THE UNIVERSITY OF HULL

Multi-Objective Optimisation of Safety-Critical Hierarchical Systems

being a Thesis submitted for the Degree of

Doctor of Philosophy

in the University of Hull

by

David James Parker BSc. MSc.

February 2010

For Anna...

Abstract

Achieving high reliability, particularly in safety critical systems, is an important and often mandatory requirement. At the same time costs should be kept as low as possible. Finding an optimum balance between maximising a system's reliability and minimising its cost is a hard combinatorial problem. As the size and complexity of a system increases, so does the scale of the problem faced by the designers. To address these difficulties, meta-heuristics such as Genetic Algorithms and Tabu Search algorithms have been applied in the past for automatically determining the optimal allocation of redundancies in a system as a mechanism for optimising the reliability and cost characteristics of that system. In all cases, simple reliability block diagrams with restrictive assumptions, such as failure independence and limited 2-state failure modes, were used for evaluating the reliability of the candidate designs produced by the various algorithms.

This thesis argues that a departure from this restrictive evaluation model is possible by using a new model-based reliability evaluation technique called Hierachically Performed Hazard Origin and Propagation Studies (HiP-HOPS). HiP-HOPS can overcome the limitations imposed by reliability block diagrams by providing automatic analysis of complex engineering models with multiple failure modes. The thesis demonstrates that, used as the fitness evaluating component of a multi-objective Genetic Algorithm, HiP-HOPS can be used to solve the problem of redundancy allocation effectively and with relative efficiency. Furthermore, the ability of HiP-HOPS to model and automatically analyse complex engineering models, with multiple failure modes, allows the Genetic Algorithm to potentially optimise systems using more flexible strategies, not just series-parallel. The results of this thesis show the feasibility of the approach and point to a number of directions for future work to consider.

Acknowledgments

There are three people who stand out from the crowd of supporters, who helped me complete this thesis. Each had a different role to play and without them, I have no doubt, there would be nothing to read here.

Following my final undergraduate year, Dr. Yiannis Papadopoulos asked me to do some software development, implementing some Fault Tree Analysis algorithms. This was my first foray into the field of Safety Engineering and it led to a Masters degree, under his supervision. An obvious glutton for punishment, he wasn't done with me yet, and so followed this PhD research. I will always be grateful for the opportunities that he gave me and for his patience and guidance along the way.

My colleague and friend, Dr. Martin Walker was a constant companion on this journey. The reader should be unknowingly grateful for his proof reading skills, as whatever they think of the text now, it was worse before he cast his eye over it. Our struggle together honing the HiP-HOPS tool is the most fun I have ever had working. Finally, he is one of a select few (two) who truly understands the greatness of Triangles.

My beautiful wife Anna has been with me since before the journey began. Despite knowing what she was letting herself in for, she still married me and gave me a wonderful little boy, Aden Alexander, who, at the rate I was going, was going to finish his thesis before I did.

My eternal thanks and affection to each of them; as well as to all the others who perhaps deserve a mention, but aren't going to clutter up my page getting one.

Contents

Figures

Tables

1. Introduction

1.1 Motivation

1.1.1 Reliability is important.

People are dependent on the products of technology. This dependence is only possible when these products are reliable.

The IEEE Reliability Society describes reliability as the ability of a product to perform its function for the stated duration in a given environment (IEEE Reliability Society - Reliability Engineering). In this thesis, reliability will refer to the quantifiable probability that a system will not fail within a given time period.

Sometimes the failure of a product can cause serious injury or loss of life and cannot be designed to fail safely. These are known as safety-critical systems and include the products of the automotive industry, the aeronautic and aerospace industry, the shipping industry, the rail industry, and the nuclear power industry. It is especially important that these systems are designed to be highly reliable.

Reliability is a valuable property in all systems, not just where safety is critical. No company would like to be associated with the manufacture of unreliable products, but the benefits of reliability extend beyond brand reputation. A company releasing a reliable product could expect fewer returns from customers, so fewer costly repairs or replacements required.

From a user's point of view, reliable systems can result in lower operational costs; failing with expensive down-time and requiring costly repair less frequently. Likewise, a high reliability can allow longer periods between preventative maintenance.

There is clearly a broad reaching need for creating safe, reliable systems. However, this need is in conflict with decreasing product lifecycles and increasing complexity of design.

1.1.2 How do we know if a system is reliable?

The reliability of a system can be determined during its deployed lifetime; how often and with what frequency does it fail? However we want to be able to improve the reliability of systems before they are built and deployed, during the design phase of the system. In the rest of this thesis, when the "reliability of a system design" is used, it refers to the modelled reliability of the system that will be based on this design.

In order to improve the reliability of a system design, it is necessary to analyse the system design to gain an understanding of its failure behaviour.

One technique that has been in use for nearly half a century is fault tree analysis (FTA). Now widespread in industry, FTA is used to help understand the relationship between failures of a system and the possible causes of those failures; it encapsulates the complex interaction between the failures of individual parts of a system and how they can combine to cause the system as a whole to fail.

Where FTA asks the question, 'What caused this system failure?', Failure Modes and Effects Analysis (FMEA) inverts the question and asks, 'If this component fails, what is the effect on the system?'.

Reliability block diagrams (RBD) can also be created. They are based on the engineering model of the design but conform to a series-parallel representation of it. The simplification of the model as an RBD allows a reliability approximation to be calculated quickly.

An increase in available computing power allows computer engineering models to be used to simulate the function of the system they are modelling. Depending on the accuracy of the simulation model, this can provide information about how the system behaves under certain conditions.

1.1.3 Making use of reliability analysis.

These different techniques are useful and important for evaluating the reliability of the systems that they analyse. They are often used, sometimes as the mandatory requirement of an industry standard, to validate and certify the safety credentials of a design.

This use certainly has its place, but in recent years there has been an impetus to make use of these system analyses to inform the design decisions and drive the design process. At each stage of the design cycle, the system can be analysed, providing the engineers with information highlighting the reliability weaknesses in the system. Armed with this knowledge, the design team can make modifications to the system in order to increase the reliability of the design and eliminate the most hazardous failure modes. Making these changes as early in the design cycle as possible minimises the cost; late stage changes are expensive.

However, the application of these techniques is challenged by complexity. In a system that has a thousand components, for example, assuming that each component has a single failure mode, there are approximately 500,000 combinations of two failures that might occur and their effects need to be considered in a thorough safety or reliability analysis. To address problems of analysis arising in complex systems, new techniques for performing the analyses had to be developed in order to allow computer automation.

FMEA for example was, and to a certain extent still is, a primarily manual process conducted by teams of engineers over multiple weeks. The cost of

doing this, in time and money, makes its application in a cyclical process impossible. Tools and methods such as AutoSteve (Price and Taylor, 2002) and HiP-HOPS automate the process, producing FMEA data and tables directly from the system models making the analysis a matter of hours, minutes, and even seconds and less.

1.1.4 Improving the reliability of a system through design modifications.

The availability of these automatic tools provides rich data to the engineers, quickly enough to be useful in informing their design modifications.

The decisions regarding the modifications still lie with the engineer, however, and rely on their experience and system knowledge to improve the reliability of the functional design.

In their book on optimal reliability design (KUO, W et al., 2001) the authors layout a number of principles for improving the reliability of a system. Some of them, such as keeping the design as simple as possible whilst fulfilling the requirements, are down to good design practices. It is also possible to improve reliability simply by using more reliable components or by using redundant configurations of components as a backup.

Also listed as principles for increasing reliability is the use of maintenance, both proactive and reactive. This in itself is a complex optimisation problem and is beyond the scope of this thesis.

1.1.5 Problems with achieving optimality.

The goal of improving the reliability of a system rarely occurs in isolation and the use of more reliable components or the use of extra components in parallel configurations carries an extra cost; not necessarily just a financial

cost but perhaps a feasibility cost in terms of the weight gain or volume constraints that are affected by the use of different or extra components.

The conflicting goals of minimising the cost of a system and maximising its reliability turn the problem of selecting the best location for a redundant strategy, or the use of an expensive extra reliable component, into an optimisation problem.

As stated previously, automated system analysis tools greatly enhance the information that the engineers have to inform their design modification decisions. However, on anything other than the simplest of systems, choosing the optimum location of expensive redundancy configurations and premium components is a non trivial problem, because complex systems can be implemented in many different ways and have many different real-time configurations. The many permutations give a design search space that is highly susceptible to combinatorial explosion.

Combinatorial explosion occurs when the number of possible solutions rapidly increases due to the effect of the combinations and permutations of the available choices. A system with only ten components, for example, that has 4 alternatives for each component has over a million different combinations of component choices. That doesn't even consider combinations of redundant strategies for the system.

An informed and experienced engineer can make educated choices to attempt optimality but the sheer number of options makes it very unlikely that it will be achieved.

1.1.6 Automatic optimisation algorithms can help.

Automated optimisation algorithms are designed for problems of this nature, where the exhaustive search of all the options falls into a prohibitive time frame.

There are many different types of optimisation algorithm. Exact mathematical methods such as dynamic programming and integer programming can guarantee that, where solutions are found, they are optimal, but can typically only achieve this on highly constrained problems.

Meta-heuristics, on the other hand, cannot guarantee the optimality of solutions but can be applied much more flexibly to problems, with fewer constraints or artificial restrictions.

One such algorithm is Tabu search which navigates through the search space one step at a time by maintaining and modifying a list of taboo moves. Alternatively, the pheromone trails of a foraging ant colony inspire the working of Ant Colony Optimisation. Also inspired by natural processes, genetic algorithms are modelled on the 'survival of the fittest' mechanisms of biological evolution and are known to be effective when applied to combinatorial problems such as this.

1.1.7 Problems with existing optimisation algorithms.

The optimisation methods listed above have been applied to the problem, of optimal location of redundant configurations and alternative components, in the literature with varying degrees of success. However, a limitation that they all typically share is that the failure model used to calculate the reliability of the candidate solutions is a reliability block diagram (RBD).

This is not ideal, as RBDs are an abstract simplification of a system under failure, and cannot fully capture the complex behaviour of a system. The

component failures in the model, as well as the failure of the whole system, are single mode, only allowing complete loss of function to be modelled. In addition to this, RBDs are not the same as the engineering model of the system, though they are based on it. This means that RBDs must be generated from and converted back to the 'real' model to be useful; an extra and potentially error prone step.

The massively iterative nature of optimisation algorithms previously relied on the main strength of RBDs, the speed of reliability calculation. With the recent advances in automatic safety and reliability analyses, however, it becomes possible to manage without these limitations and allow the more desirable situation where the architecture of the engineering model itself can be directly optimised.

This thesis argues that these limitations can be addressed by using an automatic safety and reliability analysis method in conjunction with a multi-objective genetic algorithm.

1.2 Research Hypothesis

It is conceptually possible and technically feasible to achieve architectural optimisation using a combination of emerging model-based safety analysis techniques and meta-heuristics, assisting in the exploration of large design spaces for optimal tradeoffs between cost and reliability.

1.3 Research Objectives

The overarching goal of this research thesis is to demonstrate the technical feasibility and usefulness of optimising the dependability characteristics of a safety-critical system model by modifying its architecture using a genetic algorithm.

Both existing fields of research, complex reliability analysis and multi-objective optimisation, are well established with strongly developed solutions, whereas the combination of the two fields has not been achieved before. An attempt to develop completely new analysis methods and completely new optimisation algorithms would unnecessarily spread research effort too thinly. Instead good candidates will be selected from each field and will be modified and integrated to provide new and essential functionality that was hitherto not possible.

Achieving this outcome relies on meeting several objectives:

1. **Choose from among the state of the art in safety and reliability analyses a technique that can provide fast, scalable, and automatic model-based evaluation of safety and reliability.**

At a conceptual level optimising the reliability characteristics of a system model is a simple matter; all that is needed is any optimisation algorithm using any reliability-analysis technique to provide the solution evaluations. Ensuring the technical feasibility of such a combination, however, is not as straight forward as the time available for the design of a system is strictly limited and therefore any solution must be fast enough to be useful in a practical design process.

Sophisticated optimisation algorithms typically require a large number of iterative cycles to satisfactorily explore the solution space. In this context each considered solution must be evaluated by the reliability-analysis method to allow the optimisation algorithm to make comparisons and selections between solutions.

The consequence of this requirement is that the reliability-analysis method will need to be repeated, perhaps millions of times. Realistically, even reliability-analysis methods that take just a few seconds to perform will be

unsuitable as the repetition results in prohibitively long total optimisation times.

For the purposes of this work, this means that after a suitable reliability-analysis technique has been selected it will still be desirable, if not necessary, to reduce the computational effort required by this technique. The second objective of this work is therefore to:

2. **Examine the possibility of further minimising the computational effort required to perform reliability analysis through performance enhancements to the chosen technique.**

Satisfaction of this objective will maximise the scalability of the optimisation process to allow its application to larger and more complex system models. Beyond the area of reliability analysis, this thesis also engages with the state-of-the-art in the area of optimisation. The third objective is precisely to:

3. **Choose from among the state of the art a strongly performing genetic algorithm that can conduct the optimisation.**

The meta-heuristic optimisation techniques, genetic algorithms, are known to perform well in combinatorial problems such as this and will be used as the optimisation method. Research into genetic algorithms is active and continuing and there are many variations. Not all are equal, however, and so finding one that performs well on this type of problem will be important. Having selected a suitable GA it will still be desirable to examine the possibility of improvements to the performance of the chosen algorithm. The fourth objective of this work is therefore to:

4. **Examine the possibility of minimising the computational effort required for the chosen genetic algorithm to perform the optimisation.**

Though the bulk of the total computational expense of the optimisation will likely be the reliability-analysis technique, the overhead of the genetic algorithm itself will not be zero; this is again exacerbated by the iterative nature of the algorithm. As with the reliability-analysis technique, any performance improvements that can be applied to the genetic algorithm are desirable.

Performance and efficiency are important but should only be means for achieving good design solutions. The fifth objective of this work, therefore, is to:

5. **Maximise the quality of the alternative solution designs found by the genetic algorithm.**

Due to the stochastic nature of genetic algorithms it is not possible to guarantee that optimum solutions will be found. Selecting a good genetic algorithm for the task is the first step in ensuring that the solutions found are as near optimal as possible.

After that it is necessary to select and tune the parameters of the genetic algorithm to ensure that they are as well suited to the problem as possible.

Sometimes, experience with a genetic algorithm or the particular problem can facilitate choosing good values for the parameters. However, this process cannot always rely on intuition, and choosing the wrong values can have adverse consequences on the quality of solutions produced.

Therefore, the genetic algorithms should be altered, where possible, to minimise the requirement of the user to select operating parameters.

6. **Apply the proposed optimisation approach to case studies in order to validate feasibility and evaluate the scalability and usefulness of the approach.**

Several existing approaches in the literature for optimising redundancy allocation and component selection make use of a small benchmark system to evaluate the techniques.

This same benchmark system will be used for evaluating the techniques proposed in this thesis as this will allow direct comparison with existing techniques. In addition, a case study, on an example fuel system, will be used to show the potential practical benefits of this approach compared to a traditional unaided engineering design optimisation that was performed by an expert engineer.

1.4 Thesis Structure

A brief outline of the contents of the remaining chapters of this thesis follows below:

Chapter 2 gives a review of the relevant literature providing a background to the research. It is divided in to two sections. The first looks at different reliability-analysis techniques considering their advantages and disadvantages. Particular attention is given to the feasibility of their use in conjunction with an optimisation algorithm. The second section discusses different approaches to optimisation along with their respective merits. This includes analytical methods such as integer programming and meta-heuristics such as Tabu search and evolutionary algorithms.

Chapter 3 explains that the general properties of the HiP-HOPS reliability-analysis technique make it suitable for combining with an optimisation algorithm, satisfying objective 1. The highly iterative nature of optimisation algorithms requires that a restrictive computational expense limit must be placed on each iteration to allow the whole process to complete in a reasonable time. Objective 2, to reduce the computational cost of the reliability-

analysis, is fulfilled by extensions and improvements that were designed and applied to HiP-HOPS. These are also described in this chapter.

Chapter 4 describes methods for performing architectural optimisation of system models using an automated reliability-analysis technique integrated with a genetic algorithm. The focus is on a penalty-based genetic algorithm and two Pareto-based multi-objective genetic algorithms, chosen from the literature to be evaluated for objective 3. The chapter also details modifications to the basic genetic algorithms designed to improve their performance, both in terms of computational effort (objective 4) and solution quality (objective 5).

Chapter 5 discusses and evaluates the different algorithms through numerous experiments devised by altering the parameters of the algorithms and testing against a well known benchmark problem (objective 6). The reason for varying the parameters in this way is to maximise solution quality as per objective 5.

Chapter 6 applies the combined approach to a ship's fuel oil system case study. It is compared to a previous approach that evaluated several manually configured model variants before performing a cost benefit analysis.

Chapter 7 concludes the thesis, drawing together the contributions made, why they are relevant, and whether they satisfy the objectives laid out in this introduction.

Finally chapter 8 discusses some further work that remains open at the conclusion of this thesis.

1.5 Summary of contributions

The technique developed in this thesis is the first time that complex reliability analysis algorithm has been successfully combined with an optimisation

algorithm. HiP-HOPS, the selected analysis tool, was refined to significantly improve its performance making it more suitable for this role. The multi-objective optimisation algorithm that was selected to complete the combination, NSGA-II, was also refined and enhanced to improve efficiency and the quality of the results. The improved optimisation variant is the first meta-heuristic to find all of the optimum solutions in the Redundancy Allocation Problem; this had only previously been achieved with exact mathematical techniques.

1.6 Publications

Some of the work presented in this thesis has been published in part at several conferences. Parker *et al..* (2006) describes an algorithm for automatically generating an FMEA from FTA results. Parker and Papadopoulos (2007a, 2007b, 2007c) contains results from some early experiments in Chapter 4 and 5 using the PESA-II search algorithm. Zeng *et al..* (2007, 2008) show HiP-HOPS being used to provide reliability analysis to an Asynchronous Heterogeneous Hierarchical Genetic Algorithm.

2. Background

This chapter contains a review of two areas of research: Model-based reliability-analysis, and reliability optimisation. It is divided mainly along these subjects into two sections.

Also discussed in this chapter is how both disciplines can benefit from the combination of their states of the art.

2.1 Safety analysis

Traditionally, the way in which safety analysis was used centred on the assumption that it is component failure that causes accidents. Certainly this assumption was not unfounded as this used to be true for the majority of cases.

With the advance of technology, and the increasing complexity of system designs, there has been shift towards system accidents that are not caused explicitly by component failure but from the interaction between components.

The increasing complexity of systems design also makes it difficult to determine the effects of component failures; how they propagate through and affect the system's function.

The increasing size and complexity of the design also adds a new source of error. Design of a system is typically split between numerous different groups. Communication between the groups may be less than perfect and different teams may have different methodologies and practices.

Traditionally, safety analysis was a wholly manual process. It was based on an informal knowledge of the system being analysed.

Two methodologies that gained widespread use were Fault Tree Analysis (FTA) (VESELY, W E et al., 2002) and Failure Modes and Effects Analysis (FMEA) (MILITARY, U S, 1949).

The two approaches consider failure in a system from opposite directions. FTA is concerned with the fact that a system failure has occurred, what was the cause? FMEA deals with the question from the other end, namely, a component failure has occurred, how will this affect the system?

These techniques gave valuable insight into the behaviour of a faulty system, and forced designers to consider safety. However, the traditional method for achieving this analysis is a very time consuming, manually performed, process that is carried out by teams of engineers discussing the expected behaviour of the system, based on their experience.

There are several problems with this. The first is that postulating expected failure effects requires experience of previous faults and their effects. This is less of a problem in stable designs that undergo minor revisions between releases. In fast changing fields, or areas of design that radically change between versions, previous experience is not present or its relevance is diminished.

The second is that humans are prone to error. This problem is exacerbated hugely when the systems being considered increase in scale and complexity. Accurately inferring the failure behaviour of such a system becomes difficult and the chances of details being omitted or inconsistently recorded increase.

Thirdly, the time required to perform this analysis is measured in weeks. This limits its usability to validation of safety requirements at the end of the design process. That is an important use, but shortening the time that is required for the analysis expands its potential for driving change in the design of a safety critical system.

A safety-focused design process, in which safety analysis results are fed into each design iteration, enables problems to be identified and solved earlier in the process, thereby saving time and money.

Finally, the results of the safety analysis are completely separate from the model. Changes in the system will generally invalidate the analysis results. This is clearly not desirable when repeating the analysis requires a costly 2 weeks.

Several tools made an intermediate step of enlisting computer aid for the clerical side of safety analysis, enforcing consistency and improving inter-team collaboration. However, the recent increase and accessibility of computing power has lead to a field of research that seeks to automate safety analysis to overcome the above problems and broaden the usefulness of safety analysis.

The following section considers a selection of approaches from this field.

2.1.1 Failure Logic Modelling

Determining the complete system fault tree for a complex system is a difficult task. Small changes to system components will typically invalidate the whole tree requiring repetition of the process.

The techniques that use failure logic modelling seek to address both these issues. Focussing on the failure behaviour of simpler components is a much less complicated task than considering the entire system and can be performed more quickly with less likelihood of error.

Furthermore, the failure behaviour of the system is composed of the local behaviours of the internal components, and the behaviour of components can only be influenced through their connection interface. This means that

the individual components can be altered without needing to reconsider the failure behaviour of other parts of the system.

This section considers several approaches that are based around this concept.

2.1.1.1 Failure Propagation and Transformation Notation

Failure Propagation and Transformation Notation (FTPN) is a graphical description of the failure behaviour of a system introduced by Fenelon and McDermid (1993) and applied within an assessment procedure (FENELON, P et al., 1994).

It was designed to provide a bridge between the deductive FTA and the inductive FMEA processes; FPTN is an abstraction of both techniques that can be traversed from cause to effect, as with FMEA, or from effect to cause, as with FTA.

FPTN was conceived with a focus on software intensive applications, although it is not specific to this domain.

The failure model can be defined using FPTN by abstracting the architecture of the system into modules that have inputs and outputs. Modules can be connected to other modules via the inputs and outputs. Collections of modules can be hierarchically encapsulated in a subsystem module that can then be used as a component in a higher level module.

Each module contains a set of failure modes. The failure modes are typed into broad categories, such as timing or value failures. They are further classified as either internal failures, a result of a failure of the module itself, or external failures, caused by other modules.

Finally, logical equations describe the smallest combination of failure modes required to cause a failure mode at the outputs of the module. The logical equations can propagate failure modes, unchanged, from the module inputs to the module outputs. Failure modes can also be transformed, from one failure type to another, in between the input and output of the module.

Additionally, internal failure modes can cause external failure modes at the outputs, and external failure modes at the inputs can be handled by the module thereby halting their propagation.

Wallace (2005) identifies a key deficiency with FPTN. This is that it can be easy for the failure model, given by the FPTN, to become desynchronised with the system model.

The defined failure model sits separately above the system model. Only known failure propagations are modelled in the connections between modules and so changes to a component in the model can require non-local changes to be applied to the FPTN diagram. Individual expressions in a module can be sensitive to even minor changes in other components. Determining these changes in the flow of failure would require a complete reanalysis of the entire system.

FPTN is limited as a manual method that formalizes and records the designer's knowledge. Thus it represents a step forward from ad hoc analyses of traditional FMEA and FTA but is a process that is not suitable for automation.

2.1.1.2 Failure Propagation and Transformation Calculus

Failure Propagation and Transformation Calculus (FPTC) was proposed by Wallace (2005) in an attempt to overcome the limitations he identified with FPTN.

The main way in which this is achieved is to tie the failure behaviour of the system more closely to the architectural model. The result of this is that all potentially important dependencies are identified and recorded, and not just the ones known to transmit failure.

FPTC is similar to FPTN in that the failure modes are classified e.g. value failures, omission, commission, etc. Where FPTN had a floating failure model, however, FPTC failure behaviours are annotated directly in the components of the system model.

Each component will have inputs and outputs, and a set of expressions in the component declare the propagation and transformation of failure as it passes between them. It includes the use of wild cards at both the input and output side of the expression, to indicate for example that all failure modes are transmitted to a particular output, without the need for repetition in definition.

If both sides of the expression carry the same failure mode then it represents a normal propagation, whereas different failure modes indicate that the incoming failure mode has been transformed by the component.

The same mechanism is used to annotate the connections between components as these communications protocols can also affect the error flow.

Failure modes can also be mitigated or initiated by a component and is indicated by the normal behaviour (*) token on the output and input side of the expression respectively.

Once the components are all annotated, the system acts as a token-passing network. Each expression can then be 'run' with respect to the normal behaviour token. The generated token sets are passed along the connections where they are propagated and transformed by components and connections in the system. The process continues until no new tokens are created. It has

been demonstrated that the finite number of failure modes results in a non-infinite cycle.

Useful analysis of failure consequence can be done by repeating the process with failures injected into the system and recording of the failure mode tokens appearing at key parts of the system.

Ge *et al.* (2009) extend the basic syntax of FPTC to permit probabilistic analysis of the system failures. This is done by adding a probability value to each expression. Model checking can ensure that the probability values are valid, totalling one for each input failure mode.

When the failure model is then 'run', in addition to the tokens being passed around the model the calculated probabilities are also transmitted.

Offering an advantage over FPTN, FPTC can be automated, and because the failure model is coupled with the architectural model, changes made to components can be localised and not require a complete reanalysis to up-date the annotations.

However, FPTC has a limitation of its own. In order to analyse the effect of a failure, the failure must be injected into the system. For each different failure or combination of failures the cycle must be repeated.

2.1.1.3 State Event Fault Trees

State Event Fault Trees (SEFT) were proposed to overcome the inability of standard FTs to model temporal event ordering (KAISER, B and Gramlich, C, 2004). They can be distinguished from normal fault trees in the way that they separate states, that last a period of time, from events, that are instan-taneous and typically trigger state changes (GRUNSKE, L et al., 2005).

SEFTs can express both software behaviour, which is usually the domain of State charts, and hardware failure states, which are typically modelled using Markov Chains (KAISER, B et al., 2007).

As with FPTC the failure behaviour is modelled at the component level; the system behaviour exists as an emergent property of interaction of the components. This improves the reusability of the components and the ease of determining the behaviour in the first place.

To enrich the modelling of temporal events they include several concepts, such as the Priority-AND that remembers if events have occurred in a specific order. Additionally the History-AND gate remembers previous events that have occurred. They can also make use of the NOT gate concept.

The use of the states and transitions in the SEFT makes them unable to make use of standard FTA algorithms for analysis. Instead the model is converted into a Deterministic and Stochastic Petri Net (DSPN) (CIARDO, G and Lindermann, C, 1993).

Once a DSPN exists, it can be quantifiably analysed automatically by a suitable tool, such as TimeNET (ZIMMERMANN, A et al., 1999).

One potential flaw that affects the scalability of the technique is that, as with all state-based analysis, on larger models there can be a state-space explosion.

2.1.2 HiP-HOPS

Papadopoulos and McDermid (1999) proposed a methodology, Hierarchically Performed Hazard Origin and Propagation Studies (HiP-HOPS), that can automatically synthesise system fault trees from an annotated architectural model.

The annotations are similar to those used by FPTN. Like FPTN, the expressions are logical equations that define how classed failure modes at the outputs of a component are transformed or propagated from the inputs of the component. Internal failure modes can also cause output failures.

To differentiate between external failure modes and internal failure modes, failures that occur at the inputs and outputs of the component are defined as deviations of normal behaviour: input deviations and output deviations respectively.

The failure expressions are directly applied to the components and not to a separate floating failure model. This coupling ensures that changes to the local behaviour do not require a redefinition system wide.

Once the model is completely annotated, the fault tree synthesis algorithm traverses the model, starting at the outputs of the system. Following the model connections between components, the algorithm encounters output deviations at the boundaries of components.

The output deviations each have a logical expression that describes a fault tree with leaf nodes that are either terminating internal failures or input deviations that connect externally.

The automatic traversal connects outputs to inputs and constructs a system wide fault tree or set of fault trees. These system fault trees can be analysed using standard FTA algorithms to produce minimal cut sets describing the smallest combinations of failure required to cause a system level failure.

An advantage over other methods is that component failures are explicitly defined in the components. The deductive traversal technique does not require faults to be injected as they are already defined in the model and are identified, including combinations of failures, during the traversal.

HiP-HOPS' capabilities are further extended by Parker *et al.* (2006) to automatically generate FMEA tables. As noted by the authors of FPTN there is a relationship between the deductive FTA and the inductive FMEA. This property is exploited by HiP-HOPS to further process the minimal cut sets from the FTA.

A fault tree describes how a system level failure is caused by (a combination of) component level failures. By going through the cut sets one by one and cataloguing the component level failures along with the system level effect of the cut set, the FMEA relationship can be found.

It offers a number of advantages over traditional FMEA. Firstly it is derived from the fault trees of the system. It does not require an inductive reasoning of the effect of each failure mode. Also, a benefit of harvesting the data from the fault trees is that the minimal cut sets define combinations of failures, not just the single failures of traditional FMEA. Therefore the effects of combinations of failures can be determined without the combinatorial explosion that failure injection techniques suffer from.

Further advances in HiP-HOPS stem from its FTA abilities. Sharvia and Papadopoulos (2008) describe a method for automatically analysing non-coherent fault trees, i.e. fault trees that contain NOT gates that negate branches of the tree.

Walker and Papadopoulos (2006) further extend the capabilities of HiP-HOPS by allowing dynamic fault trees to be defined and analysed. Dynamic fault trees make it possible to model the effect of the timing of failures, and the order in which they occur. This is further expanded and detailed by Walker *et al.* (2007). The advantage of this technique over SEFTs is that they define methods to allow traditional FTA algorithms to be used, rather than requiring the conversion to DSPNs.

These extensions to HiP-HOPS allow complex failure behaviour to be modelled that is simply not possible with normal FTA.

Reusable failure patterns are the focus of the extensions proposed by Wolforth *et al.* (2008). The purpose of this extension is to describe generic failure expressions that can be held with reusable components in a library. The patterns would be instantiated when used in a particular model; automatically generating the standard HiP-HOPS failure expressions with the appropriate number of input and output deviations.

HiP-HOPS has tool support through a graphical user interface for annotating and analysing Simulink models. Alternatively its functionality is also available through a safety analysis plugin to ITI's modelling and simulation tool SimulationX.

In summary, HiP-HOPS provides rich and detailed failure behaviour modelling that follows the architectural model closely. The fault tree based model provides quick and automatic safety analysis.

2.1.2.1 Component Fault Trees

Component fault trees, described by Kaiser *et al.* (2003) and Grunske and Kaiser (2005), offer the same basic functionality of HiP-HOPS, but currently lack the extensions offered by dynamic and non-coherent fault tree models.

The term component fault tree (CFT) does however neatly describe the function of the failure expressions in HiP-HOPS and in the rest of the thesis the use of the term CFT refers to the fault tree described by the component failure expression in a HiP-HOPS context.

2.1.3 Behavioural Fault Simulation

A different approach to safety analysis is to use formal modelling engines that enable simulation of the design when functioning normally. Where this is possible, the behaviour of this system can be compared with a variation that has been degraded through the injection of faults.

A benefit of these techniques is that often the failures in the system do not require extra annotation as they can be automatically extracted from the model. This does come at a cost, however, as this is only possible where the required domain has been modelled.

2.1.3.1 AltaRica

AltaRica is a formal modelling language that can be used to describe complex hierarchical models (GRIFFAULT, A et al., 1999). The failure model can include both states and events allowing temporal properties to be modelled (BIEBER, P et al., 2002).

On the basis of this formal definition the AltaRica model can be used to generate fault trees for non-dynamic failures or Petri nets for temporal analysis as with SEFTs. In both cases, external tools can then be used for analysis.

(BIEBER, P et al., 2004) identified some drawbacks to using AltaRica for modelling real-world systems. In a particular example, modelling the loss of pressure in a hydraulic system, both fluid and pressure data needs to be transmitted in opposite directions in the model. The use of these bidirectional signals to propagate failure can easily lead to loops in the model which would then be rejected by the modelling language.

Workarounds to prevent loops include the introduction of delays, but these can lead to incorrect modelling of instantaneous propagations. The delays

also introduce temporal features to non temporal faults making it impossible to generate fault trees from the model.

2.1.3.2 Formal safety analysis platform with NuSMV-SA

Formal safety analysis platform (FSAP) provides a graphical user interface for the NuSMV-SA model checking and safety analysis engine (BOZZANO, M and Villafiorita, A, 2006). FSAP/NuSMV-SA was created to provide a single environment for model design and safety analysis.

The underlying engine is capable of simulating the model, in both nominal functional mode and degraded – fault injected – mode. Standard model checking capabilities, such as property verification and counter example generation, are also provided by the NuSMV engine.

Additionally, fault trees can be generated from the model where failure modes have been added to the model. The fault trees can include NOT gates for specifying conditions where events are required not to have happened for failure to occur.

Tighter integration of the safety analysis and design modelling allow safety analysis to be performed earlier in the design cycle, enabling less expensive design changes.

As is common with model checking systems, FSAP/NuSMV-SA is susceptible to state space explosion. Using simplified models can alleviate the problem in some cases.

2.1.3.3 Model based deviation analysis

Model based deviation analysis (MDA) (HEIMDAHL, M P et al., 2002) differs from the other approaches here in that it is not concerned with faults in the

system. The assumption is that the system is not faulty; instead, MDA seeks to determine the system effects of deviations from the expected input data.

MDA is based on, and extends, the Software Deviation Analysis (SDA) developed by Reese and Levenson (1997). SDA is an exploratory, qualitative, technique that sought to answer questions such as, "What is the effect on the output if the input reading is high?".

MDA quantifies the question to allow a model checker to verify proofs of the effects of the deviations. A quantified version question would be, "What is the effect on the output if the input reading deviates in a range of 0 to 100 metres?". This can then be formulated as a temporal logic property readable by a model checker which could then show whether a given deviation is acceptable, or if not, provide an example demonstrating how the deviation could become too great.

The basic MDA approach is to create two models, one with no deviations, and one with deviations. Both models operate on the same data set with the exception of the input deviations, which are added to the normal inputs.

The models are executed and the computed states are compared to derive any critical deviations.

MDA, in common with many model checker based approaches, suffers from state-space explosion. This problem is exacerbated by the need to use two models running on separate data sets. To minimise the effect of the extra model, the creators of MDA proposed a method where both the normal model and the deviated model are embedded in the same environment, thereby using the same data set, sharing common variables, and reducing the computation required.

2.1.3.4 Deductive Cause Consequence Analysis

Deductive Cause Consequence Analysis (DCCA) (ORTMEIER, F et al., 2005) is a formal safety analysis technique that uses mathematical proofs to verify whether a component level failure is the cause of a system failure.

DCCA uses finite automata as system models and Computational Tree Logic (CTL) to formalise the model.

In order to perform the safety analysis, a set of failure modes is considered against a set of system hazards. The CTL property of criticality is automatically extracted from the finite automata model. This specifies whether the given set of failure modes can cause the system failure.

The goal of DCCA is to determine the minimal critical set, such that there exists no critical subset of failure modes. A model checker such as SMV can be used to validate the proofs.

DCCA has been modified to work with an industrial design tool SCADE (GÜDEMANN, M et al., 2007).

The advantage of this technique is that the calculation and validation of the critical sets is automated from the system model. However, the process suffers from combinatorial explosion as, to be complete, all combinations of failures must be considered.

A way to reduce the number of combinations checked is to produce a smaller set of failure modes from informal analysis techniques such as FTA. This smaller set can then be formalised using DCCA.

An extension to DCCA called Deductive Failure Order Analysis (DFOA) allows temporal fault trees to be automatically generated from the system model (GÜDERMANN, M et al., 2008). DCCA is used to generate the minimum

critical sets of failures and DFAO re-applies the ordering of the failures. The temporal properties can include simultaneous occurrence of failures as well as priority ordering.

2.1.3.5 Automatic FMEA

AutoSteve (PRICE, C J and Taylor, N S, 2002) uses failures injected into a qualitative electrical simulation to automatically generate an FMEA report.

Qualitative simulation offers an advantage over quantitative simulation in that it can be applied earlier in the design cycle when numerical data about the components may not be available. Conducting such analysis earlier reduces the costs involved in making corrective changes.

A further advantage is that qualitative simulation is less computationally expensive than quantitative simulation. This is especially important when using it to generate FMEA data via fault injections, because the simulation must be repeated with different faults activated.

The process for producing FMEA data begins with modelling the normally functioning electrical system. This is simulated to provide a baseline for the functioning of the system.

Non-functioning versions of the components are also modelled and inserted in to the model. The simulations are then repeated, each time varying the combinations of faulty components inserted in the model.

The FMEA data is then accumulated by comparing the behaviour of the faulty models with the baseline model behaviour.

To increase the readability of the FMEA information, unlikely or mutually exclusive combinations of failures are not considered. Additionally, the FMEA

results are pruned to remove non-minimal combinations of failures that are covered by a single point of failure.

They suggest that it is possible to convert the FMEA data into a FT format equivalent to the reverse of the HiP-HOPS FMEA generation. The ability to consider combinations of events greatly enhances the validity of the resultant FT.

There are several drawbacks to this method. The inductive method for generating the FMEA in this way has serious ramifications for scalability. The simulation must be repeated for every combination of failure that is considered.

Practical steps are taken to reduce the number of combinations considered, by introducing a probability threshold for failures which cannot be exceeded. The effectiveness of this mechanism is in question though as probabilities of failure would likely be lacking in the earlier stages of the design cycle.

Ricardo developed a tool that works using the same principles called AutoFMEA that works with the Matlab Simulink simulation tool (PLC., Ricardo, 2009).

2.1.4 Overview of safety analysis

Safety analysis techniques based on failure logic modelling and fault injection simulation each have advantages and disadvantages (LISAGOR, O and McDermid, J A, 2006).

The principle drawback with failure logic modelling techniques such as HiP-HOPS and CFTs is that the process of determining the failure behaviour of the components in the system models is not automatic. Although determining local failure behaviour is easier than trying to infer the failure behaviour of the entire system it can still be a time consuming process.

The corollary of this is that once the local failure behaviour has been added to the model, the generation of complete fault trees is deductive and fast. It does not suffer from combinatorial explosion.

Fault simulation techniques on the other hand make use of domain libraries of components where failure behaviour is a property of the model and can be automatically extracted.

It has its disadvantages though. Firstly, simulation is only possible where domain models exist. Secondly, the process of injecting failures into a functioning model and simulating to determine the effects of the failures is very prone to combinatorial explosion.

It seems that in most cases it would be impractical to consider combinations of more than 2 failures, and even then the large number of repeated runs could make it unusable in a further iterative process such as optimisation.

Lisagor and McDermid (2003) suggest that the two techniques could be combined; perhaps the formal model simulation techniques could automatically provide local failure behaviour directly from the formal model.

Although useful to safety analysis in general, such an extension would be beyond the scope of this thesis.

In order to be successfully applied to an optimisation problem it is necessary that the repeated analysis of an evaluation tool be quick. The failure logic modelling techniques provide quick, deductive, analysis and the slow part of the process is carried out just once, before any optimisation and is reusable as the model is evolved.

The slow portion of the simulation based techniques is the repetitive analysis and as such makes those less suitable for use with optimisation.

Most of the failure logic modelling techniques share similar foundations. However, HiP-HOPS seems to provide the richest modelling capabilities, allowing not only static FTA but also non-coherent and dynamic fault trees. It also includes algorithms for automatic generation of FMEA data.

Whilst initially the extra capabilities may not be required, it will be possible to expand the scope of the optimisation easily due to the richness of the model.

A further reason that adds to the attractiveness of HiP-HOPS for this project is that complete source access is available to the author giving unfettered scope to modify the algorithms. This will enable performance enhancements to be applied to further reduce the computational expense of calculating the evaluations required for the optimisation.

The ability to closely integrate the two parts, analysis and optimisation, is also preferable to having to treat the reliability-analysis as a 'black box'.

Although it is reasonable that other reliability-analysis techniques could be used for the purpose of providing reliability data to optimisation algorithms, for the reasons above HiP-HOPS will be used for this thesis. The methodology will be discussed in more detail, in the context of performance enhancing extensions, later in the thesis.

The next section takes a look at different optimisation algorithms, before choosing a couple of well represented algorithms to act as the optimisation engines in support of this thesis.

2.2 Optimising system reliability

The advancements in safety analysis techniques allow decision makers in the design of systems to be influenced by the rich information supplied by the techniques.

Knowing that one configuration of a functional system has higher reliability than another configuration enables the decision maker to choose it as a solution.

However, this presents a new problem. The decision maker now faces an almost infinite number of possible configuration alternatives that can improve the reliability of the design, as increasing the redundancy level increases the reliability.

Furthermore, it is highly unlikely that the reliability of the system is their only concern. The budget for the system is not infinite and the cost of applying reliability improving measures increases as reliability is increased.

The goals of increasing reliability and reducing cost are in direct conflict and it is the job of the decision maker to determine an optimum configuration with regard to both cost and reliability.

Typically, the decision maker would rely on his experience to make educated choices to attempt to achieve such optimality. However, with such a vast, complex search space of possible options available it is highly unlikely that this would be achievable in anything but the most trivial of cases.

It is for this reason that an automated optimisation algorithm, that can make use of the computer's power of repetition, is necessary.

2.2.1 Benchmark Redundancy Allocation Problem

In the literature on optimising system reliability through component selection and redundancy allocation, there is one test problem that has been extensively used.

The test system was defined (FYFFE, D E et al., 1968) as 14 functional units in a series configuration. Each of the units' functions can be fulfilled by

between three and four alternative components that can be arranged in a parallel configuration to provide redundancy. Each of the alternative components has varying cost, weight, and reliability values.

In its original form the system was to have its reliability maximised, subject to cost and weight constraints of 130 and 170 units respectively, and components could not be mixed within a functional unit.

Nakagawa and Miyazaki (1981) extended the test problem by varying the weight constraint between 159 and 191 units, with increments of 1 unit, to present 33 different cases.

A more general case, where component mixing within functional units is allowed, was proposed by Coit and Smith (1996b). This significantly increases the size of the search space calculated to be in excess of 7.6×10^{33} configurations.

The problem was demonstrated to be NP-hard (CHERN, M, 1992).

It is in this form, that the RAP problem is used most frequently and this use allows an easy comparison to be made between different published approaches.

What is particularly useful for retrospectively evaluating the quality of the solutions found by the different approaches is that in 2007 the optimal solutions were exactly calculated for all 33 cases (ONISHI, J et al., 2007). These were produced by an exact mathematical approach that guarantees the optimality of the solutions.

So instead of comparing the solutions of one approach with every other approach's solutions, each can be measured against the true optimum solution.

A useful metric in this regard is used by Coit and Smith (1996b), called Maximum Possible Improvement (MPI). Displayed as a percentage, it is "the fraction that the best feasible solution achieved of the maximum possible improvement, considering that reliability ≤ 1".

If it is used to compare with the optimal solutions then it takes the form:

Any solution that matches the optimal solution will have an MPI of 0%, the best value. Inferior solutions will have negative MPIs.

Finally, to summarise the quality of the solution set over the 33 test cases, they will be compared by the percentage of solutions that were optimal, and by the mean average of the MPIs for the test set.

2.2.1.1 Exact mathematical methods

The first attempts at solving the RAP are classed as exact mathematical methods because they work by solving a mathematical formulation of the problem. Where this formulation can be solved, the solution is guaranteed to be optimal.

There have been several approaches using exact methods including using dynamic programming (FYFFE, D E et al., 1968), integer programming using surrogate constraints (NAKAGAWA, Y and Miyazaki, S, 1981), (BULFIN, R L and Liu, C Y, 1985), and (ONISHI, J et al., 2007), and linear approximation programming (HSIEH, Y, 2002).

Initial attempts heavily constrained the problem, and component mixing was not permitted, though this limitation was later relaxed.

The principle strength of exact mathematical techniques is that the solutions found by the algorithms can be guaranteed to be optimal. Additionally the solutions are quickly calculated.

This disadvantage is that they rely on the ability of the designer to formulate the problem as a solvable complex mathematical formula. There are some cases, as with earlier attempts (FYFFE, D E et al., 1968) and (NAKAGAWA, Y and Miyazaki, S, 1981), where solutions could not be found to particular test cases because the mathematical representation of the problem did not have a solution.

There is a related problem with the generality of the approach; the formulation of one system optimisation problem is unlikely to be applicable to a different system. This limits its appeal as the basis of an optimisation framework for systems engineers.

Bulfin and Liu (1985) note that for solving larger problems the use of a heuristic, rather than an exact mathematical, method may be required.

2.2.1.2 Meta-heuristics

Penalty-based Genetic Algorithm

In 1996 Coit and Smith (1996b) declared that though the preceding exact mathematical solution methods successfully found optimum solutions they did so by artificially restricting the search space. The restriction of only allowing one component type per functional unit lacks realism, as component types could – and often would – be mixed in real world systems.

They identify genetic algorithms as a particularly effective method for solving combinatorial optimisation problems, as they are capable of navigating large and complex search spaces. This is important as even relatively small problems, such as the benchmark case, can have very large search spaces.

Genetic algorithms are modelled on biological evolution. A population of candidate solutions is maintained and are subject to genetic operators which produce new candidate solutions. When the addition of new solutions causes the population to grow beyond a certain threshold, the excess is culled.

The choice of solutions to cull follows a general process of survival of the fittest. There is a random element to the choice, but solutions which perform well in the objective are more likely to survive into the next generation.

The candidate solutions are encoded into a genome, mimicking the function of DNA. In this case, a string of integers is typically used as the encoding, each integer representing a different component choice. The encoding should contain all the information necessary to reconstruct the given solution.

New candidate solutions are created both by randomly mutating an existing encoding and by recombining the encodings from two 'parent' solutions. The latter operator is also known as crossover and creates a new solution that has characteristics from both parent solutions.

The crossover operator acts to promote convergence in the population, acting as a local search. Mutation on the other hand promotes diversity in the population through random perturbation of the solutions.

The penalty-based GA method deals with the multiple constraints of the test problem by incorporating them into a single objective function that was maximized using a single objective genetic algorithm.

This single objective was to maximize the modified reliability of the system. Scaled violations of the cost and weight constraints acted as a penalty which was then applied to the reliability value.

Thus where there are two solutions that achieved the same reliability but one of them violates a constraint, the infeasible solution's modified reliability would be lowered by the penalty. Consequently the feasible solution would be preferred by the algorithm.

Often an optimum solution is most efficiently found by allowing search to proceed through infeasible regions, so it is important that the penalties for constraint violation are not too severe. This is achieved here through the use of a dynamic penalty function that factors in not just the degree of constraint violation but also how long the GA has been running.

By factoring in the age of the algorithm like this it is able to have small penalties at the early stages and penalise more severely in the latter stages. This allows effectively unfettered access to the infeasible regions at the beginning and gradually guides the search towards feasible solutions by the end.

An earlier paper, (COIT, D W and Smith, A E, 1996a) had determined that the dynamic penalty had a superior performance compared to either a static version or allowing only feasible solutions. The dynamic penalty system also seems to work better with highly constrained cases.

When applied to the benchmark problem the penalty-based GA found feasible solutions for all 33 of the test problems. Of particular note is that, free of the restriction of single component choices, the GA was able to find solutions that exceeded the reliability of those previously deemed optimum at the time by exact methods.

The method found the optimal solution in 21.21% of the test cases, with an average MPI of -1.454%, an improvement on the previous approaches.

Tabu Search

In 2003 Kultural-Konak *et al.* (2003) applied a meta-heuristic method called "tabu search" to the RAP benchmark. Unlike the GA which performs a global search, a tabu search algorithm makes use of the neighbourhood structure inherent in the RAP.

It functions by first initializing to a random potential solution. This is evaluated. Next, all the neighbouring solutions are generated and evaluated. A neighbouring solution is any solution that can be generated with one single change from the current solution.

In the case of the RAP this could be one of three possibilities: changing one component for an alternative component, removing one component from a parallel subsystem, or adding one random component to a parallel subsystem.

Once all the options have been evaluated, the one with the highest fitness is chosen to be the next current solution and thus the algorithm deterministically navigates through the search space.

In order to prevent the algorithm from looping or getting stuck in local optima it maintains a list of forbidden or 'taboo/tabu' moves. This list is populated by recently tried solutions, forcing the algorithm to search in (recently) unexplored regions. The list is of finite size and older solution moves are removed.

This sequence is then repeated until a set maximum is reached for the number of attempts made without an improvement to the best feasible solution found.

The application to the RAP benchmark uses an adaptive penalty function that calculates a penalty for constraint violation. The size of the penalty for

infeasibility is varied automatically to either guide the search to a more feasible region or increase the diversity in the search by allowing more infeasible solutions.

Kultural-Konak *et al.* (2003) calculated that approximately 40% less effort on average was required for the tabu search than for the penalty-based GA for all test cases. Although on average more individuals were evaluated, because each move in the tabu search only altered one subsystem, only the reliability for that subsystem had to be recalculated. They note that this balance may not be favourable with all systems. This may also be highly dependent on the analysis method used and may not hold with full safety analysis based approaches.

The solutions found by the method were generally better than those found by the penalty based GA. It found the optimal solutions in 78.79% of the test cases and had an average MPI of -0.263%.

Ant colony

In nature, ant colonies use pheromone trails to establish and navigate efficient routes between their nests and food sites. Though the pheromone evaporates over time, good routes are reinforced as more ants pass over them, whereas bad routes dissipate completely.

Ant colony optimisation is inspired by this biological process and Liang and Smith (2004) proposed such a method for reliability optimisation of series-parallel systems.

It comprises multiple stages. In the first stage an ant colony is generated based on the pheromone trail of previous iterations. In the second stage, the ant solutions conduct a local search. Finally, the ants update their pheromone trails and the cycle is repeated for a predetermined number of iterations.

When applied to the test cases of the benchmark system, the ant colony algorithm performed similarly to the tabu search algorithm. It found 72.73% of the optimum solutions with an average MPI of -0.342%.

Liang and Smith (2004) note that the ant colony algorithm is more computationally expensive per run than penalty-based GA, but over 10 repeated runs, the solutions found by the ant colony method generally met or exceeded those of the GA. Therefore, the extra effort involved does yield better results, or alternatively the same results can be found with less effort.

Simulated Annealing

Simulated annealing is a search technique that is based on metallurgical annealing. When a metal is heated, the atoms are free to change their position randomly in high energy states. By controlling the rate of cooling the blacksmith can encourage the crystals in the metal to form in beneficial states.

With simulated annealing, the current solution is randomly changed to a solution that is local in the search space, i.e. a solution that is similar, but with minor changes. The acceptance of the new solution is dependent on the global temperature; when the temperature is high, highly random changes are allowed; as the temperature decreases, according to a cooling schedule, the allowable changes follow a greedy method.

A greedy method seeks to achieve a global optimum by making decisions that are locally optimum at each stage.

Kim *et al.* (2004) applied simulated annealing to one of the test cases of the benchmark problem where the weight constraint was 190 units. It was repeated 30 times and the best solution found had an MPI of -0.766%, but the average MPI was -12.99%.

As it was not tested on the remaining test cases it is not possible to know its performance for those. For the single test, whilst the best solution found was reasonably near optimal, the average over the 30 runs was relatively poor.

An advantage of this method, however, is that it is quick, completing a run in under a second.

Greedy method with natural selection

You and Chen (2005) proposed another meta-heuristic method. Based on the result of a random threshold check, it assigns components to their subsystems using either a greedy method or randomly.

The resulting solutions from this assignment process are revised by two methods: an intra-system change randomly selects two subsystems and exchanges their component choices if the result is an improvement; and an inter-system change randomly selects two subsystems in different solutions and exchanges them if the resultant solutions are an improvement.

The technique has several similarities with a standard genetic algorithm in that it is population based and its two solution change mechanisms, inner and inter-system change, are broadly similar to mutation and crossover, respectively.

It differs from genetic algorithms in several ways, however. The solutions are always feasible; no solutions that violate constraints are allowed. Solutions in the population will never be replaced by solutions with a lower fitness. Finally the resultant solutions from the two solution change mechanisms are never worse than the previous solutions.

When applied to the benchmark problem the method proved very effective. It found optimal solutions to 100% of the test problems. In their published results, the reliability of several of their solutions was less than those given

as optimal by Onishi *et al.* (2007), but this appears to be due to a rounding or typing error as the component configurations matched the optimum solutions.

These results are particularly impressive as their method requires many fewer iterations than previously described heuristics.

A drawback of this method is that it is purely a single-objective optimiser. It maximises reliability amongst solutions that do not violate cost and weight constraints.

More ants

Ant colony optimisation was again used in 2007 (ZHAO, J et al., 2007) seeking to improve upon the Liang and Smith (2004) method.

It differed from the previous attempt in two principal ways. Firstly, when the ant colony is generated in each iteration, there is a heuristic that allows for *a priori* system knowledge to be used. A state transition rule is used to balance the effect of the knowledge-based exploitation with standard exploration.

Secondly, rather than only storing a single best feasible individual for the offline pheromone update algorithm, a number of individuals are stored in rank order.

The method was applied to the benchmark RAP problem but with a maximum iteration set at 300 rather than the 1000 iterations of the previous approach.

An optimum solution was found in 42.42% of the test cases, with an average MPI of -0.691%; only a little over half of those found by the previous

ant colony approach. However, this is achieved with a third of the number of iterations.

If the ratio of iterations to optimums ratio holds, then it would seem to be a superior algorithm. However, it is not known whether the new approach would yield results of the same quality if allowed the same number of iterations.

Multi-objective tabu search

In 2006, Kultural-Konak (2006) approached the RAP problem again with tabu search.

Their original attempt combined the 3 objectives into a single objective. This time they considered the three objectives separately, and in each iteration of the algorithm, one of the objectives would be randomly selected to be active and the focus of the search.

Being fully multi-objective, the goal of their algorithm was to generate a Pareto front of non-dominated solutions that had a wide, even spread that covered much of the search space.

The concept of Pareto dominance and the Pareto front is covered later in section 2.2.3.1.

As before, all the possible single moves would be made and evaluated, but this time only for the selected objective. The best non taboo move for the selected objective is made and the process is repeated.

All of the candidate solutions that did not violate any constraints were compared to the Pareto front of non-dominated solutions and if they were not dominated, they were added. Any existing solutions in the Pareto set that were dominated by the new entries were discarded from the list.

In order to promote the spread of the Pareto front, they included a diversification procedure. When the algorithm had not found any new undominated solutions for a preset number of moves, the algorithm would randomly select one solution out of the Pareto set to be the new current solution. At the same time the tabu list is reset to empty and the search begins again.

The algorithm's goal of finding a Pareto set of trade-off solutions that had a wide spread was achieved, though it was unclear as to how good the solutions in the resultant Pareto set were as they were not compared to previous attempts in the literature.

Multi-objective Evolutionary Algorithm NSGA-II

Salazar *et al.* (2006) note that although single objective optimisations can be computationally efficient, they only give a very narrow view of the search space. Often, a broader set of possible solutions is required, to provide the decision maker with a choice amongst trade-off solutions.

In this situation, the computational efficiency of single objective optimisations is counteracted by the need to repeat the process many times, each time varying the constraints.

They suggest that using a modern, multi-objective evolutionary algorithm to generate a Pareto front of undominated solutions overcomes this, and presents the decision maker with rich information for making trade-off choices.

This is demonstrated by applying the Non-dominated Sorting Genetic Algorithm-II (NSGA-II), devised by Deb *et al.* (2002), to the benchmark problem.

In their discussion of the results, a good comparison is made of the outcomes of single objective versus multiple objective optimisation. The single

objective optimisation is tasked with maximising a single value, which is reliability in the case of the benchmark problem. This means that improvements in this dimension are always accepted over other solutions, even if it is a relatively small gain in reliability for a very large cost.

This is illustrated with an example from the result set of the NSGA-II algorithm. When a weight constraint of 191 units is set, the range of reliabilities for the solutions in the Pareto set represents only 2.84% of the reliability. However, in the cost dimension the range of those solutions is 31.71%.

A decision maker faced with this may decide that the reliability benefit is too small to be worth the extra cost, and opt for the cheaper solution. This is a benefit of treating the objectives separately, as a single objective optimisation could not provide this information.

Again, as with the multi-objective tabu search algorithm, the goal of the demonstration was to produce a wide and evenly spread Pareto front, which was shown graphically, and the quality of the solutions with regard to optimality was not covered.

However, insight into the quality can be derived from their above trade-off discussion. The range of reliabilities for the 191 unit weight constraint example had an upper value of 0.9834. This gives an MPI for that test case of -25.85%, which is relatively low; less in fact than any of the other approaches.

They conclude that although a single objective optimisation can be computationally efficient, it is very difficult to know beforehand whether this would be the case compared to a multi-objective optimisation. Further, perhaps a multi-objective optimisation could be used to provide information about the search space that could be used by the decision maker to formulate as a single objective optimisation to further refine the solutions.

2.2.2 Limitations of the Reliability Block Diagram

In the majority of cases where optimisation algorithms are used to optimise reliability, a formulation based on the reliability block diagram (RBD) model is used to calculate the reliability.

An advantage of using this method is that the calculation is a simple mathematical equation that can be quickly evaluated.

There are several limitations to this method however.

The first is that systems are designed using complex engineering models that are composed of subsystems that have deep hierarchies and many connections to and from each component.

RBDs conform to a simple arrangement of series-parallel components with an input and an output. Conversion between the two is a necessary and non-trivial step that can be time consuming and can introduce errors into the model. In any case it is a simplification of the original design.

Once the RBD system has been optimised it will be necessary to reverse this procedure in order to determine the modification required to improve the reliability of the real design.

Another disadvantage is that typically the failure model of an RBD allows just two failure states: failure and success. In reality a system can fail with many failure modes, a fact that is catered for by true safety analysis techniques.

It would seem an ideal partnership to make use of sophisticated optimisation algorithms to optimise the reliability, with respect to other objectives, of a full engineering model by using an automated safety analysis tool to calculate said reliability.

2.2.3 Multiple objectives versus single objectives

The first use of meta-heuristics in general, and for the RAP problem in particular was for solving single-objective formulations. The goal of the optimisation was to maximize (or minimize) a single objective.

Where problems come with multiple objectives, they need to be combined into a single objective. Often this is done through a weighted sum, or product, of the objectives. Alternatively, one objective is chosen as the main objective and the others are aggregated in the form of constraints.

One of the problems with doing this is that it takes what is in fact a multi-objective problem and forces it in to the confines of a single-objective method. This was achieved through the *a priori* weighting, scaling and combination of the objectives into penalizing constraints.

Reliability was not directly maximized but rather a reliability value that had been modified by an adaptive penalty function in order to allow for the additional objectives of minimizing cost and minimizing weight.

When the problem is formulated as a single objective it is clear when one solution is better than another. There is a single objective value, one is less, the other more, with the better one depending on whether the objective is being minimized or maximized.

When you consider them as multiple objectives this becomes more difficult because unless one solution clearly dominates another, one cannot be said to be better.

2.2.3.1 Pareto frontiers

Many multi-objective optimisation algorithms use Pareto dominance to distinguish between different solutions.

For an individual to dominate another it is necessary for it to be the same or better for each of the criteria. In Figure 1 there are three examples where one individual (A) dominates another individual (B). In each case individuals with lower values of the two criteria of comparison (the axes of the chart) are deemed better for that criterion, as with cost and unavailability for example.

Availability is defined as the probability that a repairable system will be operating at a specified time. When a system is not repaired, its availability is equal to its reliability. The definition of unavailability (Q), is simply that the system will *not* be operating at a specified time and is given by:

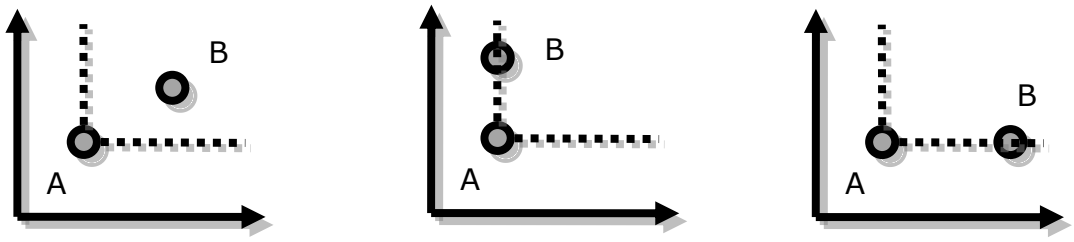

Figure 1 In each of these examples individual A dominates individual B.

In the first example, A dominates B because it has a better value (lower in this case) for both criteria. In the second and third example A dominates B because it has a better value of one criterion and is equal to B in the other criterion.

Figure 2 shows an example where neither individual A nor individual B dominates the other; they represent a Pareto optimal trade-off where moving between them cannot be achieved without making a sacrifice in at least one of the criteria. For example individual A may be a low cost, highly unavailable solution where individual B is a high cost solution with low unavailability. To move from A to B in order to achieve a better unavailability requires that more is paid in cost. Likewise a move from B to A to

achieve a lower cost is only feasible by sacrificing the availability of the solution.

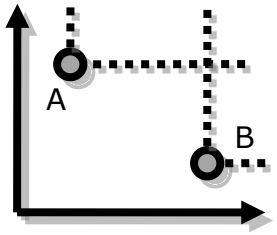

Figure 2 In this example both individual A and individual B are undominated by one another.

Figure 3 shows an example Pareto frontier with more individuals. The undominated individuals shown shaded in grey are said to be on the Pareto frontier and as such are trade-offs for each other. The unshaded individuals are dominated by the individuals on the Pareto frontier (and perhaps by each other).

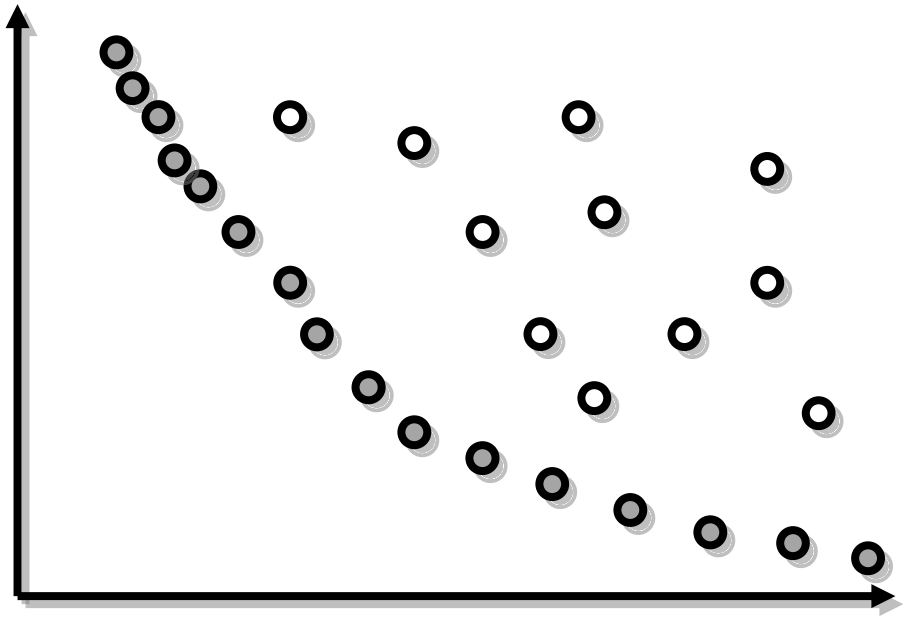

Figure 3 A Pareto frontier (shaded individuals) with dominated individuals also shown (unshaded).

In all cases it is better to select an undominated individual over a dominated individual, but given the tradeoff nature of the solutions on the Pareto frontier, it is necessary to use further criteria to distinguish between them.

A Pareto set contains solutions that do not dominate one another. However, there may be undiscovered solutions that dominate some or all of the solutions in a given set. This being the case, unless the Pareto set is globally optimal then it represents a snapshot of the current state. For this reason, a wide-spreading Pareto front can still contain poor solutions.

This would normally be left to a human agent to make a decision when selecting between tradeoffs. In the simple case how does one mechanically choose between a cheap less reliable option and an expensive more reliable option?

2.2.3.2 Pareto pruning

One of the potential drawbacks of multi-objective optimisation is that the resultant Pareto trade-off sets can become very large. This information overload can limit the usefulness of this rich information to the decision maker; good solutions that meet the more specific requirements of the user can be lost in the deluge of alternatives.

Taboada *et al.* (2007) suggest 2 different techniques for reducing the size of the Pareto set.

A Priori Objective Preference

The first is appropriate for use when the decision maker has advance preferences between objectives. This preference is encapsulated in an ordering of the objectives, for example, reliability > weight > cost. Each objective would then have a possible weighting range calculated via a weight function, such that the sum of the weights is equal to 1.

A Monte Carlo simulation method, using this weight pattern, generates a large set of actual weighting values. The weighting values are used to aggregate the multiple objectives into a single combined objective, and the best solution in the Pareto set for that objective is noted.

This process is repeated several thousand times, and the most frequently identified solutions are retained for the reduced Pareto set.

The technique has been shown to achieve a 90% reduction in the size of the Pareto set.

A similar technique is used by Kultural-Konak *et al.* (2008) to prune the Pareto set following a multi-objective tabu search.

Two different order preferences were devised as the basis for the Monte Carlo method. The Pareto set is reduced from 3801 solutions to 56 solutions for the first preference and 36 solutions for the second preference. By combining the results of both preferences only 5 solutions could be considered priority solutions as they frequently appeared in both pruned sets.

Data clustering analysis

It may be the case that the decision maker is insufficiently experienced, or not in possession of sufficient information, to be able to state a preference. This is where their second proposed technique, data clustering analysis, is advised.

The data clustering method is described in detail by Taboada and Coit (2007). The purpose of the analysis is to identify groups of solutions such that the individuals in the group are highly similar to one another whilst being highly dissimilar to members of other groups.

Once a number of clusters have been identified, representative solutions can be chosen from each cluster. Usually the solution that is closest to the centre of the group is chosen as the representative. This substantially reduces the number of solutions that a decision maker has to consider.

If further analysis is required, then attention can be focused on a particular cluster. Most useful clusters to focus on are areas where large deterioration of one objective is caused by a small improvement in another, a so-called 'knee' region.

The cluster analysis can be repeated on this chosen region to further reduce and refine the solution set; finally the decision maker can select one solution from a much smaller set than the original Pareto set from the optimisation.

A small example demonstrating the clustering technique reduced a Pareto set of 46 solutions to just 3.

Li (2009) also uses a data clustering algorithm to reduce the Pareto set to a manageable level. The difference to previous approaches is the use of a Self Organising Map to identify the clusters. Another difference is that a data envelopment analysis is performed on the clusters to provide the representative solutions, rather than just selecting the most central one.

In a small example the Pareto set was reduced from 75 solutions to just 2 solutions.

2.2.4 Choosing a Multi-Objective Evolutionary Algorithm

A historical survey of multi-objective evolutionary algorithms spanning the last two decades is provided by Coello (2006).

It states that whilst there are a wide range of exact mathematical techniques for solving multi-objective problems, they suffer from limitations

regarding the shape of the Pareto front, particularly if the front has a concave shape or is disjointed. Some also require that the objective functions and their constraints be differentiable.

A further limitation is that they mostly can only identify a single member of the Pareto set per run, requiring the repeat of the calculations from different starting locations.

Both of these limitations are overcome through the use of evolutionary algorithms such as genetic algorithms. Not only does their use of a population of candidate solutions allow multiple Pareto front members to be located in a single run, but also discontinuous and concave Pareto fronts are not problematic for a modern evolutionary algorithm.

Evolutionary algorithms are also able to solve problems for which a mathematical formulation is not available or difficult to construct.

This survey classifies multi-objective evolutionary algorithms into two categories: first generation and second generation, where the second generation improves and supersedes the first generation algorithms.

One of the key features that distinguish second generation methods is the inclusion of an archive population of non-dominated solutions in order to provide an elitism mechanism.

Elitism is a mechanism that prevents good solutions from being lost from the gene pool. This is particularly important when generating Pareto sets as individual solutions are only non-dominated with respect to the other solutions in the current population. If non-dominated solutions are lost from the population then solutions that they would have dominated could be presented to the decision maker even though they are not globally non-dominated.

The archive population allows the storage of all non-dominated solutions found by the algorithm to be saved so that any solutions that are presented to the decision maker are known to be the best of all the solutions found by the algorithm.

The focus of these Pareto-based evolutionary algorithms is to populate a Pareto optimal set with solutions that are widely and evenly distributed across the search space. The basic framework in which they work is similar; they are distinguished primarily by the different mechanisms for moving the Pareto front towards the optimum and the different diversification mechanisms for ensuring the spread of the solutions.

2.2.4.1 PESA-II

Pareto envelope-based selection algorithm (PESA-II) (CORNE, D W et al., 2001) is an improved version of the original PESA method (CORNE, D et al., 2000).

PESA-II seeks to maximise evenly distributed spread in the Pareto set by including crowding in the selection criteria of parent solutions for the genetic operators. Solutions that are found in less crowded regions are preferred for selection as these regions represent less explored areas of the search space.

For establishing crowding, the mechanism involves dividing the population with a hypergrid in objective space. A pre-specified dimension value determines the number of hyperboxes dividing up the space.

Once the grid is laid, solutions can be found to fall within one of the hyperboxes. Some hyperboxes will contain several solutions and the more solutions there are in a box the more crowded that box is said to be.

For selecting individuals for breeding, two hyperboxes are randomly chosen from all the hyperboxes that contain solutions. The number of solutions

within each box, the crowding value, is compared and the box with the smallest value is chosen.

Finally an individual from the chosen hyperbox is randomly selected to be a parent.

The crowding evaluation is also used to select individuals for deletion from the population when size limits have been exceeded. Opposite to breeding selection, solutions that are in crowded regions are preferred for removal over solutions in uncrowded regions.

In their paper PESA-II was favourably compared to the original SPEA algorithm (ZITZLER, E and Thiele, L, 1999) and also to Pareto Archived Evolution Strategy (PAES) (KNOWLES, J D and Corne, D W, 2000), when applied to a test problem.

2.2.4.2 SPEA2

Strength Pareto Evolutionary Algorithm (SPEA2) (ZITZLER, E et al., 2001) is an improved version of the original SPEA (ZITZLER, E and Thiele, L, 1999).

Unlike PESA-II, which adopts a pure elitist strategy that allows no dominated solutions to exist in the archive, SPEA2 does retain dominated solutions.

The comparison of different solutions in the SPEA2 algorithm is made with a fitness value comprising two parts: dominance strength and a density value.

The population is surveyed and each solution counts the number of other solutions it dominates. This is its dominance strength. A raw fitness for each solution is then calculated by summing together the dominance strengths of all the solutions that dominate it.

Non-dominated solutions will therefore have a raw fitness of 0.

The density value is calculated by determining the distance, in objective space, between the individual and every other solution. These distances are then sorted into ascending order and the k^{th} value, where k is the square root of the population size, is chosen.

This value has 2 added to it to ensure that the final value is greater than zero and less than one. Finally the density value is calculated by dividing 1 by it.

The final fitness value is the sum of the raw fitness, from the strength calculation, and the density value.

Completely non-dominated solutions will have a fitness of less than one.

Both dominated and un-dominated solutions are allowed in the archive population, provided that the limit is not reached. When the limit is exceeded, first dominated solutions and then un-dominated solutions with high density are removed.

2.2.4.3 NSGA-II

Non-dominated Sorting Genetic Algorithm (NSGA-II) (DEB, K et al., 2000) improved on the criticisms of the original NSGA (SRINIVAS, N and Deb, K, 1995).

NSGA-II does not have a separate archive population. Instead both non-dominated solutions and dominated solutions exist in one population.

The solutions in the population are sorted into rankings of dominance. In the first rank are the completely non-dominated solutions. In the second rank are the individuals that are only dominated by the first rank. The third rank contains the solutions that are only dominated by the first and second ranks; and so on.

The elitism is provided by giving selection preference to non-dominated solutions. When selecting, two solutions are randomly chosen from the population. The individual with the highest non-dominance rank is chosen.

Where the two solutions are of equal dominance rank they are chosen on the basis of their crowding.

Crowding is calculated by ordering all of the solutions in each of the dominance ranks by each of their objective values in turn. The difference between the values of the nearest neighbour on each side is averaged for all objectives. This gives an individual crowding distance.

For breeding selection, as with PESA-II the individual with the largest surrounding space is chosen to promote spread of the Pareto front.

When the population exceeds its limit, the entire population is ordered, first by dominance rank, and then by crowding distance. Solutions are then removed from the bottom of the list until the population is within limits again.

This ensures that the undominated solutions with low crowding are preserved at the expense of dominated solutions in crowded regions.

2.2.4.4 Comparison of Techniques

A survey of different multi-objective GAs (KONAK, A et al., 2006) suggested that PESA, and by extension PESA-II, is easy to implement and computationally efficient. NSGA-II is also described as efficient and is widely used and well tested.

SPEA2, by contrast, is criticised for being computationally expensive.

Both NSGA-II and SPEA2 were applied to a hierarchical redundancy allocation problem by Kumar *et al.* (2009). The quality of the Pareto set was found to be better when NSGA-II was used than when SPEA2 was used.

In another application (HIROYASU, T et al., 2005), optimising fuel economy and emissions in a diesel engine SPEA2 and NSGA-II were compared and found to have similar solution quality.

For the purposes of reliability optimisation using automated safety analysis techniques, efficiency is particularly important. Both the PESA variants and NSGA-II have been noted for being efficient algorithms.

Additionally NSGA-II has been successfully applied to many problems, including a redundancy allocation problem, and not just by the authors of NSGA-II.

For these reasons NSGA-II and PESA-II have been chosen for investigation in this thesis. They will both be implemented and evaluated when applied to the benchmark problem with the automated safety and reliability analysis tool HiP-HOPS providing the reliability fitness values.

A more detailed description of their mechanisms and implementation, including efficiency improvements, can be found in later sections of this thesis.

3. Extending an automated reliability analysis technique

In chapter 2, HiP-HOPS was identified as a good candidate for providing reliability analysis to optimisation algorithms. In this chapter HiP-HOPS is discussed in greater detail, providing a basis for the extensions that were developed to improve the efficiency of the algorithms. The performance enhancements are discussed and evaluated.

HiP-HOPS is a safety analysis technique based on deductive FTA. It works by automatically synthesising, and then analysing, multiple interconnected system fault trees for models that have been augmented with component failure data.

In addition to traditional FTA, further analysis of the fault tree cut sets automatically generates FMEA tables that, unlike traditional manual FMEA, consider more than one failure mode.

As discussed in the background section, many safety analysis techniques are inductive, with high computational expense, making them unsuitable for use with optimisation algorithms due to the need for many iterations.

The automatic fault tree synthesis of HiP-HOPS is deductive and scalable, opening up the possibility of optimisation of system models with dependability objectives.

The rest of this chapter describes the three HiP-HOPS phases, annotation, synthesis, and analysis, along with the performance enhancements that have been developed to minimise the evaluation overhead in optimisation. In order to introduce some of the major concepts, a simple example of a standby recovery system, shown in Figure 4, will be used.

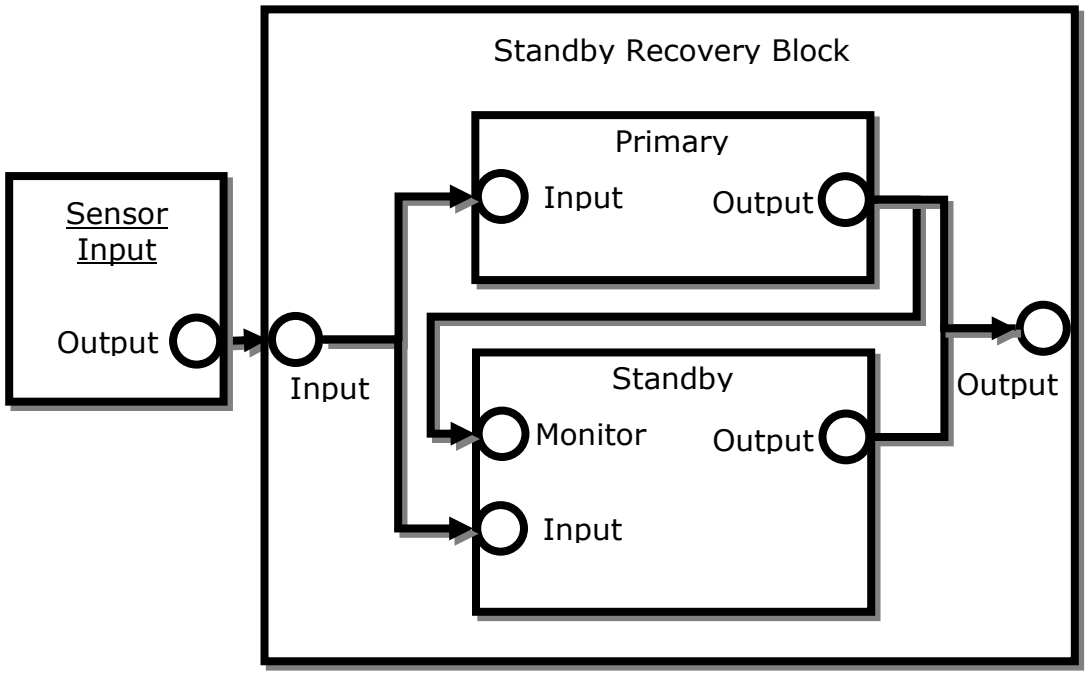

Figure 4 - A simple example of a standby recovery system

It is composed of one input component, 'SensorInput', and one subsystem, 'Standby Recovery Block', which is composed itself of two sub-components, 'Primary', and 'Standby'. 'Primary' is the main subcomponent of the standby-recovery system and processes the input from the 'SensorInput'. The 'Standby' component monitors the output from 'Primary' and is designed to take over operation if it detects a failure of the 'Primary'.

3.1 Annotation Phase

HiP-HOPS can be performed on any system model that identifies functions or components along with the energy, material, or data connections between them. For the purposes of HiP-HOPS, the components of the system are augmented with failure behaviour annotations. These annotations describe the causes of deviations of the normal output of the component. Output deviations can be caused either by internal failure modes of the component,

or through the propagation or transformation of deviations of component input.

The relationship between the output deviations and their causes is described using Boolean logic.

The deviations, both input and output, are composed of two parts: the failure class of the deviation and the port of the component that it occurs at.

Failure classes generally fall into one of several categories:

- Omission: input or output expected but none provided
- Commission: input or output unexpectedly provided at a port
- Value failure: input or output provided at a port but with a value outside of intended range
- Timing failure: input or output provided at port either before expected (early), or after expected (late)

In principle, however, HiP-HOPS can cope with any definition of failure class providing that they are used consistently throughout the model. An example of a complete deviation is 'Omission – Primary.Output', which states that the output of component named 'Primary' is omitted.

Table 1 shows the failure modes of the components in the example system.

Table 1- Failure modes for standby recovery system

Component	Failure Modes	Probability
Standby Recovery Block	ElectroMagneticInterferance	0.001
Primary	InternalFailure	0.03
Standby	InternalFailure	0.02
SensorInput	InternalFailure	0.05

Table 2 shows the failure annotations for the components in the example system.

Table 2- Failure data for standby recovery system

Component	Output deviations	Failure expressions
Standby Recovery Block	Omission - Output	Omission – Primary.Output AND Omission – Standby.Output OR ElectoMagneticInterference
Primary	Omission – Output	Omission – Input OR Failure
Standby	Omission – Output	Omission – Monitor AND (Omission – Input OR Failure)
SensorInput	Omission - Output	Failure

Table 2 defines that the omission of output of the SensorInput component is caused only by its internal failure.

The 'Monitor' port detects omissions of output of the 'Primary' component before activating the 'Standby' component. If the 'Primary' component is functioning then the 'Standby' cannot fail (or its failure is irrelevant) as it is inactive. If the 'Standby' component is thus activated then an output omission is caused by either an internal failure or the propagation of input omission.

An omission of 'Primary' output is caused by an internal failure or the propagation of an omission of input.

At the top level, the system output of the 'Standby Recovery Block' has an output deviation of type 'Omission' that is caused by either the failure mode 'electromagnetic interference', or by the conjunction of omissions occurring at the outputs of both the primary and standby components.

The Boolean failure expressions in the component annotations each define a CFT that describes propagation of failure through it. However, such a fault

tree is incomplete because its leaf nodes will not all be failure modes – some may be input deviations, which are failures originating in other components. Similarly the top node is an output deviation that may be relevant in further components in the system.

These CFTs for the example are shown in Figure 5 (for the 'Standby Recovery Block'), Figure 6 (for the 'Primary' component), Figure 7 (for the 'Standby' component), and Figure 8 (for the 'SensorInput' component). In these diagrams the circles with an arrow denote an input deviation where the arrow is entering the circle and an output deviation where the arrow is leaving the circle.

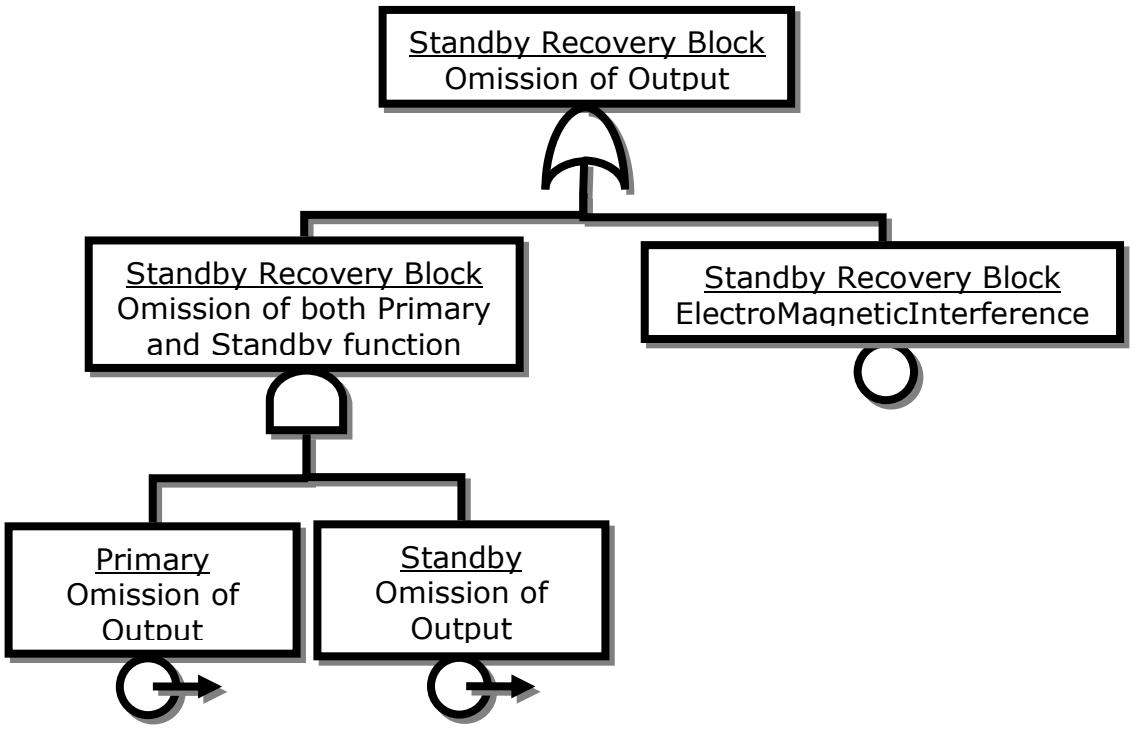

Figure 5 CFT described by annotation for standby recovery block subsystem

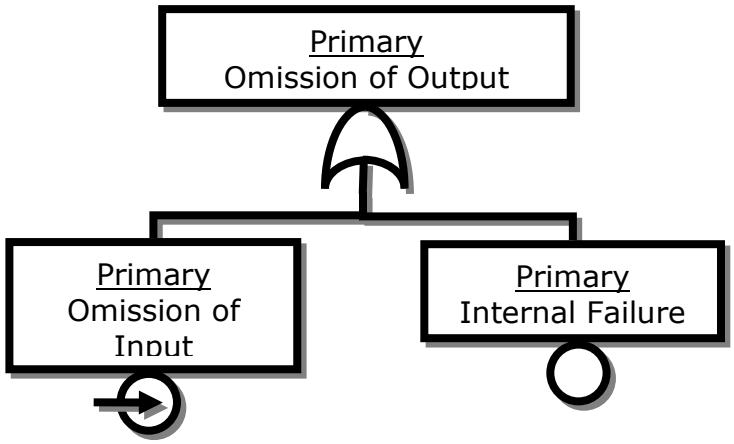

Figure 6 - CFT described by annotation for primary component

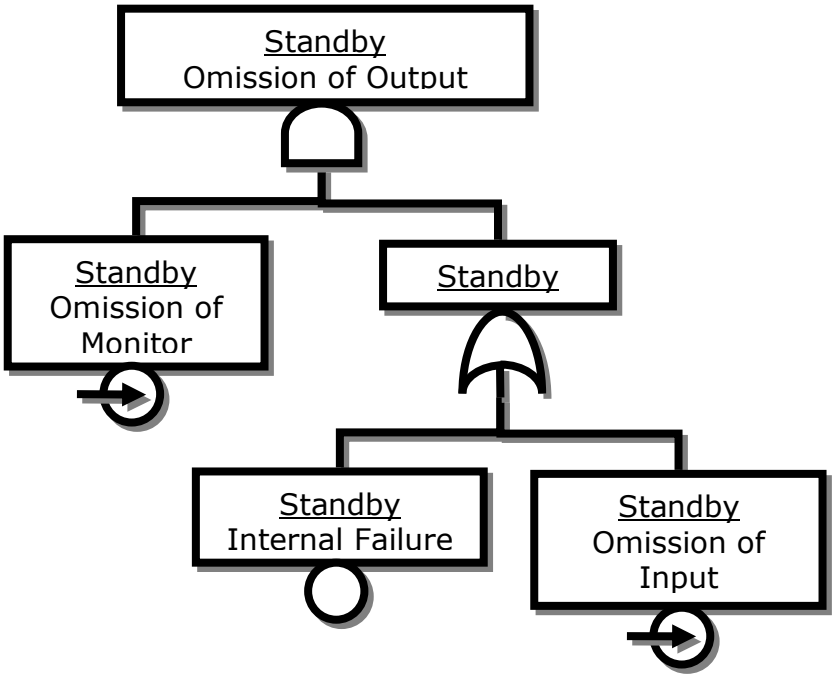

Figure 7 - CFT described by annotation for standby component

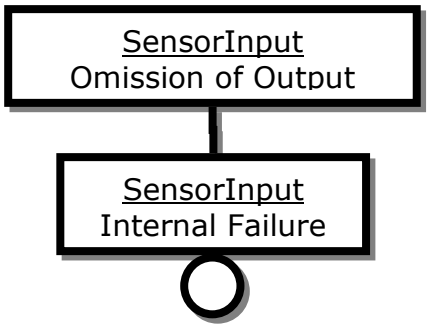

Figure 8 - CFT described by annotation for SensorInput component

Once the components of the model have been fully annotated, the manual phase of HiP-HOPS is complete and the remaining stages are fully automatic.

3.2 Synthesis Phase

During the annotation phase the designer manually adds failure expressions to each component. As explained before, these expressions represent CFTs that describe the propagation, generation, and transformation of failure between the inputs of the component and its outputs.

As the components are linked through their ports to other components, failures can be propagated between components.

The synthesis phase begins with deviations of the system outputs. The algorithm locates the CFT for each output deviation and traverses the tree until it locates a terminal input deviation. The input port that is associated with that input deviation is then selected.

The algorithm then follows the connections from the selected component port to the output ports of the connected components. The output deviation matching the failure class of the connected input deviation is then selected. Its CFT is joined to the input deviation from the connected component.

This cycle of traversing CFTs and connections is repeated until there are no more unconnected input deviations and the system fault trees are complete.

In the standby recovery example in Figure 4, the system output is the omission of output of the 'Standby Recovery Block'. This is the top node of a CFT in the 'Standby Recovery Block' and marks the start of the synthesis for this example.

This CFT has two input deviations at its leaves, 'omission of Primary output' and 'omission of Standby output', and so the algorithm finds the corresponding components where it discovers further CFTs. The top nodes of these are added as child branches to the input deviation leaf nodes and the process is repeated, each time connecting output deviation CFTs to input deviation leaf nodes.

The omission of 'Primary' output is partly caused by an omission of its input, so the connection at the input is followed to the 'SensorInput' component where the output deviation that exists there is connected to the growing system fault tree. As the deviation of 'SensorInput' output is only caused by an internal failure, the propagation of that branch is terminated.

The 'omission of Standby input' branch is treated in the same way and thus a complete system fault tree, describing the propagation of failure throughout the whole model, is synthesised from the failure expressions. The result of synthesis of the example model is shown in Figure 9. It can be seen from this diagram how the system fault tree is composed of the mini fault trees shown in Figure 5 to Figure 8. The output deviations that are the top nodes of the component fault trees are shaded.

In the next HiP-HOPS phase the fault tree is analysed to extract quantitative and qualitative information.

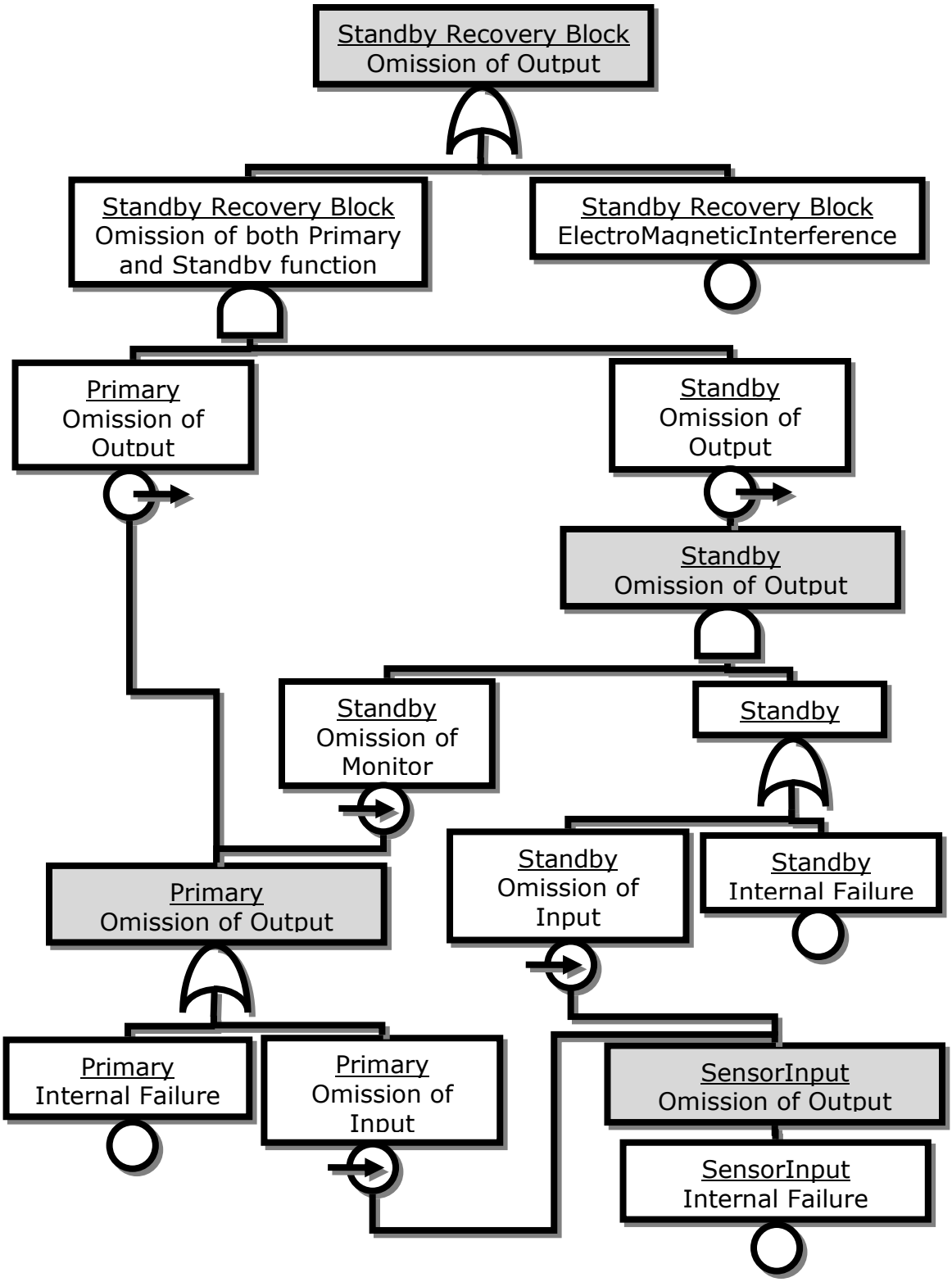

Figure 9 - Fault tree synthesised from standby recovery example.

3.3 Analysis

The result of the synthesis process is a set of one or more interconnected fault trees and therefore the next stage of the HiP-HOPS technique is to analyse those fault trees using FTA.

The fault trees represent the propagation of failure logic through the system, but they can often be large and complex. By reducing the fault trees to their minimal cut sets we retain the relationship between the basic events and the top level system event but strip out the intermediate propagation paths.

The primary cut set generating algorithm used by HiP-HOPS is MICSUP (MInimal Cut Sets UPwards) (PANDE, P K *et al.*, 1975). MICSUP, as the name suggests, is a bottom-up algorithm for obtaining minimal cut sets from a fault tree.

One advantage of MICSUP is that the cut sets can be stored in the intermediate nodes as they are generated and minimised as the algorithm returns from the basic events. This ability to reuse the results in shared branches without needing to reanalyse saves computational effort.

A second advantage is that it is easy to use cut set order pruning: when a cut set exceeds the maximum size limit, it is simply discarded at that point and cannot contribute to cut sets further up the tree.

The main computational expense when minimising the cut sets is the redundancy checking. Several methods of increasing the performance of this process, including modularisation, fault tree contraction, and use of cut set cataloguing, are discussed in a later section on performance increases.

The following Boolean laws can be applied to obtain minimal cut sets:

The law of absorption: E1 + ~~E1.E2~~ = E1

The cut set containing E1.E2 was removed as the action of E1 alone is sufficient to cause the top event and is therefore in its minimal form.

The laws of idempotence: E1.~~E1~~ = E1 and E1 + ~~E1~~ = E1

The former removes repeated events within cut sets and the latter removes repeated cut sets.

In order to keep the number of checks to a minimum the cut sets are checked for redundancy as they are created so that redundant combinations are quickly identified and removed. This ensures that they cannot affect or be combined with more cut sets later in the traversal of the fault tree.

Once the minimum cut sets have been identified, they can subsequently be used for quantitative analysis to calculate the system unavailability Q_s (where basic events have quantitative data) using the approximate Esary Prochan (1970) method:

(Where n is the number of independent cut sets and Q_{CS} is the unavailability of the cut set i).

In addition to the quantitative analysis that can be performed on the minimal cut sets, a further qualitative stage can be applied to generate an FMEA. Figure 10 shows the inverse relationship between the diagnostic failure propagation information in the fault trees, where the component failure modes that cause a system failure can be determined, and the causative nature of the FMEA, where a basic event (or combination of several events) have an effect on the system level.

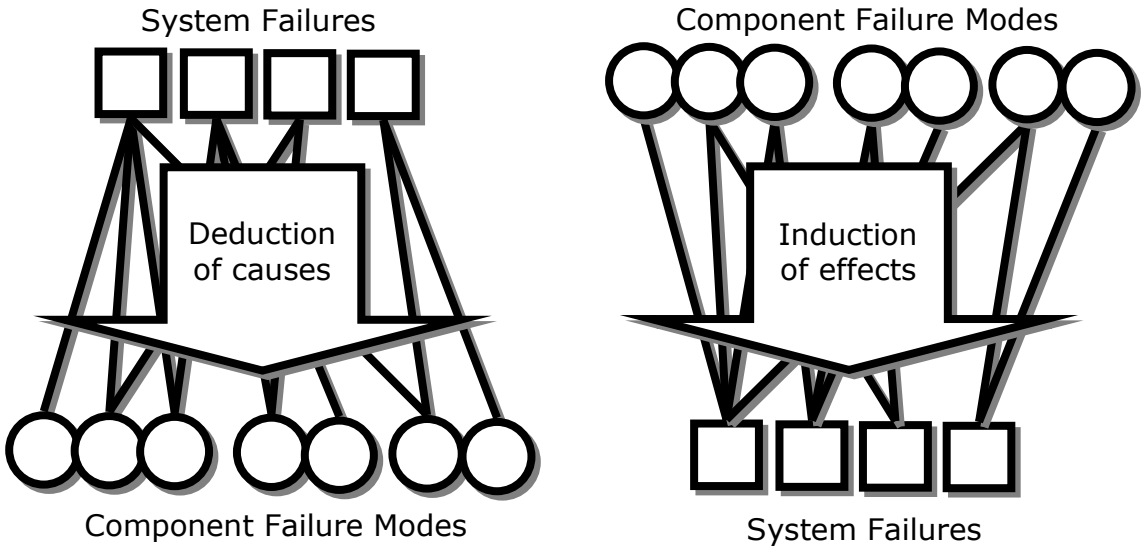

Figure 10 - Inverse relationship between fault trees (left) and FMEA (right)

The minimal cut sets contain the non-redundant propagation of failure in the fault tree and an algorithm is used to catalogue each component failure mode in each fault tree and note which system failures they cause and in combination with which other component failure modes. This information is the core of an FMEA.

The deductive nature of this process is important for safety analysis as it allows large combinations of basic events to be considered in the FMEA, unlike traditional manual methods that could only consider single points of failure or fault injection simulation methods that are similarly limited by combinatorial explosion.

The method and data structure for FMEA cataloguing is discussed further in the following section on performance enhancements that were done in the context of this work in order to support fast iterations of analysis in optimi-sation.

3.4 Performance Enhancements

It is desirable when conducting single safety analyses to have computationally efficient algorithms that minimise the time needed to perform the calculations. However, when those analyses are used to provide fitness evaluations for optimisation algorithms, it becomes crucial.

Optimisation algorithms require many iterations, not just one or two or three but thousands, resulting in possibly millions of analyses. To put this into real values, consider two algorithms, one of which takes a second longer than the other. This is a negligible difference for a single analysis, but when it is applied to an optimisation algorithm requiring the analyses to be run a million times, that one second difference becomes a 12 day difference. These apparently small delays can be the difference between optimisation being viable and impractical.

It is with this context in mind that the remainder of this section discusses algorithms that have been implemented to increase the performance of the HiP-HOPS fault tree analysis.

3.4.1 Fault tree contraction

The automatic process of fault tree synthesis in HiP-HOPS constructs a fault tree by traversing the compositional system model and combining the failure logic. The resultant fault tree is not necessarily in its most compact or simplified form and may contain redundant elements that can be removed without changing the meaning of the fault tree.

This can be seen in the example fault tree in which was synthesised for the standby recovery example system.

Firstly all nodes in the tree which only have a single child act merely as a proxy label and can be removed.

Once this is done (see Figure 11), note how two AND gates appear together near the top of the tree. The extra logic causes extra computational expense, and in their description of SETS, Worrell and Stack (1978) explain how two or more gates of the same kind appearing consecutively in the fault tree may be coalesced to produce a simpler structure for analysis. An example of this can be seen in Figure 12.

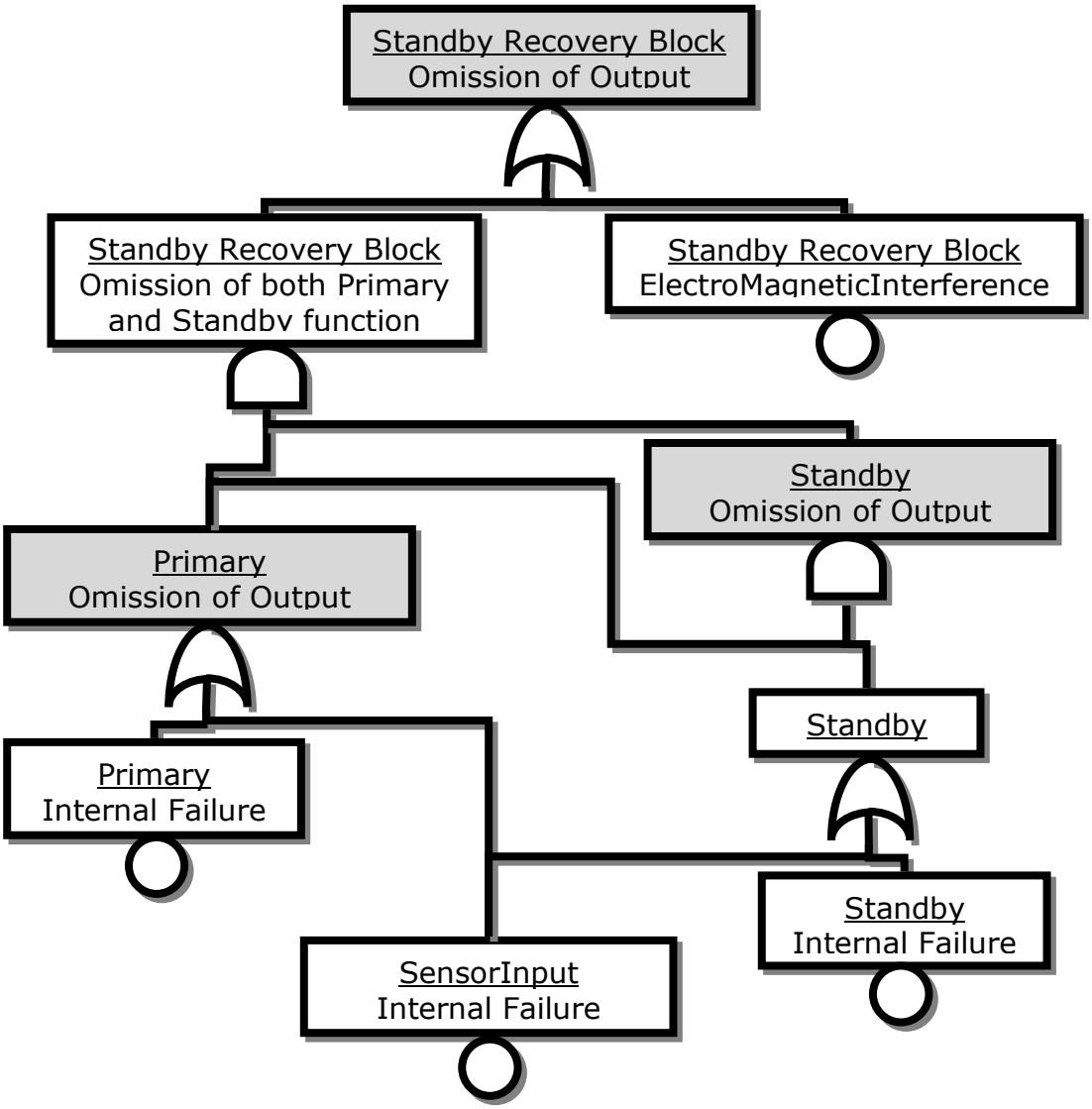

Figure 11 – Synthesised fault tree example with proxy nodes that have single children removed.

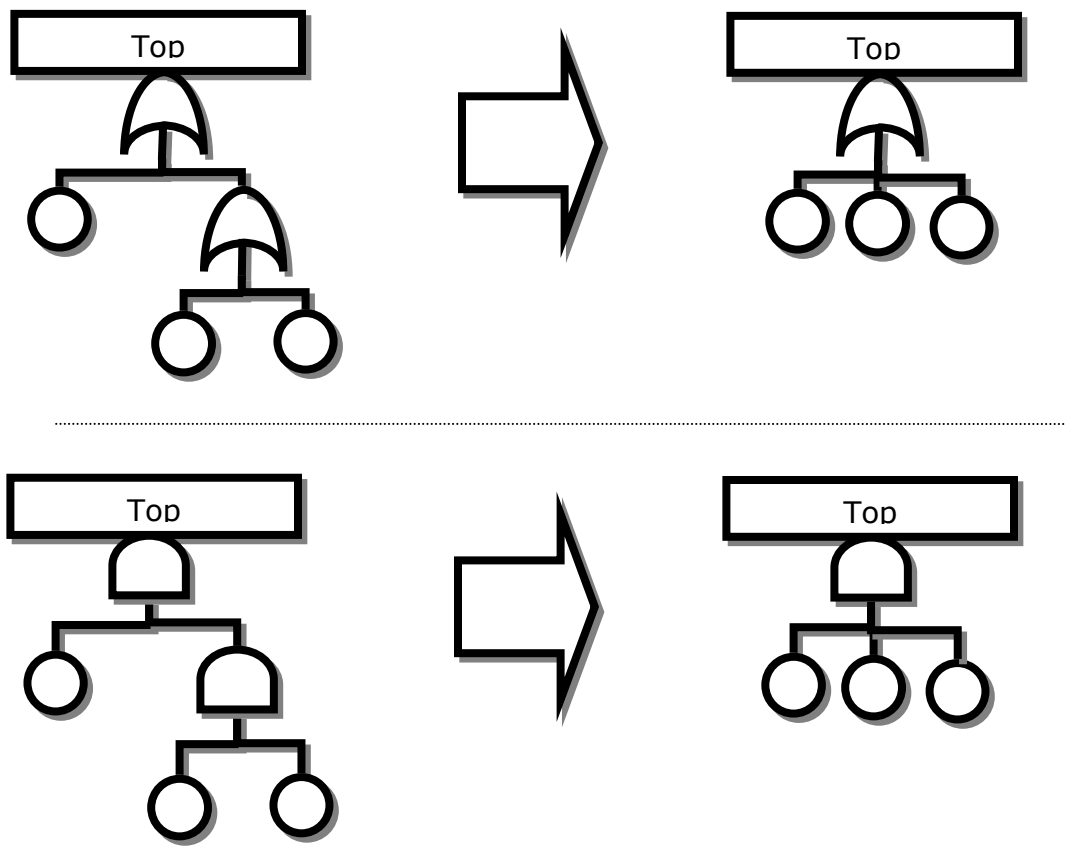

Figure 12 - An example of coalescing fault tree gates where two or more consecutive gates of the same type get combined.

The combination of these simplification rules leads to a much simpler, smaller fault tree that still contains the same logical information, as shown in Figure 13.

The contraction method was tested on a large, complex fault tree to determine the effectiveness of the contraction algorithm. The fault tree is synthesised from a model with hundreds of components and generates over 7000 cut sets (PAPADOPOULOS, Y and Grante, C, 2005).

The timings were measured using MicroFocus Devpartner, which instruments the development code, thus allowing detailed analysis. It is worth noting that the instrumentation has a timing overhead. This does not affect the

consistency of the results, only that real world timings are less than the figures displayed.

For each timing test, the algorithms were run 3 times and the mean result was taken.

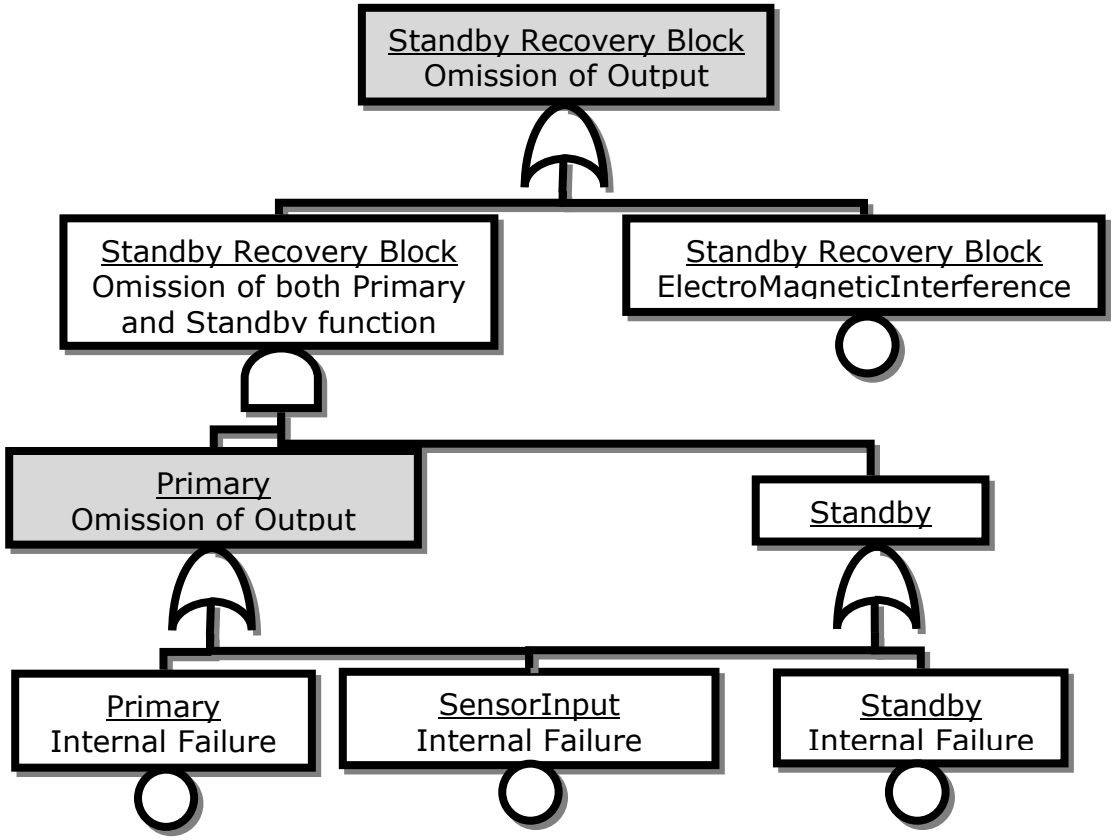

Figure 13 - The fault tree synthesised from the standby recovery example model shown in after it has undergone contraction.

Table 3 contains the combined times taken to contract the fault tree and then analyse it, compared to performing analysis on an un-contracted fault tree. Contraction of the fault tree led to an overall 61.34% reduction in the time taken despite the extra computational overhead of contracting the tree.

Table 3 - Comparison of times taken to minimise cut sets of a large fault tree with and without fault tree contraction

	Contracted	Un-Contracted	% Reduction
Time taken	1.83s	4.75s	61.34%

3.4.2 Modularisation

In SETS, Worrell and Stack (1978) talk about the use of independent sub trees (also called modules). If a branch of a fault tree contains only basic events that do not occur elsewhere in the tree, it is an independent sub tree and the cut sets that are generated from this branch do not need to be checked for redundancy. This can clearly result in a reduction in effort; however, there is also an overhead involved in identifying the modules.

In their paper, Dutuit and Rauzy (1996) propose an algorithm for identifying fault tree modules in linear time. This is achieved in two passes of the fault tree. The first pass is a depth-first, left-most traversal of the fault tree. At each step the 'date' counter is incremented and for each node in the tree the date of the first arrival at that node, the second arrival, and the latest arrival at that node is stored.

When visiting a gate node the first visit date is the first time that the gate is visited. The second visit is counted when returning from the last child of that gate. The last visit is updated on each subsequent visit. When visiting basic events, which occur as leaf nodes, the first and second visit are set as the same as there are no children. The latest visit is updated as normal.

After this first pass is completed, a second pass is carried out where for each gate the minimum date of the first visit of all its children and the maximum last visit of all its children is stored.

A gate is a module if and only if the minimum visit of its children is more than the date of the gate's first visit and the maximum visit of its children is less than the gate's second visit.

In Figure 14 there is a fault tree with such modules where the modules are shown enclosed in the dashed boxes (G3, G6, G7, G8).

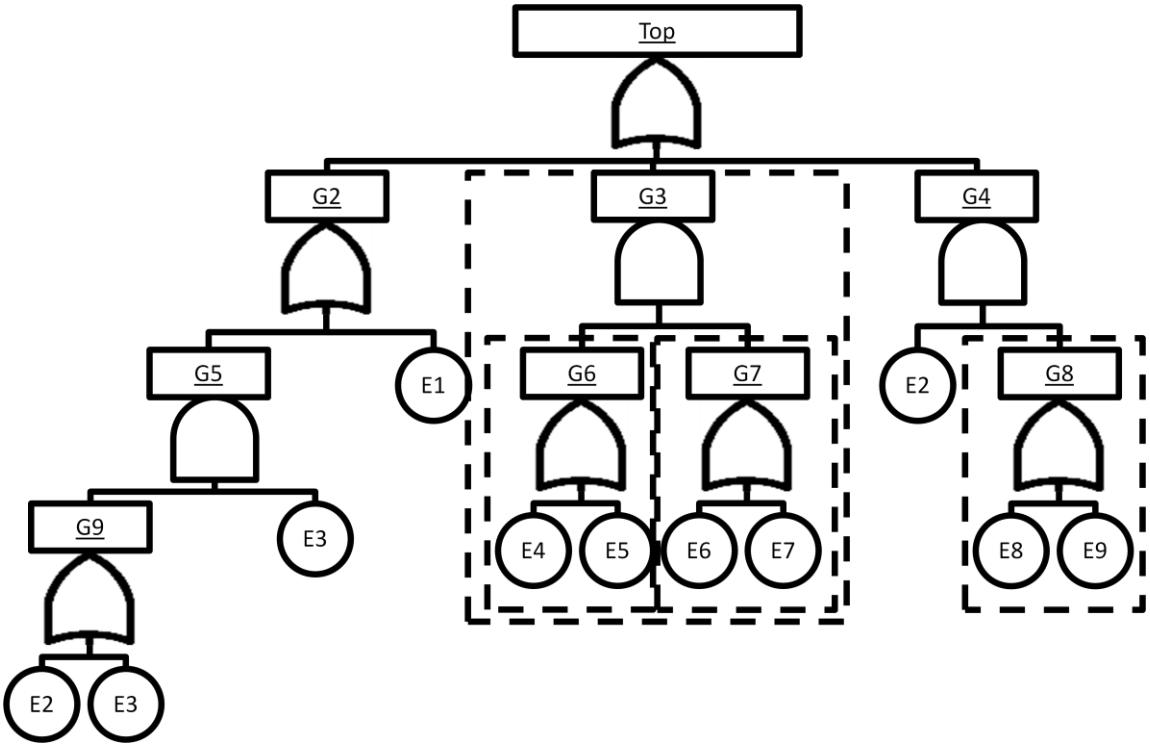

Figure 14 - Fault tree containing independent sub trees (modules). Modules are shown in the dashed boxes.

In Table 4 can be found the results from the linear module identifying algorithm for the example fault tree in Figure 14. Note that G3 is correctly identified as a module because the minimum visit of its children (12) is more than the date of the first visit to G3 (11) and the maximum visit of its children (19) is less than the date of the second visit to G3 (20). This is also true for G6, G7, and G8.

Table 4 - Results for the modularisation visit counting algorithm for example Figure 14 modules are shown in bold

	Nodes																
Visits	G2	**G3**	G4	G5	**G6**	**G7**	**G8**	G9	E1	E2	E3	E4	E5	E6	E7	E8	E9
1st	1	11	21	2	12	16	23	3	9	4	5	13	14	17	18	24	25
2nd	10	20	27	8	15	19	26	6	9	4	5	13	14	17	18	24	25
Last	10	20	27	8	15	19	26	6	9	22	7	13	14	17	18	24	25
Min.	4	12	4	4	13	17	24	4									
Max.	22	19	26	22	14	18	25	22									

Table 5 shows the results of the timed runs of cut set minimisation of a large fault tree (PAPADOPOULOS, Y and Grante, C, 2005). Again, the fault tree is synthesised from a model with hundreds of components and generates over 7000 cut sets. The fault tree contains 10 modules and the analysis is conducted with and without modularisation.

This shows that there is a significant performance increase shown when modularisation is used. The run using modularisation is 440 times faster than when modularisation is not used. Obviously this benefit is only achieved in fault trees that contain modules.

Table 5 - Comparison of times taken to minimise cut sets of a large fault tree with and without modularisation

	Modularised	**Not Modularised**	**% Reduction**
Time taken	1.836s	808.54s	99.773%

As explained previously, the main computational expense during analysis is checking for redundant cut sets. The speed increase offered by modularisation is tied to a reduction in the number of redundancy checks required. This can be seen in Table 6 where modularisation results in nearly 650 times fewer redundancy checks.

The discrepancy, between the 650 times fewer redundancy checks and the 440 times faster timings, is accounted for by the overhead of the modularisation algorithm.

Table 6 - Comparison of number of redundancy checks minimising cut sets of a large fault tree with and without modularisation

	Modularised	Not Modularised	% Reduction
Number of redundancy checks	38140	24765075	99.846%

3.4.3 Cut set order pruning

Performance in terms of algorithm speed can also be balanced with the accuracy of the estimation of unavailability. By limiting the maximum size of the cut sets allowed one can significantly reduce the time taken to generate the cut sets. The sacrifice is that as the cut sets are used to calculate the reliability of the system, omitting some of the cut sets from this calculation reduces its accuracy. As mentioned before there is a balance to be made and higher order cut sets reduce in significance with regard to system unavailability.

Table 7 shows the result of reducing the maximum allowed cut set size on the large fault tree previously used to test contraction and modularisation. The unavailability calculation is heavily influenced by the single order cut sets and so reducing the cut set size down to 1 has no effect on the accuracy of the calculation.

There is however a significant reduction in the time taken to analyse the fault tree. Reducing the cut set order limit from 4 to 3 decreased the number of final cut sets 1.44 times with a 1.47 times reduction in time taken. Further reducing the cut set limit to 2, results in 48 times fewer cut sets and a 5.28 times reduction in computation time. Finally, reducing to cut set order 1 gives 750 times fewer cut sets and takes 10.3 times less processing time.

Table 7 - Comparison of times taken to minimise cut sets of a large fault tree with varying cut set order pruning

	4	3	2	1
Number of Cut Sets	7498	5194	154	10
Unavailability	1.74E-11	1.74E-11	1.74E-11	1.74E-11
Time taken	1.836	1.243s	0.348s	0.178s
% Reduction of Max	n/a	32.32%	81.03%	90.31%
% Reduction in Unavailability	n/a	0%	0%	0%

In the case of this model, reducing the cut set order did not affect the resultant probability calculation, so the reduction in computational effort is without cost. This is certainly not the general case.

Table 8 shows the results of the same experiment carried out on a different large fault tree (PAPADOPOULOS, Y *et al.*, 2001). This tree has 6319 cut sets up to order 6. Reducing the maximum cut set order to 5 generates 3.77 times fewer cut sets and takes 2.49 times less time to process; the calculated probability is only 0.037% reduced. Reducing from cut set order 6 to order 3 and the time taken to perform the analysis is reduced by 7 times with only a 2.3% reduction in the calculated probability value.

Table 8 - Comparison of times taken to minimise cut sets of another large fault tree with varying cut set order pruning

	6	5	4	3
Number of Cut Sets	6319	1676	324	133
Unavailability	0.857977	0.857657	0.846941	0.838377
Time taken	0.546s	0.219s	0.125s	0.078s
% Reduction of Max	n/a	88.07%	93.19%	95.75%
% Reduction in Unavailability	n/a	0.037%	1.286%	2.284%

It seems that there is definitely scope for using this feature, although it may vary in its effectiveness depending on the model and the cut sets that it produces. In any case where greater accuracy is required for a particular

problem it may be possible to quickly reduce the search space using the pruned cut sets and then refine the search with unpruned sets.

One potential problem with using this technique would occur when the maximum cut set order is set to below the maximum number of components that are permitted in parallel in a redundant configuration. In this situation, pruning the cut sets would make the selection of higher parallel redundancies impossible because there would be no reliability benefit (due to the pruning) for doing so, yet still an increased cost.

3.4.4 FMEA Catalogue

Consider the following example involving 7 non-redundant cut sets: A.B, B.C, B.E, B.F.G, C.E, C.F.G, D. Now assume a new cut set A.C.D is being added and has to be checked for redundancy.

With a naive algorithm it would be necessary to compare the new cut set for redundancy against the complete non-redundant set, totalling 7 comparisons, before discovering that it is redundant because of cut set D and the law of absorption.

However, if the cut sets are arranged in a cataloguing data structure, then the number of redundancy checks can be reduced by skipping checks on cut sets that could not possibly cause redundancy.

The catalogue structure for each top event has 3 levels. The first level is a list of all the basic events that contribute to the top event. Below this each basic event has a list of the sizes of all the cut sets that contain that basic event.

Note that if there is a single order cut set then there will only be one entry in this list as all higher sized cut sets for this basic event would be redundant due to the law of absorption.

Finally in the third layer is a list of all the cut sets that are of the given size. These levels can be seen in Figure 15 where, for example, the top event has cut sets that contain the basic event B. Of these cut sets there are 3 that are of order 2 and 1 of order 3.

All of the cut sets that are stored in the data structure have their basic events sorted in order of their unique identifier. That way, a combination of basic events always appears the same, e.g. A.B.C always appears in that order and never B.A.C or C.A.B or B.C.A, etc.

Within the final level, the cut sets are separated into 2 groups: one where the basic event of the first level is the first basic event in the sorted cut set and the other group where the basic event appears in subsequent positions in the cut set.

This enables a reduction in the number of checks required, for example, when checking cut set A.C.F against cut sets B.C, C.E, and C.F.G in the basic event C column. As basic event C appears in the second position in A.C.F (and not the first) we only need to check against cut sets that start with basic event C, thus avoiding the check against B.C.

Similarly, when we are checking only the cut sets that start with the basic event of the column, we can also limit the checks to cut sets that are of an order less than or equal to the new cut set as the new cut set would have to contain at least one basic event not contained in the higher order cut sets, thus making redundancy impossible in those cut sets.

In Figure 15 the non-redundant cut sets from the naïve example are catalogued in the FMEA data structure. When doing a redundancy check on the new cut set (A.C.D), we consider each of the basic events of the new cut set in turn: A, C, and D.

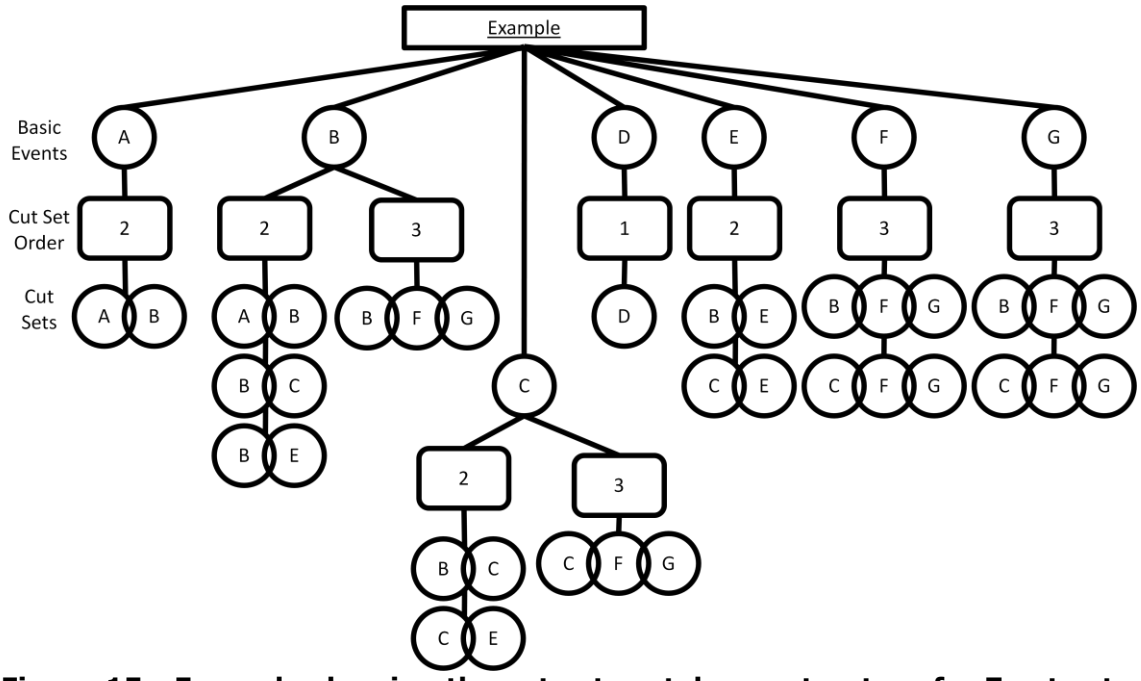

Figure 15 - Example showing the cut sets catalogue structure for 7 cut sets (A.B, B.C, B.E, B.F.G, C.E, C.F.G, D)

First we check the A column where there is one cut set (A.B) which does not cause redundancy. The basic events in a cut set are ordered by their unique identifier and within each column of cut sets they are separated into two groups, those that have the basic event of the column as the first event in the cut set and those that contain the basic event but not as the first event.

As when we check the C column we are not checking the first event in the cut set, we can skip the group of cut sets that do not start with that event as those cut sets contain at least one event that does not appear in the new cut set and so cannot cause redundancy. This means that even though an existing cut set B.C will be indexed in both the B and the C lists it will only be checked in the first instance when looking in the B column.

That just leaves one order 2 cut set to check (C.E) and one order 3 cut set (C.F.G), neither of which cause redundancy. Finally we check the D column where there is one order 1 cut set (D) and this causes redundancy in the

new cut set through the law of absorption. A total of 4 redundancy checks were required as opposed to the 7 checks required by the naïve approach.

Using the FMEA catalogue for checking redundancy significantly reduces the number of checks required and will never perform worse than the naïve approach. The performance benefit would also be more marked as the number of cut sets involved increases.

In addition to the performance increase in cut set minimisation, using the FMEA catalogue also maintains the cut sets in a data structure that directly relates to the FMEA information contained within them.

This is achieved because the structure (as in Figure 15) contains all the cut sets for each failure mode for a particular fault tree. The only additional requirement is to link the failure modes directly to the multiple system failures that they cause.

Figure 16 shows such an example where there are 3 basic events that directly (or indirectly) cause 2 top events. Using the structure it is possible to see that failure modes B and C contribute to both top events but that A only contributes to top event 1. It can also be seen that B is a single point of failure in top event 2 but that it only contributes to top event 1 as it must occur either with A or C in order to cause the top event.

As this information is inherent in the catalogue structure, it is not necessary to perform extra passes of the cut set data in order to collate the FMEA. A naïve approach would require this extra step.

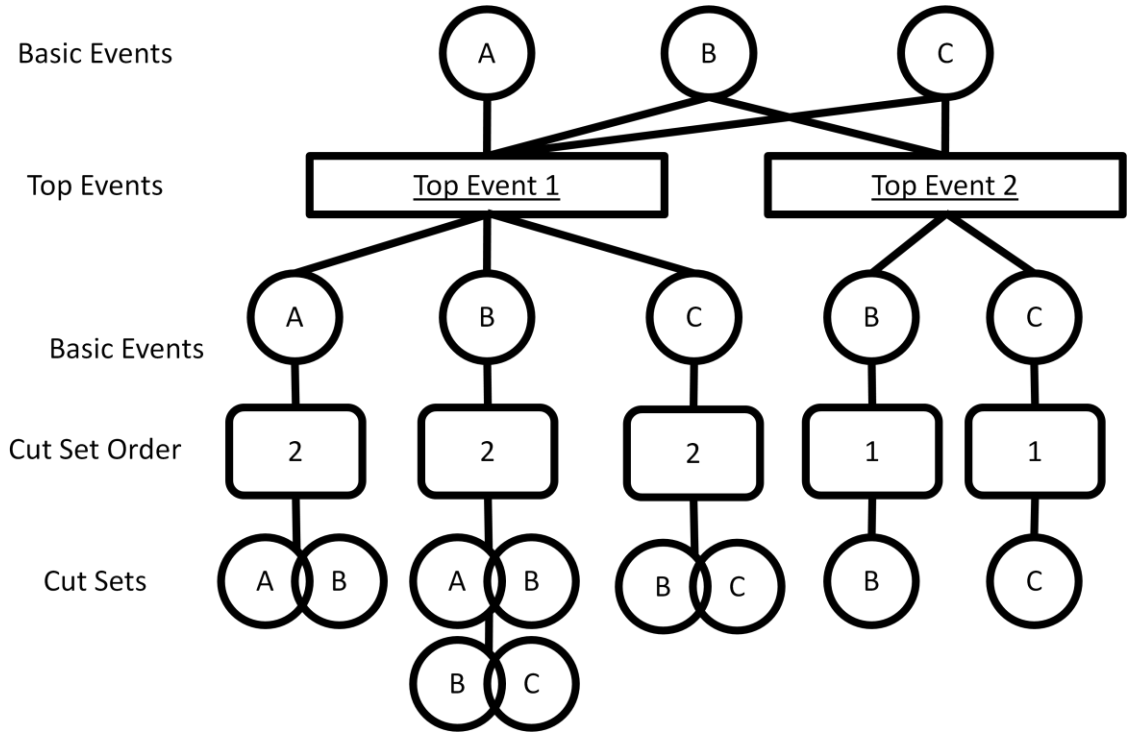

Figure 16 - Example showing complete FMEA structure for 3 basic events causing 2 top events as part of different cut sets.

Table 9 shows that using the FMEA structure reduces the time required to analyse the large test fault tree by 6.3 times.

Table 9 - Comparison of times taken to minimise cut sets of a large fault tree using the FMEA structure for storing cut sets

	FMEA Structure	Standard	% Reduction
Time taken	1.836s	11.55s	84.108%

This increase in performance is again achieved by reducing the number of redundancy checks required. Table 10 shows that the FMEA cataloguing structure resulted in 15.9 times fewer redundancy checks, when analysing the test fault tree.

Table 10 - Comparison of number of redundancy checks needed to minimise cut sets of a large fault tree using the FMEA structure for storing cut sets

	FMEA Structure	Standard	% Reduction
Number of redundancy checks	38140	605395	93.700%

3.4.5 AnalySynth

AnalySynth was developed to specifically improve the performance of the RAP benchmark analysis, though it has a wider application. Despite the conceptual simplicity of the RAP model, its fault tree (see also section 3.5) is very large and took a long time to traverse with the techniques described so far.

In the case of the RAP problem, there is a series of parallel components logically conjoined with an AND gate. There are always 14 series units, but the number of parallel components in each unit can vary up to 4. In the worst case scenario there are 14 units in series, each with 4 components in parallel.

As the synthesis algorithm traverses it encounters the first unit. There are 4 components in parallel in this unit so that creates 4 branches. Each of the branches connects to the next unit in the series, which also has 4 components in parallel. The 4 branches that are generated from this unit are connected to the bottom of each of the 4 branches from the original unit; now there are 16 branches.

This is repeated at each of the 14 units until finally there are 4^{14} = 268,435,456 branches. Merely traversing a fault tree of this size takes a long time and at each gate during analysis redundancy checks would need to be repeated.

To solve this problem, AnalySynth was developed to combine the action of Analysis and Synthesis. The basic principle behind it is that the CFT for each unit could be minimised using the MICSUP algorithm before it is synthesised. The cut sets can then be converted back into a mini-fault tree that is of the sum of products form.

In the case of the RAP problem, the branching AND gate in each unit is terminated by component failure modes. This leaves only a single branch that propagates to the next unit. Instead of the number of branches growing exponentially with each unit, the number of branches remains constant and the resultant fault tree can be traversed quickly.

Another benefit of this technique is that analysing small fault trees is disproportionately easier than analysing large fault trees. By performing the analysis as soon as possible on the small CFTs when the analysis is easier, it reduces the effort required as the tree grows larger.

In order to test this, a model was created to represent the worst case RAP benchmark. All 14 series units had 4 components in parallel. It was tested with both AnalySynth activated and with the standard separate synthesis and analysis algorithms.

Table 11 shows the different timings for the Synthesis and Analysis phases and the combined time of both phases. With the standard algorithms the total time taken to synthesise and analyse the model was completely impractical in an optimisation setting. With AnalySynth activated the total time taken is a staggering 236086 times less.

Table 11 - Comparison of times taken for the synthesis and analysis phases of the Fyffe RAP model using the Analysynth algorithm

	AnalySynth	Standard	% Reduction
Synthesis	0.228s	56729.97s	99.9996%
Analysis	0.027s	3472.30s	99.9992%
Total	0.255s	60202.26s	99.9996%

There is a reduction in both the synthesis phase where fewer branches are created, and the analysis phase because the resultant fault tree will be mostly minimal already.

Further testing of this technique is necessary to establish a general usefulness in other models. It is postulated that it would yield particular benefit in

models with deep hierarchies as the resultant tree for each top level subsystem would be the minimised combination of all the sub trees.

Another area to improve is that if there was a single, unique, working copy of each CFT then the excessive branching would be tamed because each branch would link back to the single copy. This remains as a future development, though it is thought that even if this was successful there would still remain a useful application of AnalySynth, particularly, as mentioned, on highly hierarchical models.

3.4.6 Performance enhancement summary

Several performance enhancements have been implemented to reduce the computational effort required for HiP-HOPS safety analysis. These include fault tree contraction, fault tree modularisation, cut set pruning, AnalySynth, and FMEA cataloguing.

The first three methods are implementations of existing algorithms from the literature and general practice. AnalySynth and the FMEA cataloguing algorithm are both novel approaches and appear to offer increased performance.

Table 12 shows the performance benefit of applying fault tree contraction, fault tree modularisation, and FMEA cataloguing on a large fault tree. Cut set pruning generally results in a reduction in probability calculation accuracy so it is not included in this test. AnalySynth is also not included as it is designed to resolve a particular problem not present in this test model.

Table 12 - Comparison of times taken for synthesis and analysis phases for a large fault tree with contraction, modularisation, and FMEA cataloguing on versus off.

	All enhancements	Standard	% Reduction
Time taken	2.56s	8301.19s	99.969%

The result for this test model is that the combined synthesis and analysis phases complete over 3200 times quicker with the performance enhancements in place. This greatly improves the feasibility of using HiP-HOPS in an automatic optimisation application.

It is worth noting that the long term trend in computing advances has been for a doubling in performance roughly every two years. This trend only increases the feasibility of computationally expensive tasks such as optimisation, although clearly it is not something that should be relied on to the exclusion of algorithmic improvements.

3.5 Modelling the benchmark system with HiP-HOPS

Previous approaches for solving the benchmark RAP problem used an RBD model to calculate the reliability. HiP-HOPS can be used to replace the use of RBDs as a means of calculating reliability, as discussed in the preceding chapter.

In the first instance, this hypothesis will be tested on the benchmark RAP model. For this, it is necessary to have a HiP-HOPS model of the benchmark system and the SimulationX modelling environment was used to create this.

Each of the subsystems has a single input and output port and the output of each component is connected to the input of the following subsystem in a series configuration. This is illustrated in Figure 17 which is a modified screen capture from SimulationX where the middle subsystems (3 – 12) have been omitted to fit.

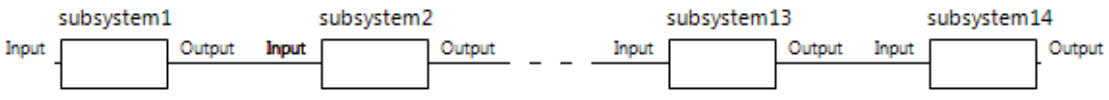

Figure 17 - HiP-HOPS model template for Fyffe *et al.* (1968) RAP problem

The failure model of each alternative component comprises one output deviation occurring at the output port. In each case the output deviation is an omission and is caused by either a corresponding omission at the input port or an internal failure, as shown in Figure 18.

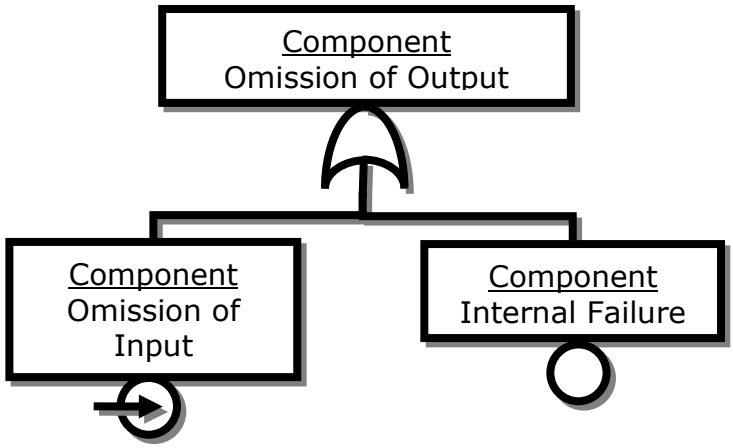

Figure 18 - CFT for each of the alternative implementations

Table 13 shows the cost and weight for each of the alternative components for the 14 subsystems, duplicated from (FYFFE, D E et al., 1968). It also shows the reliability probability for the internal failure modes.

Table 13 - Table of alternative implementation attributes for the Fyffe *et al.* (1968) RAP problem

	Alternative components											
	1			2			3			4		
subsystem	r	c	w	r	c	w	r	c	w	r	c	w
1	0.90	1	3	0.93	1	4	0.91	2	2	0.95	2	5
2	0.95	2	8	0.94	1	10	0.93	1	9	-	-	-
3	0.85	2	7	0.90	3	5	0.87	1	6	0.92	4	4
4	0.83	3	5	0.87	4	6	0.85	5	4	-	-	-
5	0.94	2	4	0.93	2	3	0.95	3	5	-	-	-
6	0.99	3	5	0.98	3	1	0.97	2	5	0.96	2	4
7	0.91	4	7	0.92	4	8	0.94	5	9	-	-	-
8	0.81	3	4	0.90	5	7	0.91	6	6	-	-	-
9	0.97	2	8	0.99	3	9	0.96	4	7	0.91	3	8
10	0.83	4	6	0.85	4	5	0.90	5	6	-	-	-
11	0.94	3	5	0.95	4	6	0.96	5	6	-	-	-
12	0.79	2	4	0.82	3	5	0.85	4	6	0.90	5	7
13	0.98	2	5	0.99	3	5	0.97	2	6	-	-	-
14	0.90	4	6	0.92	4	7	0.95	5	6	0.99	6	9

4. Method for Architectural Optimisation using Genetic Algorithms and Automatic Safety Analysis

In chapter 2, two Multiple Objective Evolutionary Algorithms (MOEAs), PESA-II and NSGA-II, were selected for further investigation. In this chapter, the general MOEA approach is described, with specific details of the different GAs where appropriate.

In addition to the two MOEAs, Coit and Smith's (1996b) penalty-based GA will also be implemented, using HiP-HOPS to calculate the solution reliability. The reason for including this is that it is a successful use of a GA for solving the benchmark problem. The principle GA framework remains the same for the two MOEAs and the penalty-based GA, with only the details of the selection mechanism being significantly different. This will permit a useful comparison of the techniques.

The structure of this chapter follows the pattern of a GA, with subsections detailing each of the parts.

1) Devise an encoding for the solutions.

2) Randomly initialise population of encodings.

3) Select some 'fit' individuals from the population for breeding

4) Generate child solutions by applying genetic operators, mutation and recombination, to the selected parent solutions.

5) If population size exceeds maximum then select 'unfit' individuals for culling from the population.

6) If maximum generations not reached then return to step 3.

4.1 Solution encodings

The operators of a GA act on an encoding of the potential solution and not the actual solution itself. One of the keys to the successful application of a GA to a problem is the selection of an encoding mechanism. The encoding mechanism is required to store the variable information necessary to reconstruct and configure a potential solution for evaluation and comparison to other potential solutions.

Two different encoding systems will be considered; one a fixed length string containing integer numbers and the other a tree-based encoding structure.

4.1.1 Fixed length real number string encoding

Coit and Smith (1996b) used a fixed length encoding for their penalty-based GA. The complete encoding for the system is made up of smaller fixed length encodings, one for each of the subsystems that can be altered in the system.

Each subsystem can consist of up to 5 component slots in parallel, with each of the available 'slots' able to be filled by one of 3 or 4 functionally equivalent alternative components. For each subsystem the fixed length encoding is 5 digits long. Each of the digits in the encoding can hold the value 1, 2, 3, or 4 to represent the specific alternative component being used in that 'slot'.

It can also be set to a value to represent the 'slot' being empty; Coit and Smith's version used a value that was equal to the number of alternative components for a subsystem plus 1, so if there were 4 different alternative components the value to represent an empty slot would be 5. Coit and Smith do not explain the reason for this choice; one possible reason for this could be than when sorting the encoding in ascending order the empty slots would always appear at the end of the string (the significance of ordering

the encoding is explained below). In the experiments for this thesis the value 0 was used instead as this better represents the concept of an empty or unused slot.

The encoding used in this experiment differs from the encoding used by Coit and Smith (1996b) in another way. In Coit and Smith's 1996b paper, before assigning the identifying integers to alternative components, the components were ordered by their reliability value, such that component 1 is the most reliable, component 2 the second most reliable and so forth. Again the reason for this is not given by Coit and Smith.

For this thesis this was not done because it is not a requirement (either practical or conceptual) of the GA and, whilst it makes direct comparison with Coit and Smith's (1996b) results easier, the majority of examples in the literature that use the RAP benchmark do not do this and so comparison across the board becomes more difficult.

Where, in this thesis, the encodings of solutions found by Coit and Smith (1996b) are compared with the rest of the literature examples they have been converted to the common form.

The use of real integer values in the encoding has been shown to be more efficient (ANTONISSE, J, 1989) in combinatorial optimisation applications than the traditional use of binary encodings using only 1 and 0 values.

The encoding for the subsystem is then ordered by the ordinal value of the integers. This reduces the size of the search space by removing solution duplicates with different encodings. For example, a subsystem with a configuration of component 1 in parallel with component 2 and component 3 could be encoded in the following 6 combinations: 123, 132, 213, 231, 312, 321; all of which decode to the same solution.

By ordering the encoding digits they all resolve to the same encoding: 123.

To make it easier to see the components being used in the encodings the empty slots value 0 always appear at the end of the ordered sequence; for example: 12300.

In order to evaluate the solution described by the encoding it is necessary to configure the HiP-HOPS model with the encoding. The HiP-HOPS model for the fixed length encoding experiments consists of 14 subsystems in a series configuration. Each of the subsystems has either 3 or 4 alternative implementations (depending on the subsystems).

When the model is configured, the encoding is split in to its separate alleles, one for each subsystem, and this part-encoding is used to configure the individual subsystems.

The part-encoding is stripped of zeros so, for example, 12300 becomes 123. The remaining part-encoding now fully describes the subsystem with the number of digits giving the number of parallel components in the subsystem and the digits themselves identifying the implementation for each of the sub components.

So to configure the subsystem the digits are counted. With our example, encoding 123 there are three digits so three components are generated in the subsystem.

The inputs and outputs are connected to the corresponding input and output of the subsystem with the connection between the outputs having a logical AND to produce the parallel redundancy effect.

Finally each of the components has its implementation set to the value of the corresponding digit from the encoding; in our example the first component is set to implementation 1, the second to implementation 2, and the third to implementation 3. This example can be seen in Figure 19.

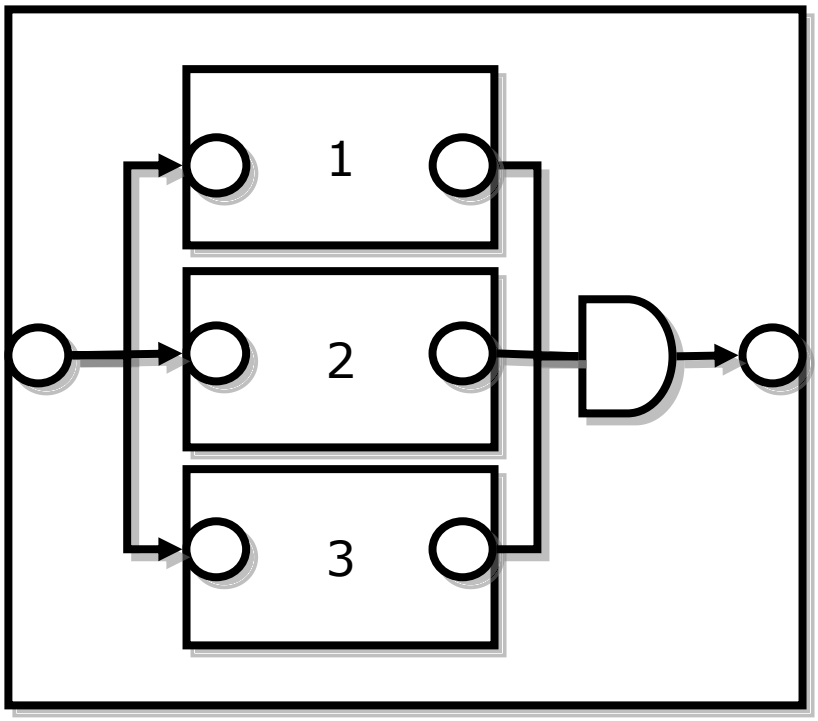

Figure 19 Example configuration of a single subsystem with the part-encoding 12300.

4.1.2 Tree encoding

The fixed length encoding is easy to implement and its use has been well documented in many examples with Coit and Smith's use just being one. It does however have a drawback: its lack of flexibility when applied to arbitrary system hierarchies.

Its demonstrated use with redundancy allocation problems has been for systems that are decomposed to just a single level and where the failure model is that of the reliability block diagram, restricting it to use with systems that can be represented in this way as series-parallel topologies.

A tree-based encoding frees us from these restrictions, thus allowing the optimisation of models with both variable levels of hierarchy and variable

topologies. The use of tree encodings, however, brings complications ensuring the viability of encoding solutions that are generated via recombination. These will be discussed in section 4.2.2.2. The tree-based encoding proposed here is similar to that described by Kumar *et al.* (2009).

Creating a HiP-HOPS model for use with a tree encoding involves more effort than the simpler model for the fixed length encoding as the user must specify the alternative hierarchies. This however is not necessarily a problem as hierarchy is typically employed in design as means of managing large scale and complexity. An additional benefit is that, once hierarchical modelling has been done, one can then optimise these alternative hierarchies in addition to the alternative components to be used in those hierarchies.

Figure 20 shows how alternative implementations of the subsystems in the benchmark system could be represented in a HiP-HOPS model for use with a tree encoding.

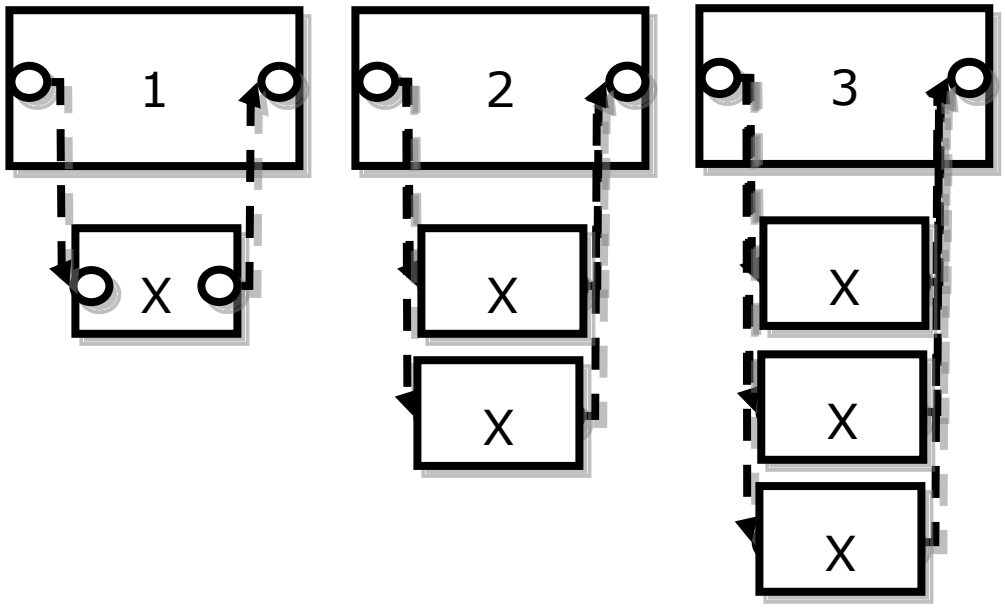

Figure 20 Alternative hierarchies for a subsystem in the Fyffe *et al.* benchmark problem.

The first implementation of the top level subsystem contains just one component. The second implementation contains two components connected with AND logic (not shown) to the output of the subsystem. The third implementation contains 3 components and so on.

Although in this example the hierarchy is simple in order to allow comparison to the benchmark problem, there is no reason why the alternative architectures could not be anything the designer chooses; different redundancy architectures for example. This allows a great deal of flexibility in the system to be optimised.

In addition to alternatives to the hierarchy being represented, it is still possible to optimise the choice of components in that hierarchy through the selection of alternative implementations of the leaf node components (shown in Figure 20 by an X). Figure 21 shows a tree encoding example that would configure a HiP-HOPS model to the equivalent of the fixed length encoding 12300|44000|11200.

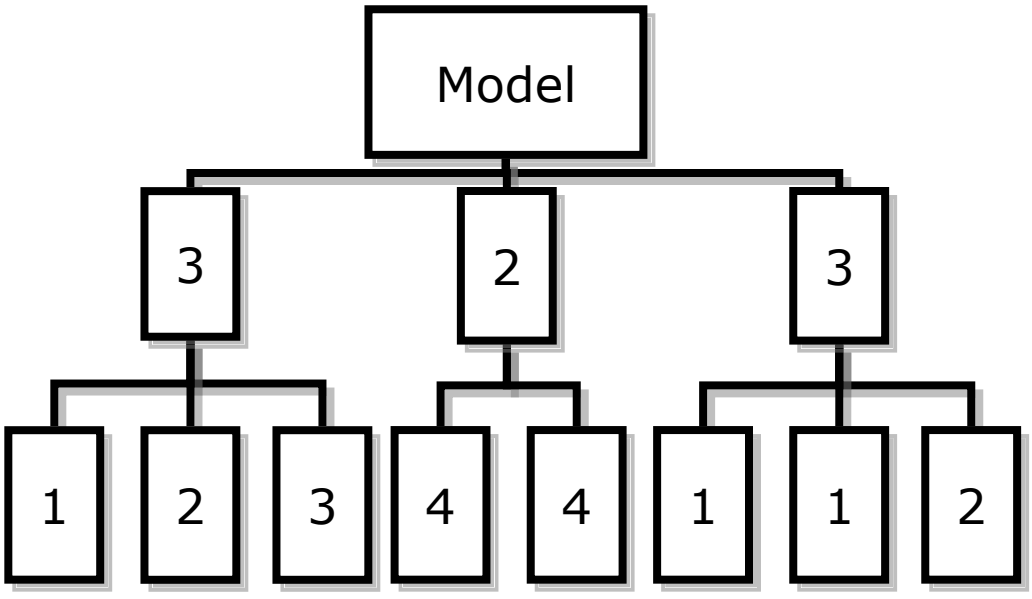

Figure 21 Tree encoding example for a small three subsystem model that has the equivalent fixed length encoding 12300|44000|11200.

To illustrate this with an example we will use the equivalent encoding of the 12300 fixed length encoding to configure a subsystem in a HiP-HOPS model.

Figure 22 shows a tree encoding in the left side of the panel. Each node describes the implementation of a component or subsystem in the model. In order to configure the model the tree encoding is traversed in a depth first traversal, and when a node is encountered, its corresponding subsystem/component has its implementation set to the value indicated by the encoding node.

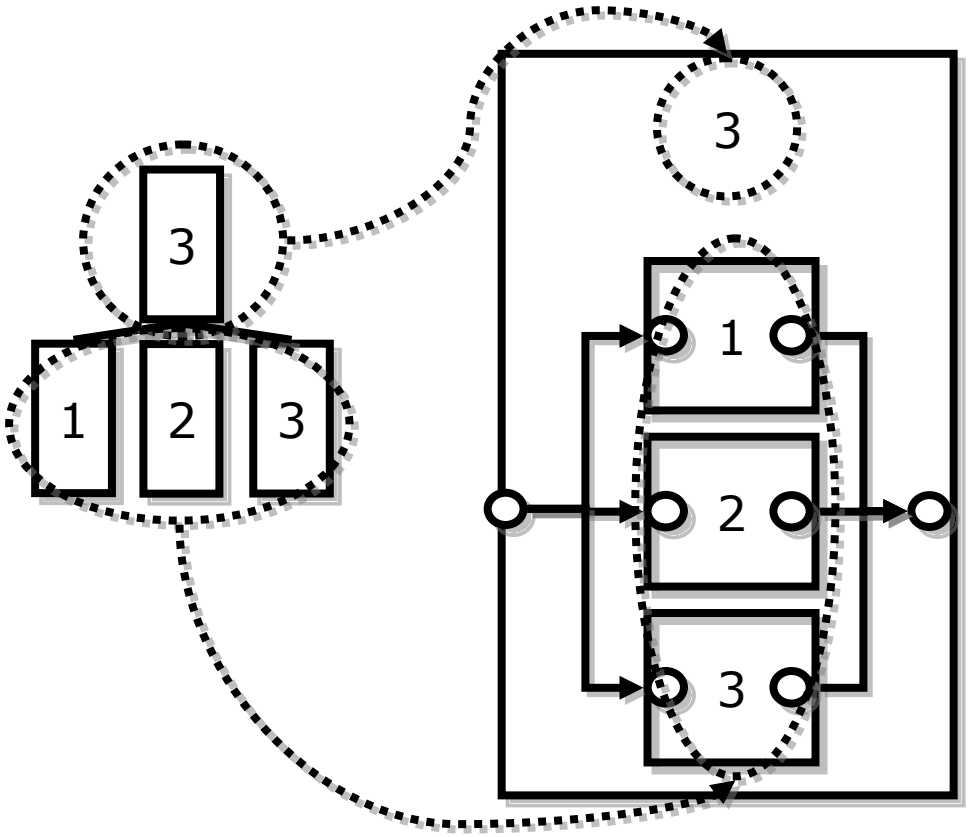

Figure 22 Example tree encoding shown on the left with the configured subsystem on the right.

In the example the top encoding node is 3 so the implementation of the top level subsystem, which the node corresponds to, is set to implementation 3.

Implementation 3 of the subsystem has a sub-architecture that contains 3 components connected by AND logic to the output of the subsystem.

Next the sub nodes of the tree encoding are visited, setting the implementation of the subsystem's sub-components.

This achieves the same as the equivalent fixed length encoding with value 12300 and for this benchmark problem does not add anything. However, it should be seen how, as the tree encoding structure can work to any arbitrary depth, it allows unlimited flexibility in the kind of models it can represent and consequently the complexity of the models that can be optimised.

4.2 Fill child population

For each of the generations of the GA it is necessary to generate a new population of child solutions to try to improve the general population. The mechanism for producing this child population is different during the initialisation phase to that of a normal, subsequent, generation.

Both procedures are described below and generate a child population that has a number of encodings specified by the child population limit variable. For example if the child population limit is set to 150, then 150 new encodings are created each generation, as well as during initialisation.

4.2.1 Random initialization of new population

When the GA is being initialized the population of encodings is generated completely at random. The differences between randomly generating a fixed length and a tree encoding are described below.

4.2.1.1 Fixed length encoding

In order to randomly generate a fixed length encoding for the redundancy allocation problem the HiP-HOPS model needs to provide two pieces of information.

The first is the maximum number of components that are permitted in parallel for each subsystem as this determines the length of each allele (and combined with the number of alleles, the entire encoding length).

The second is the number of alternative implementations there are for each component as this determines the range of values that can be used in the encoding digits.

For the benchmark problem the maximum number of components permitted in parallel is 5 and the number of subsystems in series is 14. Therefore the length of the encoding string is 5 x 14 = 70 with five digits per subsystem allele.

The subsystem alleles are generated by selecting an integer number to fill each of the slots in the allele. For each of the slots the number has a 50% chance of being non-zero, where zero denotes an empty slot. Where the number is non-zero, it is randomly selected in the range of 1 to the number of alternative implementations for the component.

Once the allele has been filled, it is sorted in ascending order but with all the empty slots at the end of the allele as explained before.

For an example, consider one subsystem where the maximum number of parallel components is 5 and the number of alternative implementations is 4. For the first of the five allele slots a virtual coin toss is made and the result is heads so the slot is set to zero.

For the second slot the result of the coin toss is tails so the slot is set to a number randomly chosen between 1 and 4, the number of alternative implementations. In this case the result was 3.

For the third slot the coin toss is tails again and the randomly selected implementation is 1. The last two slots are filled with zeros following two virtual coin tosses resulting in heads.

This gives an allele with the slots filled as follows: 03100. This is then sorted giving the completely generated allele for that subsystem of 13000, two components in parallel with the implementations 1 and 3 respectively and 3 empty slots.

Finally, the alleles from all the subsystems are joined together to make the complete fixed length encoding.

4.2.1.2 Tree encoding

In some respects generating a tree encoding for a model is easier as you don't need to store and retrieve the maximum number of parallel components; this information is inherent in the topology of the model and the alternative architectures.

There is one node in the tree for each component/subsystem that can be optimised and the tree is constructed randomly in the following way.

A depth first traversal of the model is performed, and where a subsystem is visited, a node is created with the fully-qualified name of the subsystem. Also a randomly selected implementation is chosen from the alternatives for that subsystem.

The implementation for that subsystem is then set according to the selection and the substructure of that subsystem is traversed. Any nodes created whilst in that substructure are added as child nodes to the encoding.

For example, consider a single subsystem called Subsystem1 that has 5 different implementations. One of the implementations has a substructure with just one component in it; the second has two subcomponents; the third three; and so on.

In a depth first traversal an encoding node is created for this top level subsystem and its name is set to Subsystem1. A number between 1 and the number of alternative implementations is selected for that subsystem. In this case the number is 2 and this value is set in the encoding node and the implementation for the subsystem is set to implementation 2.

Implementation 2 for this subsystem has two sub-components, named Component1 and Component2, each with 4 alternative implementations. The components are visited in turn creating two new encoding nodes called Subsystem1.Component1 and Subsystem1.Component2, the fully qualified names for the components.

Their implementations are randomly selected as 3 and 1 and they are added as child nodes to the encoding node Subsystem1. As they have no substructure the traversal finishes resulting in the encoding tree shown in Figure 23. This encodes the same subsystem as the above example for the fixed length encoding.

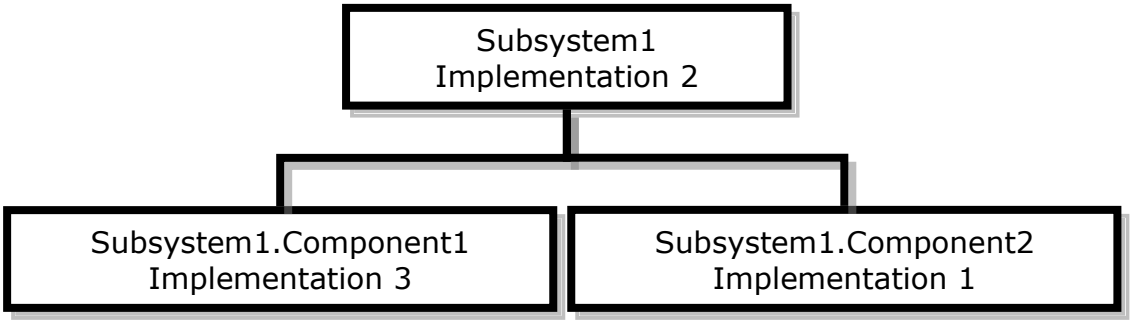

Figure 23 Example randomly generated tree encoding for a single subsystem.

4.2.2 Generate child population from main population

During each normal generation of the GA (not the initialization phase described above) the child population is not purely randomly generated, but rather through the action of three operations: selection, recombination and mutation. These are explained below.

4.2.2.1 Selection algorithms

The operators, mutation and recombination, directly alter the encoding representation of the potential solutions, but before they can be applied it is first necessary to select individuals in the population to act as the operands.

It is necessary to guide the optimisation search and so the use of a purely random selection mechanism is not desirable. Evolutionary algorithms are modelled on biological evolution and the selection mechanisms mimic natural selection: survival of the fittest. In any given population the individuals best adapted to survive in the environment are most likely to survive long enough to reproduce, passing their successful genetic encoding to the next and subsequent generations.

The selection mechanism in a GA uses the same concept; based on the evaluated fitness of the potential solution it represents, an encoding is more likely to be chosen for genetic operation if it has a higher fitness.

A key thing to note at this point is that the better (more fit) individuals are more likely, but not exclusively likely to be chosen. It is possible that the optimum solution is found as a variation of one of the worst (least fit) individuals in the population at the time.

The three selection algorithms for the different GAs being compared are given below; they are for the Coit and Smith's penalty based GA (1996b), the PESA-II GA, and the NSGA-II algorithm.

Penalty-based fitness

In the selection algorithm used by Coit and Smith (1996b), selection is based on establishing a penalised fitness which is the defining characteristic of the approach. The approach is explained below.

As stated before, most real world optimisation problems have multiple objectives. The general case of the RAP benchmark has three objectives: maximize reliability, minimise cost, and minimize weight. This presents a problem for the traditional GA as it is designed to optimise a single objective.

There are several strategies for overcoming this restriction by combining the different objectives in to one hybrid objective, perhaps a weighted sum or product of the objectives. This is a complicated task because of difficulties in determining how much importance to assign each objective for the weightings, or if the magnitude of the values of the objectives is hugely different should they be normalized to compensate.

Another way of crafting multiple objectives as a single objective is to optimise one of the objectives and define the remaining objectives as constraints that should not be violated. For example: maximize reliability whilst ensuring that the solutions do not cost more than a threshold value and do not weigh more than a corresponding weight threshold.

There are a number of methods for handling constraints. The so called death penalty removes infeasible (constraint violating) solutions from the population. Another approach is to either ensure that the genetic operators always produce feasible solutions or applying repair functions to change infeasible solutions to feasible ones.

The death penalty is too blunt a tool, however, as the most efficient route to a feasible optimum solution can often be through an infeasible region of the search space. Equally it is important to end up with a feasible solution so allowing unrestricted wanderings in the infeasible regions is also undesirable.

Coit and Smith (1996a) describe a dynamic adaptive penalty scheme which applies a penalty for constraint violations that becomes increasingly strict as the generations of the GA increase.

The formula for calculating the penalised fitness is shown below:

It shows that the penalised fitness of a solution F_{ip} is given by the reliability of the solution R_i less the penalty for constraints violations. and are the magnitude of the weight and cost constraint violations. For example, if the cost constraint is maximum 130 and the solution cost is 136 then = 6.

The portion of the equation ensures that if the solution with the highest reliability found so far (feasible or infeasible, given by) is a feasible solution (is the feasible solution with the highest reliability found so far) then no penalty will be applied.

The k exponent is a severity parameter preset to 2 by Coit and Smith.

The last part of the equation is the near feasibility threshold (NFT). The NFT marks an infeasible area of the search space that is close to the feasible region. Solutions in this near feasible region have only a minor penalty applied whereas solutions that lie outside of this region are more heavily penalised.

Below the NFT formula is given:

$$\frac{\quad}{\quad}$$

is given as an upper bound for the threshold and is 100 for the cost constraint and equal to a third of the weight constraint. is a constant to ensure that the entire infeasible region is considered and is chosen to ensure that the NFT is not contracted too quickly by the generation parameter . is set to 0.04 for both cost and weight constraints by Coit and Smith.

In summary: An NFT is defined whose size depends on the current generation of the GA. In the early stages of the search, the NFT is relatively large allowing free search of the infeasible region. As the GA progresses the size of the NFT decreases, with solutions outside of the NFT increasingly penalised. Hence infeasible solutions considered fit at the beginning of the search will be considered relatively unfit at the end.

Penalty-based fitness selection

The first selection algorithm used in this work is the same as Coit and Smith used from Tate and Smith's 1994 paper.

The individuals in the population are ranked by their (penalised) fitness and then a number is randomly chosen from the range 1 to $\sqrt{P}$, where P is the size of the population. The individual with the rank closest to the square of that number is selected. Tate and Smith (1994) used this selection method

as the degree of fitness bias could be dynamically altered by varying the power of the root of P, though this feature was not used by Coit and Smith (1996).

In the example below is a population of 7 individuals.

Fitness
12
16
27
13
11
29
9

It is first ordered by the individuals' fitness.

Rank	Fitness
1	9
2	11
3	12
4	13
5	16
6	27
7	29

Then a random number is chosen in the range 1 to √7, in this example 1.922, and then squared, giving 3.692. This value is rounded to give the index of the individual being selected, as highlighted below the individual with rank 4.

Rank	Fitness
1	9
2	11
3	12
4	13
5	16
6	27
7	29

Pareto-based selection

The penalty based approach reduces the multiple objectives of the benchmark problem to a single objective that can be maximised. The general problem, however, is still a multiple objective one, and Pareto-based MOEAs are designed to consider all objectives simultaneously.

It is desirable for the returned Pareto set to be an evenly spread set of solutions to covers the whole of the Pareto-optimal set. Thus it is important to minimize the crowding of solutions in particular areas of the search space by encouraging search in relatively less crowded regions of the frontier.

Figure 24 shows a Pareto frontier with gaps representing areas of low crowding highlighted. Search should be directed in these regions as it represents an area of unexplored search space. Both of the Pareto-based selection algorithms used in this work, PESA-II and NSGA-II use the concept of crowding to select between individuals to perform genetic operations on. The differences in the way that the crowding is calculated are described for each of them below.

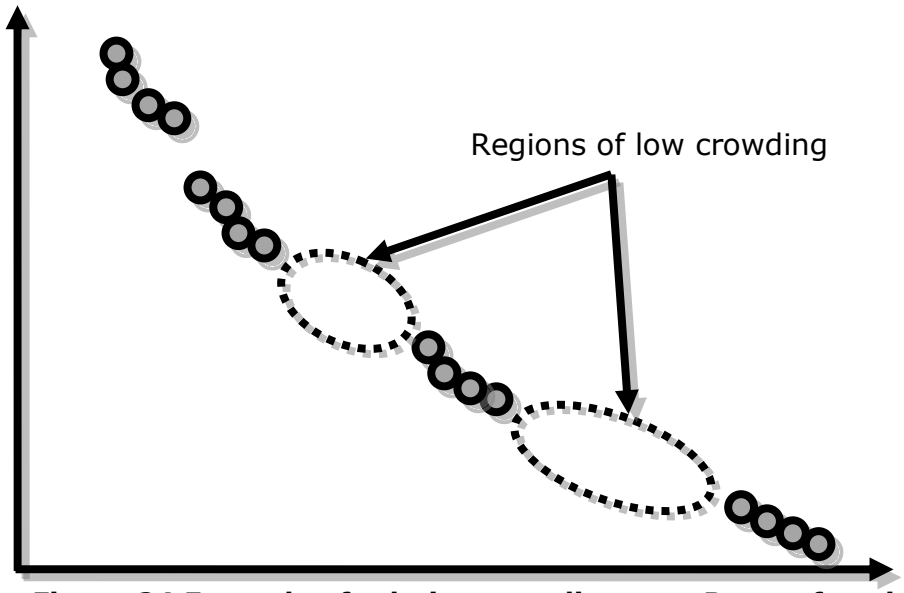

Figure 24 Example of relative crowding on a Pareto frontier.

Pareto envelope-based selection (PESA-II)

The Pareto envelope-based selection algorithm (CORNE, D W et al., 2001) creates a hypergrid of a specified size. This hypergrid divides up the normalized solution space and all of the individuals then fall into one of the hyperboxes that make up the hypergrid. The number of individuals that populate a hyperbox gives the crowding density for that hyperbox.

As the hypergrid divides up normalized solution space it is necessary to obtain a normalized value for each of the objectives. In order to minimize the overhead for such a step the maximum and minimum value for each objective in the population was maintained and the normalized values for the individuals in the population were only recalculated when the maximum or minimum value changed, invalidating the existing normals. The normal value for each objective was calculated according to the following formula:

For example an individual with a raw cost of 535 in a population with a maximum cost of 1300 and minimum cost of 400 has the following normalized cost:

And the individual's raw unavailability is 0.01 in a population with unavailability that ranges from 0.001 to 0.28 giving a normalized unavailability of:

The normalization calculation scales the objective values into a range between 0 and 1, which can then be scaled into the hypergrid dimensions by multiplying by the grid size and rounding the figure down to give a whole number.

So for the hyperbox for the individual in the previous example, given a hypergrid size of 4, has the coordinates:

Which rounded down to give a whole number gives the hyperbox coordinate (1,1).

Figure 25 shows a solution space that has been divided into a hypergrid of size 4. The binary tournament selection method is used to select an individual. First two hyperboxes are randomly selected from the hypergrid. Then the hyperbox with the lowest crowding value is selected and the other ignored. For example if hyperbox A and hyperbox B are chosen from the example then hyperbox A would be selected as it has a crowding value of 2, which is less than hyperbox B's value of 4 and thus deemed to be in a less explored region of the search space.

Once a hyperbox is selected, one individual from the hyperbox is randomly chosen as the operand. One disadvantage of using the hypergrid mechanism to establish crowding is that it requires an additional parameter for the algorithm to work, the hypergrid size, and the choice of that parameter seems non-trivial. The effect of altering the hypergrid size will be investigated later.

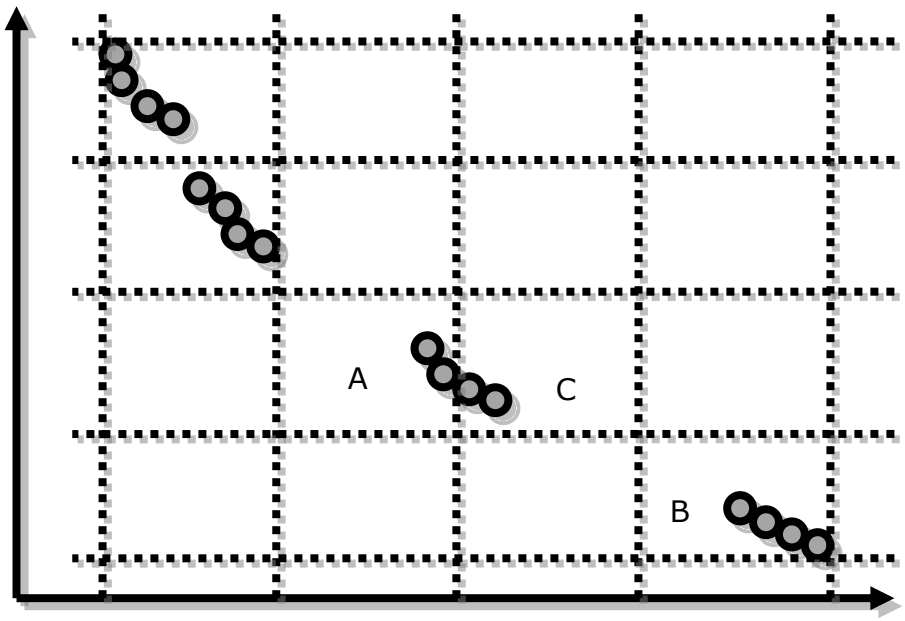

Figure 25 Solution space divided by a hypergrid with size 4.

In addition to using the hypergrid crowding value for selection a further step was added to allow the search to be focused inside a constrained area. PESA-II did not make use of constraints.

Figure 26 shows how the use of a lower and upper bound to objective values can divide the search space into a feasible and infeasible region. Though it is desirable to focus the search on feasible solutions it is important to not be too strict when applying the constraints as the most efficient route to an optimum feasible solution can often be through an infeasible region of the search space.

In order to add feasibility as a criterion during selection each individual's feasibility was calculated as a true or false value, feasible or infeasible. If any of the constraints in any of the objectives was violated then the individual was infeasible.

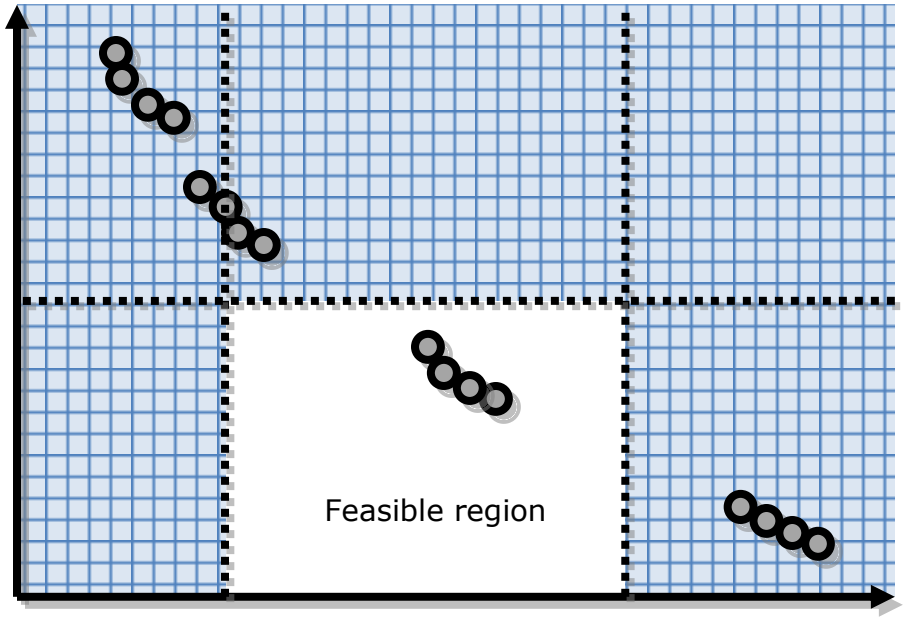

Figure 26 Example showing a constrained search space with a feasible and infeasible region.

As the crowding mechanism in PESA-II is region based and the feasibility is individual based, the order of the selection is tricked slightly. First the algorithm selects two hyperboxes at random as before, but this time, as we want to select based on feasibility and then crowding, we skip the discard of one of the boxes and choose one individual from each hyperbox at random (as before). Then the feasibility of the individuals is compared and if one is feasible and the other is not then the feasible individual is chosen. Otherwise, when the feasibility is equal (either both feasible or both infeasible) then the individual that is from the box with the lowest crowding is selected. If the crowding value is equal, then one is randomly chosen to break the tie.

If no boundary constraints are included then the algorithm uses only hypergrid crowding to select individuals.

Non-dominated Sorting (NSGA-II)

Unlike PESA-II, which is a pure elitist algorithm allowing only undominated individuals, NSGA-II allows dominated solutions to remain in the population. The population is sorted into sub-populations based on its Pareto-dominance.

Conceptually, this is achieved by first determining the undominated solutions in the population and putting these into a sub-population. The remaining solutions, which are dominated by individuals in this first sub-population, are then separated again based on dominance amongst the remaining solutions; the previously undominated solutions are excluded from this sort. This is repeated until all of the solutions have been allocated to a sub population. This process provides the solutions with a dominance ranking determined by which sub-population it was allocated to.

Figure 27 shows a population of undominated and dominated solutions divided into separate subpopulations based on their domination rank. Individuals in the first population are completely undominated, and they have a domination ranking of zero. Individuals in the second sub-population are dominated by only the individuals in the first population and have a dominance ranking of one; and so on.

This dominance ranking is the first criteria for selecting between two solutions. However, if both solutions have the same level of dominance then an extra factor is required to choose between them; this is the crowding density.

Since the solutions are already selected based on their dominance ranking, it is only necessary to determine the crowding density within each undominated sub-population. This is done by sorting the sub-population by each of the objective values in turn. So if, as in the RAP benchmark, the search

objectives are cost, weight, and reliability, each sub-population is sorted by cost, then its crowding density in the cost domain is calculated. This is repeated for the weight domain, and the reliability domain.

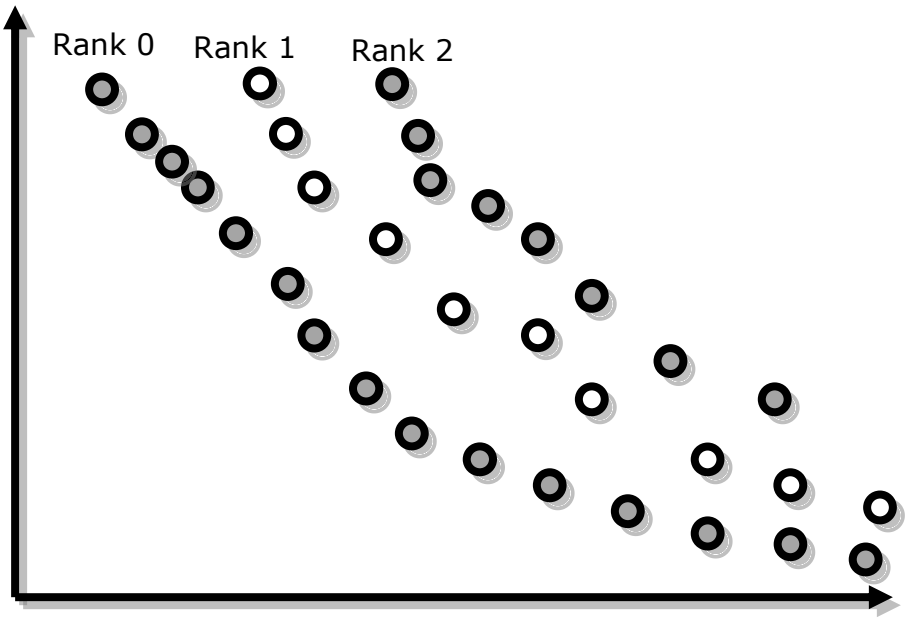

Figure 27 Example showing multiple nondominated layers with different domination rank.

The crowding density for each domain is given by the difference between the objective value in that domain of the previous and next individual in the sorted order. To ensure that individuals that exist at the edges of the Pareto frontier are preserved, the crowding density is set to infinity. For example, in the cost domain, given a individual cost sequence of 5, 9, 12, 15, 19 the individual that costs 12 has a crowding density of 6 (15-9). The individuals with costs 5 and 19 have crowding densities of infinity as they appear at the edge of the frontier.

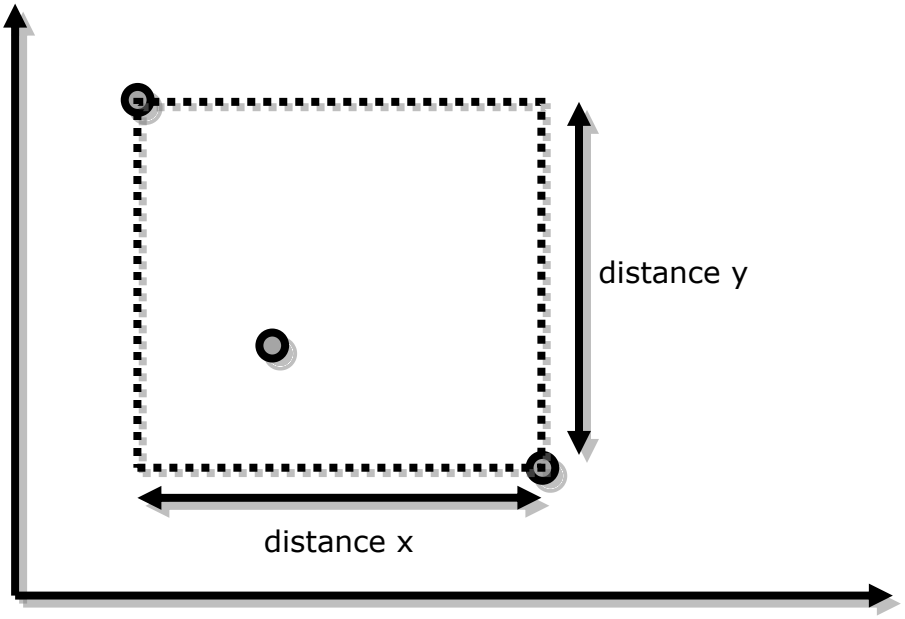

Figure 28 Example showing the crowding density for individual A.

Figure 28 illustrates how the distance between the two closest individuals relates to the solutions space.

The individuals crowding densities calculated for all domains are then summed to give the solution's crowding density. For example:

We can now select between two individuals that have the same dominance ranking by choosing the individual that has the largest crowding value as this solution will be in a less crowded region of the Pareto frontier, indicating a region of search interest.

In the penalty-based GA, a penalty function was used to roll the constrained objectives into a single fitness value for all three objectives. The constraints act to focus the search on particular areas of interest in the search space. In this purely multi-objective algorithm, however, each objective constraint is considered separately and a solution is considered either feasible or infeasible based on whether or not it violates any of the constraints. The simple

Boolean value of feasibility is established in the same way as the modified PESA-II algorithm described earlier, but it is more straightforward to include as all the selection mechanisms in NSGA-II are individual based and not region based. The feasibility criterion is then inserted between the dominance ranking and the crowding density check so that two individuals can be differentiated first by their dominance ranking, then by feasibility, then by crowding distance.

An individual A is better than individual B if:

A has a lower dominance ranking than B
OR
They have equal dominance ranking
AND
A is feasible
AND
B is infeasible
OR
They have equal dominance
AND
they have equal feasibility
AND
A is less crowded than B

This gives priority to the dominance ranking but favours the feasibility of the solution over its crowding value.

In an update to their original description of NSGA-II (DEB, K et al., 2002) a similar method for handling objective constraints is described, except that the feasibility of the solution is given priority over the dominance ranking. In fact the feasibility is rolled into the definition of dominance in a new constrained-dominance.

Individual A will constrained-dominate individual B if:

A is feasible
AND
B is unfeasible
OR
A is unfeasible
AND
B is unfeasible
AND
A has a smaller constraint violation than B
OR
A is feasible
AND
B is feasible
AND
A dominates B.

This constrained-dominance is then used in sorting and separating the population into the sub-populations and the remainder of the algorithm is unchanged.

Another difference is that the feasibility metric they describe is not a simplistic Boolean value but a measure of the extent of the boundary violations.

4.2.2.2 Genetic operators

There are three operators in a GA that work to effect change in the population; one is the selection mechanism that has been described already, and the other two are mutation and recombination. It is the emergent behaviour of the combination of these operators that generates the optimisation effect associated with evolutionary algorithms.

Mutation

The purpose of the mutation operator is to randomly perturb the population to encourage diversity and discourage the GA from becoming stuck in local minima.

Fixed length encoding

The mutation operator used by (COIT, D W and Smith, A E, 1996b) is used for the experiments that use a fixed length encoding.

Each digit in the encoding of the selected individual is considered in turn. With a mutation probability of 0.05 the digit is altered with an equal chance of either being changed to a zero, representing an empty slot, or to the value of any of the possible component alternatives for that subsystem. In the penalty based GA the current fittest individual in the population is never selected for mutation so as not to be changed and subsequently lose that information. The Pareto-based GAs do not have to do this as mutations are carried out on a copy of the selected individual.

Consider as an example the encoding below.

1	1	2	3	0

It has 5 digits resulting in 5 mutation tests. This is done by generating a random number between 0 and 1. If the random number is below the mutation probability, 0.05, then the digit is mutated. In this example the second digit is mutated. A virtual coin flip is performed to determine if the new value will be 0, an empty slot, or one of the other component identifiers, each with equal probability. In the example the result is to choose one of the other component alternatives. As there are 4 possible component alternatives, a random number between 1 and 4 is chosen and the mutated digit is

changed to it. In this example the random digit is 4 and so the encoding string becomes:

1	4	2	3	0

After any change has been made to an encoding, it is put in numerical order, with all 0s at the end of the encoding:

1	2	3	4	0

Tree encoding

For experiments using a tree encoding a mutation operator similar to that described in (KUMAR, R et al., 2009) is used.

As an example, consider the tree encoding below, describing a subsystem with 2 sub components. This is implementation 2 of the subsystem and the subcomponents are implementation 1 and implementation 3 respectively.

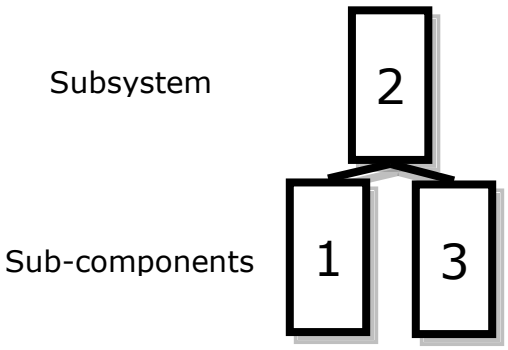

Figure 29 Example tree encoding

A depth first traversal of the tree in made and the first time that each encoding node is encountered a 50-50 random coin flip is performed to determine whether that node is mutated. In the simplest case a leaf node is selected for mutation as in Figure 30. The new implementation is randomly

selected from the available alternatives; in the example the implementation of the subcomponent changes from 3 to 2.

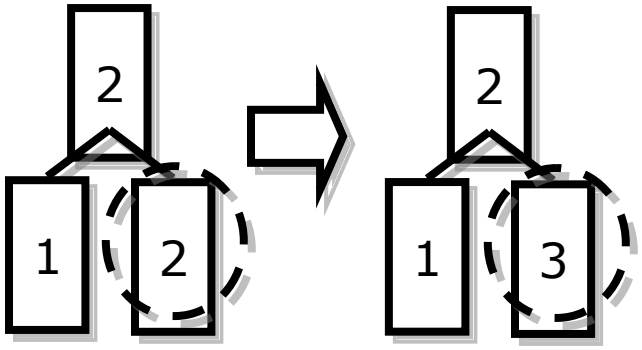

Figure 30 Example tree encoding has bottom level mutated

If an encoding node is selected that is not a leaf node (has sub nodes), as in Figure 31, then the simple case does not apply because a change of implementation of the subsystem invalidates the component choices below. The change in subsystem may be between two entirely different architectures with incompatible component choices. Thus, when a non-leaf node is selected for mutation, the sub-branch must be randomly generated as in the initialization phase of the algorithm.

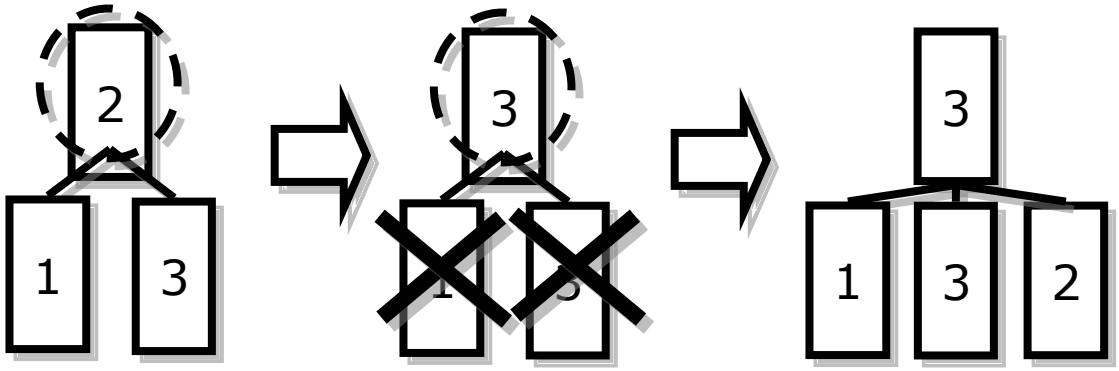

Figure 31 Example tree encoding has top level mutated

Recombination (Crossover)

The recombination operator in a GA mimics biological sexual reproduction; it is a method to produce new potential solutions by mixing and combining the encodings of existing potential solutions. As the parents of this new child are selected with a bias towards the fitter individuals, it is more likely that the new genetic information will contain successful traits. Recombination acts to search an area of the search space that is local to known current good solutions and promotes convergence in the GA.

Fixed length encoding

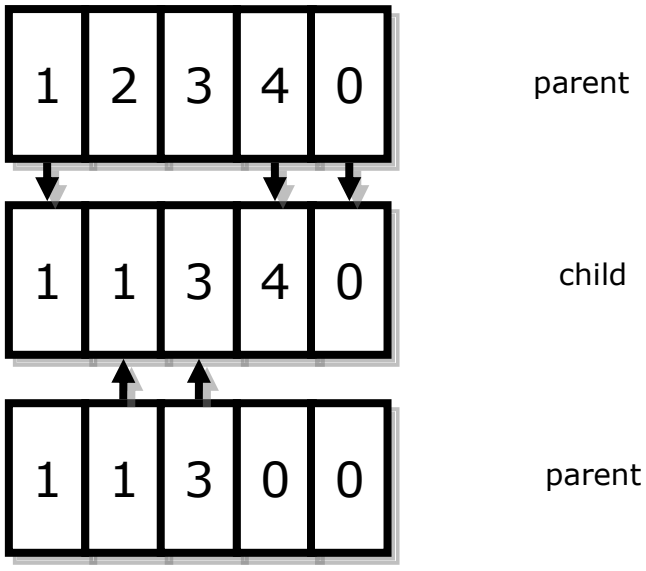

Figure 32 Two parent fixed encodings produce child encoding through crossover

Syswerda (1989) showed that in optimising combinatorial problems, such as this one, the use of uniform crossover is superior to traditional single cross-over points. In uniform crossover the new child potential solution is created by considering each of the encoding digits from the parent solutions in turn and selecting one or either of them with equal probability to exist in the child solution.

This is illustrated in the following example (Figure 32) using two selected parent encodings. The child's encoding is comprised of a corresponding digit from one of the parents. As in this example there are often occasions when the corresponding digits from each of the parents are the same making the child's digit a certainty. This is in keeping with the idea that successful genetic information is more likely to be propagated through each generation.

Although not necessary in this example the encoding is ordered after a change as with mutation.

Tree encoding

For experiments using a tree encoding a crossover operator similar to that described by Kumar *et al.* (2009) is used.

In Figure 33 there are two parent encodings selected for producing a child from recombination.

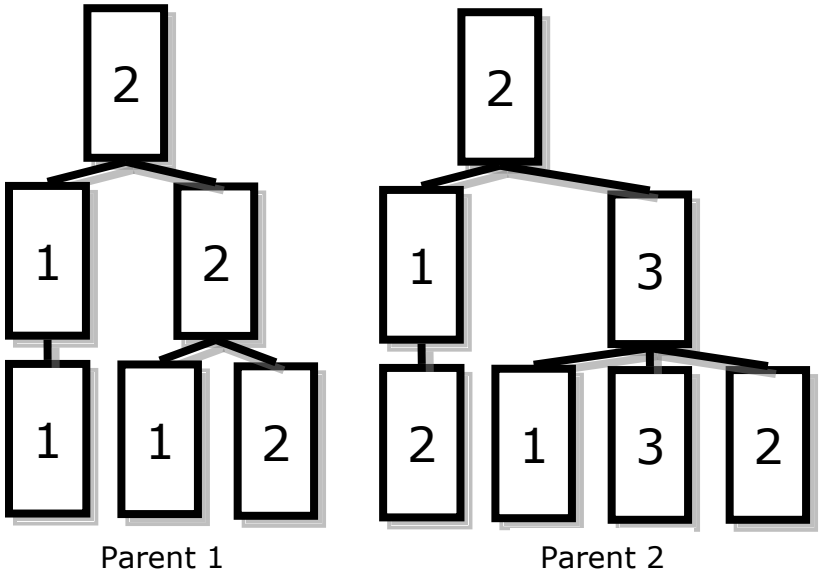

Figure 33 Example tree encodings

The child is constructed by performing a simultaneous depth first traversal of the two parent trees. Where the encountered nodes are the same, the

child's new node is certain (as both parents carry the same gene). When the encountered nodes in the two parents differ, the child's new node is chosen at random with a 50% chance from either of the parents.

In the example, Figure 34, the top node is the same in both parents (implementation 2) so it is chosen for the child node. The next node in the traversal of the parent encodings is also the same (implementation 1) and added to the child encoding tree. The leaf node of this branch is different in the two parents (implementation 1 and implementation 2 respectively) and so the child node is randomly chosen from between them; in the example implementation 2 is selected and added to the child encoding.

Finally, the other branch in the parent encoding trees has a different vertex node (implementation 2 and implementation 3 respectively) so again one is randomly selected from the two; implementation 2 is selected in this example. This node in the tree encoding has a sub branch. In order to maintain the validity of the node that is selected the entire sub-branch is also selected for the child encoding and the traversal of this branch is terminated.

4.3 Merge populations

In the penalty-based GA, the new individuals created in each generation are simply added to the main population but in the Pareto-based GAs, a new individual can only be added to the main population based on its dominance. In addition to this, a new individual may also dominate one or more individuals of the existing population and require that the existing individual is removed from the population.

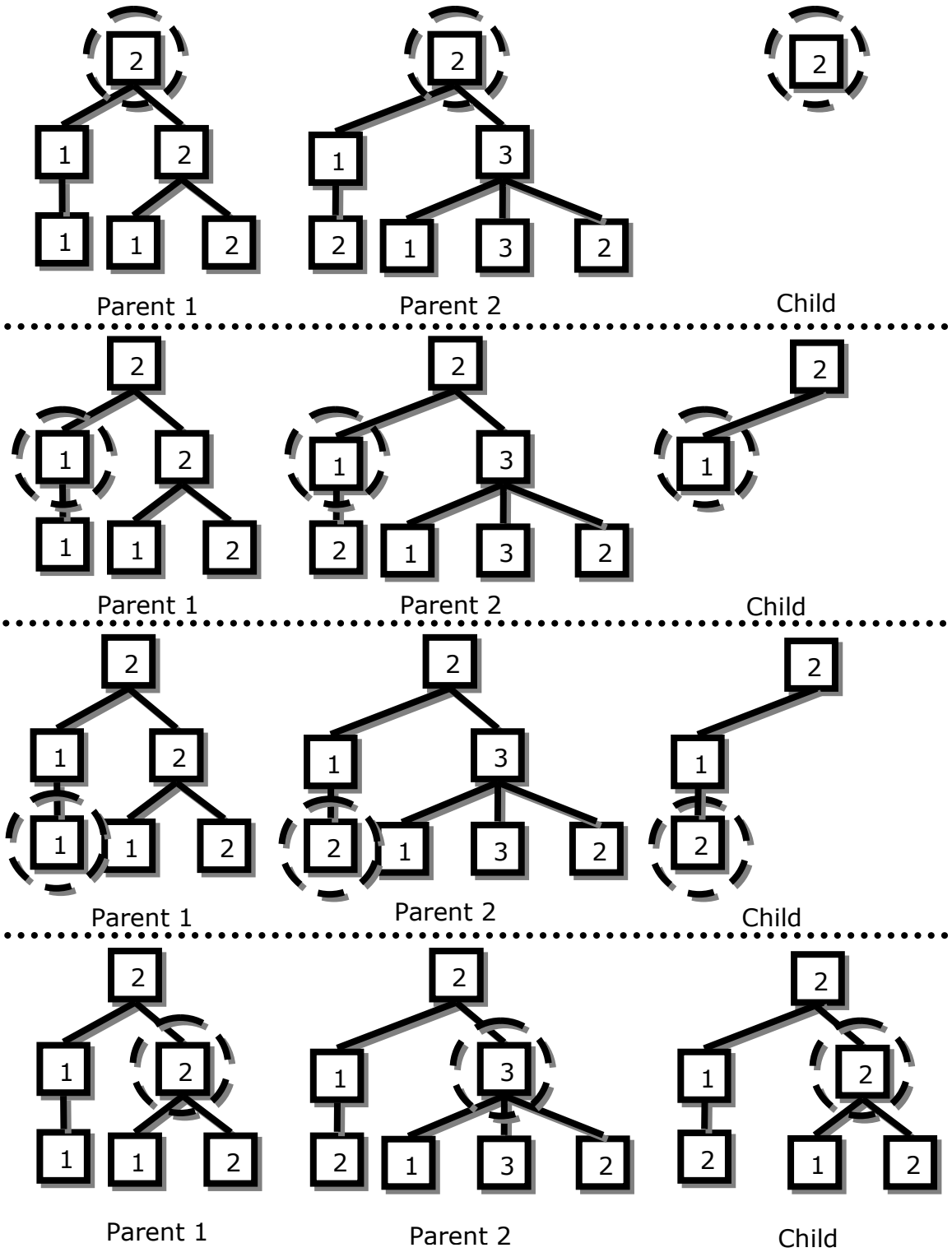

Figure 34 Recombination of example tree encodings to produce new child encoding

What happens exactly differs slightly between when there is only one non-dominated population such as in the pure elitist PESA-II and when there are multiple populations of decreasing dominance as in standard NSGA-II.

4.3.1 Multiple Pareto fronts

With the standard NSGA-II algorithm, (DEB, K et al., 2002) describe a sophisticated indexing and cataloguing system in order to establish the dominance level of all the individuals in an efficient manner. This process, for merging the existing population with the new child population, starts with simply adding all the new individuals into the main population.

The next stage is to establish which dominance level each individual has and this is achieved by doing a dominance comparison against every other individual in the joint population. If an individual is dominated then the count of individuals that dominate it is incremented and it is added to the list of dominated individuals in the other individuals.

The dominance level is then established by processing the dominance information that was collected in the previous step.

All the individuals in the population that had a dominance count of zero were undominated by any individuals; they therefore are dominance level zero and are added to a list of individuals at the current dominance level (in this case level 0). For each of these individuals in the current dominance level list, the list of individuals that are dominated by them are processed, decrementing the count of individuals that dominate it. If the dominance count reaches zero, then the individual is at the current dominance level and added to the list of individuals at this level. This new list of individuals at the current dominance level (now level 1) is processed in the same way and this sequence is repeated until the individuals in the current level have no list of

individuals that they dominate and thus all the individuals have been assigned a dominance level.

This process is repeated each generation and requires that each individual is compared to every other individual.

An alternative to this algorithm is described here that seeks to improve the efficiency of this section of the algorithm by significantly reducing the number of dominance comparisons that are required. A similar approach is given by Johnson *et al.* (2007) for their SNDL algorithm.

The method improves on the NSGA-II method in two ways: maintaining an undominated population level store that is persistent (i.e. not recreated every generation), and by not checking every individual against every other individual.

During the initialization period of the GA the undominated population has no individuals in it; only a population of random new untested individuals exists. This random population may contain individuals of varying dominance levels which must be added to the correct population.

The first efficiency benefit comes from the fact that it is only necessary to make dominance comparisons against individuals that are known to be undominated. When you add the first individual to the undominated population there is no need to check it as it must be undominated being the only individual in there. Subsequent individuals from the new population only need to be checked against the individuals that already have been added to the undominated population; thus the second individual only has to compare against the first.

If during the comparison one individual is found to dominate the other (either the new individual or one of the individuals in the undominated population) then it is immediately removed and added to a list of rejects,

and further tests are not carried out at this level. If, after the new individual has been compared to all undominated individuals, it has not been rejected, then it is added to the undominated population.

Once the new population is empty you are left with a population of undominated individuals (with dominance level 0) and a list of rejects. The above process is repeated with a new undominated population and the rejects from the previous undominated population.

Each repeat, the rejects list gets smaller until eventually you are left with no rejects and a list of undominated populations each at an increasing level of dominance.

The second improvement in efficiency comes from having this list of undominated populations remain persistent from generation to generation. The only thing that affects the dominance of existing individuals is new individuals, therefore it is not necessary to retest existing individuals against all other existing individuals as the datastructure already holds that information.

The process of merging the new and existing populations is very much the same after the initialization period (when the undominated populations structure is created and populated). The only difference is that there is an existing set of undominated populations and the new individuals are first checked against the top level (level 0) and the rejects are then checked against level 1 and so on.

It is believed that the use of persistent populations offers an improvement upon the 'test everything' approach. Instead of re-evaluating the whole population each generation, including new individuals, it is only necessary to check the new individuals and existing individuals that are dominated by the additions process.

The efficiency of the two methods will be experimentally compared measuring the differences in the number of dominance evaluations that are required.

4.3.2 Pure elitist

The PESA-II algorithm is pure-elitist: it only holds completely undominated individuals in its population. All dominated individuals are discarded.

The standard NSGA-II algorithm, as explained above, holds multiple populations in addition to the completely undominated level 0 population. However, the differences between the performance of the algorithm in its standard form and if changed to be pure elitist is a matter of interest.

In order to make NSGA-II pure elitist (and to provide the pure elitism in PESA-II), all that is required is to discard any dominated individuals when they are discovered and not maintain additional dominated populations.

4.4 Crop population

When the population size is limited, it is necessary to remove some individuals from the population to ensure that the population remains within limits after new individuals are added each generation. How the individuals are selected for this differs between the algorithms and these differences are discussed separately below.

4.4.1 Pareto envelope selection (PESA-II)

In PESA-II the same selection algorithm is used for cropping the population as for selecting the operands for the genetic operators. The only difference is that individuals in more crowded regions are chosen for removal as opposed to selecting uncrowded regions.

The two actions of removing individuals in crowded regions and encouraging new individuals in uncrowded regions encourages the algorithm to produce an evenly spread Pareto frontier which is the goal.

Similarly, when objective constraints are applied, individuals that are infeasible are preferred for deletion over feasible individuals.

4.4.2 Non-dominated sorting (NSGA-II)

There are multiple populations of decreasing dominance levels in the NSGA-II GA but not all levels are created equal. Most notably the completely undominated population at level zero is given precedence with each level beneath decreasing in order of dominance ranking. Each undominated population is sorted by its individuals' feasibility and then by their crowding density. The populations are then stacked one on top of the other in order of dominance ranking with the completely undominated population (rank zero) at the top.

Counting down the individuals in the sorted population tower, the individuals that have a count below the population limit are kept and all the others are deleted. Any sub-populations that are emptied by this process are also deleted as shown in Figure 35.

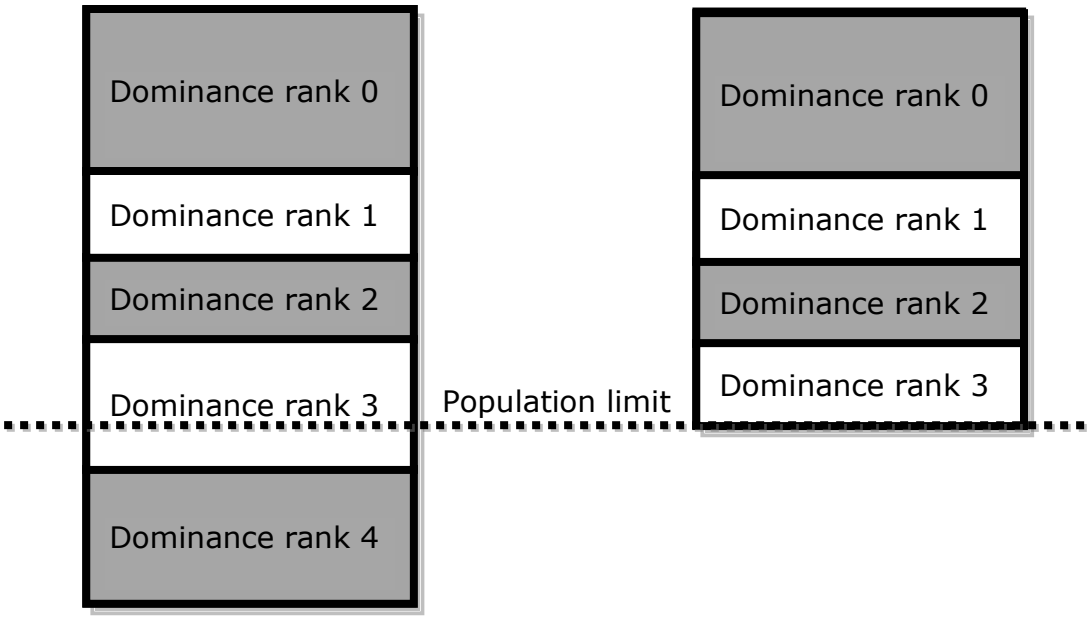

Figure 35 Cropping the ranked populations in NSGA-II.

4.5 Rinse and repeat

After the population has been cropped to the population limit, the generation is complete and a new child population is created for the next generation. This process is repeated for a specified number of generations before the GA is terminated.

4.6 Summary

The state of the art was examined, and various algorithms were discussed and theoretically evaluated in terms of their suitability for the problem examined in this thesis. A number of algorithms were chosen and it was shown how they can be adapted to enable multi-objective optimisation of models expressed in HiP-HOPS.

Overall, the contribution of the chapter is the definition of a method for genetic encoding and optimisation of HiP-HOPS models using a variety of algorithms and variants of those. In the next chapter, experiments using

those algorithms are reported and the effectiveness and efficiency of the various algorithms are compared.

5. Experimentation and Evaluation

In this chapter a number of experiments are conducted and evaluated. In each case there are two (linked) goals: maximise the effectiveness of the algorithms by finding as many of the optimum solutions as possible, and maximise the efficiency off the algorithms by finding the optimums as quickly as possible.

5.1 Can't we just use trial and error?

The optimum allocation of components and redundancy as it was defined in section 2.2.1 is not a trivial problem. In general the problem has been shown to be an NP-hard problem by Chern (1992). A particular formulation of the problem which has become a benchmark and forms one of the case studies in this thesis was calculated by Coit and Smith (1996b) to have 7.6 x 10^{33} combinations making the solution search space very large and well beyond the realms of an exhaustive search. In order to show just how badly a random search performs an implementation was created that would randomly generate a certain number of solutions. The best solutions for each of the 33 weight constraints would be kept.

In their paper, Coit and Smith (1996b) use a penalty-based GA to solve the benchmark problem. This is considered in more detail in the next section but their method evaluates 15,840,000 solutions. Since the effort required to perform the evaluation is the major overhead in the process, this number of evaluations will be set as a limit for the approaches described in subsequent sections to allow for a direct comparison of results for a given effort. For this reason 15,840,000 is the number of random solutions that were generated and the results are given in Table 14. As expected, simply using a random approach of trial and error produces poor results, generating significantly worse than optimum solutions with the average MPI being -273%.

Table 14 - Best individuals after the random generation of 15,840,000 individuals compared to the optimum results calculated by ISC method (2007).

	ISC (ONISHI, J et al., 2007)		**Results of random search**			
Weight	**Cost**	**Reliability**	**Weight**	**Cost**	**Maximum Reliability**	**MPI**
191	130	0.98681	189	110	0.95321	-254.74%
190	130	0.98642	189	110	0.95321	-244.43%
189	130	0.98592	189	110	0.95321	-232.34%
188	130	0.98538	187	112	0.95166	-230.60%
187	130	0.98469	187	112	0.95166	-215.70%
186	129	0.98418	184	106	0.94208	-266.06%
185	130	0.98351	184	106	0.94208	-251.17%
184	130	0.98299	184	106	0.94208	-240.62%
183	129	0.98226	183	106	0.93227	-281.70%
182	130	0.98152	181	102	0.92185	-322.85%
181	129	0.98103	181	102	0.92185	-311.91%
180	128	0.98029	180	99	0.91748	-318.68%
179	126	0.97951	174	111	0.91361	-321.53%
178	125	0.97840	174	111	0.91361	-299.97%
177	126	0.97760	174	111	0.91361	-285.61%
176	124	0.97669	174	111	0.91361	-270.63%
175	125	0.97571	174	111	0.91361	-255.64%
174	123	0.97493	174	111	0.91361	-244.55%
173	122	0.97383	169	104	0.88231	-349.65%
172	123	0.97303	169	104	0.88231	-336.31%
171	122	0.97193	169	104	0.88231	-319.24%
170	120	0.97076	169	104	0.88231	-302.48%
169	121	0.96929	169	104	0.88231	-283.23%
168	119	0.96813	166	106	0.88206	-270.00%
167	118	0.96634	166	106	0.88206	-250.32%
166	116	0.96504	166	106	0.88206	-237.36%
165	117	0.96371	165	88	0.86943	-259.83%
164	115	0.96242	163	92	0.86292	-264.78%
163	114	0.96064	163	92	0.86292	-248.28%
162	115	0.95919	162	98	0.86094	-240.74%
161	113	0.95804	161	86	0.85524	-244.95%
160	112	0.95571	160	94	0.82981	-284.30%
159	110	0.95457	159	102	0.82651	-281.84%
					Mean MPI	**-273.40%**

5.2 Penalty based GA

The penalty based GA is included here for comparison as an existing heuristic method that produced moderately good solutions to the benchmark RAP. It demonstrated that HiP-HOPS could reasonably be used to provide the evaluation for the fitness function of the GA.

The following results were gathered from the experimental procedure described earlier. A feasible solution was found for each of the allowable weight constraints and the fixed cost constraint was also met for all tests.

Table 15 shows the reliability and cost of each of the obtained solutions for each of the weight constraints.

The solutions that equalled the optimum solution are highlighted in grey and the HiP-HOPS penalty GA used in this experiment found the optimum solution in 15 cases where Coit and Smith's original penalty GA from (1996b) only found the optimal solution in 8 cases. The HiP-HOPS penalty GA also found superior solutions to the original in 18 cases and equalling it in a further 10. The best solution found by the HiP-HOPS penalty GA was worse than the original penalty GA in 5 cases.

It would be nice to imagine that the use of HiP-HOPS as the fitness evaluator caused the observed improvements in the solutions; however, the reliability value for a given encoding is identical when calculated by HiP-HOPS or using the RBD method employed by the other researchers and thus offers no explanation.

It is possible that, given the stochastic nature of GAs, the improvements were the result of chance but this seems unlikely as the HiP-HOPS penalty GA found equal or superior solutions to the original penalty GA in nearly all cases.

Table 15 - Best individuals after 10 runs of HiP-HOPS penalty based GA with 1200 generations, population size 40, for each of the 33 weight limits compared to optimum results calculated by ISC method (2007). Solutions matching optimum highlighted.

	HiP-HOPS penalty based GA					
Weight	**Cost**	**Maximum Reliability**	**Mean Reliability**	**Minimum Reliability**	**Standard Deviation**	**MPI**
191	130	0.98592	0.98552	0.98422	0.000625	-6.76%
190	130	0.98543	0.98499	0.98454	0.000414	-7.30%
189	129	0.98558	0.98426	0.98203	0.001154	-2.46%
188	129	0.98518	0.98451	0.98366	0.000427	-1.35%
187	130	0.98469	0.98361	0.98166	0.000870	0.00%
186	129	0.98332	0.98252	0.98079	0.001038	-5.39%
185	130	0.98351	0.98231	0.98105	0.000914	0.00%
184	130	0.98242	0.98169	0.97896	0.001086	-3.39%
183	130	0.98188	0.98098	0.97980	0.000814	-2.14%
182	130	0.98124	0.98003	0.97813	0.001117	-1.52%
181	129	0.98045	0.97968	0.97810	0.000972	-3.04%
180	130	0.98001	0.97891	0.97743	0.000868	-1.42%
179	129	0.97927	0.97805	0.97674	0.000980	-1.13%
178	128	0.97817	0.97739	0.97635	0.000805	-1.07%
177	126	0.97760	0.97654	0.97540	0.000782	0.00%
176	125	0.97649	0.97586	0.97454	0.000817	-0.84%
175	126	0.97536	0.97480	0.97389	0.000679	-1.44%
174	124	0.97479	0.97347	0.97291	0.001042	-0.55%
173	125	0.97381	0.97276	0.97076	0.001047	-0.08%
172	123	0.97303	0.97170	0.97076	0.000884	0.00%
171	122	0.97193	0.97137	0.96988	0.000893	0.00%
170	120	0.97076	0.97027	0.96953	0.000636	0.00%
169	121	0.96929	0.96852	0.96836	0.000338	0.00%
168	119	0.96813	0.96733	0.96661	0.000692	0.00%
167	118	0.96634	0.96584	0.96573	0.000752	0.00%
166	116	0.96504	0.96389	0.96360	0.000509	0.00%
165	117	0.96371	0.96299	0.96231	0.000567	0.00%
164	115	0.96242	0.96128	0.96053	0.000478	0.00%
163	114	0.96064	0.96010	0.96005	0.000189	0.00%
162	115	0.95919	0.95816	0.95761	0.000450	0.00%
161	113	0.95804	0.95683	0.95654	0.000437	0.00%
160	114	0.95567	0.95549	0.95422	0.000456	-0.10%
159	110	0.95433	0.95355	0.95324	0.000303	-0.53%
					Mean MPI	**-1.23%**

The standard deviation over each of the 10 test runs for each of the 33 weight constraints shows a slight increase in standard deviation in the HiP-HOPS penalty GA when compared to the original penalty GA with the standard deviation being lower in just 8 cases. The mean standard deviation over all 33 test cases was 0.000692 and 0.000445 respectively for the HiP-HOPS penalty GA and the original penalty GA; not a very large difference.

Another possible source of difference between the two implementations is the pseudo-random number generator (PRNG) used to provide the random elements of the algorithm. The HiP-HOPS penalty GA uses the GALib (WALL, M, 1999) implementation of the ran2 algorithm (PRESS, W H et al., 1992). This gives robust random numbers, certainly more so than the stock random number generators in the C libraries.

The PRNG used by Coit and Smith (1996b) is unspecified, making direct comparison impossible. A study into the effect of choice of PRNGs on the performance of a GA showed that the use of a particular PRNG in combination with particular use cases of GAs can affect performance significantly (MEYSENBURG, M M and Foster, J A, 1999).

This is particularly true in the initializing of the population, with the effect on other operations being negligible. In a later work this is confirmed but it is concluded that the good fit of a particular PRNG with a particular GA is a lucky and unpredictable outcome and that the general advice would be to use the best PRNG available in order to avoid the skewing of results in this way (CANTÚ-PAZ, Erick, 2002).

In both cases it seems that the quality of the random numbers being generated does not have a significant effect on the performance; high quality random numbers do not produce significantly better performing GAs than lower quality random numbers.

A difference in the choice of PRNG between HiP-HOPS penalty GA and the original penalty GA could be the cause of the differences in results, but again without knowing the PRNG used for the latter it is impossible to be sure.

Certainly the other minor differences between the two GA implementations (the use of 0 instead of m+1 to denote empty component slots and the non-ranking of components by reliability before assigning encoding IDs) seem to be superficial with regard to this and would have no bearing.

Another thing that can be seen in the results is that generally better solutions are found for the test cases where the weight constraint is tighter. This is explained by Coit and Smith (1996b) as being a result of the increased severity of the penalty function improving the performance of the GA.

What is worth noting is that the differences between the calculated optimum solutions and the near-optimal (optimality not guaranteed) solutions of the GA are small both in the cases of the reliability achieved and the configuration of components required to achieve it.

This is good as it supports the premise that while the GA cannot guarantee optimality it can achieve good (enough) solutions to complex problems whilst retaining generality.

The main purpose of this experiment though was to establish whether the simplistic RBD failure model for calculating reliability could be feasibly replaced by an analysis tool such as HiP-HOPS that could potentially allow more complex systems to be optimised.

As explained previously the reliability value calculated by HiP-HOPS was exactly the same as the reliability value calculated using the RBD failure model; this issue of whether the results obtained are accurate is therefore satisfied. The remaining question is with regard to the performance of the calculations. This is particularly relevant as the GAs optimise potential

solutions over many iterations, so if the calculation takes too long the cumulative time cost can become too expensive to be realistically used.

Although Coit and Smith (1996b) did not include timings for their GA runs, a version of the implementation created for this thesis was made that used the RBD failure model calculation mechanism as with their original work.

On average, using the RBD failure model, it took approximately 20 seconds for each run of the GA; when using HiP-HOPS to provide the evaluation calculation that time went up to 1 hour.

Clearly there is a significant overhead that accompanies the use of a more complex failure modelling tool. However, there are some things to bear in mind when considering these times. Firstly, while 1 hour is without doubt considerably longer than 20 seconds it is not an unreasonable period of time to wait for an optimisation to complete. Even if it was longer, in any given day there are at least approximately 16 hours of 'idle' time, out of office hours, where such an optimisation could be run without impacting on work time.

Furthermore there are significant portions of the GA that would lend themselves to parallelisation. In each generation, 18 new potential solutions are created through recombination. These occur independently of one another and so could potentially be performed in parallel.

Equally the 22 mutations that occur every generation could be achieved in parallel. Given 1200 generations with operations occurring in series, that makes for 18 x 1200 = 21600 recombinations and 22 x 1200 = 26400 mutations. That is a combined total of 21600 + 26400 = 48000 genetic operations, each of which requires a (high overhead) HiP-HOPS analysis.

If performed in parallel, however, the same number of operations are required in total, but many can be performed simultaneously; specifically, all

the recombinations and all the mutations in each generation. That would result in the equivalent of 1 recombination time period and 1 mutation time period per generation: 1 x 1200 = 1200 for recombination and 1 x 1200 = 1200 mutations over the course of the GA. This totals 1200 + 1200 = 2400 genetic operator time periods compared to the previous 48000.

This is 20 times less time making 1 hour into just 3 minutes. This is a crude approximation that doesn't take into account any of the overheads that accompany parallelization, but it does illustrate the potential for improving the performance of the algorithm.

It seems clear that the use of a complex failure model and accompanying analysis tools in an optimisation application is feasible and whilst the test example does not have any complex features that would warrant such a use, it does prove the concept and encourage further investigation.

5.3 Pareto based GAs

Generally we are more interested in the true multi-objective GAs, PESA-II and NSGA-II. Each of these search algorithms has a number of different parameters that can be altered to modify the performance of the search and the following experiments will show the effect of changes to these parameters and their importance. The parameters are: main population limit, child population limit, constraints use, archive use, encoding type, pure elitism, and - for PESA-II only - the hypergrid size. Also to be investigated is whether taking the best solution from 10 short runs (like with the Coit and Smith penalty based GA) or 1 long run produces better solutions. A comparison of the standard NSGA-II dominance algorithm and the persistent population store will also be conducted and discussed. Finally, we shall determine if the size of the main population is an important factor, and if so,

how we can determine an effective limit without inefficient experimental trial and error.

To allow accurate comparisons to be made, the number of solutions evaluated is always kept the same and is set to 15,840,000 evaluations (the number of evaluations used by Coit and Smith in their penalty-based GA). Therefore if the number of new individuals created each generation is 150 then the number of generations the algorithm is to run for is 15840000 / 150 = 105600 generations for one long run or 105600 / 10 = 10560 generations for 10 repeated runs. If 1500 individuals are created in each generation, there are 15840000 / 1500 = 10560 generations for one long run or 10560 / 10 = 1056 generations for 10 repeated runs.

5.3.1 Main Population Limit

Population size is important as it applies pressure to the search. If the population size is too small then good solutions can become extinct from the population (or never be evolved) and if the population is too large then there is no pressure on the population to adapt better solutions. To illustrate this consider a biological example: in a population of herbivores where there is no predation a shortage of food limits the population and forces the population to adapt; perhaps to different food sources or more efficient use of existing sources. If however the food constraint is removed, effectively unlimiting the population then there is no pressure to adapt or improve. A creature would not have to be efficient if there was no competition for resources. Poorly adapted individuals could happily coexist with better individuals as in effect there would be no such thing as poorly adapted, just differently adapted.

This will be shown empirically by running the algorithms with different population limits to compare the performance. The populations will be limited to 150 individuals, 1500 individuals, and unlimited, with the experi-

ments repeated to include all other property combinations (encoding, child population, archive, elitism, focussing).

Table 16 comparison of MPI for different main population limits for all combinations of other properties.

	Population limit		
	150	**1500**	**unlimited**
Min	-307.124%	-32.983%	-4.855%
Mean	-42.120%	-5.608%	-0.940%
Max	-10.377%	0.000%	0.000%

Table 16 shows that when the population is unlimited the average MPI is better than the other population limits (smaller negative MPI). However the results are badly biased by the experimental runs where the algorithm was unfocussed to the constraints. The performance of the algorithm when unfocussed was, in effect, so bad with such large negative MPIs that it unbalanced the results. A more accurate result was achieved by excluding the unfocussed results thus giving the results in Table 17.

Table 17 comparison of MPI for different main population limits for all combinations of other properties but with the unfocussed experiments excluded.

	Population limit		
	150	**1500**	**unlimited**
Min	-307.124%	-0.761%	-0.791%
Mean	-48.181%	-0.121%	-0.143%
Max	-12.277%	0.000%	0.000%

This shows that setting the population too small has a disastrous effect on the performance of the algorithm and that limiting the population to a reasonable size (in this case 1500) results in better performance than when the population is not limited at all.

5.3.2 Child population limit

The size of the new child population created in each generation is different in the described methods of the two Pareto based GAs. In PESA-II the child

population is small (about 10% of the main population) and in NSGA-II the child population limit is equal to the main population limit.

The creation of new individuals in the child population is one of the computationally expensive portions of the algorithm as it requires the evaluation of fitness for that individual. In order to make a fair comparison of computational effort, it is necessary to alter the number of generations the algorithm is run for along with the size of the child population. This way, the number of individual evaluations remains the same for all experiments. The child population limits being tested are 150 and 1500. Obviously the child population cannot be unlimited as this would cause an infinite loop. The experiments where the child population size is 150 will be run for 10 times as many generations to compensate for the 10 times reduction in evaluations.

Table 18 shows the results from these experiments though as before some of the experimental runs were excluded from the results data as they were so poorly performing that their MPI values unfairly skewed the results; the unfocused algorithms and experiments where the main population was limited to 150 were excluded.

Table 18 Comparison of mean MPIs for different child population limits. The experiments where the algorithm is not focussed to the constraints or with a main population limit of 150 are excluded to remove their unfair bias from the data.

	Child population limit	
	150	**1500**
Min	-0.737%	-0.791%
Mean	-0.101%	-0.163%
Max	0.000%	0.000%

The results show that having a smaller child population limit (with corresponding larger number of generations) produces a higher performing algorithm. An explanation for this is that individuals are selected from the current population to become parents for the new solutions each generation.

In theory the GA is designed to converge on the optimum solutions with each generation being better than the last. So selecting a smaller number of parents from each generation (but overall selecting the same number of parents) means that more parents are selected from later (theoretically better) generations

5.3.3 Solution Archiving

As explained above, the population size can have a significant effect on the performance of the algorithm; if the population size is too small then good solutions can be lost from the population. A way of ensuring that optimal solutions are not lost is to maintain a separate archive of undominated solutions. This ensures that all undominated solutions encountered at any point during the search will be kept. The downside to this approach is that it adds an overhead to the algorithm as separate dominance checks must be made on the archive population as well as the standard population.

Each experiment will output the archived solution set in addition to the standard final population to compare the usefulness of including an archive. Table 19 shows that when you include an archive population the algorithm produces much better results, or more accurately, the good solutions that are discovered by the algorithm are not lost as they are when the archive population is not present.

Table 19 Comparison of mean MPIs when archive is used and when no archive is used.

	Archive	No Archive
Min	-39.400%	-307.124%
Mean	-5.221%	-16.866%
Max	0.000%	0.000%

5.3.4 Using constraints

Where the specification of constraints is possible, it seems reasonable that focusing the search into the region of feasible solutions will make for a more efficient search. Exactly how much of a difference does it make to the quality of solutions? By comparing the results of the experiments when the algorithm is constrained, using the RAP weight constraints of 159-191 and the cost constraint of maximum 130, and when it is unconstrained, we can establish the effectiveness of search focus.

As previously, some of the settings for the algorithm result in a performance that is so bad that it unfairly weights the results. In this case setting the main population limit to 150 had this effect so the experiments that had this setting were excluded from the results of this experiment shown in Table 20.

What is shown is that, understandably, the performance of the algorithm within a certain range is significantly better when the algorithm is actually focused to search within that range.

Where knowledge of the search space is not available and so reasonable constraints to focus the algorithm are not known, it seems reasonable to allow the algorithm to run for a set period with no population limit and no constraints. The resultant solution set can then be visualised by the user, as in Figure 36, and a reasonable set of constraints applied. The algorithm could then be continued with the focus applied to the search to maximise the exploration of the area of search space of particular interest to the user.

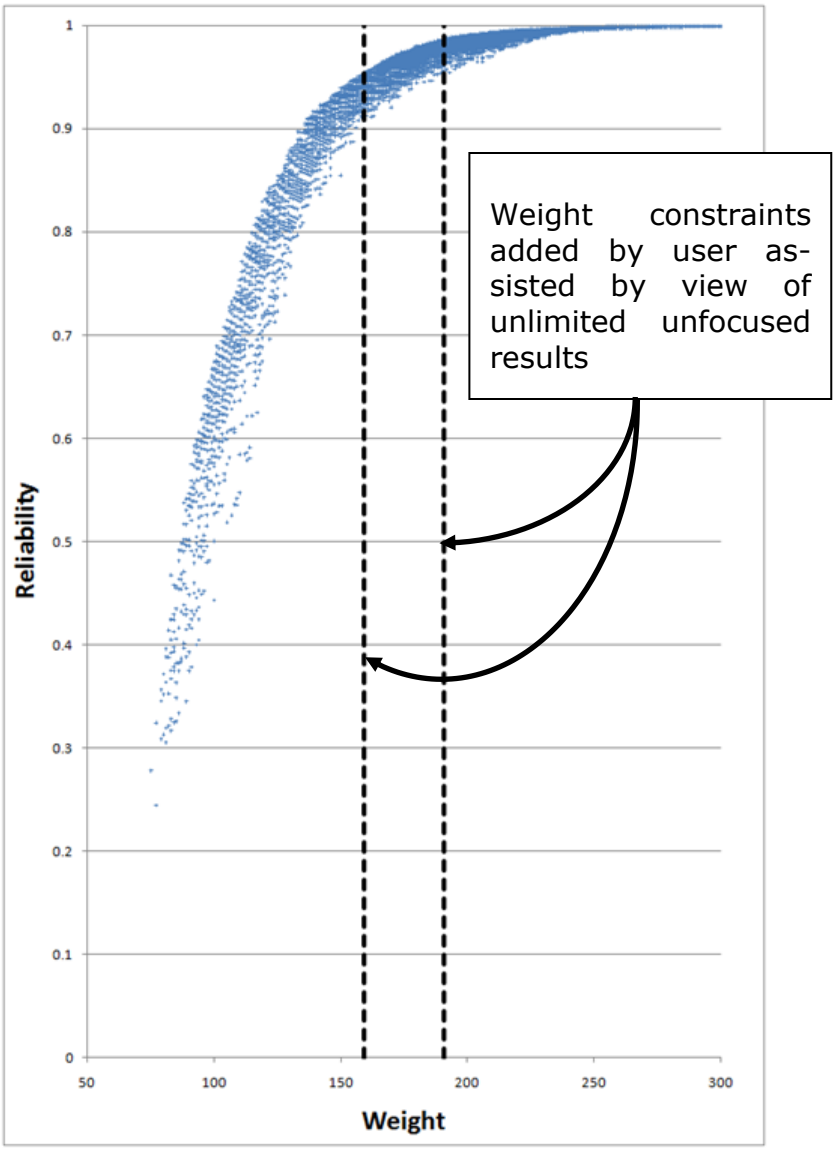

Figure 36 Constraints could be added following a preliminary run of the GA (unlimited and unfocused) to provide a visualisation of the search space to the user.

Table 20 Comparison of mean MPIs when algorithm is focused between constraints and when it is unfocused.

	Focused	Unfocused
Min	-0.791%	-32.983%
Mean	-0.132%	-6.417%
Max	0.000%	-0.030%

5.3.5 Encoding types

The penalty-based GA made use of the fixed length. Using a tree encoding allows much more complex structures to be described for the error model. However, this raises the question of whether the use of the different encoding structures make any difference to the effectiveness of the algorithm. Both encoding types are compared to find out with the results in Table 21. As with previous experiments the unfocussed algorithms and the algorithms with the population limited to 150 were excluded from these results due to the unfair bias they caused.

Table 21 Comparison of mean MPIs when algorithm using a tree encoding and using a fixed length string encoding.

	Fixed encoding	Tree encoding
Min	-0.791%	-0.761%
Mean	-0.100%	-0.164%
Max	0.000%	0.000%

The fixed length encoding generally performs better on this benchmark problem though the difference is not large. There are a couple of plausible explanations for this.

Firstly, the tree encoding is more susceptible to catastrophic mutation during the late generations of the GA. This happens when one of the upper level nodes of the tree gets mutated, forcing the entire sub tree to be randomly re-generated. This can make fine tuning of the lower level nodes by the algorithm more difficult.

A possible solution to this, that has not been tested and is a subject for future work, is to have a variable mutation rate. Nodes lower in the tree encoding are subject to a higher mutation rate than nodes at the higher levels. This means that the fine tuning of configurations defined by the higher level nodes can be achieved with a reduced risk of a mutation causing the entire sub tree to be reset. A further variation to this could include

generation age as a factor for varying the mutation rate, where mutation rate at higher levels of the tree encoding diminishes at higher generation levels.

A second plausible explanation is that for this benchmark problem the tree conceptually consists of two levels, one to select the number of components in parallel, and the lower level to select the specific component used in each slot of the redundancy. The fixed length string encoding is sorted to ensure that for each allele the bits of the string are always in the same order, effectively reducing the search space by ensuring that there was a one-to-one relationship between a solution and its encoding. For example, without sorting, an encoding 4213 produces the same solution as 3214, but if they are sorted by their bits the encoding becomes 1234 in both cases. This can be done with the fixed encoding because the redundancy configuration of the components always equates to the same series parallel arrangement.

This is not possible to do with tree encoding because, as illustrated in Figure 37, two nodes at the same tree level may not relate to an equivalent component, even if they have the same implementation number.

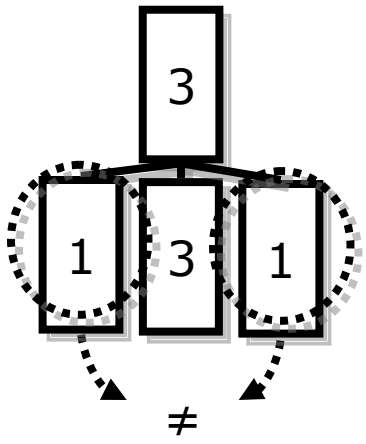

Figure 37 Shows that two nodes at a particular tree level that have the same implementation number do not necessarily relate to the same component.

Whilst it may be possible to hard-code such equivalent functionality as the fixed length sort into the tree encoding for the experiments, there isn't a straight forward solution to achieve this and it unnecessarily constrains the general case.

5.3.6 Pure elitism

PESA-II uses a pure elitist strategy where only the undominated solutions are kept. NSGA-II also includes dominated solutions in separate subpopulations of increasingly dominated individuals. Does the effectiveness of the NSGA-II algorithm change if it uses a pure elitist strategy too? The multiple subpopulations of NSGA-II are compared to using NSGA-II with only one pure elitist population. As before, the poorly performing unfocused algorithm and population limit set to 150 were excluded from the results to avoid unfair results bias.

As shown in Table 22 adopting a pure elitist strategy, where only undominated individuals are kept, results in an improvement over the multiple levels of dominance of the standard NSGA-II algorithm.

Table 22 Comparison of mean MPIs when algorithm using a pure elitism as opposed to the multiple dominance level populations.

	Pure elitist	Multiple dominance levels
Min	-0.761%	-0.791%
Mean	-0.106%	-0.159%
Max	0.000%	0.000%

5.3.7 Hypergrid size

PESA-II uses a hypergrid to establish the crowding metric for the algorithms selection mechanism. This adds an extra parameter, the hypergrid size, to those needing to be specified by the user. The algorithm will be tested at different hypergrid sizes starting at 32, as used in the PESA-II paper, and

doubling it to 64, 128, 256, until 131072 to see what effect the hypergrid dimensions have on the algorithm.

As good settings for the other parameters had been determined experimentally from the NSGA-II experiments, these were used here:

- Child population size – 150.
- Main population size – 1500.
- Pure-elitist. (PESA-II is anyway)
- Focused within constraints.
- Fixed length encoding.
- Include archive population of undominated feasible solutions.

The result of this experiment can be found in Table 23 and Figure 38.

Table 23 Comparison of the effects of hypergrid size.

Hypergrid size	Mean MPI over 33 weight constraints	Number of optimums found
32	-12.937%	0.000%
64	-16.212%	0.000%
128	-14.052%	0.000%
256	-16.016%	0.000%
512	-9.623%	0.000%
1024	-6.716%	0.000%
2048	-2.948%	6.061%
4096	-0.441%	21.212%
8192	-0.298%	21.212%
16384	-0.199%	36.364%
32768	-0.155%	63.636%
65536	-0.203%	30.303%
131072	-0.139%	57.576%

It is clear from the results that choosing the correct hypergrid size for the PESA-II algorithm is very important. In this set of experiments the beginning of the plateau in the graph occurs at a hypergrid size of 4096.

There remains a problem, however, that there doesn't appear to be a systematic (or intuitive) way of determining a good hypergrid size. In fact even

when the hypergrid size was set to a 'good' value it still performed significantly worse than the NSGA-II algorithm, which does not require the user to be lucky in their choice of hypergrid size.

Therefore further investigation of PESA-II was discontinued in favour of the NSGA-II variant algorithm.

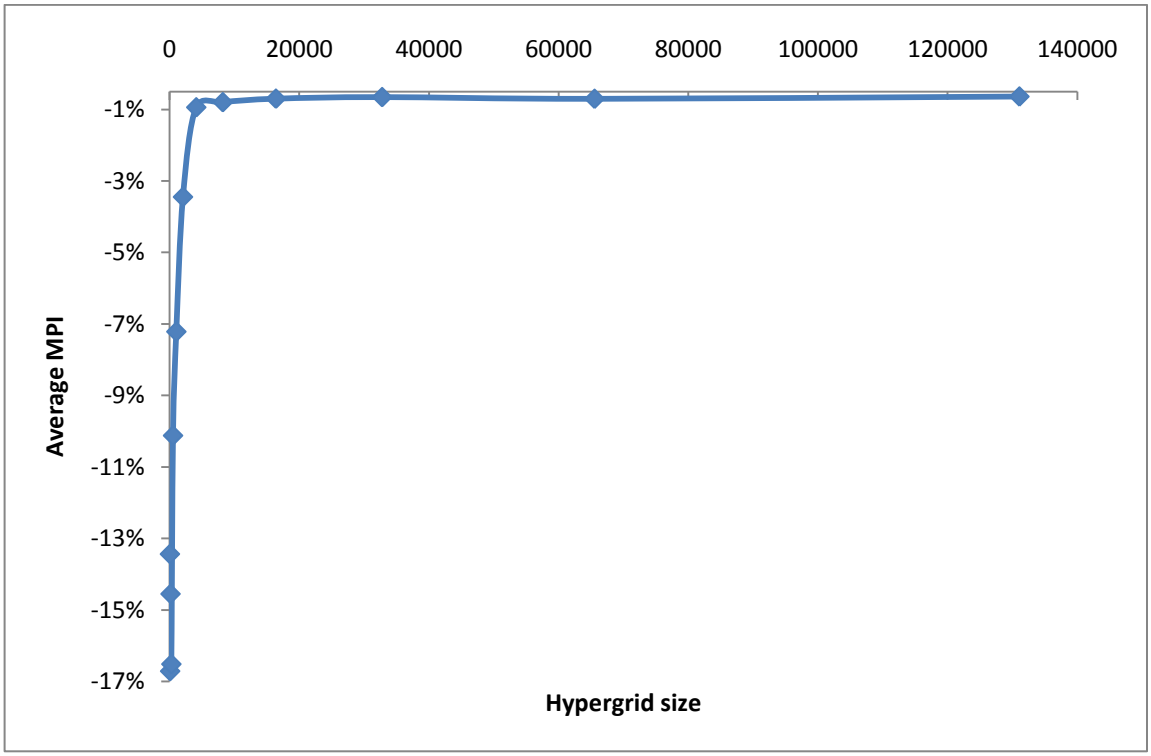

Figure 38 Chart showing effect of hypergrid size.

5.3.8 Best of ten, or one long run?

The penalty based GA described by Coit and Smith (1996b) conducts 10 runs of the algorithm and then selects the best solution from the ten runs. This is done to minimize the effect of the random initial population.

The penalty based GA is strongly convergent with each run often converging to a single solution and the makeup of the initial population is a strong factor influencing the point of convergence. In the multi-objective algorithms, initial make up does not have such a strong an effect and so it may be possible that improved results can be obtained by running the algorithm once for ten times as long, rather than ten times for a smaller number of generations.

This was tested by using the previously found best settings:

- Child population size – 150.
- Main population size – 1500.
- Pure-elitist.
- Focused within constraints.
- Fixed length encoding.
- Include archive population of undominated feasible solutions.

The number of generations was set to 105600 for each of the 'single long runs' and 10560 for each of the '10 short runs'. In order to establish the consistency of results the 'single long run' was repeated 10 times and the '10 short runs' were repeated 10 times making 100 runs in total. The results are shown in Table 24 with the percentage of runs that the optimum was found for each weight constraint.

What the results show is that the algorithm performed significantly better when allowed to run for longer finding 100% of the optimums for 32 of the weight constraints, and 70% for the weight constraint 182. This is compared to the very variable results from doing short runs, most significant of which is the low 8% discovery of the 182 weight optimum. On average the long run found the optimum 99.091% outperforming the short runs' 93.545%.

Table 24 The percentage of experimental runs that found the optimum solution for 100 short runs compared with 10 long runs.

Weight	short runs	long runs
191	93.000%	100.000%
190	99.000%	100.000%
189	100.000%	100.000%
188	99.000%	100.000%
187	99.000%	100.000%
186	97.000%	100.000%
185	85.000%	100.000%
184	99.000%	100.000%
183	98.000%	100.000%
182	8.000%	70.000%
181	97.000%	100.000%
180	99.000%	100.000%
179	100.000%	100.000%
178	99.000%	100.000%
177	98.000%	100.000%
176	100.000%	100.000%
175	81.000%	100.000%
174	100.000%	100.000%
173	100.000%	100.000%
172	100.000%	100.000%
171	98.000%	100.000%
170	100.000%	100.000%
169	99.000%	100.000%
168	100.000%	100.000%
167	100.000%	100.000%
166	100.000%	100.000%
165	97.000%	100.000%
164	100.000%	100.000%
163	99.000%	100.000%
162	71.000%	100.000%
161	100.000%	100.000%
160	73.000%	100.000%
159	99.000%	100.000%
Min	**8.000%**	**70.000%**
Mean	**93.545%**	**99.091%**
Max	**100.000%**	**100.000%**

Despite this variation the two actually have similar success rates at finding the optimums because when using the short runs strategy, the best solution found after ten runs is chosen, so that success rate more than or equal to 10% would result in a successful optimum discovery over the ten runs. In

contrast the long run is only performed once, so the optimum must have a 100% discovery rate to be successful. What this translates to in the results (as seen in Table 24) is that both 10 short runs and 1 long run find the optimums for 32 out of the 33 weight constraints, with both not finding the 182 global optimum.

Though they perform similarly in discovering the optimums, there is another performance factor in which they are not equal. This is the percentage of the total number of generations that were required to find the optimums. In Table 25 it can be seen that the number of generations for the long run can on average be reduced by nearly 95% whereas the shorter runs would only allow an average saving of 58%.

In reality, making direct use of this property is difficult as the GA is not guaranteed to find optimum solutions and the convergence of multi-objective algorithms is difficult to ascertain. It is therefore not possible to establish the best number of generations prior to performing the algorithm. However, given a number of generations, it seems reasonable that you could have a greater expectation of better performance from the single longer run than from multiple shorter runs.

Table 25 The mean percentage of total number of generations that were required to find the optimum solutions, for 100 short runs compared with 10 long runs.

Weight	short runs	long runs
191	46.135%	6.229%
190	40.964%	5.247%
189	33.975%	3.155%
188	33.858%	3.118%
187	37.331%	4.038%
186	46.608%	4.614%
185	41.710%	8.027%
184	46.742%	4.373%
183	46.896%	5.465%
182	61.824%	41.057%
181	47.335%	5.299%
180	45.832%	4.941%
179	33.443%	3.337%
178	38.115%	3.958%
177	42.177%	4.332%
176	39.758%	4.038%
175	50.192%	6.759%
174	34.675%	3.390%
173	38.425%	3.674%
172	40.833%	4.169%
171	40.946%	3.808%
170	36.564%	3.595%
169	42.820%	4.182%
168	37.249%	3.822%
167	43.277%	3.825%
166	38.437%	3.684%
165	41.880%	3.914%
164	38.355%	3.486%
163	41.167%	4.123%
162	50.099%	8.358%
161	40.282%	3.773%
160	52.010%	8.807%
159	43.254%	3.949%
Min	**33.443%**	**3.118%**
Mean	**42.217%**	**5.714%**
Max	**61.824%**	**41.057%**

5.3.9 How to choose population limit

We established through experimentation that the size limit of the population makes a big difference to the solutions found by the GA. It can be set too small, causing the algorithm to thrash about (never finding good solutions),

or too large, never putting sufficient selective pressure on the population to converge. Only setting the limit 'just right' produces the best solutions. But how does one determine a good population limit without performing expensive experimental trial and error?

A proposed solution to this problem is to set the population to unlimited for 10% of the total generations run of the GA. Once it has reached 10% the population limit should be set to the size of the archive population of feasible undominated individuals.

To test this, the best settings from the previous experiments were chosen as follows:

- Child population size – 150.
- Pure-elitist.
- Focused within constraints.
- Fixed length encoding.
- Single long run of 105600 generations.
- Include archive population of undominated feasible solutions.

The experiment was repeated 10 times to ensure consistency in the results and compared with the same procedure but setting the main population limit to 1500.

The result of this experiment is as follows:

All had a high success rate at finding the optimums. The set population limits (1588 and 1500) found 32 of the 33 optimums 100% of the time, in all 10 of the experimental runs. For the remaining optimum (where the weight limit is set to 182) the fixed population size found the optimum solution 9 out of the ten times (90%) and 7 out of ten times (70%) respec-

tively for population limit 1588 and 1500. The remaining runs for this weight limit found the same local optimum with MPI -0.27%.

The automatic variable population size algorithm had a 100% success rate in finding the optimum therefore finding the optimums more frequently that the fixed population sizes and it achieved this with the additional benefit of automatically setting a suitable population limit.

It clearly performs better than setting the population limit to 1500; this is likely because the population limit that the automatic variable limit settled on was different (higher) than the 1500 fixed limit ranging from 1583 to 1595 and this was a more effective limit.

The mean of the final population limit from the 10 runs of the automatic variable limit was 1588 and when the experiment was repeated with the population limit fixed at 1588 the difference between the fixed and variable rate of finding optimums was much smaller with only one out of the ten runs failing to find all the optimums.

Arguably it performed better in one to one comparison of the two experiments because it found the same optimums but on average it found them earlier in the generation runs, as shown in Table 27. Also the complete run of all the generations is quicker with the fixed population limit as the initial unlimited population generations of the automatic variable limit GA result in larger populations which require more dominance evaluations.

However, this does not take into account the many experiments required to determine that 1588 (in this example) is a good population limit. The important detail here is that using the automatic variable populations limit eliminates the need to for the population limit to be manually set after a wasteful trial and error experiment.

Table 26 Results of automatic variable population limit experiments compared to fixed population limit of 1500 and 1588 showing the percentage of runs that discovered the optimum solutions for the weight constraints.

Weight	Fixed 1500 limit	Automatic variable limit	Fixed 1588 limit
191	100.000%	100%	100%
190	100.000%	100%	100%
189	100.000%	100%	100%
188	100.000%	100%	100%
187	100.000%	100%	100%
186	100.000%	100%	100%
185	100.000%	100%	100%
184	100.000%	100%	100%
183	100.000%	100%	100%
182	70.000%	100%	90%
181	100.000%	100%	100%
180	100.000%	100%	100%
179	100.000%	100%	100%
178	100.000%	100%	100%
177	100.000%	100%	100%
176	100.000%	100%	100%
175	100.000%	100%	100%
174	100.000%	100%	100%
173	100.000%	100%	100%
172	100.000%	100%	100%
171	100.000%	100%	100%
170	100.000%	100%	100%
169	100.000%	100%	100%
168	100.000%	100%	100%
167	100.000%	100%	100%
166	100.000%	100%	100%
165	100.000%	100%	100%
164	100.000%	100%	100%
163	100.000%	100%	100%
162	100.000%	100%	100%
161	100.000%	100%	100%
160	100.000%	100%	100%
159	100.000%	100%	100%
Min	**70.000%**	**100%**	**90%**
Mean	**99.091%**	**100%**	**99.697%**
Max	**100.000%**	**100%**	**100%**

Table 27 Results of automatic variable population limit experiments compared to fixed population limit of 1500 and 1588 showing the percentage of the total number of generations required to find the optimums.

Weight	Fixed 1500 limit	Automatic variable limit	Fixed 1588 limit
191	6.229%	7.297%	5.312%
190	5.247%	6.364%	3.639%
189	3.155%	6.521%	3.569%
188	3.118%	6.043%	3.231%
187	4.038%	5.451%	3.906%
186	4.614%	9.091%	4.633%
185	8.027%	8.698%	5.719%
184	4.373%	8.419%	4.370%
183	5.465%	7.703%	4.685%
182	41.057%	22.447%	25.761%
181	5.299%	7.949%	4.427%
180	4.941%	7.819%	4.385%
179	3.337%	6.130%	3.535%
178	3.958%	6.835%	4.386%
177	4.332%	8.847%	4.560%
176	4.038%	8.089%	4.275%
175	6.759%	8.315%	4.840%
174	3.390%	6.515%	3.624%
173	3.674%	8.815%	4.200%
172	4.169%	8.677%	4.192%
171	3.808%	8.307%	4.456%
170	3.595%	7.541%	4.052%
169	4.182%	9.686%	4.646%
168	3.822%	7.864%	3.947%
167	3.825%	8.203%	4.622%
166	3.684%	7.350%	4.395%
165	3.914%	8.541%	4.771%
164	3.486%	8.087%	3.686%
163	4.123%	7.565%	4.505%
162	8.358%	10.502%	6.461%
161	3.773%	8.574%	4.213%
160	8.807%	10.664%	6.658%
159	3.949%	7.923%	4.356%
Min	**3.118%**	**5.451%**	**3.231%**
Mean	**5.714%**	**8.389%**	**5.091%**
Max	**41.057%**	**22.447%**	**25.761%**

5.3.10 NSGA-II dominance level algorithm comparison

For comparing the effectiveness of the standard NSGA-II dominance algorithm with the persistent non dominated populations algorithm proposed in this thesis, the code was instrumented to count the number of times the evaluate dominance function was called. The initial population and random seed of the algorithm were set so that the GA would produce exactly the same solution set, just using different methods to determine dominance level.

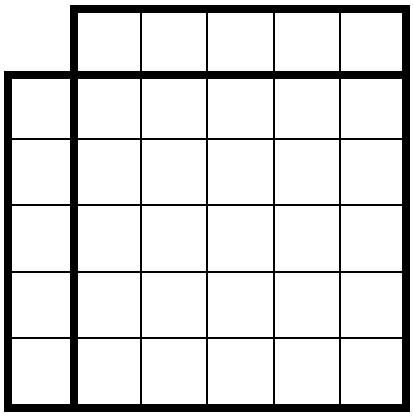

Figure 39 The original NSGA-II algorithm requires n^2 evaluations.

The standard algorithm described in pseudo-code in Deb *et al.*'s 2002 paper has the following external loop:

```
For each p Э P
{
        For each q Э P
        {
         …
        }
}
```

This loop nesting compares every individual against every other individual for dominance. This means that a population of size n requires n^2 dominance evaluations, as shown in Figure 39.

This can be improved upon because when you establish whether p dominates q you also establish whether q dominates p. As shown in Figure 40 this reduces the number of evaluations required by a factor of 2:

—

The persistent population structure described in this thesis was compared to the modified $n^2/2$ algorithm described above.

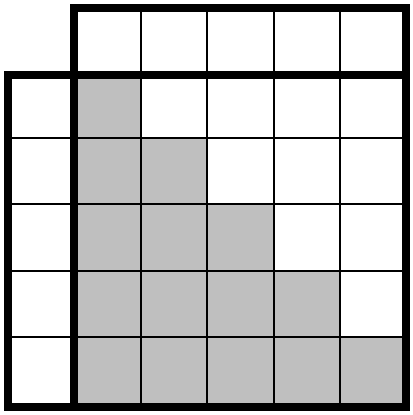

Figure 40 The grey shaded areas are unnecessary evaluations as they are duplicates of the unshaded areas. This gives $n^2/2$ evaluations.

In order to test the effect of the persistence only an additional experiment was run where the nondominated population layers were regenerated each generation to allow it to be compared to the same settings but with persistence activated. Finally the settings of the algorithms were set to the best performing options from the previous experiments:

- Child population size – 150.
- Focused within constraints.
- Fixed length encoding.
- Single long run of 105600 generations.
- Variable automatic main population limit.

Both the pure-elitist and multiple dominance layers options were tested and the results are shown in Table 28.

The results show that the fully modified dominance level algorithm including a persistent datastructure performed significantly fewer evaluations than the standard algorithm (just 3.3%), when set to be pure-elitist.

Table 28 Comparison of the number of dominance evaluations required by standard NSGA-II and modified versions.

	Number of dominance evaluations		
Pure-elitist	**Standard NSGA-II**	**Modified (no persistence)**	**Modified (persistence)**
True	191,121,000,000	168,329,000,000	6,280,710,000
False	194,591,000,000	170,282,000,000	8,041,400,000

The standard algorithm is designed to specifically deal with the layers of dominance, i.e. when the population is not pure-elitist, but while the difference between the effort of the two algorithms narrows when not pure-elitist, the gap is still very large (4.1%)

It is the persistence of the population in the modified algorithm that provides the biggest improvement, but even when the persistence is 'turned off', and the data-structure is regenerated each generation, the number of evaluations required is 88.1% of the standard algorithm when set to pure-elitist.

5.3.11 Ageism

When making the comparison of the original NSGA-II dominance evaluation algorithm and the persistent population structure described in this thesis, it was necessary to make sure that all of the decisions made by the algorithms for creating new individuals were kept the same. This allowed a direct comparison of the algorithms by ensuring that they worked on the same dataset.

In order to achieve this, an extra comparison step was added to the individual dominance check for the population sorting. The individuals in the population are sorted first by dominance level, then by feasibility, then by

crowding density, then (as an additional step) ties are broken by ordering by creation ID.

The creation ID is a unique identifier given to each individual when it is created. The identifier is assigned from an incrementing count of all individuals that have been created. This can abstractly be described as a birth time where individuals created at the start of the GA will have a low creation ID and individuals created at a later generation will have a higher creation ID.

An experiment was devised to establish whether the preference, in this final step, of young (low creation ID) or old (high creation ID) has a significant effect.

Table 29 shows that an ageist strategy favouring younger individuals performs significantly better than giving preference to older individuals.

This is understandable as favouring older individuals would tend to cause the algorithm to stagnate in regions of the search space that have already been searched (potentially extensively). Favouring younger individuals would seem to encourage search in relatively unexplored regions.

Table 29 Comparison of preference of young individuals versus preference of older individuals.

	Percentage of runs optimum found		Percentage of generations for mean case	
Weight	**young**	**old**	**young**	**old**
191	100%	50%	7.297%	72.004%
190	100%	80%	6.364%	47.968%
189	100%	100%	6.521%	52.660%
188	100%	90%	6.043%	45.785%
187	100%	100%	5.451%	49.728%
186	100%	0%	9.091%	no optimums
185	100%	60%	8.698%	72.976%
184	100%	40%	8.419%	86.191%
183	100%	50%	7.703%	51.924%
182	100%	0%	22.447%	no optimums
181	100%	20%	7.949%	92.187%
180	100%	40%	7.819%	66.417%
179	100%	90%	6.130%	44.602%
178	100%	70%	6.835%	43.105%
177	100%	40%	8.847%	33.034%
176	100%	80%	8.089%	63.262%
175	100%	0%	8.315%	no optimums
174	100%	90%	6.515%	55.699%
173	100%	70%	8.815%	41.086%
172	100%	80%	8.677%	65.546%
171	100%	70%	8.307%	43.851%
170	100%	90%	7.541%	35.115%
169	100%	70%	9.686%	53.019%
168	100%	90%	7.864%	39.889%
167	100%	80%	8.203%	49.764%
166	100%	80%	7.350%	35.933%
165	100%	90%	8.541%	58.712%
164	100%	90%	8.087%	47.471%
163	100%	80%	7.565%	46.572%
162	100%	40%	10.502%	83.523%
161	100%	90%	8.574%	42.260%
160	100%	40%	10.664%	61.426%
159	100%	90%	7.923%	48.402%
Min	**100%**	**0%**	**5.451%**	**no optimums**
Mean	**100.000%**	**65.152%**	**8.389%**	**49.397%**
Max	**100%**	**100%**	**22.447%**	**92.187%**

5.4 Results summary

This chapter began by establishing that using a purely random approach to search, results in very poor solutions. The mean MPI for a purely random search was -273.4%. When considering a search space of any realistic size, an optimisation algorithm is necessary.

The penalty-based GA, implemented as a slight variant to Coit and Smith's (1996b) algorithm, achieved an MPI of -1.23%. The use of HiP-HOPS to provide the reliability analysis introduced a significant computational overhead, however, the added time required to perform the optimisation did not render it infeasible. The quality of solutions found by the penalty-based GA were a vast improvement on random search, however, there is scope for improvement as less than half of the optimums were discovered.

The results of the Pareto-based experiments showed that the inability to intuitively set a good hypergrid size for the PESA-II GA represented a significant problem for its use. Further experimentation did not achieve good solutions (when compared to both the penalty based GA and NSGA-II) even after a good hypergrid size was found using costly trial and error.

The NSGA-II based algorithm did produce excellent results. The best results were achieved under the following conditions:

1) The main population size limit is set using a variable algorithm that allows an unlimited population early in the search. After a preset number of generations the population size is then limited to the number of feasible individuals in the population.

2) Given a set number of solution evaluations, it is better to generate a small number of children solutions each generation, and have more

generations, than to generate a large number of children, and have fewer generations.

3) Where sensible constraints for the objectives are known, it is better to focus the search using these constraints. It seems possible to run the search unconstrained for a preset number of generations and then set constraints based on the solutions found so far.

4) It is always better to retain undominated, feasible solutions in an archive population so that they aren't lost from the population. This does, however, carry an overhead from needing to check solution dominance for this extra population.

5) The fixed encoding did perform marginally better across the board than the tree encoding. However, the benefits of generally using a tree encoding are substantial as they allow much more complex systems to be optimised due to its flexibility. A number of possible improvements to the handling of the tree encoding were proposed for future work.

6) NSGA-II is designed to allow dominated solutions to remain in the population to retain diversity. However, where it was forced to be pure-elitist (by discarding all dominated solutions) it performed better in these experiments.

7) Given a set number of solution evaluations, the algorithm performed better when allowed to run for a single long run, rather than taking the best solution from ten runs. This is both in terms of the quality of the solutions produces, and the percentage of the total generations required to find them.

8) The use of a persistent non-dominated population structure in NSGA-II offers substantial computational effort reduction compared to the original algorithm.

The NSGA-II variant making use of these settings found all of the optimum solutions for the 33 benchmark weight constraints giving it an average MPI of 0.0%, significantly outperforming the penalty-based GA's -1.23% MPI (Coit and Smith, 1996b). In fact the NSGA-II variant developed in this thesis is the first meta-heuristic to achieve this 'perfect' score with the benchmark problem, as it had only been achieved before using precisely constrained mathematical methods (Onishi *et al,* 2007). Additionally, the GA achieved this whilst remaining more flexible, giving the benefit of likely achieving similar results with general models of varying complexity, with little (if any) modification. The previous best meta-heuristic was Tabu-search achieving an MPI of -0.263% (Kulturel-Konak *et al,* 2003) but that did not find the optimum solution in more than 20% of the cases.

The original NSGA-II algorithm was significantly improved for this thesis, and the literature review showed it to be a strong candidate before this. The new sorting algorithm results in just 3.3% of the comparisons of the original algorithm. A further improvement comes from implementing a novel age discrimination for solutions, where newer solutions are preferred to older solutions. This resulted in the success of the algorithm in a range where preferring new solutions found all the optimums and preferring old solutions found the optimums in only ~65% of cases.

6. Fuel Oil System Case Study

In this chapter, the developed combination, of automatic reliability analysis and a multi-objective optimisation algorithm, was applied to a ship fuel oil design problem. Previously, an expert engineer had used HiP-HOPS to analyse multiple design configurations. The design modification, and evaluation of the analysis results was carried out manually by the engineer in order to select a design variant that exceeded the reference solution.

The original study did not make use RBDs for analysing the reliability of the system design. In the original study, the principle overhead for the process is the manual configuration of the system for analysis. There was nothing to be gained by using a simplified analysis, such as that provided by RBDs, over a more complex and complete analysis technique, such as HiP-HOPS, as any performance improvement was swallowed up by the overhead and became insignificant. Where the modelling allowed for by the more complex analysis techniques is more complete and realistic, and this can be achieved within a reasonable time then it is logical to make use of it.

In a deliverable for the SAFEDOR (Design, Operation and Regulation for Safety) FP7 project (Erich Rüde, 2007), a fuel oil service system (see Figure 41) for a cargo ship is considered. When the fuel oil system fails, there is a loss of engine propulsion that can lead to the ship becoming grounded as a result of drifting.

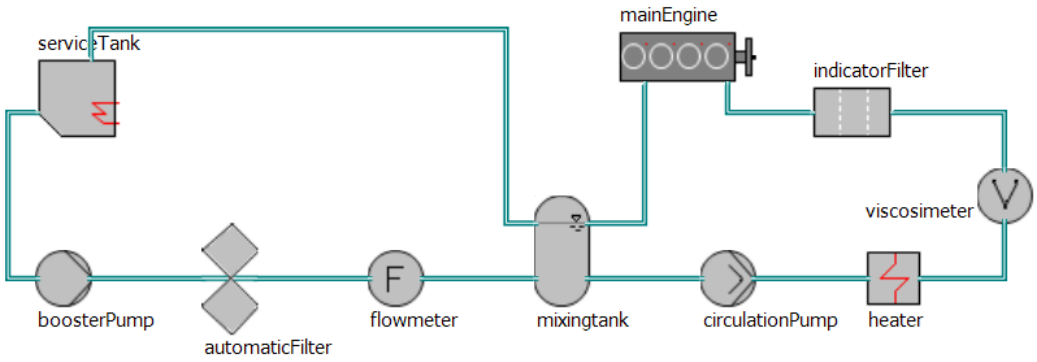

Figure 41 Base solution with no component redundancies. (source: Rude, 2007)

The cost and failure rates of the components of the system can be found in Table 30. Data for the main engine is not included because this is immutable in the system and therefore not considered.

Table 30 - Table of cost and failure rate attributes for components of the fuel oil system.

Components	Cost	Failure Rate
Indicator filter	1500	5.0E-7
Viscosimeter	2500	2.5E-6
Pre-heater	2000	6.7E-6
Circulation pump	6000	3.2E-5
Mixing tank	2000	1.6E-5
Flow meter	2000	1.0E-5
Automatic filter	2000	1.0E-5
Booster pump	5000	3.2E-5
Service tank	1500	1.6E-5

In an attempt to find the optimum solution, the base configuration with no redundancies was analysed for cost and unavailability. Then a single redundancy was added to the system and the system was re-analysed. This process was repeated, each time adding a redundancy to the next subsystem. In all 12 system variations were analysed.

One of these configurations is known to be acceptable for this system, shown in Figure 42. This system contained redundancies in the two pump subsystems and in the heater. Figure 43 shows the Pareto front formed by these 12 solutions, where the base system and the reference system are indicated by the arrows.

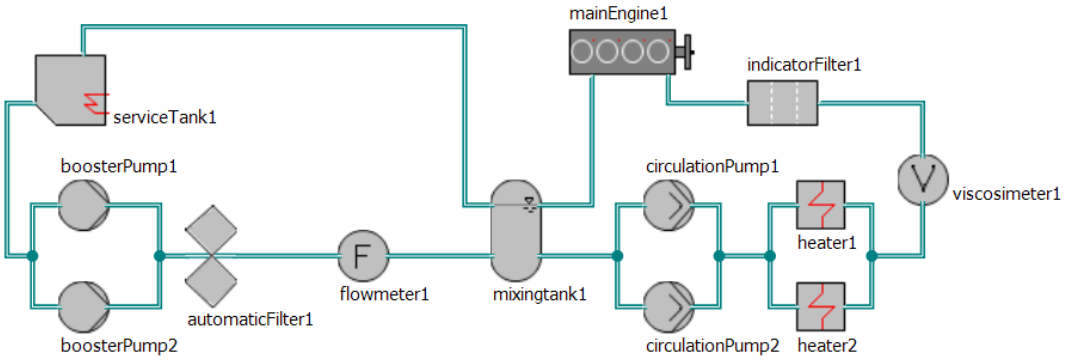

Figure 42 Reference solution has redundancies with the booster pump, circulation pump, and heater. (source: Rude, 2007)

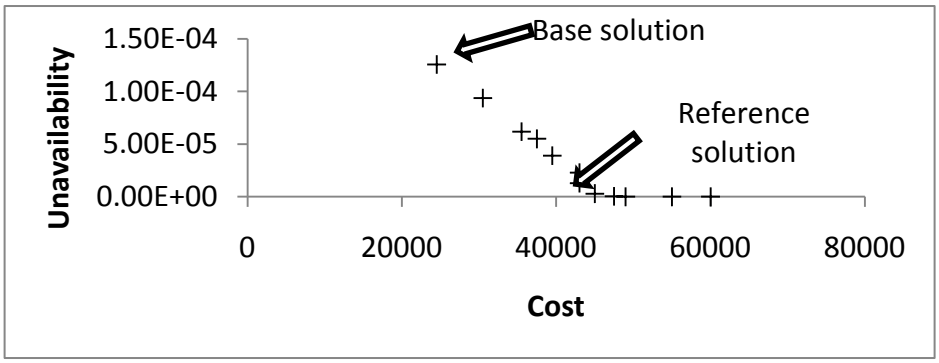

Figure 43 Pareto front of manually created and analysed solutions with base solution and reference solution indicated.

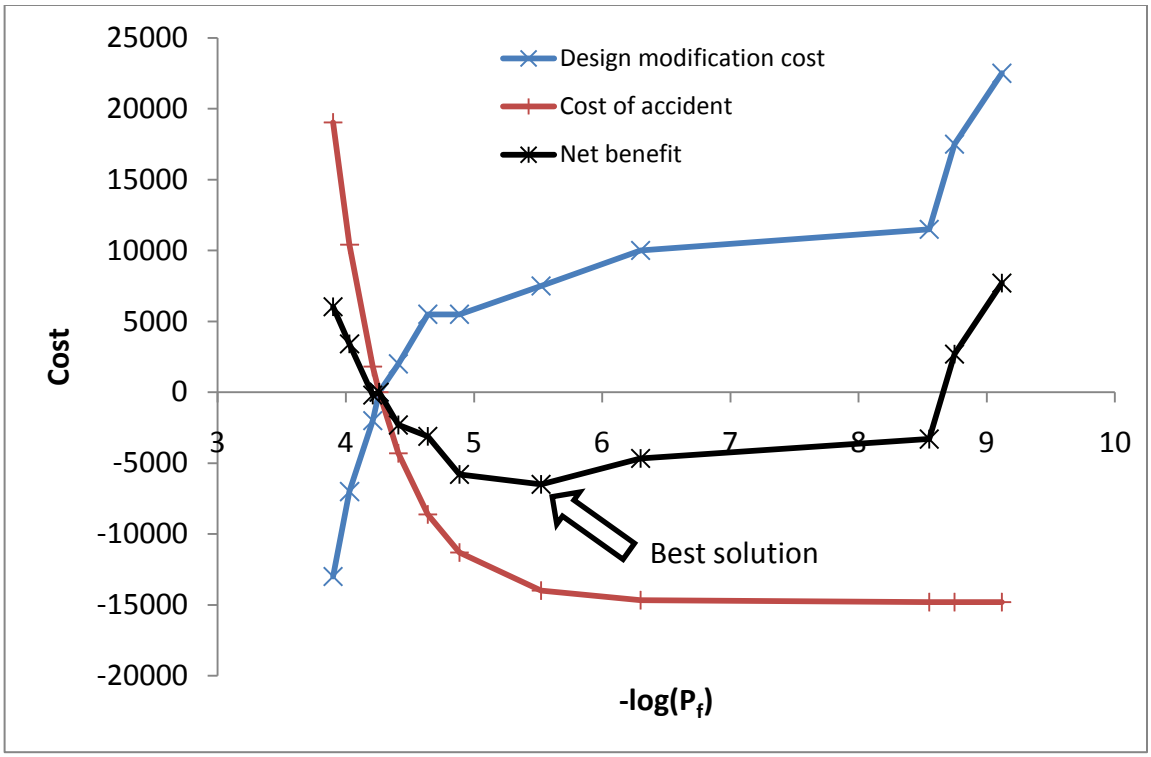

Figure 44 Cost benefit analysis from manual study; solution with highest net benefit indicated. (source: Rude, 2007)

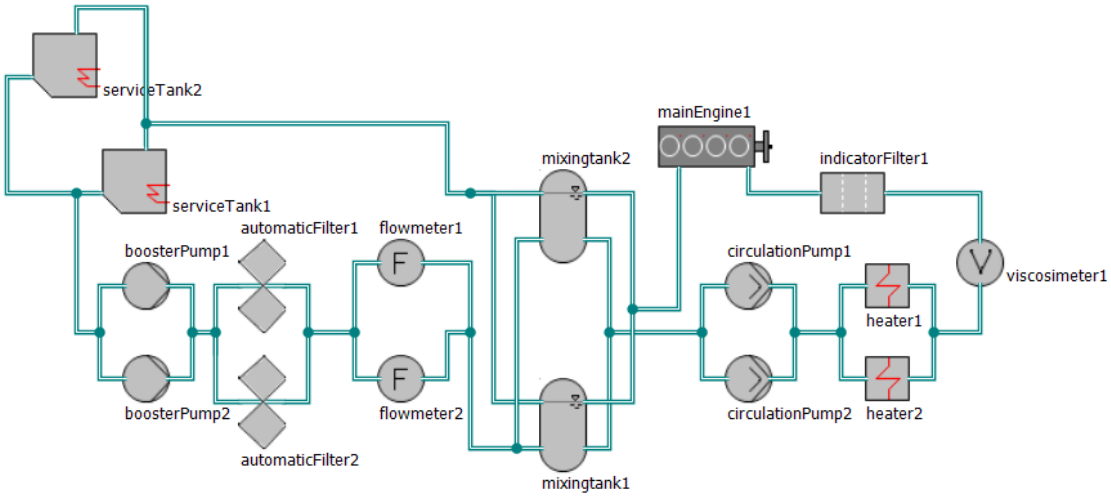

Figure 45 Best solution following manual analysis. (source: Rude, 2007)

The manual analysis did not consider alternative components, only differing levels of redundancy for each component; zero, one, or two redundant

components for each of the nine components (not including the engine). Even given these constraints, and therefore a relatively small search space of 19683 combinations possible, it was not feasible for the designer to consider many systems; only 12 were considered.

The combined process of HiP-HOPS and an optimisation algorithm from this thesis was applied to this fuel oil system.

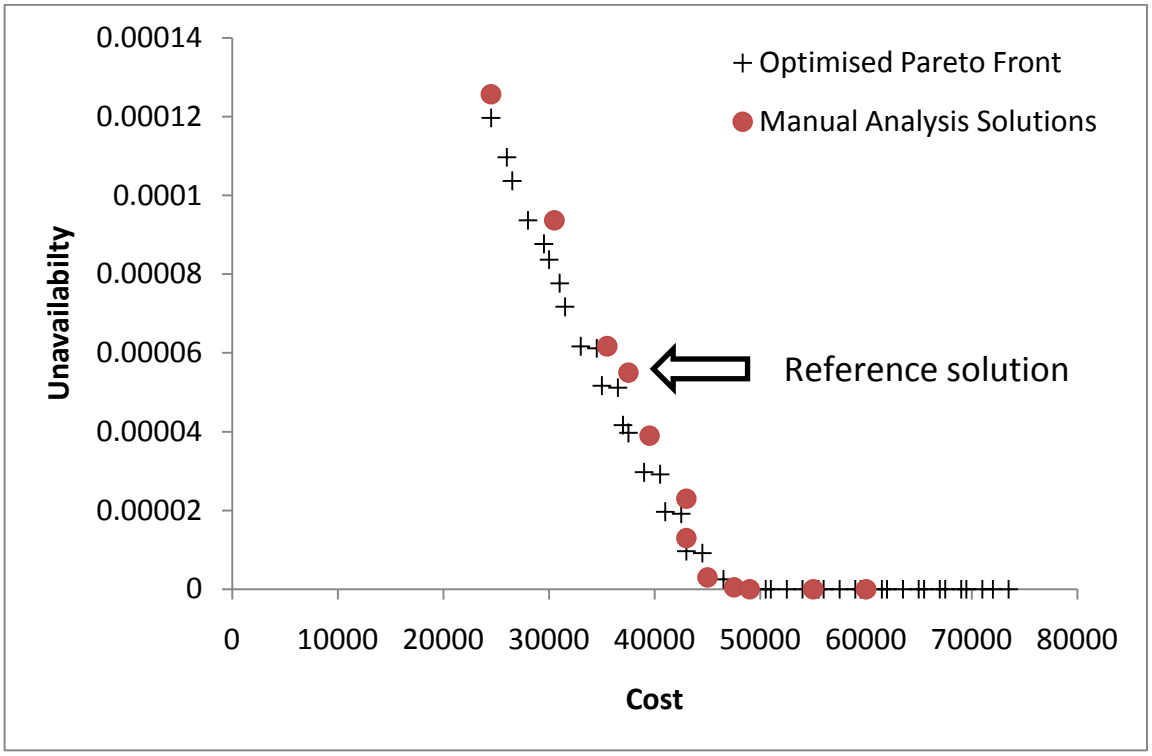

Figure 46 Pareto front found by optimisation algorithm; manual analysis solutions also shown. Note many of the manual solutions are dominated by the optimisation results.

Following 50 generations of optimisation, 46 undominated trade-off solutions were found. The Pareto front is shown in Figure 46. The manual analysis solutions from the previous study are also plotted. It is worth noting that many cases, including the reference system, are actually dominated by solutions found by the optimisation algorithm.

The cost benefit analysis was applied to the undominated solutions from the optimisation, the results of which are shown in Figure 47 with the optimum solution indicated with an arrow.

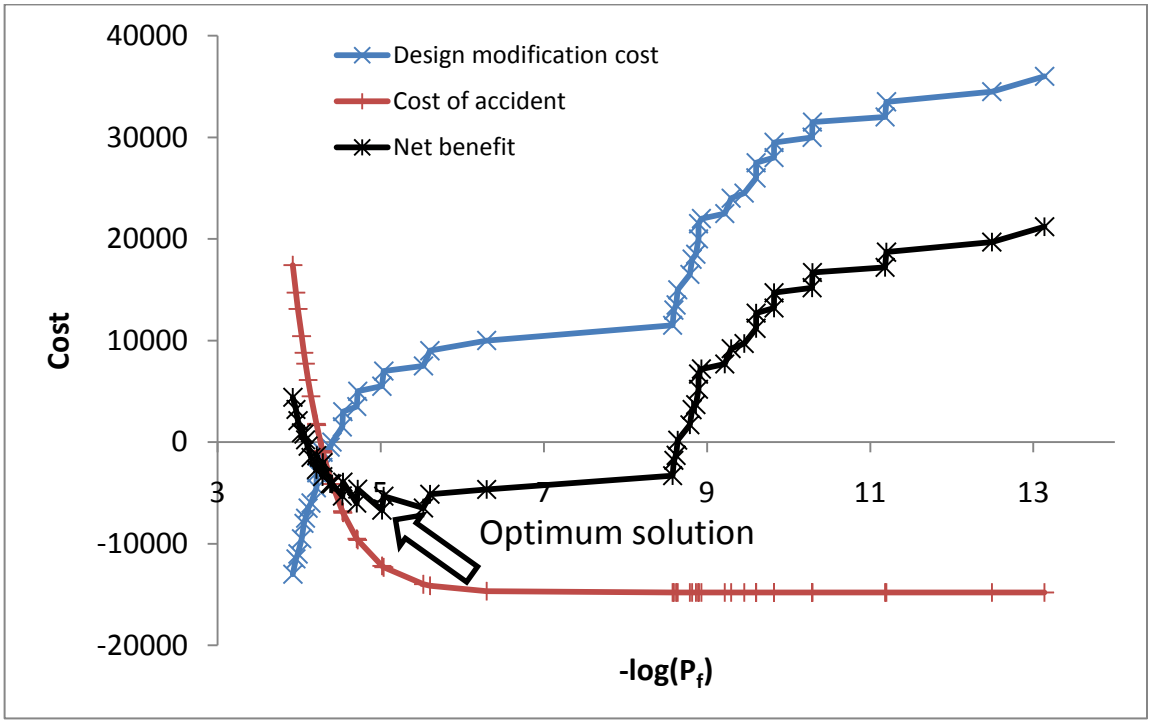

Figure 47 Cost benefit analysis from optimisation results; solution with highest net benefit indicated.

The optimum solution found by the optimisation algorithm is shown in Figure 48. It has a slightly higher unavailability (0.0000097) to the best solution found by the manual analysis, but the component cost is 2000 less at 43000. This is achieved by using only a single heater, whereas the manual analysis optimum solution had a single redundancy on the heater component.

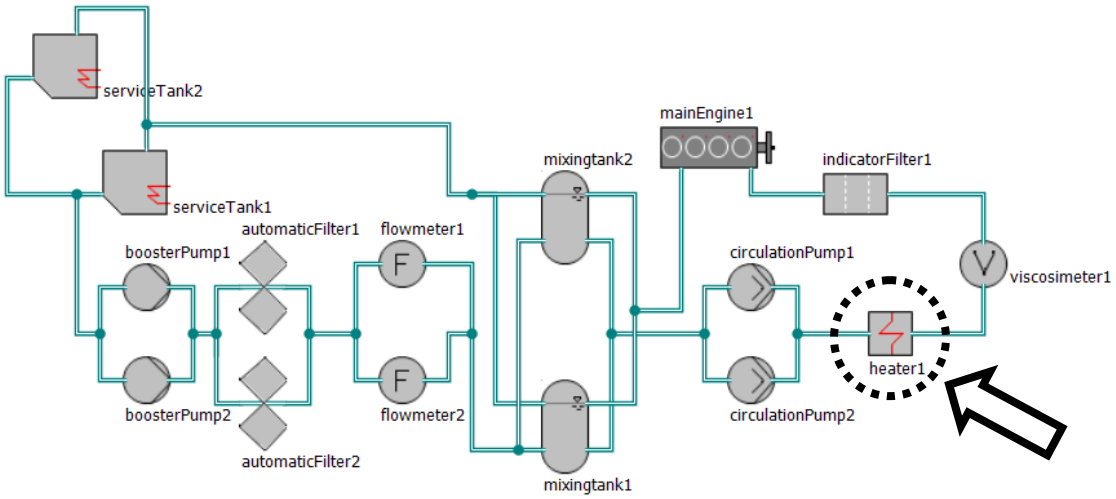

Figure 48 Optimised solution uses only one heater; the best solution from the manual analysis used two heaters.

Using manual analysis, it was already impractical to consider many variations of the system to discover an optimum configuration. This problem is greatly increased when alternative components are considered for each of the subsystems.

Table 31 - Table of cost and failure rate attributes for components of the fuel oil system with 3 alternatives.

	Alternative 1		Alternative 2		Alternative 3	
Components	**Cost**	**Failure Rate**	**Cost**	**Failure Rate**	**Cost**	**Failure Rate**
Indicator filter	1500	5.0E-7	2500	2.0E-7	3222	1.0E-7
Viscosimeter	2500	2.5E-6	3178	1.0E-6	3814	5.0E-7
Pre-heater	2000	6.7E-6	2505	5.0E-6	3956	1.0E-6
Circulation pump	6000	3.2E-5	13380	2.0E-5	18000	7.0E-6
Mixing tank	2000	1.6E-5	2963	8.0E-6	4444	2.0E-6
Flow meter	2000	1.0E-5	3000	1.0E-6	4444	5.0E-7
Automatic filter	2000	1.0E-5	2647	5.0E-6	3529	1.0E-6
Booster pump	5000	3.2E-5	10682	2.0E-5	12500	5.0E-6
Service tank	1500	1.6E-5	1957	5.0E-6	2739	1.0E-6

Allowing 3 functionally equivalent alternative components for each of the 9 subsystems increases the search space size to 20,661,046,784. This makes

it even more unlikely that a manual process would discover the optimum configuration.

Figure 49 shows the Pareto front of 366 undominated tradeoffs found by the optimisation algorithm after 4000 generations.

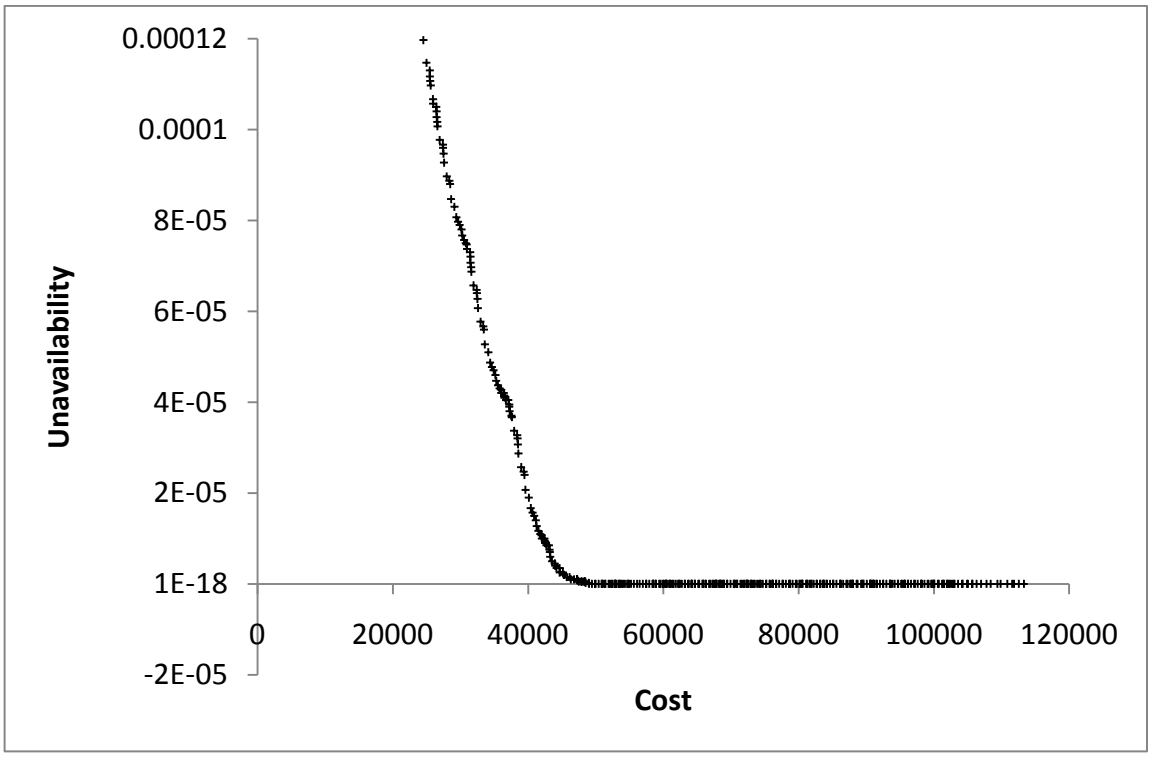

Figure 49 Pareto front of solutions from optimisation algorithm where alternative components are available for each component.

As before, a cost benefit analysis was carried out on the solutions found on the Pareto front. The results of that analysis are shown in Figure 50; the chart has been zoomed in to show detail around the optimum area.

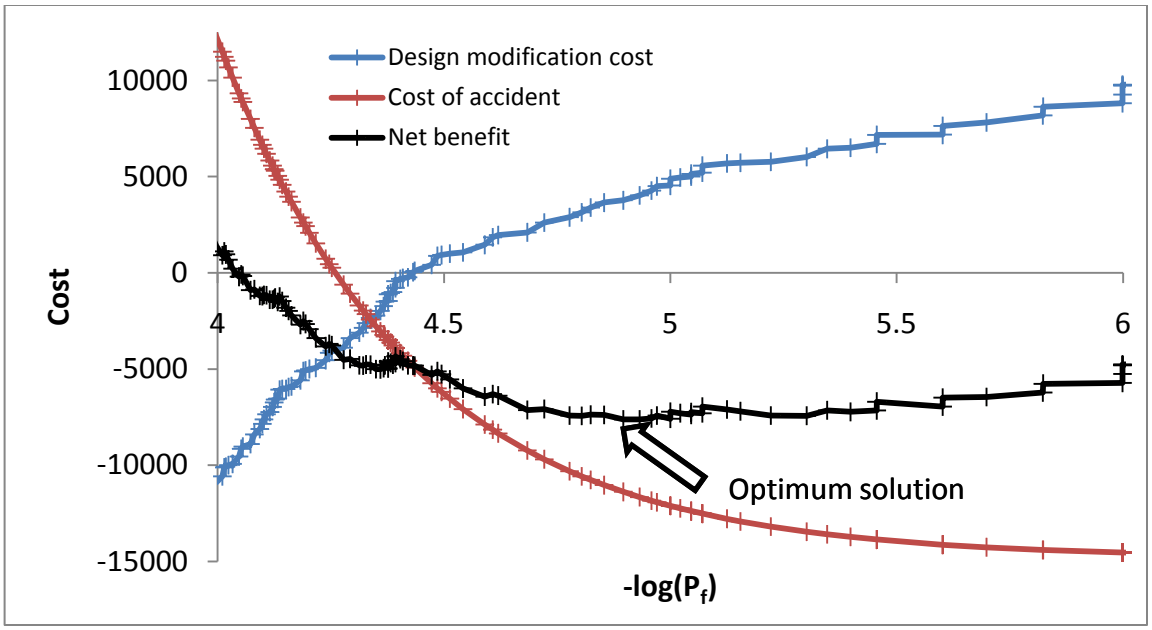

Figure 50 Cost benefit analysis of solutions from optimisation algorithm where alternative components are available for each component. Optimum solution is indicated.

Being able to consider alternative components for each of the subsystems allows a new optimum configuration. This is shown in Figure 51, where the automatic filter and the flow meter have a single component, alternative version 3 and 2 respectively, in place of a redundant configuration.

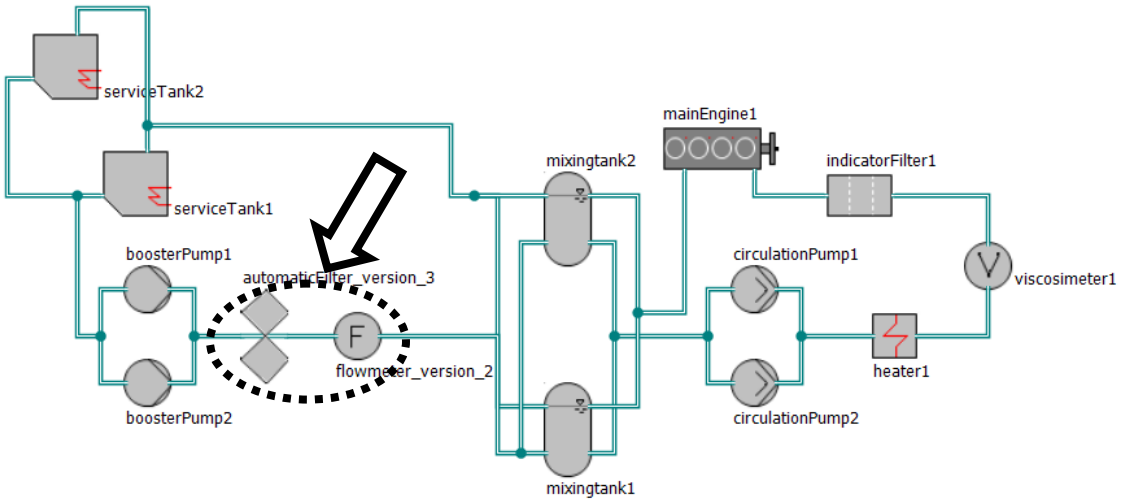

Figure 51 Use of alternative automatic filter and flow meter components enables new optimal design.

The system's unavailability of 0.0000117 is worse than the previous two best solutions but the component cost is a further 1471 less.

6.1 Case study summary

The case study clearly shows that combining the use of an optimisation algorithm and an automatic safety analysis technique improves on the results that can be achieved using safety analysis alone.

The automated safety analysis tool enabled the user to evaluate several potential solution system variants. This is more that would have been achievable in the same time using manual analysis techniques. However, the number of solutions evaluated is still very small compared to the size of the search space. This makes it very unlikely that any of the evaluated solutions would be optimum.

This is true when the problem is constrained to keep the search area small, as when only one component type is allowed; it becomes even more the case when the search size is increased.

The genetic algorithm found a solution to the constrained case that was 4.4% less costly than found by the manual analysis, leading to a 3% improvement in net benefit. Increasing the size of the search space by allowing alternative component choices gave the GA a greater advantage finding a solution with a 17% improvement in net benefit.

7. Conclusions

In the first chapter of this thesis the following hypothesis was stated:

It is conceptually possible and technically feasible to achieve architectural optimisation using a combination of emerging model-based safety analysis techniques and meta-heuristics, assisting in the exploration of large design spaces for optimal tradeoffs between cost and reliability.

A number of objectives were defined that needed to be met to support the above hypothesis. These objectives are repeated below, each with a discussion summarising how they were met.

1. **Choose from among the state of the art in safety and reliability analyses a technique that can provide fast, scalable, and automatic model-based evaluation of safety and reliability.**

Following a review of the literature of safety analysis, HiP-HOPS was selected. The core of a HiP-HOPS analysis is the synthesis of system fault trees from the component-level failure behaviour expressions. The generated system fault trees can then be further analysed using traditional FTA techniques.

The first criterion for selection is that it should be automatic without human interaction required. This is true of HiP-HOPS as both the synthesis of the system fault trees and the subsequent FTA is programmatically achieved.

The speed criterion comes from the need to iterate the analysis many times during the optimisation. The technique needs to be fast, or at least fast enough for an optimisation to complete in reasonable time. To a certain extent this is tied in with the need for scalability. The ability to apply the analysis to larger, more complex models enables these same models to be

optimised. If the speed of the analysis degraded rapidly as the size of the model increased then that would restrict its use.

HiP-HOPS scales well with model size as the synthesis of the system fault trees does not suffer from combinatorial explosion, which is a particular problem for the fault injection technique (CHEN, D J et al., 2008). HiP-HOPS is able to analyse large models with hundreds of components in multiple level hierarchies in sub-second times.

Improving the speed of the algorithms is covered in the next objective.

2. **Examine the possibility of further minimising the computational effort required to perform safety and reliability analysis through performance enhancements to the chosen technique.**

The analysis of the system fault trees generated by HiP-HOPS uses traditional FTA techniques. As fault trees have been used for many years now there has been much research aimed at improving the performance of the techniques and algorithms.

Several methods from the literature were applied, including fault tree modularisation and fault tree contracting, each generating significant improvements to the performance of the analysis phase. Cut set pruning also offered a speed increase, but at the expense of results accuracy.

A new method was described that maintained the cut sets from the fault trees in an FMEA-like catalogue. This significantly reduced the number of redundancy checks required when determining minimum cut sets.

A particular problem was identified with the benchmark model where the nested AND logic from the parallel redundancy would generate a very large fault tree. A new technique dubbed Analysynth combines the analysis and the synthesis phases of HiP-HOPS to flatten and minimise the fault tree as it

is being generated. This removes the nesting during the synthesis process and solves the issue. The technique may also be useful in the general case as minimising small fault trees is disproportionately easier and quicker than large fault trees. This was not explored.

The combination of the different performance enhancements led to the speed increasing by a factor of over 3000, compared to unenhanced performance.

3. **Choose from among the state of the art a strongly performing genetic algorithm that can conduct the optimisation.**

GAs were chosen as the general optimisation technique to use as they are very flexible and can be applied to problems where a mathematical model is not known. Many real world optimisation problems have multiple conflicting objectives and so the current state-of-the-art MOEAs were reviewed and NSGA-II and PESA-II were selected for further investigation.

In addition to this an existing GA solution was implemented from the literature to provide a comparison.

4. **Examine the possibility of minimising the computational effort required for the chosen genetic algorithm to perform the optimisation.**

The NSGA-II algorithm allows both undominated and dominated solutions to exist in the population. Solutions are favoured for selection by their dominance ranking. The algorithm in the original paper recalculated the dominance ranking every generation, even though the changes would be relatively small.

In order to reduce this, a persistent population data structure was implemented so that solutions were maintained in a population specific to their

dominance ranking. Since a solution would only be displaced from its position by a new solution, dominance checking only need be carried out on small numbers of new solutions.

In fact the method of using the population structure and cascading the dominated individuals reduced the number of dominance checks over the original NSGA-II algorithm, even without persistence of the structure from generation to generation. When the structure was made persistent it used just 4% of the dominance checks of the original algorithm.

This was further improved to just 3% by making the algorithm pure-elitist and removing the dominated individuals altogether. This was even found to generally improve the solutions found in the benchmark experiments.

5. **Maximise the quality of the alternative solution designs found by the genetic algorithm.**

Several aspects of the GAs were investigated including:

- the solution encoding used,
- whether the algorithm was pure-elitist or not,
- whether a solution archive was present,
- whether a constraints focussing algorithm was applied,
- the effect of population size,
- is it wrong to be ageist?
- whether selecting the best individual from several short runs is better than a single long run.

In addition to these variables the PESA-II algorithm also has a hypergrid size parameter. Experimentation showed that the effectiveness of the algorithm was dependent on selecting the correct size and the selection of the correct size was not an intuitive process. In addition to this, the solutions

found by the PESA-II algorithm compared poorly to those of the other algorithms even when an apparently good hypergrid size was selected.

The NSGA-II algorithm variant performed well. It discovered all the optimums in the benchmark example (as established by the exact ISC method). The settings for the parameters that achieved the best solutions for the benchmark were:

- Fixed length encoding
- Pure-elitist
- Include a solution archive
- Focus the search using constraints
- Vary the population size automatically using the proposed algorithm
- Preferring 'younger' solutions during selection
- Having a single long run rather than selecting the best from 10 shorter runs.

One item particularly worth noting is the encoding type. There was a marginal improvement over the experiments found when using the fixed length encoding. However, one problem with the fixed length encoding is that it is very rigid. While this makes little difference to the benchmark example, since it does not have a significant component hierarchy, it is felt that in general a tree-based encoding would be more flexible and would allow arbitrary hierarchies to be manipulated.

It is also interesting that, on average, the optimum solutions were found after a relatively low percentage of the total generations. This suggests that

the GA can be run for a reduced time, which is good for tight design schedules.

6. **Apply the proposed optimisation approach to case studies in order to validate feasibility and evaluate the scalability and usefulness of the approach.**

The proposed approach was tested on a well used benchmark example from the literature. It successfully found the optimum solutions to the problem, which indicates promise in the approach.

Clearly it is feasible, at least on a basic level, to marry together state-of-the-art automatic safety analysis techniques with state-of-the-art evolutionary algorithms.

When applied to the cost benefit analysis of a fuel oil system, the combined algorithm found solutions that surpassed those found by a previous approach that included only the manual analysis of a few alternatives.

RBDs have been used in the past to provide a simplified reliability model of systems being optimised. The beneficial trade-off for this simplification is that the analysis of this simplified model can be performed relatively quickly. This performance aspect would make it ideal for use in optimisation applications, but there remains the problem of the model simplification.

Firstly, because the RBD is a simplified model of the system, it is necessary to translate between the full system model and the RBD (and back again). The translation process requires extra design effort, and it is an opportunity for errors to be introduced. Secondly, even if the first issue was resolved there would remain the inherent problem (the sacrifice that gives it its performance benefit), that it is a simplified model.

It is desirable for the model being used to be as accurate a representation of reality as possible. Certainly, for one-off analyses this rules out RBDs as more complex and complete modelling tools are available and the time constraint is largely removed.

The many iterations of analysis required by optimisation do add a time-efficiency constraint as the algorithm cannot be allowed to run indefinitely; the results would arrive far outside the permitted design period. The work in this thesis indicates that it may be possible to use more accurate design models than RBDs without the optimisation time becoming prohibitively long.

The scalability of the approach remains an open question. The size of the search space in the benchmark problem is large, but it is a relatively small system. Further testing on larger systems is required. This and other future work is discussed in the following section.

8. Future Work

To paraphrase the comic Dara Ó Briain: *If science knew everything it would stop*. With that in mind, there remain several avenues of particular interest for future research in this area.

The experiments carried out for this thesis focussed on a literature benchmark system in order to make comparisons to previous approaches. However, the principal advantage of integrating HiP-HOPS with optimisation algorithms is that it is applicable to complex engineering models, and not limited to simple series-parallel systems.

There are several features that future testing could expand upon:

1) Optimising systems with multiple (interacting) failure modes. The benchmark system was limited to just success or failure.

2) Optimising systems using a variety of reliability improving strategies such as majority voters and standby recovery systems. The benchmark system used a simple parallel redundancy.

3) The HiP-HOPS framework presents a variety of evaluation metrics that could supplement reliability as optimisation objectives.

4) Optimising systems that have temporal ordering characteristics, making use of HiP-HOPS Pandora technology.

5) Application of the approach to larger systems to investigate the scalability of the approach when faced with real-world examples.

Although the NSGA-II variant was successfully used in experiments, it would be interesting to apply other state-of-the-art optimisation heuristics such as

SPEA2 and others. Some algorithms are better suited to particular problem types and there may be improvements to be found.

The scalability of the approaches was touched upon above. To this end, there are a couple of ideas that seek to improve the performance of the algorithms but have been left for future improvements. These are Binary Decision Diagrams and parallelisation.

Converting a fault tree into a binary decision diagram representation has been shown in the literature to provide a significant performance benefit by inherently creating minimal (or near minimal) cut sets, thus reducing the number of redundancy checks required ((BARTLETT, L M and Andrews, J D, 2001), (SINAMMON, R M and Andrews, J D, 1997), (RAUZY, A, 1993)). However, the conversion process requires that the basic events of the fault tree are given an ordering, usually based on either their relative location in the fault tree or some kind of importance weighting. Whilst there have been several proposals for generating the basic event ordering, it still remains a problem to be solved as none of the proposals generates a minimal BDD for all fault trees; importantly, when it doesn't, it can make things worse.

With the increasing ubiquity and advancements in multi-core computing it seems that performance in fault tree analysis could be improved by tailoring the algorithms to be multiple-thread compliant. One way to achieve this would be that every time a branch is encountered in the fault tree during analysis, the processing of each child branch could be passed to a separate thread. These multiple threads could be distributed to the different processors to maximise efficiency. Although there is some overhead that is introduced in order to manage the multiple threads and prevent the threads from interfering with each other, at a naive level a fault tree with four major branches could be run on a quad core processor in a quarter of the time required by a single threaded application.

A performance increase should be possible by applying parallel processing techniques to the GA. A number of new solutions are independently created in each generation. These could each be processed in a separate thread running on a separate CPU core. As with the parallelisation of the safety analysis algorithms, the benefit from achieving this is dependent on the fraction of the algorithm that can be parallelised. This was described by Amdahl (1967).

9. References

AMDAHL, G H. 1967. Validity of single-processor approach to achieving large scale computing capability. *In*: *Proceedings of the AFPIPS Conference*. Reston, VA, pp.483-485.

ANTONISSE, J. 1989. A new interpretation of schema notation that overturns the binary encoding constraint. *In*: *Proceedings of the 3rd International Conference on Genetic Algorithms*., pp.86-91.

BARTLETT, L M and J D ANDREWS. 2001. An ordering heuristic to develop the binary decision diagram based on structural importance. *Reliability Engineering and System Safety*. **72**, pp.31-38.

BIEBER, P, C BOUGNOL, C CASTEL *et al.* 2004. Safety assessment with AltaRica. *In*: *IFIP Congress Topical Sessions*., pp.505-510.

BIEBER, P, C CASTEL, and C SEGUIN. 2002. Combination of fault tree analysis and model checking for safety assessment of complex systems. *In*: *Proceedings of the 4th European Depting Conference on Dependable Computing*., pp.19-31.

BOZZANO, M, C JOCHIM, and F TAPPARO. 2006. The FSAP/NuSMV-SA Safety Analysis Platform. *In*: *ECAI 2006*. Riva del Garda, Italy.

BOZZANO, M and A VILLAFIORITA. 2006. The FSAP/NuSMV-SA Safety Analysis Platform. *In*: *ECAI 2006*. Riva del Garda, Italy.

BULFIN, R L and C Y LIU. 1985. Optimal Allocation of Redundant Components for Large Systems. *IEEE Transactions on Reliability*. **R-34**(3), pp.241-247.

CANTÚ-PAZ, Erick. 2002. On Random Numbers And The Performance Of Genetic Algorithms. *In*: *Proceedings of the Genetic and Evolutionary Computation Conference*., pp.311-318.

CHEN, D J, R JOHANSSON, H LÖNN *et al.* 2008. Modelling Support for Design of Safety-Critical Automotive Embedded Systems. *In*: *Lecture Notes in Computer Science: Computer Safety, Reliability, and Security*, Berlin: Springer, pp.72-85.

CHERN, M. 1992. On the computational complexity of reliability redundancy allocation in a series system. *Operations Research Letters*. **11**, pp.309-315.

CIARDO, G and C LINDERMANN. 1993. Analysis of deterministic and stochastic Petri nets. *In*: *Proceedings of the 5th International Workshop on Petri nets and Performance models PNPM'93*. Toulouse, France.

COELLO, C. 2006. Twenty Years of Evolutionary Multi-Objective Optimization: A Historical View of the Field. *IEEE Computational Intelligence Magazine*. **1**(1), pp.28-36.

COIT, D W and A E SMITH. 1996a. Penalty guided genetic search for reliability design optimisation. *Computers and Industrial Engineering*. **30**(4), pp.895-904.

COIT, D W and A E SMITH. 1996b. Reliability optimisation of series-parallel systems using a genetic algorithm. *IEEE Transactions on Reliability*. **45**(2), pp.254-260.

CORNE, D W, N R JERRRAM, J D KNOWLES, and M J OATES. 2001. PESA-II: Region-based Selection in Evolutionary Multiobjective Optimisation. *In*: *Proceedings of the Genetic and Evolutionary Computation Conference GECCO*. Morgan Kaufmann Publishers, pp.283-290.

CORNE, D, J D KNOWLES, and M J OATES. 2000. The Pareto Envelope-Based Selection Algorithm for Multi-objective Optimisation. *In*: *Proceedings of the 6th International Conference on Parallel Problem Solving from Nature*., pp.839-848.

DEB, K, S AGRALWAL, A PRATAP, and T MEYARIVAN. 2000. A fast elitist non-dominated sorting genetic algorithm for multi-objective optimisation: NSGA-II. *In*: Schoenauer et AL, (ed). *Parallel Problem Solving From Nature PPSN VI*, Berlin, Germany: Springer, pp.849-858.

DEB, K, A PRATAP, S AGARWAL, and T MEYARIVAN. 2002. A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*. **6**(2), pp.182-197.

DUTUIT, Y and A RAUZY. 1996. A linear-time algorithm for finding modules of fault trees. *IEEE Transactions in Reliability*. **45**(3), pp.422-425.

ESARY, J D and F PROSCHAN. 1970. A Reliability Bound for Systems of Maintained, Interdependent Components. *Journal of the American Statistical Association*. **65**(239), pp.329-338

FENELON, P and J A MCDERMID. 1993. An integrated toolset for software safety analysis. *Journal of Systems and Software*. **21**(3), pp.279-290.

FENELON, P, J A MCDERMID, M NICOLSON, and D J PUMFREY. 1994. Towards integrated safety analysis and design. *ACM SIGAPP Applied Computing Review*. **2**(1), pp.21-32.

FYFFE, D E, W W HINES, and N K LEE. 1968. System reliability allocation and a computational algorithm. *IEEE Transactions on Reliability*. **R-17**, pp.64-69.

GE, X, R F PAIGE, and J A MCDERMID. 2009. Probablistic Failure Propagation and Transformation Analysis. *Computer Safety, Reliability, and Security*. **5775**, pp.215-228.

GERMAN, R and J MITZLAFF. 1995. Transient analysis of deterministic and stochastic Petri nets with TimeNet. *In*: *Proceeding of the 8th International Conference on Computer Performance Evaluation, Modelling Techniques, and Tools and MMB*. Heidelberg, Germany, pp.209-223.

GRIFFAULT, A, A ARNOLD, G POINT, and A RAUZY. 1999. The AltarRica formalism for Describing Concurrent Systems. *Fundamenta Informaticae*. **34**.

GRUNSKE, L and B KAISER. 2005. An Automated Dependability Analysis Method for COTS-Based Systems. *Lecture Notes in Computer Science*., pp.178-190.

GRUNSKE, L, B KAISER, and Y PAPADOPOULOS. 2005. Model-driven Safety Evaluation with State-event-based Component Failure Annotations. *In*: *8th International Symposium on Component-based Software Engineering*., pp.33-48.

GRUNSKE, L and R NEUMANN. 2002. Quality improvement by integrating non-functional properties in software architecture specification. *In*: *EASY'02 Second Workshop on Evaluating and Architecting System Dependability*. San Jose, California, USA, pp.23-32.

GÜDEMANN, M, F ORTMEIER, and W REIF. 2007. Using deductive cause-consequence analysis (DCCA) with SCADE. *In*: *26th International Conference in Computer Safety, Reliability and Security*., pp.465-478.

GÜDERMANN, M, F ORTMEIER, and W REIF. 2008. Computing ordered minimal critical sets. *In*: *Proceedings of Formal Methods for Automation and Safety in Railway and Automotive Systems*.

HEIMDAHL, M P, Y CHOI, and M W WHALEN. 2002. Deviation analysis though model checking. *In*: *Proceedings of the 17th IEEE International Conference on Automated Software Engineering*. Edinburgh, UK.

HIROYASU, T, S NAKAYAMA, and M MIKI. 2005. Comparison study of SPEA2+, SPEA2, and NSGA-II in diesel engine emissions and fuel economy problem. *In*: *The 2005 IEEE Congress on Evolutionary Computation*. Edinburgh, Scotland, pp.236-242.

HSIEH, Y. 2002. A linear approximation for redundant reliability problems with multiple component choices. *Computers and Industrial Engineering*. **44**, pp.91-103.

IEEE Reliability Society - Reliability Engineering. [online]. [Accessed 05 Nov 2009]. Available from World Wide Web: < HYPERLINK "http://www.ieee.org/portal/site/relsoc/menuitem.e3d19081e6eb2578fb2275875bac26c8/index.jsp?&pName=relsoc_level1&path=relsoc/Reliability_Engineering&file=index.xml&xsl=generic.xsl" http://www.ieee.org/portal/site/relsoc/menuitem.e3d19081e6eb2578fb2275875bac26c8/index.jsp?&pName=relsoc_level1&path=relsoc/Reliability_Engineering&file=index.xml&xsl=generic.xsl >

JOHNSON, M D, D R TAURITZ, and R W WILKERSON. 2007. SNDL-MOEA: stored non-domination level MOEA. *In*: *Proceedings of the 9th annual conference on Genetic and evolutionary computation*. London, England: ACM, pp.837-844.

KAISER, B and C GRAMLICH. 2004. State-Event-Fault-Trees - A Safety Analysis Model for Software Controlled Systems. *Computer Safety, Reliability, and Security*. **3219**, pp.195-209.

KAISER, B, C GRAMLICH, and M FORSTER. 2007. State/event fault trees - A safety analysis model for software-controlled systems. *Reliability Engineering and System Safety*. **92**, pp.1521-1537.

KAISER, B, P LIGGESMEYER, and O MÄCKEL. 2003. A new component concept for fault trees. *In*: *Proceedings of the 8th Australian Workshop on Safety Critical Systems and Software (SCS'03)*.

KIM, H, C BAE, and S PARK. 2004. Simulated annealing algorithm for redundancy optimization with multiple component choices. *In*: *Advanced Reliability Modelling, Proceedings of the 2004 Asian Internationals Workshop*. World Scientific, pp.237-244.

KNOWLES, J D and D W CORNE. 2000. Approximating the Nondominated Front Using the Pareto Archived Evolution Strategy. *Evolutionary Computation*. **8**(2), pp.149-172.

KONAK, A, D W COIT, and A E SMITH. 2006. Multi-objective optimisation using genetic algorithms: A tutorial. *Reliability Engineering and System Safety*. **91**, pp.992-1007.

KULTUREL-KONAK, S, D W COIT, and F BAHERANWALA. 2008. Pruned Pareto-optimal sets for the system redundancy allocation problem based on multiple prioritized objectives. *Journal of Heuristics*. **14**(4), pp.335-357.

KULTUREL-KONAK, S, A E SMITH, and D W COIT. 2003. Efficiently solving the redundancy allocation problem using tabu search. *IIE Transactions*. **35**, pp.515-526.

KULTUREL-KONAK, S, A E SMITH, and B A NORMAN. 2006. Multi-objective tabu search using a multinomial probability mass function. *European Journal of Operational Research*. **169**, pp.918-931.

KUMAR, R, K IZUI, M YOSHIMURA, and S NISHIWAKI. 2009. Multi-objective hierarchical genetic algorithms for multilevel redundancy allocation optimisation. *Reliability Engineering and System Safety*. **94**(4), pp.891-904.

KUO, W, V R PRASAD, F A TILLMAN, and C L HWANG. 2001. *Optimal Reliability Design: Fundamentals and applications*. Cambridge, UK: Cambridge University Press.

LIANG, Y C and A E SMITH. 2004. An ant colony optimisation algorithm for the redundancy allocation problem (RAP). *IEEE Transactions on Reliability*. **53**(3), pp.417-423.

LI, Z, H LIAO, and D W COIT. 2009. A two-stage approach for multi-objective decision making with applications to system reliability optimisation. *Reliability Engineering and System Safety*. **94**, pp.1585-1592.

LISAGOR, O and J A MCDERMID. 2006. Towards a practicable process for automated safety analysis. *In*: *24th International System Safety Conference*.

MARSAN, M A and G CHIOLA. 1987. On Petri nets with deterministic and exponentially distributed firing times. *In*: *European Workshop on Applications and Theory of Petri nets*., pp.132-145.

MEYSENBURG, M M and J A FOSTER. 1999. Randomness and GA Performance, Revisited. *In*: *Proceedings of the Genetic and Evolutionary Computation Conference*., pp.425-432.

MILITARY, U S. 1949. Procedure for performing a failure mode effect and criticality analysis. *United States Military Procedure MIL-P-1629*.

NAKAGAWA, Y and S MIYAZAKI. 1981. Surrogate constraints algorithm for reliability optimization problems with two constraints. *IEEE Transactions on Reliability*. **R-30**, pp.175-180.

ONISHI, J, S KIMURA, R J W JAMES, and Y NAKAGAWA. 2007. Solving the redundancy allocation problem with a mix of components using the improved surrogate constraint method. *IEEE Transactions on Reliability*. **56**(1), pp.94-101.

ORTMEIER, F, W REIF, and G SCHELLHORN. 2005. Deductive cause-consequence Analysis (DCCA). *In*: *The 16th IFAC World Congress*.

PANDE, P K, M E SPECTOR, and P CHATTERJEE. 1975. *Computerised Fault Tree Analysis: 'TREEL' and 'MICSUP'*. University Of California Operations Research Centre, California, USA.

PAPADOPOULOS, Y and C GRANTE. 2005. Evolving car designs using model-based automated safety analysis and optimisation techniques. *Journal of Systems and Software*. **76**(1), pp.77-89.

PAPADOPOULOS, Y I and J A MCDERMID. 1999. Hierarchically performed hazard origin and propagation studies. *In*: *18th International Conference in Computer Safety, Reliability and Security*. Toulouse, France, pp.139-152.

PAPADOPOULOS, Y, J A MCDERMID, R SASSE, and G HEINER. 2001. Analysis and synthesis of the behaviour of complex programmable electronic systems in conditions of failure. *Reliability Engineering and System Safety*. **71**(3), pp.229-247.

PARKER, D and Y PAPADOPOULOS. 2007. Automatic Dependability Versus COst Optimisation Of Distributed Vehicle Control Systems. *In*: *11th European Automotive Congress*. Budapest, Hungary.

PARKER, D and Y PAPADOPOULOS. 2007. Effective Multicriteria Redundancy Allocation Via Model-Based Safety Analysis. *In*: *Intelligent Manufacturing Systems*. Alicante, Spain.

PARKER, D and Y PAPADOPOULOS. 2007. Optimisation of Networked Control Systems Using Model-based Safety Analysis Techniques. *In*: *IEEE International Conference On Networking, Sensing and Control ICNSC'07*. London, UK.

PARKER, D, M WALKER, Y PAPADOPOULOS, and C GRANTE. 2006. Component-based Automated FMEA of Advanced Active Safety Systems. *In*: *FISITA'06 31st World Automotive Congress*. Yokohama, Japan.

PARKER, D J, M D WALKER, Y PAPADOPOULOS, and C GRANTE. 2006. Component-Based, Automated FMEA of Advanced Active Safety Systems. *In*: *31st World Automotive Congress FISITA*. Yokohama, Japan: JSAE.

PLC., Ricardo. 2009. *AutoFMEA™ Safety Analysis and FMEA Report Generation for Simulink® and Stateflow®*. [online]. [Accessed 20 Nov 2009]. Available from World Wide Web: < HYPERLINK "http://www.ricardo.com/en-gb/Engineering-Consulting/Automotive-Expertise/Controls--Electronics/Software-Tools/AutoFMEA/" http://www.ricardo.com/en-gb/Engineering-Consulting/Automotive-Expertise/Controls--Electronics/Software-Tools/AutoFMEA/ >

PRESS, W H, B P FLANNERY, S A TEUKOLSKY, and W T VETTERLING. 1992. *7.1 Uniform Deviates*. [online]. [Accessed 21 Dec 2008]. Available from World Wide Web: < HYPERLINK "http://www.nrbook.com/a/bookcpdf/c7-1.pdf" http://www.nrbook.com/a/bookcpdf/c7-1.pdf >

PRICE, C J and N S TAYLOR. 2002. Automated Multiple Failure FMEA. *Reliability Engineering and System Safety*. **76**(1), pp.1-10.

RAUZY, A. 1993. New algorithms for fault tree analysis. *Reliability Engineering*. **40**, pp.203-211.

REESE, J D and N LEVESON. 1997. Software Deviation Analysis. *In*: *International Conference on Software Engineering*.

RÜDE, E. 2007. *Methods and techniques in support of ship (SAFEDOR D 4.5.6)*.

SALAZAR, D, C M ROCCO, and B J GALVAN. 2006. Optimisation of constrained multiple-objective reliability problems using evolutionary algorithms. *Reliability Engineering and System Safety*. **91**, pp.1057-1070.

SHARVIA, S and Y PAPADOPOULOS. 2008. Non-coherent modelling in compositional safety analysis. *In*: *IFAC 17th World Congress*. Seoul, South Korea.

SINAMMON, R M and J D ANDREWS. 1997. New approaches to evaluating fault trees. *Reliability Engineering and System Safety*. **58**, pp.89-96.

SRINIVAS, N and K DEB. 1995. Multi-objective function optimisation using non-dominated sorting algorithms. *Evolutionary Computation*. **2**(3), pp.221-248.

SYSWERDA, G. 1989. Uniform crossover in genetic algorithms. *In*: *Proceedings of the 3rd International Conference on Genetic Algorithms*., pp.2-9.

TABOADA, H A, F BAHERANWALA, D W COIT, and N WATTANAPONGSAKORN. 2007. Practical solutions for multi-objective

optimisation: An application to system reliability design problems. *Reliability Engineering and System Safety*. **92**, pp.314-322.

TABOADA, H A and D W COIT. 2007. Data clustering of solutions for multiple objective system reliability optimisation problems. *Quality Technology and Quantitative Management*. **4**(2), pp.191-210.

TATE, D M and A E SMITH. 1994. A genetic approach to the quadratic assignment problem. *Computers and Operations Research*. **22**, pp.73-83.

VESELY, W E, M STAMATELATOS, J DUGAN *et al.* 2002. *Fault Tree Handbook with Aerospace Applications*. USA: NASA office of Safety and Mission Assurance.

WALKER, M, L BOTTACI, and Y PAPADOPOULOS. 2007. Compositional temporal fault tree analysis. *In*: *Computer Safety, Reliability, and Security*., pp.105-119.

WALKER, M and Y PAPADOPOULOS. 2006. Pandora: The Time of Priority-AND gates. *In*: *12th IFAC Symposium on Information Control Problems in Manufacturing*. St Etienne, France, pp.237-242.

WALL, M. 1999. *GAlib: Matthew's Genetic Algorithms Library*. [online]. [Accessed 21 Dec 2008]. Available from World Wide Web: < HYPERLINK "http://lancet.mit.edu/ga/" http://lancet.mit.edu/ga/ >

WALLACE, M. 2005. Modular architectural representation and analysis of fault propagation. *Electronic Notes in Theoretical Computer Science*. **141**(3), pp.53-71.

WOLFORTH, I, M WALKER, and Y PAPADOPOULOS. 2008. A language for failure patterns and application in safety analysis. *In*: *Proceedings of the 3rd International Conference on Dependability of Computer Systems*., pp.47-54.

WORRELL, R B and D W STACK. 1978. *'SETS', User Manual for Fault Tree Analysis*. Sandia Labs, Albequerque, NM, USA.

YOU, P and T CHEN. 2005. An efficient heuristic for series-parallel redundant reliability problems. *Computers and Operations Research*. **32**, pp.2117-2127.

ZENG, W, Y PAPADOPOULOS, and D PARKER. 2007. Reliability Optimisation of Series-Parallel Systems Using Asynchronous Heterogeneous Hierarchical Genetic Algorithm. *Journal of Mind and Computation*. **1**(4), pp.403-412.

ZENG, W, Y PAPADOPOULOS, and D PARKER. 2008. Multi-objective Optimisation of Dependability Attributes Using An Asynchronous Heterogeneous Hierarchical Parallel Genetic Algorithm. *In*: *Intelligent Manufacturing Systems IMS'08*. Szczecin, Poland.

ZHAO, J, Z LIU, and M DAO. 2007. Reliability optimisation using multiobjective ant colony system approaches. *Reliability Engineering and System Safety*. **92**, pp.109-120.

ZIMMERMANN, A, R GERMAN, J FREIHEIT, and G HOMMEL. 1999. TimeNET 3.0 Tool Description. *In*: *International Conference on Petri Nets and performance models*. Zaragoza, Spain.

ZITZLER, E, M LAUMANNS, and L THIELE. 2001. SPEA2: Improving the strength Pareto evolutionary algorithm. *In*: *Proceedings EUROGEN 2001 Evolutionary methods for Design, Optimisation and Control with Applications to Industrial Problems*. Athens, Greece.

ZITZLER, E and L THIELE. 1999. Multiobjective evolutionary algorithms: A comparative study and the strength Pareto approach. *IEEE Transactions on Evolutionary COmputation*. **3**(4), pp.257-271.