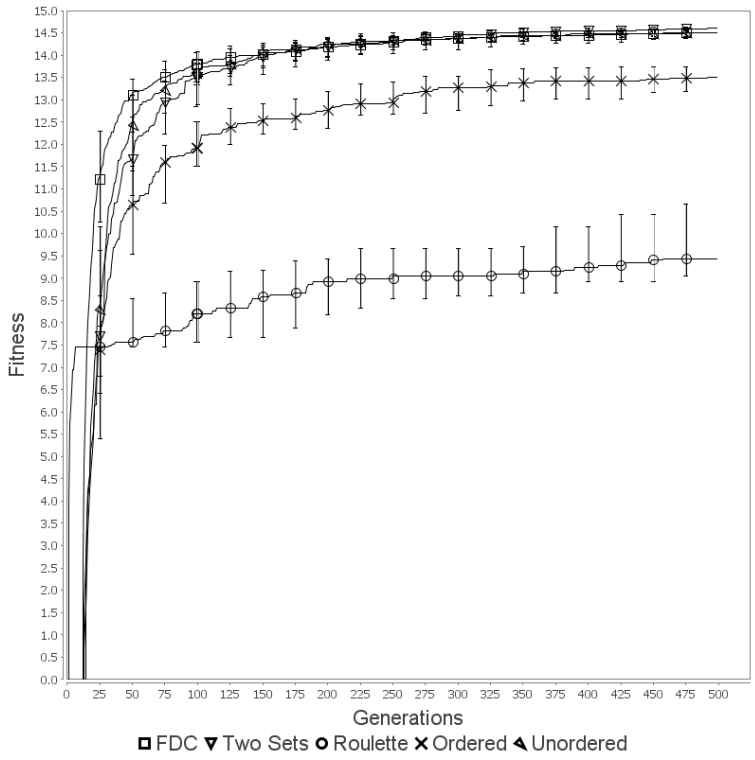

Figure 5.3: TEDA$_\alpha$ - With 10 Parents

probability of 0.2. Gene 10 will then be selected as it is first in this list. As it has a probability of 1 it will always be set. Gene 2 will be selected next and set with a probability of 0.8. Gene 4 will be selected next, and so on.

5.3.2.5 *Results*

To test these four methods, three sets of tests were run:

- In the first test a breeding pool size of 10% of the population is used.
- In the second test a breeding pool size of 20% of the population is used.
- In the third test a breeding pool size of 40% of the population is used.

In all tests the breeding pool was selected through truncation selection as this is a commonly used selection technique for EDAs [18][122]. Figures 5.3, 5.4, 5.5 show the results for 10 parents, 20 parents and 40 parents respectively. Each graph also shows how FDC performs for comparison.

As we can see from these graphs only two of the methods perform well, Unordered and Two Set. The Ordered List method is probably too exploitative and leads to a loss of diversity. It should be noted, however, that in these tests an exploitative selection method, truncation selection, is used throughout the test and so it is to be expected that the use of exploitative control point selection methods would create an overly exploitative algorithm. The Roulette

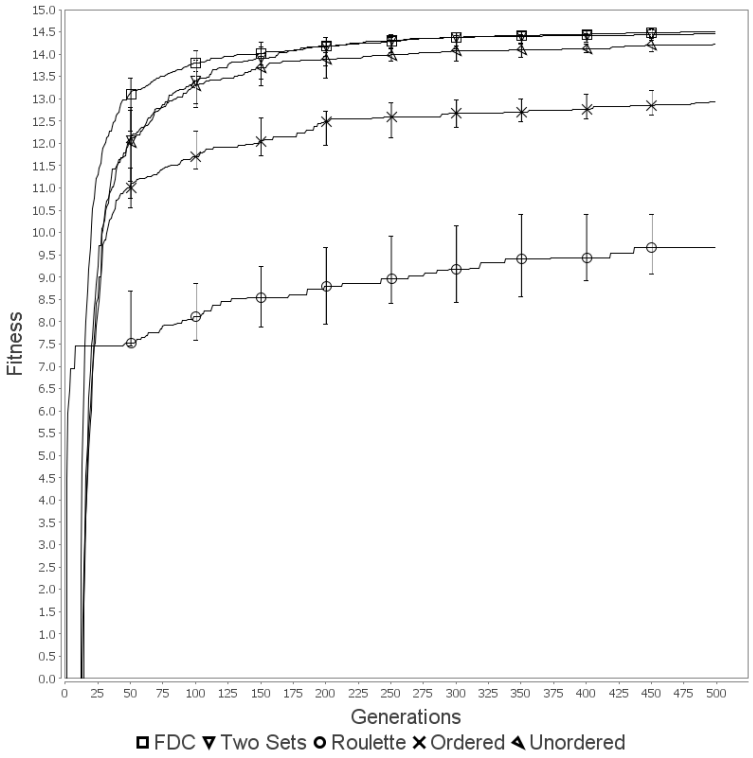

Figure 5.4: TEDA$_\alpha$ - With 20 Parents

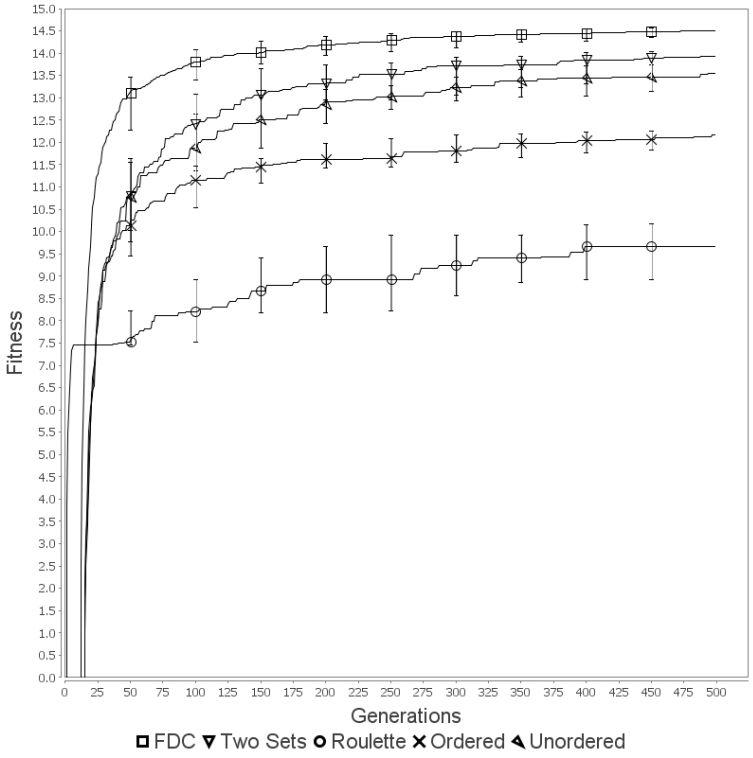

Figure 5.5: TEDA$_\alpha$ - With 40 Parents

method very quickly achieves a high rate of fitness in the first few generations, before the other approaches, but then quickly stabilises at a much lower maximum fitness than the other approaches. It appears that this method, too, leads to premature convergence.

The Two Set and Unordered methods perform to a similar standard to FDC with 10 parents and perform slightly worse with 20 parents. With 40 parents their performance is noticeably worse. With 10 parents it is hard to say whether Two Set or Unordered performs best although, as the number of parents is increased, Two Set appears to gain the advantage.

One possible explanation is that, as the size of the breeding pool is increased, the set of control points that are found in at least one of the solutions within it and so have a probability greater than 0, a set that we shall we shall denote as S, grows. In the Unordered method the probability of control point i being set as the first control point to be set is $1/|S| \times \rho_i$ as it first needs to be selected before being set with its marginal probability ρ_i. Clearly when S is large the value of ρ_i is less important as the low value of $1/|S|$ means that all control points, even those with a high value of ρi are unlikely to be set. If I_T is also small then the only control points set are the few which happen to be randomly selected. For this reason, when the breeding pool is large the two set method acts as a way of ensuring that strong control points are still selected and the process of selecting control points does not become essentially random as is the case with the unordered method.

Of course this relies on the assumption that, as the breeding pool becomes large, there are still control points strong enough to appear in every individual in the breeding pool. In general the unordered method may lose good contol points when a larger breeding pool is used that the two set method does not lose.

5.3.3 *Selection*

To combine FDC with an EDA we must choose a flexible selection method to reflect the fact that GAs including FDC and EDAs tend to use different selection techniques to reflect differences between these two approaches. FDC was designed with a selection technique that has a low selection pressure. In Godley's work [48], tournament selection with a tournament size of 2 was used and in the earlier part of this chapter we have established that we should use a tournament size of 5 for this algorithm. EDAs, by contrast, commonly use truncation selection and therefore use a much higher selection pressure [18][122]. In order to achieve this flexibility, in this section we explore different selection methods that have a tunable selection pressure. The two selection techniques that we have chosen to explore are as follows:

- Tournament Selection: This is a simple selection method that is popular for GAs. It is explained in more detail in Chapter 2.5. It is tunable as a larger tournament leads to a higher selection pressure.

- Two Pool Selection: This is a method developed specifically for TEDA that has the advantage that it can 'mimic' either tournament selection as used in FDC or truncation selection, as is popular in EDAs.

5.3.3.1 *Two Pool Selection*

Two Pool Selection is performed through the following process:

1. a selection pool is created through truncation selection of size s.
2. the breeding pool of size b is then selected from this selection pool through tournament selection.

A smaller value of s will increase the selection pressure as only the s fittest solutions may be used to breed new solutions. When $b = s$ this selection method is identical to truncation selection. When s is equal to the size of the population this method is identical to tournament selection. A larger value of b will cause the algorithm to behave more like an EDA as it will cause it to create a model from a larger number of parents. With b is equal to 50% of the population TEDA will build a model from half the population, equivalent to many EDAs found in literature [69], whereas with a b of 2 it will essentially become a probabilistic FDC, identical to FDC_2 as discussed in Section 5.2.

As 10% and 20% of the population have so far been observed to be good sizes for a breeding pool, for this section we have conducted two tests using two pool selection. Figure 5.6 shows the results of tests where b is equal to 20 and Figure 5.7 shows the results of tests where b is equal to 10. In both cases we use a value of s that is equal to $b \times 2$. Tests were run using the two better performing methods of control point selection described in the previous section, two set and unordered. Once again we include FDC in these graphs for comparison.

With a b of 10 and an s of 20 it appears from Figure 5.7 that the Two Pool method when used in combination with the Two Set method of control point selection outperforms FDC later on in the test although early on it less effective.

5.3.3.2 *Tournament Selection*

In the tests in this section tournament selection was used to produce a breeding pool with a size of 10% of the population. Tests were carried out with a tournament size of 5%, 10% and 20%. The results in Figure 5.8 show how $TEDA_a$, as it has been developed so far, performs with these different tournament sizes when used in combination with the two set control point selection method. The results in Figure 5.9 show how it performs when used in combination with the unordered control point selection method.

None of the results presented here appear to outperform FDC, unlike some of the results in the previous section using two pool selection. Using unordered control point selection with all tournament sizes $TEDA_a$ appears to perform worse than FDC, whereas with two set control

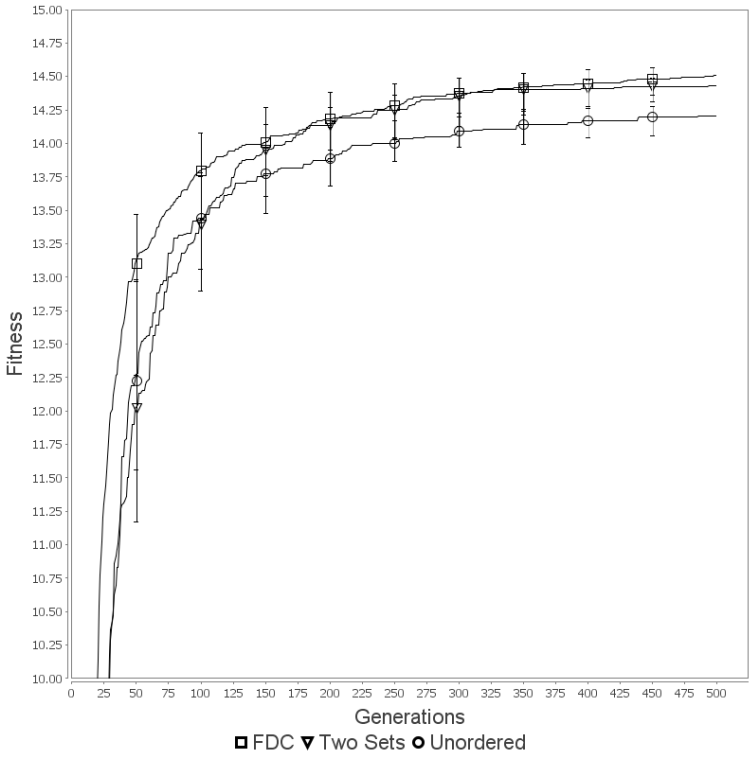

Figure 5.6: TEDA$_a$ - Selecting 20 Parents from 40

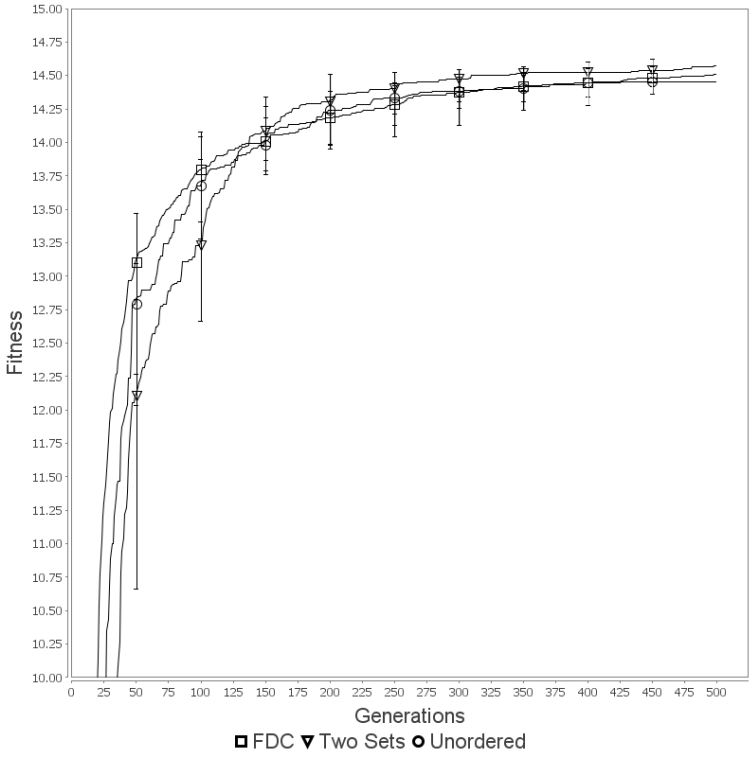

Figure 5.7: TEDA$_a$ - Selecting 10 Parents from 20

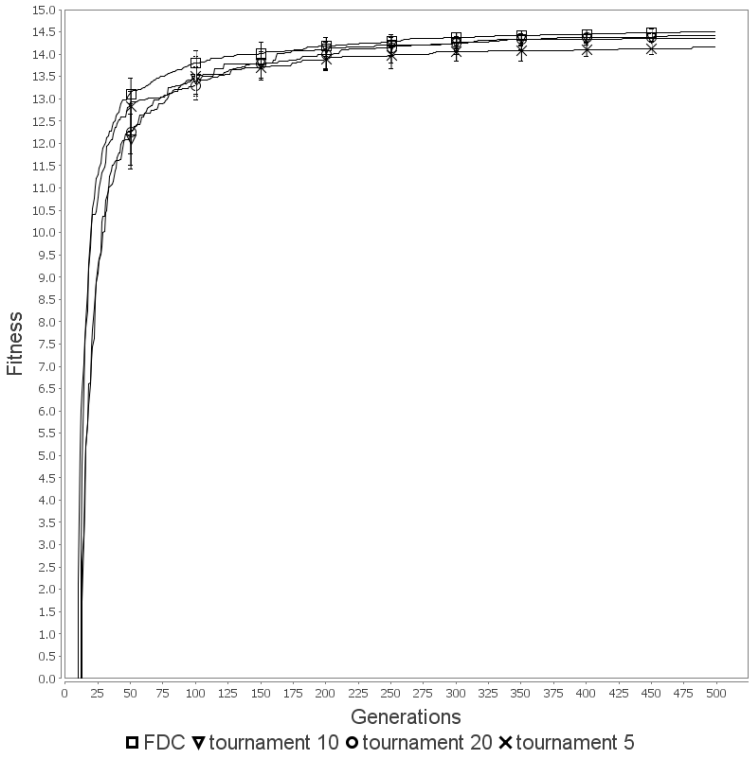

Figure 5.8: TEDA$_a$ - Using Tournament Selection with Two Set Control Point Selection

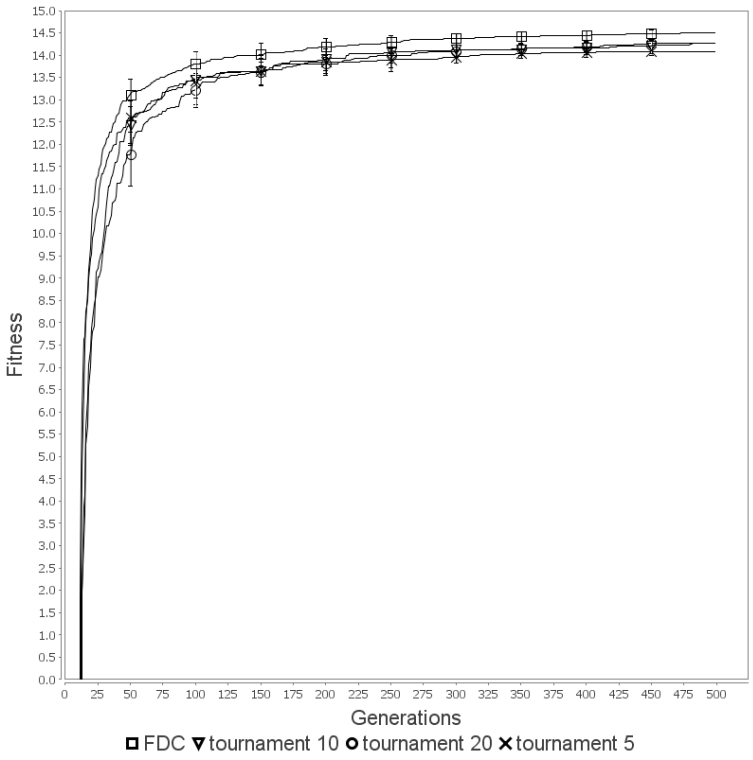

Figure 5.9: TEDA$_a$ - Using Tournament Selection with Unordered Control Point Selection

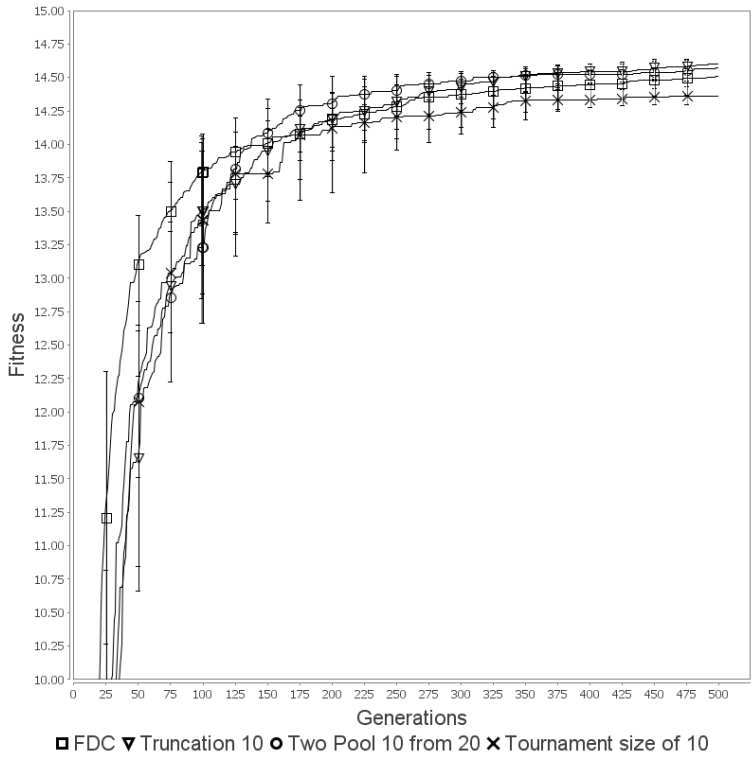

Figure 5.10: TEDA_α - Selection Summary

point selection TEDA_α's performance is similar to that of FDC. The size of tournament does not appear to have a large effect although, using both control point selection methods, a larger tournament size appears to lead to slightly worse performance earlier on.

5.3.3.3 *Summary of Selection Methods*

In this section we compare simple truncation selection with tournament and two pool selection. For each case the configuration that appeared in the previous tests most likely to offer an improvement over FDC later on in the test is compared. As TEDA will ultimately operate like FDC before convergence the performance of each approach early on was not considered. In all cases a breeding pool of size 10 is selected. Figure 5.10 therefore shows the performance of TEDA_α using truncation selection with a breeding pool size of 10% of the population, two pool selection with a b of 10% and an s of 20% and tournament selection with a tournament size of 10% used to select a breeding pool of size 10%. In all cases the two set method of control point selection is used.

It can be seen from these results that FDC performs better than all of these approaches for the first 100 generations. There is, however, some indication that, later on in the test, the two pool method might perform slightly better than both of the other selection methods and FDC. As we have established that this is the method that will allow us to transition from behaviour matching that of FDC to behaviour matching that of an EDA, we shall begin the process of introducing this transitioning into TEDA, using the two pool selection method, with

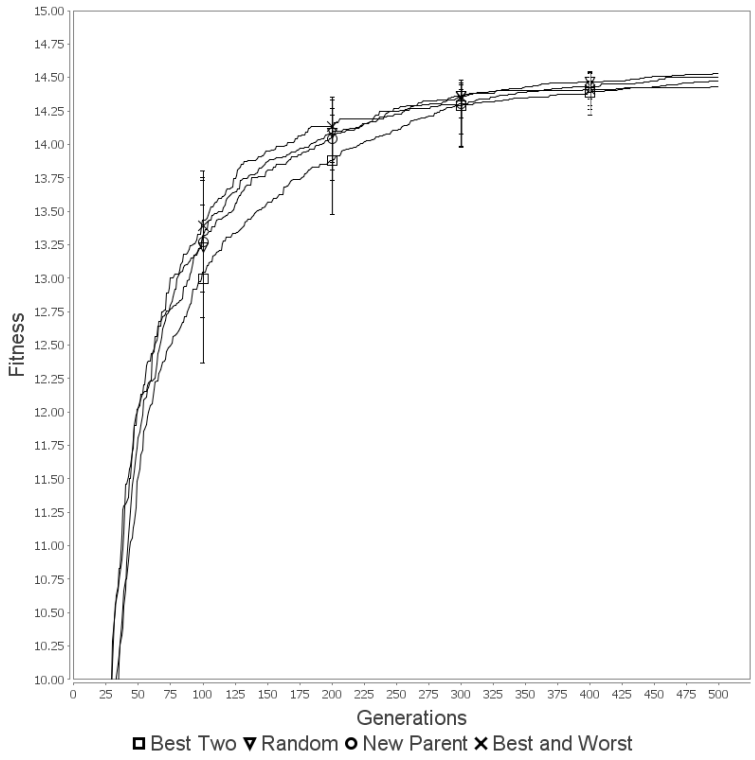

Figure 5.11: TEDA$_\alpha$ - Second Parent for Targeting

the intention that we may be able to create an algorithm that is able to perform better than FDC in both the earlier and the later stages of the test.

5.3.4 *Targeting*

As EDAs generate new solutions from more than 2 parent solutions a decision needs to be made for TEDA as to which 2 of these solutions should be used in the FDC rule (see Section 5.2) to decide on the control point number. In the tests described here the fittest parent is always used as one of the two parents and the question being asked is which individual to use as the second, less fit parent. In the tests that have been run up to this point, the second parent was always the least fit parent in the EDA breeding pool. Aside from this possiblity, the *Best and Worst* method, there exists several alternative ways of solving this problem. In the *Best Two* method the fittest 2 parents in the breeding pool are used. In the *Random* method, the 2nd parent is selected at random from the breeding pool. In the *New Parent* method a new parent that is not necessarily in the breeding pool at all is selected from the selection pool using tournament selection. The results in Figure 5.11 show how TEDA$_\alpha$ performs when using these different techniques.

From the results in Figure 5.11 it appears as though the only technique that noticeably performs worse is the best two method. This may be because the difference between these two

solutions is likely to be small and so, using these, targeting is likely to be weak. Using the best and worst parent should, in theory, cause the fastest movement in control point number and so quickly drive the number of control points used in the appropriate direction. The risk with this approach is that, especially before the population has converged, the least fit individual might have any number of control points set. In this early phase of the test the least fit individual is essentially a random individual and so any lessons learnt from it should be treated with caution. The control point number might, therefore, be driven in the wrong direction. However, this sort of error is acceptable in the early, highly explorative, stage of the process.

5.3.5 *Elitism*

Elitism is the process of keeping the top solution or solutions from one generation to the next. It has been explored for EDA based techniques and is used with FDC [48]. One example of the use of elitism in EDAs was in a paper by Ahn [2]. Ahn demonstrated that elitism is able to improve the performance of Compact Genetic Algorithm (CGA), an algorithm that maintains a probability model ρ which is repeatedly used to generate two new solutions which in turn are used to improve ρ through a learning rule. CGA, therefore, uses an EDA style probability model to save memory and remove the need for a population. Although CGA is effective for easy problems it is less able to solve difficult problems as it is prone to losing useful information as it does not store good solutions that have already been discovered. It is possible to compensate for this by increasing the selection pressure which CGA exerts, however this involves increasing the tournament size which effectively means generating more individuals and so there is a memory cost associated with this. It is also difficult to compute by how much the selection pressure should be increased for a given problem. Ahn found that, when elitism was added to CGA, this provided a way of saving useful information and so this removed the need to increase selection pressure. Despite this study, literature describing elitism in more conventional EDAs appears to be more limited.

In the case of TEDA the question of whether or not to use elitism is complicated. The two main reasons for using elitism in TEDA is firstly to allow it to behave like FDC and secondly to ensure that good solutions are not lost. However, the problem is that TEDA always uses the fittest individual in the breeding pool to decide on the number of control points to use. Therefore, this individual has a large impact on the breeding process. After convergence has taken place and the two pool selection method is behaving the same as truncation selection this individual will, in fact, be the fittest individual in the population. If this is an elite individual that survives from generation to generation then we have a situation where one fit individual is heavily influencing the breeding process generation after generation, thus causing a loss of diversity.

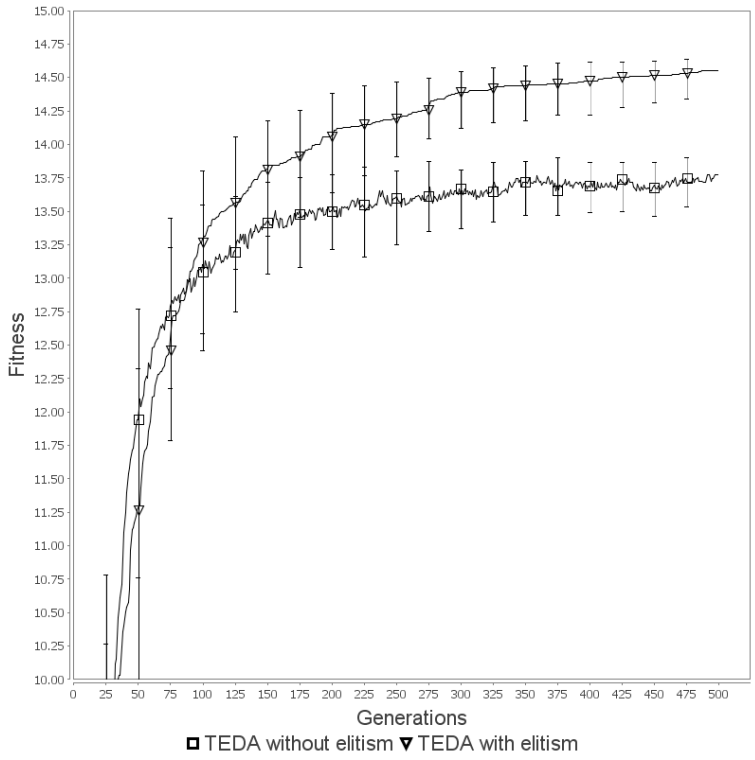

Figure 5.12: TEDA_{α} - Elitism

Figure 5.12 captures TEDA's performance with and without elitism. *TEDA with elitism* was run with a single elite individual whereas *TEDA without elitism* does not use any elitism.

The results shown in Figure 5.12 suggest that elitism leads to improved performance in TEDA from around generation 100 onwards. As we have not yet introduced transitioning the concern that elitism could lead to a loss of diversity post convergence is not being tested but it is still our decision that elitism should form part of TEDA. Post convergence a loss of diversity is a less significant issue than it is early on in the test. As a result of this test, all following tests will be carried out with elitism.

5.3.6 *A Note on the Number of Children Generated*

We now have a system for selecting a breeding pool and for generating new solutions from this breeding pool. One question that still needs to be answered is how many children should be generated from the breeding pool. There is an efficiency issue associated with generating a small number of children from each set of parents as the process of selecting parents and building a probability model will have to be carried out a large number of times to generate an entire population. Conversely, generating a large proportion of the population from a small number of parents will cause the population to lose diversity. Traditional GA crossover techniques such as one and two point crossover will, by default, produce two children for

every two parents. It is important that there is no loss of diversity and that TEDA should behave like a GA before convergence and so generating two new solutions from two parents seems like a logical rule to follow. After convergence, a potential loss of diversity is less of an issue and an increased selection pressure means that the probability of selecting a significantly different set of parents each time a breeding pool is selected is low anyway. At this point, it does seem sensible to take advantage of the efficiency gains associated with generating a larger number of solutions for each breeding pool. For this reason we have chosen to follow the policy that the number of children generated should be the same as the size of the breeding pool.

5.4 TRANSITIONING

We have now seen in Section 5.3 that introducing more parents into the FDC approach and adopting an EDA based breeding strategy may help an algorithm to ultimately find fitter solutions but that early on standard FDC still performs better. In this section we explore whether, by starting with a standard FDC and later moving towards an EDA, we are able to produce an algorithm that performs as well as standard FDC early on and later benefits from the superior exploitative capabilities of an EDA.

In order to achieve this we need to first of all decide when to transition from a GA to an EDA and also decide how to transition. The main principle is that the transitioning process should complete and the algorithm should be behaving like an EDA after the population has started to converge on the global optimum. These questions will be discussed in this section. This section describes the development of the novel transitioning technique that ultimately became, along with the technique for introducing targeting into EDAs described last section, one of the main two novel, distinguishing features of TEDA.

5.4.1 *The Transitioning Technique*

The difference between EDAs and GAs is that EDAs build a model from a large proportion of the population whereas GAs simply carry out crossover on two members of the population. In order to build an effective model, an EDA will need to use a set of solutions that are highly likely to be fit. It has already been established that EDAs are more likely to use a high selection pressure method of selection like truncation selection than FDC.

In Section 5.2.2 it was established that a modified version of FDC, FDC_2, performs to an equal standard on the chemotherapy problem as the original version of FDC. In FDC_2 control points are given a probability based on the fitness of the solutions in which they are found. This process is essentially a two parent equivalent of the modified version of UMDA that is used as the basis of TEDA. This demonstrated that the only significant differences between FDC and

TEDA$_a$, the pre transitioning version of TEDA that was developed in the earlier part of this chapter, are:

- EDAs use a higher selection pressure to select parent solutions than FDC.
- EDAs breed new solutions from a larger number of parents than FDC.

We therefore use the transitioning process to control these two variables. The size of the selection pool, s and the size of the breeding pool, b. A larger selection pool creates a lower selection pressure, as explained in Section 5.3.3.1.

The transitioning process should therefore consist of adjusting s and b between pre specified maximum and minimum values s_{max}, s_{min}, b_{max} and b_{min}. Each generation a convergence score should be obtained. This convergence score should be used in an equation that will ensure that before the population has converged s is close to s_{max} and b is close to b_{min} and later the reverse is true. s_{max} should equal the size of the entire population and b_{min} should equal 2 as this will mean that early on new solutions are bred from 2 parent solutions selected through simple tournament selection. b_{max} and s_{min} should be equal in order to ensure that the algorithm ultimately transitions to using pure truncation selection.

5.4.2 *When to Transition*

In order to transition we first need to know how to measure when the population has converged. In this section we explore different methods of using various measurements of convergence as well as simply transitioning over time. The transitioning methods can be divided into 3 main categories.

- transitioning at a fixed rate over time.
- measuring population convergence using the variance in fitnesses across the population.
- measuring population convergence using the variation between chromosomes in the population.

Measuring population convergence through chromosome variation is one technique that has recieved attention in the literature. A technique developed by Shimodaira [114] is the Diversity-Control-Orientated Genetic Algorithm. This measures the hamming distance between solutions and uses this to ensure that a diverse pool of individuals survive from one generation to the next [130]. The Shifting-Balance Genetic Algorithm also measures the hamming distance between pairs of solutions and obtains a *containment factor* that is used to give an indication of how diverse the population is. This is used to determine a ratio of parent individuals to be selected to form the next population according to their diversity. The remaining parents will be selected according to their fitness [95].

5.4.3 *Transitioning Over Time*

Transitioning from FDC to an EDA at a fixed rate during the course of the test is a simple, naive method of carrying out transitioning. As it cannot be assumed that different problems converge at the same rate this method is used primarily to demonstrate whether transitioning can, in principle, work rather than to be considered as a reasonable transitioning method. By looking at any of the results in the previous section it is clear that fitness levels tend to level out after approximately 200 generations. We shall therefore create a transitioning situation where the algorithm will move from FDC to an EDA in exactly 200 generations. As mentioned, one would not normally be aware of how long a given algorithm takes to converge on a given problem but if we establish that transitioning can produce an improved performance when it takes place at an approximately optimal rate then we can move on to exploring how this optimal rate may be determined dynamically.

Our method of transitioning over time is to simply decrement and increment the values of s and b respectively by a small amount each generation from their starting values of s_{max} and b_{min}. Equation 5.5 shows how the values of s and b at generation g, s_g and b_g, are calculated. δ_s and δ_b is the amount by which s and b are changed each generation. In this test δ_s is equal to $(s_{max} - s_{min})/200$ and δ_b is equal to $(b_{max} - b_{min})/200$ so that the transitioning process is complete after 200 generations.

$$\begin{aligned} s_g &= s_{max} - g\delta_s \\ b_g &= b_{min} + g\delta_b \end{aligned} \tag{5.5}$$

This approach was tested with two different values of b_{max} and s_{min}. In one test, both of these values were set to 10% of the population and in another test both of these values were set to 20% of the population. These values are derived from the testing done in Section 5.3.4, which suggested that a smaller pool of around 10% of the population may be the optimal size. A slightly larger pool of 20% is also used in this section due to the fact that the optimal maximum size of breeding pool may be larger than the optimal fixed size of breeding pool as observed in previous tests.

The results in Figure 5.13 compare the performance of TEDA using this time based transitioning technique with FDC.

As we can see from Figure 5.13 this technique works well, possibly performing better than FDC, especially later on in the run. A value for b_{max} and s_{min} of 10% appears to perform better than 20%.

This result shows that transitioning can work and we will now look at methods of achieving this performance that do not rely on knowing the convergence rate for each problem.

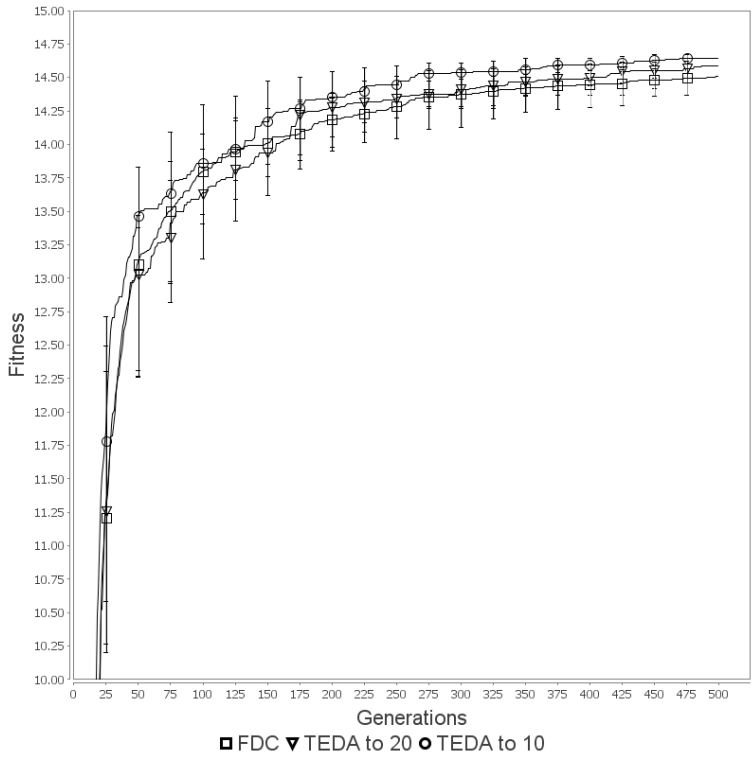

Figure 5.13: Transitioning over Time - Performance

5.4.4 *Types of Convergence Measures*

The two main options for measuring convergence are to use the variation in fitness within the population or to use the genetic variation between chromosomes.

Fitness Variation has long been considered as a method for measuring convergence. As explained in a recent paper by Crepinsek [34], the assumption is that a population that has converged will contain only solutions with a similarly high fitness value as all low fitness solutions will have been eliminated. The root of this concept can be traced back to the work of Rechenberg in 1973 [106], where a theory referred to as the *1/5th rule* is presented which states that an algorithm should shift from an explorative approach to an exploitative one when one fifth of changes result in an improvement in fitness. In [24] Burke describes how the number of unique fitness values or the proportion of the population covered by each fitness value could be used to measure convergence.

The problem with using fitness is that, for some problems, the fitness function is volatile, leading to situations where a sharp drop in fitness variance may not necessarily indicate convergence. Genetic Diversity has therefore also been used to determine population diversity and several different methods are discussed in [24]. Genetic Diversity has, for example, been used to control crossover, mutation and selection in a GA proposed by McGinley in 2011 [90]. This technique, referred to as the SPD method, obtains the sum of the euclidian distances

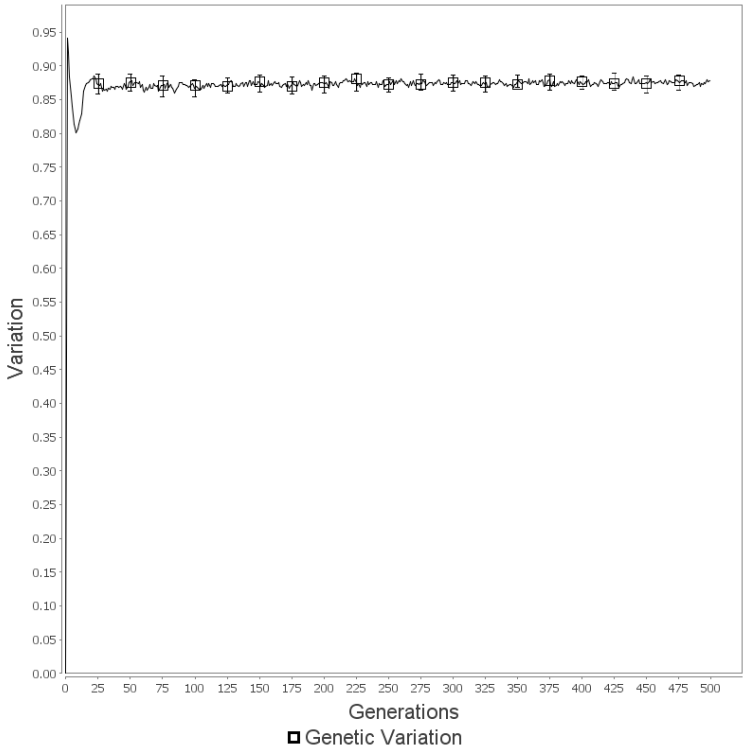

Figure 5.14: Change in Genetic Variation

between the average individual in the population and the rest of the population. This is used as the diversity measure.

In order to establish which convergence measure is best for TEDA, first of all we select the fittest $n\%$ of solutions within the population. Results are presented using both 50 and 10 as values of n and for both two measures of convergence are presented, one based on fitness variation and one based on genetic variation. The fitness variation measure used is simply one standard deviation from the mean fitness of the top $n\%$ of individuals. The genetic variation is obtained using the Hamming distance as follows:

1. Select the fittest $n\%$ of solutions.
2. Divide these solutions randomly into pairs.
3. For each pair obtain d, the Hamming distance.
4. Calculate $v = \frac{d}{l}$ where l is the length of the chromosome.
5. Calculate $\bar{v}$, the mean value of v across all n solutions.

The results of performing this analysis are shown in Figure 5.14, which shows the genetic variation measured from a 500 generation run of FDC. Figure 5.15 shows the standard deviation of fitnesses measured from the same run. In this problem fitnesses range from a high of around 15 to a low of below -6×10^6 and so the standard deviation of fitness is, early on, so high that

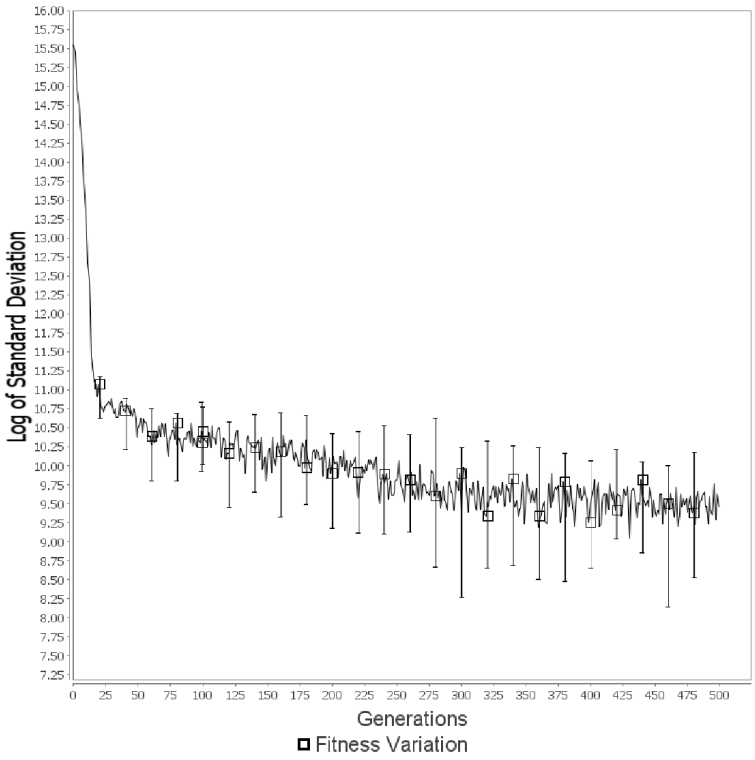

Figure 5.15: Change in Fitness Variation

a logarithmic scale has been used to represent these values. The fact that fitness variation drops very suddenly, in the first few generations, from an extremely high value to a much lower value as very poor solutions are eliminated early on will make it difficult to reliably use fitness as a means of measuring convergence. It is also problematic that this is dependent on the problem. Using fitness to indicate when the population has converged may necessitate some prior knowledge of what magnitude the optimal fitness for a problem is. Genetic variation is, regardless of the problem, limited by the total length of the chromosome and so is potentially a much more manageable system of measuring convergence. It appears, however, that in this situation genetic variation remains high throughout the test. Due to the difficulties of using fitness variation it was initially decided that we should use genetic variation and run a series of tests to assertain whether a useful change in variation is ever seen for this problem.

5.4.5 *Transitioning Through Genetic Variation*

This section shows the results of various experiments in which we attempted to control transitioning by measuring genetic variation. We measure genetic variation as described above and then use it to control s and b. Initially we use a technique whereby s and b move between s_{max} and s_{min}, b_{max} and b_{min} as in equation 5.6. In this equation $\bar{v}$ is the mean Hamming distance between pairs of solutions.

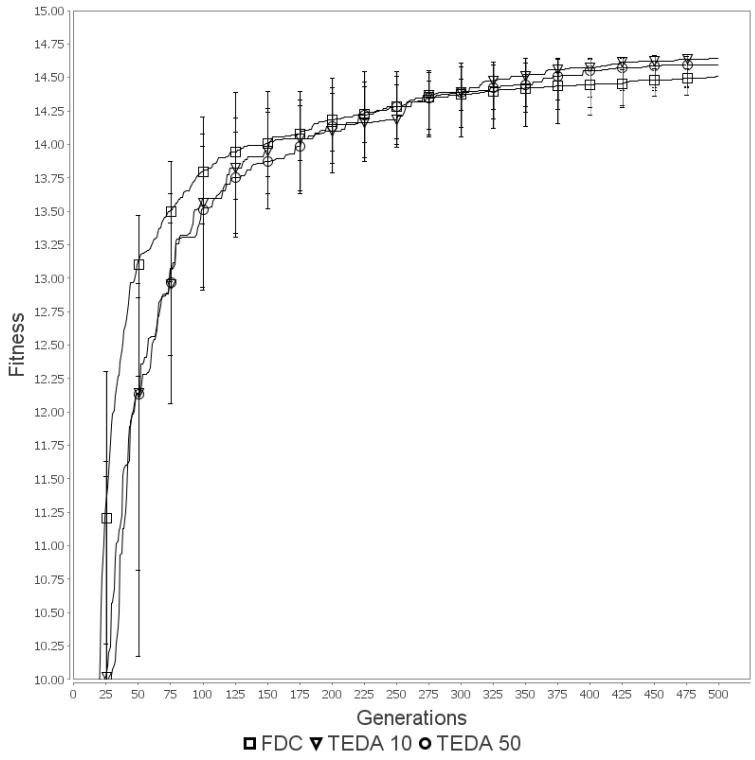

Figure 5.16: Transitioning Through Genetic Variation - Performance

$$s = s_{min} + \bar{v}(s_{max} - s_{min})$$
$$b = b_{max} - \bar{v}(b_{max} - b_{min}) \tag{5.6}$$

The results in Figure 5.16 show how well TEDA performs using this technique. In this test s_{min} and b_{max} were given a value of 10%.

The results in Figure 5.17 show how the convergence score itself ($\bar{v}$) changed over the course of the test. This is equivalent to the results in Figure 5.14 except that it shows how the variation changes in TEDA while it is actually being used for transitioning, instead of in FDC.

As can be seen from the results in Figure 5.16, in this situation TEDA performs worse than FDC early on. Both techniques perform to a similar standard later on and it is possible that TEDA has the slight advantage. It is interesting that this performance is worse than transitioning over time.

When we look at the change in variation in Figure 5.17, we can see that the variation level does show a smooth change over time. This is in contrast to when we measured the change in variation through FDC (see Figure 5.14), so it seems as though this measure shows a higher degree of convergence when applied to TEDA than FDC. It can be seen that it takes around 75 generations for variation to reach its minimum value. This means that transitioning takes around 75 generations, much less time than the 200 generations that it was allowed to take during the fixed rate transitioning test. The rate of transitioning appears to be the only

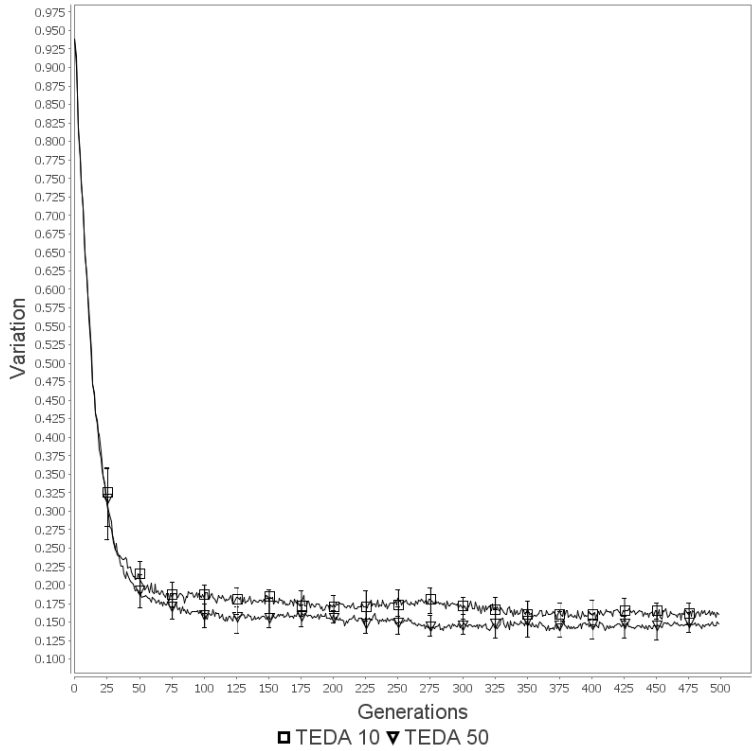

Figure 5.17: Transitioning Through Genetic Variation - Variation Measure

difference between this test and the fixed rate test and so this overly fast transitioning is a possible explanation for the worse performance.

If this measure of variation indicates convergence long before convergence has actually taken place then it may be because of the effect that targeting has on the number of control points in solutions. Figure 5.18 shows how the median number of control points in the fittest solution in the population changes over time.

It can be seen that the number of control points drops dramatically from 90 to only 10 in the first 50 generations of the test. This is also the time period over which genetic convergence was recorded. These observations indicate a possible link. When the number of control points in solutions becomes small then the vast majority of genes will have the value 0. Any two solutions will therefore have a lot of very similar genes simply because they both have a lot of zero genes. Two solutions with a very small number of control points will therefore be recorded as similar even if the actual set of control points that they use is completely different.

Because of this, in problems where the number of control points drops quickly, transitioning will tend to be too fast. It should be possible to slow down convergence by multiplying every recorded difference score by a tunable parameter α, where α has a value between 0 and 1. This will mean that $\bar{v} \times \alpha$ will replace $\bar{v}$ in equation 5.6. Like transitioning through time, this is a problematic approach as it relies on knowing in advance whether the problem at hand is one that will create solutions with very few control points quickly and to what extent. We

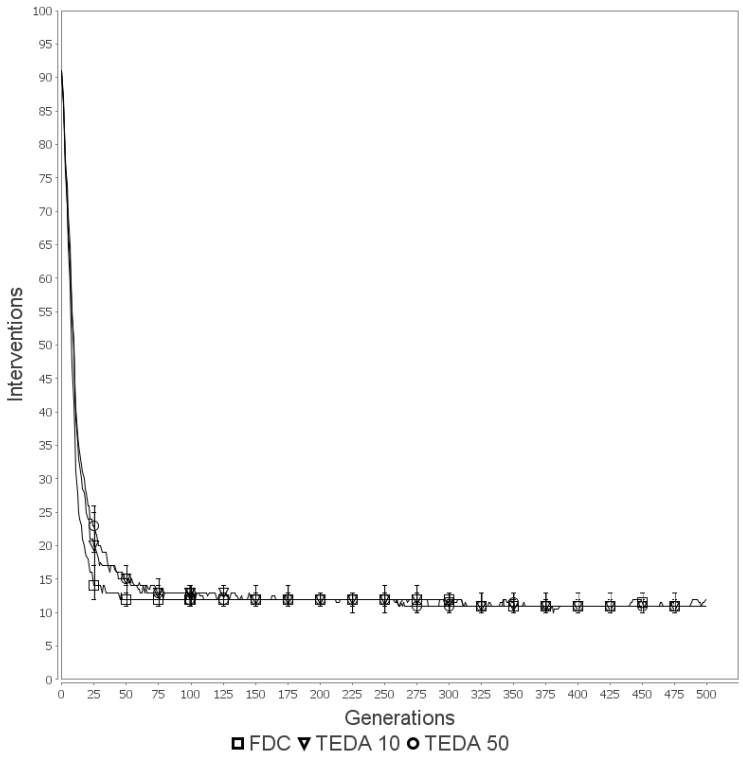

Figure 5.18: Transitioning Through Genetic Variation - Control Points (Interventions)

will, however demonstrate the performance of TEDA with this strategy and various different values of α to show whether or not this approach may be useful with any value of α.

The results in Figure 5.19 show the performance of TEDA with an α of 0.5, 0.25 and 0.75. The results from the previous test, before α was introduced, are placed in this graph for comparison and marked as having an α of 1. All tests were run by measuring convergence in the top 10% of the population. These results demonstrate that even by changing the α this method of transitioning still does not perform noticeably better than FDC.

5.4.5.1 *Control Point Adjusted Variation*

One solution to the problem of solutions with few control points appearing to be more similar than they actually are is to adjust the difference measure according to the number of control points in solutions. One method is to normalise the number of shared control points by the total number of control points in both solutions. Equation 5.7 shows how we may obtain the difference score ν between two individuals x_1 and x_2 through this method. I_{x_1} and I_{x_2} is the set of control points that are in x_1 and x_2 respectively. ν is calculated for all pairs and its mean, $\bar{\nu}$, is used to control convergence in the same way as in the previous method.

$$\nu = 1 - \frac{|I_{x_1} \cap I_{x_2}|}{|I_{x_1} \cup I_{x_2}|} \tag{5.7}$$

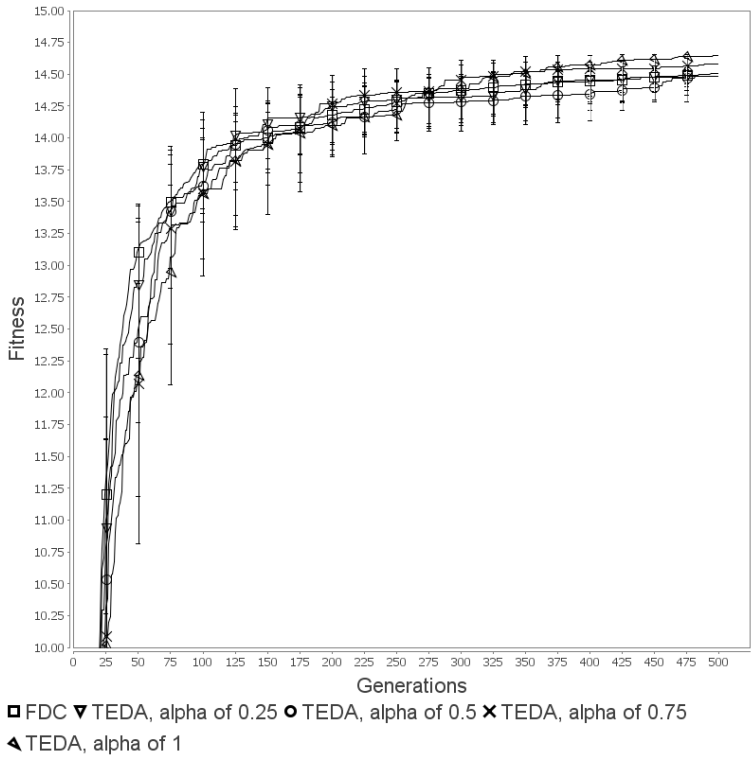

Figure 5.19: Transitioning Through Variation - Introducing α

The results in Figure 5.20 show the performance of TEDA using this kind of transitioning. Figure 5.21 shows how genetic variation changes over time when measured in this way. Both sets of results were produced from the same run of TEDA using this transitioning technique with the other parameters matching those used in previous tests.

As can be seen in Figure 5.21 no real convergence is seen using this technique. Figure 5.20 shows that, using this technique, TEDA's performance is similar to FDC's performance. This is to be expected as, without convergence, transitioning will not take place and TEDA will remain close to FDC in its behaviour. The lack of convergence detected may be due to the fact that, later on, the test solutions have very few control points. As can be seen from Figure 5.18 solutions may contain less than 10 control points. Only a few genes being different is sufficient to create a low variation score in proportion to the number of control points in the solution. To add to this, the mutation rate is relatively disruptive for solutions with a number of control points this low. This is because the mutation probability is 0.05 and it is attempted a number of times equal to the total length of the solution, 100 genes in this case. Therefore, simple background noise may be enough to disrupt the convergence measure.

BINARY TRANSITIONING In the previous tests, variation was observed to remain high throughout the test and so we now consider a change to the method described above that should decrease the amount of variation recorded and ensure that it better reflects the fact that we are primarily interested in where control points take place instead of what their value

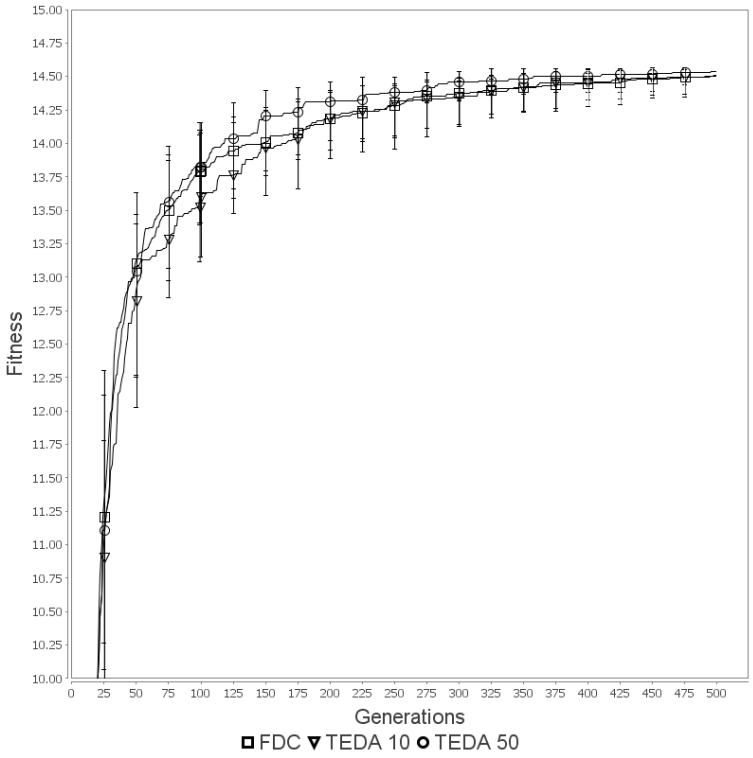

Figure 5.20: Control Point Ajusted Variation - Performance

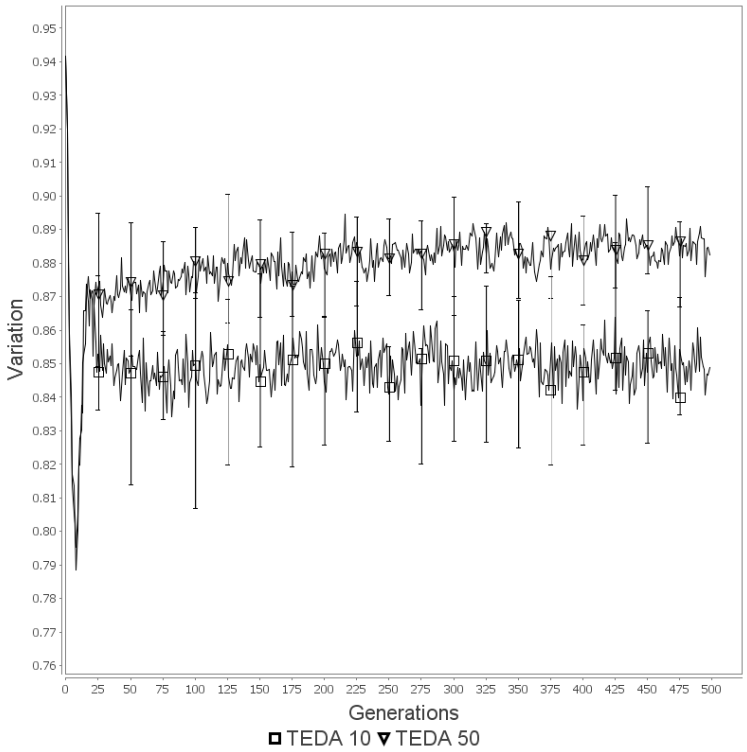

Figure 5.21: Control Point Ajusted Variation - Variation Measure

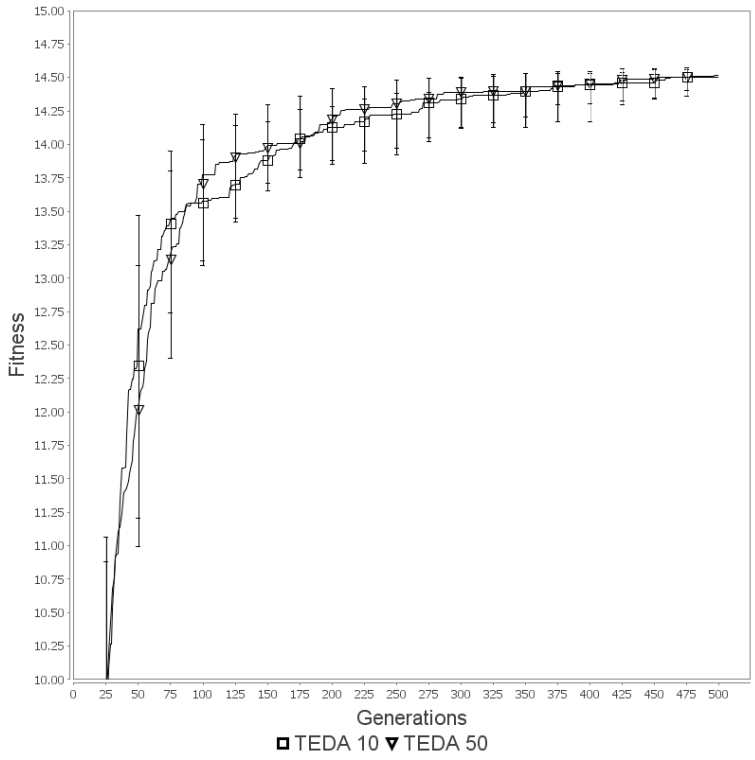

Figure 5.22: Binary Control Point Adjusted Variation - Performance

is. In the new method chromosomes are treated as binary encoded chromosomes. This is appropriate as TEDA essentially views solutions as binary entities. Specific integer values are chosen in a relatively random fashion in TEDA and so one would not expect these values to decrease in diversity to a great extent during the course of a test. Through this method we measure variation in the same way as before, except that we treat every chromosome as a binary string. That is to say that each gene with a positive value has its value temporarily changed to "1" before the process of calculating the variation measure v begins, thereby removing all differences that exist between chromosomes beyond a simple binary distinction of whether a gene is on or not. Figure 5.22 shows how TEDA performs using this technique and Figure 5.23 shows the variation scores obtained from this technique.

This change does not appear to improve performance in any way. Surprisingly, variation seems to now change in the opposite direction, starting out low and later becoming high. The explanation can be seen when we look at the change in number of control points in Figure 5.18. In the first few generations the best solutions contain almost all of the control points, averaging over 90 out of 100 control points. If all solutions being compared contain most control points then, when converted to binary, they will nearly all be almost identical, consisting primarily of 1s.

ESTABLISHING WHETHER VARIATION EVER DECREASES It is possible that the problem lies in the fact that, if a method does not lead to transitioning early on, then the nature of the

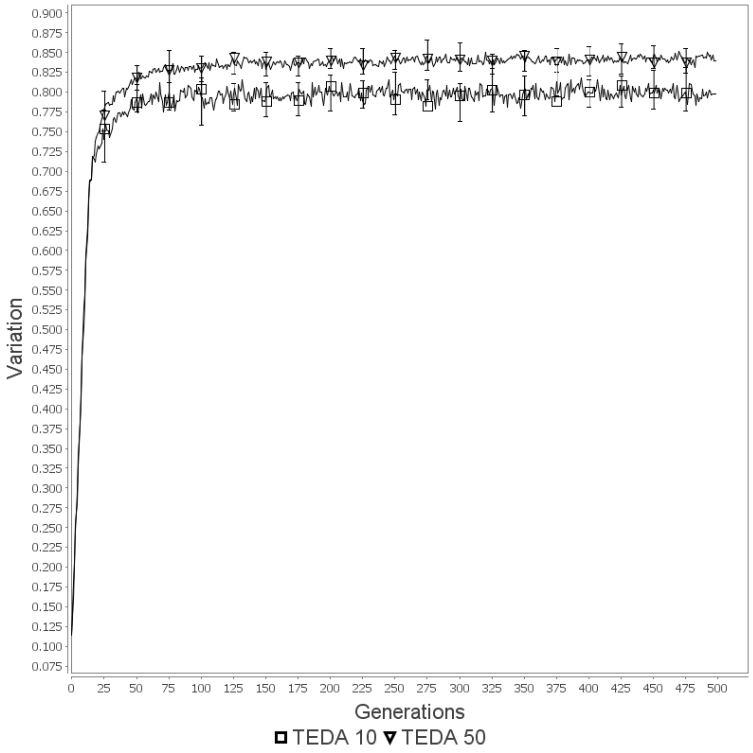

Figure 5.23: Binary Control Point Adjusted Variation - Variation Measure

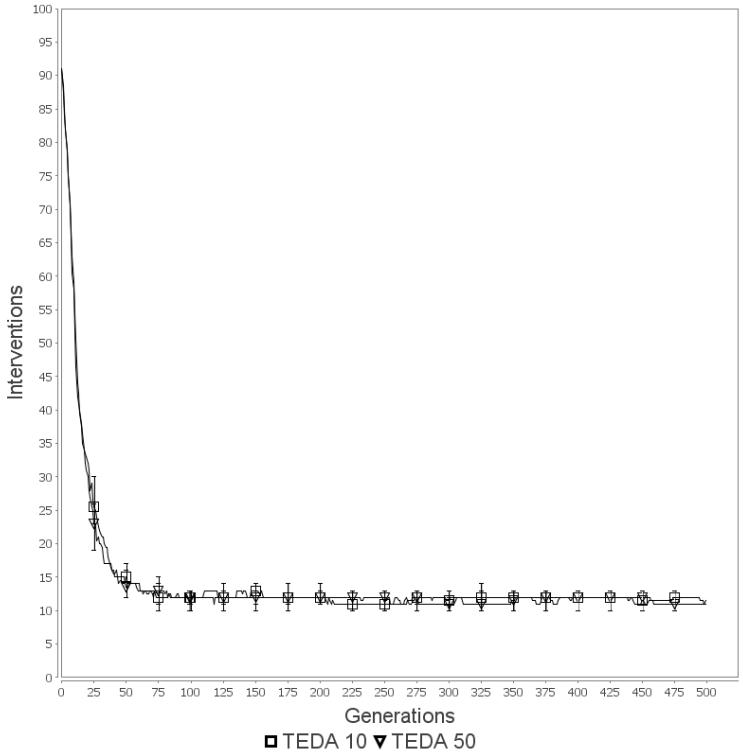

Figure 5.24: Binary Control Point Adjusted Variation - Control Points (Interventions)

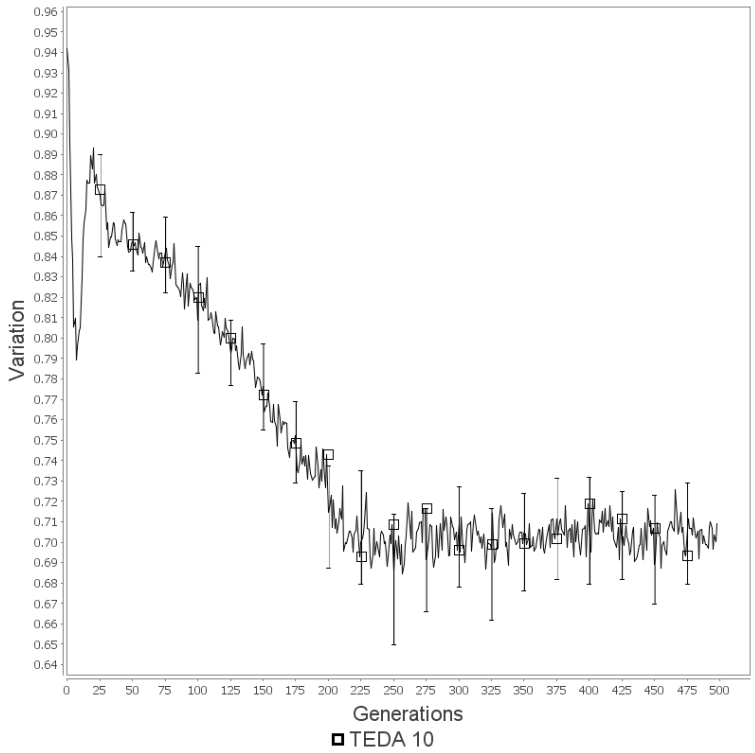

Figure 5.25: Transitioning over Time- Variation Measure

population will not change in such a way that variation, as measured by these techniques, decreases at all. To test this hypothesis, we first of all need to establish whether a decrease in variation, as measured through the non binary but control point adjusting transitioning technique, is indeed recorded in situations where we know that TEDA transitions. If this is the case then it may mean that some method of ensuring that transitioning takes place in some form or other will produce an effective transitioning technique when combined with a method of responding to variation within the population to ensure that transitioning is dynamic and does not assume a fixed rate of convergence.

The results shown in Figure 5.25 show the variation, as measured through the control point adjusted non binary technique, change over time in a run of TEDA in which the actual transitioning takes place at a fixed rate over time, as in Section 5.4.3. In the results shown variation is obtained from the top 10% of the population. In this test, as with the tests at the start of this chapter, transitioning takes exactly 200 generations and so this should tell us whether a change in variation occurs while we know that transitioning is occuring.

As we can see from the results in Figure 5.25 variation does decrease over time in this situation.

5.4.5.2 *Combining Time Based Transitioning with Dynamic Transitioning*

In the last section we made two key observations. Firstly, it was noted that transitioning at a fixed rate through time is effective but relies on knowledge of the rate at which convergence is

likely to occur and so cannot be generalised to any problem. Secondly, although transitioning dynamically by measuring variation would seem to offer a more adaptable approach, when it was used on its own the variation scores obtained did not appear to change to a great enough extent to effectively control the transitioning process. It was, however, observed that when transitioning does take place, for example when it is forced to take place as a fixed rate transitioning scheme is used, the same variation scores do show a significant change. It is therefore possible that variation scores could be used to control transitioning provided that some other technique catalyses or starts the transitioning process. For this reason, we propose combining the two approaches of transitioning at a fixed rate over time and transitioning through a measure of variation. The idea is that, while this approach is not ideal as it still relies on deciding a rate at which convergence should take place, using time should ensure that transitioning does take place while measuring variation will ensure that transitioning does dynamically adjust itself based on the nature of the problem.

In the proposed approach, transitioning takes place through the non binary control point adjusted approach as described above except that each convergence score v is multipled by a real number δ. δ starts out at 0 and increments by a fixed rate α every generation. This approach is described in equation 5.8 in which g is the current generation and δ_g is the current value of δ.

Figure 5.26 shows the results of using this approach and how it performs with an α of 0.01 and 0.1 when compared to FDC as well as to the non binary control point adjusted method without using an α (marked as *TEDA 1*).

$$\begin{aligned} \delta_g &= g\alpha \\ v &= 1 - (\frac{|I_{x_1} \cap I_{x_2}|}{|I_{x_1} \cup I_{x_2}|}\delta_g) \\ s &= s_{min} + \bar{v}(s_{max} - s_{min}) \\ b &= b_{max} - \bar{v}(b_{max} - b_{min}) \end{aligned} \tag{5.8}$$

This technique performs similarly to FDC throughout the test. There is a possibility of a slight improvement towards the end of the test if an α of 0.01 or 0.1 is used, as can be seen if we focus on fitnesses between 14-15, as shown in Figure 5.27.

5.4.5.3 *Probabilistic Transitioning*

The previous section suggests that a transitioning process based on measuring genetic variation within the population has a high chance of not effectively transitioning unless combined with transitioning at a fixed rate over time, thereby introducing an additional parameter that needs tuning. Finding a measure of transitioning that does not suffer from a similar drawback appears to be a difficult task and so it would seem sensible to rethink the transitioning process

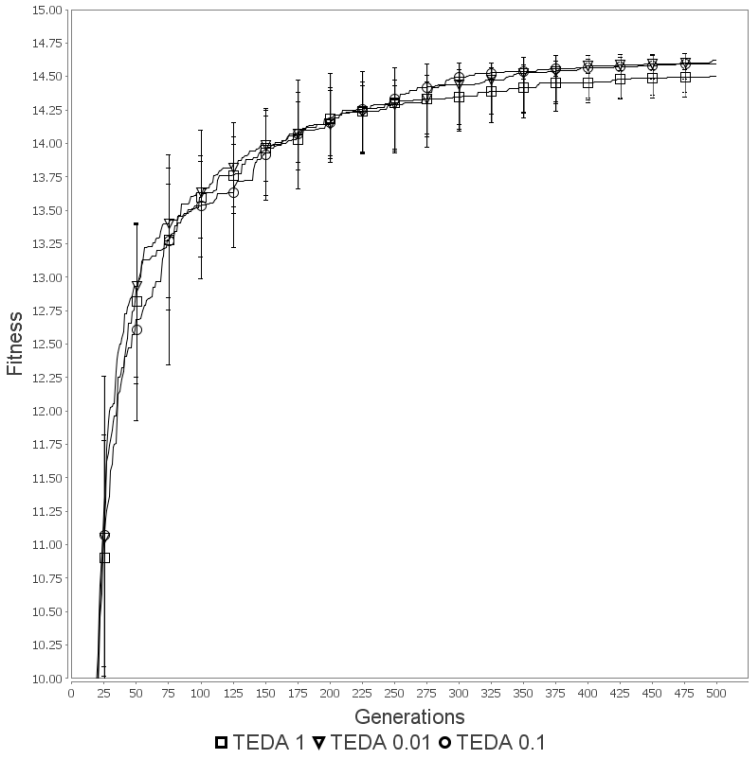

Figure 5.26: Time and Variation Based Transitioning - Performance

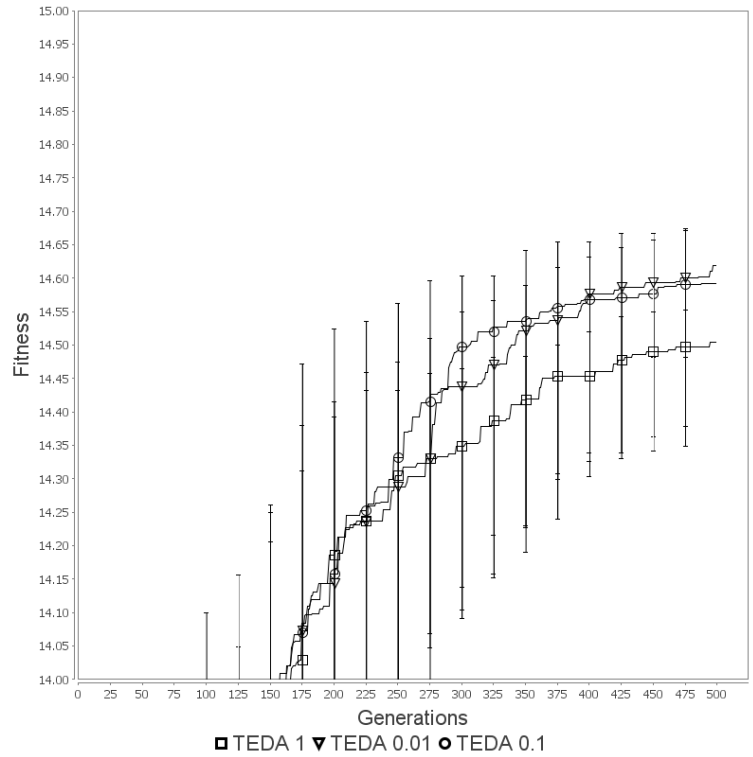

Figure 5.27: Time and Variation Based Transitioning - Performance (Detail)

from first principles. One option is to control transitioning through repeated small steps, each chosen stochastically, as this might create a smoother transitioning process that works without having to find an overall measure of variation that can be relied upon to change at a smooth rate.

In this section we explore whether it is possible to transition not by measuring variation across the top $n\%$ of individuals in the population each generation but instead by occasionally adding individuals to the breeding pool with some probability based on how diverse the breeding pool currently is. In the process adopted, which shall be referred to as *probabilistic transitioning*, each breeding pool B begins with the selection of two parents. The decision on whether or not to add another parent to B is made with a probability p. p is $1 - v$ where v is the difference score v as calculated in equation 5.7 where x_1 and x_2 are the last two parents that were added to B. Initially they will be the first two parents in the pool. The probability of adding a new parent is therefore higher when these two parents are similar.

If a parent is added according to this rule, the process is repeated until the first occurrence of a parent not being added or b reaches b_{max}. When a new parent is added, s is decreased and so the selection pressure increases. The selection pressure therefore increases as more instances where x_1 and x_2 are highly similar are encountered, expected to happen as the level of variation within the population decreases. s may be decreased until it reaches s_{min}.

This technique has a number of advantages over simply measuring convergence each generation. Firstly, by only adding one parent to the breeding pool at a time it helps to ensure that transitioning is a gradual process. A sudden jump from FDC to an EDA is not possible. Also, the process of adding a parent to the breeding pool when diversity is low ensures that the breeding pool remains diverse. The incremental aspect also ensures that transitioning is a smooth process that starts with pure FDC with a b of 2 and an s of 100% of the population and transitions towards an EDA. With the other methods that measure fitness across the population there is no guarantee that a low measure of variation will not lead to the first or early generations operating like an EDA and causing a decrease instead of an increase in b and a decrease instead of an increase in s. Although just using two solutions at a time to judge the convergence of the population would seem to be unreliable, the overall effect should be relatively accurate. This is because each two solutions can only cause a change in b or s of 1 and the overall trend is made reliable by the large number of these small decisions.

Figure 5.28 shows the results of testing the performance of TEDA using this approach when compared to FDC. TEDA using this approach performs worse than FDC in the early stage of the test before later matching FDC's performance.

Figure 5.29 shows how variation changes over time when this method is used. Variation was measured from the top 10% and 50% of the population using the same method as used in the previous section. Note that the loss in variation is rapid. It appears from this figure as though the transitioning technique leads to an overly rapid loss of population diversity,

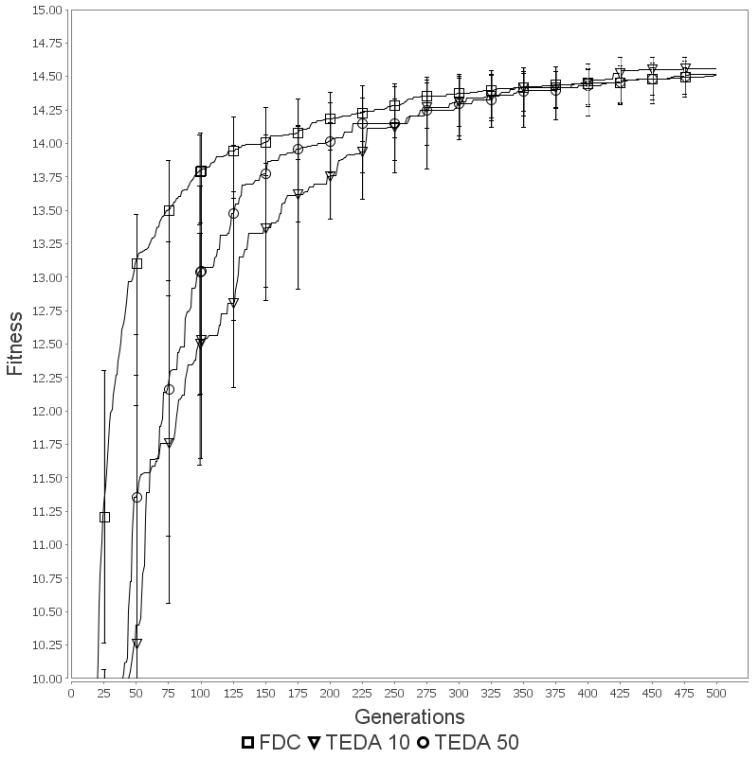

Figure 5.28: Probabilistic Transitioning - Performance

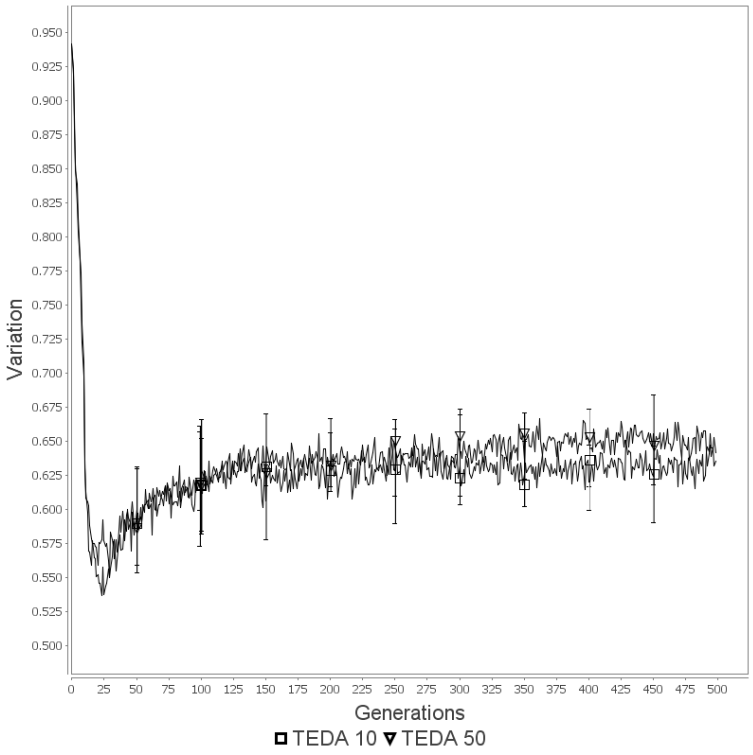

Figure 5.29: Probabilistic Transitioning - Variation Measure

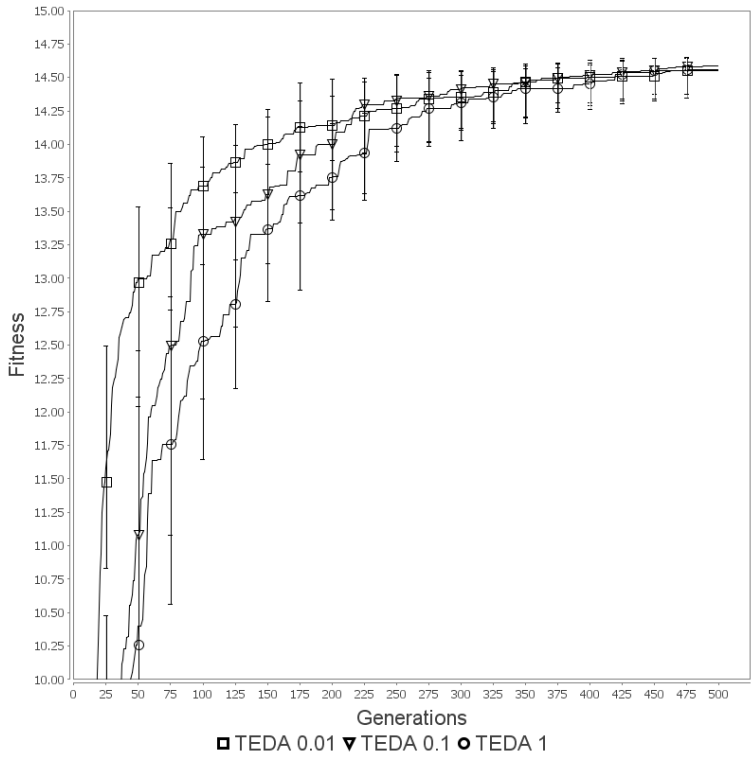

Figure 5.30: Probabilistic Transitioning with δ - Performance

although the later improvement in performance combined with the increase in variation after the initial drop suggests that it also manages to escape from any local optima that it may have become trapped in.

One option to slow down the rate of transitioning is to combine it with fixed transitioning over time as we did with variation based transitioning. The approach is the same, p is multipled by a δ value that increments, starting at 0, by a small pre set α value each generation. The disadvantage of this technique, as with the equivalent technique applied to non probabilistic transitioning, is that it requires an additional tunable parameter.

Figure 5.30 shows the result of applying this technique with α values of 0.01 and 0.1. The plot *TEDA 1* shows, for comparison, how TEDA performed without using an α value. This modification does not appear to improve the situation. Both versions of TEDA still perform worse than FDC.

5.4.6 *Discussion of Transitioning*

Before transitioning was introduced FDC performed better than TEDA for the first 100 generations of the test (see Figure 5.10 in Section 5.3.3.3). Before introducing transitioning we established that transitioning potentially may result in an algorithm that performs better than FDC by showing that this is the result if transitioning happens at a fixed rate over time. This

was judged to be unsatisfactory as it relies on knowing at what rate a population is likely to converge. We therefore explored several different methods of measuring when the population has converged. Through this exploration we were unable to develop a version of TEDA that has been proven to perform better than FDC at every stage of the test and so the best algorithms to arise from this exploration are those that:

- Do not appear to perform worse than FDC at any stage of the test, unlike the pre-transitioning version of TEDA.
- May possibly perform better than FDC later on in the test.
- Definitely carry out transitioning. This is in contrast to some techniques that were shown to not measure any noticeable convergence, and so therefore remain in the FDC phase of the TEDA process.

It transpired that, due to the fact that a reduction in population diversity only occurs when transitioning is encouraged to at least begin, the only technique that matches this description used a combination of transitioning through time and transitioning by measuring population diversity. This is not ideal as it depends on a tunable parameter but it appears as though, for now, this is necessary. For this reason, the technique that we shall be using for transitioning from now on can be described as follows: Transitioning by measuring genetic diversity, the control point adjusted and non binary method, combined with a fixed transitioning rate α of 0.01. Our next task will be to establish whether this technique does in fact offer any statistically significant improvement over FDC at any stage of the test and whether it offers any improvement over any other techniques.

After considering how to measure convergence we still need to answer the question concerning which sample of solutions convergence should be measured from. So far we have been using the top 10% and the top 50% of the population. There seems to be little difference between how either of these samples perform. For efficiency reasons we will consider using the top 10% to be the better option. The other alternative that we shall now explore is to measure convergence from the selection pool that we are already using for selecting the breeding pool, S.

Figure 5.31 shows the results of how TEDA transitioning by measuring how the variation within S performs when compared to TEDA transitioning by measuring the variation within the top 10% of the population.

These results do not show a significant difference for either approach. For this reason we shall continue to use the top 10%.

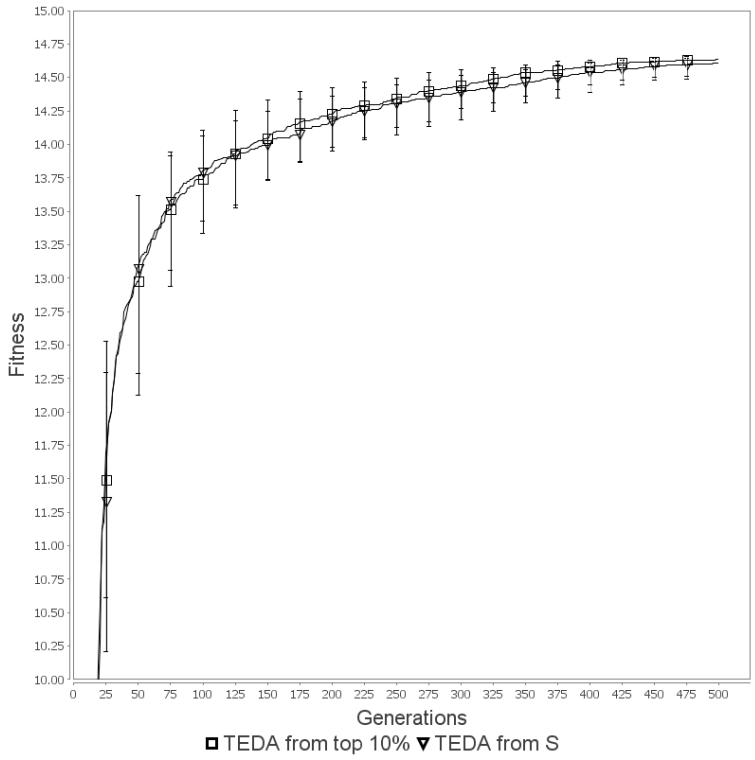

Figure 5.31: Transitioning from S

5.4.7 *Incremental Learning*

In the previous section a version of TEDA was developed, the main features of which are as follows:

- TEDA transitions from FDC to an EDA through a method in which the genetic variation of the top 10% of individuals in the population is measured and this is used in combination with a transitioning process that progresses at a fixed rate over time.
- To facilitate this transitioning process, the two pool selection method is used to select a breeding pool in which tournament selection is applied to a subset of the population selected through truncation selection.
- A probability model based on UMDA is obtained from this breeding pool. This uses the marginal probability of each control point based on the total fitness of all solutions in which it is found.
- A target number of control points is obtained through the FDC rule using the fittest and least fit individual in the breeding pool.
- Control points are selected by first of all using all control points set in all parents in the breeding pool and then randomly selecting control points found in only a subset of parents.

This section explores algorithms that deviate from this standard version of TEDA by considering alternative methods for generating the probability model to that used in UMDA. Specifically, we explore the concept of maintaining and improving a probability model from generation to generation. UMDA each generation calculates a new vector of marginal probabilities from the current population whereas the EDAs that are explored in this section maintain one probability model throughout the run and use a pre specified learning rate to slightly improve this model each generation.

The two algorithms that we explore are Population Based Incremental Learning (PBIL) (see Section 2.8.1.2) and the Compact Genetic Algorithm (CGA) (see Section 2.8.2). For each generation in PBIL, the vector from which new solutions are sampled from P is moved in the direction of the fittest individual in the population and away from the least fit individual. CGA (see Section 2.8.2) is not a conventional EDA at all as it does not use a population. Instead, it repeatedly generates two individuals, uses these to update the probability vector and then generates the next two individuals from the updated vector. Equation 5.9 provides the learning rule that is used by PBIL. In this rule, the sampling vector P is moved in the direction of a good solution S by a learning rate Δ. In CGA the probability vector P is improved upon in a similar manner, as detailed in Equation 5.10. Whereas in PBIL P moves in the direction of one fit solution, in CGA it moves simultaneously away from the less fit solution out of the two that have just been generated and towards the fitter of the two. P moves towards the fitter of the two solutions S_2 for genes where it disagrees with the less fit solution S_1.

$$P'_x = ((1-\Delta)P_x) + (\Delta S_x) \tag{5.9}$$

$$\begin{aligned} &\text{if } S_{1_x} \neq S_{2_x} \\ &P'_x = P_x + \Delta S_{1_x} \end{aligned} \tag{5.10}$$

Using these algorithms we have created four new variants of TEDA. As TEDA is a population based approach we do not intend to replicate CGA's population free behaviour. We are instead interested in the fact that CGA, unlike PBIL, does not use the best, and in some variants the worst, individual in an entire population to train P but instead simply uses two individuals that have just been generated. This is equivalent to selecting with a low selection pressure [2] as there is no population from which the best individuals may be chosen. Every individual generated is used to breed and there is no competition. Also, in CGA P is updated at a faster rate as it is updated for every two individuals instead of for every n individuals where n is the size of the population, as in PBIL.

In all of the approaches explored here, first of all P is initialised so that each element within it is equal to 0.5. Following this, each generation a new population is produced using one of the following processes:

PBIL BASED TEDA 1

- A probability vector, ρ, is produced from a breeding pool using the same method as in standard TEDA.
- The sampling vector P is moved in the direction of ρ using equation 5.9.
- P is used to generate new individuals in the same manner as in standard TEDA.
- As in standard TEDA, a new breeding pool is selected, a new vector ρ is produced and the process repeats.

PBIL BASED TEDA 2

- P is moved in the direction of S_1 using equation 5.9 where S_1 is the fittest individual in the population.
- P is used to generate a new population as in standard TEDA. Although a breeding pool is not used to generate P, breeding pools are selected through the standard TEDA method and used to decide which value a control point that has been selected should be set to, as with standard TEDA.

PBIL BASED TEDA 3

- The sampling vector P is moved in the direction of S_1 and away from S_2 using equation 5.10 where S_1 is the fittest individual in the population and S_2 is the least fit individual in the population.
- P is used to generate a new population as in standard TEDA. Once again, breeding pools are used to choose a value for each control point.

CGA BASED TEDA

- A breeding pool is selected using the standard TEDA selection process.
- P is moved in the direction of the fittest parent in the breeding pool and away from the least fit parent using Equation 5.10.
- New solutions are generated using P through the standard TEDA technique. When a control point has been selected, the value to set control points to is decided through the standard TEDA process.
- This process is repeated until a new population has been generated.

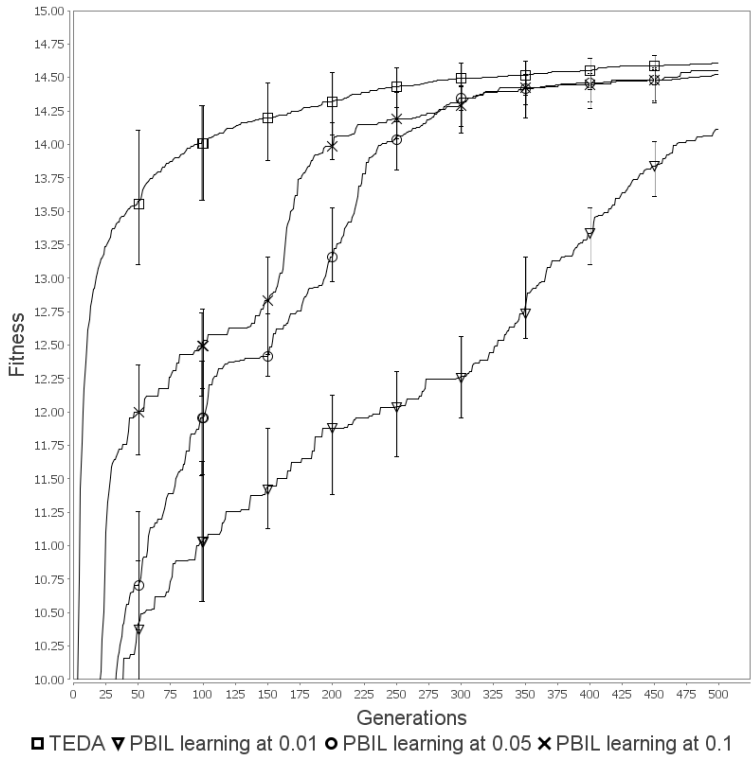

Figure 5.32: PBIL based TEDA 1

The results of applying these techniques to the chemotherapy problem are provided below. In each test the learning rate provided is equal to δ in the equations given in Section 2.8.1.2. It is the rate at which the probability vector changes and at which learning takes place.

The results in Graphs 5.32, 5.33 and 5.34 show how the performance of TEDA based on PBIL (denoted “PBIL") compares to that of the standard UMDA based TEDA (denoted “TEDA"). The graphs demonstrate versions 1, 2 and 3 of PBIL based TEDA, as described above. The results in Figure 5.35 compares TEDA based on CGA to UMDA based TEDA.

Using a PBIL based method appears to be detrimental to the performance of TEDA. In all three situations, it appears to be the case that the slower the learning rate the worse the performance is and that the TEDA variation which does not use a learning rate at all, standard TEDA, still performs best. The second variation, learning from the best individual in the population, appears to perform slightly better than the other two. CGA based TEDA performs very poorly as well. It has been previously observed that CGA performs poorly on difficult multimodal deceptive problems due to its low selection pressure [2].

In all cases TEDA performs worse when the probability model is learnt incrementally. The incentive for using incremental methods is that, in standard UMDA, a probability model that is produced from an early population may over exploit random patterns in the population and may lead to premature convergence. For this reason, incremental methods ensure that, early on, the probability model is such that new solutions are generated almost at random, with no danger of over fitting or of a loss of diversity. In TEDA, by contrast, the population in

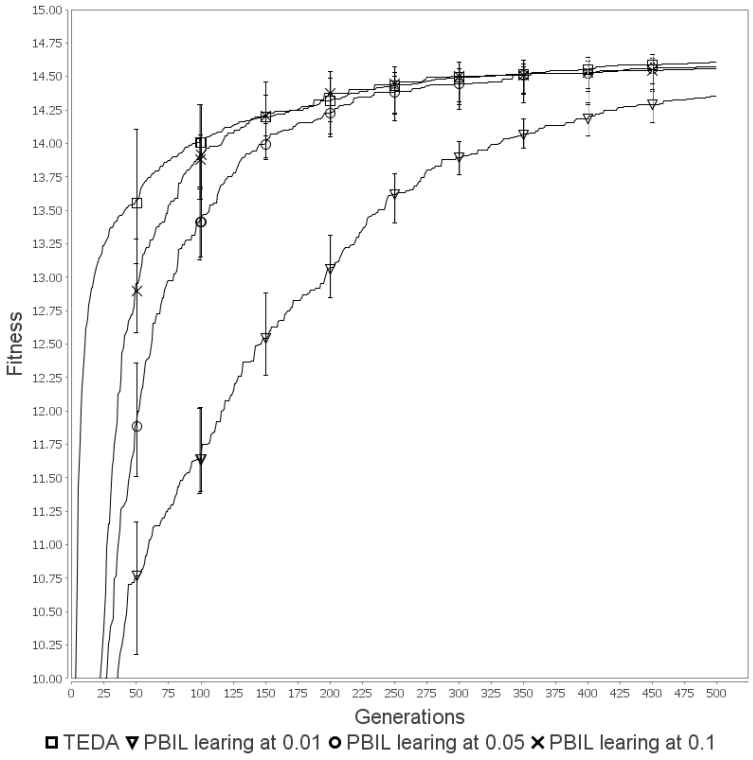

Figure 5.33: PBIL based TEDA 2

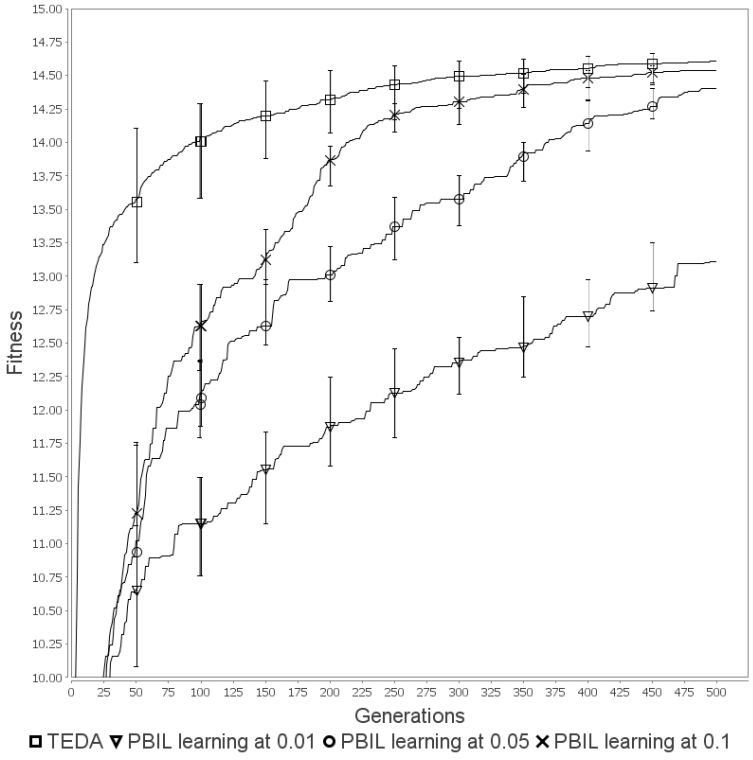

Figure 5.34: PBIL based TEDA 3

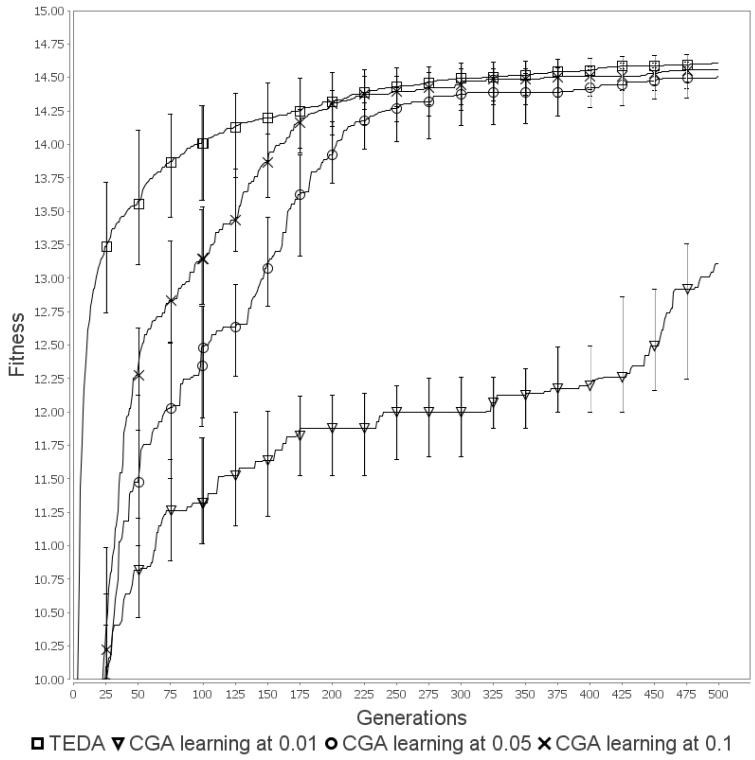

Figure 5.35: CGA based TEDA

early generations is generated through a GA based method. With a sufficiently low selection pressure most individuals will be created from different parent solutions. This means that they'll be less chance of a loss of diversity and so the situation that incremental learning methods were developed to solve does not occur. Given these results, UMDA will continue to be used as the basis for TEDA.

5.5 COMPARATIVE TESTING

In this section TEDA, as developed in the earlier part of this chapter, is compared against FDC and state of the art EDAs and GAs. These tests are run 200 times and run for 2000 generations. For each set of results, Kruskal Wallis (KW) non parametric analysis of difference is computed to determine whether or not TEDA offers a statistically significant advantage over any other techniques.

5.5.1 *TEDA Compared to FDC*

The results in Figure 5.36 compares TEDA to FDC. Figure 5.37 shows the same results but focusses on fitness scores between 14 and 15.

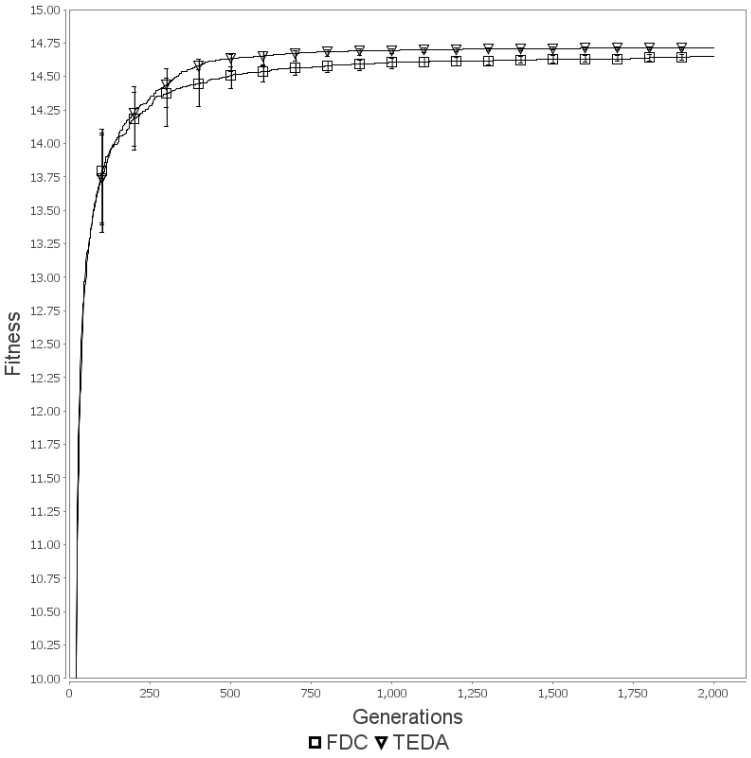

Figure 5.36: TEDA vs FDC - Performance

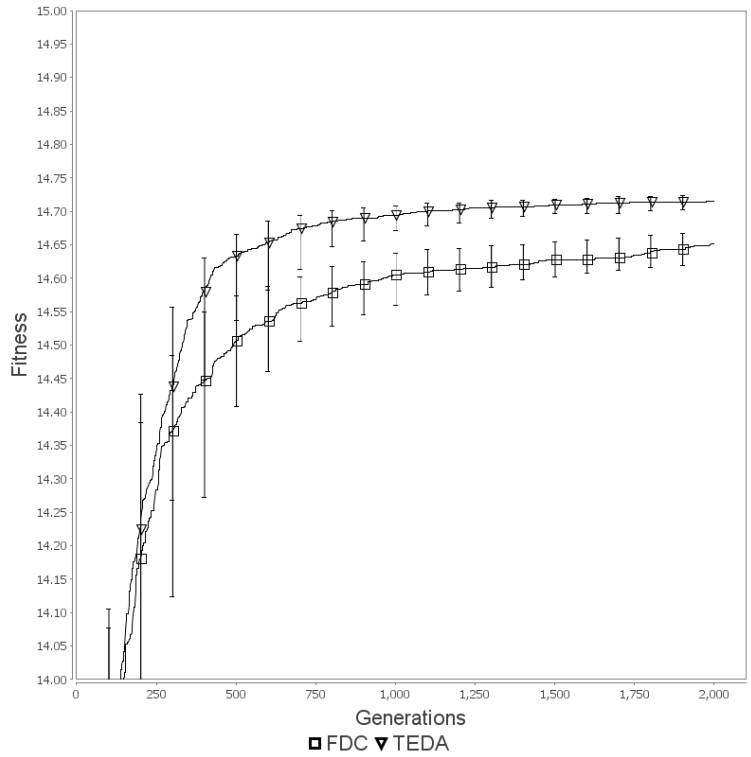

Figure 5.37: TEDA vs FDC - Performance (Detail)

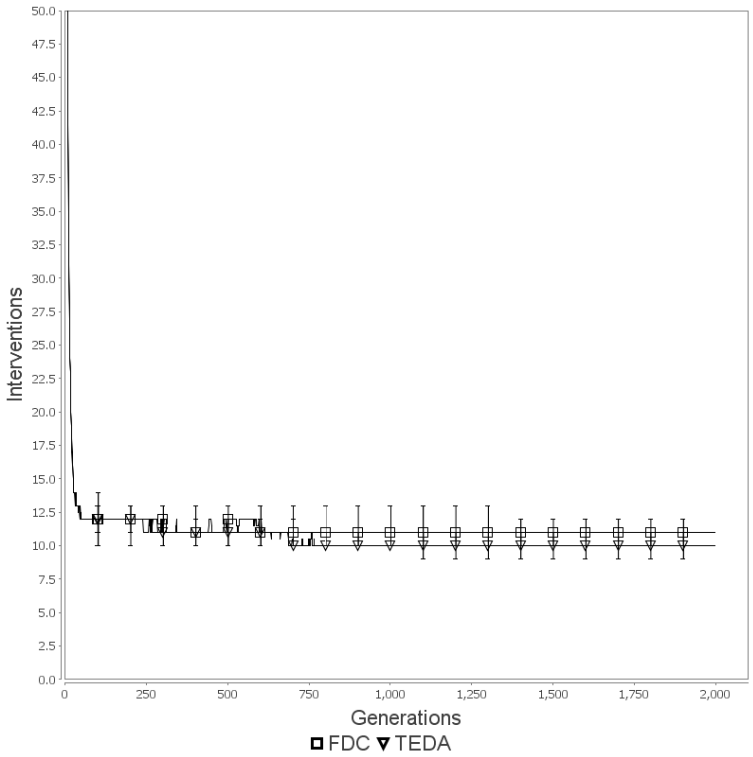

Figure 5.38: TEDA vs FDC - Control Points (Interventions)

These results show that TEDA appears to outperform FDC at every generation after around generation 250. When we look at the median number of control points in solutions that these approaches discover in Figure 5.38 the explanation for this can be seen. Their difference in fitness scores may be explained by TEDA finding solutions with slightly fewer control points than FDC. TEDA appears to be targeting slightly better than FDC. As the targeting technique is the same this must be due to which solutions TEDA is using for targeting. These individuals will be fitter in proportion to the rest of the population than in FDC later in the process as they are selected with a higher selection pressure. We must acknowledge, however, that the difference that this offers is relatively small.

5.5.2 *TEDA Compared to EDAs and GAs*

In this section TEDA is compared against both a standard EDA using UMDA and a standard GA using UC. Two versions of UMDA are used for comparison. *UMDA1* was run with the most common parameters as found in literature whereas *UMDA2* was run with exactly the same parameters as TEDA. As such *UMDA1* provides a state of the art benchmark whereas *UMDA2* can be compared directly with TEDA to show us what effect TEDA's targeting and transitioning produces. The three differences between the two implementations of UMDA are as follows:

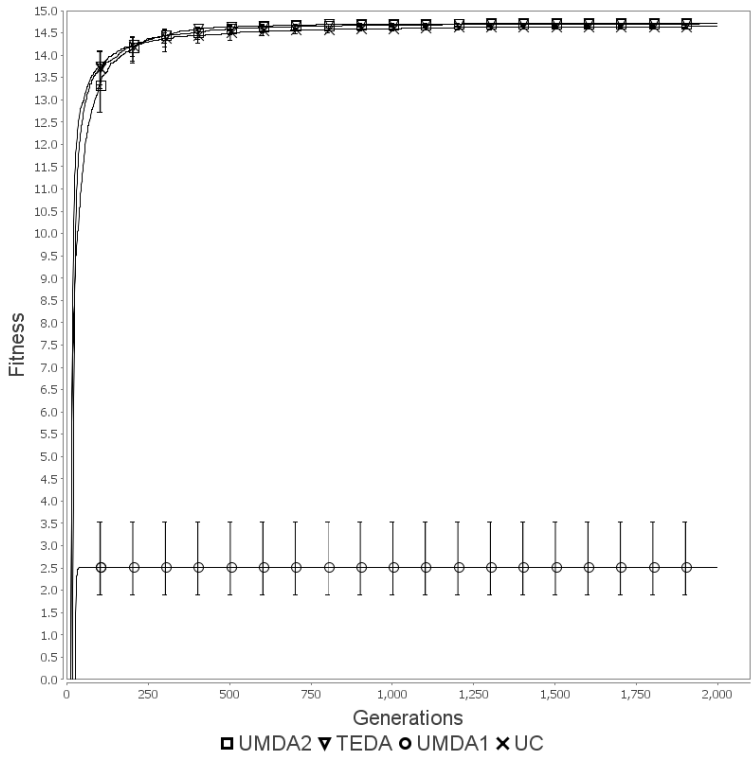

Figure 5.39: TEDA vs EAs - Performance

- Mutation: By default, EDAs do not use mutation [83] and so *UMDA1* was run with a mutation rate of 0 whereas *UMDA2* was run with the same mutation rate as all other algorithms (0.05).
- The Probability Model: *UMDA1* uses equation 5.2 in Section 2.8 to build the probability model whereas *UMDA2* uses equation 5.3. As justified in Section 5.3, TEDA creates a fitness based model whereas traditional UMDA uses a model solely based on the frequency of genes.
- The Breeding Pool Size: *UMDA1* samples from a breeding pool containing the best 50% of the population whereas *UMDA2* samples from the best 10%. This is because, with b_{max} set to 10, TEDA's breeding pool will transition to a maximum of 10% yet 50% of the population is a commonly used pool size for EDAs [70].

Figure 5.39 shows the result of this comparison, showing fitness scores in the range 0 to 15. Figure 5.40 shows the same results but zoomed in to only show fitness scores in the range 14 to 15.

As can be seen from Figure 5.39 all approaches perform well on this problem except for a standard EDA. Looking in more detail, we can see from Figure 5.40 that TEDA appears to show a slight improvement over UMDA2.

Table 5.4 shows the Kruskal Wallis difference scores for this test. TEDA offers a statistically significant improvement over UMDA2 at generations 1200,1400 and 1800 and is close in the

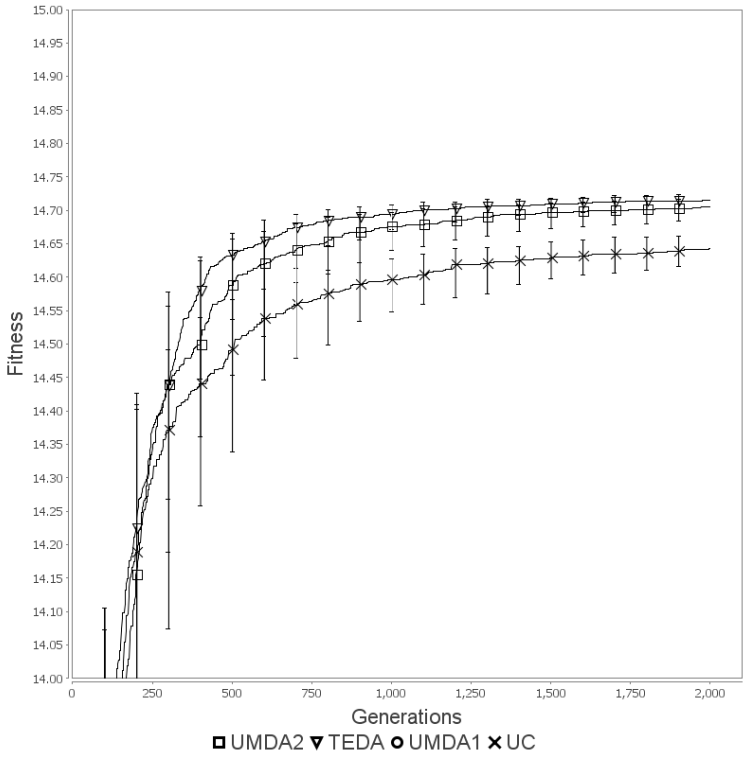

Figure 5.40: TEDA vs EAs - Performance (Detail)

Generations	TEDA vs UMDA2	TEDA vs FDC	TEDA vs UMDA1	TEDA vs UC
200	39.75	31.72	***524.34***	32.06
400	75.18	***176.29***	***608.66***	***189.21***
600	79.72	***240.9***	***643.54***	***259.47***
800	80.29	***286.8***	***667.47***	***308.61***
1000	76.86	***322.66***	***685.62***	***348.66***
1200	***87.22***	***362.19***	***703.43***	***369.96***
1400	***81.77***	***377.29***	***709.56***	***384.74***
1600	78.52	***380.27***	***709.97***	***386.64***
1800	***85.14***	***384.55***	***714.77***	***394.98***
2000	79.15	***378.96***	***713.32***	***400.76***
Threshold	80.67			

Table 5.4: Kruskal Wallis - Chemotherapy

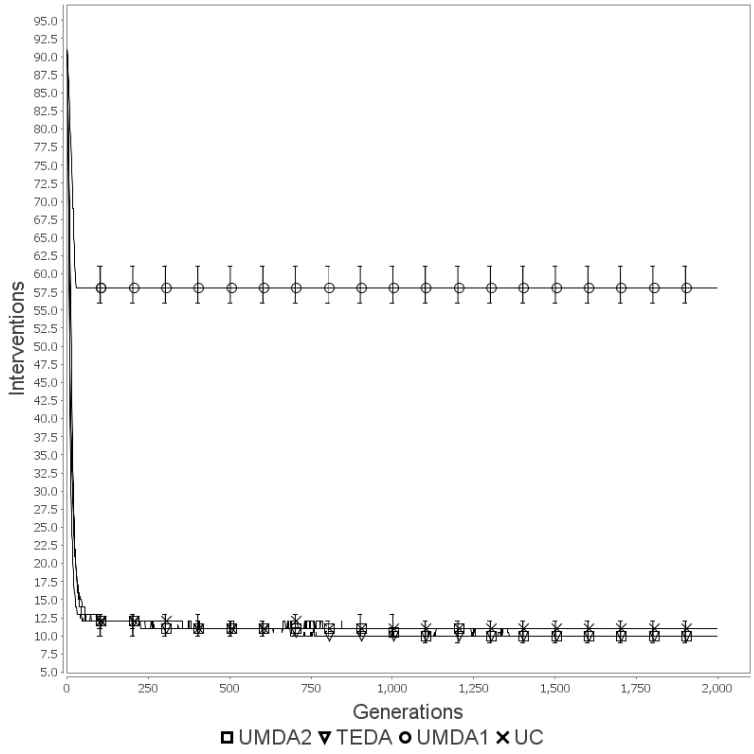

Figure 5.41: TEDA vs EAs - Control Points (Interventions)

other generations. From this we can say that the performance of TEDA is promising but that this is a problem that a standard EDA is effective at solving. The small error bars for both UMDA2 and TEDA tend to suggest that both algorithms consistently find the optimum or near optimum. In order to prove the effectiveness of TEDA, it should be applied to problems that a standard EDA is unable to quickly solve. It is interesting that a standard EDA (UMDA1) performs poorly at this problem and appears to prematurely converge. It should be noted, however, that as a bad fitness in this problem may be as low as -6×10^6 that its performance is still relatively good.

Figure 5.41 shows the median number of control points found in the solutions discovered by each algorithm.

As can be seen FDC, UC and TEDA all find solutions with a similar number of control points but a standard EDA (UMDA1) is unable to find solutions with a small number of control points. This may account for its ineffectiveness as solutions with a large number of control points are penalized.

It should be noted that, as previously mentioned, in the first few generations the fittest solutions may contain 90 control points even though the solutions that are ultimately the fittest at the end of the test may only contain 5 control points. Therefore, if an algorithm performs well it must be able to explicitly or implicitly reduce the number of control points in the solutions found. TEDA and FDC explicitly do this through targeting and it appears that UMDA2 and a GA using UC are able to implictly do this. UMDA1, on the other hand, is unable

to. This may be due to the large number of parents from which it breeds. When there is a population that contains a small number of solutions that both use a small number of control points and are fit, genetic algorithms andEDAs that use a smaller number of parents may occasionally breed from these to produce a similar small and fit solution. However, these solutions are likely to be obscured in a method that builds a probability distribution from 50 parents, most of which are likely to contain a large number of control points. Effectively, the useful information is averaged out by the high level of noise.

5.5.3 *Transitioning from True FDC*

In this test we establish whether TEDA truly does transition from behaving like an FDC early on to behaving like an EDA later on. TEDA was designed to do this in order to build on the already promising performance of FDC. The only difference between pre converged TEDA and FDC is that TEDA selects control points based on the fitness of the parents whereas FDC does not.

In this test we attempt to remove this difference and create a version of TEDA that exactly matches FDC early on. The reason for this concerns the tendency that some versions of TEDA have shown to prematurely converge. The conclusion that was drawn from this tendency was that TEDA had transitioned from FDC to an EDA too quickly. It is however, also possible that the slight difference between pre convergent TEDA and FDC also adds to the exploitative nature of TEDA and increases the probability of premature convergence.

For this test we set a rule that before TEDA has converged and when it is using only two parents, it should behave exactly like FDC. This means that if the number of parents is two the following process will be followed, exactly matching that used by FDC:

1. Choose at random control points found in both parents (S_{dup}) and set them.
2. Choose at random control points set in only one parent (S_{single}) and set them.
3. Stop when I_t control points have been set.

This is in contrast to the current process where, in place of step 2, control points are set based on their probabilities, giving control points found in the fitter of the two parents a higher probability of being set.

Figure 5.42 shows the performance of this approach compared to standard TEDA. These results suggest that creating a version of TEDA that exactly matches FDC early on has no effect on performance. It seems likely that TEDA does, in all important respects, match FDC exactly early on and that any tendency towards premature convergence is due to the rate at which it transitions to more EDA like behaviour.

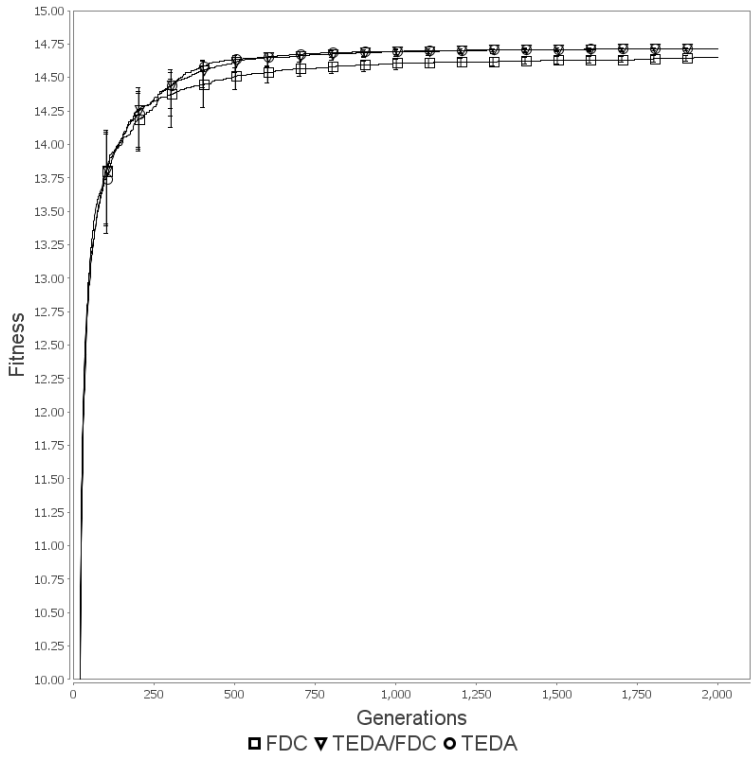

Figure 5.42: Transitioning from True FDC

5.5.4 *The Importance of Targeting*

It appears that adding EDA functionality to FDC may cause it to perform better than a standard EDA. In this section, we ask whether this improvement is due to the targeting or the transitioning. To test this, the results in Figure 5.43 compare the performance of TEDA to the performance of an algorithm that is exactly the same as TEDA but without control point targeting.

Both techniques show identical performance. These results demonstrate that it is the transitioning process that is responsible for the improvement in performance and not the targeting method. Both with and without targeting the performance is similar.

Figure 5.44 shows the number of control points found in the solutions discovered by these algorithms.

It appears as though without explicitly targeting, the TEDA transitioning process is still able to find solutions with a similar number of control points to those that it finds with explicit targeting. In this problem the necessary reduction in number of control points appears to happen automatically without needing to be explicitly controlled. This is further reenforced due to the fact that in this situation FDC performs to the same standard as UC, as demonstrated in Figure 5.45. The only difference between these two techniques is also targeting.

This is surprising as Godley [48] demonstrated that FDC is able to outperform a standard GA using UC on this problem. The only difference between these results and Godley's are

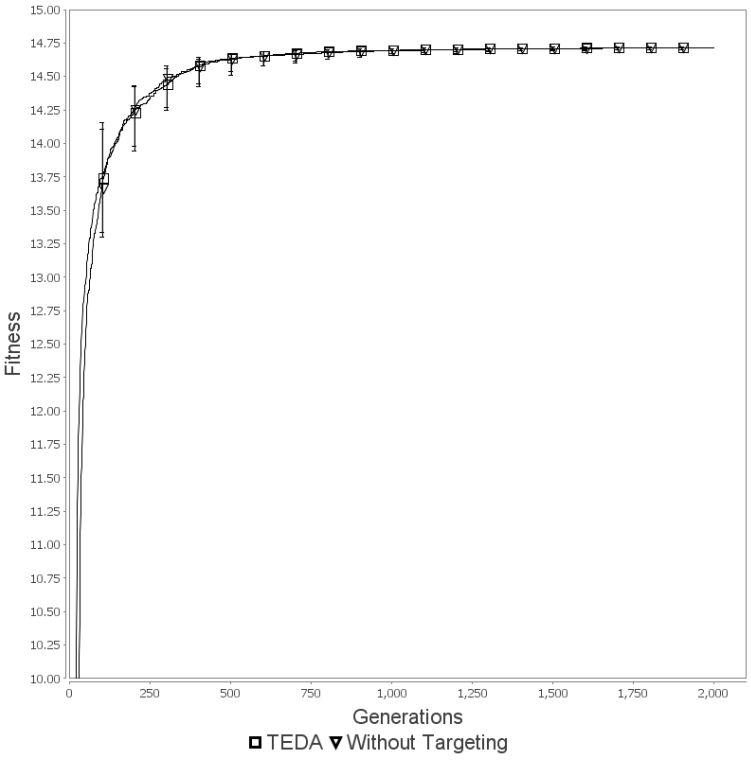

Figure 5.43: TEDA Without Targeting - Performance

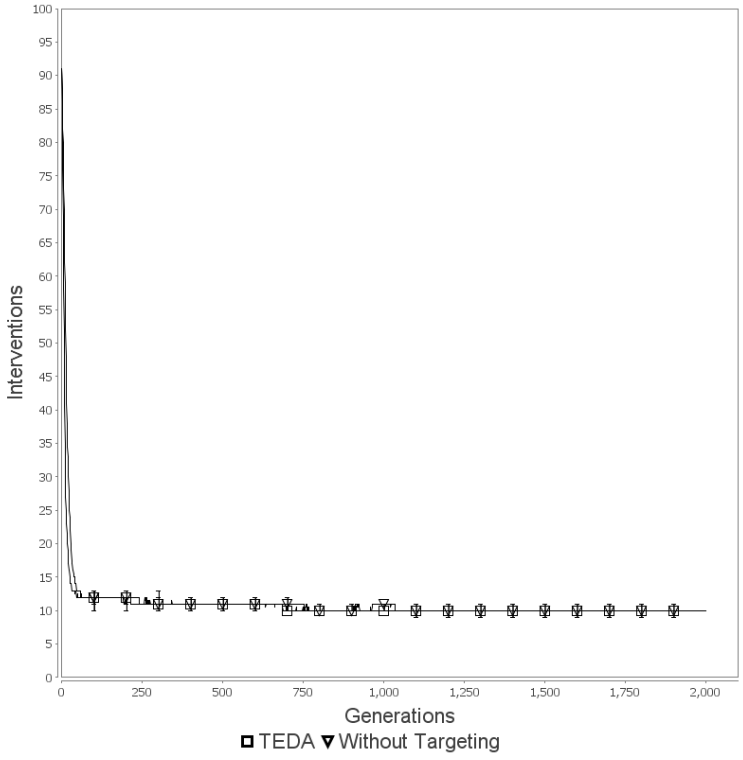

Figure 5.44: TEDA Without Targeting - Control Points (Interventions)

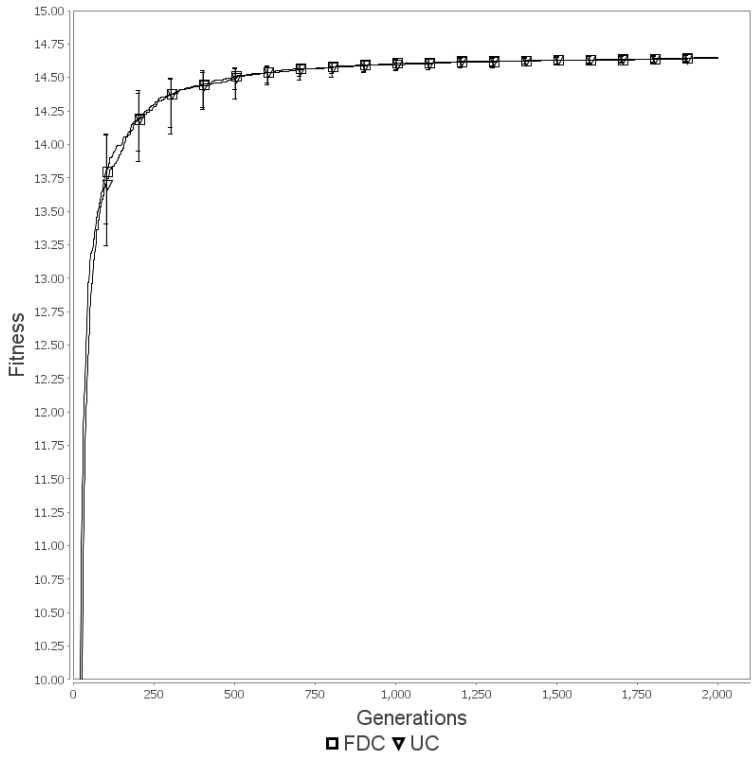

Figure 5.45: FDC vs UC

that Godley used a slightly different experimental setup. He used a tournament size of 2, in contrast to our tournament size of 5, and a mutation rate of 0.01 instead of 0.05. It is possible that FDC performed better than UC in these tests as, with the lower mutation rate, UC became trapped in a local optimum. Targeting may have given FDC an advantage as it enabled it to move into the region of the global optimum before converging, given that the toxicity constraint is likely to place the global optimum into the space of solutions of with a particular, small, number of control points. It is possible that, with these new experimental parameters, TEDA may also be beneficial and so tests were run comparing untargeted TEDA and FDC with these parameters. The results of this comparison are shown in Figure 5.46.

These results show that TEDA performs better than an equivalent algorithm that features no targeting. Explicit transitioning therefore does have a benefit with these parameters. However, FDC does outperform TEDA in this situation. Figure 5.47 shows the median number of control points in solutions generated by these techniques. It is clear from this graph that in this situation the targeting approach is needed to effectively drive down the number of control points used. Solutions found by TEDA use less control points than those found by the equivalent untargeted approach. Solutions found by FDC use less control points still.

It seems that, with a lower mutation rate targeting is necessary to find solutions that are both optimal and that use minimal control points. With a higher mutation rate targeting is not needed. Such solutions will be found with or without it. The use of a hybrid approach, by contrast, slightly improves performance when compared to a standard GA at a higher

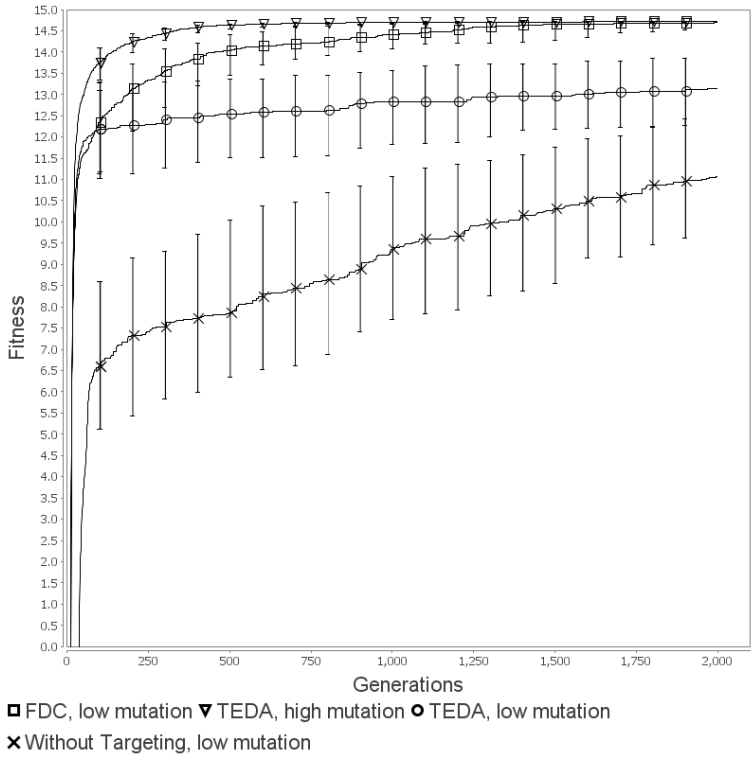

Figure 5.46: Low Mutation Rate Comparisons - Performance

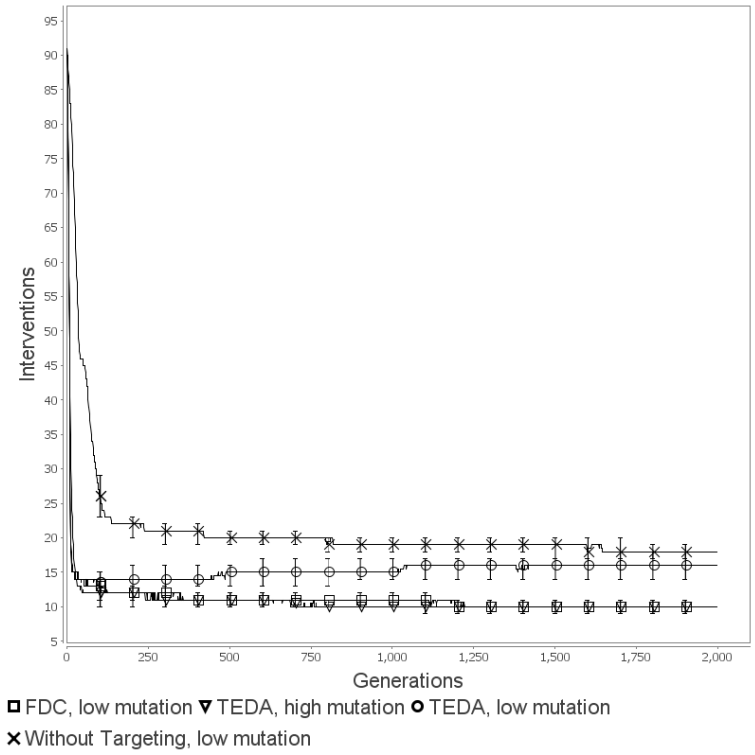

Figure 5.47: Low Mutation Rate Comparisons - Control Points (Interventions)

mutation rate but performs worse at a lower mutation rate. This is possibly because EDAs are more exploitative than GAs, even when incorporated into a hybrid structure.

5.6 SUMMARY

In this chapter we have described the steps involved in the initial development of TEDA through the cancer chemotherapy problem. We have developed a version of TEDA that shall form the starting point for further explorations of TEDA in subsequent chapters.

The main conclusions to draw from this process were:

- With a high mutation rate TEDA is an improvement on FDC although for this problem targeting is not in fact necessary.
- With a low mutation rate targeting is necessary and TEDA does sucessfully use it to improve upon an equivalent algorithm without targeting although in this situation FDC performs better than TEDA.

Overall the best results of all for this problem were seen with a combination of TEDA and a high mutation and so TEDA appears to be a promising approach even though the nature of this problem means that targeting is not necessary. For this reason we shall continue to test TEDA on problems where targeting is necessary to assertain whether it is, in some cases, more effective than any other approach.

As this chapter describes the development of TEDA, below we summarise the main design decisions made during this process. We clarify which possibilities were chosen at each step to form the final version of TEDA.

- At each generation a convergence score $\bar{v}$ should be calculated from the genetic variation among the top 10% of the population multipled by δ, a value that increments by a pre defined α value each generation. The control point adjusted non binary measure as introduced in Section 5.4.4 shall be used to measure convergence. We shall use an α of 0.01.
- $\bar{v}$ should be used to determine the size of two pools. S is the selection pool comprising the fittest s% of individuals in the population. B comprises the parents from which breeding takes place. Using tournament selection B will be selected from S.
- Once B has been obtained it should be used to build a probability vector as in UMDA.
- The number of control points to set should be determined from the FDC rule using the fittest and least fit individual in B (Section 5.3.4).
- The Two Set method should be used to decide which control points to set (Section 5.3). Control points found in every solution in B shall be selected at random and set first and

then control points found in a subset of B should be chosen at random and set according to their marginal probability.

- Elitism with one elite individual should be used (Section 5.3.5).

6

ROUTING

The routing problem, as described in Section 3.2.1, involves finding a route across a landscape by selecting a subset of routers from an available set, arranging them into an order and drawing a route between them. This route is optimized by minimizing the height changes across it but also by minimizing the total number of routers used.

This problem differs from other routing problems because of this second requirement, which means that a route that involves a greater change in height may in fact be preferable if it involves fewer links and so passes through fewer routers. However, a version of this problem will be tested in which the length of routes is not explicitly penalised. There are also no constraints on which order routers may be visited. As explained in 3.2, this is in all other respects a traditional routing problem represented through a 3D landscape instead of a 2D landscape.

6.1 VARIABLE LENGTH CHROMOSOMES

The main difference between this problem and the cancer chemotherapy problem is that, as explained in Section 3.2.1, the logical method of encoding is as a variable length integer string. This string is simply a list of integers consisting of the indexes of the routers that are used in the order in which they are used. It is always therefore exactly the same length as the number of routers in the described route. This representation was chosen for a couple of reasons. At each index a router has to be specified out of a large set and and the only practical way of doing this is through a representation in which each gene contains an integer value specifying a router. An alternative system of using a binary encoding system to specify which routers are on and which are off at each index could be used. However, this would produce excessively long genomes and extra constraints would need to be implemented to ensure that two routers are not set at the same index. A variable length representation is used for a similar reason. It would be possible to map each gene to an index within the route and to use a fixed length representation but this would require for routes to be modified to ensure that there are no gaps in them. The variable length method of encoding is also space efficient as there is no need for genes representing the routers that aren't used to be stored, only the routers which are used.

Because of this representation, this problem creates challenges beyond those that were found in chemotherapy. A breeding process such as that used for chemotherapy that first

selects an index where an intervention should appear and then chooses a value for this index as a secondary concern is not suitable for this problem for two reasons. Firstly, because the genome must always be a valid ordered list without gaps and secondly, because the values that each gene should be set to are as important as the indexes themselves and both the order of routers and which routers are used are intrinsically linked and should be considered together.

As TEDA was based on an EDA of the UMDA variety, which used binary encoded strings, the simplest solution to this problem is to convert the chromosomes into a binary format and add constraints to ensure that a valid route is produced. We used this format to run tests with TEDA, UMDA1 and 2 and FDC. In the binary format each router was encoded using n bits, with n being the maximum router index and so a maximum value of a gene. To denote that the router at index i has a value k a 1 would be placed in the genome at index $in + k$. All other genes would have a value 0. This format is still variable length, except that now the length is equal to nl bits where l is the total number of routers used. This new format allows us to run a simple version of UMDA, using equation 5.2 as described in Section 2.8. In this case ρ is a probability vector of length $n(l_{max})$ where l_{max} is the maximum possible number of routers and each value in ρ is the marginal probability of a particular router being found at a particular index.

As this was not the natural format for this problem this created two additional complications. In order to ensure that UMDA produces a genome that does not contain multiple routers at the same index and that does not contain gaps between routers the breeding process had to be changed as follows:

1. The chromosome is converted into binary format.
2. The probability model is built.
3. Elements in ρ are selected in a random order. If ρ_i has been selected then element i is set in the new solution with probability ρ_i.
4. Once a value has been selected for a given index all other possible values for that index are removed from ρ and cannot be selected.
5. The chromosome is converted back to integer format.
6. The integer chromosome produced will inevitably contain gaps, leaving a pattern of 0s and positive values. The 0s are simply removed and the chromosome compressed. For example, the string '500708000' would become '578'. This means that the exact location of the intervention is not preserved although the order is preserved. The order is more important than the universal index of a router within the route as we wish for the overall shape of a route to be maintained even as unnecessary detours are removed.

This provides us with a version of UMDA with which to test TEDA against. A modified version of TEDA is built on to this version of UMDA and compared against it to establish whether the TEDA targeting and transitioning approach offers improvements. The version of FDC against which we compare is simply TEDA using only 2 parents selected through tournament selection from a selection pool equal to the whole population. As explained in Chapter 5, this is functionally equivalent to FDC. For one and two point crossover there was no need to adapt chromosomes in this way and so the natural, variable length integer format was used.

6.2 EXPERIMENTS AND RESULTS

In this section we compare the performance of TEDA on this problem with the performance of FDC, standard GAs and EDAs.

6.2.1 *Experimental Method*

TEDA and the other EAs were tested on the routing problem with the same parameters as used for the other problems, outlined in Section 4.2. As well as TEDA, the algorithms used for comparison are UMDA, FDC and one and two point crossover. All of these approaches except for one and two point crossover are adapted to the variable length encoding system through the method described in the previous section. One and two point crossover were used in place of uniform crossover for this problem as they do not require the kind of adaptation described previously and can be applied directly to variable length encoding systems.

Parameters specifically connected to the routing problem are given in Table 6.1. A full explanation of these parameters is given in Section 3.2.1 but a summary is provided here. The start (s_x, s_y) and end (e_x, e_y) points are positioned at the edge of deep valleys in the landscape that an optimal route should be able to follow. The function used to produce the landscape is given in Equation 6.2, in which $h(x, y)$ is the height of the landscape at coordinate (x, y). K is the set of routers. x_{min}, x_{max}, y_{min} and y_{max} make up the boundaries of the landscape. z provides the granularity with which a route is sampled to obtain the change in height across its length. w is a scaling parameter used to define how high the hills of the landscape are.

The fitness function is given in Equation 6.1. The fitness of route C, $f(C)$ is calculated as the sum of the height changes across each segment of the route, denoted as $t(g, g+1)$, the change in height between node g and node $g+1$. A penalty value multiplied by the number of routers is then subtracted from this initial fitness value.

$$f(C) = \sum_{g=0}^{g=|C|-1} t(g, g+1) + |C|penalty \tag{6.1}$$

$$h(x, y) = w \cdot \sin(2(\frac{2x}{\cos(y) + 2})) \tag{6.2}$$

Parameter	Value
$\|K\|$	1000
s_x	5
s_y	-5
e_x	5
e_y	5
x_{min}, x_{max}	-5,5
y_{min}, y_{max}	-5,5
z	0.1
$max\|C\|$	50
g	10
Generations	2000
w	10
penalty	1

Table 6.1: Routing Parameters

To match the variable length format, mutation and initialisation have also had to be slightly altered.

6.2.1.1 *Initialisation*

To initialise a population for each solution a length l of between 0 and the maximum permissble length of a route, $max|C|$, is chosen at random. A new empty chromosome of length l is then generated. This is iterated through and, for each gene, a value from 0 to $|K|$ is chosen at random.

6.2.1.2 *Mutation*

The only change that needed to be made to the mutation technique is that new genes are inserted into the solution at a random point within it's current length and all following genes have their indexes incremented by one, thus increasing the length of the solution by 1. Likewise, if a gene is removed a random gene is chosen and removed and all following genes are moved down one place. In the same way as before, mutation is attempted a number of

times equal to the chromosome's maximum length ($max|C|$) and carried out each time with a probability of 0.05. If mutation is to be carried out then either a gene is added or a gene is removed, with equal probability. The value of a new gene added is chosen at random from 0 to $|K|$.

6.2.2 *TEDA for Variable Length Chromosomes*

The encoding of this problem introduces new challenges for TEDA. When adapting the version of UMDA described above to work with TEDA it was necessary to answer two questions.

1. For transitioning, it is necessary to establish how to measure chromosome variation in variable length chromosomes.
2. We need to establish in which order routers are selected as the system used in chemotherapy of giving priority to a set of control pointss found in all solutions (the S_{all} set) is complicated by the variable length structure.

These two questions are answered in Sections 6.2.2.1 and 6.2.2.2 respectively.

6.2.2.1 *Transitioning*

The existing method of transitioning involves measuring diversity by measuring the hamming distance between pairs of solutions (See Section 5.4.4). A given gene in two solutions is considered to match only if it shares the same value at the same index. This is problematic in variable length chromosomes. As gaps between routers are not allowed, the final stage of crossover is, as explained previously, to simply remove any gaps and condense the resulting chromosome. This means that two chromosomes that are almost identical except for one containing a router early on which is lacking in the other will be considered to not match at every index after this router, creating a genetic difference that doesn't reflect their true similarity.

An alternative is to obtain the similarity between solutions by counting the number of routers that appear in both solutions regardless of where they appear. The overall process of transitioning remains the same as in Section 5.20. The measure of variation used to determine the size of the selection pool and breeding pool, ν, is still calculated through equation 5.7. The only difference is that the set of interventions that is in both parents, $I_{x_1} \cap I_{x_2}$, now consists of the routers that appear in both parents regardless of their index and likewise the set that is in only one parent consists of routers that appear in just one parent.

As it was observed in the chemotherapy problem that the best performance was achieved when this process was controlled by an α of 0.01 that ensured that transitioning occurred at a smooth rate even when the change in actual population diversity is not as smooth. This same α was therefore used in these tests for both variations.

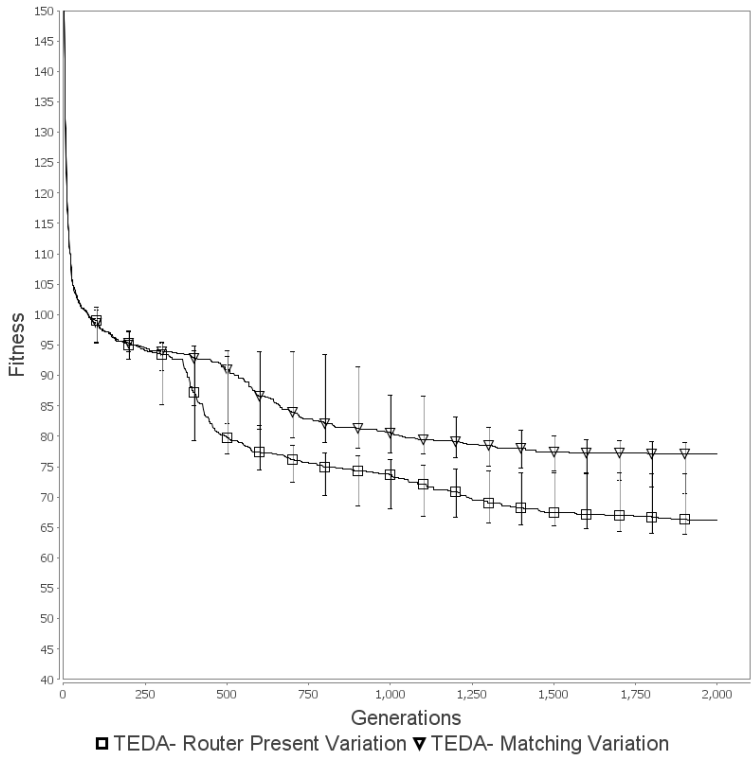

Figure 6.1: Convergence Measures for Variable Length Chromosomes (The Routing Problem)

In Figure 6.1 we compare the two methods. In *TEDA- Router Present Variation* the new method based on which routers appear is used whereas in *TEDA- Matching Variation* the original transitioning method in which both the router and the index need to match is used. In all other respects, TEDA is as described in Chapter 5 and the underlying EDA is as described in Section 6.1 for both variants.

The results of this study can be seen in Figure 6.1. It can be seen that the router present variation performs better than the original approach for this problem.

We also considered whether this affects our overall observation that transitioning through measuring population convergence with an α of 0.01 is the most effective transitioning method. We compared this method with the probabilistic transitioning technique, as introduced in Section 5.28 in the previous chapter. In both techniques the difference between solutions, whether used to calculate a mean population difference in the case of the population diversity method or used to compare two solutions as in the probabilistic method, was obtained through the router present technique.

Figure 6.2 shows how these two approaches compare. We can see that, unlike in chemotherapy, the probabilistic transitioning method performs better throughout the test than the population convergence based method and so the probabilistic method will be used in subsequent testing unless otherwise stated.

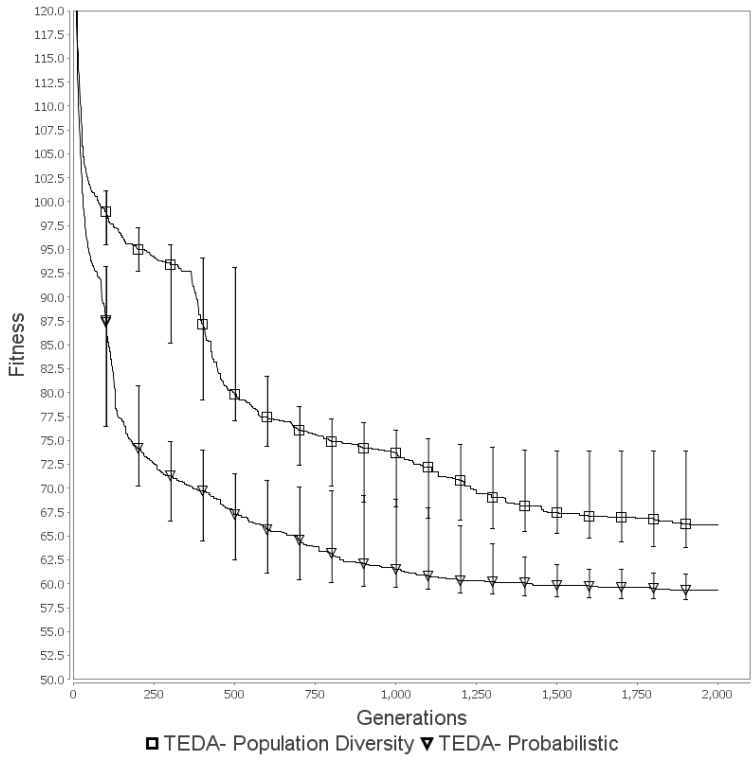

Figure 6.2: Transitioning for Variable Length Chromosomes (The Routing Problem) - Probabilistic vs Variation Based

6.2.2.2 *The Order of Selection*

In TEDA the order in which routers are selected to be potentially set is of crucial importance as only a small number of routers are used and only the first few routers have a chance of being selected before I_T has been reached and no more routers are considered. A key design feature of TEDA is that routers which are found in every solution are placed into the set S_{all} and are selected first so that they appear in the majority of new solutions. The question when TEDA is applied to a variable length chromosome problem is how do we define a router as appearing in every solution. In the chemotherapy problem this is judged to be the case if a gene has a value greater than zero in every solution. It is not possible to do this in a variable length problem because chromosomes do not contain zero valued genes.

We could simply adopt this approach but for the sparse binary strings into which the EDAs convert the original integer strings. This would place into S_{all} every index/value pair that appears in every solution instead of just every index. The disadvantage of this approach is that there are so many possible index/value combinations that it seems unlikely that a single combination will appear in a large number of parents. This will mean that S_{all} is likely to be an empty set and so the order in which routers are selected will be completely random. However, early on when the algorithm is expected to be explorative this is unlikely to be a problem and it is possible that later on patterns are discovered so that some routers do

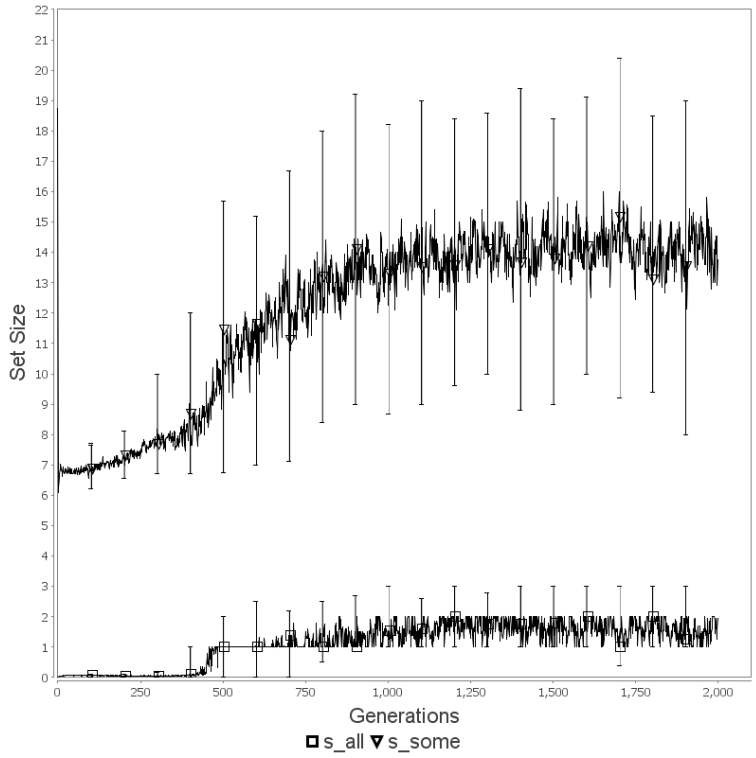

Figure 6.3: Intervention Selection for Variable Length Chromosomes (The Routing Problem)

appear at the same index in multiple solutions. This method was tested in Figure 6.3 where the results show the mean size of S_{all} and S_{some} throughout this test.

As can be seen, the size of S_{all} to begin with is 0. It quickly increases to around 2. It is therefore not empty throughout the test. As solutions have a maximum size of 50, whether prioritising only 2 solutions is useful or not depends on how long the solutions generated on average are. The results in Figure 6.4 show the median size of the solutions that TEDA generated while using this approach. As can be seen, in the first generation the fittest solutions have a length of only 2 routers but later on the length of these solutions increases to around 5 or 6. With solutions of this size it seems likely that even if the set S_{all} contains only two solutions it is still fulfilling its purpose of ensuring that positive patterns are passed on. For this reason, simply applying the standard method of creating S_{all} to the sparse binary vector would appear to be effective and we shall therefore maintain this approach.

6.2.3 *Comparing TEDA to other approaches*

In this section the performance of TEDA is compared to other EDAs and GAs. To begin with, TEDA is compared to two standard GAs using one and two point crossover.

The results of this comparison are shown in Figure 6.5. TEDA appears to consistently perform better than both approaches throughout the test. Table 6.2 shows the different scores

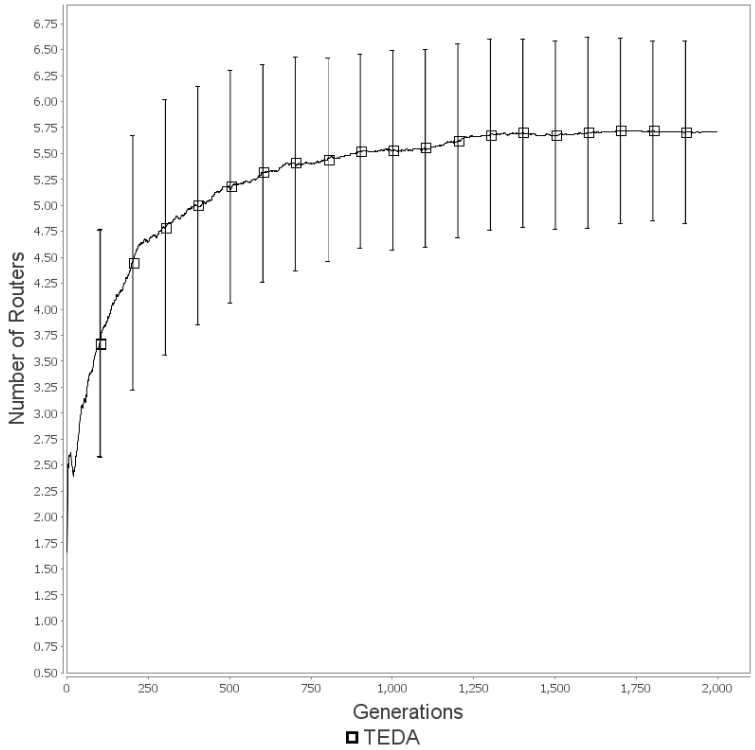

Figure 6.4: TEDA - Control Points (Routers)

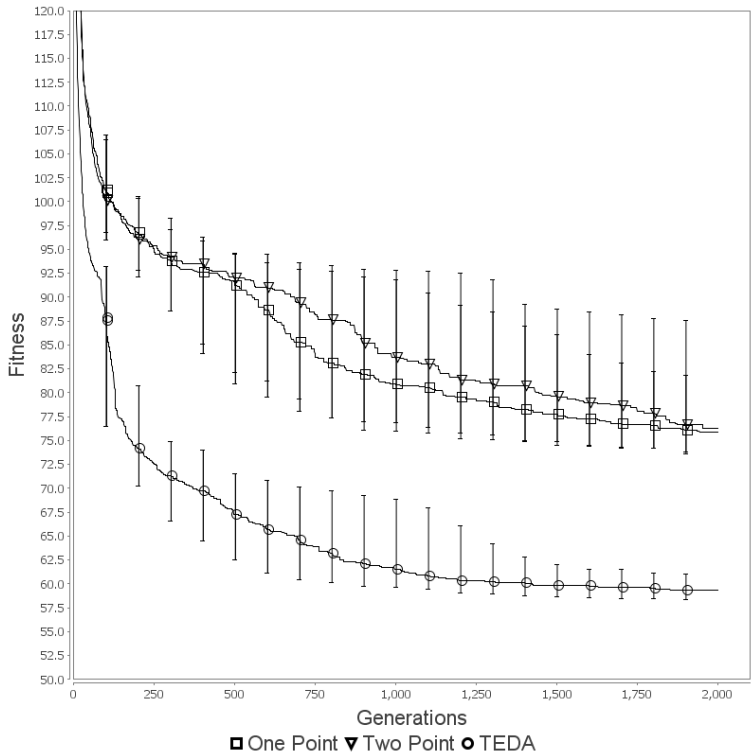

Figure 6.5: TEDA vs GAs - Performance

obtained through a Kruskal Wallis analysis of these results. It is clear from this table that the improvement that TEDA offers is statistically significant at every stage. The results in Figure 6.6 show the median size of solution discovered by these approaches.

Generations	TEDA vs One Point	TEDA vs Two Point
200	***262.74***	***255.88***
400	***252.47***	***269.53***
600	***261.54***	***283.33***
800	***268.9***	***288.82***
1000	***266.81***	***295.97***
1200	***270.73***	***299.39***
1400	***276.47***	***297.75***
1600	***279.04***	***298.61***
1800	***280.02***	***295.61***
2000	***282.64***	***293.11***
Threshold	41.5	

Table 6.2: TEDA vs GAs- Kruskal Wallis for Routing

From these results we can see that in the initial population the fittest solutions are very small. Due to the penalty, in the first few generations before genuinely useful solutions have been discovered, the fittest solutions are simply the smallest solutions regardless of whether they contain useful patterns for solving the problem. This remains the case for the first few generations as solutions similar to these small solutions proliferate across the population until routes that follow the landscape are discovered and larger, but most likely genuinely effective, solutions are generated. As the most effective solutions are larger than those in the first few generations the average size of solution within the population must be increased. In the GAs this happens to a lesser extent than in TEDA as it seems that TEDA's explicit targeting enables it to discover the optimal size of solution and drive towards it.

In the next set of results TEDA is compared to FDC. These results are shown in Figure 6.7. It appears from these results as though TEDA consistently finds better solutions than FDC. Kruskal Wallis analysis (Table 6.3) demonstrates that throughout the test the improvement that TEDA offers is statistically significant.

Figure 6.8 shows the size of solutions generated. It is clear that there is a similar pattern to the case of one and two point crossover, early on TEDA produces very small solutions but then self corrects to discover larger solutions. FDC, despite using targeting, is unable to do this. It may be too explorative. One possible explanation is that, if there exists a population where

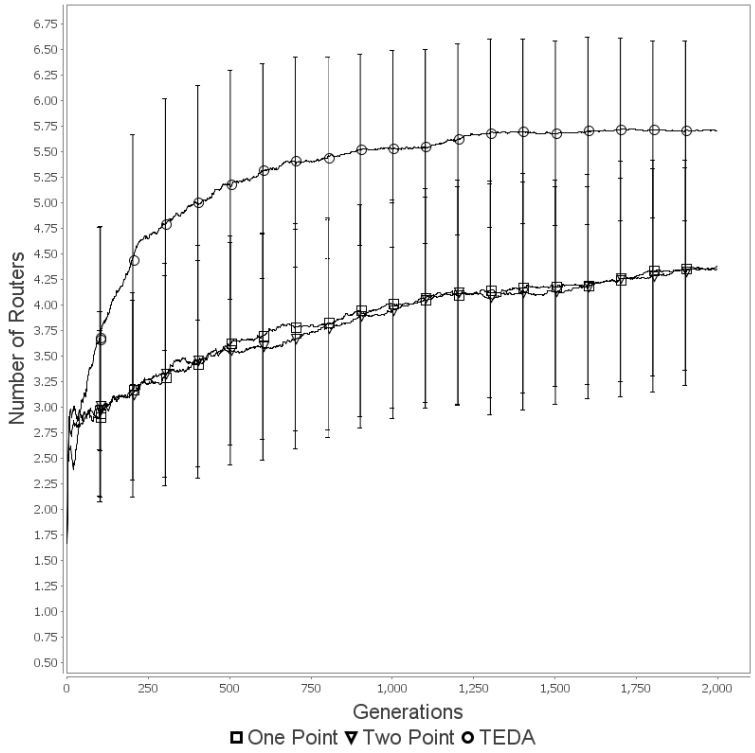

Figure 6.6: TEDA vs GAs - Control Points (Routers)

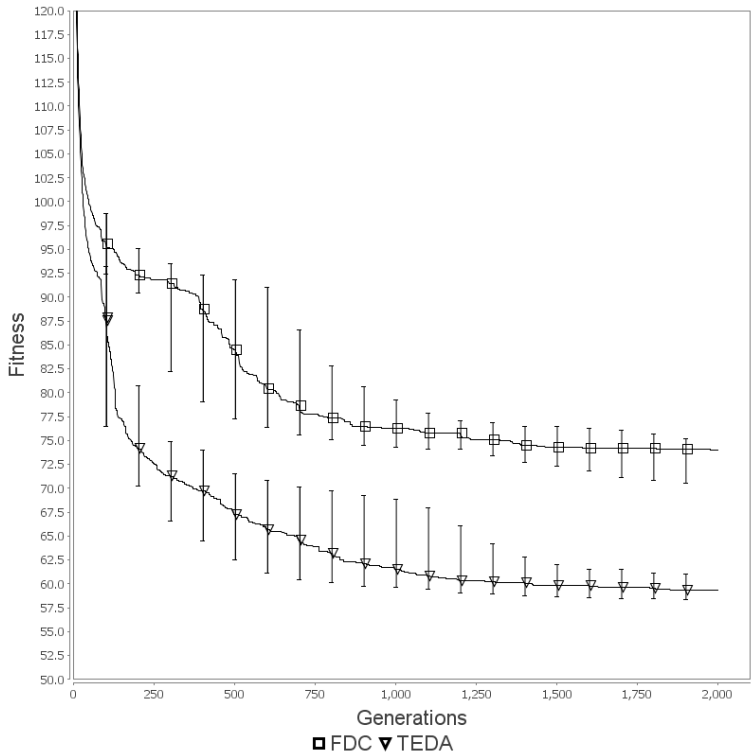

Figure 6.7: TEDA vs FDC - Performance

Generations	TEDA vs FDC
200	***147.67***
400	***160.79***
600	***173.51***
800	***179.86***
1000	***183.49***
1200	***185.69***
1400	***186.7***
1600	***186.94***
1800	***185.61***
2000	***183.11***
Threshold	22.66

Table 6.3: TEDA vs FDC - Kruskal Wallis for Routing

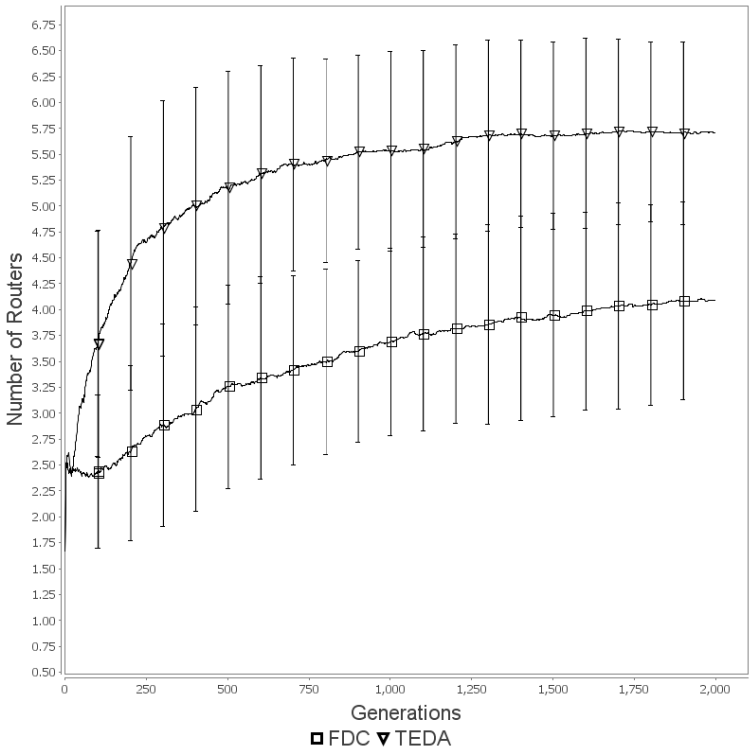

Figure 6.8: TEDA vs FDC - Control Points (Routers)

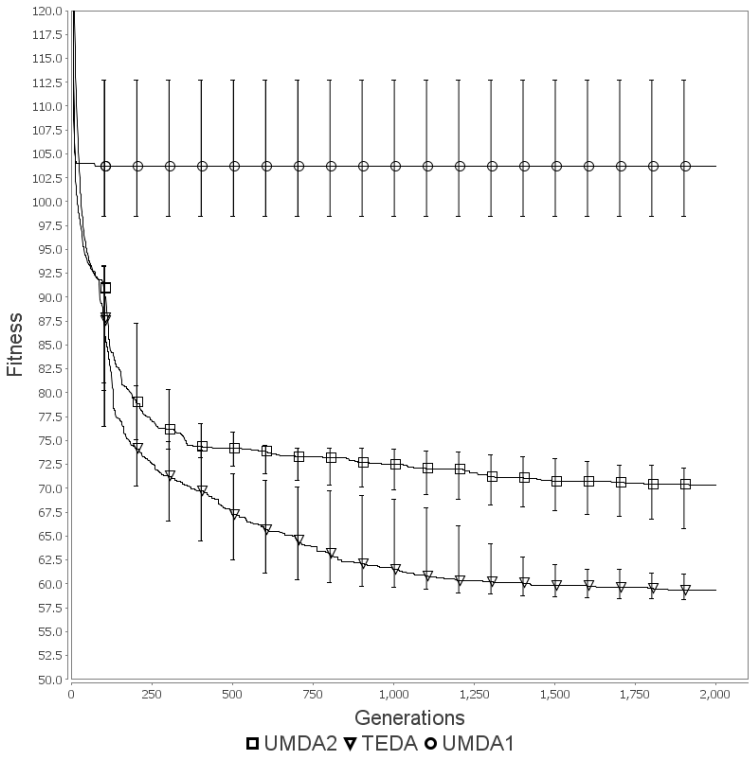

Figure 6.9: TEDA vs EDAs - Performance

the majority of solutions are extremely small and it is only a small number of the very fittest solutions that are slightly larger, then a high selection pressure is needed in order to select these solutions and so for targeting to be effective.

Figure 6.9 shows the results of comparing TEDA with UMDA. As with chemotherapy (see Section 5.5.2), two types of UMDA are used, UMDA2 uses identical parameters to TEDA whereas UMDA1 uses parameters that more closely match those found in literature.

TEDA performs better than both variations of UMDA throughout the test. UMDA1, it appears, prematurely converges and remains stuck at a relatively poor fitness. The Kruskal Wallis analysis of these results, as shown in table 6.4, confirms that TEDA consistently outperforms UMDAs 1 and 2 by a statistically significant margin.

The results in Figure 6.10 shows the solution size for this problem. As with the other tests, all algorithms initially find very small solutions but then TEDA is more effective than the standard EDAs at increasing the size of solution to an optimal level. UMDA1 appears to perform poorly due to it's inability to increase the size of solutions after the initial drop. This may be for reasons similar to those discussed in the context of chemotherapy, because a probability model made from 50% of the population is likely to provide information of which routers are popular where but is not likely to provide useful information on the presence of a few larger, fitter, solutions.

It has now been established that the targeting capabilities of TEDA enable it to find solutions of a higher quality than non targeted methods. It has also been established that TEDA performs

Generations	TEDA vs UMDA2	TEDA vs UMDA1
200	***73.17***	***330.33***
400	***105.82***	***351.01***
600	***126.96***	***362.36***
800	***142.25***	***370.29***
1000	***150.53***	***374.56***
1200	***162.16***	***380.46***
1400	***167.42***	***383.13***
1600	***170.06***	***384.46***
1800	***171.29***	***385.07***
2000	***170.93***	***384.89***
Threshold	41.5	

Table 6.4: TEDA vs EDAs - Kruskal Wallis for Routing

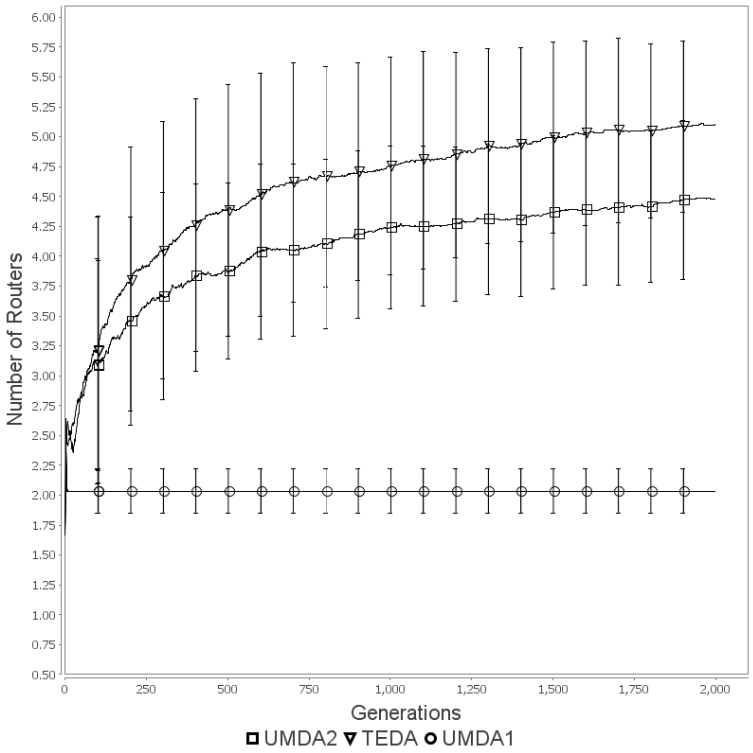

Figure 6.10: TEDA vs EDAs - Control Points (Routers)

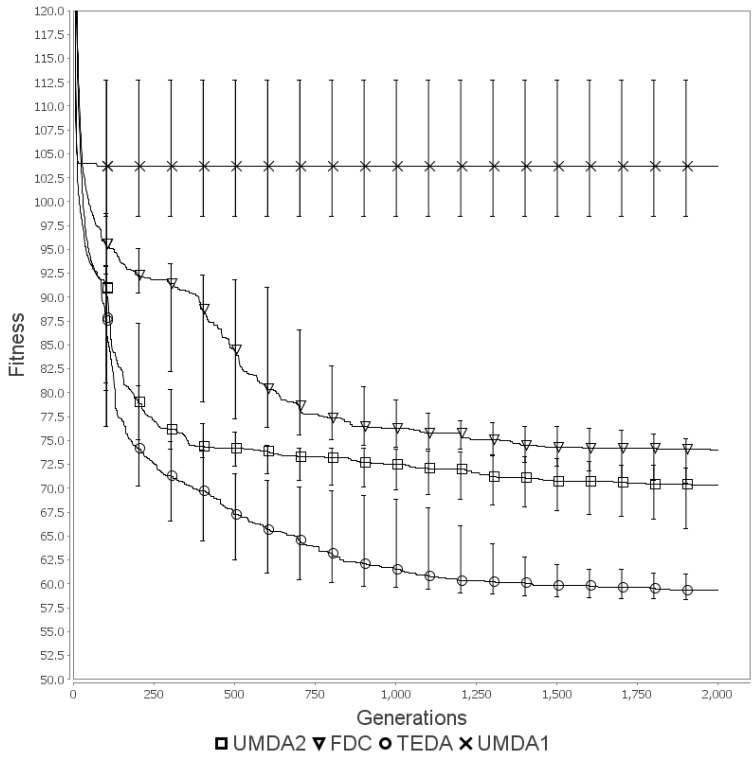

Figure 6.11: EDAs vs FDC - Performance

better than FDC on this problem. In the following two graphs FDC is compared to EDAs to establish whether targeting within the FDC framework is still enough to perform better than untargeted EDAs or whether TEDA is necessary. Figure 6.11 shows the results of this comparison in terms of fitness scores and Figure 6.12 shows the comparison in terms of solution size.

FDC, unlike TEDA, continues to perform worse than UMDA2 throughout the test. Looking at the size of solutions discovered by these approaches (Figure 6.12) it can be seen that, although the targeting ability allows it to recover from over targeting to some extent, it is less successful at self correcting than TEDA or even UMDA2 and continues to only find solutions that are smaller than the optimal size throughout the test.

6.2.4 *The Importance of Targeting*

With chemotherapy we discovered that the only benefits that TEDA showed were due to its transitioning approach and targeting seemed irrelevant except for when the mutation rate was lowered. We therefore, now compare TEDA to an identical approach which is only different in that the targeting does not take place. The results in Figure 6.13 show the performance of this algorithm compared to standard TEDA.

It appears as though TEDA, with targeting performs slightly better than an equivalent algorithm without targeting. This improvement is small but the results of a Kruskall-Wallis

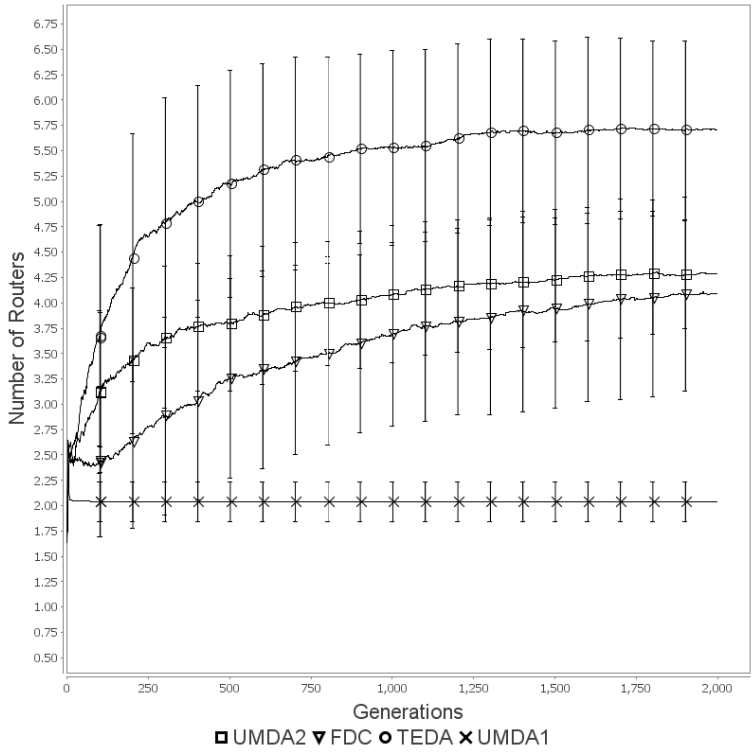

Figure 6.12: EDAs vs FDC - Control Points (Routers)

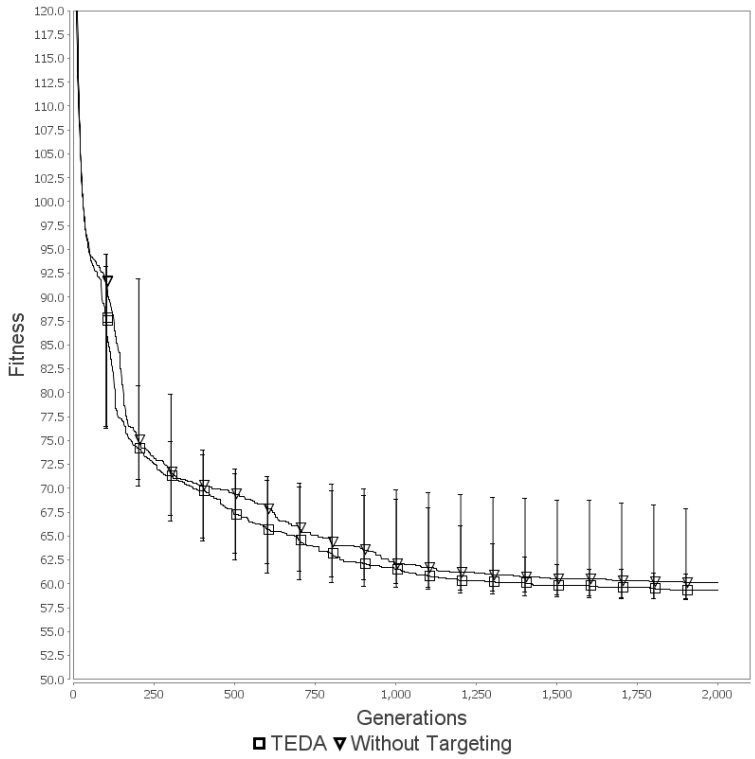

Figure 6.13: TEDA without Targeting - Performance

analysis, given in Table 6.5, show that it is statistically significant towards the latter half of the test.

Generations	TEDA vs Without Targeting
200	***23.21***
400	10.55
600	18.58
800	20.74
1000	18.35
1200	***24.42***
1400	***30.01***
1600	***33.74***
1800	***31.54***
2000	***30.55***
Threshold	22.55

Table 6.5: TEDA without Targeting - Kruskal Wallis for Routing

The results in Figure 6.14 show the size of solutions discovered by these methods. This shows that with targeting, TEDA finds solutions that are larger than those found without targeting as it is better able to correct for the initial drop in solution size. In general, targeting can provide a benefit but it is not always necessary. If present it does not cause a significant loss in performance.

6.2.5 *Comparisons Using Population Variation Transitioning*

At the start of this chapter we compared different transitioning techniques. We established that measuring the difference between solutions had to be performed differently due to the different encoding of this problem. We also established that a different overall method of transitioning, the probabilistic method, performs better on this problem than the population diversity method that performed best on chemotherapy. It would be a disadvantage of TEDA if it is necessary to choose the transitioning method based on the problem in order to make it an effective algorithm and so in this section we will establish whether TEDA does still perform better than the other algorithms when the population diversity based transitioning technique is used. The results shown in Figure 6.15 compare the performance of TEDA using the population diversity transitioning technique with an α of 0.01, as in chemotherapy, with the other algorithms.

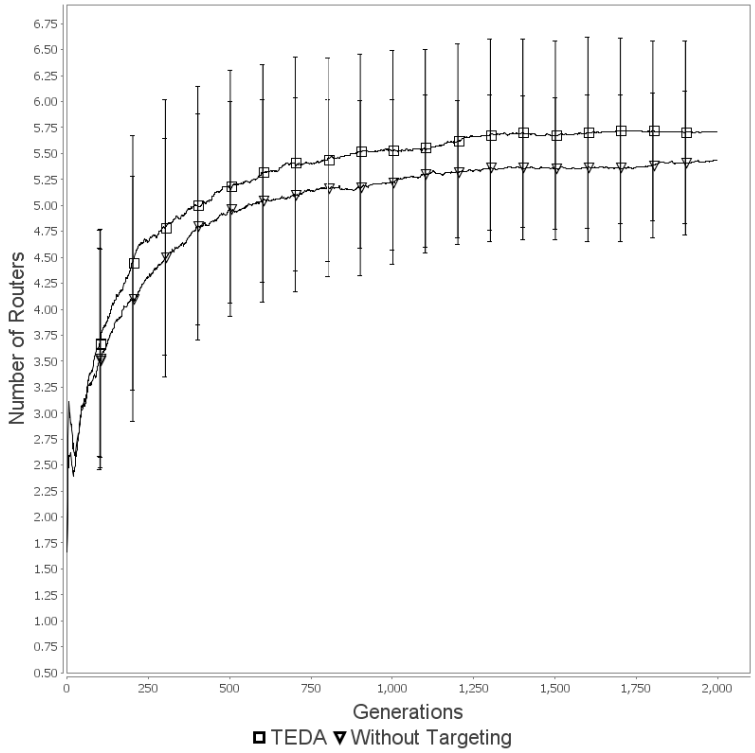

Figure 6.14: TEDA without Targeting - Control Points (Routers)

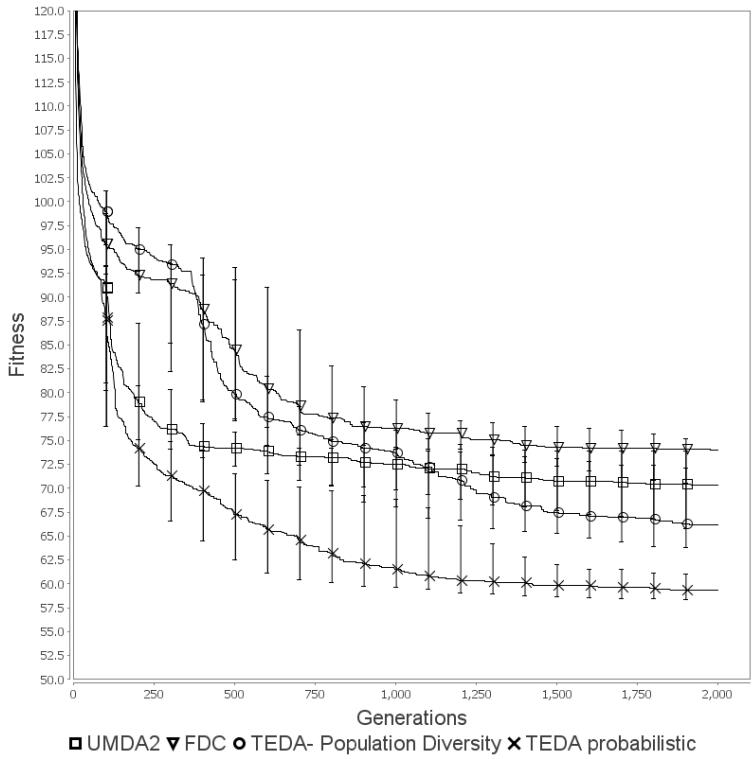

Figure 6.15: Routing Problem using Variation Based Transitioning - Performance

It can be seen from this comparison that, unlike with probabilistic transitioning, this version of TEDA is initially slow at finding good solutions and ultimately finds solutions of the same standard as UMDA2. This approach can therefore be used on this problem and the transitioning technique is sufficient to elevate it above the performance of the non transitioning FDC. However, a probabilistic transitioning technique is needed to see any real benefits. In the chemotherapy problem the probabilistic transitioning technique performs slightly worse than FDC and EDAs (see results from the chemetherapy chapter, Section 5.28).

It will be necessary to establish which version of TEDA should be considered the standard version. As the chemotherapy problem is less difficult than the routing problem and can be solved to provide a near optimal solution by any of the techniques without the need for targeting at all, it is our initial inclination that we should design TEDA based on its performance on problems such as the routing problem and so the probabilistic technique should be our standard version. Although TEDA using probabilistic transitioning appears to be slower than the other approaches at solving the chemotherapy problem it does ultimately achieve the same optimum. Probabilistic transitioning also appears to perform well without the need to introduce a tunable parameter to control the speed of transitioning, as in the population diversity based method. To establish whether the probabilistic transitioning method is the most suitable standard variation both variations will be tested on the feature subset selection problem (see Chapter 7).

6.2.6 *Testing Without a Penalty*

The following tests show how these different approaches perform on the problem when the penalty incorporated into the fitness function is equal to 0. These tests are necessary to establish whether it is the fact that in this problem solutions of a certain length, specifically smaller solutions, are explicitly rewarded that enables TEDA to perform well or whether TEDA will still perform well even when smaller solutions are not rewarded.

The results in Figure 6.16 show how TEDA and the other approaches performed on the routing problem without a penalty. The results are similar to those with a penalty, TEDA still performs worse than a standard EDA early on but later outperforms all of the alternative approaches. The KW results shown in Table 6.6 confirms that the advantage that TEDA shows over the other approaches is statistically valid from generation 800 onwards.

The results in Figure 6.17 show the size of solutions generated during these tests. Now that the penalty has been removed the solutions found are larger, ultimately reaching a size of 7 or 8, not 5 or 6. One and Two point crossover also find solutions that are larger than in the penalised situation. The overall pattern, however, is the same. Both with and without a penalty, the initial solutions found are too small and each algorithm must correct for this fact and increase the size of solution found. This is unsurprising as, even without a penalty the

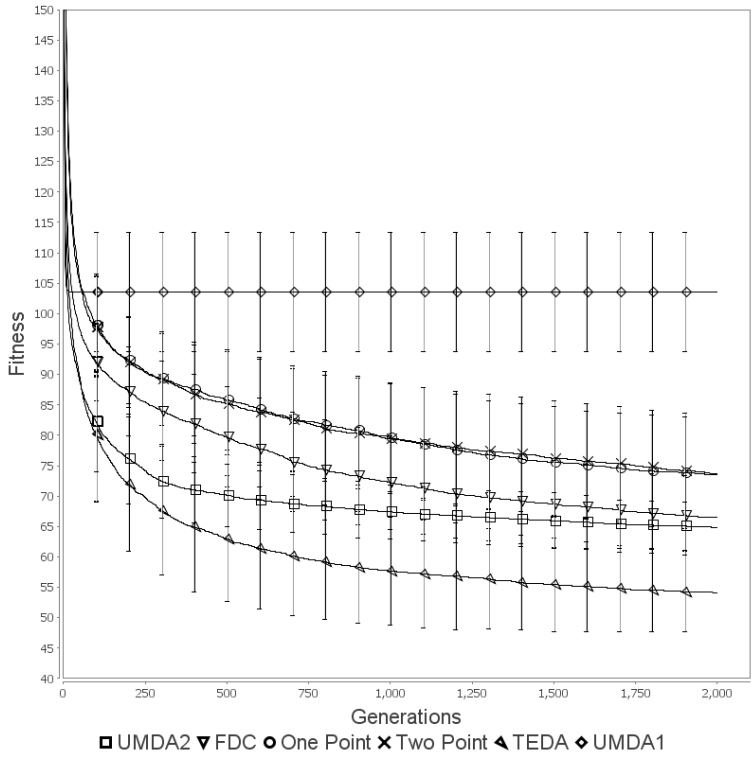

Figure 6.16: Routing Problem with no Penalty - Performance

Generations	TEDA vs UMDA2	TEDA vs FDC	TEDA vs One Point	TEDA vs Two Point	TEDA vs UMDA1
200	***382.66***	10.35	***230.58***	***217.97***	***147.42***
400	84.48	***285.59***	***489.02***	***461.15***	***280.23***
600	101.26	***395.34***	***604.71***	***595.54***	***494.31***
800	***146.2***	***366.76***	***605.92***	***596.2***	***549.89***
1000	***159.34***	***348.49***	***593.89***	***589.51***	***580.83***
1200	***180.62***	***335.35***	***574.38***	***587.29***	***618.17***
1400	***193.57***	***328.24***	***558.69***	***583.59***	***641.56***
1600	***206.52***	***318.88***	***560.33***	***573.71***	***665.82***
1800	***219.26***	***311.69***	***552.39***	***570.01***	***685.49***
2000	***231.08***	***307.54***	***556.13***	***560.82***	***703.03***
Threshold	107.41				

Table 6.6: Penalty=0 - Kruskal Wallis for Routing

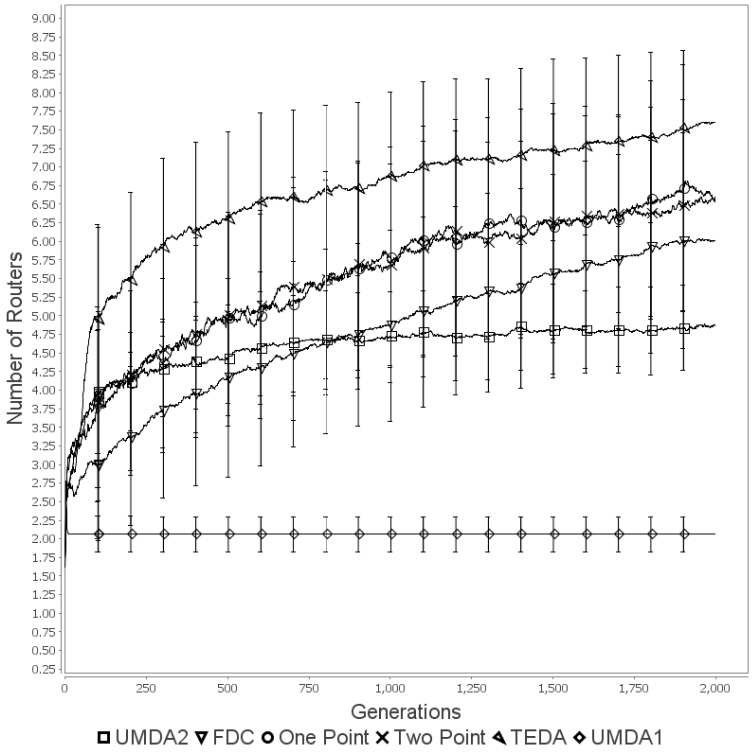

Figure 6.17: Routing Problem with no Penalty - Control Points (Routers)

quickest route across a landscape before any information about the topology has been learnt is likely to be a direct route. Both with and without a penalty TEDA is more effective than the other approaches at increasing the size of solutions.

6.2.7 *The Routes Found*

To demonstrate to what extent TEDA is in fact finding routes that follow the contours of the landscape we have included graphical representations of various routes across the landscape. The landscape is shown as a 2D image with lighter colours indicating higher areas and darker colours indicating lower areas. A fit route is one that involves minimal changes in colour. Potential routers are shown as black dots in the landscape and routes are shown as black lines.

Figure 6.18 shows the best route found in the initial population for 50 runs of the test. Figure 6.20 shows the best route found at generation 2000 by TEDA in the non penalized version of the problem and Figure 6.19 shows the best route found at generation 2000 by TEDA with a penalty of 1. In both cases TEDA is able to find routes that generally follow the contours of the landscape. Interestingly, applying a penalty appears not to limit TEDA's ability to find routes that follow the contours of the landscape to any great extent. The routes found by TEDA in the penalised approach contain less routers, as we have shown, but this may be due to

the fact that situations where the same router appears twice or more in a row in a route are eliminated. Where no penalty is applied there is no incentive to eliminate these situations.

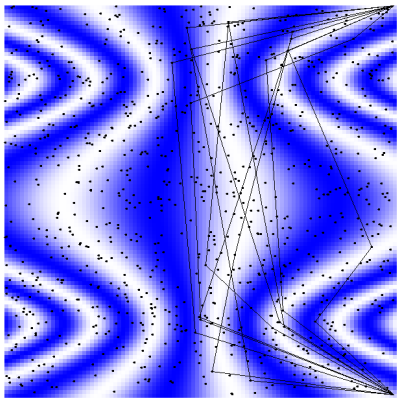

Figure 6.18: Map of Routes Found in Initial Population in Routing Problem

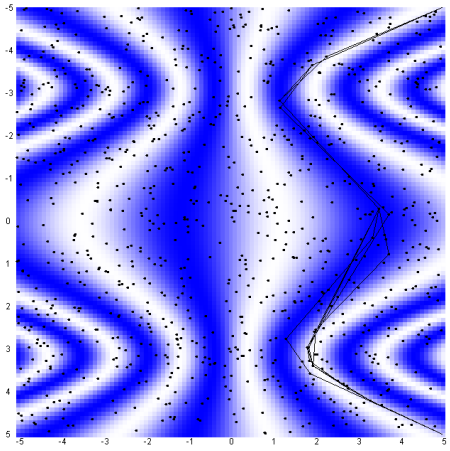

Figure 6.19: Map of Routes Found in Final Population in Routing Problem (Penalty = 1)

6.3 SUMMARY

In this chapter TEDA has been applied to the problem of finding a route across a landscape of minimum cost, with minimal height changes and minimal hops. Overall TEDA performs better than any of the untargeted methods against which it was compared or against FDC.

We have demonstrated that TEDA ultimately finds shorter and fitter solutions for this problem than any of the alternative methods that it was tested against. It appears as though, for this problem, a targeting principle is beneficial and that the targeting process can benefit from the abilities of an EDA to hone in on the best solution. For this reason TEDA performs better both than FDC and the untargeted approaches.

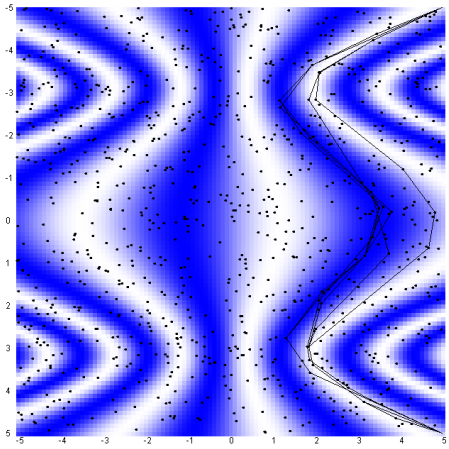

Figure 6.20: Map of Routes Found in Final Population in Routing Problem (Penalty = 0)

The benefits that TEDA shows apply both when a penalty that explicitly encourages the discovery of smaller solutions is applied and when there is no such penalty. It is interesting that TEDA performs well in both of these situations as we may expect that with a penalty TEDA will be driven too quickly to find the smallest solutions possible and may ultimately lose positive patterns that are associated with good solutions and yet this does not appear to happen. Conversely, it may be expected that without a penalty there is no advantage to be gained in finding solutions of a particular size and so targeting may not be useful at all, and yet apparently there is still an optimal size of solution that TEDA appears to be more effective at discovering than the other approaches.

It was our intention to develop a version of TEDA that would perform well on any problem. However, when comparing this problem to chemotherapy it has been noted that one version of TEDA, a version using probabilistic transitioning, performs well on this problem and poorly on the chemotherapy problem whereas an alternative version of TEDA performs well on the chemotherapy problem but less well on this problem. It was our overall conclusion that TEDA should use the probabilistic method and that chemotherapy, though it is possible to solve it with TEDA, is a problem that is sufficiently easy for non targeting algorithms to solve that the benefits of TEDA are less relevant. As has already been noted, in this thesis the relatively easy single drug variant of the chemotherapy problem was used which lacks an additional element concerning toxic side effects which formed part of the more challenging muti-drug variant.

The reason why this benefits from TEDA whereas chemotherapy does not may be because the ideal size of solutions is deceptive in this problem. In the first few generations all algorithms are led towards generating the smallest solutions possible, both with and without a penalty, even though the optimal solution size is larger. TEDA, with its explicit targeting, is more effective than the other approaches at correcting for this tendency to ultimately find solutions of the optimal size. Through these tests we have also shown that TEDA can successfully be

applied to a problem that requires a very different, variable length, encoding system to the chemotherapy problem.

7

FEATURE SELECTION

This chapter discusses how TEDA was applied to a number of Feature Subset Selection (FSS) problems. FSS problems were introduced in Section 3.3. It was explained in this section that, when optimizing FSS problems, the objective is to find feature sets that are small yet effective for the task of classifying a given sample set. One advantage of finding small feature sets is that they are expected to allow for faster classification.

Within the domain of wrapper methods, methods which optimize FSS problems by testing alternative complete feature sets using the final classification technique, a key factor in how long an algorithm takes to find an optimal classifier is the amount of time that it takes to evaluate each candidate feature set. In fact, because wrappers operate by evaluating multiple feature sets they are often considered to be computationally inefficient "brute force" approaches [55] and there is much scope for improving the efficiency of these techniques.

It is for this reason that, in large problems, a wrapper approach that quickly removes a large proportion of features from consideration and greatly reduces the size of the majority of feature sets in the population should be able to evaluate a population quickly. TEDA's targeting capability may enable it to do this.

It was decided that FSS would be a suitable domain on which to test TEDA both for this reason and because it would establish whether TEDA is able to effectively target when the total number of control points is orders of magnitude greater than any problem tested previously.

7.1 THE IMPORTANCE OF FEATURE SET SIZE

To emphasise the importance of finding small feature sets, we present in Figure 7.1 the length of time that was taken to classify a dataset using feature sets of varying sizes. The dataset used is from the Dexter problem, a sparse problem which has a maximum feature set size of 20,000. This graph clearly shows the correlation between the number of features and classification time, with feature sets close to the maximum of 20,000 taking almost half a second to classify the dataset. From this data, we measured a positive correlation coefficient of 0.783742336 between the size of the feature set and the length of time that classification takes when it used.

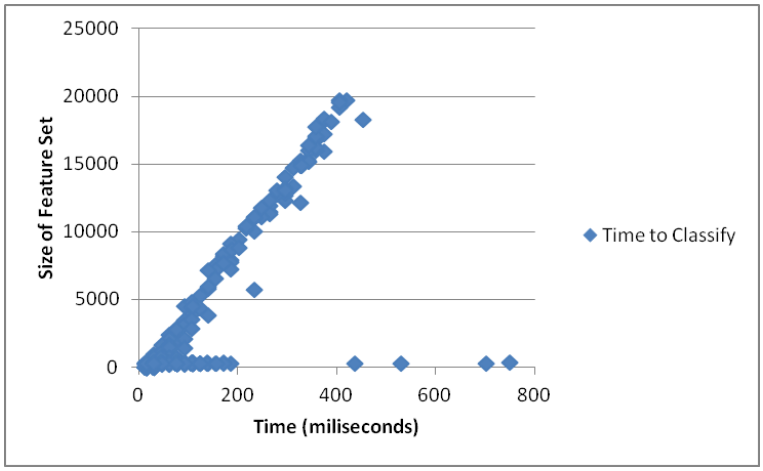

Figure 7.1: Classification Time against Feature Set Size

7.2 State of the art feature selection algorithms: techniques used

Section 3.3 provides an overview of the main FSS techniques that exist. In this section we discuss in more detail the state of the art techniques which have been used as benchmarks against which TEDA has been compared. We considered both filter methods which assess features independantly of the classifier used and wrapper methods which assess complete feature subsets and test each subset with the final classifier.

7.2.1 *Forward and Backward Selection*

Forward Selection (FS) and Backward Selection (BS) were introduced in Section 3.3. These are two simple and popular selection methods that use a greedy search technique. They are considered to have a computational advantage over other approaches [55] and so, if we wish to improve the efficiency of FSS algorithms, it is important to first compare TEDA's performance to these techniques. FS starts with an empty feature set and adds features one at a time, each time choosing the feature that leads to the greatest increase in classification accuracy. BS does the opposite. It starts with a complete feature set and repeatedly removes feature. Many features will be detrimental to classification accuracy and so the accuracy will actually improve when they are removed. In BS the single feature that leads to the greatest improvement in classification accuracy by its removal will be removed. As with FS, this is repeated until no further improvement can be achieved.

A version of FS that uses an alternative to the greedy search techniques, and therefore does not require as many iterations, first orders the features by usefulness and then use the top n features as the feature set. A decision then needs to be made as to where to draw the line between features used in the feature set and features that are to be discarded. One method

that has been discussed before [121] is to introduce a random dummy feature or *probe* to the feature set and to discard any features that perform less well at the classification task than the probe.

7.2.2 *Filter Methods*

For large classification tasks filter methods are often used instead of wrapper methods due to computational cost of repeatedly applying a classifier to large subsets of features [139] and so TEDA will also be compared against two filter techniques. The techniques that have been used are the Fast Correlatation Based Filter (FCBF) [139] and Correlated Feature Selection (CFS) [57]. FCBF is selected as an example of a filter method that has been successfully applied to a large feature set with 650 features in a study by Yu et al [139]. It is also especially suited to finding small feature sets in high dimensional and noisy problems as it explicitly removes redundant features. CFS is another technique which was tested alongside FCBF in Yu's study of large feature selection problems. It also considers redundancy when assessing the usefulness of features. As the ability to find small feature sets in high dimensional problems is expected to be a strength which TEDA will bring to feature selection, it will be interesting to see how TEDA compares to these techniques.

7.2.2.1 *Fast Correlatation Based Filter*

FCBF [139] is a technique that chooses features based on a correlation score. It begins by ranking all features in the order of how closely they are correlated to the class. From this ordered list all features with a correlation score higher than a predefined threshold are used to obtain the feature set. Redundant features are then removed from this feature set. A feature F_1 that is more highly correlated to F_2 than it is to a class, class C, is judged to be redundant if F_2 is also correlated to class C, and to a greater extent than F_1 is. The correlation between two variables X and Y is obtained by calculating how much information is gained when the entropy of X given Y, (H(X|Y)) is calculated. This *information gain (*IG*)* is calculated through the equation below.

$$IG(X|Y) = H(X) - H(X|Y) \tag{7.1}$$

The information gain is then made symmetrical to correct for the bias towards features with more values through the *symmetrical uncertainty* equation which is given below.

$$SU(X,Y) = 2\left[\frac{IG(X|Y)}{H(X)+H(Y)}\right] \tag{7.2}$$

7.2.2.2 *Correlated Feature Selection*

In CFS [57] features are ordered by how closely they are correlated to the class. How closely features correlate with each other is also calculated to ensure against redundant features. A best first search is then used whereby features are iteratively added to the feature set.

7.3 EXPERIMENTS AND RESULTS

In this section TEDA was tested on three large and noisy classification problems. TEDA is compared to FDC, a standard GA and EDAs as well as with FS and Filter methods. As with the other problems, the two versions of UMDA discussed in Section 5.5.2 were used as the benchmark EDAs against which TEDA was compared. For the Dexter problem we also compared TEDA to non evolutionary state of the art FSS Methods. These included FS and two filter methods.

7.3.1 *Experimental Method*

In this problem solutions are encoded as a fixed length binary string with each gene representing a possible feature. Where a gene has a value of true this feature will be used in the solution. As with the other problems, the test parameters outlined in Section 4.2 were used. Parameters specific to this problem are provided in Table 7.1.

Table 7.1: Feature Subset Selection Parameters

Parameter	Value
penalty	$10/l$
K for K-Nearest Neighbour	3

7.3.1.1 *Mutation*

There has been one slight change to the mutation process. In the chemotherapy and the routing problem mutation was attempted for each solution a number of times equal to the total length of the chromosome, equivalent to the maximum possible number of control points (C_{max}). In this problem mutation was attempted for each solution a number of times equal to the number of features actually used in the solution. With each attempt mutation is carried out with a proability of 0.05, the same as in the other problems.

This is because in these problems the maximum size of a solution C_{max} is much larger than in the routing and chemotherapy problems and the optimal size of a solution C_{opt} may

be much smaller in proportion to the maximum size. The Arcene problem, for example, has a maximum chromosome size of 10,000 whereas some optimal solutions contained only 10 features. If mutation is attempted on a solution C_{max} times where C_{max} is 10,000 and where the solution contains only 10 features then the effect will be too strong. With a mutation rate of 0.05, on average around $10,000 \times 0.05 = 500$ features will be added to a small solution, thus increasing its size by 5000% and completely changing it.

This technique could be applied in any situation, regardless of the length of solutions and C_{max}, although it was found empirically that mutating based on maximum length causes slightly better performance in smaller problems and so we would advise to mutate based on maximum length in all cases except for very large problems.

7.3.1.2 *Initialization*

As with the routing problem, chromosomes were initialized by intially choosing a random number n from the range 0 to l, where l is the size of the total available feature set. n features are then selected at random and added to the chromosome.

7.3.2 *The Datasets*

The datasets used were taken from the NIPS 2003 feature selection challenge [56]. The only preprocessing and data formatting steps applied to the datasets were those described in [56].

For the three datasets that we have used, Table 7.2 provides the domain, type, training set size and the test set size. All datasets represent binary classification problems and the bracketed values show the number of positive samples in each case. All information is from [56] except for the numbers of positive samples which can be found in [43]. Madelon is an artificial dataset designed to feature a high level of interdependency between features, and so by using it we are able to demonstrate how well TEDA performs in a highly multivariate environment [56].

7.3.3 *The Fitness Function*

The basis for the fitness function is an accuracy score, a. A penalty is subtracted from this to reflect the fact that smaller numbers of features are preferable. Given an accuracy value of a, a feature set of size l and a penalty of p, the fitness function f is calculated as $f = a - lp$. The penalty used in these tests is $10/l$. For each problem, two sets of tests were run using two different methods for obtaining a. In the first set a is a simple accuracy score, the percentage of samples that were correctly classified. In the second set the Balanced Error Rate (BER) was used. LIBSVM, A Support Vector Machine produced by Chang [28] is used as the classifier

Table 7.2: Feature Selection Datasets

Name	Domain	Type	Feat.
Arcene	Mass Spectrometry	Dense	10000
Dexter	Text classification	Sparse	20000
Madelon	Artificial	Dense	500
Name	Train(pos.)	Test(pos.)	
Arcene	100(44)	100(44)	
Dexter	300(150)	300(150)	
Madelon	2000(1000)	600(300)	

with all parameters kept at their default values. Some tests are also carried out using the K-Nearest Neighbour (K-NN) Classifier with a K of 3.

7.3.3.1 *The BER*

The BER is the mean of the error rate for each class. For binary classification problems, such as those used in this thesis, it is calculated through Equation 7.3 [31]. For this equation we denote the two classes as class a and not class a ($\not a$). The error rate for each class is given as the percentage of samples within that class that are incorrectly classified. The fitness function used in this chapter for tests that use the BER is $100 - \mathrm{BER}$, so that the function becomes a maximisation function to match the other tests in this chapter.

$$\mathrm{BER} = \frac{(\text{percent wrongly classified in class } a + \text{percent wrongly classified in class } \not a)}{2} \quad (7.3)$$

This method provides a more accurate impression of the classification accuracy in situations where the dataset is severely unbalanced than simply using the percentage of total samples correctly classified. For example, if the number of samples in class a in a dataset is 100 and the number of samples in class $\not a$ is only 5, the simple accuracy percentage will be determined primarily by how many samples in a are correctly classified. In this situation the number of samples in $\not a$ that are correctly classified, although just as important as the those in a, will have little impact on the final accuracy score. For this reason, the BER was one of the required metrics used in the NIPS FSS challenge [56], from which the datasets that we have used were drawn.

The following section shows the results for 50 runs of each algorithm on each of the three problems. For each problem three graphs are provided, showing the following metrics:

- The accuracy achieved by the fittest individual in the population on the y axis against the number of generations on the x axis. This accuracy given is the value a that is used in the fitness function. This will be either the simple accuracy or BER, as explained in the previous section.
- The number of features used by the fittest individual in the population on the y axis against the number of fitness function evaluations on the x axis.
- The accuracy, a, achieved by the fittest individual in the population on the y axis against classification time on the x axis. This is the mean of the times that each solution in the population took to complete the classification task. This is important as classification can be time consuming for these large problems that use a lot of features. The mean for each population at each generation is then added to a running total and it is this total that is plotted. From this data we also present, in tabular form, the length of time that each algorithm took to reach a given accuracy level.

As with the other chapters, for each graph the value plotted is the median of the 50 runs with first and third quartiles given by the variance bars.

7.4.1 *Dexter*

In this section we explore how TEDA compared to other approaches on solving the Dexter problem.

7.4.1.1 *TEDA compared to Forward and Backward Selection*

In Figure 7.2 we present the performance of TEDA on the Dexter problem compared to forward selection using the greedy iterative search method. In Figure 7.3 we compare TEDA to FS using a probe. This graph shows the fitness of the fittest solution obtained so far against the number of fitness function evaluations completed. In the case of FS an evaluation is carried out every time that the quality of a feature is assessed as this involves adding it to the feature set and then scoring the new feature set by attempting to classify with it using the LIBSVM classifier. Although the greedy search method is deterministic the probe method is stochastic as the probe itself is randomly generated. For this reason the fitness presented for the probe method is the median of 50 runs.

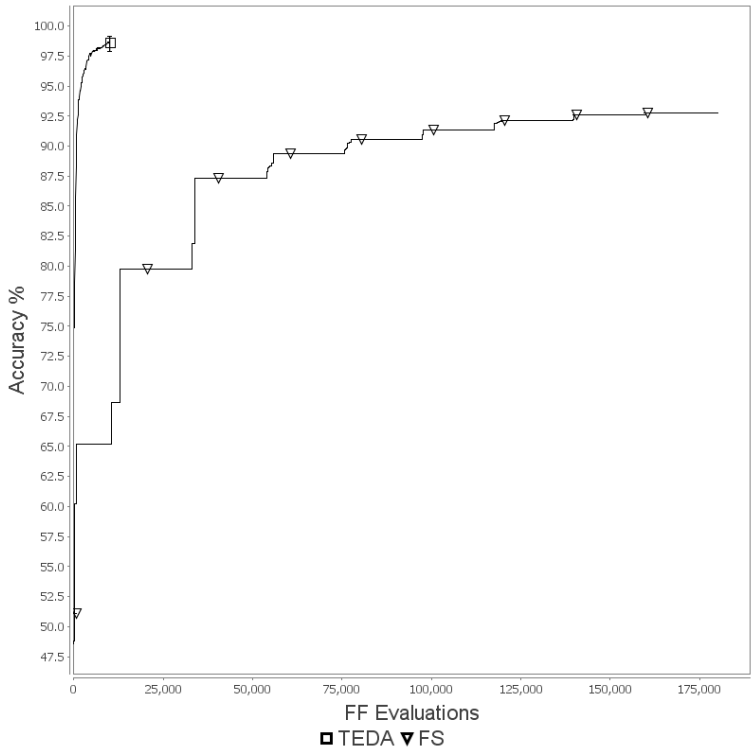

Figure 7.2: Dexter - TEDA vs Forward Selection without Probe - Accuracy against Fitness Function Evaluations

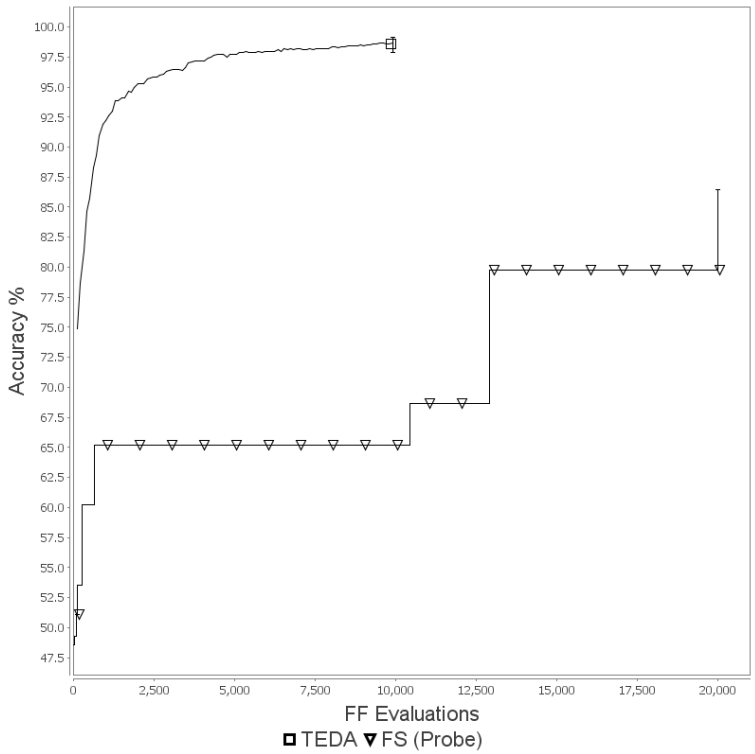

Figure 7.3: Dexter - TEDA vs Forward Selection with Probe - Accuracy against Fitness Function Evaluations

As we can see, FS takes many more fitness function evaluations than TEDA to achieve comparable fitness. Using a probe the total length of time that the algorithm takes to complete is lower, only 20,000 evaluations instead of 175,000, and yet this is still significantly more than the 10,000 evaluations that TEDA was allowed to run for and the fitness score ultimately found is lower in both cases. This is unsurprising due to the size of the problem. Forward selection and backward selection using a greedy search both have a worst case complexity of $O(n^2/2)$. This is the complexity in the situation where the number of features selected, n, is equal to the maximum size of the feature set, l, for forward selection or equal to 1 for backward selection. In general the complexity is $O(l + (l-1) + (l-2) + ... + (l-n'+1))$ [133]. If a maximum feature set size is specified, at which point the search should stop then n' is equal to that size. If, on the other hand, the algorithm is following the strategy of adding new features until no further improvement is observed then $n' = n + 1$ where n is the size of the final feature set discovered as an additional iteration is required to establish that no further improvement can be achieved. This is the strategy that is followed in these tests.

Where the number of features is 20,000, as in this problem, this means that the maximum number of evaluations that it may take is $20,000^2/2$ or 2×10^8. Even in the best case situation, where the ideal feature set contains only one feature, the number of evaluations required is 39,999. By comparison, when an EA with a population of size 100 is run for 100 generations, as is the case in the tests in this chapter, this amounts to only 10,000 generations. When a probe is used each feature only needs to be evaluated once and so just 20,000 evaluations are needed. FS is a robust method that is effective at finding good solutions on small problems and is less prone to overfitting than wrapper methods such as TEDA [55] but as this discussion demonstrates for large problems it may be less efficient than wrapper approaches such as EAs.

It should be noted, however, that comparing these algorithms by the number of evaluations taken alone does not provide a fair comparison as the amount of time taken to evaluate a feature set consisting of just one feature is likely to be much less than the time taken to evaluate a large feature set. For the first 20,000 evaluations FS is only assessing feature sets that contain just one feature. For this reason, we present two additional graphs in which are plotted the total time taken, the sum of the times taken by each evaluation, against fitness. Figure 7.4 shows the time taken by the greedy search variant of FS and Figure 7.5 shows the same metric for the probe based approach. In these graphs we can see that, although the difference between TEDA and FS is less pronounced, both variants of FS still perform worse than TEDA.

BS is not tested as it will take much longer than FS. As we shall see, the most effective solutions that TEDA is able to discover usually contain less than 1000 features. To discover solutions of this size starting with the entire feature set of 20,000 BS would take $(19,000^2/2) + 1000^2$ evaluations. Considering that, in contrast to the single feature feature sets evaluated in FS, BS would involve evaluating a great number of very large feature sets it would clearly be

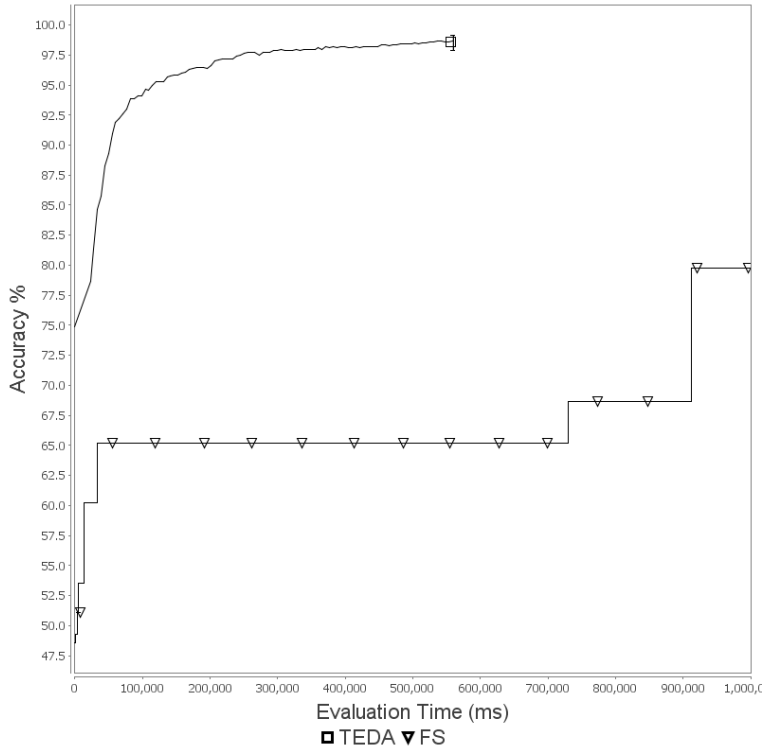

Figure 7.4: Dexter - TEDA vs Forward Selection without Probe - Accuracy against Classification Time

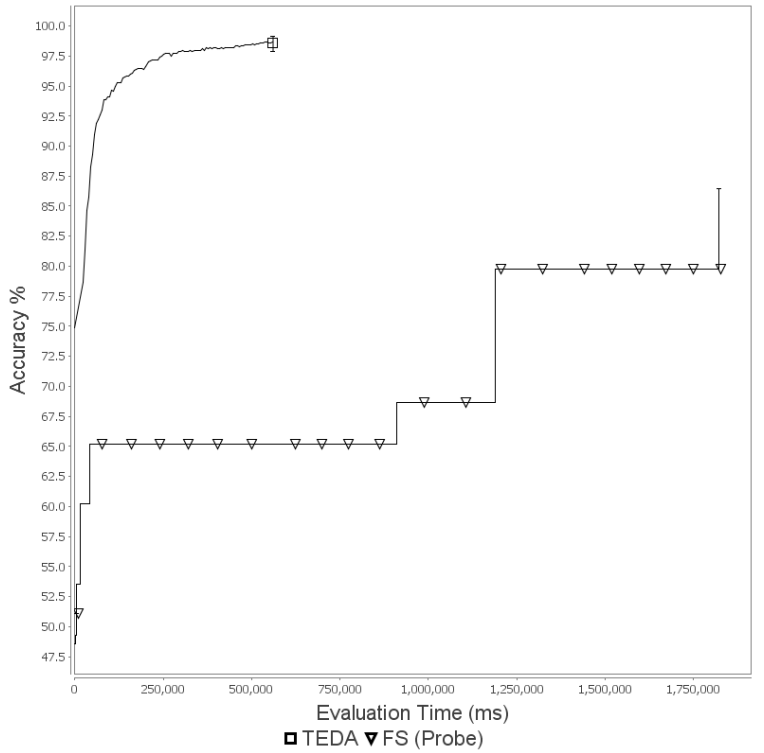

Figure 7.5: Dexter - TEDA vs Forward Selection with Probe - Accuracy against Classification Time

prohibitively expensive. The performance of this technique will therefore by even worse than FS.

7.4.1.2 *TEDA compared to Filter Methods*

In this section TEDA is compared against the state of the art filter techniques described at the start of this chapter, FCBF and CFS. All of these approaches were implemented using the WEKA machine learning environment [136]. Unless otherwise stated, the parameters used for each algorithm were the default parameters used in the WEKA environment. Three different classifiers were used. LIBSVM was used as it is the classifier on which all algorithms in this chapter have been tested. However, as it appears as though FCBF does not perform well with LIBSVM and to ensure that the performance of every technique is fairly represented we have also included the two classifiers on which FCBF was initially tested when it was developed [139]. These classifiers, also implemented in the WEKA environment, are the Naive Bayes Classifier [136] and the C4.5 [105] Classifier.

Table 7.3 shows the fitness score obtained by these algorithms using each of the classifiers. As these are algorithms that run to completion, unlike EAs that can run for an arbitrary length of time, the accuracy given is that obtained after each algorithm is allowed to complete. The size of feature set obtained is also given.

Table 7.3: Dexter - Filter Method Performance

Algorithm	Feature Set Size	Accuracy with LibSVM	Accuracy with C4.5
FCBF	35	53.33	82.33
CFS	7751	85.33	80.67

Algorithm	Accuracy with Naive Bayes	Feature Set Size
FCBF	83.67	382
CFS	90.67	7751

For comparison the median best feature set found after 100 generations when using TEDA over LIBSVM had an accuracy of 99%. None of these techniques are therefore able to find solutions that are as accurate as those obtained by TEDA. It should be noted that in the case of the EAs tested we used a population size of 100 and ran the tests for 100 generations and so this equates to 10,000 fitness function evaluations whereas by definition, in filter methods the classifier is applied only once. Even though a good fitness is in fact achieved by TEDA long before the 100th generation (see Figure 7.2) filter methods are probably able to find features with an accuracy of around 80%-90% before most EAs, including TEDA. However, without integrating another approach neither of these techniques has any mechanism to improve

further on these feature sets. TEDA is therefore able to find more accurate feature sets than either of these filter methods.

In terms of size the best feature sets found by TEDA had a median size of 1536 features. This is larger than the feature set found by FCBF but smaller than that found by CFS.

7.4.1.3 *TEDA compared to EDAs and GAs - Using Accuracy*

We now move on to comparing the performance of TEDA against alternative EAs. The results in Figure 7.6 show the accuracy achieved by TEDA and the other algorithms against number of evaluations when the accuracy measured is the percent of samples correctly classified.

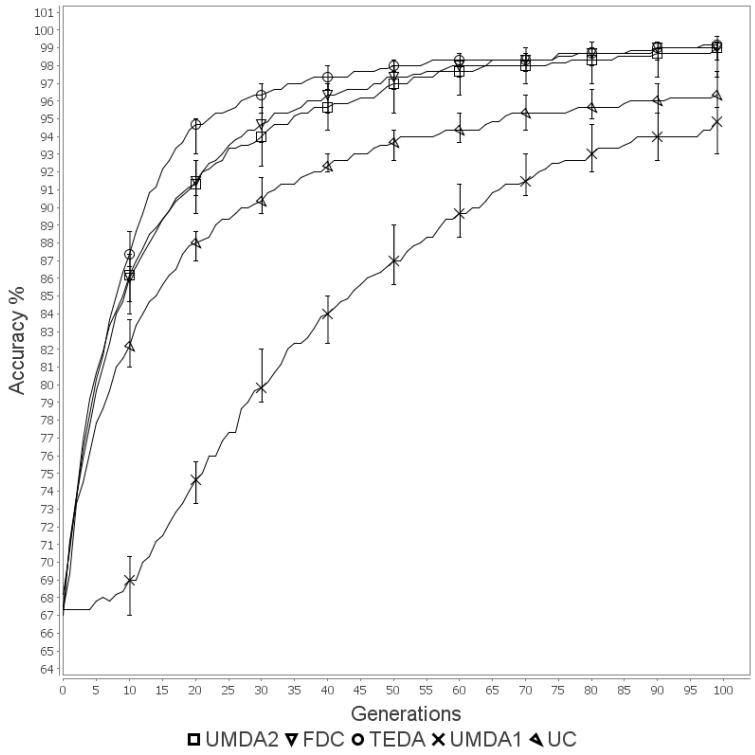

Figure 7.6: Dexter - Accuracy against Fitness Function Evaluations

TEDA is consistently able to find better solutions than any of the other techniques up until at least the 50th or 60th generation when UMDA2 and FDC start to find comparable solutions. UMDA1 performs considerably worse than any other technique throughout the test. Next we look at the size of feature set. The results in Figure 7.7 demonstrate that the algorithms that are most effective at finding accurate feature sets also tend to be more effective at finding smaller feature sets. The exception is FDC, which finds feature sets that are of an accuracy similar to those found by UMDA2 but tend to be smaller, the same size as those found by TEDA.

The results in Figure 7.8 shows us that when we compare performance against time rather than against number of evaluations the margin of difference between TEDA and UMDA2, the GA and UMDA1 is slightly greater. This is because the feature sets that TEDA finds are smaller and so quicker to evaluate. Generations are therefore completed in less time. KW analysis, as

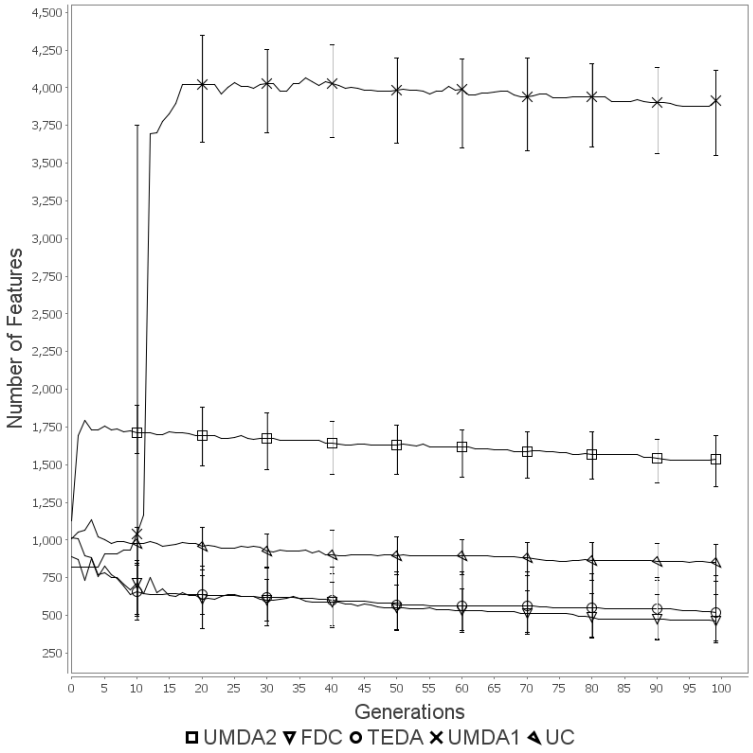

Figure 7.7: Dexter - Control Points (Features)

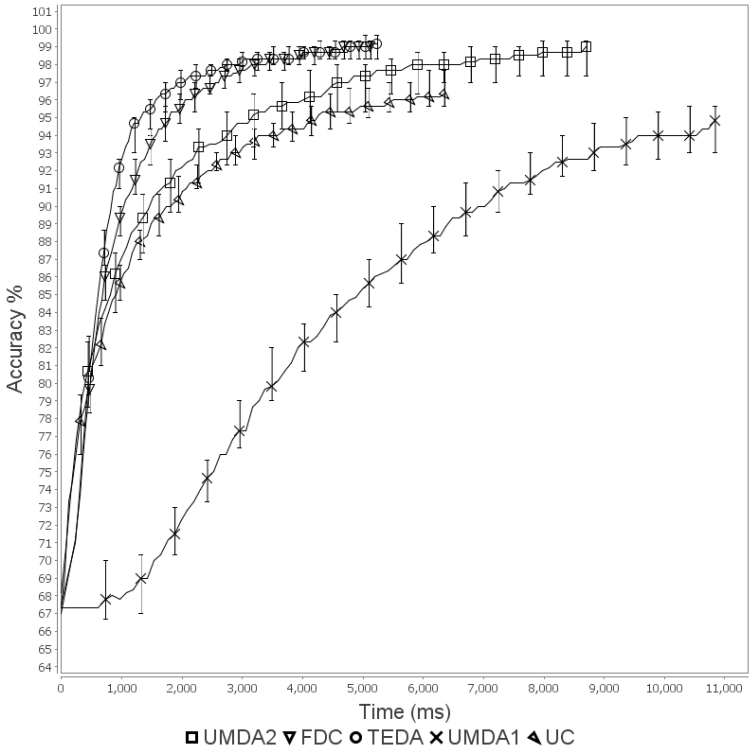

Figure 7.8: Dexter - Accuracy against Classification Time

Table 7.4: Dexter - Evaluation Times

Accuracy (%)	UMDA2 (ms)	FDC (ms)	TEDA (ms)	UMDA1 (ms)	UC (ms)
80.0	0.44	0.51	0.45	3.59	0.51
84.0	0.72	0.62	0.6	4.56	0.78
88.0	1.16	0.87	0.75	6.06	1.3
92.0	1.9	1.27	0.95	7.98	2.44
96.0	4.02	2.06	1.57	-	5.51
100.0	-	-	-	-	-

Table 7.5: Dexter - Kruskal Wallis for Evaluation Times

Accuracy (%)	TEDA vs UMDA2	TEDA vs FDC	TEDA vs UMDA1	TEDA vs UC
80.0	4.95	9.04	***138.28***	39.23
84.0	***50.46***	17.72	***162.12***	***80.5***
88.0	***83.03***	33.98	***181.84***	***110.65***
92.0	***91.04***	33.88	***185.04***	***118.54***
96.0	***89.8***	30.3	***179.86***	***117.24***
100.0	***117.48***	6.32	***161.52***	***63.52***
Threshold	40.6			

shown in Table 7.5, shows that for accuracies between 80% and 100% TEDA is faster than all other approaches except for FDC by a statistically significant margin. Table 7.4 shows how long each algorithm took to reach these accuracies.

It is interesting that it appears that this problem is unsuitable for a conventional implementation of UMDA. it might be the case that fit solutions can only be found once the size of the feature set has been substantially reduced and that all very large feature sets are similarly ineffective due to the high level of noise in this problem. In the initial population it is possible that some extremely small feature sets are generated by chance. Due to the penalty at least, these are likely to have a high fitness. In a conventional implementation of UMDA the large breeding pool is likely to obscure these solutions as they will have little effect on the marginal distribution. A GA might occasionally select such solutions as one of its two parents and when it does so it is likely to breed a smaller child solution. Whilst GAs might by chance produce new solutions of the same size as these small solutions, TEDA and FDC do this explicitly and in fact drive beyond the size of these solutions to find even smaller feature sets. This explains how FDC is able to finder shorter and more accurate solutions than a conventional GA.

UMDA2, which uses a smaller breeding pool and mutation like a GA, is able to overcome the noise that affects UMDA1 while also taking advantage of the ability of EDAs to exploit patterns within the population. For these reasons it is relatively effective at this problem. The exploitative ability of EDAs also explains why TEDA outperforms FDC.

7.4.1.4 *The Importance of Targeting*

As with the other problems, we will now explore whether it is the overall structure of TEDA, with its transitioning approach that is offering an improvement or whether targeting is necessary for this problem. The results in Figures 7.9 and 7.10 show the accuracy achieved by TEDA when compared to an identical algorithm that does not use targeting. Figure 7.9 shows the accuracy against evaluation count and Figure 7.10 shows the accuracy over time. The tests were run with the same parameters as the previous section and so accuracy was used as the basis of the fitness score.

From these graphs we can see that targeting is essential for TEDA to demonstrate the good performance that it does on this problem.

7.4.1.5 *The Transitioning Method*

In the previous chapter it was demonstrated that the transitioning technique that performed best for the chemotherapy problem was not the same as that which performed best for the routing problem (see Section 6.2.5). It was concluded that the transitioning technique that performed best for the routing problem, probabilistic transitioning, should be considered the standard transitioning approach in favour of the method that performed best for the chemotherapy problem, population variation based transitioning. This was because the

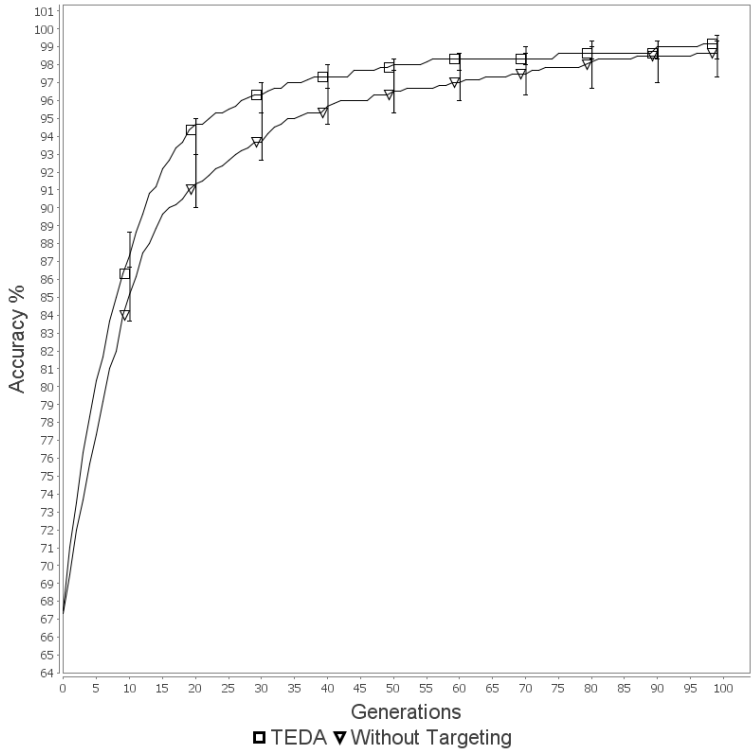

Figure 7.9: Dexter - TEDA without Targeting - Accuracy against Fitness Function Evaluations

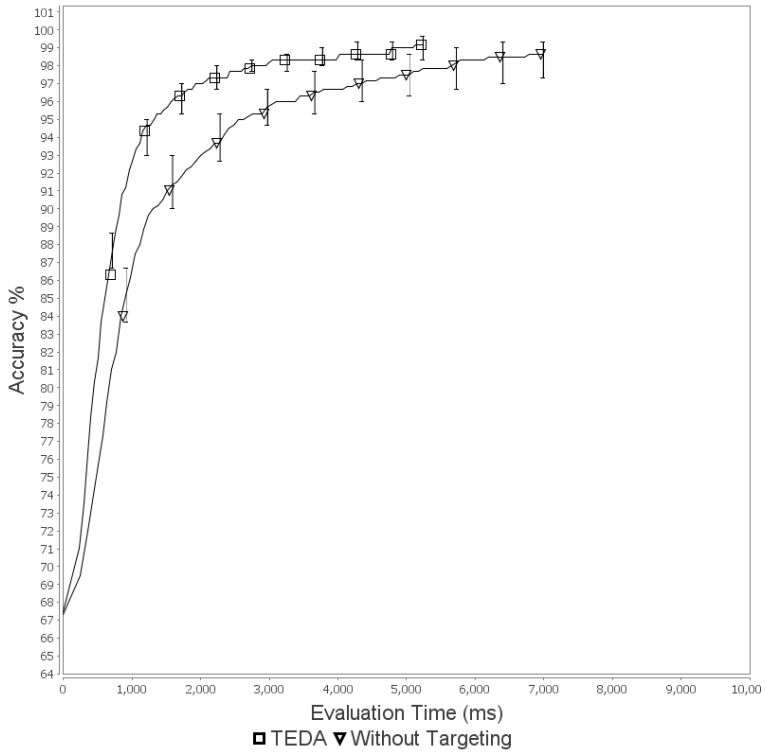

Figure 7.10: Dexter - TEDA without Targeting - Accuracy against Classification Time

chemotherapy problem was identified as a problem for which TEDA was not necessary anyway and both versions of TEDA did ultimately find optimal solutions for chemotherapy, albeit slightly slower than the other approaches in the case of probabilistic transitioning. Because of this, all tests since that conclusion have been run using probabilistic transitioning.

In this section we will confirm whether probabilistic transitioning is, in fact, the best approach in this problem. In the graphs below we present the comparison between TEDA using the probabilistic approach and TEDA using the population variation based approach. For comparison, these graphs also show the performance of FDC and UMDA2. This is because UMDA2 was demonstrated as the best performing alternative to TEDA and including FDC demonstrates whether both forms of transitioning actually lead TEDA to behave in any way that is different to its pre-transitioning state, FDC. The results in Figure 7.11 shows the accuracy over number of evaluations and Figure 7.12 shows the accuracy over time.

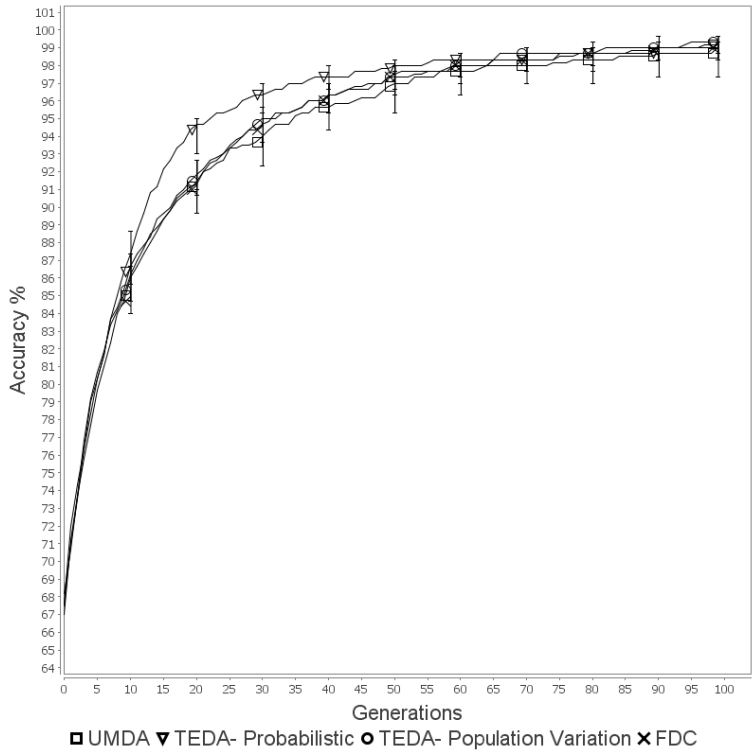

Figure 7.11: Dexter - Probabilistic vs Variation Based Transitioning - Accuracy against Fitness Function Evaluations

These results confirm that probabilistic transitioning is the better approach for this problem. TEDA using population variation based transitioning performs in an identical manner to FDC and finds fit solutions slightly faster than UMDA2. These results support our decision to identify the probabilistic method as the method to be used in a standard implementation of TEDA. They also show that neither method significantly harms the performance of TEDA and so if, based on its better performance on the chemotherapy problem, a user should decide to use the population variance based transitioning method this will not carry high risk.

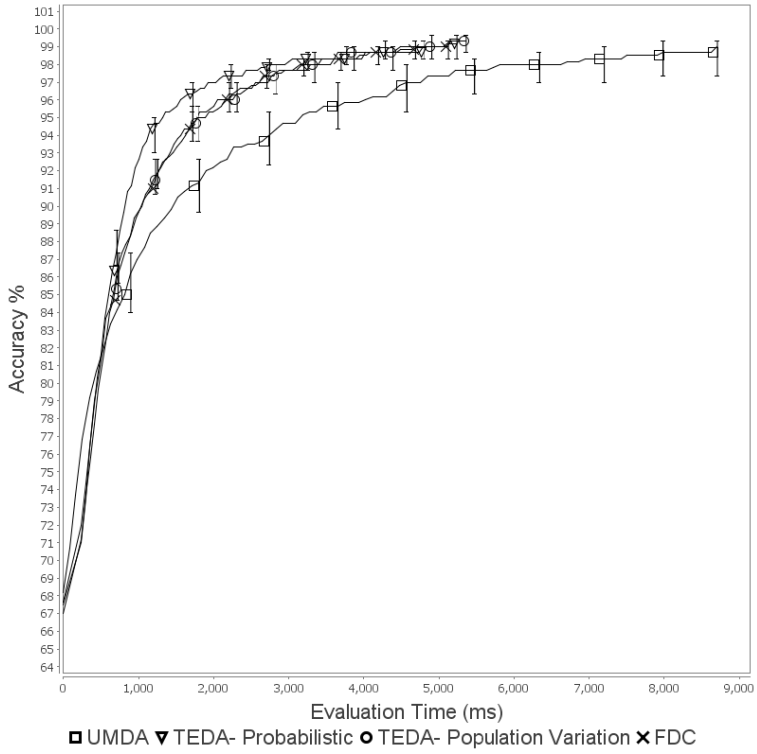

Figure 7.12: Dexter - Probabilistic vs Variation Based Transitioning - Accuracy against Classification Time

7.4.1.6 *TEDA Compared to EDAs and GAs - Using BER*

The graphs in this section show the results of running the same test when the Balanced Error Rate (BER) is used instead of the accuracy to obtain solution fitness. The results in Figure 7.13 shows how the BER improves from generation to generation, Figure 7.14 shows how the size of feature set changes and Figure 7.15 shows how the BER changes over time. The pattern is similar to when accuracy is used. TEDA appears to find good solutions in less time than the other approaches. However, it appears that the improvement that TEDA offers over the other approaches is less significant than when a simple accuracy score is used.

7.4.1.7 *Tests with KNN*

Figures 7.16, 7.17, and 7.18 show how TEDA performs when compared to the other approaches in tests in which the K-Nearest Neighbour (K-NN) was used as the classifier. K-NN is used as a second classifier to confirm whether TEDA's performance is sensitive to the classifier used. In these tests the fitness score was based on the accuracy, not the BER.

From these graphs it appears as though, when the K-NN Classifier is used, UMDA2 is more effective than TEDA, finding good soutions quicker. However, Figures 7.19, 7.20 and 7.21 show the comparison between SVM and K-NN. The techniques that performed best on both approaches, TEDA and UMDA2, are presented in these graphs using both classifiers. The main

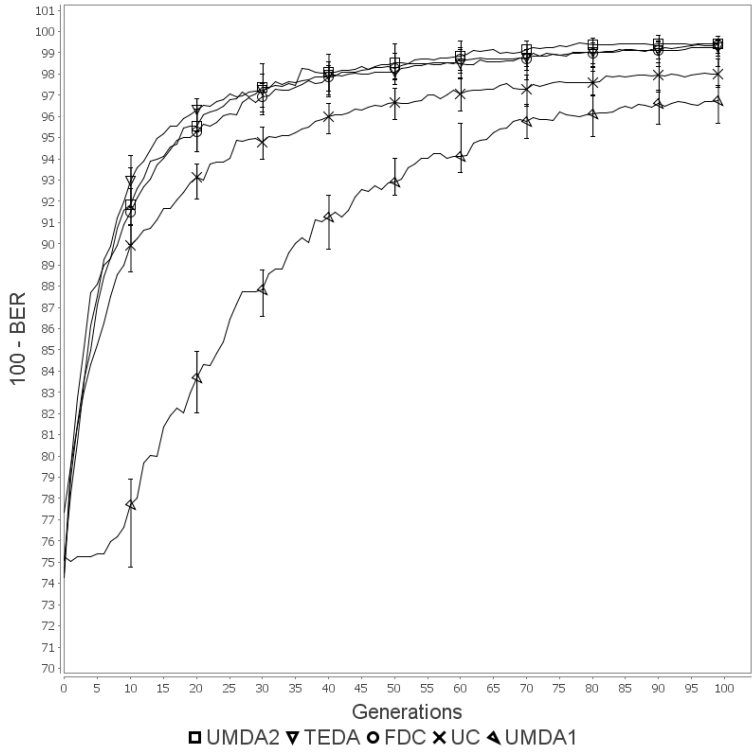

Figure 7.13: Dexter - BER against Fitness Function Evaluations

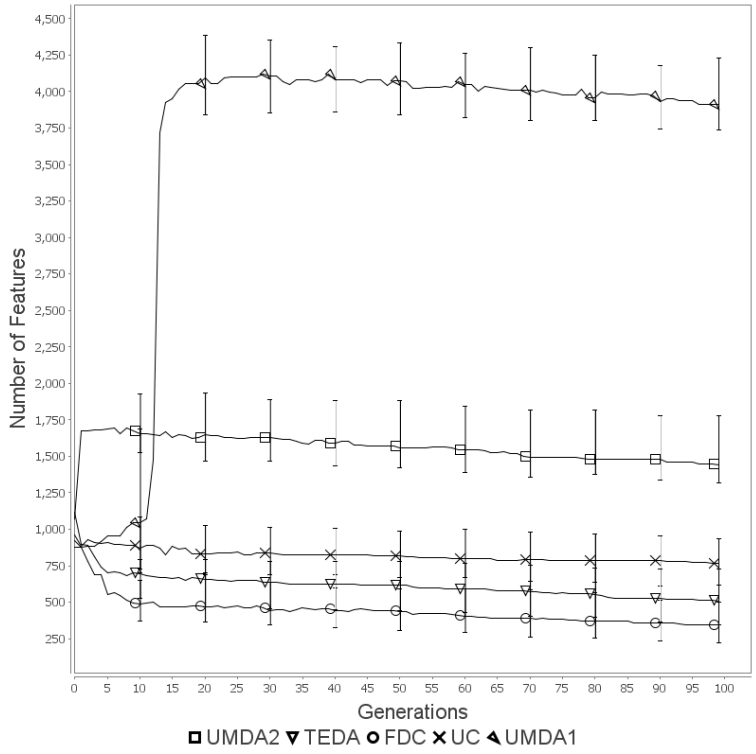

Figure 7.14: Dexter - Control Points (Features) when using BER

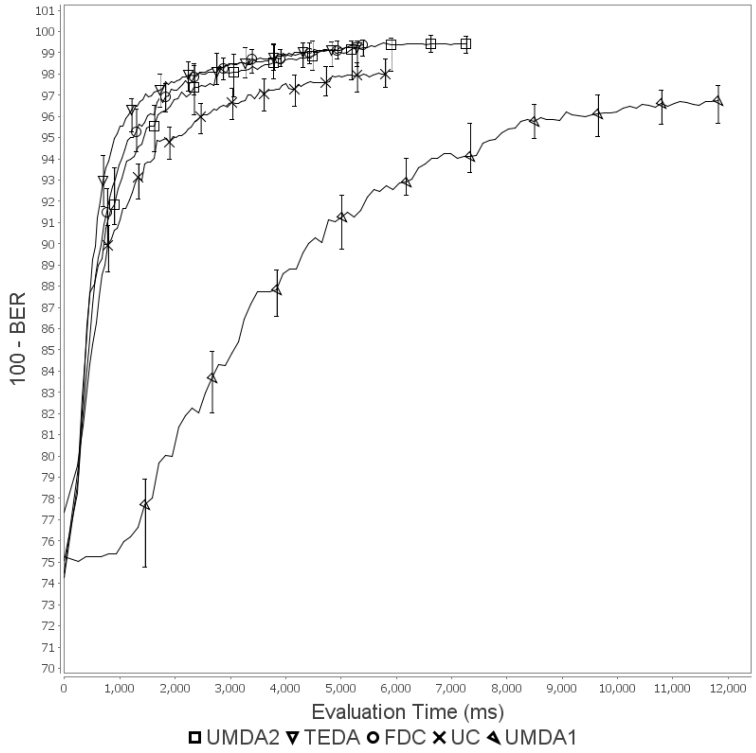

Figure 7.15: Dexter - BER against Classification Time

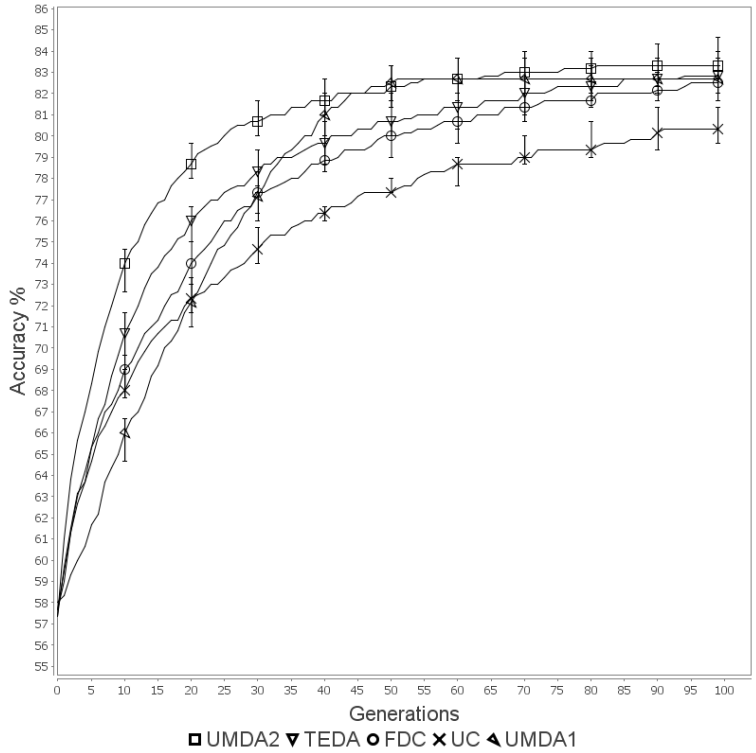

Figure 7.16: Dexter - Accuracy against Fitness Function Evaluations using KNN Classifier

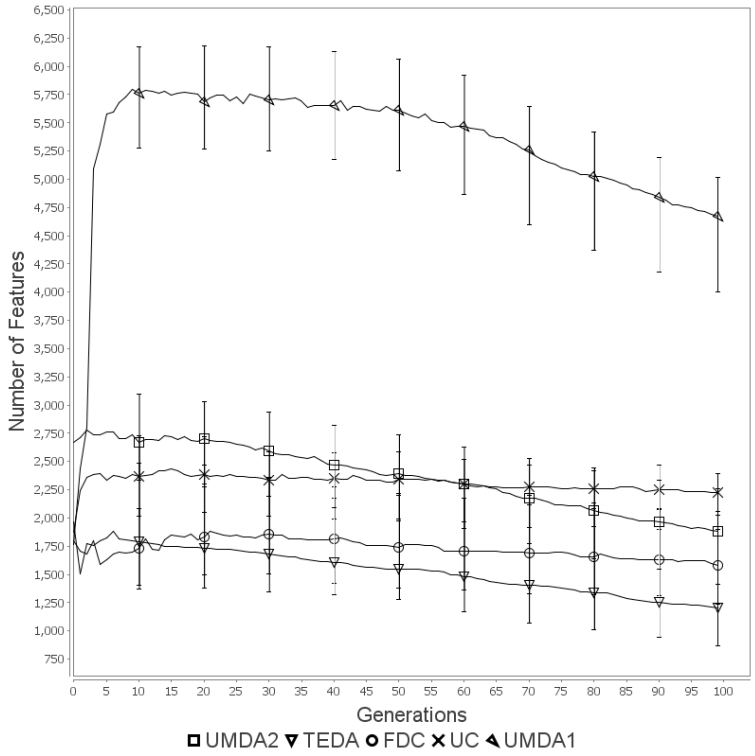

Figure 7.17: Dexter - Control Points (Features) using KNN Classifier

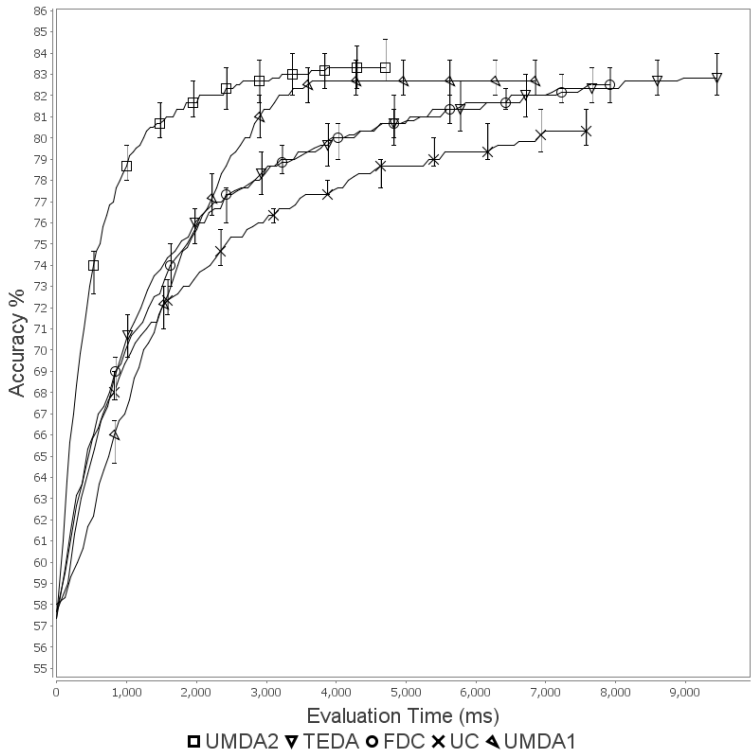

Figure 7.18: Dexter - Accuracy against Classification Time using KNN Classifier

observation to make from this comparison is that, due to the fact that with both algorithms SVM performs much better than K-NN, overall a combination of SVM and TEDA is the best combined approach that we have seen.

When we look at the numbers of features used (Figure 7.20), it appears as though smaller feature sets are found using the SVM method. K-NN appears to require larger feature sets and to be less effective with these feature sets. On this problem it is a less efficient classifier than SVM. Because K-NN does not work with small feature sets the targeting aspect of TEDA appears to be detrimental in this case, with TEDA over aggressively driving towards small feature sets that are too small to be effective with K-NN. In terms of this problem it is possible to say that TEDA is less effective on classifiers requiring large feature sets but also that classifiers requiring small feature sets are more effective anyway. Further testing with a range of cassifiers and problems woud be needed to indicate whether this generalises.

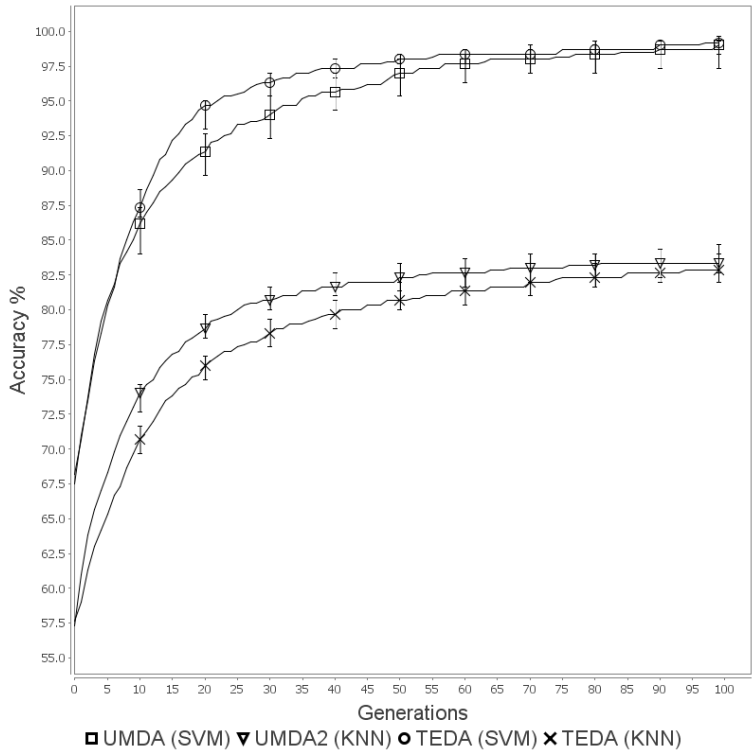

Figure 7.19: Dexter - KNN and SVM Compared - Accuracy against Fitness Function Evaluations

7.4.2 *Arcene*

7.4.2.1 *Accuracy tests*

We see from the results in Figure 7.22 that in Arcene TEDA and FDC is able to find better solutions earlier on than the other approaches. UMDA2 and UC both start to perform slightly better than TEDA from around generation 25 onwards but for the first 10 generations they are

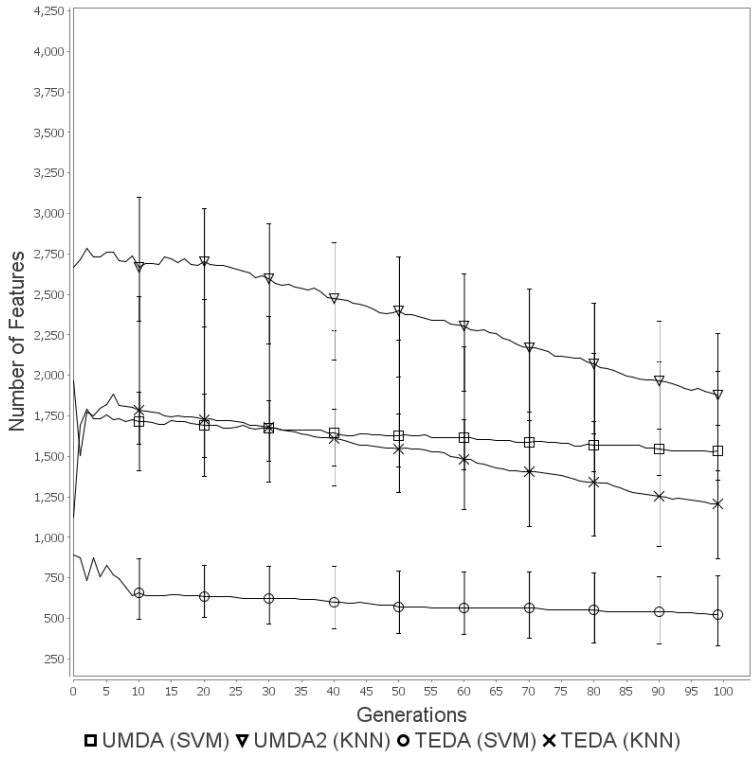

Figure 7.20: Dexter - KNN and SVM Compared - Control Points (Features)

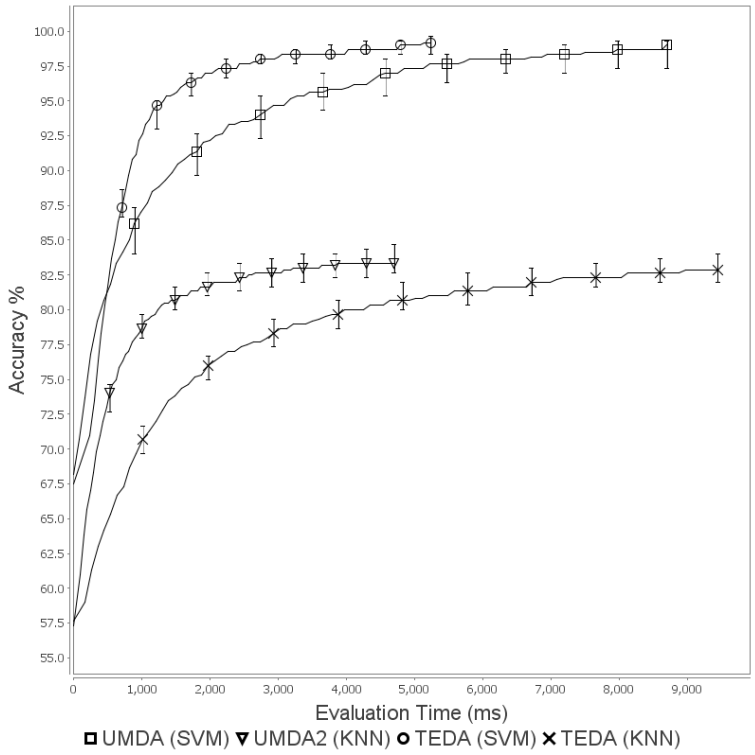

Figure 7.21: Dexter - KNN and SVM Compared - Accuracy against Classification Time

completely unable to improve upon the fittest individual in the initial population. UMDA1 is only able to start improving after about generation 70.

By looking at the number of features used in Figure 7.23 we can see that, for all untargeted approaches, the fittest solution in the initial population has a median size of around 70 to 80 for a period of time matching the period in which the algorithm does not improve upon the fitness of this individual. This individual is considerably smaller than the maximum feature set size for Arcene, 10,000 features. It would be expected that the sizes of solutions in the randomly generated initial population is evenly distributed across the range 1 to 10,000. In the case of the EDAs, this extremely small individual would, as discussed in the context of Dexter, be effectively invisible to the probability model. This would be especially true for UMDA1 where the model is built from the top 50% of the population. It seems as though the situation is the same for both Arcene and Dexter. It appears to be the case that, due to high levels of noise, until an algorithm starts to explore smaller solutions all solutions are equally ineffective. As with Dexter, a GA might by chance select a small short solution and breed a new, similarly sized solution and UMDA2 uses a small enough pool of parents to behave in a manner not that dissimilar to the GA. TEDA, however, accelerates this process by making it explicit.

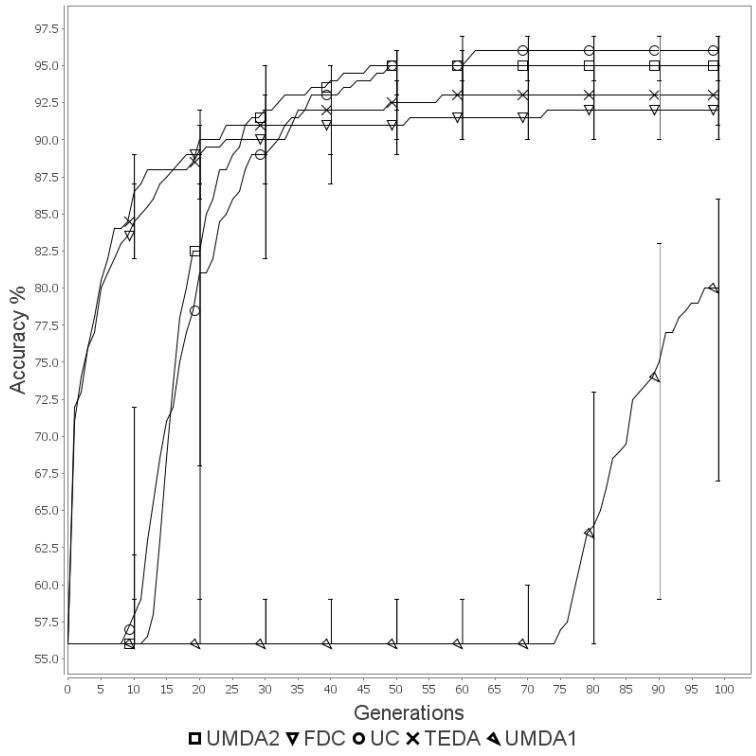

Figure 7.22: Arcene - Accuracy against Fitness Function Evaluations

As with Dexter, these small solutions can be classified more efficiently than larger solutions and so, when plotted against time, we see that TEDA and FDC have almost completed a 100 generation run before UMDA2 and the GA start to discover effective solutions (Figure 7.24).

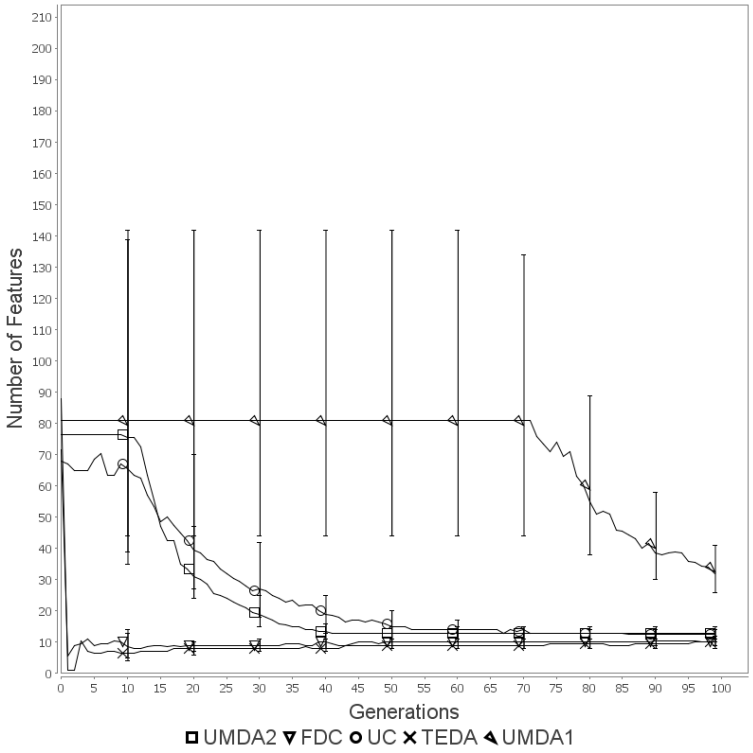

Figure 7.23: Arcene - Control Points (Features)

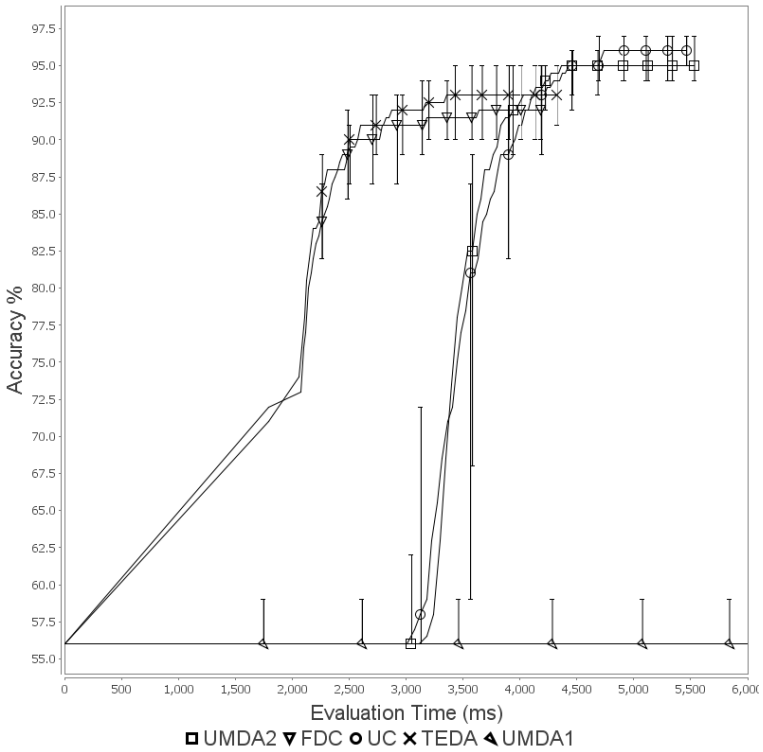

Figure 7.24: Arcene - Accuracy against Classification Time

Table 7.6: Arcene - Evaluation Times

Accuracy (%)	UMDA2 (ms)	FDC (ms)	UC (ms)	TEDA (ms)	UMDA1 (ms)
80.0	3.49	2.14	3.56	2.12	26.78
84.0	3.62	2.25	3.67	2.18	-
88.0	3.69	2.39	3.8	2.31	-
92.0	3.93	3.64	4.04	2.87	-
96.0	-	-	4.74	-	-
100.0	-	-	-	-	-

Table 7.7: Arcene - Kruskal Wallis for Evaluation Times

Accuracy (%)	TEDA vs UMDA2	TEDA vs FDC	TEDA vs UC	TEDA vs UMDA1
80.0	***87.74***	1.88	***94.14***	***170.0***
84.0	***86.5***	1.58	***94.24***	***170.58***
88.0	***77.22***	8.52	***87.38***	***168.28***
92.0	***48.02***	4.38	***57.44***	***152.46***
96.0	39.52	17.4	37.24	***139.84***
Threshold	40.6			

The KW analysis in Table 7.7 shows that TEDA is faster to a statistically significant extent than all algorithms except for FDC at reaching accuracies between 80% and 92%. Table 7.6 shows the time that each algorithm took to reach these accuracy levels.

It should be noted that TEDA later on performs slightly worse than UMDA2 and UC. This may be due to TEDA losing a certain amount of diversity early on as it is overly aggressive with feature targeting. The ineffectiveness of large solutions may create a situation where, in the initial population, the penalty penalising large feature sets is the only factor differentiating different solutions and so TEDA and FDC push towards the smallest feature sets regardless of quality. For this reason, the populations at generations 2 and 3 in FDC and TEDA consist entirely of feature sets with only one or two features in them. The fact that after this TEDA and FDC are able to recover towards producing solutions with between 10 and 20 features and are able to eventually find solutions with an accuracy of over 90% shows that these techniques have some ability to self correct, although this should be investigated further.

7.4.2.2 *Balanced Error Rate*

There appears to be little difference between how the different approaches perform when the BER is plotted against generations, except for UMDA1 which performs noticeably worse. As with the previous set of results, there is a possibility that TEDA and FDC find good results faster than the untargeted approaches while the untargeted approaches ultimately find slightly better results, although differences are slight. However, from the results for fitness over time (Figure 7.27) it is clear that FDC and TEDA are both able to reach good solutions ahead of the other approaches.

7.4.3 *Madelon*

As we can see in Figure 7.28, in the Madelon problem both TEDA and UMDA2 find good feature sets quicker than the other techniques but UMDA1 later on overtakes both techniques. Both FDC and the standard GA are less effective.

UMDA1 is much more effective at this problem than at the other problems possibly because the need to dramatically reduce the size of feature set seen in the other problems does not apply in this case. There is less noise and so feature sets that use a large proportion of the available features can be very effective.

To confirm this, Figure 7.29 shows no steep declines or sudden drops in feature set size as seen in the other problems although the basic pattern, whereby TEDA and FDC show the greatest reduction in the size of feature set and UMDA1 shows the least reduction remains the same.

Despite it not reducing the feature set size as fast or as far as in the other problems, in Figure 7.30 we see that, plotted against time, TEDA is able to find better solutions than the

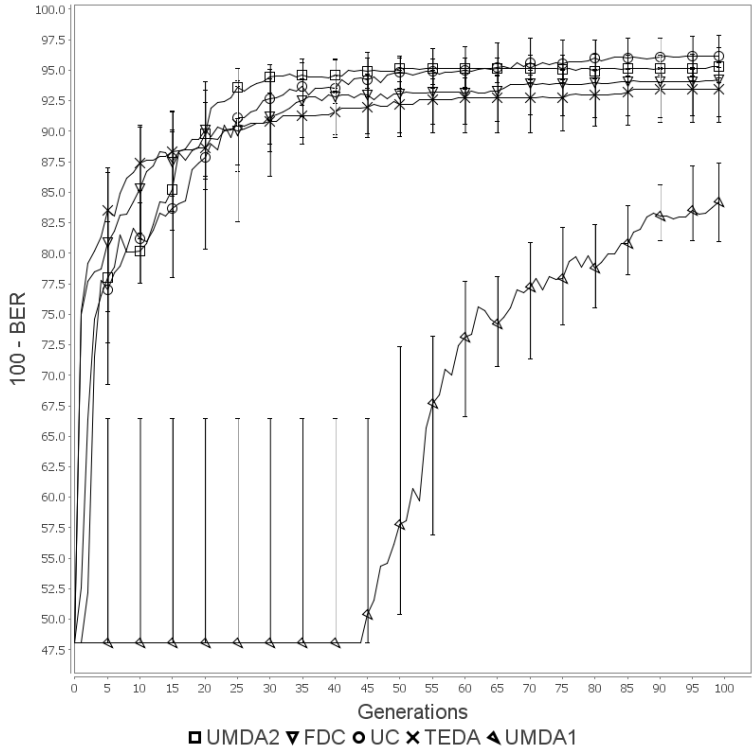

Figure 7.25: Arcene - BER against Fitness Function Evaluations

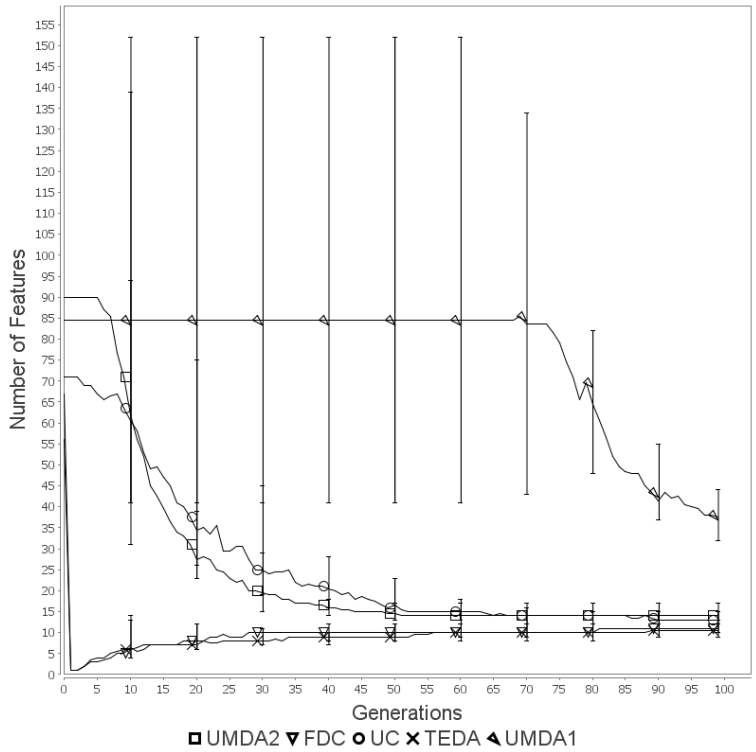

Figure 7.26: Arcene - Control Points (Features) when using BER

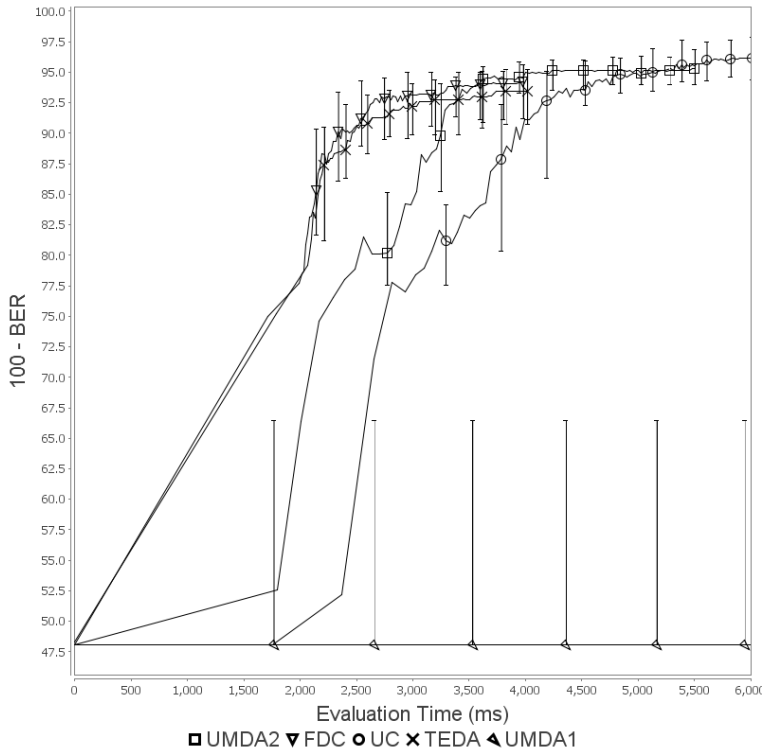

Figure 7.27: Arcene - BER against Classification Time

other techniques for almost the entire duration of its run. After this the other approaches reach the same level of fitness and UMDA1 ultimately finds fitter solutions. KW analysis, as shown in Table 7.9, confirms that for accuracies up to 88%, TEDA is faster than all of the other approaches by a statistically significant margin. Unlike in Arcene and Dexter, this includes FDC. Table 7.8 shows the time that each algorithm took to reach these accuracy levels.

Table 7.8: Madelon - Evaluation Times

Accuracy (%)	UMDA2 (ms)	FDC (ms)	TEDA (ms)	UMDA1 (ms)	UC (ms)
80.0	86.84	80.55	60.72	124.12	118.57
84.0	148.65	130.2	97.86	208.74	218.72
88.0	273.71	319.76	191.76	336.7	436.19
92.0	578.33	-	-	512.68	-
96.0	-	-	-	-	-
100.0	-	-	-	-	-

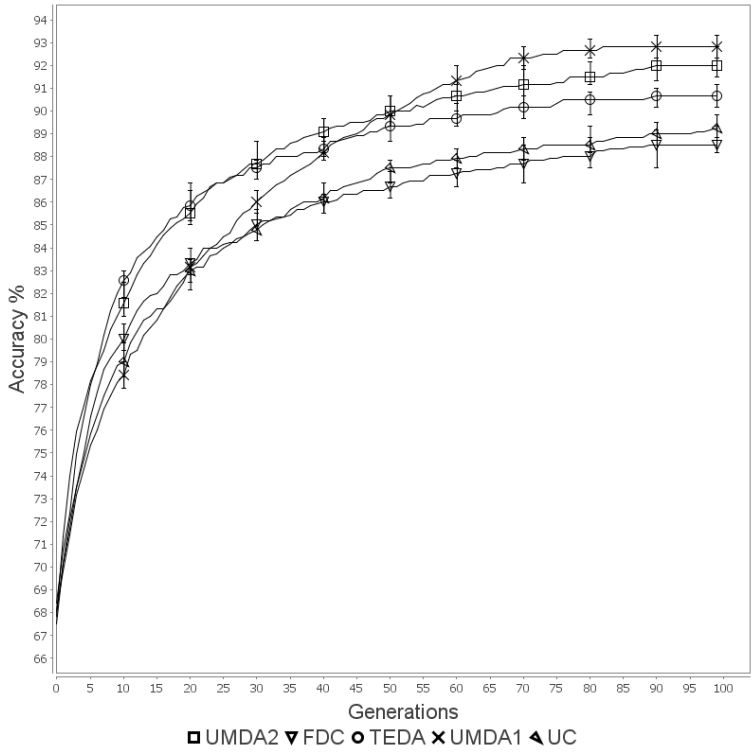

Figure 7.28: Madelon - Accuracy against Fitness Function Evaluations

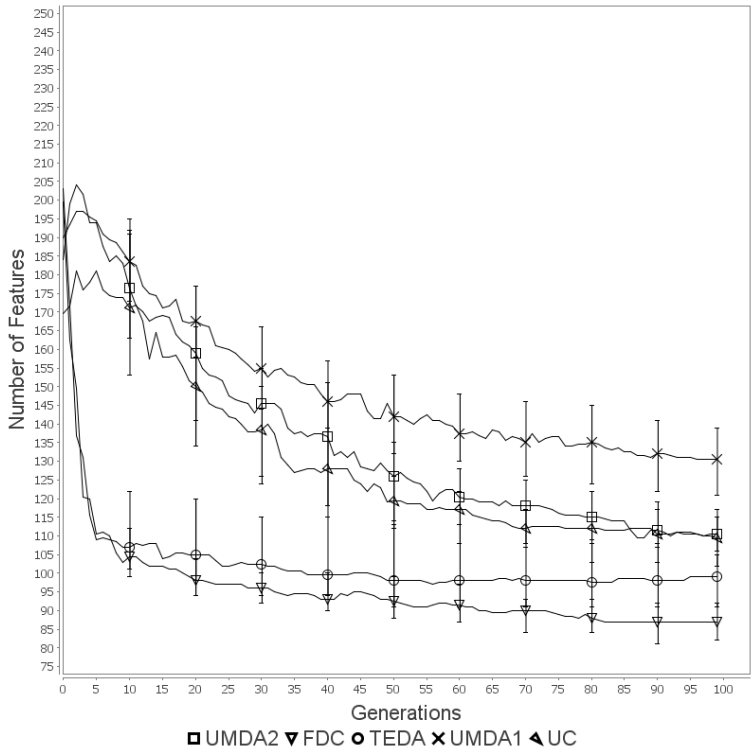

Figure 7.29: Madelon - Control Points (Features)

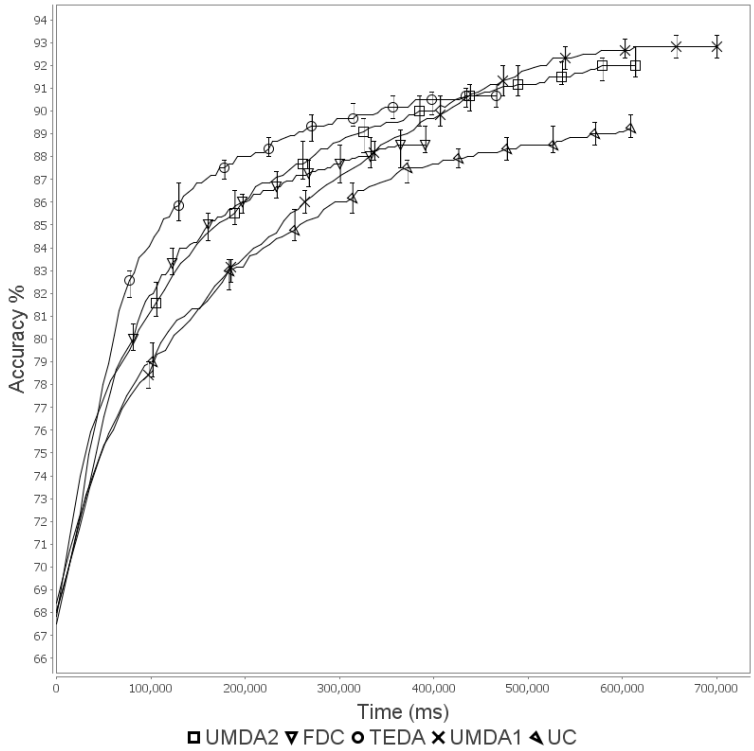

Figure 7.30: Madelon - Accuracy against Classification Time

Table 7.9: Madelon - Kruskal Wallis for Evaluation Times

Accuracy (%)	TEDA vs UMDA2	TEDA vs FDC	TEDA vs UMDA1	TEDA vs UC
80.0	***66.72***	***45.0***	***136.32***	***126.06***
84.0	***66.18***	***49.76***	***128.06***	***134.3***
88.0	***43.5***	***90.3***	***75.52***	***134.98***
92.0	36.04	11.5	9.12	***68.24***
Threshold	40.6			

7.4.3.1 *Using the Balanced Error Rate*

Figures 7.31, 7.32 and 7.33 demonstrate how each algorithm performs on the Madelon problem when the BER is used. These graphs show the BER, the feature set size and the BER over time respectively.

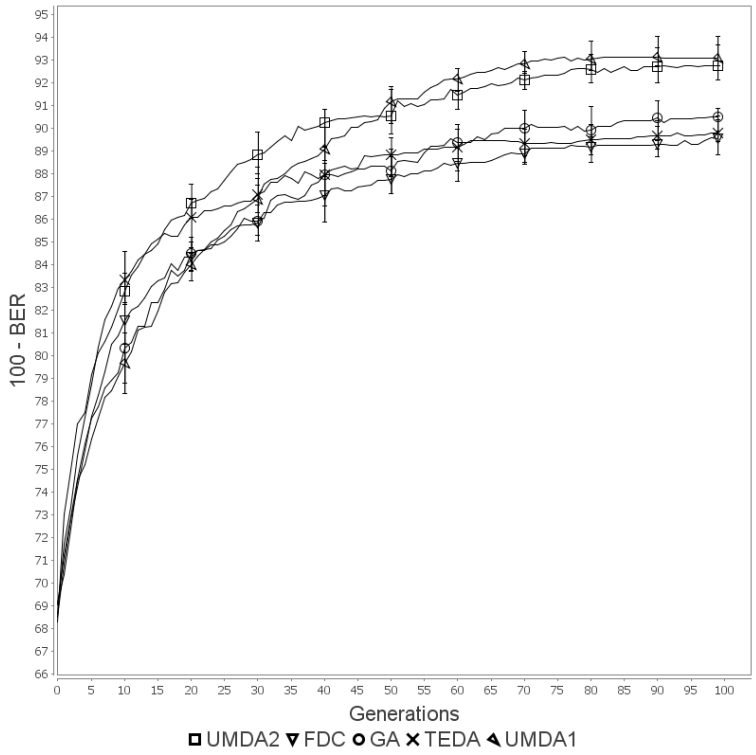

Figure 7.31: Madelon - BER against Fitness Function Evaluations

When the BER is used TEDA performs less favourably when compared to the other algorithms. It can be seen from the time graph (Figure 7.33) that early on it still performs slightly better than the other approaches. This is due to the fact that it still finds smaller solutions. Later on, however, TEDA performs significantly worse than UMDA2 and ultimately UMDA1 as well. It may be the case that the highly multivariate nature of the Madelon problem limits TEDA's performance. When initially reducing the feature set size, TEDA may not be taking into account the fact that some features are only useful when combined with other features. It should also be noted that the initial feature set size for this problem, 500, is noticeably less than in the other problems and that a large proportion of these 500 features appear to be useful. Both EDAs find good solutions that use over 100 features, around a 5th of the initial feature set size. TEDA finds smaller solutions than this and these smaller solutions appear to be less accurate. It seems likely that TEDA is more effective on problems where a lower proportion of features are useful otherwise there may be a tendency for it to over aggressively drive down the feature set size.

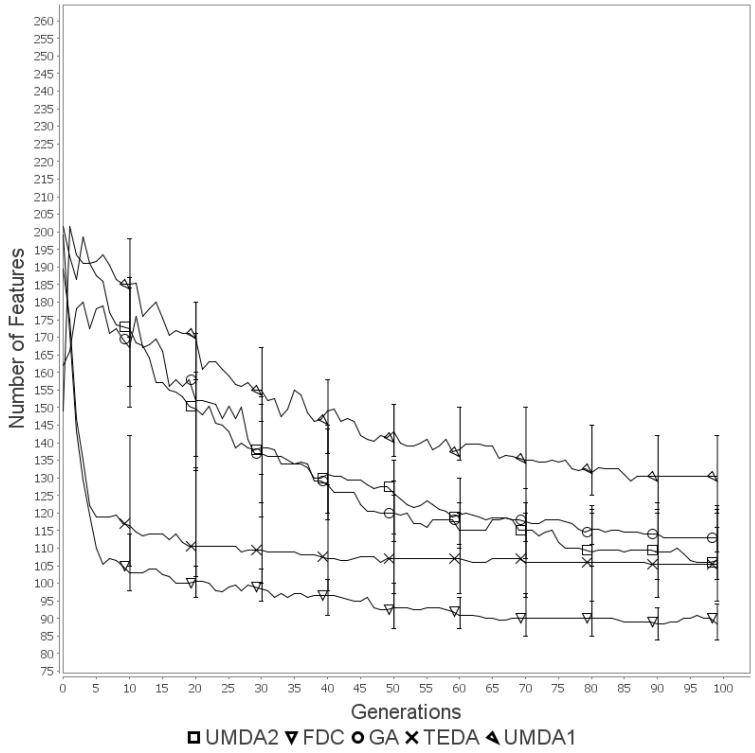

Figure 7.32: Madelon - Control Points (Features) using BER

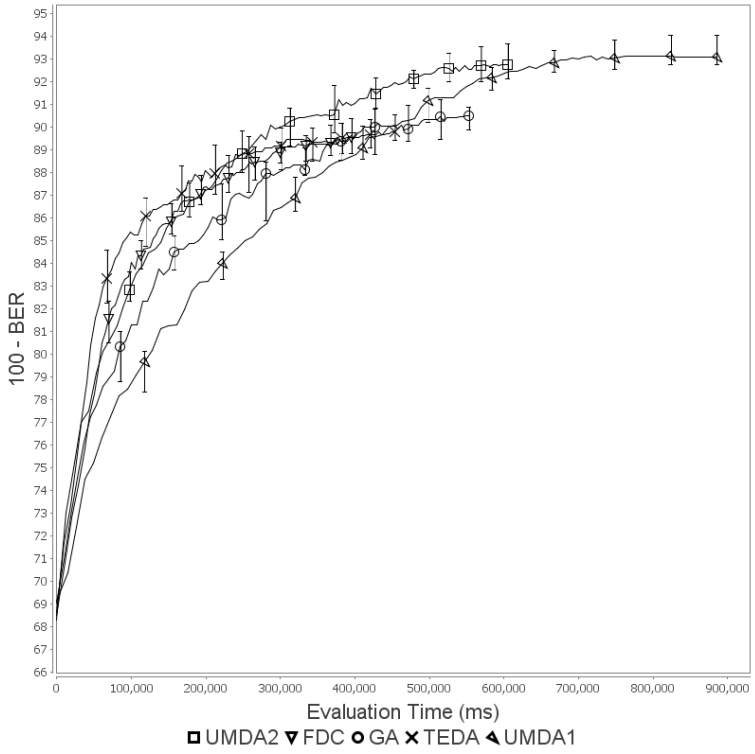

Figure 7.33: Madelon - BER against Classification Time

In this chapter TEDA has been applied to FSS problems. We have tested TEDA on three FSS problems from literature and compared it to FDC, standard EAs that do not use targeting, forward and backward selection and to established filter methods of FSS.

The overall conclusion is that TEDA is able to find accurate solutions quicker than EAs that do not use targeting by finding smaller solutions that are quicker to evaluate. We can also say that, for FSS problems, the speed with which TEDA finds good solutions relies on TEDA's targeting capability. FDC is also often more effective than an untargeted GA and so this is further evidence that targeting is beneficial for these problems. TEDA and FDC do, however, ultimately not always manage to find solutions as good as those found by methods that do not use targeting. TEDA's probabilistic model building capabilities as an EDA is also beneficial as, when compared to FDC, it is often the better performing of the two approaches in terms of both the time taken to find good solutions and the quality of solutions ultimately found and in no case does it perform worse than FDC.

In one of the three problems, Arcene, TEDA was able to find effective feature sets early on even though conventional EAs were noticeably delayed. This was because, on this problem, it appeared to be the case that effectively searching for good solutions was only possible after the feature set size had been dramatically reduced. With explicit targeting, TEDA was able to reduce the feature size and so begin effectively searching much sooner than the other techniques.

Areas in which TEDA did not perform better than other approaches include when a different classifier was used, the K-Nearest Neighbour (K-NN) Classifer, and when a different scoring function, one based on balanced fitness instead of accuracy was used for the Dexter problem. Using the K-NN classifier TEDA's performance, slightly worse than that of UMDA, appears to be attributable to the fact that K-NN does, in this case, require a large number of features and TEDA drives down the size of the feature set too far. However, we can say that for this problem a combination of TEDA and a classifier that is effective at eliminating redundant features is the most effective approach. When a balanced scoring system was used in Dexter, TEDA's performance matched that of UMDA and no approach performed better than either of these two approaches. In Madelon, although with a balanced scoring system TEDA still performs well early on, ultimately it performs quite significantly worse than a standard EDA. This is possibly due to the smaller size of this problem and its multivariate nature.

Overall TEDA can be said to be most useful on very large and noisy problems. In Dexter, solutions with an accuracy of 99% were found that contained only 500 features, only 2.5% of the complete set of 20,000 features. In Arcene, solutions with an accuracy of over 90% were found using only 10 out of 10,000 features (0.1% of available features). Larger feature sets actually proved unable to effectively classify the data in these problems. We refer to

these problems as large and noisy as the total number of features is in the order of 10^4 but the vast majority of these features appear to be useless or even detrimental to classification accuracy. Although it is on problems such as these where TEDA is most useful, it still finds relatively good solutions even on the Madelon problem, finding solutions of a similar quality to a standard GA and less effective only than those found by an EDA. We can therefore say that TEDA can be used successfully even on problems that it is not optimal for.

When compared to standard FSS methods TEDA was, in every case that we tested, able to find better solutions than the other approaches. Forward and Backward selection were found to be too slow at finding good solutions to these very large feature selection problems. When we compared TEDA against state of the art filter methods it was found that, although they have the advantage of being able to find reasonable solutions while only needing to carry out the full classification process once, TEDA is able to find better quality solutions to any of the methods against which it was tested.

Overall, it appears that the main areas to explore if TEDA is to be further investigated as a FSS algorithm are how to integrate a multivariate element into TEDA so that it does not lose dependency information in problems such as Madelon and how to correct a tendency to overly aggressively target in less noisy problems such as the Madelon problem.

From the results in this chapter we can say that TEDA could be considered in situations where the initial feature set is very large and noisy and where the speed with which good solutions are found is more important than finding the best possible solutions. This is useful as fast learning methods may be of benefit in situations where feature selection has to take place on the fly or problems arise that have to be solved quickly [41].

8

MULTIVARIATE EDAS

For some deceptive problems univariate algorithms have been proven to be insufficient [62] and it is difficult to build an accurate model of a problem using a univariate EDA when the structure of solutions is as important as the actual values within that solution [13]. TEDA so far has been based on univariate EDAs only and it is a logical step to attempt to integrate multivariate EDAs into TEDA. In this chapter, we will look at a couple of deceptive multivariate problems and explore how well TEDA is able to solve them. We will then investigate two types of multivariate EDAs and begin to discuss whether it would be possible to integrate these into TEDA to improve its performance at these problems.

Through multivariate EDAs we will also explore whether it is possible to take advantage of another potential benefit that TEDA offers over approaches that do not use targeting. As seen for the feature selection problems, TEDA is able to quickly reduce the number of features used for extremely large and noisy problems. By removing the vast majority of features from consideration it effectively reduces the dimensionality of these problems dramatically. The transitioning process ensures that an EDA is used only once the population has started to converge and convergence tends to coincide with a reduction in the size of the feature set. After this point it may potentially enable sophisticated multivariate EDAs to be applied to problems that were previously considered too large for them. This is important as tests which focus on feature sets of a smaller size have already proved that multivariate EDAs may offer benefits over univariate EDAs in the solution of Feature Subset Selection (FSS) problems [70].

8.1 MULTIVARIATE EDAS

The two multivariate EDAs used in this section are ECGA and EBNA. ECGA attempts to divide the genes up into cliques, at each iteration joining the clique which minimimizes a *model complexity score*. EBNA builds a Bayesian network where each node is a gene. Both systems are explained in detail in Section 2.8.3. EBNA is used as it was previously proved by Inza et al to be successful at solving FSS problems [69] [70].

8.2 MULTIVARIATE PROBLEMS

As we have stated, FSS problems are an example of an area in which a multivariate version of TEDA may bring benefits. For this reason we use as one of our example problems in this

chapter a FSS problem that we have artificially generated to have a high level of dependency between features. The second example problem that we use in this chapter is a well known deceptive problem often used to assess multivariate EAs, the K Trap problem [36] [62].

8.2.1 *K Trap*

The K Trap problem consists of a concatenation of m trap problems of k bits each. An individual trap problem, the ith problem within the complete concatenated problem, is denoted as $trap_i$. The fitness of a vector x when used to solve $trap_i$, $f(x, trap_i)$, is calculated as detailed in Equation (8.1). x is a binary vector of length k in which we represent $true$ and $false$ as 1 and 0 respectively. An individual element within vector x is denoted as x_j.

$$f(x, trap_i) = \begin{cases} k, & \text{if } \sum x_j = k. \\ k - \sum x_j - 1, & \text{otherwise.} \end{cases} \tag{8.1}$$

When we concatenate all m problems, our final fitness function for solution x becomes:

$$f(x) = \sum f(x, trap_i) \tag{8.2}$$

This problem is chosen because it is known to be deceptive for any case where $k \geqslant 3$. This is because, for each subproblem a smaller number of positive genes offers a higher fitness despite the fact that, in order to achieve the optimal fitness, every gene must be positive. Any algorithm used to solve this problem must be capable of processing substructures of an order of at least k [128], making it a suitable test of the multivariate ability of any algorithm. In our tests the k trap problem was implemented with a k of 4 and an m of 10.

8.2.2 *Classifier Problems*

As previously mentioned, FSS are often highly multivariate and a major objective in developing TEDA into a multivariate algorithm that is able to solve large multivariate FSS problems by applying linkage learning techniques after the number of features have been reduced. To this end we have, first of all, developed an artificial FSS problem that is specifically designed to be highly multivariate and to be deceptive, so that a univariate FSS algorithm should not be effective. Once it has been established whether multivariate techniques are more effective at solving this problem than univariate techniques the size of this problem will be increased to a size of 500 features. A size that is, as shall be demonstrated, too large for conventional multivariate EDAs. A multivariate version of TEDA will then be applied to this problem to establish whether a TEDA approach may, in theory, work for large and highly multivariate

Paremeter	Value
c_{min}	3
c_{max}	5
p	0.01
n	50
l	100,500

Table 8.1: Artificial Classifier Problem Generation Parameters

problems. The use of an artificial problem with known cliques will allow us to investigate whether TEDA is in fact able to identify these cliques and whether or not these cliques are likely to be lost before the feature set size is small enough to apply a multivariate technique.

8.2.2.1 *A Multivariate Artificial Classifier*

The highly multivariate artificial feature selection problem is designed so that every feature falls into one of a number of cliques and that only entire cliques are useful for classification. It was created through the following steps:

1. n features are designated as belonging to a clique where $n \leqslant l$, l being the total number of features available.
2. These n features are divided up into cliques where each clique has a size between c_{min} and c_{max}.
3. For each clique a set of integer vectors is randomly generated, one for each class.
4. Test solutions and training solutions are produced. This is done in such a way as to be deceptive. The variables in each solution are chosen to match the vectors produced in step 3 but, as incomplete copies of these vectors exist in training samples of the wrong class any feature set that only includes a subset of clique C will incorrectly classify more samples than a feature set that includes no members of C.
5. For each solution the remaining $l - n$ features that are not part of any clique are given a random value. These are considered noise, features that are no use for the classification process. To imitate large, sparse problems of a similar nature to the Dexter or Arcene problems these features are more frequently given a value of "unknown" than they are given a numeric value. They are each given a numeric value by a pre specified probability p.

Table 8.1 shows the parameters with which this feature set was generated.

Paremeter	Value
k for K-NN	3
penalty	0.1
k for k-trap	10
m for k-trap	4

Table 8.2: Experimental Parameters - Multivariate Problems

8.2.2.2 *Fitness Function*

The fitness function used in these tests is the same as was used for the other FSS problems described in Section 7.3.3. We calculate the classification accuracy and then subtract a penalty to reflect the efficiency benefits in finding smaller solutions. The penalty is 0.1 in all cases. This corresponds to $10/l$, the same penalty as used in the previous chapter, with a feature set size of 100. The classifier used is the K-Nearest Neighbour (K-NN) classifier. This was used because the artificial classifier problem was designed specifically so that it is deceptive when the K-Nearest Neighbour (K-NN) classifier is used.

8.3 EXPERIMENTAL METHOD

The evolulionary parameters are the same for both the k-trap and the FSS problems. In all cases, solutions are encoded as fixed length binary strings.

Solutions are initialised by iterating through them and then randomly choosing a value for each gene. This value will be, with equal probability, either *true* or *false*. Mutation is carried out through the same method as all other problems in this thesis that aren't regarded as abnormally large or noisy problems. That is to say, all problems except for the large feature selection problems used in the previous chapter. This distinction is explained in Section 7.3.1.1. This means that mutation is attempted a number of times equal to the chromosome's total length or maximum possible number of features and each time will be carried out with the specified mutation probability. When it is carried out, with equal probability, either a gene set to true is selected to be set to false or a gene set to false is selected to be set to true.

Table 8.2 provides the main parameters used in these tests.

In all tests EBNA and ECGA are implemented with the same parameters as UMDA1. This means that they are run with a mutation rate of 0 and a breeding pool size of 50% of the population. These are standard EDA parameters and are in accordance with those used in other publications [62][69][70].

In this chapter, two different population sizes are used, 100 and 1000. In all graphs presented in this section results are plotted against generations on the x-axis. These are equivalent to 100 fitness function evaluations for a smaller population and 1000 for a larger population. In graphs where two different size populations are compared, results are plotted against every 100 fitness function evaluations and so one generation with a larger population will cover a greater space. For the FSS problem graphs in this chapter show accuracy against evaluations, number of features against evaluations and combined fitness against evaluations. The combined fitness graph plots the fitness function against generations, consisting of accuracy combined with the penalty for the number of features used. Where algorithms have a similar accuracy this shows which ones are more effective at removing redundant features, given that redundant features don't have a detrimental effect on fitness in this problem.

8.4 TEDA FOR THE K-TRAP PROBLEM

In this section a range of univariate and multivariate algorithms are applied to the k-trap problem. The results in this section are for just 50 generations as no changes were observed after this point.

8.4.0.3 *Univariate Algorithms*

First we compare various different univariate techniques for solving the K trap problem. These include UMDA, a standard GA featuring UC, FDC and TEDA.

The pattern observed in Figure 8.1 is that, due to the deceptive nature of this problem, none of the algorithms are especially effective but that TEDA appears to be slightly more effective than the other approaches. For the first few generations the fitness improves but, for every algorithm, the fitness quickly becomes trapped in a local optimum and plateaus. TEDA achieves a slightly higher fitness than the other approaches.

Figure 8.2 plots the total number of genes with a value of true, *positive genes*, in the solutions that each approach discovers. In all cases, the solutions discovered are significantly smaller than the size of the ideal solution of 40. This suggests that they are following the deceptive slope towards smaller and smaller solutions. FDC and UMDA2 ultimately reduce the solution size to 0 whereas UMDA1 finds slightly larger solutions and TEDA finds slightly larger solutions still. In TEDA, FDC and UMDA2 the size of solutions drops in the first couple of generations but at this point, in TEDA, the solution size increases slightly again. As the solution size and the fitness increases it is possible that TEDA has selected a solution that does, despite being small, contain at least one intact clique. As this solution will be slightly larger and slightly fitter than the very small solutions that are fit simply due to the deceptive slope we can see how TEDA may, at this point, breed a slightly larger solution with a reasonable probability of

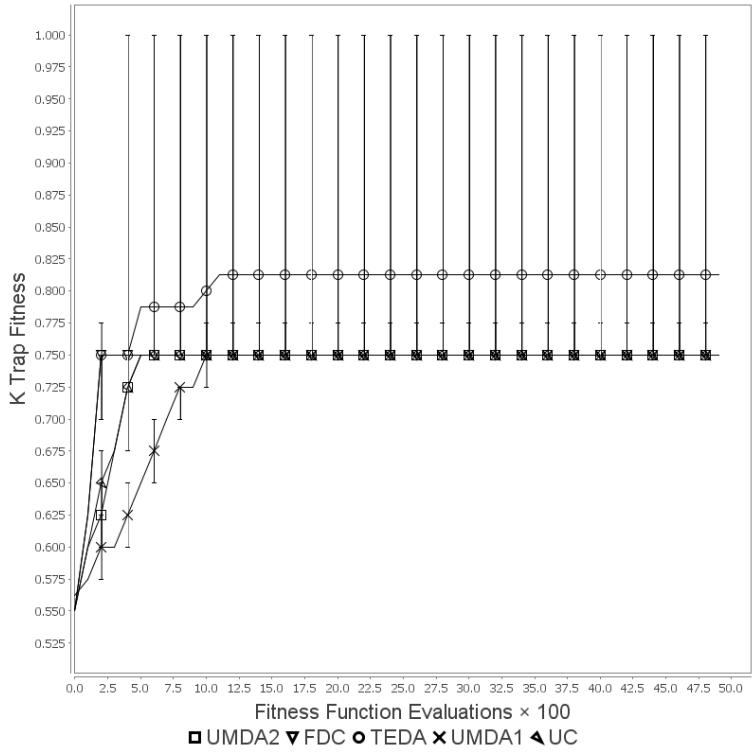

Figure 8.1: K-Trap - Univariate Approaches - Performance

it keeping the clique that was present in its parent intact. These solutions are most likely not being selected in FDC due to the lower selection pressure.

We have also observed that the spread of fitnesses in TEDA as well as in FDC, as shown in Figure 8.3, seems to suggest that these approaches do occasionally discover the optimal solution whereas the spread of fitnesses associated with the other approaches, as shown in Figure 8.4, suggests that the likelihood of these discovering optimal solutions is lower. This may be due to the fact that the fitter solutions in a population will tend to fall into two categories. The first category will likely form the majority. These are solutions with few positive genes, solutions that are following the deceptive slope of this problem. The second category will consist of a smaller number of solutions that are fitter due to the fact that they have a large number of entire cliques and so contain a much larger number of positive genes. Occasionally it will be the case that solutions in the second category are selected and drive the evolutionary process in the first generation. If this is the case in a targeted algorithm then the targeting will drive towards larger solutions. In the next generation the proportion of solutions in the second category will be higher and so it is more likely that TEDA and FDC will drive towards larger solutions than smaller solutions. If this happens then it is probable that in a few generations TEDA and FDC will drive towards the largest solutions possible. In the case of the k-trap problem this solution will also be the optimal solution.

This does, however, only work due to the fact that the k-trap problem has this property that a maximum size solution is also the optimal solution. Real problems are unlikely to have a

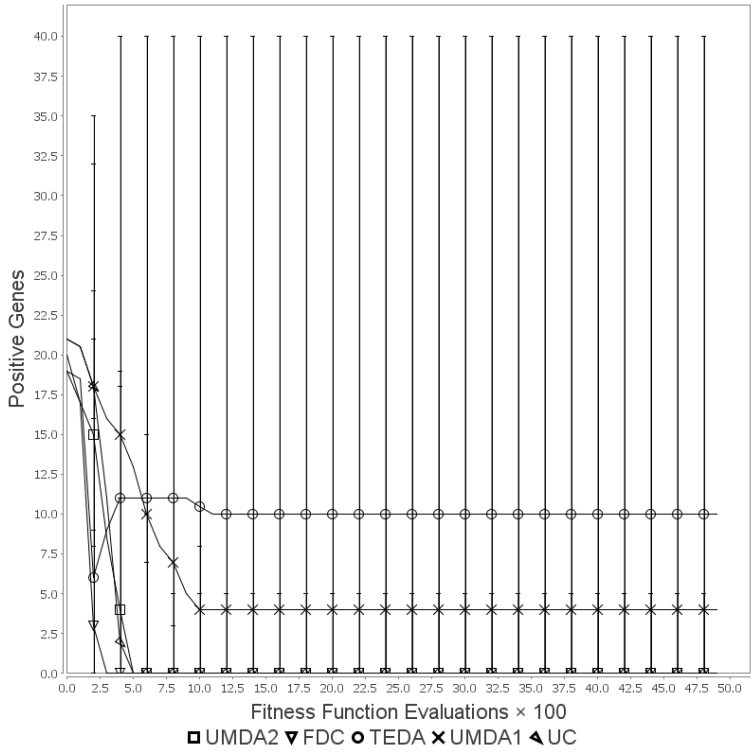

Figure 8.2: K-Trap - Univariate Approaches- Control Points (Positive Genes)

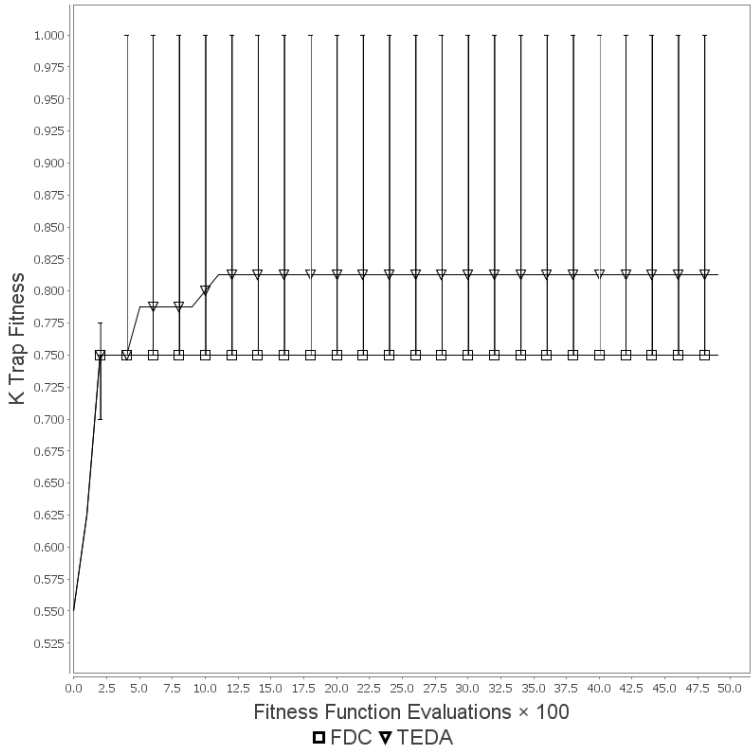

Figure 8.3: K-Trap - Targeted Univariate Approaches - Performance

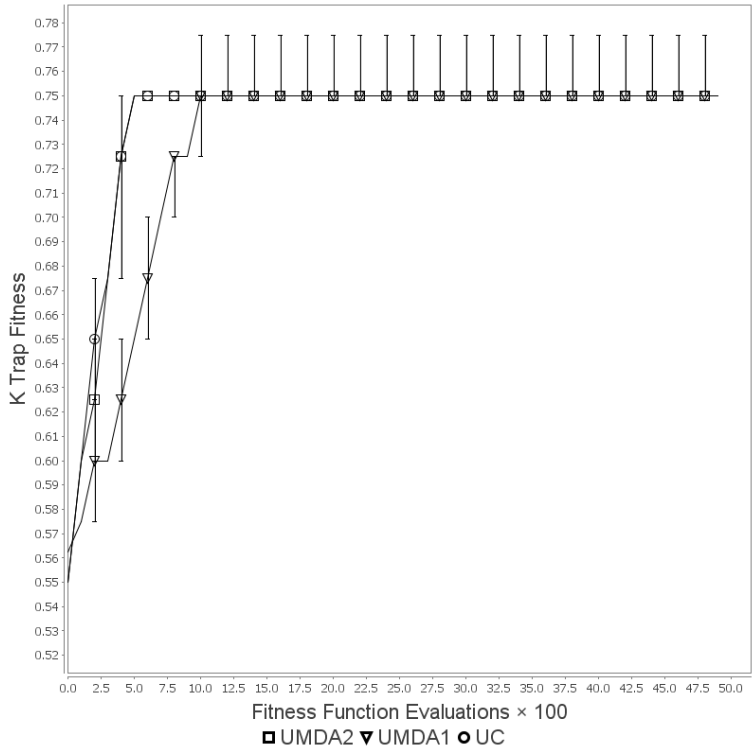

Figure 8.4: K-Trap - Untargeted Univariate Approaches - Performance

simple rule such as this that TEDA and FDC are able to exploit. The main conclusion seems to be that multivariate approaches should be used for a problem such as this but that if a technique that is able to discover cliques is used then targeting may offer further benefits in some problems.

8.4.0.4 *Multivariate Algorithms*

We now show how well the two multivariate EDAs investigated, EBNA and ECGA, perform on this problem.

In Figure 8.5 we can see that EBNA performs better at this problem than ECGA. The results in Figure 8.6 show the number of positive genes in the solutions discovered and show that only multivariate EDAs are able to avoid the trap of ultimately driving towards solutions with almost no positive genes. When we compare these results to the results in the previous section that show how other univariate approaches perform on this problem, it is clear that neither univariate GAs nor univariate EDAs with or without targeting are able to consistently avoid this trap and so this problem highlights the need for integrating a multivariate approach into TEDA.

However, it can be seen from the error bars in Figure 8.5 that none of the multivariate EDAs find solutions with the maximum fitness of 1, as is the case with the targeted EAs in Figure 8.3. This suggests that, although the multivariate EDAs are able to more consistently avoid the trap than the other algorithms, the univariate approaches that use targeting, if they do by

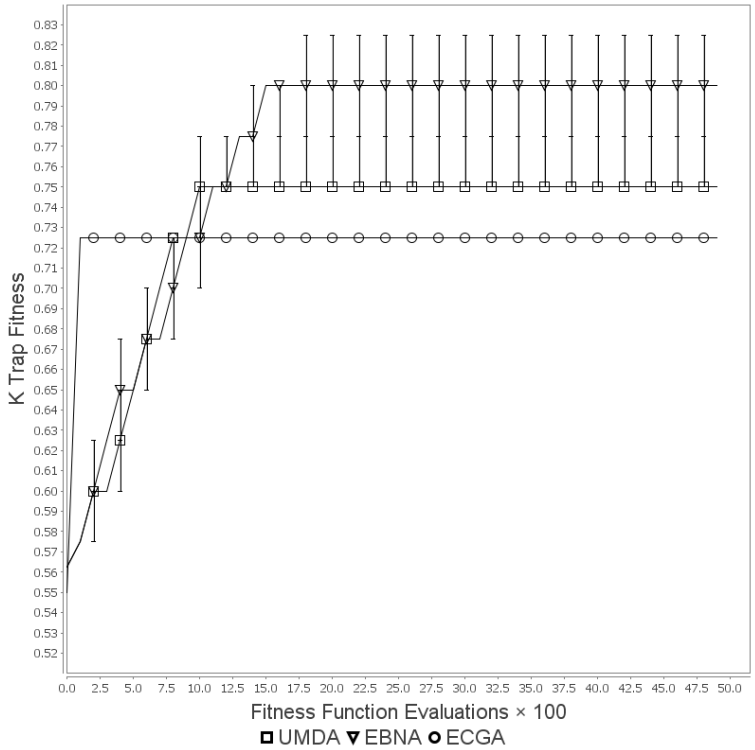

Figure 8.5: K-Trap - Multivariate Approaches - Performance

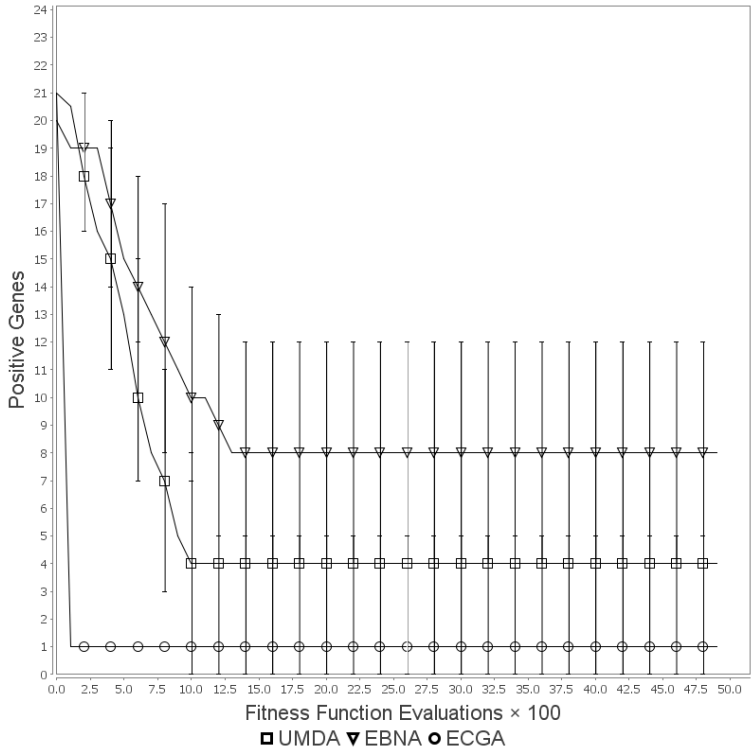

Figure 8.6: K-Trap - Multivariate Approaches - Control Points (Positive Genes)

Paremeter	Value
UMDA1 and EBNA	
breeding pool size	500
UMDA2	
breeding pool size	100
TEDA	
Tournament Size	50
max_b	100
max_s	1000
min_b	2
min_s	100

Table 8.3: EDA Scaled Parameters for a Population of 1000

chance avoid the trap, can sometimes find solutions of a higher quality than any of the other approaches. It is therefore possible that if integrating linkage learning into TEDA ensures that it avoids progressing down the deceptive slope early on then we will ultimately have produced an algorithm that is more effective than either univariate TEDA or a multivariate EDA.

8.4.1 *With a Population of 1000*

In this section we present the results of carrying out the same tests as described above but with a population of 1000. The results in Figure 8.7 show how univariate algorithms performed whereas Figure 8.8 shows how multivariate algorithms performed. A larger population helps multivariate EDAs to build an accurate model and 1000 is a size that has previously been used in a successful attempt to solve a feature subset selection problem with EBNA [70]. Table 8.3 gives the value of various EDA parameters that have been scaled up in proportion to the increased population size.

With a larger population TEDA appears to perform better than any of the other approaches, quickly reaching the optimum. The other approaches are only minimally affected by increasing the population size. Among the EDAs, UMDA performs slightly better although it is still outperformed by EBNA. The reason for TEDA's good performance is an extension of the reason, already discussed, why it occasionally performs well with a smaller population. With a large population there is a high chance that the fittest individual in the breeding pool will be fit due to having a large number of intact cliques. There is also a high chance that a medium fit individual, such as the least fit individual in the breeding pool, is fit simply because it has

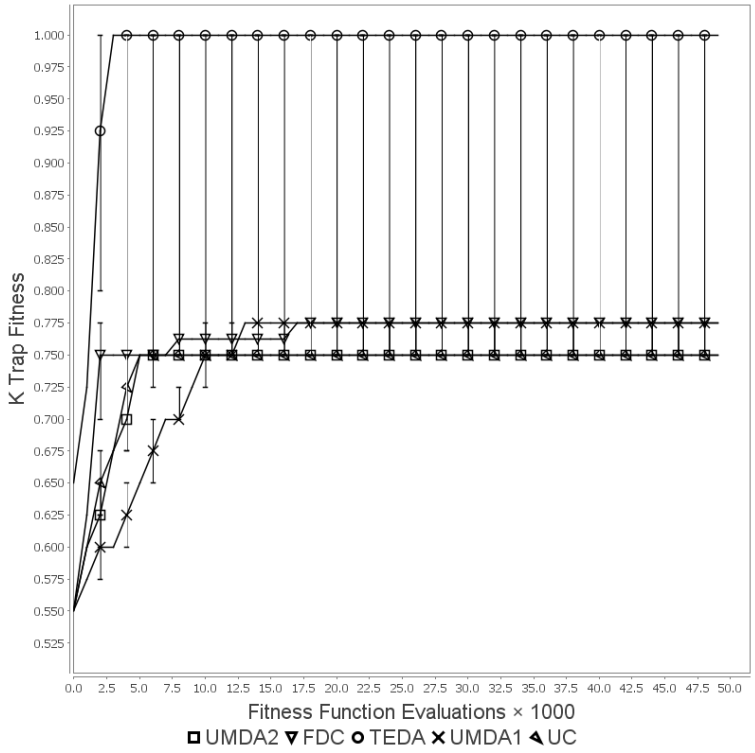

Figure 8.7: K-trap - Univariate Approaches - Population Size of 1000 - Performance

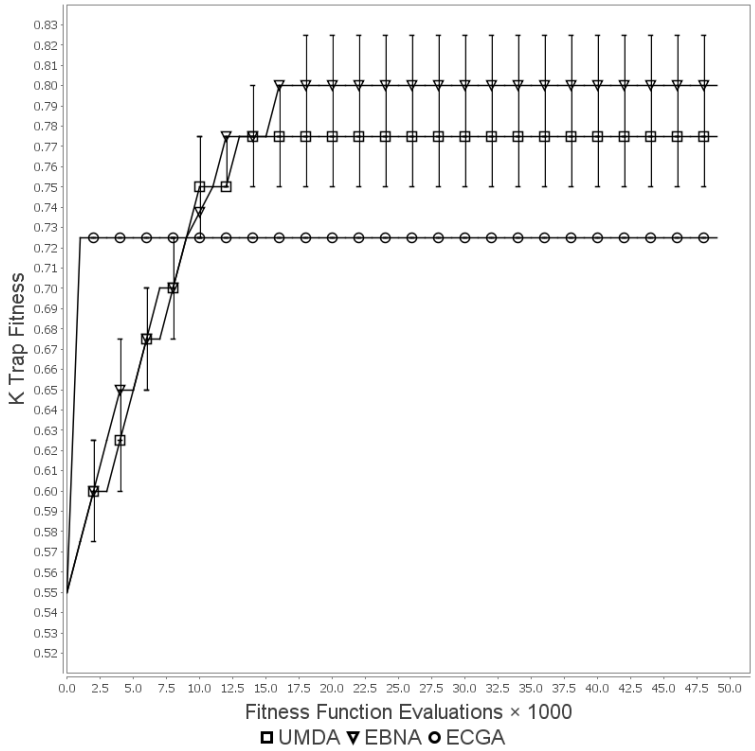

Figure 8.8: K-trap - Multivariate Approaches - Population Size of 1000 - Performance

a small number of positive genes. In this situation targeting is likely to drive the number of positive genes sharply upwards to quickly reach a situation in which all genes are positive. As mentioned, this will be the optimal solution. This therefore shows that TEDA can perform well on a multivariate problem but only by exploiting the fact that the optimal solution is at one extreme in terms of numbers of positive genes. This is a feature of this problem that is unlikely to occur in real world problems. The next step is to therefore see how TEDA performs on a more complex multivariate problem.

Now that we have investigated the performance of a number of algorithms on the k trap problem with both a small and a large population we can make a number of overall conclusions. It appears that for this small problem the fact that GAs occasionally select solutions that are fit for the right reasons whereas EDAs tend to ignore individual solutions is sufficient to ensure that GAs can perform better even than multivariate EDAs. Targeting may further improve the performance of GAs as, in this problem, a solution's fitness is very closely related to the total number of positive genes and if a targeted algorithm even once drives the number of interventions in the correct direction then it may avoid the deceptive slope and find the optimal solution. We do, however, suspect that for larger problems that cannot simply be solved through maximising the number of positive genes, the benefits that both multivariate EDAs tested, ECGA and EBNA, appear to offer over over a univariate EDA may enable TEDA to solve problems that it would otherwise be unable to solve.

8.5 TEDA FOR THE CLASSIFIER PROBLEM

This section contains a comparison of the performance of all of these algorithms on the artificial FSS problem. The algorithms compared are, as in the previous section, TEDA, UMDA1, FDC and a standard GA using UC.

8.5.1 *100 Feature Problem*

In this section we have used a version of the artificial classifier problem with an n of 50 and an l of 100. That is to say, a 100 feature problem in which 50 of these features are part of a clique and 50 are random noise. Firstly, the results in Figure 8.9 shows the comparison in the accuracy between solutions discovered using a univariate EDA, UMDA and a multivariate EDA, EBNA, with populations of size 100 and 1000. EBNA was used instead of ECGA due to the latter's relatively poor performance on the k-trap problem. From these results we can see that EBNA with a population of 1000 found fitter solutions than any of the other approaches. It is also faster at finding good solutions than UMDA with a population of 1000. This shows the benefits of multivariate EDAs for this problem but also shows that a large population is required for these algorithms to reach their full potential.

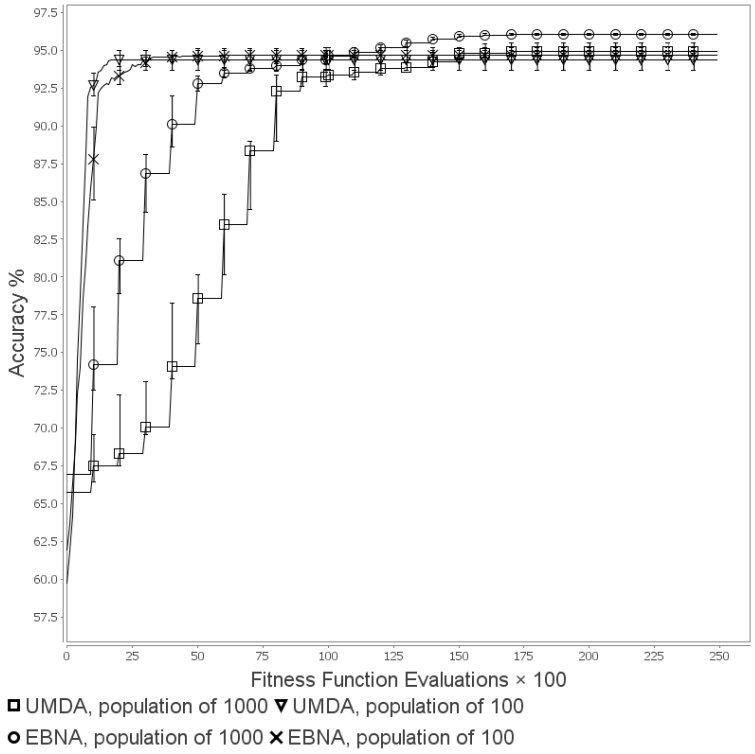

Figure 8.9: Artificial Problem with 100 Features - The Performance of UMDA and EBNA with Different Population Sizes

Having established the positive performance of EBNA with a population of 1000 on this problem, we will now demonstrate the performance of various other univariate and multivariate approaches, also using a population of 1000. The results in Figure 8.10 show how untargeted univariate approaches, a standard GA and UMDA compare to EBNA. ECGA is included as well. The results in Figure 8.11 show how the targeted approaches, TEDA and FDC, compare.

EBNA appears to be able to find good solutions faster than either UMDA or ECGA and ultimately finds better solutions than either of these approaches. In contrast, all of the approaches that start with a GA approach do perform better than EBNA early on but these, too, are all unable to find solutions as good as those found by EBNA. These include a standard GA, TEDA and FDC. TEDA performs better than FDC as FDC appears to reach an accuracy of around 91 in the first few generations and is subsequently unable to improve further.

The results in Figure 8.12 show how the targeted approaches and EBNA compare in terms of the fitness function instead of just the accuracy score. This fitness score, as previously explained, is obtained by combining the accuracy of a solution with the total number of features set in it. It is important to show this because a good algorithm should be able to remove redundant features and yet, for this problem, it is unlikely that these redundant features will actually have any effect on the accuracy score alone. These results appear to show that, in terms of the combined fitness, TEDA's performance is closer to that of EBNA.

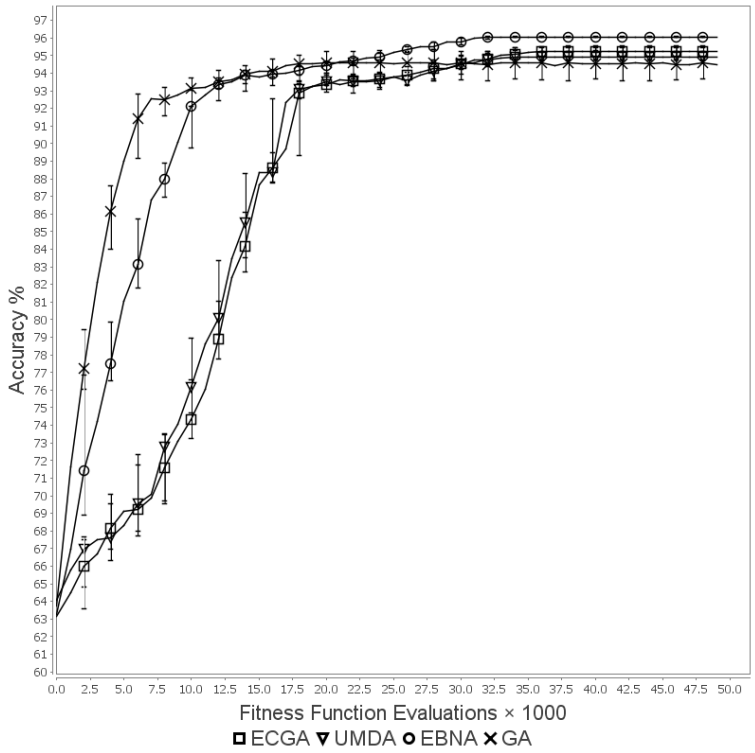

Figure 8.10: Artificial Problem with 100 Features - Accuracy - Untargeted Approaches Compared to EBNA

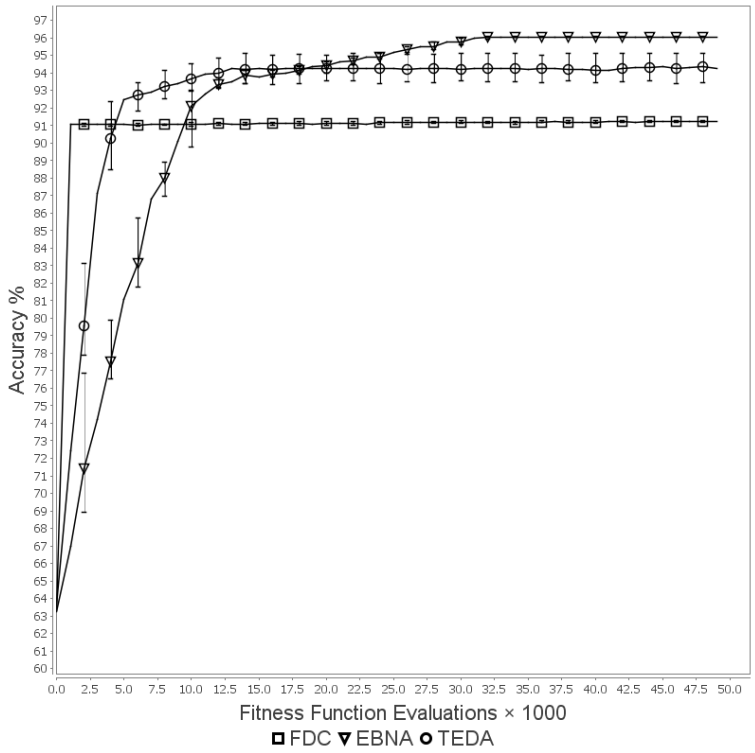

Figure 8.11: Artificial Problem with 100 Features - Accuracy - TEDA and FDC Compared to EBNA

The results in Figure 8.13 show the number of features in the fittest solutions found by the targeted approaches and by EBNA. TEDA appears to be more effective at reducing the feature set size than FDC and EBNA. EBNA is still able to remove redundant features as the size of solution does ultimately settle on a smaller value than in the starting population, presumably because features not placed into the DAG are eventually removed from consideration, but it takes longer to remove these features and removes less of them than TEDA does with its explicit targeting. This means that the solutions that TEDA finds receive a smaller penalty than EBNA and therefore their total fitness is greater in comparison to the total fitness of those found by EBNA than was the case when the level of accuracy alone was compared between the two algorithms.

FDC, by comparison, does not seem to effectively reduce the size of the feature set at all. In FDC the feature set size continues to rise and fall throughout the course of the test with little evidence of an overall trend. This, combined with the tendency of FDC to remain trapped at a lower level of fitness, as seen in graph 8.12 seems to suggest that it continues to explore the search space while lacking the selection pressure to exploit the contours of the fitness landscape.

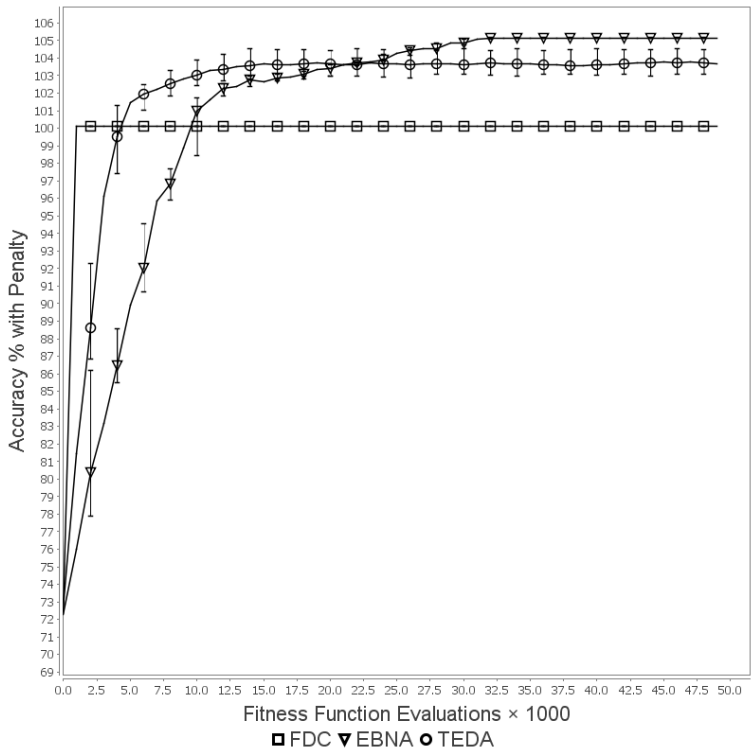

Figure 8.12: Artificial Problem with 100 Features - Fitness (Accuracy with Penalty)

Overall, TEDA is able to quickly remove redundant features but may, as evidenced by its lower accuracy score than EBNA, be also removing useful features due to it not discovering relationships. Therefore, by combining TEDA with EBNA it may be possible to develop a technique that effectively removes redundant features without losing any useful ones.

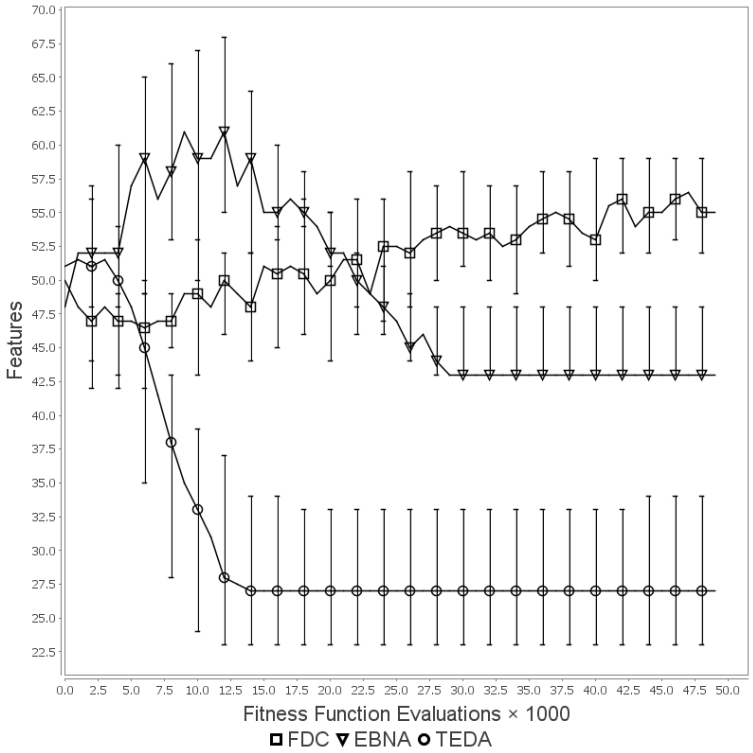

Figure 8.13: Artificial Problem with 100 Features - Control Points (Features)

8.5.2 *Scalability of Multivariate EDAs*

The Madelon problem is a 500 feature problem designed to be highly multivariate [56]. Multivariate EDAs should therefore be able to at least solve problems of this magnitude. We have mentioned that there is an efficiency issue associated with these algorithms when they are applied to larger problems. For this reason, before running performance tests, in this section we demonstrate to what extent the size of the feature set affects the time that EBNA takes to build and sample from its probability model.

We have demonstrated that EBNA is a multivariate EDA that is effective on this class of problem and so to begin with we shall explain just how complex EBNA is. Model building is the most complex process in multivariate EDAs as the number of potential relationships increases exponentially as the size of the problem increases. In order to accurately model the fitness function every potential relationship would need to be considered, at a cost of $O(l^k)$ for a problem with k variables that each have l different states [29]. The number of relationships assessed and the cost of assessing each relationship is what makes multivariate EDAs expensive. In the case of EBNA, every relationship is evaluated as a candidate to join the DAG. This means that the entropy associated with every relationship is measured, an expensive process. In addition, once a link has been joined up, all the potential links in the population have to be re-evaluated in the context of the new network. In the implementation of EBNA used here, in which the efficient algorithm B is used, only a subset of candidate pairs

are re-evaluated each iteration (see Section 2.8.3). The total number of candidate pairs for which the entropy is measured in order to form the DAG is the main determinant of how time consuming and computationally expensive the EBNA process is. The worst case complexity associated with the process of building the DAG is $O(n^2 \times (n^2/2))$ where n is the number of variables.

The results in Figure 8.14 show how many pairs were examined in order to build the DAG when EBNA was applied to the artificial classifier problem of various different sizes. The result shown is the mean of 200 runs. Also shown is the *maximum* number of pairs that would need to be examined for a network this size in the worst case scenario. The worst case scenario occurs when a new link is added at every iteraction until no more links can be added. In most cases the process will end before this point as a point will be reached when no more links improve the quality of the network. Each version of the artificial problem has 50 features that are part of a clique in a total feature set of the size given below. When the problem has 400 or 500 features the maximum number of links is presented but not the actual number due to how time consuming this process becomes for problems of this size. The results in Figure 8.15 show the average length of time taken to build the DAG in each of these tests.

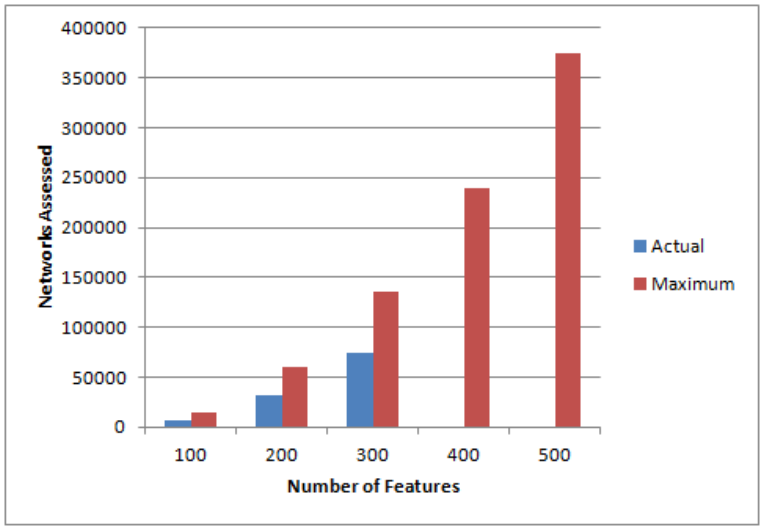

Figure 8.14: Artificial Problem - Efficiency of EBNA - Number of Comparisons against Number of Features

It can be seen from these graphs that the number of links that are assessed increases dramatically as the number of features is increased. This causes an enormous increase in the length of time taken to generate the network to the extent that this process takes, on average, almost 100 seconds each generation when the problem has 300 features, almost 10 times the length of time 200 features takes. If we look at how great the *maximum* number of links is in the situation with 400 and 500 links it appears that at this size EBNA would be impractical. This result is dependant on the implementation but it can clearly be seen that efficiency becomes a significant issue in problems with 100s of features and would be even

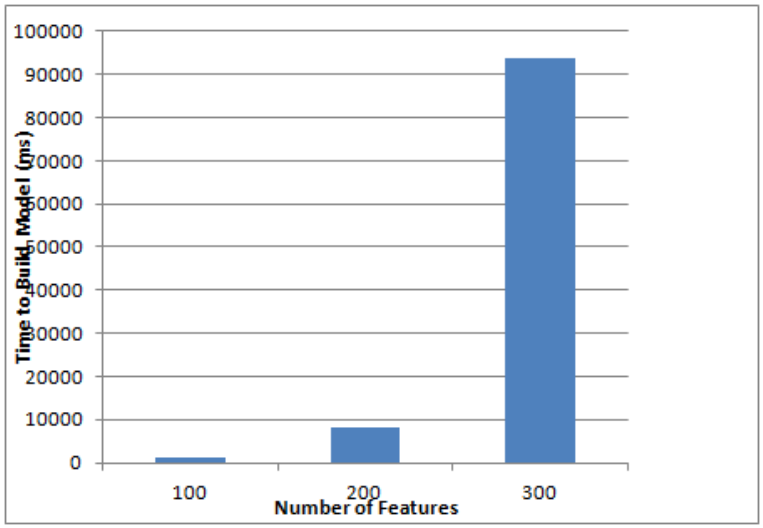

Figure 8.15: Artificial Problem - Efficiency of EBNA - Classification Time against Number of Features

more impractical with problems of a similar size to Dexter or Arcene, with 10,000 or 20,000 features. In addition, we have already stated that the search methods tested here produce a less accurate model than the ideal approach of using an exhaustive search [29]. Overall, it appears that a method that reduces the problem size before applying multivariate EDAs is essential for these larger problems and even in smaller problems, if it reduces the problem to a size where an exhaustive search strategy becomes an option at some point then it will also be of benefit.

8.5.3 *500 Feature Problem*

For the following tests, noise has been added to the artificial problem so that it now consists of 450 irrelevant probe features and 50 relevant features, making a total of 500 features. As already discussed, this increases the size of the problem to a point where applying a multivariate technique such as EBNA is impractical until the feature set size has been reduced. We will therefore now begin to consider whether it may be possible to use a version of TEDA that transitions into a multivariate EDA after a smaller feature set size has been discovered. The results in Figure 8.16 show the performance, in terms of accuracy, of TEDA, a standard GA, FDC and UMDA on this problem. A population of 1000 was used as multivariate algorithms were shown to perform better with a population of this size than with a smaller population.

In this situation all algorithms perform to a similar standard except for UMDA which takes longer to reach its optimum. However, TEDA is able to find slightly better solutions than the other approaches for most of the test up until the final few generations when the other solutions catch up with TEDA's performance. KW analysis (see Table 8.4) confirms that this improvement, while small, is statistically significant.

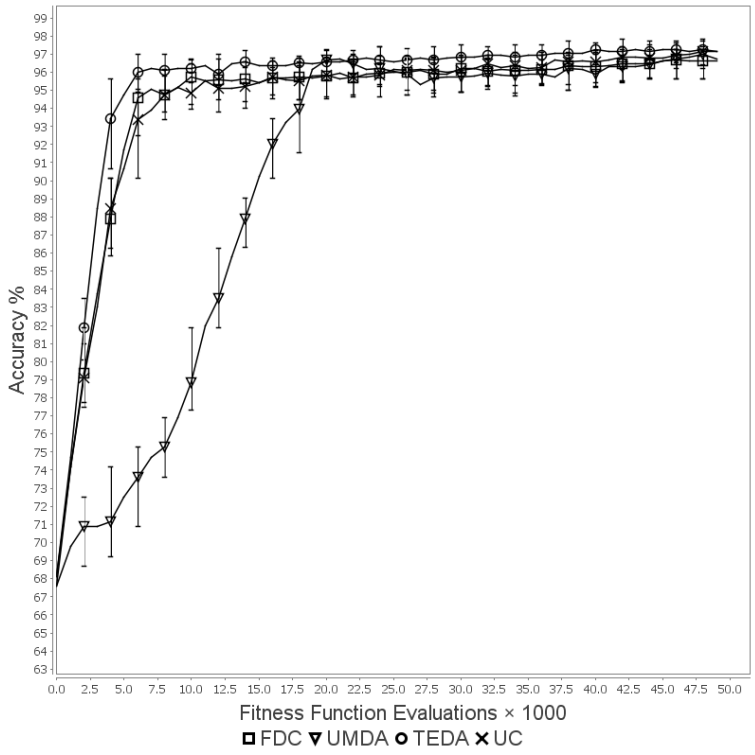

Figure 8.16: Artificial Problem with 500 Features - Accuracy

Fitness Function Evaluations*1000	TEDA vs FDC	TEDA vs UMDA	TEDA vs UC
10	26.0	*114.55*	25.62
20	*32.18*	21.39	*33.01*
30	*35.68*	*47.76*	*31.55*
40	*35.4*	*48.47*	*32.28*
50	26.09	21.27	14.9
Threshold	30.54		

Table 8.4: Artificial Problem- 500 Features - Kruskal Wallis

The results in Graphs 8.17 show the combined fitness achieved by each algorithm and Figure 8.18 shows the size of the best feature set. From these results we can see that UMDA is less able to remove redundant features than the other approaches. This is probably the cause of UMDA's initial inability to find accurate feature sets and it also has the effect that the combined fitness scores for UMDA are lower throughout the test. This appears to support the observations made in the feature selection chapter (see Chapter 7) that a univariate EDA is less effective at sparse feature selection problems than TEDA because it is unable to take advantage of the information that smaller solutions tend to be fitter whereas TEDA is able to discover this fact and drives towards producing more small and fit solutions.

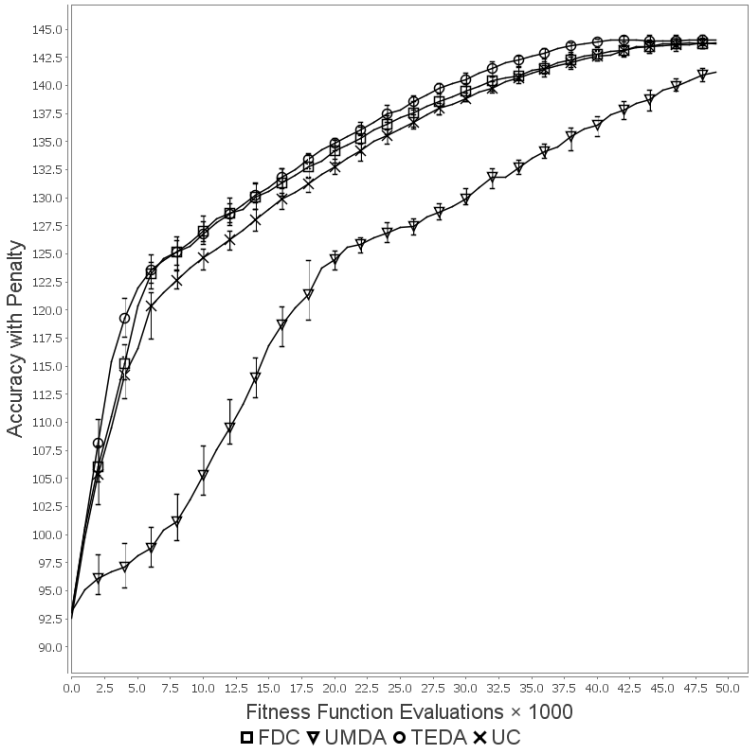

Figure 8.17: Artificial Problem with 500 Features - Fitness (Accuracy with Penalty)

Overall, univariate EDAs perform relatively poorly on this problem and multivariate EDAs have been shown to perform better on these multivariate problems than either TEDA or a univariate EDA yet cannot realistically be applied to larger multivariate problems. These results therefore seem to support the idea that using TEDA to reduce the size of the feature set before applying a multivariate EDA might produce positive performance.

8.5.3.1 *Maintaining Cliques - The Basis of Multivariate*

Before we can consider whether a multivariate approach could be applied to the later stages of the TEDA process we need to first of all ask to what extent cliques are lost before this stage is reached. If the information necessary to build cliques is lost as diversity is lost in the population then it may be difficult to rediscover them and so applying a multivariate

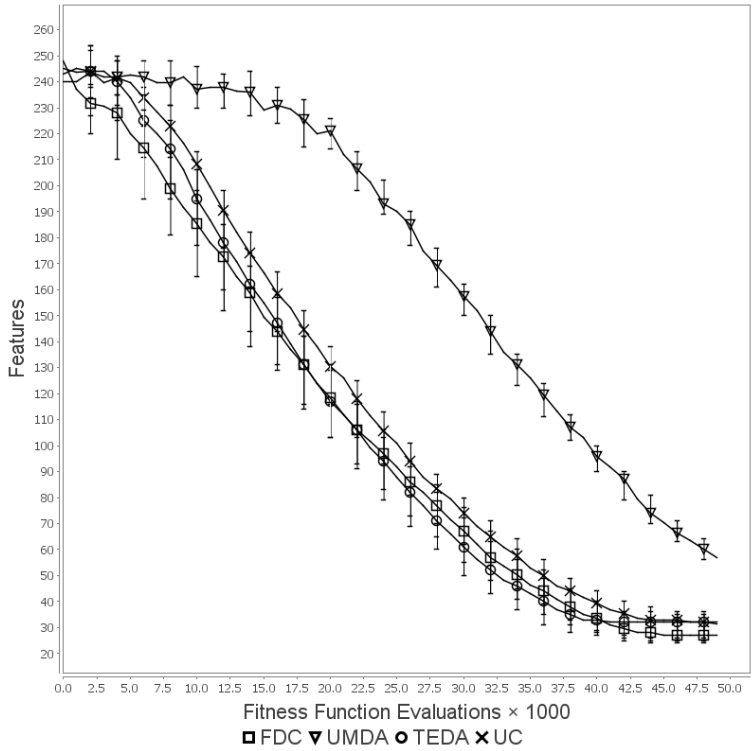

Figure 8.18: Artificial Problem with 500 Features - Control Points (Features)

approach to the later phases of the TEDA process may be futile. For this reason, in Figure 8.19 we present the number of cliques that each algorithm was able to maintain intact during the course of the test. We have plotted on the Y axis the number of complete cliques found in the fittest solution in the population at each generation. This is out of a total of 12 cliques.

It appears from these results as though UMDA is much more successful than TEDA at maintaining cliques and that a standard GA using Uniform Crossover is better than FDC at maintaining cliques. TEDA is, in turn, more effective than a standard GA. It appears to be the case that targeting is harmful to the maintanance of cliques and that EDA based approaches are better than GA based approaches.

It is interesting that UMDA is so successful at maintaining cliques but the accuracy of the solutions that it finds are lower. As mentioned, the solutions found by UMDA are larger than those found by other approaches. It is possible that, by not driving towards smaller solutions, cliques that are only partially discovered are not removed. These partial cliques are actually harmful to the accuracy of the feature set, as mentioned it is this feature that makes the problem deceptive. Keeping them therefore reduces the fitness of solutions discovered but increases the chances that the entire clique will be discovered later. It seems that overall, losing accuracy due to having partial cliques and gaining accuracy due to having more intact cliques balance each other out when comparing the solutions that UMDA ultimately discovers to those that TEDA discovers, with the result being that both find solutions of a similar accuracy.

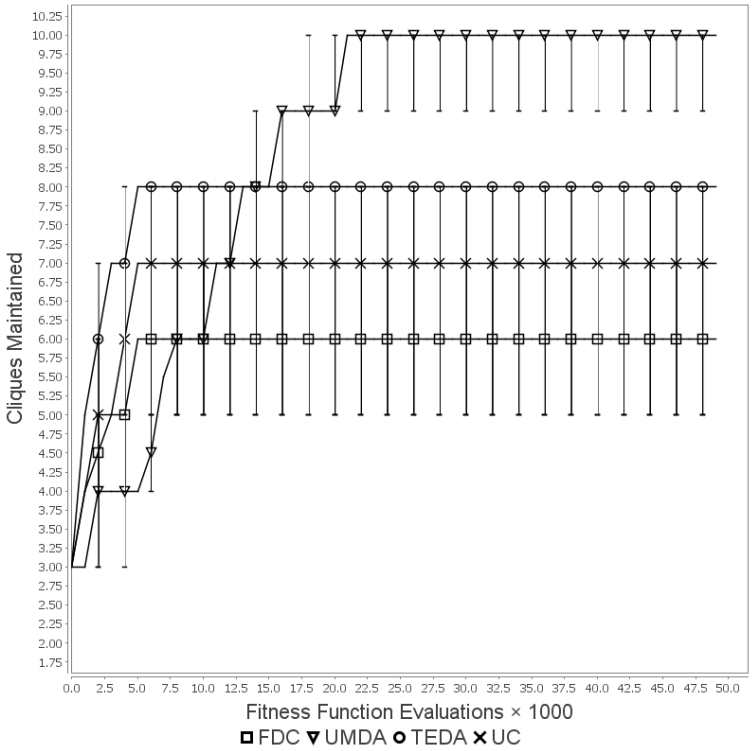

Figure 8.19: Artificial Problem with 500 Features - Cliques Maintained

UMDA's worse performance than TEDA in terms of combined fitness may be due to incomplete cliques but may also be due to its ability to remove redundant noise features.

This also provides a clue as to why the solutions ultimately found by TEDA were worse in the Madelon problem presented in the previous chapter than those found by UMDA. If TEDA loses partially discovered cliques more frequently than UMDA and UMDA later manages to complete these cliques then TEDA will produce solutions of a lower accuracy than UMDA.

The result of this finding is to cast some doubt on our assertion that it may be possible to introduce a multivariate technique to find the remaining cliques after TEDA has reduced the feature set as TEDA may lose the information needed to build these remaining cliques. It should be noted, however, that it is only UMDA that outperforms TEDA in this respect. TEDA still performs better than the standard GA and, significantly, performs better than FDC by a more substantial margin. TEDA is, therefore, better at maintaining cliques than any other technique that has been shown to quickly reduce the size of the feature set. There is a trade off in terms of how well all of these techniques are able to maintain cliques but TEDA does seem to be the best approach explored so far to reduce the size of the feature set.

8.5.3.2 *Reducing the Feature Set Size*

We have established that TEDA is able to identify small solutions and so it seems logical to conclude that it will be able to reduce the feature set size to a size where a multivariate approach can be applied. This, however, may not be the case as the number of features in

the fittest solution or in the average solution is not the only consideration. In order to build an accurate multivariate model we need to use all the information available in the breeding pool. This consists of 50% of the population in most of the tests which we have run here. We therefore need to establish the size of the complete set of features found across all solutions in the breeding pool as opposed to the size of just the fittest individual in the population. Even if the whole breeding pool is of similar size to the fittest solution, which itself is something that we cannot assume, there may be still sufficient diversity in the population for the solutions in the breeding pool to cover a set of features that is too large to be used by a multivariate algorithm.

The results in this section show how the number of features found across all solutions in the breeding pool changes over time. These results were obtained from the TEDA algorithm applied with the same parameters and to the same 500 feature problem as in the previous section. In each graph we plot the size of a set of available features F where $F = \bigcup_{i=0}^{|B|} featuresIn(B_i)$, the union of every feature set in B. B, in this case, is the fittest 50% of the population. In the first set of results shown in Figure 8.20, TEDA with a population of size 1000 and all the same parameters as used previously in this section was used.

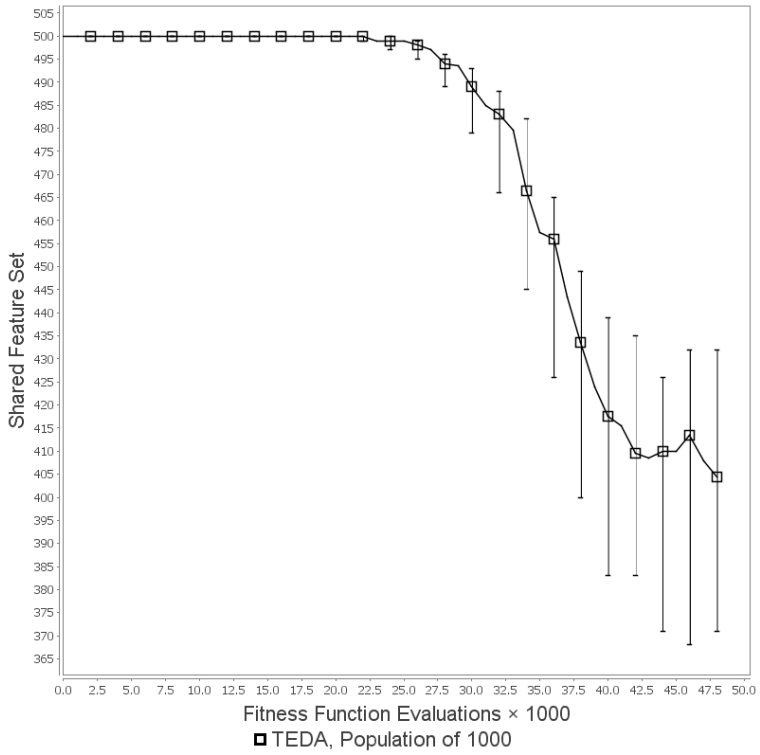

Figure 8.20: Artificial Problem with 500 Features - Population of 1000 - Shared Features

As can be seen, the number of shared features reduces from the initial 500 to around 400. This is probably not a significant enough reduction to enable a multivariate EDA to be applied. It was observed when exploring the transitioning process (see Section 5.4.5) that TEDA does

not lead to a dramatic loss of diversity and so it appears to be the case that the fitter solutions in the population still cover a large proportion of the available features.

8.5.3.3 *Reducing the Feature Set Size through a Smaller Population*

It it possible that, with a smaller population, the total number of features covered will be reduced to an acceptable level. TEDA was initially developed with a population of 100. This has been increased to 1000 for the tests in this section because it was established that this improves the effectiveness of multivariate algorithms. For this reason, in the following tests TEDA with a population size of 100 was applied to this problem. Through these tests we attempt to answer, first of all, whether with a smaller population the number of features covered will be reduced to a manageable level and secondly, whether with a smaller population the algorithm will still maintain a certain amount of cliques intact and will not simply follow the deceptive gradient and quickly drive towards tiny solutions and lose all information that a multivariate EDA would need to build complete cliques, as seen with the k-trap problem.

To answer the first question the results in Figure 8.21 show how many features can be found across B when TEDA was run with a population of size 100. B is, as with the tests above, the set containing the fittest 50% of individuals in the population. To answer the second question the results in Figure 8.22 show a comparison of how well TEDA performs on this problem with a population size of 100 and a population of 1000 and the results in Figure 8.23 show the number of cliques that these two configurations of TEDA are able to maintain. In both of these graphs results are plotted against the number of evaluations completed.

What we can see from these results is that with a population of 100 TEDA may be slightly less effective at maintaining cliques but it is able to find solutions of a similar accuracy to those found using a larger population. In fact, using a smaller population may decrease the time taken to find accurate solutions.

While reducing the size of the population has little effect on performance it does enable TEDA to effectively reduce the number of features that need to be considered to the extent that a multivariate algorithm could be applied. With a smaller population the number of features in the breeding pool drops to around 100 at generation 60 and then stabilizes. This transition appears to be clear and well defined and so marks a clear point at which a multivariate algorithm could be introduced.

8.6 TOWARDS MULTIVARIATE TEDA

In the previous section we have established that univariate TEDA can perform well on a problem that is known to be multivariate but that there is room for improvement and we have speculated that potential improvements could come from introducing a multivariate

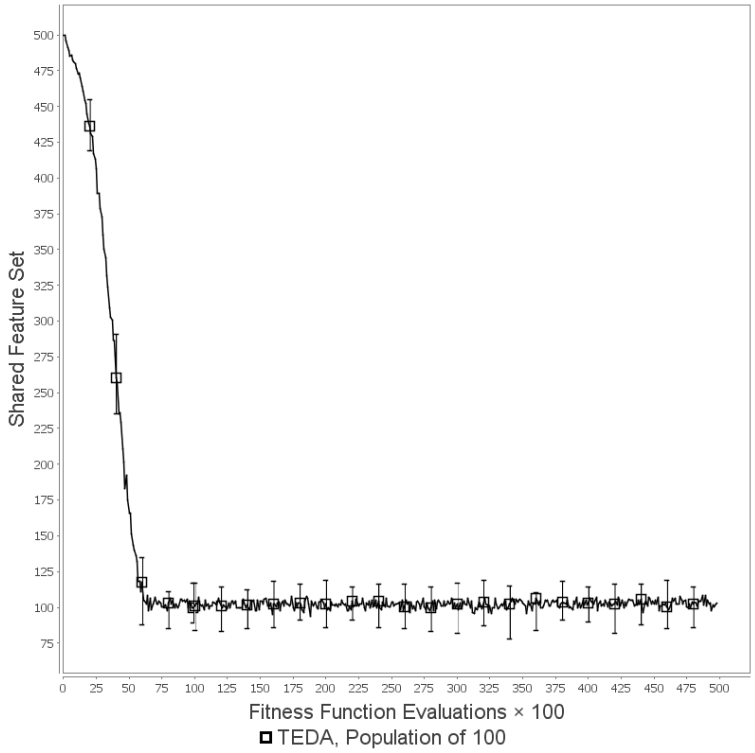

Figure 8.21: Artificial Problem with 500 Features - Population of 100 - Shared Features

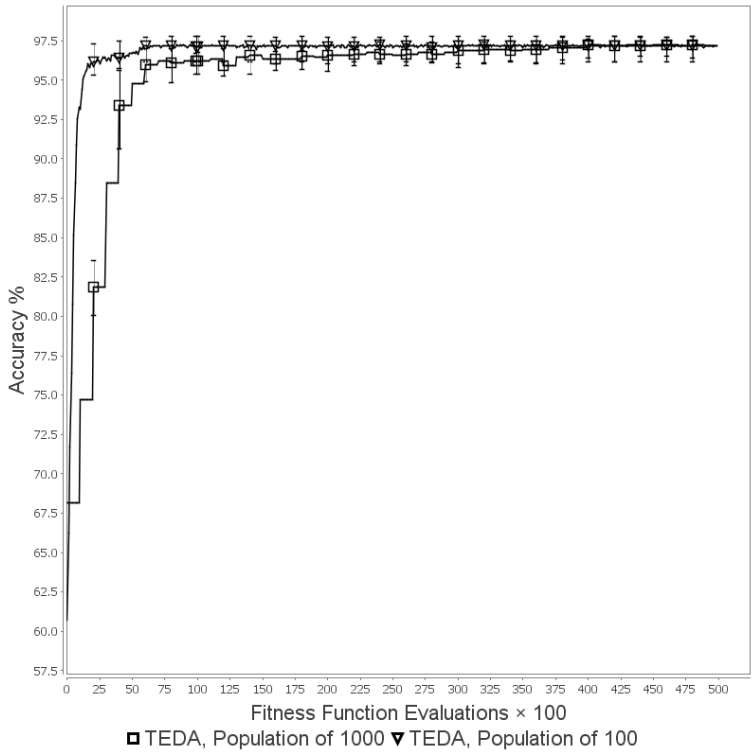

Figure 8.22: Artificial Problem with 500 Features - Population of 100 - Accuracy

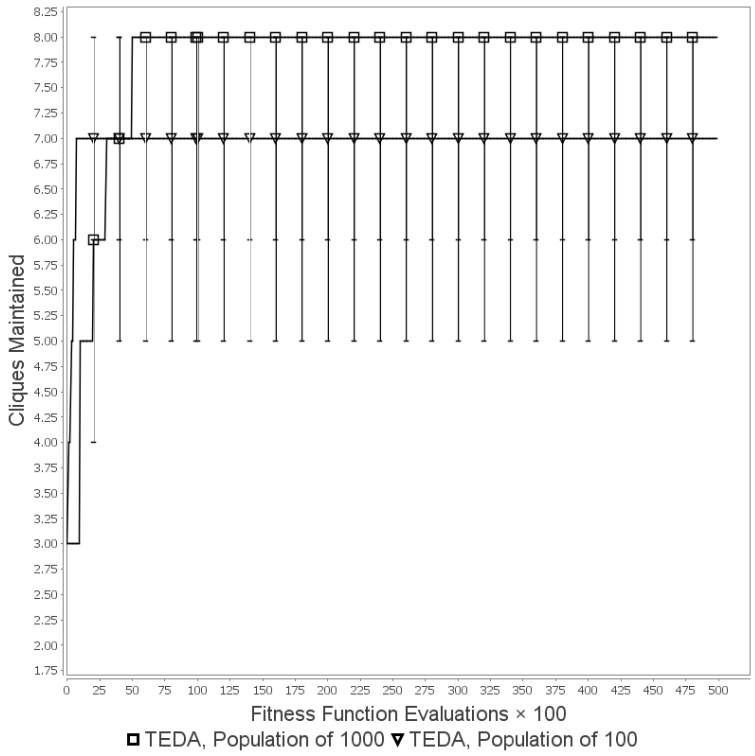

Figure 8.23: Artificial Problem with 500 Features - Population of 100 - Cliques Maintained

algorithm after TEDA has discovered a set of good features of a manageable size. To turn TEDA into a multivariate EDA we suggest that the following steps would need to be carried out:

- We would need to develop a method for establishing when the set of features which our algorithm will choose from, those found in solutions in the breeding pool, has been reduced to a manageable level. This may consist of a crude threshold based method. As it appears that the feature set size stabilizes after a period of time we could also use a method based on some measure of variance of feature set size over time.

- We would need to establish whether targeting should take place when TEDA has transitioned into a multivariate EDA. If a multivariate EDA is simply used in the same way that UMDA is in the current implementation of TEDA then targeting will be used throughout the TEDA process. In any case, targeting may be important throughout the test in order to enable TEDA to correct for driving the size of solutions too far in one direction in the early part of the test, as was the case in the routing problem. However, when multivariate EDAs are used targeting introduces some complications. The current process of selecting control points from S_{all} and then S_{some} in a random order would need to be adapted for algorithms where the order in which variables are selected is an integral part of the algorithm. In EBNA, for example, variables are selected in an ancestral order of dependency to ensure that a decision is made for each variable only after the variables upon which it is dependant have been set. As modifying this process

may damage the performance of multivariate EDAs it may be sensible to only apply multivariate EDAs after the evidence suggests that the targeting has already discovered the optimal number of control points and to then switch off the targeting process so that the multivariate EDA may be applied unmodified.

- Although TEDA should use a smaller population we could explore whether some method of increasing the size of the population after the feature set size has been reduced, possibly by randomly initializing a new set of individuals with this reduced set of features, will improve the performance of the multivariate stage of the process.

- We would need to decide on a multivariate EDA to incorporate into TEDA.

The final point, the choice of a suitable multivariate EDA, is likely be a difficult decision as this may depend on the scalability of the EDA used in relation to the size to which TEDA is able to reduce the search space. It may also depend on the nature of the problem domain and factors such as whether cliques are likely to overlap, the likely order of cliques and whether relationships are likely to be bi-directional or uni-directional. The first issue, scalability, may influence whether a bivariate or a higher order multivariate EDA should be used. Bivariate EDAs are recommended for variable sets of high cardinality as the number of dependencies that they model is lower than in multivariate EDAs [5]. They may, therefore, be more suitable in situations where the size to which TEDA is able to reduce the search space is relatively large. It may also be possible to implement a switch allowing either bivariate or higher order multivariate behaviour depending on the size of the search space. The presence of overlapping cliques may influence our choice as ECGA, for example, divides the search space up into discrete cliques and so therefore does not allow for overlapping cliques. The issue of directionality determines which underlying graph structure should be used. Graphs define how variables are connected and may be either directed or undirected. A well known class of undirected graph is the Markov network while Bayesian networks are a popular example of a directed graph. Because undirected graphs are efficient at modelling problems which directed graphs are inefficient for and vice versa it is important that an EDA based on Bayesian networks, such as EBNA, is used for problems requiring a directed graph whereas EDAs based on Markov networks are used for problems requiring undirected graphs [117].

The targeting process will also complicate this decision. If, as explained above, it is decided that targeting should be maintained when a multivariate EDA is introduced then it may be prudent to initially establish which multivariate EDAs still perform well when the order of control point selection is modified to allow for the targeting process to take place.

In general it appears as though the range of factors involved means that a set of recommendations for how different EDAs may be incorporated into TEDA as well as recommendations as to which problems these will be useful for may be preferable to choosing a single EDA. Hooker argures that empirical research into metaheuristics should focus on establishing the

relationships between problem features and algorithms rather than trying to find a typical problem for each domain and a single canonical algorithm that performs best throughout this domain [67]. Hooker's approach may be needed to answer this particular question.

However, the initial empirical evidence suggests that, for the K trap problem at least, EBNA performs significantly better than ECGA. EBNA also appears to perform well for this problem and for the multivariate FSS problem, as well as having already been shown to perform well on feature selection problems by Inza [69]. Because of this, in the FSS domain EBNA may at least provide a starting point for further investigation.

8.7 SUMMARY

In this chapter we have tested TEDA on problems that are highly multivariate. Both problems on which TEDA was tested, the K-Trap problem and an artificial FSS problem, were chosen specifically to be multivariate to the extent that they are deceptive and, when a solution only contains a subset of the genes within a clique, performance is actually worse than when a solution contains no members of the clique. The purpose of this chapter was to establish how robust the existing univariate TEDA method is in the face of multivariate problems and to begin to discuss how TEDA might be developed into a multivariate technique and whether this would be useful.

In the K-trap problem TEDA was shown to perform better than multivariate algorithms and other univariate algorithms with a large population and worse with a small population. In both cases multivariate algorithms performed better than untargeted univariate algorithms and FDC. This was due to the nature of the problem, in that a solution with all bits set to true is also an ideal solution. This reflects positively on TEDA but it was established that the more complex artificial classifier problem was needed to fully test TEDA.

In the artificial classifier problem we have shown that multivariate techniques outperform all other techniques with a large population but that they need this large population to perform at their optimum. We have also shown that TEDA is effective at reducing the dimensionality of the problem while still maintaining some linkage information, but that it only effectively reduces the dimensionality of the problem when using a smaller population. In the example explored here the number of features found in breeding parents was reduced from 500 to 100, greatly reducing the number of networks that need to be assessed to, for example, decide on a bayesian network for use by an algorithm such as EBNA. The fact that most cliques appear to remain intact suggests that the dimensionality reduction is driven by the removal of useless features, as is desirable. TEDA does, in fact, lose less intact cliques than FDC and a standard GA using UC. However, there is some loss in features which are useful and are lost before the cliques which they depend upon are discovered. Whether this balance would be different in

real world problems and whether an unacceptable loss of useful features would occur as part of the dimensionality reduction is something that will require further investigation.

Overall, we believe that future research in this area in which TEDA is used to reduce the dimensionality of the problem before applying a multivariate technique may produce a successful multivariate algorithm capable of solving complex problems with large feature sets.

9

CONCLUSION

This thesis has presented a new evolutionary algorithm, the Targeted Estimation of Distribution Algorithm (TEDA). This algorithm introduced a novel transitioning technique that caused it to progress from a Genetic Algorithm (GA) before the population has converged into an Estimation of Distribution Algorithm (EDA) as it converges. It also uses a targeting technique with which it is able to predict the optimal number of control points to set in new solutions.

The intention was to produce an algorithm that would find higher quality solutions faster than traditional EDAs and GAs that do not use the transitioning or targeting techniques.

To establish whether this was the case, TEDA was tested on several different problems from different domains. Each of these problems share the concept of control points such that the total number of control points used in a given solution is of comparable importance to where these control points are placed. Each problem was also chosen to present a different challenge to TEDA. These problems consisted of:

- An optimal control problem concerning cancer chemotherapy scheduling where an ideal solution should be effective at removing cancerous tumours while minimising the number of days in which potentially harmful drug treatment should be applied.
- A route finding problem where the aim was to find a route across a landscape that minimises both the changes in height across the route and the number of direction changes.
- Feature Subset Selection (FSS) problems.
- High order multivariate problems, including an artificial FSS problem and the K-Trap Problem.

9.1 GENERAL COMMENTS

TEDA was most successful on the routing problem, where it offered improvements over all alternatives on all versions of the problem. On some feature selection problems it was able to find good solutions faster than any of the other approaches. This was especially notable in situations where traditional EDAs were unable to optimise for an extended period of time at the start of the test. However, in some cases traditional EDAs ultimately found fitter solutions than TEDA.

On the chemotherapy problem TEDA performed to a similar standard to traditional EDAs but ultimately found fitter solutions than standard GAs including one using FDC. Targeting did not have an effect on this problem.

Although attempts to establish overall rules from a limited set of case studies must be treated with caution, there are a number of tentative conclusions that can be made from these results. Testing on a wider range of problems will confirm whether or not these generalise.

When compared against traditional EAs and GAs, these standard techniques are both able to find the ideal number of control points in problems where the available range is small and where there is no deception involved in choosing this number. This is the case in the chemotherapy problem, where almost all algorithms are able to reduce the number of interventions in solutions from a maximum of 100 to around 10 without the need for targeting. However, some problems are more difficult, such as the Arcene feature selection problem where the ideal number of features is also only around 10 despite the fact that 10,000 features are available or the routing problem where the deceptive nature of the problem drives algorithms to initially overshoot the ideal number of routers and find solutions that are too small. For these problems, both transitioning and targeting appear to be needed to achieve optimal performance. Without transitioning a standard EDA is deceived by the level of noise in these problems and prematurely converges whereas a standard GA is unable to effectively exploit the landscape. With transitioning this situation is improved and further improvement is produced when targeting is used.

As the initial motivation for the transitioning process was that the resulting algorithm should improve upon the performance of previous work using targeting, it is important to consider in what situations TEDA performed better than FDC and why. FDC appears to be less able to solve problems with a multivariate element than TEDA. This can be seen in its performance on the classification problems, especially Madelon which was specifically designed to have a multivariate aspect, on the K-Trap problem and on the classifier problem that was specifically designed to be highly multivariate and deceptive. In other difficult problems, such as the routing problem, TEDA is able to find better solutions than FDC. It seems likely that the higher level of exploitation achieved by TEDA due to its transitioning process enables it to perform better than FDC on these hard problems. For the easier chemotherapy problem FDC still performs well.

In general, we can say that in no situation was TEDA observed to be unable to solve a problem and so TEDA appears to be a robust approach. We would suggest that there is little risk in applying TEDA to any problem where targeting may be beneficial, and in some situations TEDA may offer benefits over other approaches.

In this section the points outlined above are discussed in more detail and we discuss the challenges for TEDA that each problem introduced.

9.2.1 *The Chemotherapy Problem*

The chemotherapy problem (see Chapter 3.1) was the problem through which an initial version of TEDA was developed, although the conclusion of this chapter was a version of TEDA using a different method of transitioning to the other chapters. Overall, in terms of the standard version of TEDA as used in other chapters, we can say of this problem that TEDA is not an ideal algorithm. It is able to discover good solutions and so can be applied to this problem, but it is slower at doing so than FDC and standard EDAs (see Section 5.28). It also can be made to perform slightly better than FDC and identical to UMDA through tuning of the transitioning process, as demonstrated by the performance of the transitioning technique that was in fact used (see Section 5.5).

The overall conclusion for this problem is that targeting is not necessary and that a standard EDA is the best performing approach. When the test parameters were slightly changed, through a lowering of the mutation rate and decreasing the tournament size to match the parameters with which FDC was developed [48], it was determined that targeting can be advantageous under certain conditions but that under these conditions FDC, not TEDA is the most suitable approach for this problem. The reason why TEDA does not appear to be necessary for this problem is probably because targeting is a relatively trivial task. The range of possible intervention numbers, 1 to 100, is relatively small and targeting is a simple process of reducing the number of interventions until the ideal number is reached. For this reason non targeted methods are able to find the ideal number of interventions without the need for explicit targeting.

9.2.2 *The Routing Problem*

Through the routing problem (see Chapter 6), it was demonstrated that TEDA is capable of performing significantly better both than FDC and untargeted EAs. This applies both when a penalty is used to penalise longer routes and when it is not used. This problem is harder than the chemotherapy problem as initially the best routes are simply those that use as few routers as possible. This makes the problem deceptive and creates a situation in which most algorithms simply drive towards as small solutions as possible and only TEDA is able to recover

from this situation, drive the number of routers up and find routes that are optimal and that contain the ideal number of routers.

Both the TEDA transitioning process and the targeting aspect have a role to play in this positive performance. Unlike in standard EDAs, the early explorative GA phase of the TEDA process does not lead to the loss of solutions that, while not the fittest in the population, may already contain information required to follow the contours of the landscape. As TEDA transitions into an EDA the selection pressure becomes sufficiently high to ensure that, when solutions which do follow the contours of the landscape begin to emerge, these solutions are selected and the EDA probability modelling aspect ensures that patterns within these solutions are exploited. Targeting ensures that TEDA learns the true optimal size of solution from these fitter solutions and so recovers from the situation in which the population consists primarily of overly small solutions, thus increasing the chance that more optimal solutions that follow the contours of the landscape will be discovered. The benefit of targeting alone was small but the combination of these features have made TEDA an effective algorithm for this problem.

9.2.3 *Feature Subset Selection Problem*

As detailed in Chapter 7, TEDA was tasked with finding small feature sets in three very large and noisy feature subset selection problems. It was tested against both EAs and non evolutionary FSS techniques such as Forward Selection (FS) and filter methods. On all three problems TEDA found solutions that were able to classify with a high degree of accuracy before the untargeted evolutionary approaches while the non evolutionary approaches either proved to be unsuitable for such large problems or unable to find feature sets that were as effective as those found by TEDA. On two of the three problems TEDA also performed better than FDC and on no problem did it perform worse than FDC.

TEDA's success was attributed to the fact that, by initially driving towards very small solutions that were quick to classify it was able to explore a large number of solutions in less time than the other approaches. On the Arcene problem it was also noted that reducing the size of solution dramatically was essential for effective optimization to take place at all and that techniques that do not use targeting were effectively stationary until a significant number of good quality smaller solutions had been generated by chance. TEDA ensured that these small solutions were discovered quickly and so prevented this delay, finding good quality solutions long before the other approaches.

TEDA is, however, limited in this area. In some situations, such as when a different fitness measure was used or when a different classifier was used, it was less effective. In the Madelon problem, although faster at finding good solutions than an untargeted EDA, TEDA ultimately did not find solutions of as high a quality as UMDA. In this problem reducing the size of feature set was less helpful. It appeared to be the case that good solutions in this problem

contained a greater proportion of the total feature set than in the other problems and this may be necessary due to the multivariate nature of the problem. TEDA did, however, still manage to find solutions of a reasonable fitness in less time than the alternatives.

Overall we would suggest that TEDA is likely to be useful in large and noisy FSS problems and where obtaining a good solution quickly instead of obtaining the best possible solution is the main priority.

9.2.4 *The Multivariate Problem*

Chapter 8 contained an analysis of the performance of standard TEDA on problems that were known to be multivariate and deceptive as well as a discussion of how TEDA might be developed into a multivariate algorithm. TEDA was tested on the K-Trap problem as well as on a FSS problem specifically designed to be multivariate. It was shown on the K-trap problem that if TEDA does select solutions that contain building blocks useful to successfully solve this problem then it will quickly find the ideal solution. However, this was judged to be due to an artificial aspect of this problem that the ideal solution also happened to be the largest possible solution in terms of the number of positive genes. This was, however, a capability that only TEDA showed among the two targeted approaches tested. FDC, it seemed, was not able to do this.

In the more complex classifier problem the best performing approaches were multivariate EDAs, performing better than TEDA and the other univariate approaches. Generally, multivariate approaches performed well on both problems. It was shown that some multivariate problems were too large for multivariate EDAs but that TEDA was able to reduce the size of these problems to more manageable sizes while not losing information needed to build multivariate models. TEDA was, in fact, shown to be less liable to lose this information than other univariate approaches such as FDC.

It was suggested that an approach that uses TEDA to reduce the size of a problem and then uses a multivariate EDA may prove to be an effective technique. One issue discussed was the fact that multivariate EDAs appear to require larger populations than univariate EDAs and yet TEDA requires a small population to effectively reduce the size of the problem.

9.3 FUTURE WORK

There are a number of areas that could be researched in order both to expand the capabilities of TEDA and to strengthen arguments already made regarding TEDA. The main ways in which TEDA could be expanded include:

- Developing TEDA into a multivariate algorithm.

- Exploring how TEDA might be applied to multi-objective problems.
- Expanding the range of genotypic encoding systems that TEDA is able to work with. For example, one interesting area to investigate would be whether TEDA could be adapted to work with the tree structures that are used in genetic programming.

In addition, the following studies could be carried out which may support our empirical observations about the performance of TEDA.

- A theoretical study could be carried out to predict the behaviour of TEDA. It was mentioned in Section 2.8 that the rate of convergence of EDAs and the probability of premature convergence has been predicted theoretically. As TEDA is designed to have a lower probability of prematurely converging than standard EDAs, and empirical evidence supports this, it would be useful to carry out a theoretical study to be able to confirm whether this is in fact the case.
- The efficiency of TEDA could be analysed in terms of time complexity [116]. The performance of TEDA so far has been analysed in terms of the number of fitness function evaluations required to solve a problem and, in the case of the FSS problem, the time taken to perform these evaluations. This is useful if we assume that fitness function evaluations are the most costly part of the evolutionary process, however the time taken to perform optimization should also be taken into account. As the actual length of time that this process may take will depend on the hardware used and on implementation details a formal analysis of the complexity of TEDA would be a more reliable method of doing this.

In general, we should strive to develop a better understanding of which problems TEDA is suitable for and why, in order to make more informed recommendations as to where it could successfully be employed. This will involve testing TEDA on a wider range of problems from many different domains.

In this section we will focus on two of these objectives, developing TEDA into a multivariate algorithm and producing a version of TEDA able to work in a multi-objective domain.

9.3.1 *Multivariate TEDA*

As mentioned, it has already been observed that TEDA may potentially be built into an multivariate algorithm. One study examined the performance of TEDA in a multivariate domain and lead to three main questions that future research could address:

- Which multivariate EDA would be most effective in the TEDA framework?
- How should a multivariate EDA be integrated into TEDA? Our intention is that a system should be developed for measuring when targeting has reduced the set of control points

to a point where a multivariate algorithm could be used and then transitioning from TEDA to this EDA. We would need to develop a system for detecting when this point has been reached.

- How might targeting be introduced into multivariate EDAs? This relies on the additional question of whether it is in fact beneficial to introduce targeting into multivariate EDAs at all. If multivariate EDAs are only applied once the set of available control points has been reduced to a point where most of the noise has been removed and if reaching this point was the main purpose of targeting then further targeting after the multivariate EDA has been deployed is not necessarily important and may be destructive.

We should test multivariate TEDA on a range of problems. FSS problems are a key example problem on which to test this technique and our set of problems should include problems known to be multivariate and should include both small problems where multivariate TEDA could be compared to multivariate techniques that do not use targeting and much larger problems where we could assess whether multivariate TEDA is in fact scalable to these larger problems. Our initial exploration has only considered one artificial FSS problem and in future we should investigate how multivariate TEDA performs on a diversity of real world problems. We should also research which, if any, multivariate EDAs are able to perform well on large problems to establish a baseline against which to compare TEDA.

9.3.2 *Multi-Objective TEDA*

As explained in Section 2.11, in many real world problems there are multiple, possibly contradictory objectives. TEDA is often used for problems that have two objectives, that of maximising some fitness score while minimizing a cost function that is related to the number of control points. It therefore seems logical to investigate whether it can be applied to problems with more objectives. It would be interesting to adapt it to work as a multi-objective algorithm and to compare it to other multi-objective approaches.

As these problems feature multiple fitness values, in order to adapt TEDA into an algorithm for multi-objective problems a decision will need to made with regard to which fitness value should be used in each stage of the TEDA process. We can identify 5 places in which a fitness value is used:

1. Selection of the selection pool
2. Selection of the breeding pool
3. Defining the fittest and least fit individual in the breeding pool for intervention targeting
4. Intervention targeting (applying the FDC rule)

5. Building the EDA probability distribution

The two main options for ensuring that all objectives are taken into account are to find some method of switching between the different fitness values or to combine these values into a single scaler value which can then be used as the fitness value in every situation.

If we decide to switch between different fitness values then this process could be entirely random, it could be stochastic or it could be deterministic. The stochastic approach would involve assigning to each objective a set of weights which will determine the probability that its corresponding fitness value will be used for each phase of the TEDA process. The deterministic approach may consist of allocating objectives to a specific phase of the TEDA process or switching between them according to a rota. Which of these methods will be most effective depends on what the objectives are and whether there is a natural relationship between objectives and phases of the TEDA process.

If we decide to combine the fitness values associated with different objectives into a single scaler value then the two main options are to use the Pareto rank of a solution as its fitness value or to simply take the sum of the different fitnesses multiplied by a vector of weights. These options are explained in more detail in Section 2.11 and in that section it was established that a Pareto based method is usually preferable due to the difficulty in choosing a set of weights for the weighted sum system and the assumption that this system makes with regards to the linearity of the different fitness values.

One algorithm of interest here is NSGA2. The NSGA framework, as described in Section 2.11.1 uses a Pareto based fitness that also takes into account a measure of sparsity. This ensures that solutions are well spread out across the Pareto front. Given that maintaining diversity is a key concern in TEDA and it is important that the early phase of TEDA remains explorative, this is a useful quality of NSGA and so, if the fitness values produced by NSGA were incorporated at some or all points in the TEDA process, this may produce an effective algorithm.

NSGA2 adds an additional property that could also be useful for TEDA, its use of an elitist archive to ensure that the best solutions are not lost. As explained in Section 2.11.3, in this method the highest ranking Pareto fronts are placed into an archieve that is then combined with a newly generated population and the combined population is used to produce new solutions.

BIBLIOGRAPHY

[1] C. W. Ahn and R. Ramakrishna. A genetic algorithm for shortest path routing problem and the sizing of populations. *IEEE Trans. Evolutionary Computation*, 6:566–579, 2002.

[2] C. W. Ahn and R. S. Ramakrishna. Elitism-based compact genetic algorithms. In *IEEE Trans. Evol. Comput.*, volume 7:4, pages 367–385, 2003.

[3] Ethem Alpaydin. *Introduction to machine learning*. MIT press, 2004.

[4] Neculai Andrei. Modern control theory, a historical perspective. 2005.

[5] Rubén Armañanzas, Iñaki Inza, Roberto Santana, Yvan Saeys, Jose Luis Flores, Jose Antonio Lozano, Yves Van de Peer, Rosa Blanco, Víctor Robles, Concha Bielza, et al. A review of estimation of distribution algorithms in bioinformatics. *BioData Mining*, 1:6, 2008.

[6] T. Back, U. Hammel, and H.P. Schwefel. Evolutionary computation: Comments on the history and current state. *Evolutionary computation, IEEE Transactions on*, 1(1):3–17, 1997.

[7] James E Baker. Adaptive selection methods for genetic algorithms. In *Proceedings of the 1st International Conference on Genetic Algorithms*, pages 101–111. L. Erlbaum Associates Inc., 1985.

[8] S. Baluja. Population-based incremental learning: A method for integrating genetic search based function optimization and competitive learning. Technical Report CMU-CS-94-163, Computer Science Department, Carnegie Mellon Univ., 1994.

[9] Richard Bellman. On a routing problem. Technical report, DTIC Document, 1956.

[10] T. Blickle and L. Thiele. A mathematical analysis of tournament selection. In *Proc. of the Sixth Int. Conf. (ICGA95)*, 1995.

[11] Avrim L Blum and Pat Langley. Selection of relevant features and examples in machine learning. *Artificial intelligence*, 97(1):245–271, 1997.

[12] T Bo and I. Jonassen. New feature subset selection procedures for classification of expression profiles. *Genome biology*, 3(4):1–17, 2002.

[13] J. De Bonet, C. Isbell, and P. Viola. Mimic: Finding optima by estimating probability densities. *Advances in Neural Information Processing Systems*, 9, 1996.

[14] B. Boser, I. Guyon, and V.N. Vapnik. A training algorithm for optimal margin classifiers. In D. Haussler, editor, *Proc. COLT*, pages 144–152, 1992.

[15] P. Bosman and D. Thierens. An algorithmic framework for density estimation based evolutionary algorithms. Technical Report UU-CS-1999-46, Utrecht Univ., 1999.

[16] P. A. N. Bosman and D. Thierens. Numerical optimization with real-valued estimation-of-distribution algorithms. In *Scalable Optimization via Probabilistic Modeling*, pages 91–120. Springer, 2006.

[17] Remco R Bouckaert. Properties of bayesian belief network learning algorithms. In *Proceedings of the Tenth international conference on Uncertainty in artificial intelligence*, pages 102–109. Morgan Kaufmann Publishers Inc., 1994.

[18] A. E. Brownlee, J. A. McCall, Q. Zhang, and D. Brown. Approaches to selection and their effect on fitness modelling in an estimation of distribution algorithm. In *Evolutionary Computation, 2008. CEC 2008.(IEEE World Congress on Computational Intelligence). IEEE Congress on*, pages 2621–2628. IEEE, 2008.

[19] A. E. I. Brownlee, M. Pelikan, J. A. W. McCall, and A. Petrovski. An application of a multivariate estimation of distribution algorithm to cancer chemotherapy. In *Proc. of the Genetic and Evolutionary Computation Conf. (GECCO 2008)*. ACM Press, New York, NY, USA, 2008.

[20] Alexander EI Brownlee, John AW McCall, and Martin Pelikan. Influence of selection on structure learning in markov network edas: An empirical study. In *Proc. of the fourteenth int. conf. on Genetic and evolutionary computation*, pages 249–256. ACM, 2012.

[21] Wray L Buntine. Classifiers: A theoretical and empirical study. In *IJCAI*, pages 638–644. Citeseer, 1991.

[22] E. K. Burke, M. Hyde, G. Kendall, G. Ochoa, E. Ozcan, and R. Qu. Hyper-heuristics: A survey of the state of the art. *Journal of the Operational Research Society (to appear)*, 2010.

[23] Edmund K Burke, Graham Kendall, and Eric Soubeiga. A tabu-search hyperheuristic for timetabling and rostering. *Journal of Heuristics*, 9(6):451–470, 2003.

[24] Edmund K Burke, Steven Gustafson, and Graham Kendall. Diversity in genetic programming: An analysis of measures and correlation with fitness. *Evolutionary Computation, IEEE Transactions on*, 8(1):47–62, 2004.

[25] Edmund K Burke, Sanja Petrovic, and Rong Qu. Case-based heuristic selection for timetabling problems. *Journal of Scheduling*, 9(2):115–132, 2006.

[26] Edmund K Burke, Barry McCollum, Amnon Meisels, Sanja Petrovic, and Rong Qu. A graph-based hyper-heuristic for educational timetabling problems. *European Journal of Operational Research*, 176(1):177–192, 2007.

[27] Erick. Cantï¿½-Paz. Feature subset selection by estimation of distribution algorithms. In *Proc. of Genetic and Evolutionary Computation Conf.* MIT Press, 2002.

[28] C. C. Chang and C. J. Lin. Libsvm: a library for support vector machines. *ACM Trans. on Intelligent Systems and Technology (TIST)*, 2(3):27, 2011.

[29] S. Chen and T. Yu. Difficulty of linkage learning in estimation of distribution algorithms. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pages 397–404. ACM, 2009.

[30] S. H. Chen, M. C. Chen, P. C. Chang, and Y. M. Chen. Ea/g-ga for single machine scheduling problems with earliness/tardiness costs. *Entropy*, 13(6):1152–1169, 2011.

[31] Yi-Wei Chen and Chih-Jen Lin. Combining svms with various feature selection strategies. In *Feature Extraction*, pages 315–324. Springer, 2006.

[32] Ying-Ping Chen and Meng-Hiot Lim. Linkage in evolutionary computation. *Studies in Computational Intelligence*, 157:140, 2008.

[33] R. Chertov, S. Fahmy, and N. B. Shroff. A device-independent router model. *IEEE INFOCOM*, page 9, April 2008.

[34] Matej Črepinšek, Shih-Hsi Liu, and Marjan Mernik. Exploration and exploitation in evolutionary algorithms: a survey. *ACM Computing Surveys (CSUR)*, 45(3):35, 2013.

[35] M. Dash, H. Liu, and Manoranjan. Feature selection for classification. *Intelligent data analysis*, 1:131–156, 1997.

[36] K. Deb and D. Goldberg. *Analyzing deception in trap functions*. University of Illinois, Department of General Eng., 1991.

[37] K. Deb, S. Agrawal, A. Pratab, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. In *Proc. 3rd Int. Conf. Genetic Algorithms*, 2002.

[38] Kalyanmoy Deb. Multi-objective optimization. In *Search Methodologies*, pages 273–316. Springer, 2005.

[39] Agoston E Eiben, Zbigniew Michalewicz, Marc Schoenauer, and JE Smith. Parameter control in evolutionary algorithms. In *Parameter setting in evolutionary algorithms*, pages 19–46. Springer, 2007.

[40] R. Etxeberria and P. Larranaga. Global optimization using bayesian networks. In *Second Symposium on Artificial Intelligence (CIMAF-99)*, pages 332–339, 1999.

[41] F. Fleuret. Fast binary feature selection with conditional mutual information. *The Journal of Machine Learning Research*, 5:1531–1555, 2004.

[42] L.J. Fogel. *Autonomous automata*, volume 4. Industrial Research, 1962.

[43] A. Frank and A. Asuncion. UCI machine learning repository, 2010. URL `http://archive.ics.uci.edu/ml`.

[44] Fred Glover, Manuel Laguna, et al. *Tabu search*, volume 22. Springer, 1997.

[45] P. M. Godley, D. E. Cairns, and J. Cowie. Directed intervention crossover applied to bio-control scheduling. In *IEEE CEC 2007: Proc. of the IEEE Congress On Evolutionary Computation. IEEE press*, 2007.

[46] Paul M Godley, David E Cairns, Julie Cowie, Kevin M Swingler, and John McCall. The effects of mutation and directed intervention crossover when applied to scheduling chemotherapy. In *Proceedings of the 10th annual conference on Genetic and evolutionary computation*, pages 1105–1106. ACM, 2008.

[47] Paul Michael Godley. *Directed Intervention Crossover Approaches in Genetic Algorithms with Application to Optimal Control Problems*. PhD thesis, 2009.

[48] P.M. Godley, D.E. Cairns, J. Cowie, and J. McCall. Fitness directed intervention crossover approaches applied to bio-scheduling problems. In *Symp. on Computational Intelligence in Bioinformatics and Computational Biology*, pages 120 –127. IEEE, 2008. doi: 10.1109/CIBCB.2008.4675768.

[49] D. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, 1989.

[50] David E Goldberg and Kalyanmoy Deb. A comparative analysis of selection schemes used in genetic algorithms. *Urbana*, 51:61801–2996, 1991.

[51] David E Goldberg, Kalyanmoy Deb, Hillol Kargupta, and Georges Harik. Rapid, accurate optimization of difficult problems using messy genetic algorithms. In *Proceedings of the Fifth International Conference on Genetic Algorithms,(Urbana, USA)*, pages 59–64. Proceedings of the Fifth International Conference on Genetic Algorithms,(Urbana, USA), 1993.

[52] Jörn Grahl, Stefan Minner, and Franz Rothlauf. Behaviour of umda< sub> c</sub> with truncation selection on monotonous functions. In *Evolutionary Computation, 2005. The 2005 IEEE Congress on*, volume 3, pages 2553–2559. IEEE, 2005.

[53] Greffenstette. Optimization of control parameters for genetic algorithms. In *IEEE Trans. Syst. Man Cybern. SMC-16*, volume 1, pages 122–128, Jan/Feb 1986.

[54] Crina Grosan and Ajith Abraham. Hybrid evolutionary algorithms: methodologies, architectures, and reviews. In *Hybrid evolutionary algorithms*, pages 1–17. Springer, 2007.

[55] I. Guyon and A. Elisseeff. An introduction to variable and feature selection. *The Journal of Machine Learning Research*, 3:1157–1182, 2003.

[56] I. Guyon, S. Gunn, A. Ben-Hur, and G. Dror. Result analysis of the nips 2003 feature selection challenge. *Advances in Neural Information Processing Systems*, 17:545–552, 2004.

[57] M. A. Hall. *Correlation-based feature selection for machine learning*. PhD thesis, The University of Waikato, 1999.

[58] P. J. B. Hancock. An empirical comparison of selection methods in evolutionary algorithms. *Fogarty*, pages 80–94, 1994.

[59] Nikolaus Hansen and Andreas Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary computation*, 9(2):159–195, 2001.

[60] G. Harik, F. G. Lobo, and D. E. Goldberg. The compact genetic algorithm. In *IEEE Trans. Evol. Comput.*, volume 3, pages 287–297, November 1999.

[61] Georges Raif Harik. *Learning gene linkage to efficiently solve problems of bounded difficulty using genetic algorithms*. PhD thesis, The University of Michigan, 1997.

[62] G.R. Harik, F.G. Lobo, and K. Sastry. Linkage learning via probabilistic modeling in the extended compact genetic algorithm (ecga). *Studies in Computational Intelligence (SCI)*, 33:39–61, 2006.

[63] William E Hart, Natalio Krasnogor, and James E Smith. *Recent advances in memetic algorithms*, volume 166. Springer, 2005.

[64] Mark Hauschild and Martin Pelikan. A survey of estimation of distribution algorithms. Technical Report 2011004, Univ. of Missouri, 2011.

[65] Jennifer A Hoeting, David Madigan, Adrian E Raftery, and Chris T Volinsky. Bayesian model averaging. In *In Proceedings of the AAAI Workshop on Integrating Multiple Learned Models*, pages 77–83. Citeseer, 1998.

[66] J.H. Holland. *Adaptation in Natural and Artificial Systems*. Univ. of Michigan Press, Ann Arbor, MI., 1975.

[67] John N Hooker. Needed: An empirical science of algorithms. *Operations Research*, 42(2): 201–212, 1994.

[68] J. Horn, N. Nafpliotis, and D. E. Goldberg. A niched pareto genetic algorithm for multiobjective optimization. In *Proceedings of the First IEEE Conference on Evolutionary*

Computation, 1994. IEEE World Congress on Computational Intelligence.,, pages 82–87. IEEE, 1994.

[69] I. Inza, P. Larranaga, R. Etxeberria, and B. Sierra. Feature subset selection by bayesian networks based on optimization. *Artificial Intelligence*, 123(1–2):157–184, 2000.

[70] I. Inza, P. Larranaga, and B. Sierra. Feature subset selection by bayesian networks: a comparison with genetic and sequential algorithms. *Int. Journ. of Approximate Reasoning*, 27(2):143–164, 2001.

[71] K. A. De Jong. Genetic algorithms: A 10 year perspective. In J. Greffenstette, editor, *Proceedings of an International Conference on Genetic Algorithms and their Applications*, pages 169–177, Pittsburgh, July 1985.

[72] K. De Jong. *An analysis of the behavior of a class of genetic adaptive systems*. PhD thesis, Univ. Michigan, Ann Arbor, MI, 1975.

[73] J.M. Keller, M.R. Gray, and J.A. Givens. A fuzzy k-nearest neighbor algorithm. *IEEE Trans. on Systems, Man and Cybernetics*, 4:580–585, 1985.

[74] DE Kirk. *Optimal control theory- An introduction(Book on optimal control theory covering dynamic programming, calculus of variations, Pontryagin maximum principle and iterative techniques)*. 1970.

[75] R. Kohavi and G. H. John. Wrappers for feature subset selection. *Artificial intelligence*, 97 (1):273–324, 1997.

[76] Abdullah Konak, David W Coit, and Alice E Smith. Multi-objective optimization using genetic algorithms: A tutorial. *Reliability Engineering & System Safety*, 91(9):992–1007, 2006.

[77] J. R. Koza. *On the programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, Massachusetts, 1992.

[78] William H Kruskal and W Allen Wallis. Use of ranks in one-criterion variance analysis. *Journal of the American statistical Association*, 47(260):583–621, 1952.

[79] Solomon Kullback. The kullback-leibler distance. *The American Statistician*, 41(4):340–341, 1987.

[80] Frank Kursawe. A variant of evolution strategies for vector optimization. In *Parallel Problem Solving from Nature*, pages 193–197. Springer, 1991.

[81] C. Lai, M.J.T. Reinders, and L. Wessels. Random subspace method for multivariate feature selection. *Pattern Recognition Letters*, 27(10):1067–1076, 2006.

[82] Dario Landa-Silva and Edmund K Burke. Asynchronous cooperative local search for the office-space-allocation problem. *INFORMS Journal on Computing*, 19(4):575–587, 2007.

[83] Pedro Larranaga and Jose A. Lozano. *Estimation of distribution algorithms: A new tool for evolutionary computation*, volume 2. Springer, 2002.

[84] Manuel Lozano, Francisco Herrera, and José Ramón Cano. Replacement strategies to preserve useful diversity in steady-state genetic algorithms. *Information Sciences*, 178(23): 4421–4433, 2008.

[85] Samir W Mahfoud. Crowding and preselection revisited. *Urbana*, 51:61801, 1992.

[86] J. G. March. Exploration and exploitation in organizational learning. *Organization Science*, 2:71–87, 1995.

[87] T. Marill and D. Green. On the effectiveness of receptors in recognition systems. *Information Theory, IEEE Transactions on*, 9(1):11–17, 1963.

[88] R. Martin and K. L. Teo. *Optimal control of drug administration in cancer chemotherapy*. World Scientific Pub Co Inc, Singapore, 1994.

[89] J. McCall and A. Petrovski. A decision support system for cancer chemotherapy using genetic algorithms. In *Proc. of the Int. Conf. on Computational Intelligence for Modelling, Control and Automation*, pages 65–70. IOS Press, 1999.

[90] Brian McGinley, John Maher, Colm O'Riordan, and Fearghal Morgan. Maintaining healthy population diversity using adaptive crossover, mutation, and selection. *Evolutionary Computation, IEEE Transactions on*, 15(5):692–714, 2011.

[91] Zbigniew Michalewicz. *Genetic algorithms+ data structures= evolution programs*. springer, 1996.

[92] Heinz Mühlenbein. The equation for response to selection and its use for prediction. *Evolutionary Computation*, 5(3):303–346, 1997.

[93] Heinz Mühlenbein and Dirk Schlierkamp-Voosen. The science of breeding and its application to the breeder genetic algorithm (bga). *Evolutionary Computation*, 1(4): 335–360, 1993.

[94] G. K. Neumann and D. E. Cairns. Introducing intervention targeting into estimation of distribution algorithms. In *Proceedings of the 27th ACM Symposium on Applied Computing*, pages 220–225, 2012.

[95] F. Oppacher and M. Wineberg. The shifting balance genetic algorithm: Improving the ga in a dynamic environment. In *Proceedings of the Genetic and Evolutionary Computation Conference*, 1999.

[96] E. Ozcan, B. Bilgin, and E. E. Korkmaz. A comprehensive analysis of hyper-heuristics. *Intelligent Data Analysis*, 12(1):3–23, 2008.

[97] D.E. Pelikan, M. & Goldberg. Hierarchical boa solves ising spin glasses and maxsat. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2003)*, pages 1271–1282, 2003.

[98] Martin Pelikan and Heinz Mühlenbein. The bivariate marginal distribution algorithm. In *Advances in Soft Computing*, pages 521–535. Springer, 1999.

[99] J. Pena, V V. Robles, P. Larranaga, V. Herves, F. Rosales, and M. Perez. Ga-eda: Hybrid evolutionary algorithm using genetic and estimation of distribution algorithms. *Innovations in Applied Artificial Intelligence*, pages 361–371, 2004.

[100] A. Petrovski, S. Shakya, and J. McCall. Optimising cancer chemotherapy using an estimation of distribution algorithm and genetic algorithms. In *Proc. of the 8th Annual Conf. on Genetic and Evolutionary Computation*, pages 413–418, 2006.

[101] Andrei Petrovski, Bhavani Sudha, and John McCall. Optimising cancer chemotherapy using particle swarm optimisation and genetic algorithms. In *Parallel problem solving from nature-PPSN VIII*, pages 633–641. Springer, 2004.

[102] Petr Pošík. Preventing premature convergence in a simple eda via global step size setting. In *Parallel Problem Solving from Nature–PPSN X*, pages 549–558. Springer, 2008.

[103] Mitchell A Potter and Kenneth A De Jong. Cooperative coevolution: An architecture for evolving coadapted subcomponents. *Evolutionary computation*, 8(1):1–29, 2000.

[104] P. Pudil, J., Novovicova, and J. Kittler. Floating search methods in feature selection. *Pattern recognition letters*, 15(11):1119–1125, 1994.

[105] J. R. Quinlan. *C4. 5: programs for machine learning*, volume 1. Morgan kaufmann, 1993.

[106] I Rechenberg. *Evolutionstrategie - Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Formman-Holzboog, 1973.

[107] Christopher D Rosin and Richard K Belew. New methods for competitive coevolution. *Evolutionary Computation*, 5(1):1–29, 1997.

[108] P. Ross. Hyper-heuristics. In *Search methodologies*, pages 529–556. Springer, 2005.

[109] Y. Saeys, S. Degroeve, D. Aeyels, Y. Van de Peer, and P. Rouzï¿½. Fast feature selection using a simple estimation of distribution algorithm: a case study on splice site prediction. *Bioinformatics*, 19(suppl 2):179–188, 2003.

[110] J. D. Schaffer. Some experiments in machine learning using vector evaluated genetic algorithms. Technical report, Vanderbilt Univ., Nashville, TN (USA), 1985.

[111] Gideon Schwarz. Estimating the dimension of a model. *The annals of statistics*, 6(2): 461–464, 1978.

[112] S. Shakya. *DEUM: A Framework for an Estimation of Distribution Algorithm Based on Markov Random Fields.* PhD thesis, The Robert Gordon University, Aberdeen, 2006.

[113] Siddhartha Shakya and Roberto Santana. A review of estimation of distribution algorithms and markov networks. In *Markov Networks in evol. Computation*, pages 21–37. Springer, 2012.

[114] H. Shimodaira. A diversity control oriented genetic algorithm (dcga): Development and experimental results. In *Proceedings of the Genetic and Evolutionary Computation Conference*, 1999.

[115] S. Siegel and N. J. C. Jr. *Nonparametric Statistics for The Behavioral Sciences*. McGraw-Hill, New York, NY, USA, 1988.

[116] Michael Sipser. *Introduction to the Theory of Computation*. Cengage Learning, 2006.

[117] Padhraic Smyth. Belief networks, hidden markov models, and markov random fields: A unifying view. *Pattern recognition letters*, 18(11):1261–1268, 1997.

[118] W. Spears and K. A. De Jong. An analysis of multi-point crossover. In *Proc. of the Foundations of Genetic Algorithms Workshop*, 1990.

[119] M Srinivas and Lalit M Patnaik. Adaptive probabilities of crossover and mutation in genetic algorithms. *Systems, Man and Cybernetics, IEEE Transactions on*, 24(4):656–667, 1994.

[120] N. Srinivas and K. Deb. Multi-objective function optimization using non-dominated sorting genetic algorithms. *Evolutionary Computation*, 2 (3):221–148, 1995.

[121] H. Stoppiglia, G. Dreyfus, R. Dubois, and Y. Oussar. Ranking a random feature for variable and feature selection. *The Journal of Machine Learning Research*, 3:1399–1414, 2003.

[122] J. Sun, Q. Zhang, and E. P. K. Tsang. De/eda: A new evolutionary algorithm for global optimization. *Inf. Sci.*, 169:249–262, 2005.

[123] G. Syswerda and J. Palmucci. The application of genetic algorithms to resource scheduling. *Belew and Booker*, pages 502–508, 1991.

[124] Gilbert Syswerda. Uniform crossover in genetic algorithms. 1989.

[125] Talbi and El-Ghazali. *Metaheuristics: from design to implementation*, volume 74. John Wiley & Sons, 2009.

[126] E. G. Talbi. A taxonomy of hybrid metaheuristics. *Journal of heuristics*, 8(5):541–564, 2002.

[127] Kay Chen Tan, Q Yu, CM Heng, and Tong Heng Lee. Evolutionary computing for knowledge discovery in medical diagnosis. *Artificial Intelligence in Medicine*, 27(2): 129–154, 2003.

[128] D. Thierens and D. Goldberg. Mixing in genetic algorithms. *Urbana*, 51:61801, 1993.

[129] A. Toffolo and E. Benini. Genetic diversity as an objective in multi-objective evolutionary algorithms. *Evolutionary Computation*, 11(2):151–167, 2003.

[130] Ursem and K. Rasmus. Diversity-guided evolutionary algorithms. In *Parallel Problem Solving from Nature–PPSN VII*, pages 462–471. Springer, 2002.

[131] JA Vasconcelos, JA Ramirez, RHC Takahashi, and RR Saldanha. Improvements in genetic algorithms. *Magnetics, IEEE Transactions on*, 37(5):3414–3417, 2001.

[132] Frank Vavak and Terence C Fogarty. Comparison of steady state and generational genetic algorithms for use in nonstationary environments. In *Evolutionary Computation, 1996., Proceedings of IEEE International Conference on*, pages 192–195. IEEE, 1996.

[133] D. Ververidis and C. Kotropoulos. Fast and accurate sequential floating forward feature selection with the bayes classifier applied to speech emotion recognition. *Signal Processing*, 88(12):2956–2970, 2008.

[134] Darrell Whitley. A genetic algorithm tutorial. *Statistics and computing*, 4(2):65–85, 1994.

[135] A. W. Whitney. A direct method of nonparametric measurement selection. *Computers, IEEE Transactions on*, 100(9):1100–1103, 1971.

[136] I. Witten. H., and eibe. *F.,"Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations", Morgan Kauffman*, 1999.

[137] Man Leung Wong and Kwong Sak Leung. *Data mining using grammar based genetic programming and applications*. Kluwer Academic, 2000.

[138] R. Yang and I. Douglas. Simple genetic algorithm with local tuning: Efficient global optimizing technique. *J. Optim. Theory Appl.*, 98:449–465, 1998.

[139] L. Yu and H. Liu. Feature selection for high-dimensional data: A fast correlation-based filter solution. In *MACHINE LEARNING-INTERNATIONAL WORKSHOP THEN CONFERENCE-*, volume 20, page 856, 2003.

[140] Bo Yuan and Marcus Gallagher. On the importance of diversity maintenance in estimation of distribution algorithms. In *Proceedings of the 2005 conference on Genetic and evolutionary computation*, pages 719–726. ACM, 2005.

[141] R. Zebulum, M. Vellasco, and M. Pacheco. Variable length representation in evolutionary electronics. *Evolutionary Computation*, 8(1):93–120, 2000.

[142] Qingfu Zhang and Heinz Muhlenbein. On the convergence of a class of estimation of distribution algorithms. *Evolutionary Computation, IEEE Transactions on*, 8(2):127–136, 2004.

[143] Sun J. Tsang E. Zhang, Q. Combinations of estimation of distribution algorithms and other techniques. *International Journal of Automation and Computing*, pages 273–280, July 2007.

[144] E. Zitzler. Multiobjective evolutionary algorithms: a comparative case study and the strength pareto approach. In *Evolutionary Computation, IEEE Transactions on*, 1999.