An Investigation Of Meta-Evolution Using Multi-Expression Genetic Programming

A dissertation submitted in partial fulfilment of the requirements for the Open University's Master of Science Degree in Software Development

Mark Neville Richard Smith
(U3580272)

28 September 2009

Word Count: 14261

Preface

The project has been a journey of discovery both in terms of the accumulation of knowledge and the organisation of the thoughts and ideas of others and my own.

I would like to thank my guide Mr Jonathan Fanning for his insight and encouragement along the way and Information By Design Limited for their generous sponsorship of the voyage.

Table of Contents

Abstract

Evolutionary computing has proved to be one area within the field of artificial intelligence that has begun to fulfil some of the high expectations set in the second half of the 20th century.

Tools and applications in the field are beginning to make an appearance in many fields outside very high technology areas. However, research work into making the techniques more general and easier to apply has not always kept pace with the increase of computing resources.

One problem with using EC to solve a difficult task is that a large amount of human intervention is required to tune the system to provide good results. Recent work has attempted to allow the computer to tune the implementation itself. Other work has focused on processing as many possible solutions to the problem simultaneously. Both these directions are an attempt to improve the performance of the applications to tackle a wider range of more difficult problems.

This work uses a combination of these existing methods in a new way to harness the power of modern computers. A method of meta-evolution is proposed where search operators are implemented as chromosomes within each individual and then act on the task chromosome and on themselves.

The technique is compared with its predecessors using well known test problems. Although showing some promise the issues of evolving solutions to the test problems and the evolution itself are highlighted and the strengths of the existing techniques are shown.

List of Tables

List of Figures

Glossary Of Terms and Abbreviations

ADF	Automatically Defined Functions
AES	Average number of Evaluations to a Solution
EC	Evolutionary Computing
ET	Expression Tree
GA	Genetic Algorithm
GEP	Gene Expression Programming
GP	Genetic Programming
MBF	Mean Best Fitness
MGP	Meta Genetic Programming
MEP	Multi Expression Programming
MSP	Multi-Solution Programming
SSP	Single-Solution Programming
SR	Success Rate

1.0 Introduction

1.1 Background

Evolutionary computing is a branch of artificial intelligence that attempts to automatically solve problems using techniques inspired by the natural processes of evolution (Darwin, 1859). It has successfully been used on a wide range of real world problems and sometimes produces results humans could not.

Examples can be found in engineering (Keane & Brown, 1996), data mining (Au et al, 2003), scheduling (Jensen, 2003) and communications (Spector & Bernstein, 2003).

Evolutionary computing is concerned with searching candidate solutions, denoted as a 'search space', for a suitable or optimal solution for a problem. A successful implementation must make progress towards and eventually reach a suitable solution. This is 'evolvability' (Wagner & Altenberg, 1996).

Searching for solutions is a computationally intensive activity and research has focused on techniques to improve efficiency. This project continues this theme by combining two promising techniques; meta-evolution and multi-expression programming.

1.2 Research Question

The research question therefore is:

> Can a meta-evolutionary implementation of multi-expression genetic programming achieve fitter solutions, in the same number of generations,

than standard multi-expression programming when tested using the Ackley, Griewangk, Rastrigin, Rosenbrock, and Schwefel functions?

1.3 Overview of work

This report is structured as five main sections. Section two reviews the main themes of the research in the context of work done previously. The subjects of meta-genetic programming and representation are examined in detail.

Section three shows how work previously done leads to the formulation of the research question and sets out why it would be interesting to know the answer.

Section four sets out the methods employed to achieve a valid answer to the research question.

Section five shows the results of the experimental methods in section four and assess how they bear on the answer to the research question.

Section six draws conclusions to definitively answer the research question in the context of the work carried out and suggests improvements and further work.

The appendices show the implementation details and test data considerations.

2.0 Literature Survey

2.1 Introduction

A typical scenario is that an initial population is generated, usually randomly. Then a selection of the most suitable individuals, the fittest, are combined with each other, using sexual recombination known as crossover. They exchange parts of their genetic material to produce offspring. Subsequently the offspring are subject to mutation, having some part of their makeup changed. This cycle of selection, crossover and mutation, shown in figure 1, is repeated until a desired solution is obtained.

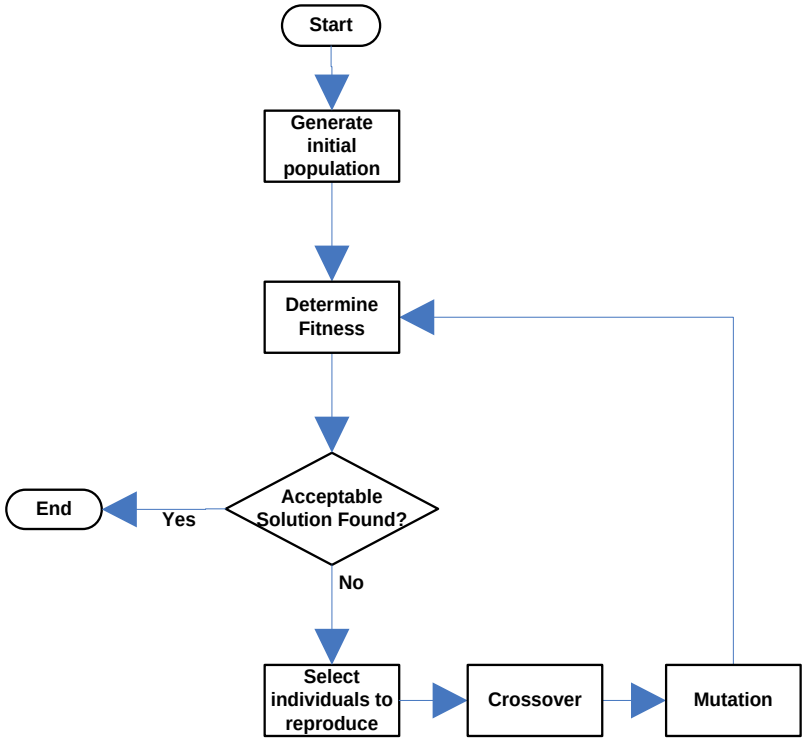

Figure 1 - A Generic Evolutionary Computing Cycle

2.2 Evolutionary Computing Strands

Evolutionary Computing began as several strands of work. An important strand, Genetic Algorithms, began with work of John Holland and his students. In (Holland,

1975) adaptation in computer systems was developed initially as a method of understanding adaptation in natural systems.

This work introduced the concept of an algorithm that evolves a population of individual candidate solutions. The population of individuals as strings of bits were subjected to artificial processes that mimicked biological equivalents of crossover and mutation until they produce a desired solution simulating the process of natural selection.

The emphasis of a formal theory of 'schemas' to explain the behaviour of genetic algorithms gave the work added importance and is discussed in the theory section below.

Holland's student De Jong in his dissertation (De Jong, 1975) placed GA on solid experimental foundations with his work using a minimalist simple genetic algorithm on five function optimisation problems.

By using careful experiments De Jong showed that the effects of varying four fundamental parameters of population size, crossover probability, mutation probability and generation overlap were in accordance with Holland's schema theory.

2.2.1 Genetic Programming

Another strand of Evolutionary Computing, Genetic Programming (GP), used similar evolutionary techniques but different problem representations. Early examples of work are (Fogel at al, 1966) which used mutation in a population of finite state machines and (Cramer, 1985) who first proposed evolving programs represented as tree structures.

Most notably Koza (1992b) began with a concept of using evolutionary techniques, as an alternative to designing computer programs working using tree based representations, s-expressions, of the Lisp programming language.
The early work of genetic programming suffered as a poor relation to genetic algorithms from a lack of theoretical foundations because Holland's existing schema theory could be used to explain the workings of GP.

Koza uses some simple well known problems to illustrate the operation of Genetic Programming. One such problem is to centre a cart moving on a frictionless track in the minimum amount of time. A designed solution to the amount of force required to centre the cart can be represented by the infix lisp s-expression of:

```
(GT ( * -1 X ) (* V (ABS V) ) )                    (1)
```

GT is a lisp function that returns +1 if its first argument is greater and -1 if the second argument is greater and ABS is the absolute value. X is the distance from the centre and V is the velocity at the time.

Figure 2 below shows the correct solution to Koza’s cart centring problem illustrated as a GP lisp s-expression tree.

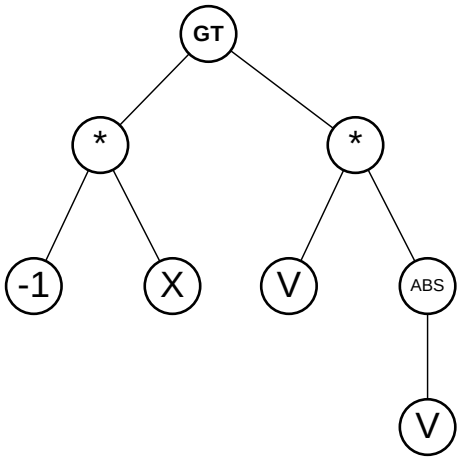

Figure 2 - Example representation – Koza's Cart Centring Problem

Koza uses the cart centring problem to show how the degrees of fitness of evolving solutions can be calculated and the steps necessary to implement a GP system that finds a solution to the problem.

Evolving programming language problem solutions, rather than bit strings, means there is a much larger search space, making it much more computationally intensive and so initially GP was limited to simple problems.

Subsequent developments in genetic programming have focused on two main concerns. Firstly to adjust the program representations and their configuration to increase the power of the technique, and secondly, to apply the technique to problems in the real world.

Koza (1994) using a lawn mowing problem and Angeline & Pollack (1994) using the towers of Hanoi and tic-tac-toe problems, adjusted representation to utilise the power of program modularity using the evolution of Automatically Defined Functions to improve performance.

Variations of representations in GP fall into three categories (Kantschik, 1999); tree based (Koza, 1992b), linear based (Nordin, 1994), (Purkis, 1994) and graph based (Teller & Veloso, 1996). GA can be considered as a specialisation of linear GP.

GP has been successfully applied to a very wide range of real world problems including control systems, financial modelling, image processing and electronic circuit design.

Examples of successful GP applications on difficult real world problems are (Teller & Veloso, 1996) using GP to recognise everyday images and (Koza et al, 1996) successfully evolving an analog band pass filter electronic circuit using GP.

2.2.2 Applying Evolutionary Computing to Different Problems

Much of the time consuming development work on techniques of EC consists of adjusting representation, selection, crossover and mutation strategies, which have had different degrees of success, depending on the nature of the problem being tackled.

To alleviate this arduous tuning, researchers have attempted to simultaneously automatically adapt or evolve representations and operators alongside the evolution to solve the specific problem. This is meta genetic programming with GP or evolving evolutionary algorithms for GA.

Greffenstette (1985) first proposed the use of a meta-level GA to adapt control parameters for a sub-level GA and meta-adaptation was suggested for GP by Schmidhuber (1987).

Subsequently other researchers have developed successful meta implementations which evolved different areas of GA and GP algorithms. Meta adaptation can be used in adapting crossover, mutation and selection, as well as other parameters such as population size and replacement strategy.

In (Spears, 1995) and (Dioşan & Oltean, 2006) it was shown that an individual based evolved crossover mechanism outperformed a traditional fixed crossover. In (Thierens, 2002) and (Blum, 2005) superior mutation is adapted and in (Eiben, 2006) selection is adapted. In (Edmonds, 2001) meta level evolution with a general operator of variation is demonstrated and similarly in (Spector, 2001) GP entities are responsible for producing their own children.

Recent work in the field has shown that meta adaptation is still an active area of research and that there is still scope for constructive variations of the technique.

2.3 Theory in Evolutionary Computing

The subject of EC is hugely complex and researchers have only modest ambitions of forming predictive models to explain the operation of EC on a given problem (Eiben & Smith, 2003). Any insight provided, even for simple problems, would therefore be gratefully received.

Theoretical ideas in EC have followed several paths. One path has been the attempts to address the specific processes occurring during evolution. Initially in (Holland, 1975) the Schema Theory was proposed, which was considered fundamental to understanding GAs until the early 1990s.

Schemas are a formalisation of the concept of building blocks composed of bit strings and wildcards which represent averages processed by a GA. The theory is expressed in terms of the survival and growth of schemas from one generation to another under the effects of selection, crossover and mutation. However Schema Theory has difficulty explaining situations where a GA is limited or enters a state of equilibrium.

Another branch of EC theory is concerned with convergence of general searches using populations of solutions. An EC system can be described as a Markov chain (Eiben, Aarts et al, 1991) providing that at any given time it can be at a finite number of states and that the probability that it will be in a particular state is solely dependent on its previous state. Markov chain models have proved accurate however they are so complicated it is difficult to draw any useful conclusions from them (Langdon & Poli, 2002).

2.3.1 No Free Lunch

A major theoretical work in the field of search algorithms, of which Evolutionary Computing is part, the first of the No Free Lunch theorems, was published in 1995. (Wolpert & Macready, 1995) mathematically proves that no search algorithm is, on average, better than any other, over the average of all possible problems.

This means comparing EC implementations A and B, if A works better than B on one problem then B will work better than A on another. Consequently much of the development work on techniques of Evolutionary Computing consisted of adjusting representation, selection, crossover and mutation techniques, which had different degrees of success depending on the nature of the problem being tackled.

This explains why so much effort has been expended by researchers in tuning standard GA and GP implementations to different specific problems. The practical implication of No Free Lunch are that it is not possible for a single general purpose GA or GP to perform well on all problems. However it is still possible to construct methods that perform well on a range of problems.

2.4 Meta-Genetic Programming

Meta-genetic programming is a technique that builds on and borrows from several areas of EC. It is an effort to improve the EC by making it easier to apply and to improve its results. This is done by using the principles of EC to adapt the EC process itself.

The sections below begins with a discussion of the work that lead to the concept of MGP. The use of MGP itself then examined. Techniques that have been successfully applied to mainstream EC and MGP are subsequently discussed.

2.4.1 Adaptation in Evolutionary Computing

Since the early days of EC, researchers have spent time adapting their evolutionary models to achieve success with the chosen application. It is natural that, rather than do this work by experimentation, to contrive schemes for a computer to do it.

A framework for categorising types of adaptation for EC was proposed in (Hinderding et al, 1997) and is partially reproduced in table 1 below.

	Adaptation Mechanism		
Level	Deterministic	Adaptive	Self-Adaptive
Environment	E-D	E-A	E-SA
Population	P-D	P-A	P-SA

Individual	I-D	I-A	I-SA
Component	C-D	C-A	C-SA

Table 1 - Types of EC Adaptation

Three adaptation mechanisms are characterised. Deterministic, where a value, e.g. mutation rate, is varied according to a formula (Fogarty, 1989), Adaptive, where a feedback mechanism is used, say to adjust operator ratios (Tuson & Ross, 1996) and Self-Adaptive where the parameters to be adapted are encoded in the chromosomes, for example, mutation rate (Bäck, 1992).

The chart also defines the level at which the adaptation takes place and some examples are given. An example of Environmental level adaptation, where a fitness calculation changes in response to niching (Darwen & Yao, 1996). Population adaptation affects all members of the population or sub-population, for example parallel populations affecting mutation rate (Lis, 1996). Individual level adaptation adjusts parameters that only affect that individual, for example crossover point (White & Oppacher, 1994). Component level adaptation, for example, adjusting component mutation probabilities is found in (Fogel et al, 1995).

Some notable examples not included in the survey above as important work in the Self-Adaptive Individual category are the adaptive crossovers in (Angeline, 1995) and adapting mutation in (Spears, 1995). Recently published work (Thierens, 2002) on mutation rate control in the Adaptive and Self-Adaptive categories shows this is still an active area of research.

The methodology of the studies of have a common factor of using an EC algorithm on a test problem or problems. Variations with adaptation (sometime by degrees) and without are tested and invariably the adaptive version proves superior.

No adaptive scheme seems to have yet gained wide acceptance because the conclusions of this research provide no insight that a particular scheme can be successfully used on any future problem.

In this respect adaptive schemes so far suffer from the same criticism, of requiring experimental tuning, that they sought to address.

2.4.2 The Meta-Evolutionary Model

To implement a model of adaptation (Greffenstette, 1985) proposed a new scheme of meta-adaptation which uses a two level GA system illustrated in Figure 3 below. The traditional GA model occupies the area inside the dashed box.

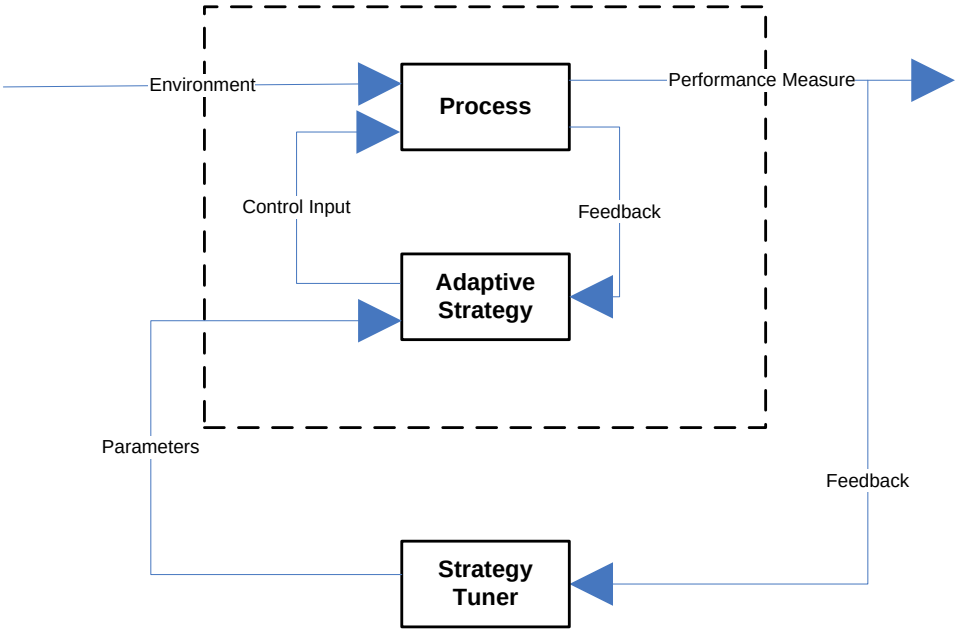

Figure 3 - Greffenstette's Meta-Evolutionary Model

This model was used to adapt a fitness scaling model and a crossover selection strategy. The fitness scaling adaptation varied the population fitness calculation to attempt to maintain selection pressure and the crossover selection applied elitism to preserve fit individuals over generations.

Standard parameter settings were used for the meta level GA which ran on 50 random GAs from a population of 1000. The results were compared with two other on a problem of real image transformation considered difficult by measuring pixel differences over numbers of trials. The meta method showed small improvements over traditional methods.

2.4.3 Meta-Implementation Successes

The meta approach was used for GA by (Fogel & Fogel, 1991) for two function optimisation problems and was able to find solutions that the standard GA could not and was also reported to require only a small additional computational cost.

The meta approach was applied to tree based GP in (Edmonds, 1998) and in a major innovation for the first time was used to evolve the action of operators rather than only parameters within the algorithm. A separate tree was used to evolve a variation operator that used a terminal set designed to operate on the selected parent trees of the base GP algorithm as well as on themselves.

The method was tested on the even parity 3 and odd parity 4 problems and initially was hampered by insufficient variation in the base population yielding smaller fitness gains than a proportionate selection technique. After some investigation it was concluded that preserving variety leading with associated jumps in fitness was less important than short term average increases in fitness. Omitting some of the terminals and implementing a fitness scheme enabled variety to be maintained and to maximise the fitness increase. After this tuning the meta approach was about twice as efficient in computational cost as the standard GP, however a note of caution is added; that the meta technique is less robust than standard GP.

Other researchers have successfully applied meta methodology to GA (Wang, Goodman et al, 1996) and Graph GP (Kantschik, Dittrich at al, 1999b). In (Samsonovich & De Jong, 2005) it is suggested that meta techniques are useful setting boundaries on the effects of the No Free Lunch theorems.

In (Spector, 2001) the paradigm of evolving meta level operators with custom functions from Edmonds was extended. Auto constructive evolution evolves programs of a stack based language Push, where individuals are responsible for producing their own children. It is reported that individuals are quickly able to reproduce and are sometimes able to solve problems but mostly a diverse population reaches equilibrium but does not solve the problem. Fitness case rotation, changing fitness cases every generation, is suggested as improving matters.

Linear Genetic Programming was used by (Dioşan & Oltean, 2006) in a meta framework where individuals were evolved using a meta level consisting of a combination of the operators initialise, select, crossover and mutate which are defined as conventional values. The system is tested on ten function optimisation problems and was found to perform better than the standard (i.e. fixed initialisation, selection, crossover and mutation) genetic algorithm for the test functions.

The MEP approach was used by (Oltean & Groşan, 2003) in a meta framework where individuals were again evolved using a meta level consisting of a combination of the operators initialise, select, crossover and mutate which are defined as conventional values. The system is tested on Griewangk's function and was reported to achieve good performance although requiring substantial amounts of memory.

A meta approach using GP with a GA phenotype to genotype mapping function is used by (Tavares, 2004) which is tested on function optimisation against a hand constructed solution. Fitness case rotation is used and good results are obtained. A suggestion of evolving a set of selection, search, replacement strategy and fitness operations in a meta method is made and some problems with this approach are discussed.

In (Dioşan & Oltean 2007) Tavares' suggestion is taken up. A meta GA approach is used to evolve binary and real value GAs using three search operators; selection, mutation and crossover. The technique is tested on ten problems including the Ackley, Griewangk, Rastrigin, Rosenbrock, and Schwefel functions and shows superior performance.

2.4.4 Meta Individual Representation

Using additional genetic material to represent the meta aspects of the algorithm gives rise to the question of how it is related to the individual. Different researchers have adopted various techniques.

In (Hinterding, 1997) two separate chromosomes are used for a meta implementation to solve a Cutting Stock problem to adapt a mutation operator type and strength of a group mutation. One chromosome is used for the problem and one chromosome is used to encode meta parameters.

Similarly in (Edmonds, 1998) used a population of separate trees for a variation search operator which co-evolved with the main problem trees.

In (Oltean & Groşan, 2003) from above, the order and frequency of the search operators is the subject of meta-evolution and the results of application to many lower level individuals form the basis of the fitness at the meta-level.

In the previously discussed (Spector, 2001) individuals produce their own children leaving individuals themselves to evolve a method to partition problem and reproductive functionality. This is held as a desirable characteristic because it is closer to the behaviour of successful living systems.

In Spector special reference is made to the desirability that meta EC should take inspiration from biological systems. In a natural setting the evolution of the ability to solve a problem (staying alive to reproduce) is carried out alongside the ability to actually reproduce and adapt to changing environments.

2.5 Tree Representations

Early implementations of EC contrasted with the use of fixed length linear representations, bit strings, in GA with the variable length Lisp expressions of GP.

Figure 4 illustrates the representation of a simple expression as a tree alongside an example bit string.

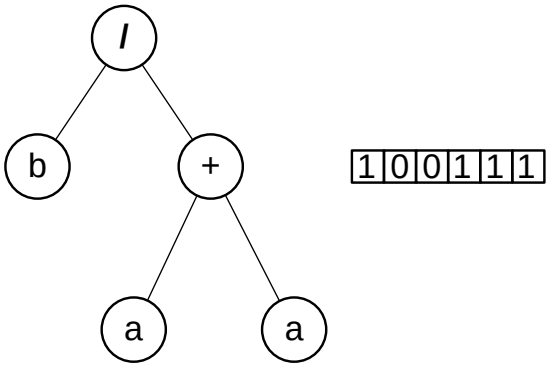

Figure 4 - Tree Expression for E = b / (a + a) and 6-bit string example

GA bit string representations have the advantage of allowing easy genetic search functions but the disadvantage of difficulties in supporting complex operations in terms of decoding for fitness calculations, as well as producing results accessible to humans.

Conversely many GP implementations suffer the inverse of these problems. They can be difficult to implement suitable genetic operators for, but are simpler to implement in fitness calculations and produce results readily accessible by humans.

2.5.1 Better Trees

Recognising this issue, arising from the lack of distinction between phenotype and genotype, Ferreira (2001) formulated Gene Expression Programming seeking to bridge this gap.

In GEP the advantages of a fixed length linear representation allowing easy search functions are combined with the simple decoding and readability aspects of expression trees. However, the GEP representation includes additional useful characteristics that, although the tree length is constant, many expressions are supported within a single expression tree, from the top until the deepest root. This

has the effect of allowing any modification of the tree by genetic operators to result in a valid result; another expression tree.

This innovation was shown as a dramatic improvement of 'four orders of magnitude' less computationally intensive, on the most complex problem tested, compared to GP.

2.5.2 Multi-Expression Programming

Building on the work of Ferreira, Oltean & Dumitrescu (2002) proposed Multi-Expression Programming which has similar properties to GEP of combining the advantages of linear and tree representations. In Oltean & Dumitrescu (2002) MEP and GEP representations are compared using an example expression:

```
E = a4 + a3 + a2 + a                                    (2)
```

Using a function set containing * and +, the arithmetic operations 'multiply' and 'add', with a single terminal a the linear GEP representation for E is:

```
+a+*+aa**a***aaaaaaaaaaaaaa                             (3)
```

This GEP chromosome has 'head' length of 13 and a 'tail' of 14 giving a total length of 27. Representing the same chromosome in MEP, using a notation where terminal have a position and designation and functions have their operation and two references to the positions of their operands, is:

```
1:    a
2:    *     1,    1
3:    *     1,    2
4:    *     2,    2
5:    +     1,    1
6:    +     3,    5
7:    +     4,    6
```

This gives a total length of 19. (The numbers 1 to 7 are for reference only and not part of the chromosome).

Two reasons for the shorter MEP representation compared to GEP are given, MEP reuses code and does not contain codes not used for expression. GEP does not reuse code and adds a non-coding 'tail'.

After some experimentation Oltean & Dumitrescu selected the binary tournament selection method. They also demonstrate the utility of MEP with one point, two point and uniform crossover strategies.

Under tests on a symbolic regression problems MEP proves superior to GEP in terms of higher success rate, lower population size and lower number of generations required to achieve a given success level. The superiority of MEP to GEP has been disputed by Ferreira (2002) but without making any direct comparison.

Also in Oltean & Dumitrescu (2002) the authors also show more successful MEP implementations of Tic Tac Toe and the Travelling Salesman Problem compared to other good techniques for these problems.

Subsequently in (Oltean & Groşan, 2004) MEP is compared with other variants of GP which are Grammatical Evolution (Ryan, Collins at al, 1998), GEP, Linear GP (Brameier & Banzhaf, 2001 and others), Cartesian GP (Millar & Thomson, 2002), GA for deriving software (GADS) (Patterson, 2002) and infix form GP (Oltean & Groşan, 2003) using five symbolic regression test problems considered difficult. In all five problems MEP proves to be superior in terms of higher success rate, lower population size and lower number of generations required to achieve a given success level.

Many techniques in linear GP can also be represented as their Multi-Solution variants and (Oltean, 2005a) demonstrates the superiority of the Multi-Solution variants over the Single-Solution for MEP, Linear GP and infix form GP (as above) for four symbolic regression test problems considered difficult (different but similar to the tests above).

In terms of success rate verses number of genes (where SR is defined successful runs divided by total runs), the Multi-Solution variants are about (results are presented as graphs) 30% to 50% better than the Single-Solution variants for the four problems. For SR verses population size the results are about 15% to 30% better for all the problems. MEP is often but not always superior to the other Multi-Solution variants.

The success of the multi-solution variants, particularly noticeable for larger gene numbers, is ascribed to them allowing variable length chromosomes and the ability of multi-solution variants to allow more function variations, but with the same decoding complexity, as single solution variants. Further work is suggested to investigate Multi-Solution variants within other evolutionary models.

MEP is used to evolve evolutionary algorithms in (Oltean & Groşan, 2003) which is discussed above. It has also been successfully used on many problems including parity problems, the knapsack problem, digital circuit design and fault finding, stock market prediction, intrusion detection and function optimisation.

Research using MEP and its variants is still ongoing, in (Hadjam, Moraga & Benmohamed, 2007) a distributed version of MEP is used to evolve digital circuits.

It can be seen that the technique of Multi-Solution programming in the form of MEP and other Multi-Solution forms has been extensively tested and found superior to many other evolutionary models. One reason for MEP superiority is given in terms of the No Free Lunch Theorem: an multi-solution representation holds many possible genotypes (Oltean, 2005a). There is no reason why a fitter genotype at any time should continue to be fitter as solutions evolve. Although MEP has been extensively tested, so far it does not seem to have been so widely adopted by researchers as GEP.

2.6 Summary

Meta-evolution is a natural extension of EC methods and researchers have been keen to realise it's potential. As expected, the most efficient meta techniques make use of the most efficient standard EC techniques.

The relationship between the meta elements and the problem solving elements is polarised between two successful strategies; co-evolution and individual representation. It is most convenient for the meta element representations to use the same representations as the problem representation.

However results of implementations have been mixed. Problems have occurred with individuals becoming over specialised and a lack of diversity in the population leading to performance below the level of the non-meta versions of the implementations.

Relatively few papers have been published on the subject compared to other specialist areas within EC. This might suggest that either the problem of achieving a

robust general meta-implementation is proving difficult or that other areas of research are more attractive (or both). However there are still substantial areas with meta EC that require further investigation.

3.0 Justification of Research Question

The research question for this project is concerned with the merits of a meta implementation of an existing technique in genetic programming.

The difference between using the two methods is the replacement of the fixed genetic operators of the original implementation with genetic operators that evolve to adapt to their prevailing circumstances.

3.1 Hypothesis One

The research question is based on three hypotheses, firstly:

> *Genetic operators that adapt during the search are more efficient than using fixed genetic operators.*

The literature gives the reasons why this hypothesis might hold true which are:

- Meta adaptation has been frequently found to increase the efficiency of EC implementations; (see section 2.2.2)
- Meta adaptation of any one of the genetic operators has frequently been found to increase efficiency; (see section 2.2.2)
- A meta-adaptation implementation acts to mitigate the effects of the No Free Lunch theorem; (see Samsonovich & De Jong (2005), section 2.4.3)
- A meta adaptation implementation is closer to the evolution of successful 'natural' living systems. (see Spector (2001), section 2.4.4)

3.2 Hypothesis Two

The second hypothesis on which the research question is based is:

Multi-Expression programming is an efficient GP representation for meta adapting implementation.

The literature gives substance to the reasons why this hypothesis might hold true which are:

- Multi-Expression programming has shown it is very efficient in evolving solutions for solving problems so it is reasonable to suppose it will also very efficiently evolve genetic operators; (see Oltean & Groşan (2004) in section 2.5.2.)
- Meta adaptation of the order of fixed genetic operators in a Multi-Expression algorithm has been found to increase the efficiency; (see Oltean & Groşan (2003) in section 2.4.4.)
- A multi-expression implementation acts to mitigate the effects of the No Free Lunch theorem. (MEP being a type of MSP). (see Oltean (2005a) in section 2.5.2.)

3.3 Hypothesis Three

The third hypothesis on which the research question is based is:

A Multi-Chromosome meta representation is an efficient representation for a meta-MEP implementation.

Similarly, the literature provides reasons which this might be true:

- A multi-chromosome approach works well in meta adaptation; (see section 2.2.2)
- A multi-chromosome approach can be implemented in a tree representation; (see Edmunds (1998) in section 2.2.2.)
- A multi-chromosome approach supports a 'natural' division of problem and meta elements. (see section 2.2.2)

The aim of the research is to test configurations of a meta-genetic program in which most aspects evolve. The intention is to test how well a fuller model of adaptation can work.

This project partially takes up the proposal in (Tavares, 2004) where he demonstrates the evolution of a mapping function.

> *"We propose evolving the following aspects of the evolutionary algorithm: mapping function, genetic operators, selection procedure, replacement strategy, and evaluation function." (Tavares, 2004:3)*

The evolution of a replacement strategy and evaluation function is not considered in this work.

The question of the efficiency of a meta-genetic implementation has only been addressed in the EC literature for less broadly based adaptations of an EC system. Usually only a single element of the algorithm is adapted, for example, only crossover, only mutation or only selection.

Work on adaptations of EC systems has progressed, sometimes simultaneously, along two scales, firstly moving from adaptive to self adaptive mechanisms and secondly from adaptation in environment, population then to individual and on to components within individuals.

It is intended that this project should continue progress towards an algorithm for which most aspects evolve. Previous work has been more limited in scope for two main reasons. Firstly the meta adaptation of the particular aspect of the algorithm

has not been previously studied and secondly the computational resources have been comparatively limited.

Adaptation of individual elements of the algorithm has been almost universally shown to be either advantageous, or at least not detrimental, to successful problem solving.

The difficulties of using the theory of EC to make predictions about how best to implement have been mentioned in the section above. In this light it would seem reasonable to adopt an empirical approach and to apply what has been previously successful.

Literature documenting the concurrent evolution of all the search operators; selection, crossover and mutation has only been found in one very recent instance, (Dioşan & Oltean 2007)

3.4 Research Answer Practicalities

Part of the justification of the research question relies on its ability to be clearly answered. The literature provides a well trodden path for measuring the computational efficiency of EC implementations which is discussed in the research methods section.

The software developed will be tested on 'standard' EC benchmarks which occur frequently in the literature. Five traditionally hard function optimisation problems will be used for testing. These are the test functions Ackley, Griewangk, Rastrigin, Rosenbrock and Schwefel. (See appendices for more details.)

These test problems which have been subjects of extensive previous study. They are used in (Dioşan & Oltean, 2006) in section 2.4.3 and Dioşan & Oltean (2006) in section 2.2.2

To ascertain if the new meta variation is more computationally efficient it will be compared with the standard version of multi-expression programming. The research question will be considered answered affirmatively under the following circumstances:

> The fitness of the best solution over 100 runs for the meta-version is superior compared to the standard version of the MEP algorithm after the same number of generations by a statistically significant margin with at least 95% confidence when using the same test data.

The calculation of the statistical significance is dealt with in the experimental results methods section below.

3.5 Contribution to Knowledge

The contribution to knowledge of the project is to further the knowledge of adapting EC algorithms using a meta-programming approach.

A successful implementation would further knowledge in EC in several ways:

- It would allow an improvement in solving problems performance which would enable more complex problems to be tackled.
- It would enable EC to be used on a wider range of problems that have not been successfully tackled using genetic operators designed by humans.

- it would mean the work of constructing the most appropriate genetic operators for particular problems would no longer be required.
- It will shed light on the nature of the problems tackled by showing what techniques evolved at the meta-implementation level work best.

The implementation will be tested using recognised methods from the literature which will validate the results obtained. These tests are detailed in the research methods section below.

The work of this project builds on the works of Oltean by taking up a suggestion for meta-implementation made by (Tavares, 2004) which is also taken up by (Dioşan & Oltean, 2007). Tavares shows success evolving a mapping function verses a hand coded version and Dioşan & Oltean show success simultaneously evolving selection, crossover and mutation in a GA context which does not use MEP.

Oltean (Oltean, 2005a) demonstrates using MSP representations in Multi-Expression Programs (MEP), using Linear Genetic Programs (MS-LGP) and Infix Form which all show performance improvements using the technique compared to the Single Solution version of the same representation.

Most systems that evolve programs or algorithms rely on hand constructed genetic operators for selection, crossover and mutation. Dioşan (Dioşan & Oltean, 2006) reports success using MEP to evolve crossover where the evolved crossover is superior to a hand constructed crossover.

It is therefore thought of interest to investigate if more of the meta environment of the system can evolve. Combining this with MEP also evolves the (genotype to phenotype) mapping and boosts the performance of all parts of the system.

The work of the project should be of interest from several viewpoints. Evolving programs or algorithms to solve problems is a computationally intensive task. In Genetic Programming, evolving solutions can take days and may process thousands of individuals (Koza, 1990:115), although continuous improvements in computer hardware are mitigating this situation. Therefore performance improvement techniques are of interest, particularly as they enable more complex problems to be tackled.

4.0 Research Methods

4.1 Literature Review Methods

The work of this research project included a continuous review of the literature related to the aims of the research. The purpose of the literature review was to learn about the themes of previous research work and ensure the research builds on and is in the context of previous work.

The literature review has concentrated on the themes of the research. These are:

- Adaptation;
- Meta-evolutionary representations and techniques;
- Multi-Solution Programming.

The literature review strategy element to find relevant papers consisted of:

- Identify significant text books;
- Use sources sections in text book to identify sources for papers;
- Work backwards (in time) through indexes of journal and conference sources searching for relevant papers;
- Use references in text books concerned with themes of the research to identify possibly relevant papers;
- Use references in relevant papers to identify further relevant papers;
- Search for other papers by authors of relevant papers via the web and within electronic sources (see resources section for electronic sources details);
- Search using keywords within electronic sources;
- Search using keywords using web search engines.

It is believed that most relevant papers have been consulted. These papers have been indexed and filed. New relevant papers have appeared from time to time and these were included in the literature review on a couple of occasions. During the research some changes of emphasis were reflected by less relevant papers being excluded and more relevant papers being included.

4.2 Experimental Software Methods

Determining the answer to the research question uses the method of collecting experimental data from software built according to the methods that form the subject of the research. The software is built using standard techniques and follows patterns typical of EC implementations, but particularly relies on the example implementation of MEP provided by Oltean (2004).

4.2.1 Oltean's MEP Implementation

Oltean's (2004) example implementation of MEP is provided as source code for a simple 'C' program. The program implements a typical EC loop of fitness determination followed by selection, crossover and mutation.
The core of MEP is implemented as an array of records or nodes each denoting a function (or operator) or a terminal (or variable). Similarly training data is represented as an array of variables and values.

Each terminal node has two argument indexes which refer to lower nodes. The initialisation and mutation of new nodes are shown as the random selection in relative proportion of the ratios of functions to terminals. i.e.

```
option = rand() MOD (NumberOfTerminals + NumberOfOperators)

if option LESS than NumberOfTerminals
      New node is variable
else
      New node is function
```

Where rand() generates a random 32 bit integer and MOD is the remainder after division.

The first or head node can only be a variable as it can have no lower arguments. Functions are given negative values to distinguish them from variables.

The evaluation of the MEP expression is accomplished as a loop counting through the node array and interpreting each value. If negative, a function it is executed on the contents of the nodes indexed, otherwise the node if set to the contents of that variables training data value.
A running value is maintained during the loop and checked against the required training data value. If the deviation from the desired value decreases a fitness vale is saved with the index of the node it occurred. If divide by zero occurs the 'division' function is mutated into a terminal.

4.2.2 Experimental Software Characteristics

The required characteristics of the experimental software are:

- To run in standard MEP and meta-adaptation mode by changing a configuration setting of the software;
- To support facilitate changes and tuning of meta-adaptation model
- To run using training data for the specific problem, supplied in a file, similar to that used in literature
- to allow collection of performance data for two purposes; firstly to support the tuning and modification of the system to achieve satisfactory performance and secondly for use in presentation and allow comparison with the work of other researchers;

The implementations will use the standard MEP technique from (Oltean, 2004) to avoid the undefined operation of divide by zero which is to mutate the divide operation into a terminal.

The current version of the experimental software is available via the web (see appendix).

4.3 Valid EC Testing Methods

4.3.1 Best Practice EC Testing Methods

Eiben and Smith (2003) suggest that work in developing an EC algorithm should generally follow a pattern along the lines of:

1. Identify a new EA or variant for solving a specific problem.
2. Identify existing EA(s) to compare the new variant against.
3. Identify existing benchmark problems from the literature.
4. Obtain a problem generator and generate random problem instances.
5. Execute all the algorithms on each problem instance 100 times.
6. Record the Average number of Evaluations to a Solution (AES) and record it's standard deviation.
7. Record the percentage of runs terminating with success, the Sucess Rate (SR).
8. Record the fitness of the best individual at termination for each run an take the average over all runs. This is the Mean Best Fitness (MBF). Also record it's standard deviation.
9. Compare the data (AES, SR, MBF) for each variant using standard statistical significance tests.

10. Make the program code and test instances available on the web

This procedure should enable other researchers to assess the strength of the work.

4.3.2 Project EC Testing Methods

In the particular case of this project the points in section 4.3.1 are addressed as follows:

1. A new meta-programming algorithm has been formulated based on a hybrid of several successful previous techniques.
2. The new meta variant will be compared with the original MEP technique. The MEP implementation will be validated against results from the MEP results for the same test problems (Oltean & Groşan, 2003) and (Oltean & Groşan, 2004).
3. The benchmarks chosen for comparison are Ackley, Griewangk, Rastrigin, Rosenbrock and Schwefel (see section 3.4)
4. This step of using a problem generator to generate random problem instances will be omitted for this work.
5. The code harness developed runs an instance 100 times by default.
6. The code harness developed can write the AES to a log file although it is not recorded during this project. (The Standard Deviation of AES could also be calculated and recorded but isn't).
7. The code harness developed can write the SR to a log file but is not recorded forthis project.
8. The code harness developed writes the MBF to a log file and could write the Standard Deviation but doesn't.
9. After the runs the log files will be processed for tests of statistical significance

10. The code and test instances may be placed on the web.

4.4 Meta Implementation Methods

To implement the evolution of the genetic operators, each individual will have a separate selection, crossover and mutation chromosome, giving a total of four chromosomes per individual.

To complete the software development a slightly modified meta version of the five major steps from (Koza, 1992b) was adopted.

For each of the four chromosomes these are:

- Determine the terminal set;
- Determine the operator set;
- Determine the fitness measure.

For the overall algorithm the steps are:

- Determine an optimum population size and number of generations to be run;
- Determine the circumstances for designating a result and terminating a run.

For determining the chromosome characteristics for the 'task' (or problem) chromosome the values are taken from literature examples. For the selection, crossover and mutation values and the population size and number of generations the values used formed part of the investigation.

The method to supporting both standard MEP and the meta variant is achieved by encoding the standard MEP search operators (crossover, mutation and selection) as

invariant 'meta' operators (i.e. as though they were meta operators). This will remove any performance differences due to 'interpreting' during the operation of the operators. Interpreting here means performing the respective operator action on the problem/task chromosome in a similar manner to (Edmonds, 2001).

4.4.1 Project Settings for Test Functions

Each function implemented in the search function requires appropriate code to implement its action on the problem chromosome. For the Ackley problem these parameters are specified as:

Criteria	Setting
Crossover	Uniform with probability 0.9 (Syswerda, 1989)
Mutation	2 mutations per chromosome
Selection	Binary tournament (2 parents from 4 individuals)
Code length	3000
Population Size	100
Function Set	{+, −, *, /, exp, sqr, sqrt, cos}
Terminal Set	N=5

Table 2 - MEP Settings for Ackley Problem

For the Griewangk problem these parameters are specified as:

Criteria	Setting
Crossover	Uniform with probability 0.9 (Syswerda, 1989)
Mutation	2 mutations per chromosome
Selection	Binary tournament (2 parents from 4 individuals)
Code length	3000
Population Size	100
Function Set	{+, −, *, /, sqrt, cos}
Terminal Set	N=5

Table 3 - MEP Settings for Griewangk Problem

For the Rosenbrock problem these parameters are specified as:

Criteria	Setting
Crossover	Uniform with probability 0.9 (Syswerda, 1989)
Mutation	2 mutations per chromosome
Selection	Binary tournament (2 parents from 4 individuals)
Code length	3000
Population Size	100

Function Set	{+,−, *, sqr}
Terminal Set	N=5

Table 4 - MEP Settings for Rosenbrock Problem

For the Rastrigin problem these parameters are specified as:

Criteria	**Setting**
Crossover	Uniform with probability 0.9 (Syswerda, 1989)
Mutation	2 mutations per chromosome
Selection	Binary tournament (2 parents from 4 individuals)
Code length	3000
Population Size	100
Function Set	{+, −, *, sqr, cos}
Terminal Set	N=5

Table 5 - MEP Settings for Rastrigin Problem

For the Schwefel problem these parameters are specified as:

Criteria	**Setting**
Crossover	Uniform with probability 0.9 (Syswerda, 1989)
Mutation	2 mutations per chromosome
Selection	Binary tournament (2 parents from 4 individuals)
Code length	3000
Population Size	100
Function Set	{+, −, *, sqrt, sin}
Terminal Set	N=5

Table 6 - MEP Settings for Schwefel Problem

The problem fitness functions will taken from the two problem sources (Oltean & Groşan, 2003) and (Oltean & Groşan, 2004).

The training data was generated using the algorithms in the appendices and checked using MATLAB code from (Hedar, 2007). For each problem fifty rows of training data was arbitrarily selected from 100,000 randomly generated rows.

4.4.2 Meta Implementation Specifics

The implementation of standard MEP coded as the selection, crossover and mutation chromosomes proved the concept of the meta layer design. The operation of the select chromosome which encoded the binary tournament selection required access to the fitness for itself and the random tournament opponents. This was

accomplished by setting the select chromosome training data to be an array of all the populations fitnesses. The tournament winners array entry in the training data is set to true to indicate that individual is eligible to be a parent.

Similarly the crossover and mutation chromosomes require access to it's own and another parents MEP chromosome codes. This was achieved by each parent copying all its partners chromosome code into a combined crossover-mutation training data array which can be acted upon (i.e. updated or written to) by the crossover and mutation chromosomes. This process is could be considered as equivalent to the biological process of mitosis.

The crossover chromosome then copies parts of its individual code into the training data array as appropriate during uniform crossover (Syswerda, 1989). A copy node operator is implemented for the crossover.

The mutation chromosome also acts on this crossover-mutation training data array by 'mutating' the array values. Once the process is complete a 'child' is constructed from the crossover-mutation training data array for each 'mother' individual.

4.4.3 Search Operator Virtual Machine

The selection, crossover and mutation meta chromosomes require an extension to the linear evaluation loop of MEP. Also an additional type of MEP node, designated a 'Constant' node is implemented. (This nomenclature can be confusing as MEP Variables are fixed and MEP Constants are variable). These constants hold loop counters and limits as well as search threshold probabilities. The MEP Variables are not used by the meta chromosomes.

4.4.4 Meta Search Chromosome Operators

The operator sets for each meta-search chromosome are as follows:

Operator	Action
INDEX_SELF	Get own index in population
INDEX_RAND	Get a random index in population
FITNESS	Get fitness of argument index from training data
GREATER_THAN	Evaluate two nodes from MEP arguments & return result
IF_GOTO	Modify MEP instruction index if argument is true
STORE_TARGET	Update training data to indicate individual can be parent
GOTO	Unconditionally modify MEP instruction index

Table 7 -Meta Selection Chromosome Operators

Operator	Action
RAND01	Get random number between zero and one
GREATER_THAN	Evaluate two nodes from MEP arguments and return result
IF_GOTO	Modify MEP instruction index if argument is true
RAND32	Returns a random 32 bit integer
CCOPY	Copies node indexed by MEP argument to training data
INC	Adds one to 'Constant' node indexed by argument
GOTO	Unconditionally modify MEP instruction index

Table 8 -Meta Crossover Chromosome Operators

Operator	Action
RAND01	Get random number between zero and one
GREATER_THAN	Evaluate two nodes from MEP arguments and return result
IF_GOTO	Modify MEP instruction index if argument is true
MUT_NODE	Mutate training data node value
MUT_ADDR1	Mutate training data argument value
MUT_ADDR2	Mutate training data argument value
INC	Adds one to 'Constant' node indexed by argument
GOTO	Unconditionally modify MEP instruction index

Table 9 -Meta Mutation Chromosome Operators

The mutation mechanism takes place in the context of the chromosome where the mutation occur i.e. task chromosome nodes can only mutate into task operators or

variables and, say, selection chromosome nodes can only mutate into selection operators or Constants.

This is achieved by modifying the MEP node generation to add Constants, and generate each respective node type in the proportions of the original chromosome. The value of constants generated was from set four equally possible values, zero, the population size, and random number between zero and one or a random integer.

The meta implementations began with the selection, mutation and crossover chromosome set to the standard MEP values of tournament selection, uniform crossover and MEP-type mutation. If no parents resulted from the action of the select chromosome a random partner was chosen.

4.5 Experiment Results Methods

The software automatically logs to text files (when requested) all the required data to be collected to a comma separated values (CSV) file(s) suitable analysis purposes.

These mean values for the two variants over 100 runs will form the basis of comparison using the Mean Best Fitness and Average Fitness for each generation and problem.

A period of testing and tuning is required for a good meta-implementation. Changes to the software project in terms of the parameters varied during will be recorded and logged as the project continues.

The data will be analysed to determine statistical significance using similar measures as (Oltean, 2005b). That is, to check if the two datasets have the same variance using an F-test (or Fisher Test), which verifies the validity of the following T-test.

Then the T-test for statistical significance will be performed to determine if the differences are significant at a 95% confidence level.

For each test function the system will be run 100 times. The results for the Mean Best Fitness and the Mean Fitness are averaged across the 100 runs. The standard deviation is calculated according to the formula:

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \overline{x})^2}.$$

Which means

1. For each value x_i calculate the difference $x_i - \overline{x}$ between x_i and the average value $\overline{x}$.
2. Calculate the squares of these differences.
3. Find the average of the squared differences. This quantity is the variance σ^2.
4. Take the square root of the variance.

4.5.1 F-Test Method

The f-test value is ratio of the variances (when >1). The confidence level or p-value can then be checked against standard tables which show standard ratios for a F-distribution for a set number of degrees of freedom. The degrees of freedom can be calculated:

df = (Number of first samples -1) + (Number of second samples - 1)

If the calculated value is greater than the value in the tables the null hypothesis (that there is no differences in the means or variances) can be rejected.
In practice the Microsoft Excel (2007) FTEST formula was used to calculate the confidence level. Once the f-test has verified that a normal distribution is present in both samples the data can be compared using a standard t-test.

4.5.2 T-Test Method

The t-test establishes a confidence level that the differences between the standard MEP and the META MEP are not due to chance. This can be calculated by hand using the steps:

1. Calculated the sum of the MEP values and the sum of the Meta value.
2. Calculated the squares of the both sets of values and sum those
3. Calculate the t-ratio using the formula:

$$t = \frac{M_X - M_Y}{\sqrt{\left[\frac{\left(\Sigma X^2 - \frac{(\Sigma X)^2}{N_X}\right) + \left(\Sigma Y^2 - \frac{(\Sigma Y)^2}{N_Y}\right)}{N_X + N_Y - 2}\right] \cdot \left[\frac{1}{N_X} + \frac{1}{N_Y}\right]}}$$

Where:

Σ is the sum

M_X is the mean for the first set of values (X)

M_Y is the mean for the second set of values (Y)

N_X is the number of scores in the first set of values

N_Y is the number of scores in the second set of values

Similarly by calculating the degrees of freedom as with the f-test using the standard table the t-ratio can be compared to give a probability p-value.

In practice the Microsoft Excel (2007) TTEST formula was used to calculate the confidence level using the 2 tailed distribution and a two sample unequal variance parameters. i.e.

p -= TTEST(RANGE1, RANGE2, 2, 3)

5.0 Research Results

The data for each of the five test functions is presented in the graphs below. For each function two graphs are produced. One graph shows the Mean Best Fitness averaged over 100 runs for the generational version of both algorithms and the steady state version of both algorithms. The other graph shows the Mean Fitness of all individual for each generation for the generational and steady state versions. All the graphs show progress over 1000 generations. The graphs for each test function have the same fitness scale and range. The MEP and Meta-MEP ran using identical training data.

The fitness values represent the differences between the value obtained by the evolutionary computation and the actual training data real value. Therefore a lower value is fitter and represents an evolved solution closer the actual solution. A zero fitness shows a solution that exactly matches the training data has been found.

5.1 Plots of Results Data

5.1.1 Ackley Function

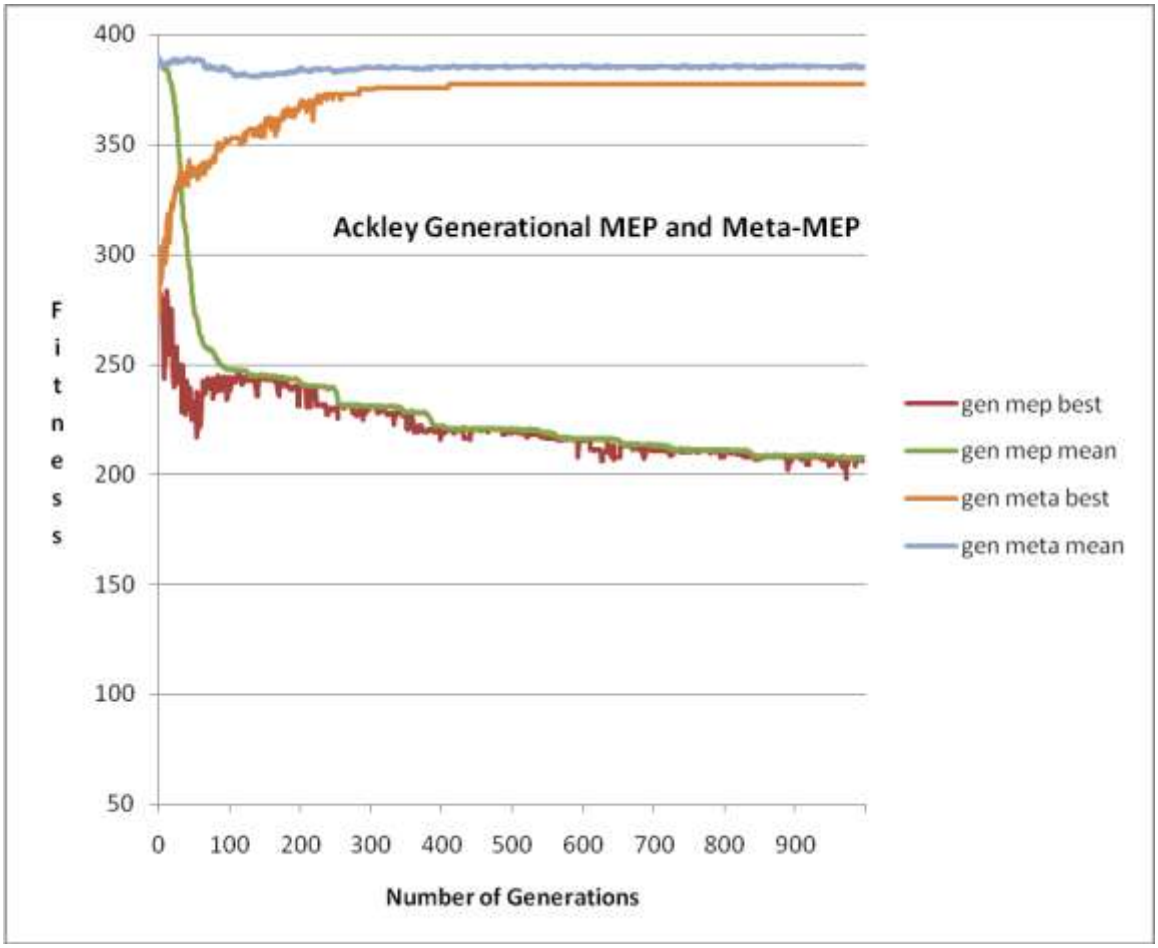

Figure 5 - Ackley Generational MEP and Meta-MEP Fitness

The Generation variant for the Ackley problem shows two different behaviours for the standard MEP and the Meta version. The MEP variant shows steep initial progress until about the 70th generation then a more gradual but slow progress of about 50 fitness divisions over 900 generations. It can be seen that after the initial promise the MEP best fitness is very close to the mean fitness which shows a lack of fitness diversity within the population.

In contrast the Meta version population fitness barely changes throughout the run and initially the best fitness increases. This implies parent selection by the select chromosome based on fitness is not taking place. There is more difference between

the mean and best fitnesses implying reasonable diversity but if fitter variants are not chosen as parents this has no effect.

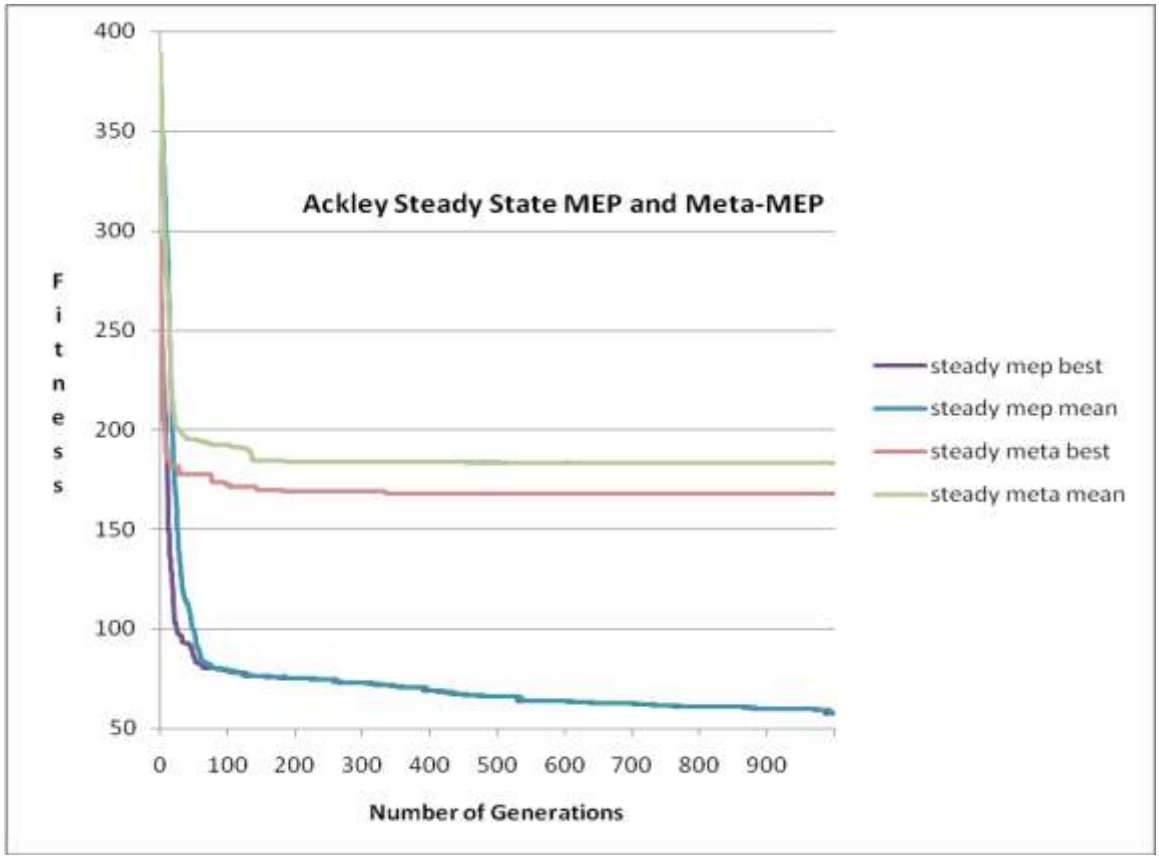

Figure 6 - Ackley Steady State MEP and Meta-MEP Fitness

With the steady state variant only fitter individuals replace less fit individuals, so for the meta variant progress stops after about 100 generations. The meta search is then concerned with finding search chromosomes that are superior to the initial standard MEP. The lack of progress shows this is a challenging problem.

The standard MEP version makes better progress than the Meta-MEP initially and overall but slows to a flatter curve with little population diversity making it unlikely that jumps to produce significantly fitter individuals are taking place.

The steady state Meta-MEP consistently achieves a better MBF and average fitness than the generational standard MEP over 100 generations for the Ackley problem but the steady state MEP shows the best average performance for this test.

5.1.2 Griewank Function

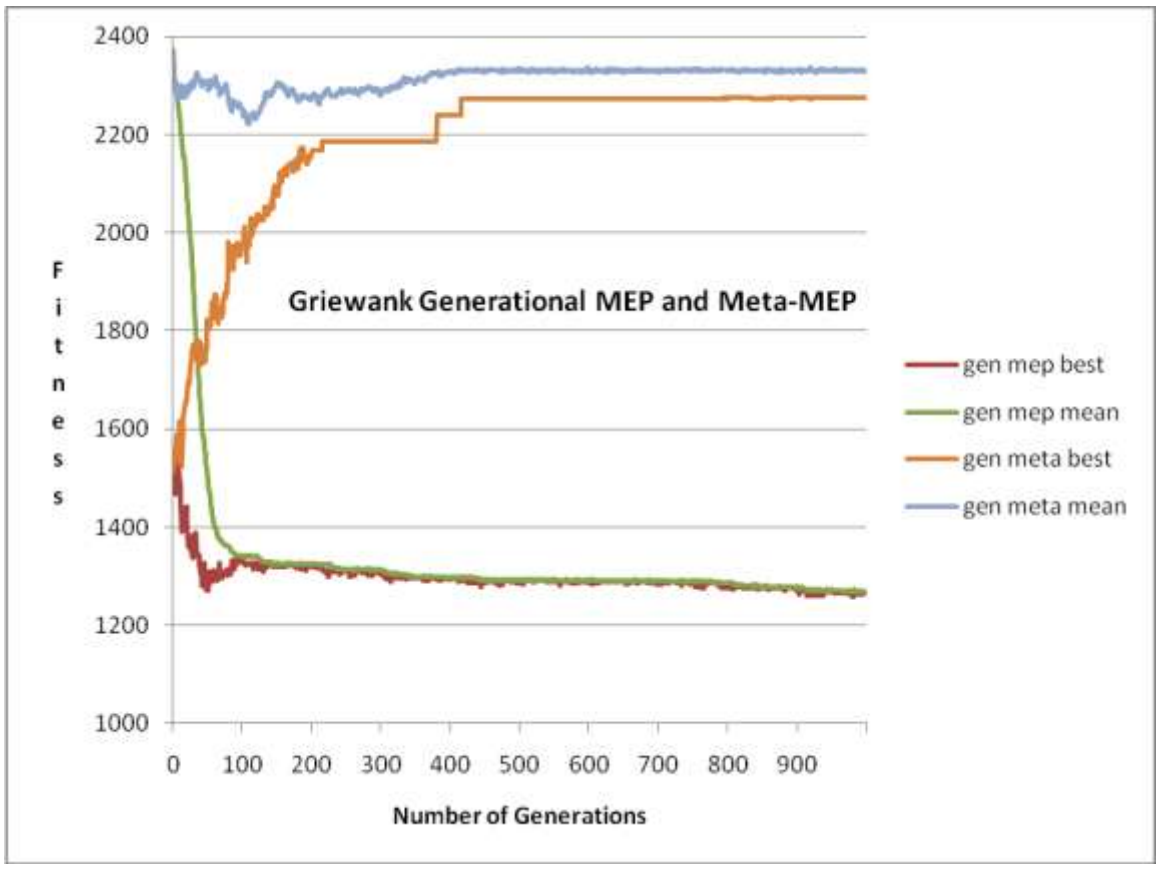

Figure 7 Griewank Generational MEP and Meta-MEP Fitness

The standard MEP makes good initial progress until about 100 generations has been reached then the lack of diversity causes progress to slow. The Meta-MEP variant shows best fitness (MBF) initially increasing and mean fitness not decreasing. By about 430 generations mean and best fitness are almost constant but distinct. This shows fitness evolution no longer taking place suggesting the either fit parents are not being selected or fitter children are not being produced or both.

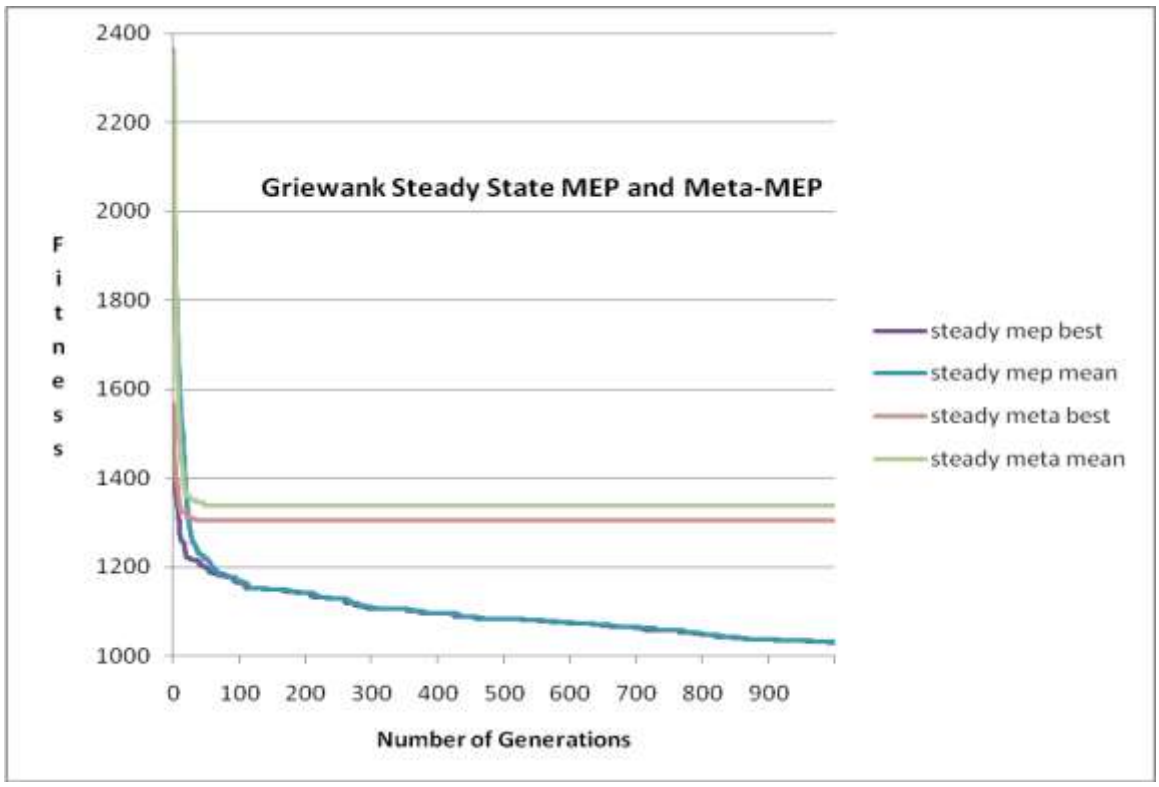

Figure 8 - Griewank Steady State MEP and Meta-MEP Fitness

The Steady State model for the Griewank problem shows the MEP and Meta versions making initial steep increases in fitness up until about generation 40. Subsequently the MEP version continues making progress at a slower pace while the Meta version stagnates searching for search operators that improve task fitness while retaining greater individual fitness diversity.

The steady state MEP shows the best performance for the Griewank problem and generational MEP is also superior to both Meta variants

5.1.3 Rastrigin Function

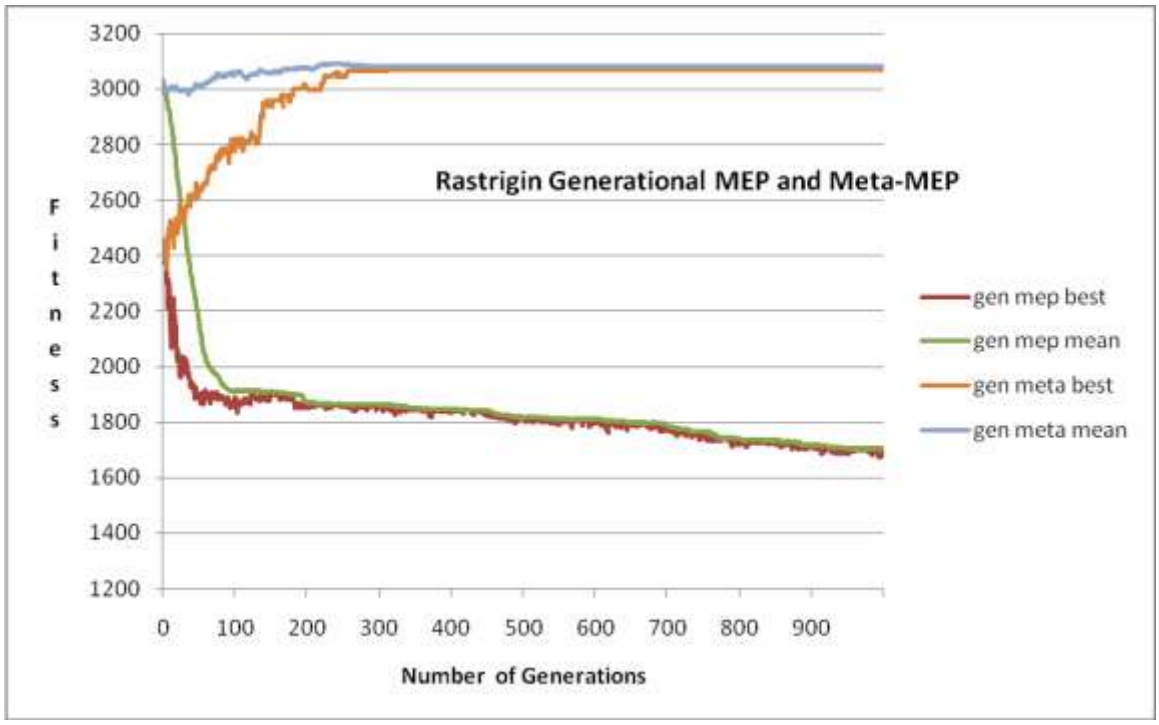

Figure 9 - Rastrigin Generational MEP and Meta-MEP Fitness

The generational Rastrigin shows a similar pattern to that of the Rosenbrock problem. The Meta-MEP variant show an immediate increase in both mean and best fitness and both converge after about 250 generations at a higher level than the original mean fitness showing very little individual fitness diversity from then on. This implies a lack of ability to choose parents with better fitness or the ability to produce children that vary in fitness compared to their parents or both.

The MEP version makes good initial fitness gains until about the 80th generation then fitness diversity declines and progress is slower as less large fitness improvements are not evolved.

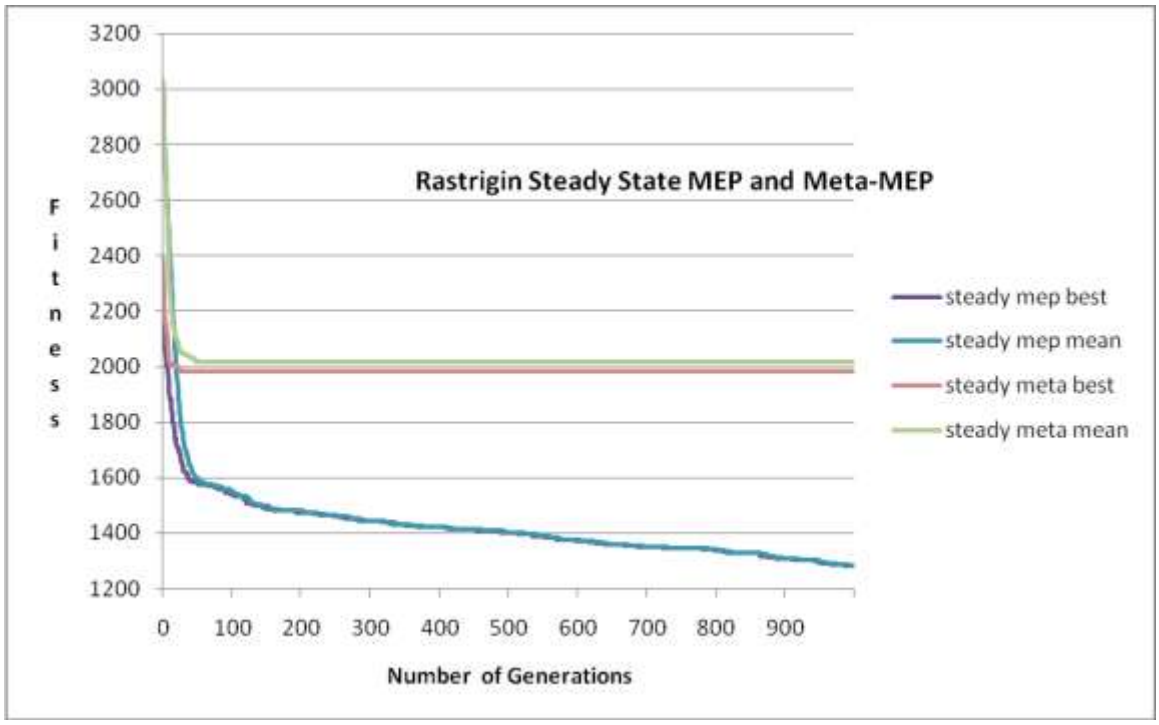

Figure 10 - Rastrigin Steady State MEP and Meta-MEP Fitness

Again the steady state Rastrigin shows a similar pattern to that of the Rosenbrock problem. Initial large gains in fitness are made by both the standard MEP and the Meta-MEP After about 15 generations new Meta-MEP fitter parents are not being chosen and fitter children than their parents are not being produced.

Both the generation and the steady state MEP are superior to the meta variants

5.1.4 Rosenbrock Function

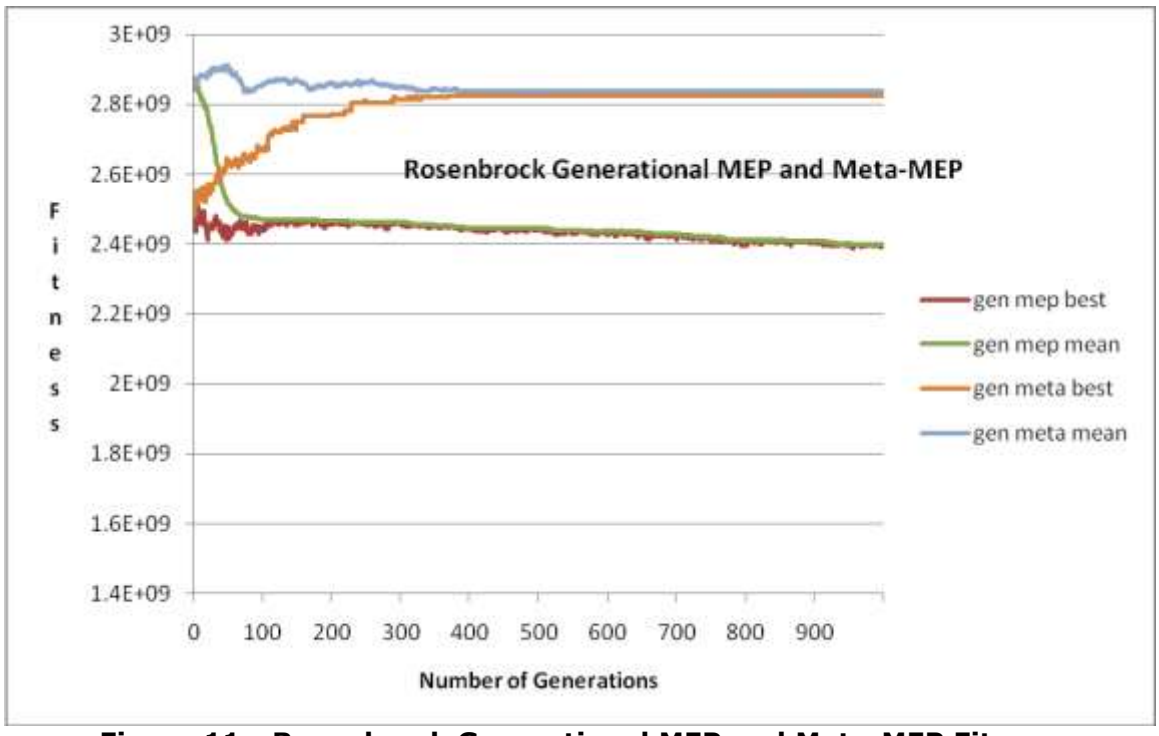

Figure 11 - Rosenbrock Generational MEP and Meta-MEP Fitness

The generational Rosenbrock shows the MEP variant best fitness making little progress over the generations and the mean fitness initially declines and is almost the same as the best fitness after about 70 generations. After this loss of diversity the MEP makes very slow progress.

The Meta MEP immediately begins to increase in both best and mean fitness initially and the initial best fitness value is never regained. The meta mean fitness does decline slightly but meets the best fitness after about 350 generations and after this no progress in fitness improvement is made at all.

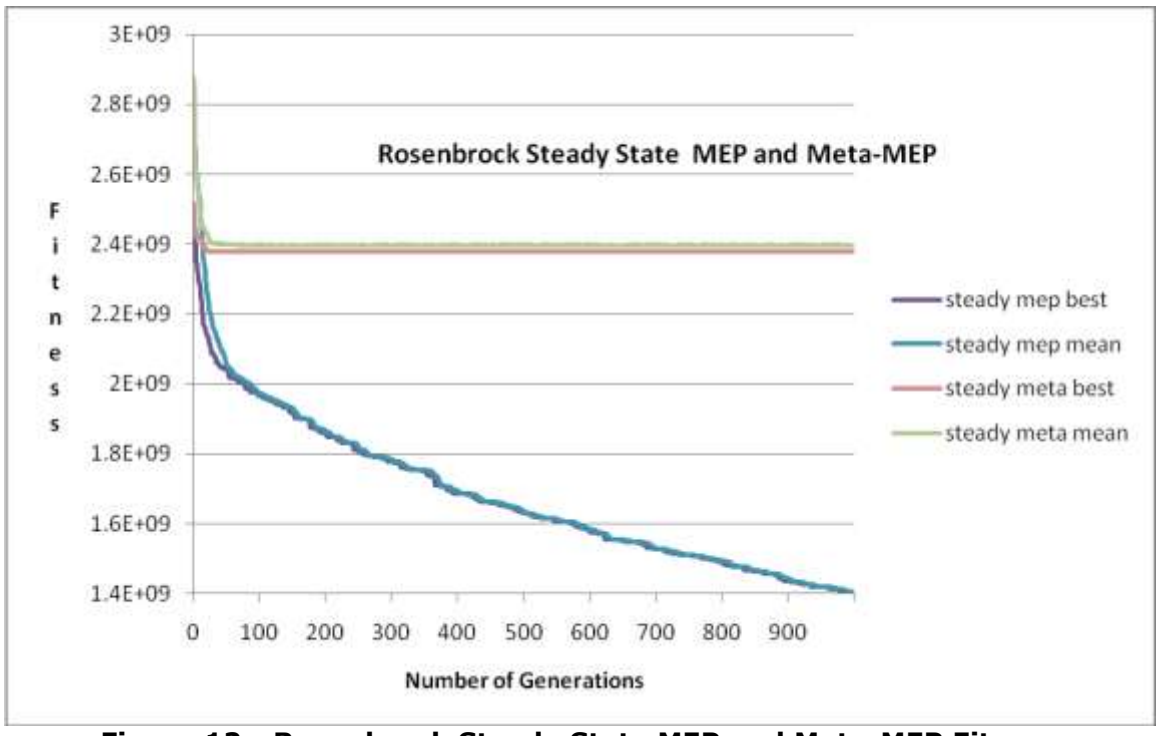

Figure 12 - Rosenbrock Steady State MEP and Meta-MEP Fitness

The steady state Rosenbrock shows very steep increases in fitness during the first few generations for both standard and Meta MEP. The MEP variant continues to make reasonable progress but progress with the Meta version stagnates as the search operators evolve to be ineffective. Consequently the meta-MEP maintains a constant diversity. The MEP variant's diversity is restricted after about 60 generations.

The steady state Meta-MEP is superior to the generational MEP but the steady state MEP shows the best overall performance for the Rosenbrock problem.

5.1.5 Schwefel Function

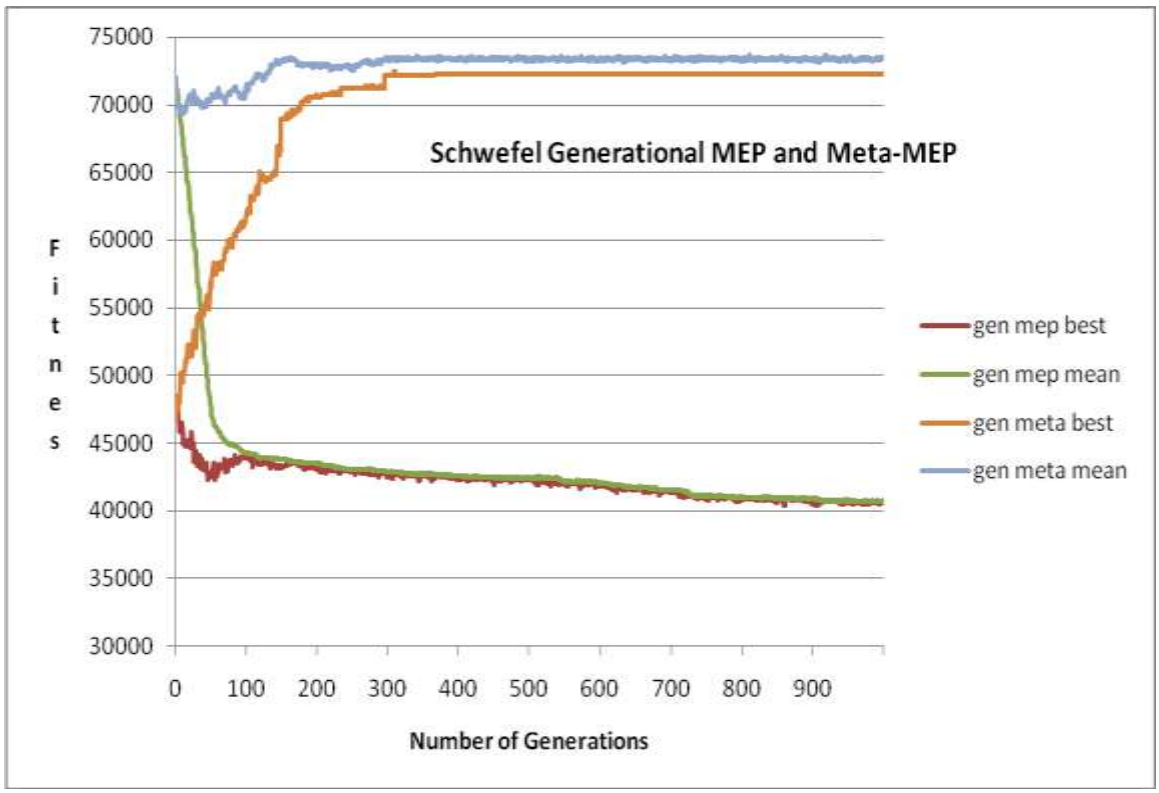

Figure 13 - Schwefel Generational MEP and Meta-MEP Fitness

Continuing the theme, the Schwefel generational run shows the MEP making initial mean fitness gains but the best MEP fitness makes slower progress and the pattern of low diversity and slow fitness gains is established after 100 generations. The Meta-MEP decreases in mean fitness and sharply in best fitness but retains diversity after reaching equilibrium after 300 generations implying lack of ability to choose most fit parents but perhaps some activity by the crossover and mutation chromosomes.

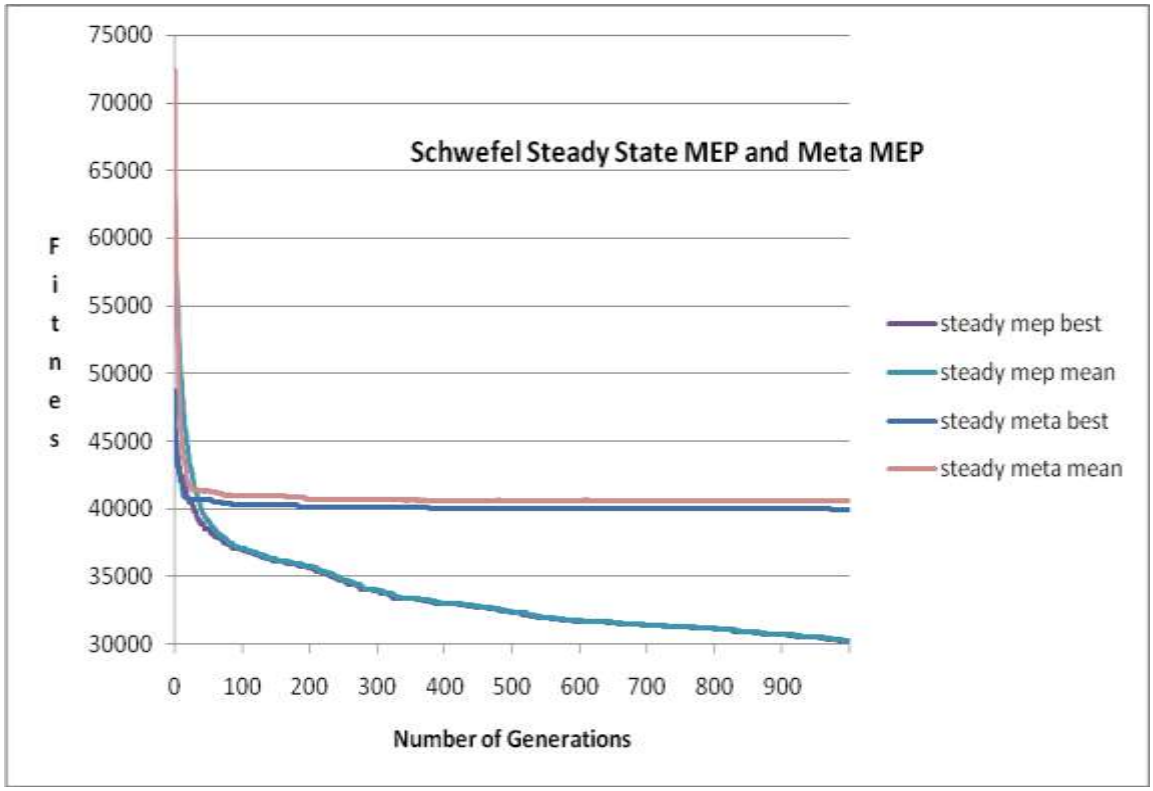

Figure 14 - Schwefel - Steady State MEP and Meta-MEP Fitness

The Schwefel MEP implementation makes good progress for about 50 generations but then looses diversity and makes slower thereafter. The Meta-MEP continues to make only marginal progress after about 20 generations and maintains some fitness diversity. Some small gains in the best fitness show after 200, 380 and 980 generations.

The MEP version again makes more gradual progress after about 40 generations following the initial steep fitness improvement.

The steady state Meta-MEP achieves better fitness than the MEP generational but the steady state MEP shows the best performance overall.

5.2 Statistical Significance

The tables below shows the confidence p values for f-tests and t-tests for the results obtained, comparing each generational version and each steady state version of the standard and meta implementations of MEP.

5.2.1 Generational Statistical Results

Test Function	Mean Best Fitness		Mean Fitness	
	f-test p value	t-test p value	f-test p value	t-test p value
Ackley	0.055928	n/a	0	0
Griewank	0	0	0	0
Rastrigin	3.48E-39	0	0	0
Rosenbrock	0	8.3E-234	0	0
Schwefel	0	0	0	0

Table 10 - Generational f-test and t-test p values for Best and Mean Fitness

The f-test Ackley function shows that the differences in distributions for MBF for the MEP and Meta-MEP are only 94% likely not to be due to chance so the results of the t-test have been disregarded.

The other comparisons show valid t-test results indicating that there is great certainty that the standard MEP and Meta-MEP methods produce statistically significant differing results. All zero values are less than 1E-308.

5.2.2 Steady State Statistical Results

Test Function	Mean Best Fitness		Mean Fitness	
	f-test p value	t-test p value	f-test p value	t-test p value
Ackley	2E-186	0	8.4E-117	0
Griewank	0	0	2.74E-85	0
Rastrigin	0	0	4.3E-175	0
Rosenbrock	0	0	0	0
Schwefel	6.8388E-137	0	0	0

Table 11 - Steady State f-test and t-test p values for Best and Mean Fitness

For the steady state results all values show that the samples are normally distributed and are suitable for t-test comparison. The t-tests show that the MEP and meta-MEP

variants produce results that are statistically significantly different. Again All zero values are less than 1E-308.

5.3 Best Fitness and Run Times

The data below show the overall numbers for the test runs and highlights the practical aspects of actually solving the test problem and how long the software took trying to do this.

It is interesting to note that none of the test configurations actually resulted in a zero fitness score where the test problem was solved. This shows that all the variations require additional work to solve the test problems.

5.3.1 Absolute Best Fitness Levels

The table below shows the best fitness levels achieved by each variant over all the 100 runs.

Test Function	Generational		Steady State	
	MEP	Meta-MEP	MEP	Meta-MEP
Ackley	15.37956629	69.53575687	13.35368913	15.37896337
Griewank	964.86	1036	806.868	1042.91
Rastrigin	1188.247319	1554.641702	903.357	1582.091917
Rosenbrock	1749917709	2027679961	814133913.8	2043542254
Schwefel	25888.2	38717	20470.2	28400.6

Table 12 - Absolute Best Fitness - all runs

These fitness scores show that the steady state MEP consistently produces the most fit solution during the test runs. For the Ackley problem Meta-MEP produced a very slightly fitter solution the generational MEP but not for the other test problems.

5.3.2 Run Times

The table below shows the time in decimal hours that each variant took for the 100 runs.

Test Function	Generational		Steady State	
	MEP	Meta-MEP	MEP	Meta-MEP
Ackley	61.41	50.33	65.35	43.83
Griewank	61.43	43.88	62.24	44.58
Rastrigin	60.11	43.22	62.24	42.33
Rosenbrock	56.86	52.02	56.77	37.69
Schwefel	59.53	48.98	57.74	42.21

Table 13 - Time for 100 Runs in Decimal Hours

These times in hours show that all the test runs took about 2 to 2 1/2 days each to run.

5.4 Summary of Results

The graphs of the results show a common thread for the generational and steady state models.

5.4.1 Generational Model

The generational model shows the standard MEP variants to show good progress initially. After 100 generations or less progress slows and the best and mean fitnesses converge showing low fitness diversity within the population.
For the meta variants best fitness increases and mean fitness either slightly increases of does not improve. In most cases some fitness diversity is maintained but no further progress is made.

5.4.2 Steady State Model

The steady state model show the MEP variant making good progress for longer than in the generational model then progress declines but is still better than the generational model. Similarly the fitness diversity reduces coinciding with the decreased performance.

The steady state Meta-MEP shows fitness steeply declining alongside the MEP variant initially but then fitter solution are no longer produced. The meta population loses the ability to produce fitter children either because the most fit parents are not being selected or children with fitness changes are not being produced or both. Good fitness diversity is maintained as the existing successful individuals are not replaced.

5.4.3 General Observations

The steady state MEP consistently produces the best results. Progress for the generational and steady state MEP continues but at a faster rate the steady state model.

The generational model for Meta-MEP stagnates as the initial individuals with intact search operators are lost and effective new search chromosomes are not produced whereas in the steady state model the initial fitter individuals survive but individuals with search operators that result in improve fitness do not evolve within 1000 generations.

The statistics show that the MEP and Meta-MEP are statistically significantly different but the results plots show that the Meta-MEP is not superior to the standard MEP.

6.0 Conclusions

6.1 Review of Project

This project began with a general introduction to evolutionary computation and then focused on design aspects intended to reduce the human and computational effort to solve problems.

The project highlighted attempts to change aspects of the basic search metaphor by evolving solutions to the problem of evolving of solutions to problems themselves. This is meta-evolution.

This research attempted to determine if fitter solutions could be produced for the same computation effort, in terms of number of generations, with a meta-implementation compared with the existing non-meta version.

A method of implementing meta-evolution was proposed based on previous work that was shown to be effective for non-meta and meta implementations which is Multi Expression Programming . This was to evolve the search operators; selection, crossover and mutation.

A novel method to evolve these search operators in the context of the problem evolution was designed and constructed. This was to implement the search operators as separate chromosomes within an individual where the action of the search chromosome implemented each search operator.

The method was validated by implementing the standard non-evolving MEP search operators in this way. The 'standard' MEP results were then compared with the meta variant where the individuals evolve by the search operator acting both on the task chromosome and on themselves.

Five test problems considered difficult were used to benchmark the 'standard MEP' against the Meta-MEP. The fitnesses achieved for both variants were plotted using a generational and steady state population replacement strategy.

6.2 Research Question Considerations

After obtaining the results running the standard MEP against the Meta-MEP variants it is shown that the standard MEP consistently produces individuals with better fitness than the Meta-MEP variants.

It is shown that a meta-evolutionary implementation of multi-expression programming does NOT produce fitter individuals than standard multi-expression programming when tested using the Ackley, Griewangk, Rastrigin, Rosenbrock and Schwefel functions.

This result has shown that MEP meta-search chromosomes cannot evolve at a sufficient rate to improve over the fixed MEP search chromosome if they are evolved by exactly the same method as the task chromosome under the conditions of this meta-implementation. This conclusion can be seen in the light of the No Free Lunch theorems but also in the light of biological evolution.

6.3 Further Work

While the Meta-MEP may not improve over the 'standard' MEP it does make progress in solving the test problems under some conditions. It is common with EC implementations to have an extended period of exploration to learn about the characteristics of the evolution may be necessary and so improve on it.

During the design and implementation of Meta-MEP some decisions were made which are likely to have affected performance. Further work can be done to vary these characteristics.

The most productive area for further work is to explore lengthening the very short search chromosome length to allow more variants of the MEP code in them. Other aspects to explore include mutation rates, population sizes and mutation mechanisms.

7.0 References

Angeline P J, Pollack J B (1994), *Coevolving High-Level Representations*, Artificial Life III.

Angeline P J (1995), *Two Self-Adaptive Crossover Operations for Genetic Programming*, Advances in Genetic Programming Vol. 2, Ed. Angeline P J & Kinnear, K E, MIT Press.

Au W, Chan K C, Yao X (2003) *A novel evolutionary data mining algorithm with applications to churn prediction*, IEEE Transactions on Evolutionary Computation, Vol.7, Iss.6, 532-545

Back T (1992), *Self-Adaptation in Genetic Algorith*ms, Varela and Bourgine, [723], p263-271.

Blum S (2005), *Adaptive Mutation Strategies for Evolutionary Algorithms*, DYNARDO - Dynamic Software and Engineering GmbH, Weimar, Germany.

Cramer N L (1985), *A Representation for the Adaptive Generation of Simple Sequential Programs*, [ICGA85], p183-187.

De Jong K A (1975), *An Analysis of the Behaviour of a Class of Genetic Adaptive Systems*, Doctoral thesis, Dept. of Computer and Communication Sciences, University of Michigan, Ann Arbor.

Darwen P, Yao X (1997), *Every niching method has its niche: Fitness Sharing and Implicit Sharing compared*, Fourth International Conference on Parallel Problem Solving from Nature (PPSN-4) , p398-407. Berlin, Germany.

Darwin C (1859), *The Origin of Species*, John Murray

Dioşan L, Oltean M (2006), *Evolving crossover operators for function optimization*, EuroGP2006 & EvoCOP2006, Lecture Notes in Computer Science, Springer Verlag Berlin, Budapest, Hungary, p97-108.

Dioşan L, Oltean M (2007), *Evolving Evolutionary Algorithms using Evolutionary Algorithms*, GECCO 2007, London, UK

Edmonds B (2001), *Meta-Genetic Programming : Co-evolving the Operators of Variation,* Centre for Policy Modelling, Manchester Metropolitan University, Manchester, UK.

Eiben A E (2006), *Boosting Genetic Algorithms with Self-Adaptive Selection*, 2006 IEEE Congress of Evolutionary Computation, Vancouver, Canada.

Eiben A E, Smith J E (2003*), Introduction to Evolutionary Computing*, Springer-Verlag, Berlin.

Ferreira C (2001), *Gene Expression Programming: A New Adaptive Algorithm for Solving Problems,* Departamento de Ciencias Agrarias, Universidade dos Acores, Angra do Heroismo, Portugal.

Ferreira C (2002), *Re: [GP] MEP better than GEP,* Yahoo Groups: Genetic Programming, http://tech.groups.yahoo.com/group/genetic_programming/message/641 (accessed 17/11/2007)

Fogarty T C (1989), *Varying the probability of mutation in genetic algorithms*, Proceedings of Third International Genetic Algorithms.

Fogel L J, Owens A, Walsh M (1966), *Artificial intelligence through simulated evolution*, Wiley, New York.

Fogel L J, Angeline P J, Fogel D B (1995), *An Evolutionary Programming Approach to Self-Adaption on Finite State Machines*, Proceedings of the Forth Annual Conference on Evolutionary Programming, Ed: McDonnel, Reynolds et al, MIT Press.

Goldberg D (1989), *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley.

Greffenstette J J (1985), *Optimization of Control Parameters for Genetic Algorithms*, Vanderbilt University, Nashville, TN.

Groşan C (2004), *A comparison of several algorithms and representations for single objective optimization*, Genetic and Evolutionary Computation Conference (GECCO), Ed. Deb K et al., Springer-Verlag Germany, p788-790, Seattle.

Hadjam F Z, Moraga C, Benmohamed M (2007), *Cluster-based Evolutionary Design of Digital Circuits using Improved Multi-Expression Programming*, GECCO 2007, London, UK

Hedar A (2007), Global Optimisation Test Functions, web: http://www-optima.amp.i.kyoto-u.ac.jp/member/student/hedar/Hedar_files/TestGO_files/Page364.htm, (accessed 19/8/2007).

Hinterding, R (1997), *Self-adaptation using Multi-chromosomes*, Fourth International IEEE Conference on Evolutionary Computation, Ed. Eberhart R, Angeline P, IEEE.

Hinterding R, Michalewicz Z, Eiben E A (1997), *Adaptation in Evolutionary Computation: A Survey*, IEEECEP: Proceedings of The IEEE Conference on Evolutionary Computation, IEEE World Congress on Computational Intelligence

Holland J (1975), *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, USA

Jensen M T (2003), *Generating robust and flexible job shop schedules using genetic algorithms*, IEEE Transactions on Evolutionary Computation, Vol. 7, Iss. 3, p275-288

Kantschik W (1999), *Meta-Evolution in Graph GP*, Proceedings of EuroGP'99, LNCS, Vol. 1598. Springer-Verlag, p15-28.

Keane A J, Brown SM (1996) *The design of a satellite boom with enhanced vibration performance using genetic algorithm techniques*, Adaptive Computing in Engineering Design and Control '96 - Proceedings of the Second International Conference, I.C. Parmee (ed), 107-113. University of Plymouth.

Koza J R (1990), *Genetic Programming: A Paradigm for Genetically Breeding Populations of Computer Programs to Solve Problems*, Computer Science Department, Stanford University, Margaret Jacks Hall, Stanford, CA

Koza J R (1992a), *Genetic Programming II: Automatic Discovery of Reusable Programs*, MIT Press, Cambridge, MA.

Koza J R (1992b), *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, MIT Press, Cambridge, MA.

Koza J R (1994), *Scalable Learning in Genetic Programming using Automatic Function Definition*, Advances in Genetic Programming Vol. 1, Ed. Kinnear K E, MIT Press.

Koza J R, Bennett F H, Andre D, Keane M A (1996), *Automated WYWIWYG Design of Both the Topology and Component Values of Electrical Circuits Using Genetic Programming*, Genetic Programming 1996: Proceedings of the First Annual Conference, Ed. Koza J R, Goldberg D E, Fogel D B, Riolo R L

Langdon W B, Poli R, (2002), *Foundations of Genetic Programming*, Spinger-Verlag

Lis J (1996), *Parallel Genetic Algorithm with the Dynamic Control Parameter,* International Conference on Evolutionary Computation, p324-329.

Matsumoto M, Nishimura T (1998), *Mersenne Twister: A 623-dimensionally equidistributed uniform pseudorandom number generator*, ACM Trans. on Modelling and Computer Simulation Vol. 8, No. 1, January p3-30

Nordin P (1994), *A Compiling Genetic Programming System that Directly Manipulates the Machine Code*, Cambridge, MIT Press.

Oltean M (2004), *Multi Expression Programming Software Source Code*, http://www.mep.cs.ubbcluj.ro/Source%20Code.htm (accessed 24/3/2004).

Oltean M (2005a), *Improving the Search by Encoding Multiple Solutions in a Chromosome*, Evolutionary Machine Design, Chapter 15, Nova Science Publisher, New-York, Ed. Nedjah N et al.

Oltean M (2005b), *Evolving evolutionary algorithms using Linear Genetic Programming*, Evolutionary Computation, MIT Press, Cambridge, Vol. 13, Issue 3.

Oltean M, Dumitrescu D (2002), *Multi Expression Programming*, Technical-Report, UBB-01-2002, Faculty of Mathematics and Computer Science, Babes-Bolyai University, Cluj-Napoca, Romania.

Oltean M, Groşan C (2003), *Evolving Evolutionary Algorithms using Multi Expression Programming*, The 7th European Conference on Artificial Life, September 14-17,

2003, Dortmund, Ed. Banzhaf W et al, LNAI 2801, p651-658, Springer-Verlag, Berlin.

Oltean M, Groşan C (2004), *A Comparison of Several Linear Genetic Programming Techniques,* Complex Systems Vol. 14, No. 4.

Samsonovich A V, De Jong K A (2005), *Pricing the 'Free Lunch' of Meta-Evolution*, GECCO 2005, Washington, DC, USA

Shan Y (2003), *Program Evolution with Explicit Learning: a New Framework for Program Automatic Synthesis,* School of Computer Science, University College, University of New South Wales, Australia Defence Force Academy, Canberra, Australia.

Schmidhuber J (1987), *Evolutionary principles in self-referential learning*, Diploma thesis, Technical University, Munich.

Spears W M (1995), *Adapting Crossover in a Genetic Algorithm,* Naval research Laboratory, Washington DC.

Spector L (2001), *Auto constructive Evolution: Push, PushGP, and Pushpop*. Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001), Ed. Spector L et al, p137-146, San Francisco, California, USA, 2001. Morgan Kaufmann.

Spector L, Bernstein H J (2003), *Communication capacities of some quantum gates, discovered in part through genetic programming,* Proceedings of the Sixth International Conference on Quantum Communication, Measurement, and Computing. Ed. Shapiro, Jeffery H & Hirota O, Princeton, NJ: Rinton Press. p500-503

Syswerda G (1989) Uniform *crossover in genetic algorithms*, Proceedings of the Third International Conference on Genetic Algorithms, Ed. Schaffer J, Morgan Kaufmann

Tavares J (2004), *On the Evolution of Evolutionary Algorithms*, Proceedings of the EuroGP, 7th European Conference on Genetic Programming, Coimbra, Portugal

Teller A, Veloso M (1996), *A New Learning Architecture for Object Recognition*, Symbolic Visual Learning, Oxford University Press, p81-116.

Thierens D (2002), *Adaptive Mutation rate control schemes in genetic algorithms,* Institute of information and computing sciences, Utrecht University, Netherlands.

Tuson A, Ross P (1996), *Co-evolution of Operator Settings in Genetic Algorithms*, Proceedings of the Third AISB Workshop on Evolutionary Computing, Springer.

Wagner P W, Altenberg L (1996). *Complex adaptations and the evolution of evolvability*, Evolution 50, p967-976

Wang G, Goodman E D, Punch W F (1996), *Simultaneous multi-level evolution*, 2nd Online Workshop on Evolutionary Computation (WEC2).

White T, Oppacher F (1994), *Adaptive Crossover Using Automata*, Proceedings of the 3rd PPSN, Ed. Davidor Y, Schwefel H.

Wolpert D H, Macready W G (1995), *No Free Lunch Theorems for Search*, The Sante Fe Institute, NM.

Yao, X (1999), *Evolutionary programming made faster*, IEEE Transactions on Evolutionary Computation, Vol. 3(2), p82-102.

Zijlstra, E (2008) *A C++ command line parser*, web: http://sourceforge.net/projects/amiclargs, (accessed 1/2/2008).

8.0 Index

9.0 Appendices

9.1 Implementation Software Design

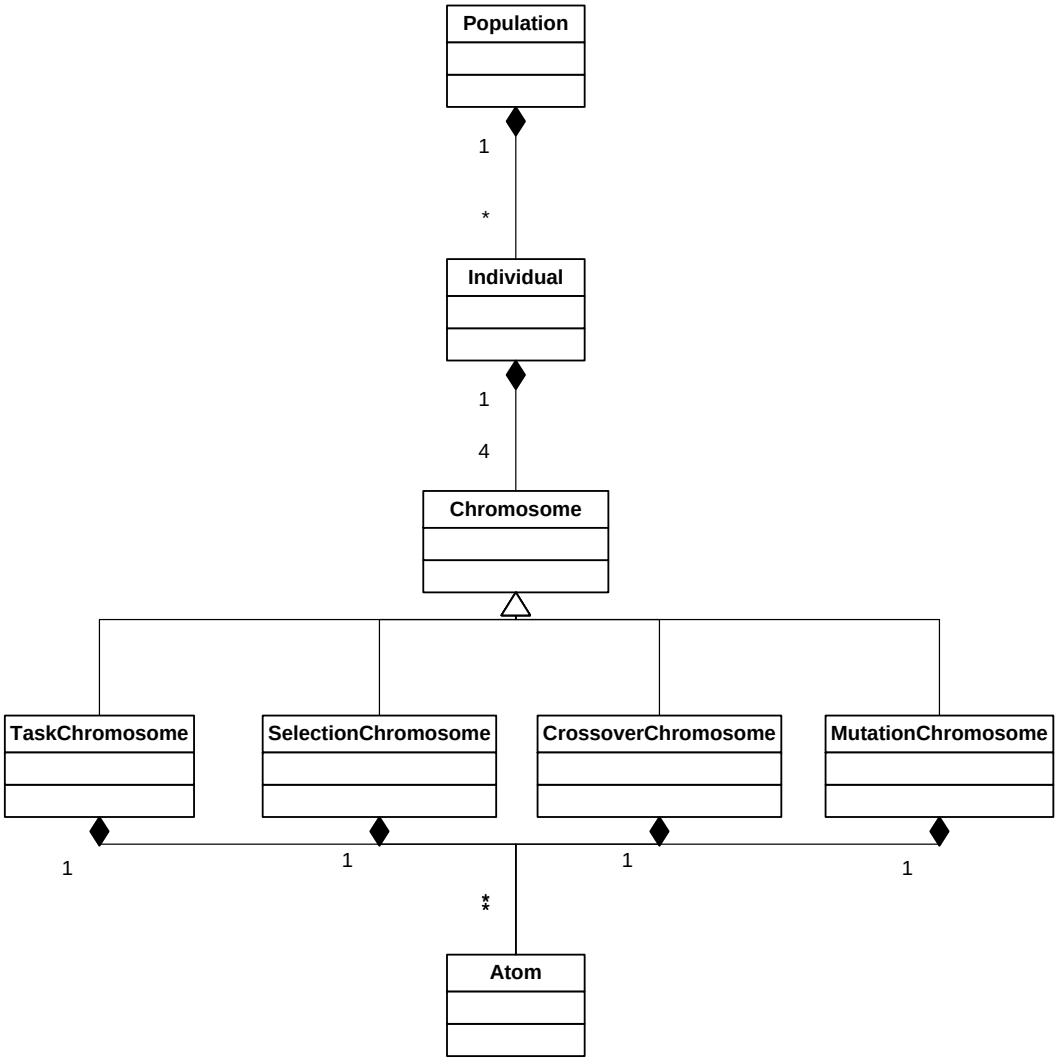

Figure A – UML Class Diagram for Meta-Genetic Programming System

The software was implemented using portable standard C++ code and the std::lib deque container class, as well as a third party random number generator and

command line parser (see below). Using these techniques should allow any modern C++ compiler to run the program.

9.2 Implementation Software and Hardware

The software for this project was implemented in Microsoft Visual C++ 2008 as a console application running under the Windows x64 system.

The test runs were done on two Dual 2.2Ghz Opteron Dual Core PC's running Windows Server 2003 x64

9.3 Source Code and Test Data Access

The source code and test data for this project can be found at:

http://www.luisamark.com/sfiles/ (user name: anon, password: fountain)

9.4 The Ackley Test Function

The function can be expressed (Hedar, 2007) as:

$$f(\mathbf{x}) = 20 + e - 20e^{-\frac{1}{5}\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}} - e^{-\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)}.$$

It can be written as:

```
f(x) = 20+e-20*exp(-0.2*(sqrt((1/n)*sum(x(i)^2))-
          exp((1/n)*sum(cos(2*∏*x(i)))
```

The function can be shown for n=2 as:

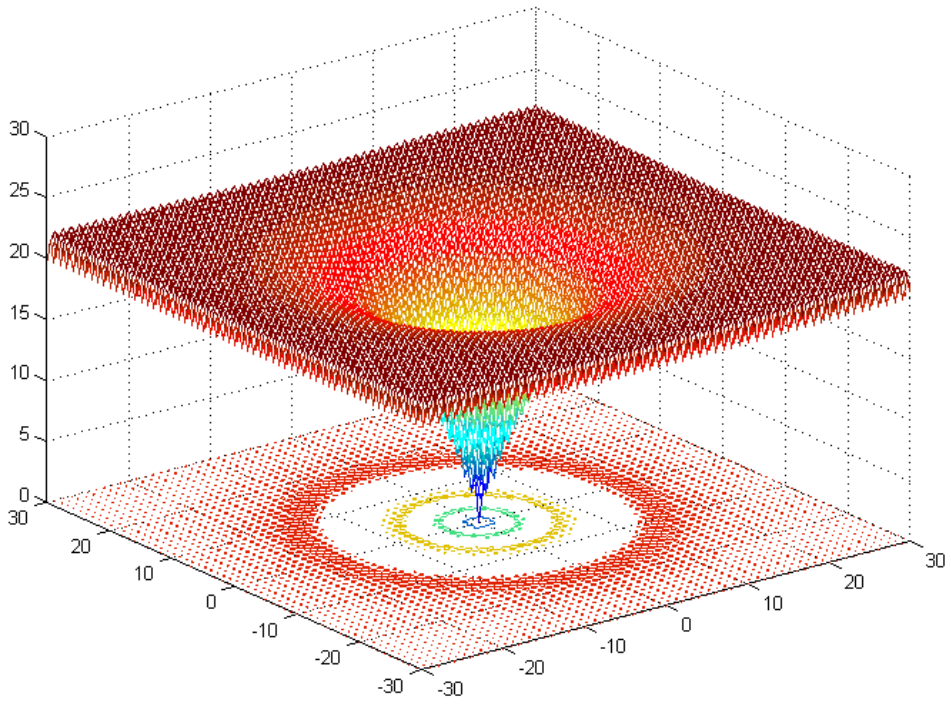

Figure E - Ackley Function for n=2 (Hedar, 2007)

In (Dioşan & Oltean, 2006) the domain is [-32,32], a=20, b=0.2, c=2∏ and n=5.

An algorithm to generate data for n=5 is:

```
n = 5;
a = 20; b = 0.2; c = 2*pi;
s1 = 0; s2 = 0;

for i=1 to n
   s1 = s1+x(i)^2;
   s2 = s2+cos(c*x(i));
end
y = -a*exp(-b*sqrt(1/n*s1))-exp(1/n*s2)+a+exp(1);
```

9.5 The Griewangk Test Function

The function can be expressed (Hedar, 2007) as:

$$f(\mathbf{x}) = \sum_{i=1}^{n} \frac{x_i^2}{4000} - \prod_{i=1}^{n} \cos\left(x_i/\sqrt{i}\right) + 1.$$

It can be written as:

```
f(x) = 1/4000*sum(xi-100)^2 - prod((xi-100)/sqrt(i)) + 1
```

The function can be shown for n=2 as:

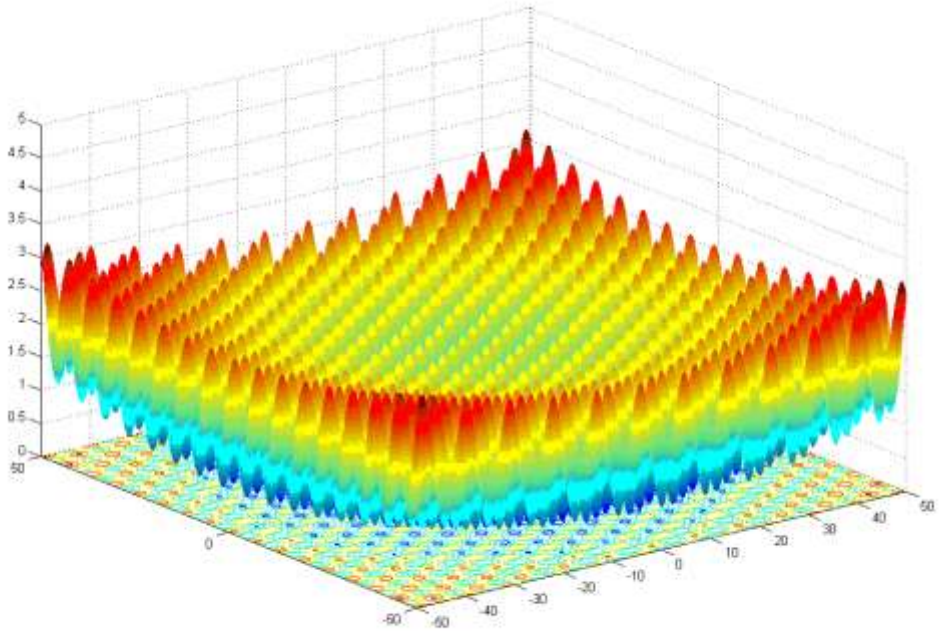

Figure B - Griewangks Function for n=2 (Hedar, 2007)

In (Dioşan & Oltean, 2006) the domain is [-500,500] and n=5.

An algorithm to generate data for n=5 is:

```
n = 5
fr = 4000
s = 0
p = 1

for j = 1 to n
        s = s+x(j)^2
end

for j = 1 to n
        p = p * cos (x(j)/ sqrt (j))
end

y = s/fr-p+1
```

9.6 The Rastrigin Test Function

The function can be expressed (Hedar, 2007) as:

$$f(\mathbf{x}) = 10n + \sum_{i=1}^{n} \left(x_i^2 - 10\cos\left(2\pi x_i\right)\right).$$

It can be written as:

```
f(x) = 10.0*n + sum(x(i)^2 - 10.0*cos(2*∏*x(i)))
```

The function can be shown for n=2 as:

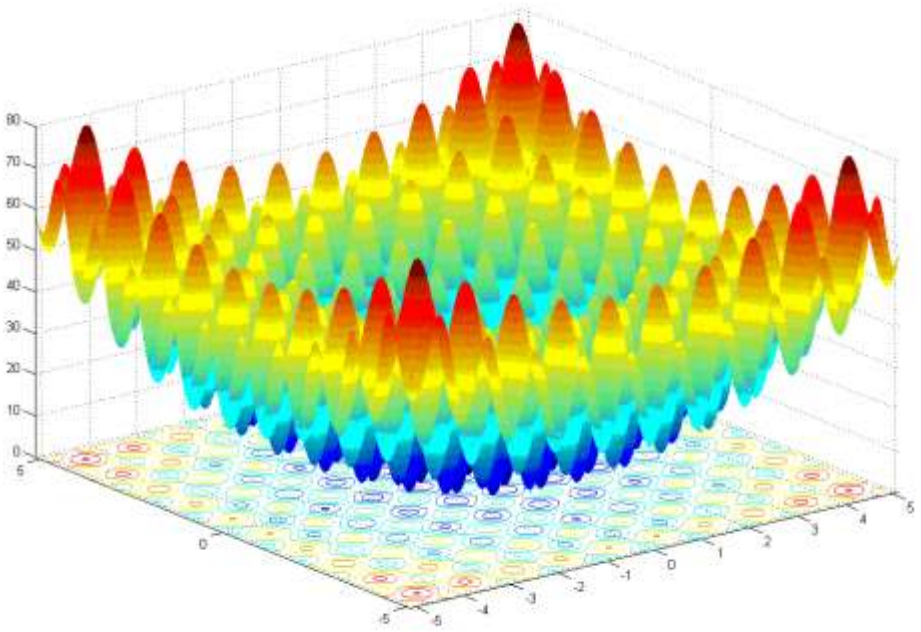

Figure D - Rastrigin Function for n=2 (Hedar, 2007)

In (Dioşan & Oltean, 2006) the domain is [-5,5] and n=5.

An algorithm to generate data for n=5 is:

```
n = 5;
s = 0;
for j = 1 to n
    s = s+(x(j)^2-10*cos(2*pi*x(j)));
end
y = 10*n+s;
```

9.7 The Rosenbrock Test Function

The function can be expressed (Hedar, 2007) as:

$$f(\mathbf{x}) = \sum_{i=1}^{n-1} \left[100 \left(x_i^2 - x_{i+1} \right)^2 + \left(x_i - 1 \right)^2 \right].$$

It can be written as:

```
f(x) = sum(100*(x(i)-x(i-1)^2)^2 + (1-x(i-1))^2
```

The function can be shown for n=2 as:

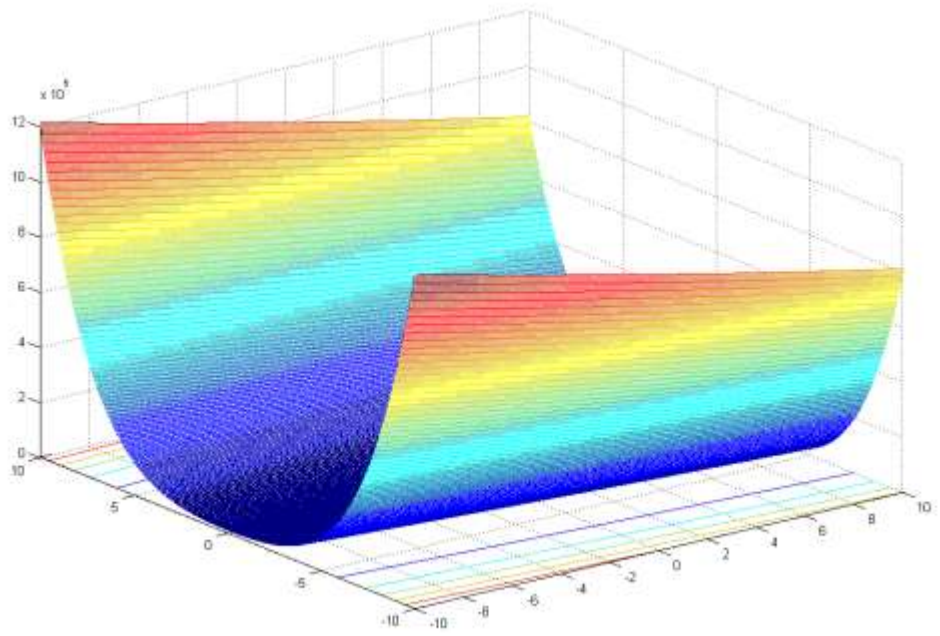

Figure C - Rosenbrock Function for n=2 (Hedar, 2007)

In (Dioşan & Oltean, 2006) the domain is [-30,30] and n=5.

An algorithm to generate data for n=5 is:

```
n = 5;
sum = 0;
for j = 1 to n-1
    sum = sum+100*(x(j)^2-x(j+1))^2+(x(j)-1)^2;
end
y = sum;
```

9.8 The Schwefel Test Function

The function can be expressed (Hedar, 2007) as:

$$f(\mathbf{x}) = 418.9829n - \sum_{i=1}^{n} \left(x_i \sin \sqrt{|x_i|} \right).$$

It can be written as:

```
f(x) = 418.9829*n + sum(-x(i)*sin(sqrt(abs(x(i)))))
```

The function can be shown for n=2 as:

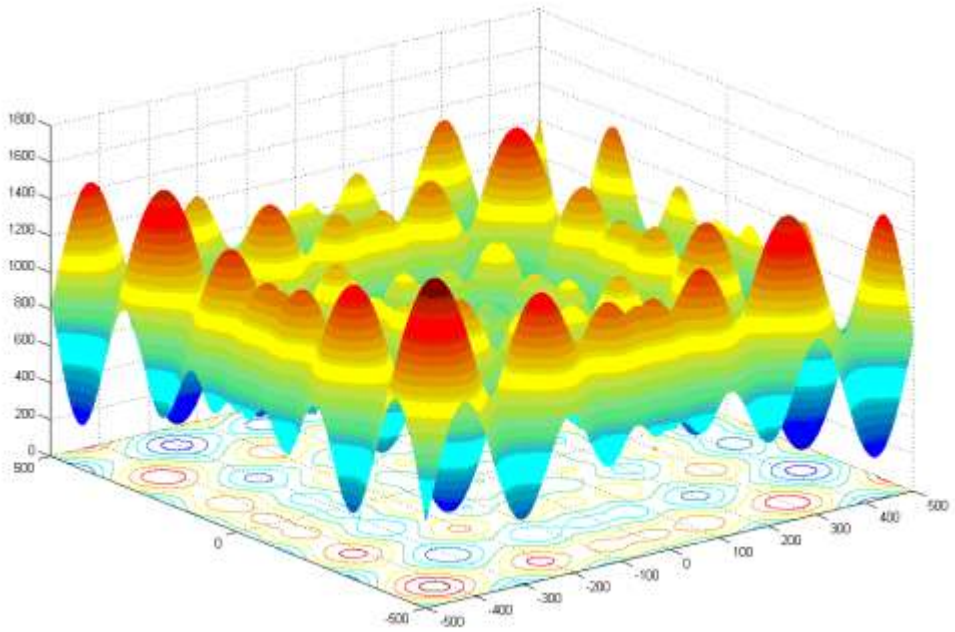

Figure F - Schwefel Function for n=2 (Hedar, 2007)

In (Dioşan & Oltean, 2006) the domain is [-500,500] and n=5.

An algorithm to generate data for n=5 is:

```
n = 5;
s = sum(-x.*sin(sqrt(abs(x))));
y = 418.9829*n+s;
```

9.9 Random Number Generator

The test software and test data generator programs use a pseudo random number generator that improves on that provided by the Microsoft C++ 2008 language. This is the Mersenne Twister MTRand package. The author gratefully acknowledges the work of Makoto Matsumoto and Takuji Nishimura.(http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/emt.html) (Matsumoto & Nishimura, 1998)

9.10 Software Command Line Parsing

The test software used a C++ command line parsing package CLArgs and gratefully acknowledges the work by Eric J Zijlstra. (Zijlstra, 2008)

The command line system enabled test to be run using a command line specified as follows:

```
Meta-MEP Processor v0.8 ®Copyright Mark N R Smith 2008, All Rights
Reserved
-t <datafile>: specifies the training data file
-g <VAL> : number of generations - an integer VAL
-s <VAL> : size of population - an integer VAL
-n <VAL> : number of MEP task nodes - an integer VAL
-r <VAL> : 1 = MEP, 2 = Meta-MEP
-i <Letter> : P=Pythagoras, A=Ackley, G=Griewank, O=Rosenbrock,
R=Rastrigin, S=Schwefel
-a <Letter> : G = Generational, S = Steady State
```