Searching For Flexible Solutions To Task Allocation Problems

(Submitted for the degree of Doctor of Philosophy)

Paul Emberson

Department of Computer Science,
The University of York

November 2009

Abstract

Consumers of modern avionics and automotive systems expect many, well integrated features. Efforts are continually being made to make engineering processes better equipped to adapt to enhancement requests. Within both the avionics and automotive industries, standardisation of hardware and interfaces has allowed software to be mapped to hardware at a later stage of the design process and for this mapping to be more easily changed.

Tools which automatically perform the mapping of tasks and messages onto a hardware platform are called task allocation tools. The primary requirement of a task allocation tool for hard real-time systems is to find a mapping and schedule such that all tasks and messages respond before their deadlines. However, there are other qualities which can be used to further differentiate between solutions, two of the most important being flexibility and adaptability.

This thesis builds on previous task allocation work by extending a heuristic search algorithm to produce solutions with improved flexibility. Inspiration is drawn from scenario based architecture analysis methods. These methods interrogate an architecture model to see how it will react to different change scenarios. This idea is used within a search algorithm to encourage it to produce solutions which can meet the needs of provided scenarios with no or very few changes. It is shown that these solutions are also more flexible with respect to upgrades which differ from the scenarios.

Run-time adaptability is another quality which can be affected by the choice of task allocation. Real-time systems can specify multiple task sets representing different modes of operation. The system will switch between modes at run-time to adapt to environmental changes and must do so efficiently. The task allocation algorithm is adapted for multi-moded systems and it is shown that solutions can be found which allow the system to transition between modes with minimal disruption.

Safety-critical real-time systems have become dependent on software to provide critical functionality such as fly-by-wire control and engine emission regulation. These systems must be fault-tolerant and support graceful degradation, another form of adaptability. In the final part of this thesis, the task allocation algorithm is modified to select a number of replicas for each task as well as their allocation so that the system can withstand as many processor failures as possible before the level of service provided by the system falls below a safe threshold.

Contents

List of Figures

List of Tables

Glossary

ABAS Attribute-Based Architecture Style. An association between a particular architectural view and its analysis based on a particular set of quality attributes. 72

APEX Application Executives. Part of the IMA specification which gives a way of partitioning and scheduling resources between applications on a single LRM. 33, 40

ATAM Architecture Tradeoff Analysis Method. Methodology for trading off architecture qualities. 72, 74

BCET Best Case Execution Time. The shortest possible time a single activation of a task needs to execute for before completion. 49, 88

CAN Controller Area Network. Network communications protocol which is very common in automotive systems. 31, 44–46, 88

CASE Computer Aided Software Engineering. Used in the context of computer based tools used in the software engineering process. 24, 25, 34

COTS Commercial Off-The-Shelf. Used to describe components which are usually mass produced and hence cheaper and more readily available than bespoke components. 32

DMPO Deadline Monotonic Priority Ordering. In fixed priority scheduling, the assignment of priorities to tasks such that the shorter the deadline of the task, the higher its priority. 43, 51, 52, 61

ECU Electronic Control Unit. Used in the context of automotive systems to describe a control unit containing a processor and memory. Provides processing resources for tasks. 30–32

EDF Earliest Deadline First. A dynamic priority scheduling policy where the task with the earliest deadline is given the highest priority. 45, 49, 59, 65, 73

IMA Integrated Modular Avionics. A layered architecture for avionics systems with standardised interfaces. 33, 34, 40, 45, 55, 56, 78, 221

LRM Line Replaceable Module. More modern version of an LRU. Fits into IMA cabinets. 33, 78

LRU Line Replaceable Unit. Processing resource for tasks in avionics systems. Avionics equivalent of ECUs in automotive architectures. 32, 33

MOEA Multi-Objective Evolutionary Algorithm. An evolutionary search optimisation algorithm which generates a non-dominated set of solutions for multiple criteria. 66, 75

SAAM Scenario Based Architecture Analysis Method. A process for evaluating and improving a system architecture by testing scenarios on an architecture model. 71–73

SBSE Search Based Software Engineering. The field of applying search algorithms to optimisation problems found in software engineering. 24, 161, 181, 214

TTCAN Time Triggered CAN. Time triggered version of CAN protocol. 46

WCCT Worst Case Communication Time. The longest time a message takes to travel across a network. 46, 88, 108, 109

WCET Worst Case Execution Time. The longest possible time a single activation of a task needs to execute for before completion. 38, 39, 44, 49, 66, 88, 92, 99, 107, 122

WCRT Worst Case Response Time. The longest possible time a single activation of a task to complete including factors such as interference from other tasks, contention for shared resources and jitter. 41, 43, 44, 50, 51, 109

Acknowledgements

I would like to thank Iain Bate as friend and supervisor for all his advice and guidance over the last 5 years. Thank-you for always being available to read papers only hours before the deadline!

I thank my parents for all their love and support throughout my life and never doubting that I would get to the end of this thesis.

Thank-you also to everyone I worked with on the DCSC and SEBASE projects and in the HISE and RTS groups at York. Many of you sat through my talks and provided useful feedback. Thanks to John Clark who was always a source of interesting ideas.

Many friends have helped me to complete this thesis. Some have been directly involved by giving suggestions and ideas. Others have provided welcome distractions when the work became stressful. In particular, I want to thank: Rob Alexander for proof reading and comments; David White for last minute proof reading; Simon Poulding for advice and encouragement, especially during the TSE paper; Rob Davis for many useful discussions on task set generation. Richard Hawkins, Peter Laurens, Sevil Sen and Jan Staunton, all of whom I enjoyed sharing and office with; Martin Hall-May for being such an easy going house mate and great cook; Simon Bates for climbing in the peaks and nights out in Leeds and York; Rob Collyer for "taking the edge off" with "a few beers"; Catherine Menon for encouraging me to finish during the last couple of months; Jim Adamson for games of tennis and table-tennis; Ioanna Symeou for great times in York, Wales and Cyprus; Mike and Claudia Jordon for some unforgettable holidays and fun abroad; all my friends in London including Nicky Hipkiss, Gav Smith and Mark Ackroyd for being supportive despite telling me to "come to London and get a proper job"; Steve Mace for always making the effort to keep in touch.

I have used open source software to write this thesis, develop code, analyse data, distribute experiments between computers and listen to music while I work. Thanks to everyone who makes these programs freely available.

Declaration

I declare that all the work in this thesis is my own, except where attributed to another author.

Certain parts of this thesis have been included in previously published papers or accepted to appear in papers which will be published in the near future. The following papers contain methodology and results directly relevant to this thesis and are cited in the text where appropriate:

- Paul Emberson and Iain Bate, *Minimising Task Migration and Priority Changes In Mode Transitions*, In proceedings 13th IEEE Real-Time And Embedded Technology And Applications Symposium (RTAS 2007), pp. 158–167, 2007.
- Simon Poulding, Paul Emberson, Iain Bate and John Clark, *An Efficient Experimental Methodology for Configuring Search-Based Design Algorithms*, In proceedings 10th IEEE High Assurance System Engineering Symposium (HASE 2007), pp. 53–62, 2007.
- Paul Emberson and Iain Bate, *Extending A Task Allocation Algorithm For Graceful Degradation Of Real-Time Distributed Embedded Systems*, In proceedings 29th IEEE Real-Time Systems Symposium (RTSS 2008), pp. 270–279, 2008.
- Paul Emberson and Iain Bate, *Stressing Search with Scenarios for Flexible Solutions to Real-Time Task Allocation Problems*, To appear IEEE Transactions on Software Engineering, DOI: 10.1109/TSE.2009.58

Chapter 5 uses methodology developed in the collaborative paper with Simon Poulding. The introduction of that chapter clearly explains his contributions versus my own.

The following papers contain related content which the above papers built upon, but is not used directly in this thesis.

- Iain Bate and Paul Emberson, *Design For Flexible And Scalable Avionics Systems*, In proceedings of IEEE Aerospace Conference, 2005.
- Iain Bate and Paul Emberson, *The Use Of Scenarios To Improve The Flexibility In Embedded Systems: Research Challenges And Preliminary Results*, In proceedings 2nd UK Embedded Forum, pp. 137–156, 2005

- Paul Emberson, Iain Bate and Mike Bennett, *Task Allocation In An Industrial Avionics Application*, In proceedings 3rd International Workshop on Dependable Embedded Systems (WDES 2006), pp. 44–48, 2006
- Iain Bate and Paul Emberson, *Incorporating Scenarios And Heuristics To Improve Flexibility In Real-Time Embedded Systems*, In proceedings 12th IEEE Real-Time And Embedded Technology And Applications Symposium (RTAS 2006), pp. 221–230, 2006.

1

Introduction

The greatest strength of using software as a component of an engineered system is the ease with which it can be modified to enhance functionality and improve the performance of the system. At the same time, this flexibility also presents one of the greatest challenges in constructing a software based system; as a system grows and becomes more complex, so does the problem of managing change.

The majority of software systems continually evolve throughout their lifetime. David Parnas used the following phrase to describe those that don't.

> *"The only programs that don't get changed are those that are so bad that nobody want to use them. Designing for change is designing for success."*
>
> – David Parnas [1]

"Designing for change is designing for success" is the adopted motto of this thesis; it is a study of how the statement can be applied to the selection of software task to hardware mappings for safety-critical real-time systems. The problem of finding a good mapping, also known as the *task allocation problem*, is well suited to automated optimisation and constraint satisfaction techniques. Research into algorithms for distributed multiprocessor scheduling has been active since the late 1970s [2]. The popular papers of Ramamritham in 1990 [3] and 1995 [3] as well as Tindell's much cited 1992 paper [4] made it a core topic of real-time systems' research. This thesis describes algorithms for allocating real-time tasks which treat flexibility of solution as a key requirement.

This introductory chapter gives a broad view of the relevant topic areas of architecture flexibility, automated software engineering tools and safety-critical real-time systems. These provide background and motivation for the thesis aims presented at the end of this chapter.

1.1 Maintenance In Software Engineering

Reducing the costs of software maintenance is a major motivation for improving flexibility of software architectures. This section explains the reasons why software maintenance is needed, why it is so expensive and current methods of managing it.

1.1.1 Software Maintenance

A topic of concern at the NATO software engineering conference in 1968 was the cost of maintaining software systems.

> *"The economics of software development are such that the cost of maintenance frequently exceed that of the original development."*
>
> – Harold R. Gillette. [5], p. 111

Since this time, maintenance costs have continued to grow as a proportion of total life-cycle costs. Based on a number of surveys, Pigoski [6] suggests that, by the 1990s, 90% of life-cycle costs were devoted to maintenance. This value had risen from around 40% in the 1970s. As the number of architectural components increases, and the number of lines of code grow, the larger the potential of a change having an undesired side effect. Gaining a good understanding of the rationale behind the choice of the architecture and design of the system is essential to prevent this and accounts for up to 60% of maintenance costs [6].

Software engineers have often tried to gain insight from other engineering disciplines such as bridge building [7]. One of the areas where these comparisons fall down is the way in which the engineered artefact is used throughout its lifetime. A bridge will need to be maintained to withstand the effects of the environment and possibly to increase its strength if the volume of traffic exceeds original expectations. The purpose of a bridge — to allow its load to traverse above an obstacle — will not change. For software systems, the inducements to carry out maintenance are much more varied. Bennett [8], citing Lientz and Swanson [9], gives the following categories of maintenance for software systems: corrective, perfective, adaptive and preventive.

Corrective maintenance is fixing errors that were not discovered during the testing phase of development. An implementation mistake and an incorrectly specified requirement are both classed as errors.

Perfective maintenance is enhancing existing features and adding new ones based on user feedback. The word "perfective" is misleading since it suggests adding the finishing touches to a system. For many systems, more development will be done following the first release than prior to it. In an iterative development method, where the customer is closely involved in the development process, perfective maintenance and development are parallel activities [6].

Adaptive maintenance is the activity of modifying the system to support a change in the operating environment. A common reason for this type of maintenance is obsolescence of system dependencies and components. This has been a problem in avionics systems which use old hardware technologies for their reliability and predictability. For example, some space shuttle upgrades were due to the risk of components becoming obsolete [10].

Preventive maintenance is taking action before it is needed in order to make future maintenance tasks easier. This applies to source code, formal designs and other documentation. As new features are added which weren't originally planned for, the quality of the architecture will gradually degrade [1]. Preventive maintenance ensures that the software continues to be comprehensible and the architecture maintains its integrity.

1.1.2 Processes And Technologies Supporting Maintenance

By the late 1960s, it had been recognised that the size and complexity of software systems had outstripped available software engineering technologies.

> *"We undoubtedly produce software by backward techniques."*
>
> – Douglas McIlroy [5], p. 17

This remark was made in reference to the "rudimentary stage of development" of software engineering at the time. Since then, software engineering practitioners and researchers have improved software engineering processes and put tools at the disposal of engineers which help manage these processes and automate parts of them where possible. Many of the advancements made have been driven by the need to reduce the effort required to perform maintenance tasks.

1.1.2.1 Processes

The first widely adopted software engineering process was the Waterfall model [11] which was highly influential throughout the 1970s [12] and is still used for some large scale projects [13]. Whilst it acknowledges iteration between adjacent refinement steps [12], it is still mainly a linear process beginning with requirements and working through to implementation and deployment. In this model, maintenance is seen as something that begins after deployment [6, 14]. The trouble with this is that fixing errors later in the development process is significantly more expensive as shown in Boehm's COCOMO cost estimation model [15]. It also depends on accurate requirements for the system being known in advance and not changing substantially throughout its development and deployment. As commented on by Parnas [1], accurately predicting the future and expecting requirements to remain stable is unrealistic. Problems with waterfall style development were acknowledged in the early 1980s and lead to Boehm's spiral model of software development and

enhancement [12] which specifically included maintenance in the process. More recently, iterative and incremental processes [13, 16, 17] have gained popularity. One of the key concepts of these processes is to release software to the customer early in the product life-cycle and to keep the customer closely involved in development [13]. A perceived benefit of this tighter feedback loop is to find errors earlier in the life-cycle so that the cost of fixing them is lowered [18]. This results in maintenance activities proceeding in parallel with feature development prior to deployment.

1.1.2.2 Technologies

Improvements in compiler technologies such as machine level optimisations and automated garbage collection have allowed development at higher levels of abstraction. Model driven engineering is an effort to further raise levels of abstraction. A model is constructed in a language which is often graphical and transformed into lower level code automatically. Schmidt [19] states that "complexity stems from the semantic gap between the design intent [...] and the expression of this intent [...]". Model driven engineering aims to close this gap. If the translation process is not fully automated then there is a risk of the model and implementation diverging when changes are made [20].

Compilers, model building tools and automatic code generators are all examples of Computer Aided Software Engineering (CASE) tools. A software engineering environment is a collection of CASE tools which support the software process [21]. The benefits from the use of CASE tools have not always been as great as expected. Sommerville presents this criticism against existing CASE tools.

> *"Software engineering is, essentially, a design activity based on creative thought. Existing CASE systems automate routine activities but attempts to harness artificial intelligence technology to provide support for design have not been successful."*
>
> – Sommerville [13]

Tools with a creative ability are not in mainstream use within the software industry. However, improving intelligence in software engineering tools is an active area of research and progress has been made. Search Based Software Engineering (SBSE) [22] treats software engineering problems as design optimisation problems and then employs search algorithms to find a high quality design. Although there is a significant human influence in defining suitable heuristics to guide search algorithms, they have the potential to find solutions which a human may not. In particular, SBSE can be used to optimise a manually constructed design as a form of preventive maintenance. An example is reassigning methods and attributes in a class hierarchy to improve coupling and cohesion [23].

Figure 1.1 outlines the process of automated model improvement. The model is only partially specified and free parameters are set using a design tool. During

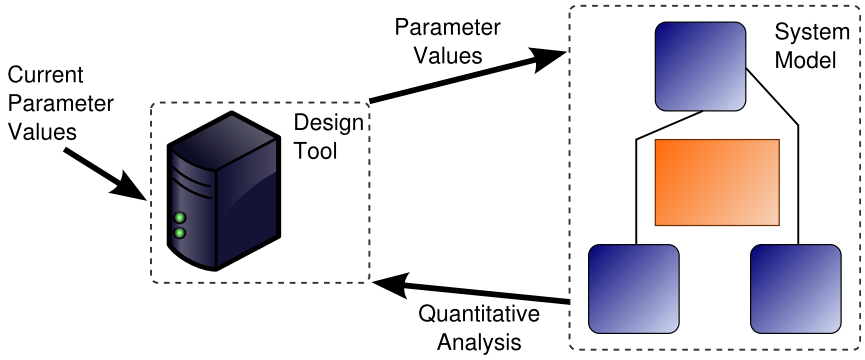


Figure 1.1: Automated model improvement

maintenance, previously chosen parameter values may be used to initialise optimisation algorithms. One of the prerequisites to using automated design tools is having suitable quantitative measures for the qualities of interest. Decomposing system qualities into measurable attributes is one way to overcome this. As part of a larger hierarchy of system qualities, Boehm stated that maintainability could be decomposed into understandability, modifiability and testability [24]. Maintainability is hard to assess directly, but it is possible to construct metrics for architecture level modifiability [25] and code understandability [24].

A specific domain will have its own architecture selection problems. Automotive systems' architectures contain a number of difficult design optimisation problems [26] that are well suited to CASE tools since automotive systems are comparatively well specified and system models can be quantitatively analysed [27].

1.1.3 Flexibility And Adaptability

To deal with the problem posed at the beginning of this chapter, that of managing change, the ability of a system to handle change must be viewed in a number of different ways. This ability is sometimes less elegantly, but more conveniently, described as "changeability" [28, 29]. Maintainability is chiefly concerned with changes made by software engineers within development iterations. Changeability also encompasses changes when the system is operative.

There have been different approaches to decomposing changeability [28, 29]. Fricke and Schulz consider four different aspects: flexibility, adaptability, agility and robustness. The definitions of flexibility and adaptability are consistent throughout the literature. Flexibility describes how easy it is to change the system where the instigator of the change, known as the change agent [29], is external to the system. In the context of development, flexibility relates closely to maintainability.

Adaptability is the ability of the system to autonomously change itself to cope with a changing environment. Note that this is inconsistent with the definition of adaptive maintenance where the change agent will usually be external.

Agility describes how much time is required for a change process to complete.

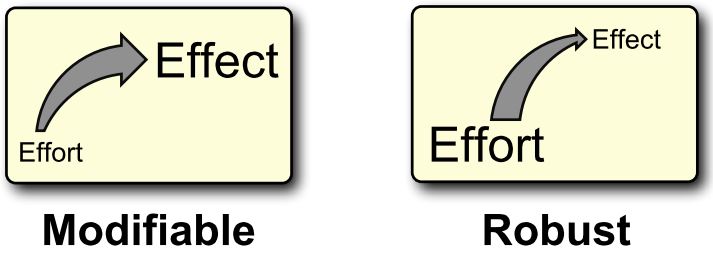


Figure 1.2: Effort / effect relationship for modifiable and robust systems

Fricke and Schulz only associate it with flexibility [28] but a system should also adapt promptly to deal with a changing environment, especially in an emergency situation [30]. Ross et al. do not include agility within their definition of changeability [29] viewing it as a modifier to changeability. Something which is easy to modify, does not necessarily mean it can change quickly. For example, correcting loop bounds on a critical buffer overflow may only require changing a few characters but rebuilding and redeploying the system might take several days. Financial cost is related to changeability in a similar way. A component can be physically easy, but prohibitively expensive, to replace.

Ross et al. [29] break down change into a three stage process.

1. A change agent, which can be external or internal, instigates the change.
2. This initiates a change mechanism which
3. has an effect on the system

Within this scheme, it becomes clear that cost and time are attributes of the change mechanism. Cost and time can both counteract ease of modification so are an essential component of overall changeability.

Robustness, the final aspect of changeability in Fricke and Schulz's taxonomy [28] is dealt with alongside modifiability and scalability by Ross et al. [29]. If a system is robust to a particular change, then the change has little or no effect on the parameters of the system. For example, a bridge may be said to be robust to an increase in wind speed if its movement is sufficiently limited to still allow traffic to pass over it. Modifiability is the opposite of robustness. A system which is modifiable with respect to a change will exhibit an effect with relatively little effort. Robustness and modifiability are shown pictorially in figure 1.2. Scalability is a special case of modifiability. It describes the ease of changing the level of a parameter as opposed to changing the set of parameters available [29].

It is common for a module of a system to be robust and modifiable at the same time depending on the change. Changing the implementation of one module should not be arduous but an associated module should be robust to the change and hence reduce ripple effects. The opposite is true if the other module needs to be changed. All changeability related terms are usually used in a positive context. Therefore if a

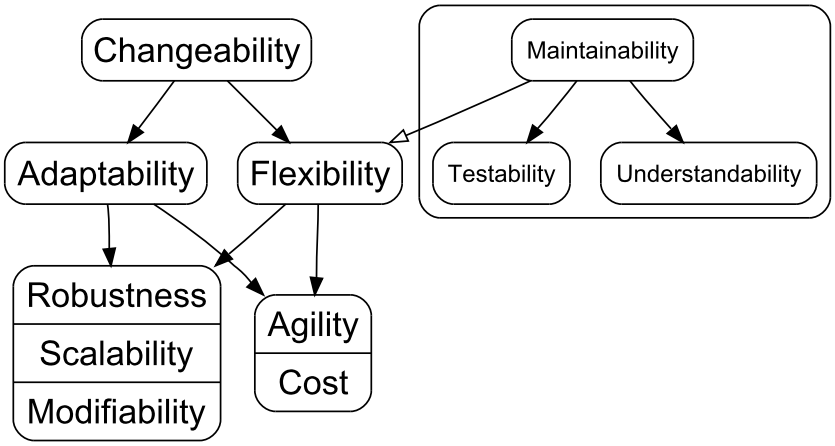


Figure 1.3: Terms related to changeability

change did adversely affect a related module, it is said to be not robust to the change rather than being modifiable.

A component can also be modifiable internally to increase robustness externally. Electromagnetic radiation can cause bit flips in computer memory [31]. The memory can be made robust to this by physically shielding it. However, if a bit flip does occur, then making the memory modifiable to an error correction mechanism allows it to be corrected [31]. If the memory is correctly returned to its original state, no change has occurred from an external viewpoint and so the memory is robust to the radiation.

A summary of the terms relating to changeability are shown in figure 1.3. This combines both the models of Fricke et al. [28] and Ross et al. [29]. The link with maintainability is also shown. The hollow arrow linking maintainability to flexibility represents a specialisation. Maintainability requires the same attributes as flexibility. In addition, testability and understandability are specific to maintainability.

Changeability has links with several other qualities. Dependability is a motivation for internal adaptability and reconfiguration [32]. Barbacci et al. [33] also associate maintainability to dependability with the view that better maintainability reduces the mean time to repair (MTTR) and hence improves availability. Modifiability is at the heart of many architecture trade-off methods [34]. If a modification to a system adversely affects qualities such as performance or security to a significant extent then further work and architectural analysis will be needed. In this sense, changeability is linked to every system quality, though some of these links will be stronger than others depending on the importance of each individual quality in a particular system.

1.2 Real-Time And Safety-Critical Systems

Although there are a number of common challenges to engineering flexible and adaptable systems, each project will have domain specific issues. The fields of real-time and

safety-critical systems garner particular attention in literature related to changeability. There is interest in real-time systems since timing gives an additional architecture trade off point [35] which must be taken into account when evaluating an architecture. Safety-critical systems are expensive to build and expensive to change [36] due to the long and rigorous validation and verification processes required for system certification. The challenges undertaken in this thesis relate to the changeability of systems which are both safety-critical and real-time. This section explains terminology used to describe these types of systems.

1.2.1 Safety-Critical Systems

There is lack of agreement on a definition for safety-critical systems [37]. Debating borderline cases is not necessary since the work in this thesis is directed at systems which fall comfortably inside any commonly used definition. Some definitions relating to safety, safety-related systems and safety-critical systems are provided by Storey [37] and by Leveson [38].

> *"Safety is a property of a system that it will not endanger human life or the environment."*
>
> *"A safety-related system is one by which the safety of equipment or plant is assured."*
>
> – Neil Storey [37]

Safety-related system is used as a synonym for a safety-critical system [37].

> *"Safety is freedom from accidents or losses."*
>
> *"Software system safety implies that the software will execute within a system context without contributing to hazards."*
>
> *"Safety-critical software is any software that can directly or in-directly contribute to the occurrence of a hazardous system state."*
>
> – Nancy Leveson [38]

The texts of both Leveson [38] and Storey [37] feature sections on the importance of timing requirements in safety-critical systems. This is unsurprising since many safety-critical systems have roles which are also time critical. Examples include flight stability systems, airbag deployment in cars and anti-lock braking systems. All of these systems control actuators which must respond within a given time window; reacting too early or too late could have disastrous consequences.

1.2.2 Real-Time Systems

A real-time system is one where its timing requirements are an integral part of its behavioural specification. The following definition is given by Kopetz.

> *"A real-time computer system is a computer system in which the correctness of the system behaviour depends not only on the logical results of the computations, but also on the physical instant at which these results are produced."*
>
> – Hermann Kopetz [39]

A real-time system is made up of a set of software tasks scheduled to run on one or more processors. Timing requirements are specified in terms of task deadlines. A deadline is the instant by which a task must complete relative to an earlier point in time. Tasks are often periodic meaning that they run repeatedly at a given frequency whilst the system is operative. The times at which a task is executing will depend on the length of its period and the way in which it is scheduled with other tasks in the system. Although high performance is a feature of many real-time systems, predictability is the more dominant characteristic. As noted by Liu [40], a task completing in advance of its deadline may offer no benefit and a large amount of variation in the completion time of a task, its *jitter*, can be detrimental in many systems. Task attributes such as deadlines and periods are often derived from the responsiveness of sensors and actuators [40] and control loops [39].

An important classification of real-time systems is hard real-time versus soft real-time. There is more than one characteristic which can be used to decide whether a system is soft real-time, hard real-time or a mixture of the two. All revolve around the potential effects of task deadline misses on the system, its users and the environment. Some characteristics for differentiating between soft real-time and hard real-time are [40]:

1. The usefulness or utility of data which is output by a task which completed after its deadline — if stale data is either useless or potentially harmful, then the system is hard real-time. If its utility gracefully decreases with time, then the system is soft real-time.
2. The expected number of deadline misses in a given time period — if the number of deadline misses per unit of time which can be tolerated is zero or nearly zero and this is predictable, then the system is hard real-time. If the system is able to tolerate a moderate number of deadline misses and / or the number is not predictable then the system is soft real-time.
3. The potential effects on the system, its users and the environment if task complete unexpectedly late — if a timing error in the system could cause harm to humans or the environment then the system is hard real-time. If a timing error causes a decrease in level of service that is not catastrophic then the system is soft real-time.
4. The level of validation of timing properties which would be expected during development or maintenance — if every change to the system must undergo a rigorous validation and verification procedure requiring formal proof of correct-

Characteristic	Hard Real-Time	Soft Real-Time
Utility of result after deadline	Zero or negative	Reduced
Number of deadline misses	Extremely small and predictable	Unpredictable
Severity of deadline miss	Very high / fatal	Low to medium
Validation of timing properties	Provable / exhaustive testing	Confidence building

Table 1.1: Characteristics for classifying hard real-time versus soft real-time systems

ness or exhaustive testing, then the system is hard real-time. If testing is only undertaken as a confidence building exercise, then the system is soft real-time.

These characteristics are summarised in table 1.1. The first two items in the list above are broadly accepted. The third one, which links hard real-time systems to safety-critical systems, is common though not universal. A system may be hard real-time according to the first two classifiers but not safety-critical. Kopetz makes this distinction, calling such a system firm real-time as opposed to hard real-time [39]. The final classification, linking hard real-time systems to a requirement for rigorous validation is given by Liu [40]. If predictability is essential, then this must be demonstrated at some stage of development.

1.3 Trends In Automotive And Avionics Systems

The aviation and automobile industries are two of the most visible producers of safety-critical systems. Modern examples of these systems are heavily reliant on safety-critical hard real-time software [41, 42]. Although the same problems relating to flexibility and adaptability are also present in other systems with similar properties, these industrial domains provide a wealth of motivation and examples which will be used to help ensure relevance and applicability of the methods proposed in this thesis.

Since the 1970s, both aircraft [42] and automobiles [41, 43] have become increasingly reliant upon digital control systems. This section draws on a number of examples to illustrate the following trends in avionics and automotive systems:

1. Systems are expected to provide more functionality and are increasing in size as a result.
2. There is an increasing amount of interaction between components.
3. Systems have moved from centralised to distributed architectures.
4. Systems have become dependent on software based systems for critical functionality and also to meet regulations.

1.3.1 Automotive Technologies

Modern (post circa 2004) automotive systems contain between 20 and 80 Electronic Control Units (ECUs) [41]. These control everything from critical functionality such

Subsystem	Functionality
Powertrain	Engine control, transmission, powertrain diagnostics, immobilizer
Chassis	Antilock braking system (ABS), electronic stability control (ESC), traction control, 4 wheel drive
Body	Dashboard, wipers, lights, doors, windows, seats, mirrors, climate control
Telematics	Mobile telephones, entertainment, navigation, toll collection, tracing stolen vehicles, traffic information, remote vehicle diagnostics
Safety	Belt pre-tensioner, airbags, tyre pressure monitoring, adaptive cruise control, lane departure warning

Table 1.2: Automotive features summarised from Cook et al. [41] and Navet et al.[43]

as engine and chassis control through to radio volume and windscreen wiper speed. An automotive system is divided into separate subsystems. These subsystems and some of the functionality they provide are shown in table 1.2. Of the subsystems listed, the real-time and safety-critical elements are concentrated within the powertrain, chassis and safety subsystems [43]. Currently steering and braking can be controlled mechanically but future systems may choose to rely entirely on electronic control. There are a number of motivating factors for using software based systems to provide such a large amount of functionality. New regulations on emissions could not be met without advanced engine management systems which control air-fuel ratios [41]. At the same time, there are increasing demands for better fuel efficiency for both economic and environmental reasons. A key technology in this area is variable cam timing (VCT), part of the powertrain subsystem, which also requires electronic control [41]. Dynamic chassis control allows manufacturers to improve handling for performance and safety reasons. Many of the increases in complexity of these systems are driven by a desire to gain a competitive advantage in the marketplace [41, 44].

There is a demand for subsystems to be well integrated. For example, wheel speed sensors can affect steering effort, suspension control and even wiper speed [41]. Central locking has to interact with security systems, door open sensors, interior lights and collision detection so doors do not remain locked in the event of a crash [41].

Early automotive systems tried to contain functionality within a single ECU [43, 44]. The increase in functionality and requirements for increased integration between functions led to firstly point to point links and then eventually shared data buses [43]. Developed in the 1980s, the Controller Area Network (CAN) bus [45], has been by far the most popular communications bus used in automotive systems [43]. Standardisation has helped integration of systems from different subcontractors. Newer communications buses are now being developed to allow for more network traffic whilst still meeting hard real-time requirements [26].

These technological advances have had a significant effect on the development and maintenance of automotive systems. Cars are mass market items and minimising the

number of ECUs used can generate huge cost savings over the production period [44]. This means that code is often tightly optimised making maintenance more difficult due to reduced understandability and modifiability. The size of a modern automotive platform makes it impractical for a manufacturer to develop and maintain all components themselves. Therefore, subsystem development is often outsourced or commercial off-the-shelf (COTS) components are used. This can lead to difficulties at the time of integration, especially with timing and synchronisation [26]. In addition to this, each vehicle model is made available in several different configurations to meet different demands [44]. A premium car could realistically have 2^{80} different possible configurations [44] so it is essential that components can be easily composed with predictable behaviour.

The use of model based development is being introduced into the automotive industry to help alleviate these problems. AUTOSAR [46] is an industry wide effort to develop a common automotive architecture with well defined interfaces to improve composability of components. A return to a more centralised general purpose architecture would reduce the number of ECUs required and simplify development [44]. Further to this, a standardised platform allows separation of the application layer from the hardware layer allows for reuse of software across multiple vehicle programs [26]. This separation leaves the mapping of software to hardware as a free parameter in the architecture as was depicted in figure 1.1. Any method for selecting this mapping must ensure that all real-time, fault-tolerance and other quality requirements are still met. A further layer of separation can be introduced by designing functionality separately from the mapping of functions onto software tasks.

1.3.2 Avionics Technologies

Avionics technologies have evolved in a similar manner to their automotive counterparts. Architectures were originally centralised, evolving from analog systems [47]. Requirements for new functionality have been driven by removing mechanical systems to reduce weight and improve efficiency and improving safety standards in the presence of increasing air traffic. Table 1.3 lists some functionality commonly found in civil avionics systems. Military systems have additional requirements for weapon control and mission planning. Many modern aircraft no longer have a mechanical backup for the flight control system [42]. With so much safety related functionality, avionics systems must be certified before they can be put into operation. The cost of certification is the largest single cost in the development process, with verification taking up as much as 50% of the cost [36]. It is weight rather than cost which drives hardware requirements since certification expense far exceeds that of hardware [48].

Avionics systems initially moved from a centralised architecture to a federated architecture with major subsystems spread between a number of interconnected Line Replaceable Units (LRUs) [47]. Placing these units closer to their related sensors

Subsystem	Description
Flight control	Primary flight control and autopilot controlling roll, pitch and yaw.
Flight management	Navigation via air data, radio, intertial referencing, GPS. Fuel status. Airport, flight plan database.
Engine control	Engine start, fuel control, cooling, electrical power.
Environmental control	Provision of bleed air, cabin pressure, avionics conditioning, oxygen supply.
Landing gear	Extension/retraction, steering and braking.
Communication	Radio, satellite, air traffic identification, Traffic Collision And Avoidance System (TCAS).
Displays	Process and display data from all systems on flight deck displays
Maintenance	Built-In Test (BIT), provide capability to download and/or display flight data from all systems.

Table 1.3: Common civil avionics functionality collated from Moir and Seabridge [42]

and actuators reduces the amount of cabling required. However it can make physical accessibility for maintenance more difficult and can be more difficult to develop software for [47]. Avionics system designers must also take into account issues such as heat dissipation and separation of redundant units when deciding how to physically package the avionics system into the aircraft. As in automotive software, decentralisation brought about development of standardised communication buses. These were ARINC 429 for civil aircraft and MIL-STD-1553B for military purposes [42]. More recently ARINC 629 has superseded ARINC 429 [42].

Faced with the same system integration problems as the automotive industry, the Integrated Modular Avionics (IMA) platform was developed [47, 42]. Avionics has tended to lead the automotive industry given that IMA development began in 1988 [49] where as AUTOSAR was launched in 2003. Planes such as the Boeing 777 are already in operation using IMA [42, 49].

IMA is a cabinet based architecture. The LRUs of a federated architecture are replaced with Line Replaceable Modules(LRMs) which slot into the cabinet. Multiple cabinets can be used to provide redundancy and to place functionality in the most convenient physical location. Each cabinet contains a built in communications bus such as ARINC 629 and also provides power to the LRMs. This brings reductions in volume, weight and power usage [42]. Standardisation of processor types across LRMs helps to ease application development and improve reuse. IMA architectures also make more efficient use of the available processing power by combining separate functions on to the same LRMs. This has implications to the certification process since it raises the possibility of a task affecting others in unexpected ways especially if a fault occurs [50]. Part of the IMA specification are Application Executives (APEX) which should provide robust partitioning of time and memory for applications running on the same processor [42].

In order to improve design flexibility, IMA contains the concept of blueprints [50]. Blueprints provide a way of delaying and more easily changing design decisions in how the system is configured. This includes definition of the schedule and software / hardware mapping. The ability to change the set of tasks running in the system and their location via blueprints raises the possibility of reconfigurable IMA systems. Reconfiguration mechanisms can be limited to manual reconfiguration on the ground, or dynamic, so that the aircraft can reconfigure itself whilst in flight. This latter option has the potential to enhance adaptability but there are significant challenges in ensuring the safety of such a system [50].

1.4 Thesis Aims

1.4.1 Summary Of Requirements

The trends listed in section 1.3 have led to the development of standardised distributed platforms for hard real-time systems such as AUTOSAR and IMA. Both of these leave the open problem of how to map software tasks and messages to a hardware platform. Tighter integration increases task dependencies and, at the same time, demands for enhanced functionality requires distribution of tasks over a large number of processors. Providing engineers with an efficient way of finding a valid, high quality allocation and schedule for complex systems gives great benefits to both the avionics and automotive industries. This has already motivated a substantial amount of research which features hard real-time task allocation [4, 3, 51, 52, 53, 54, 55, 56]. Further details of this previous work are given in chapter 2.

How tasks are placed and scheduled can impact performance, tolerance to faults, design flexibility and run-time adaptability. Allocation algorithms should therefore consider these goals in addition to finding a solution which is valid in terms of timing requirements. Changing requirements must be managed throughout several iterations of the development process, not only in a post deployment maintenance stage. CASE tools need to evolve existing solutions as well as develop initial ones. Costs of recertification prohibit starting with a new design for every small enhancement.

It is intended that the method of selecting a suitable task mapping will be an automated optimisation method, building on previous work on task allocation. There are a number of requirements which are readily apparent for any CASE tool which is concerned with changeability. In keeping with a flexible development process, the method of generating a suitable solution to the problem must itself be flexible. If the design provides sufficient flexibility to allow the system to grow, then the method of building it should also take account of how future systems will be maintained. More formal requirements and a thesis hypothesis is given in chapter 3 following a more in depth review of related literature in chapter 2.

In the context of this thesis, both flexibility and adaptability are measured in

terms of how many individual changes need to be made to the allocation and schedule of tasks to complete a change to the system instigated by a change agent, internal or external. The change to the system could be a design change caused by a feature enhancement or a run-time reconfiguration initiated by the system itself.

Figure 1.3 shows flexibility and adaptability as dependent upon modifiability, robustness, cost and agility. Reducing the number of changes between two task mappings will aid all of these qualities. If a design change is needed then reusing parts of a solution will make other activities enforced by the changes, such as recertification, cheaper and faster. The reimplementation and testing of the system will be simpler if the change is contained in a small part of the system. If few allocation changes are needed for changes to the properties of tasks then the system is more modifiable with respect to these consequent changes. With respect to the allocation changes themselves, the system is more robust if fewer are needed.

1.4.2 Benefits

The desired outcomes of this work are related to viewing task allocation in the context of all aspects of changeability. The choice of task allocation is by no means the only factor affecting the flexibility of a hard real-time system's architecture. However, a task allocation tool which meets the above requirements would generate solutions which:

1. re-use parts of allocations and schedules from previous development iterations;
2. are flexible — they have the ability to withstand requirements changes without severe disruption to the design;
3. are adaptable — they require few changes to the allocation and schedule when a system reconfigures itself;
4. aid system robustness in the presence of faults by suitable arrangement of redundant tasks.

In general, task allocation tools also assist the development process and improve maintainability because they:

1. automate the task mapping design stage for successive development iterations.
2. help manage complexity by separating task specification from scheduling and task mapping.
3. allow fast what-if analysis to gain insight into new requirements.

1.5 Thesis Structure

This section summarises this chapter and outlines each of the remaining chapters that make up this thesis.

Chapter 1. This chapter has introduced the software architecture qualities of maintainability, flexibility and adaptability and the idea of improving them via automated tool sets. This was then put into the context of hard real-time systems, specifically automotive and avionics systems. The real-time task allocation problem was identified as one which is amenable to automated optimisation methods and is part of the real-time architecture decision making process which can affect the flexibility of real-time systems.

Chapter 2. This chapter reviews related literature and previous work concerned with enhancing flexibility of real-time architectures and solving task allocation problems. Experimental methodologies for assessing optimisation algorithms are also assessed.

Chapter 3. Following the review of related work, this chapter presents requirements for a tool to produce flexible solutions to task allocation problems and a hypothesis stating that certain techniques will be successful at meeting these requirements.

Chapter 4. This chapter presents the real-time architecture model that will be assumed throughout and gives a method of synthesising problem instances for experimentation.

Chapter 5. This chapter explains the design of the basic task allocation algorithm which will be extended in later chapters to support flexibility. The algorithm is configured and evaluated using problems generated by the problem synthesis algorithm from chapter 4.

Chapter 6. This chapter is concerned with the problem of finding similar task allocation solutions for different task sets. This is motivated by multi-moded systems where the task configuration in each mode should be as similar as possible to reduce mode change overheads.

Chapter 7. This chapter shows how change scenarios can be used with the task allocation algorithm developed in chapters 5 and 6 to produce solutions with greater flexibility.

Chapter 8. In this chapter, the task allocation algorithm is extended to generate solutions which degrade gracefully when permanent processor faults occur.

Chapter 9. This chapter summarises the work done in this thesis and assesses whether experimental results support the hypothesis set out in chapter 3.

2

Task Allocation And Real-Time Architecture Optimisation

2.1 Introduction

This chapter reviews literature which will help to uncover the nature of the task allocation problem. The goal is to find strengths and weaknesses of current task allocation work with respect to changeability issues. More general optimisation of flexibility and other real-time architectural qualities is also studied outside of the context of task allocation.

Section 2.2 explains real-time system models and some of the schedulability tests which are used by task allocation algorithms. This leads to a definition of the task allocation problem. This section also covers mode change and fault tolerance mechanisms which are directly related to making systems adaptable.

Section 2.3 presents previous work on task allocation and the solution methods which were used to solve problem examples. This includes a range of work on basic task allocation as well as that which tries to optimise solution qualities.

Section 2.4 covers the definition, modelling and analysis of software architectures. This includes scenario based analysis, a method of predicting how architectural qualities will be affected by possible changes. Section 2.4.4 casts task allocation as a traditional software architecture selection problem and discusses the applicability of scenario based analysis to task allocation. Section 2.4.5 looks at existing work on automation of real-time architecture selection and quality trade-offs.

Section 2.5 summarises the content of the work reviewed in the chapter and identifies the areas of research to be taken forward throughout the rest of the thesis.

Symbol	Description
$\mathcal{T}$	set of all tasks
$\mathcal{M}$	set of all messages
$\mathcal{S}$	$\mathcal{S} = \mathcal{T} \cup \mathcal{M}$, the set of all schedulable objects
τ_i	a schedulable object, i.e. an element of $\mathcal{S}$
$\mathcal{P}$	set of processors
$\mathcal{N}$	set of networks
σ_j	a scheduling resource, i.e. an element of $\mathcal{P} \cup \mathcal{N}$
$\sigma_{j\,lat}$	latency of a network σ_j
$\sigma_{j\,ban}$	bandwidth of a network σ_j
C_i	worst case execution time (WCET) (or message communication time)
D_i	deadline
T_i	period
S_i	maximum size for messages

Table 2.1: Symbols relevant to real-time architecture model components and attributes

2.2 Real-Time Systems Software Architectures

A real-time system is one where the time at which computational results are produced is an integral part of the system's requirements. Definitions of hard and soft real-time systems were previously given in section 1.2.2. This section establishes the main architectural components of a hard real-time system and how models of real-time system architectures can be analysed.

The information is built up as follows. The real-time task model is explained in section 2.2.1. The scheduling of tasks on a single processor, known as *uniprocessor scheduling*, is described in section 2.2.2. The system model and scheduling policies are extended for distributed scheduling in section 2.2.3. This gives the necessary background with which to define a task allocation problem in section 2.2.4.

Sections 2.2.5 and 2.2.6 look at mode change and fault tolerance mechanisms. These are two areas which link a real-time architecture to the system's run-time adaptability. Both impact the task allocation problem.

2.2.1 The Real-Time Task Model

Real-time software is decomposed into functional units called *tasks* [57, 58, 59]. Tasks are activated by an interrupt, generated, for example, by a sensor or clock tick, causing them to release a *job* which is executed on a processor [58, 59]. While this terminology is common, it is not universal, for example Liu [40] uses the same terms differently. Sometimes "task" is used more loosely to describe both the unit of functionality and the executing objects they generate. The set of all tasks is denoted $\mathcal{T}$ in this thesis. A full list of symbols related to the real-time system model used in this section and future chapters is given in table 2.1.

At the design level, tasks are assumed to be able to run concurrently unless an explicit dependency requires that one cannot start until it is triggered by a message

from another. The system may also contain shared resources when one task is blocked waiting for access to a resource another task is currently using.

2.2.1.1 Periodic Task Model

The simplest recognised real-time task model is the purely periodic model where each task τ_i has a worst case execution time (WCET) C_i and period T_i. All tasks run in a continuous loop releasing a job every T_i time units which executes for a maximum duration of C_i time units. Such a model is suitable for pure control systems [60] such as a PID (Proportional, Integral and Derivative) controller [40]. Each task uses a proportion C_i/T_i of the processing resource. This quotient is called the *utilisation* of the task. If U_i is the utilisation of each task τ_i then the total utilisation, $\sum_i U_i$, is the *task set utilisation*.

2.2.1.2 Sporadic Task Model

The maximum amount of time allowed from the point a job can be released to the point it completes execution is the *relative deadline* of the task, denoted D_i. In the simple periodic task model, a task set is said to be schedulable under a particular policy if every job released by a task always completes before the task releases another job. That is, the relative deadline of a task is equal to its period so $D_i = T_i$ for all i.

In some systems, deadlines are allowed to take arbitrary values making D_i an explicit task attribute. An obvious motivation is to specify requirements for applications containing tasks which need to respond in a shorter time-frame than the length of the task's period. Setting arbitrary deadlines is also used to influence other timing properties. It is often the end-to-end deadline, the time taken for data captured by a sensor to be processed by a chain of tasks resulting in actuation, that is important rather than intermediate deadlines for tasks in the chain. Intermediate deadlines can be set to achieve this and may be both longer or shorter than periods [61].

Sporadic tasks [57] also motivate arbitrary deadlines [62]. Sporadic tasks are those which do not need to run continuously but are activated to respond to an event such as a button press. For a sporadic task, the value T_i represents the minimum inter-arrival time of two successive jobs. The maximal response time required by the application may be significantly lower than the minimum inter-arrival time so sporadic tasks can have deadlines which are much smaller than periods [62].

2.2.2 Scheduling Policies And Analysis For Uniprocessor Systems

A major design decision in the construction of a real-time system architecture is the choice of scheduling policy. There are many scheduling policies to select from and some systems use more than one. Partitioned systems can take a hierarchical approach where each partition is scheduled using one particular policy and then each partition is assigned time on the processor by a separate policy [63]. This is the

Task	C_i	T_i
τ_1	150	500
τ_2	200	1000
τ_3	250	1000
τ_4	150	1000

Table 2.2: Example task set with four tasks

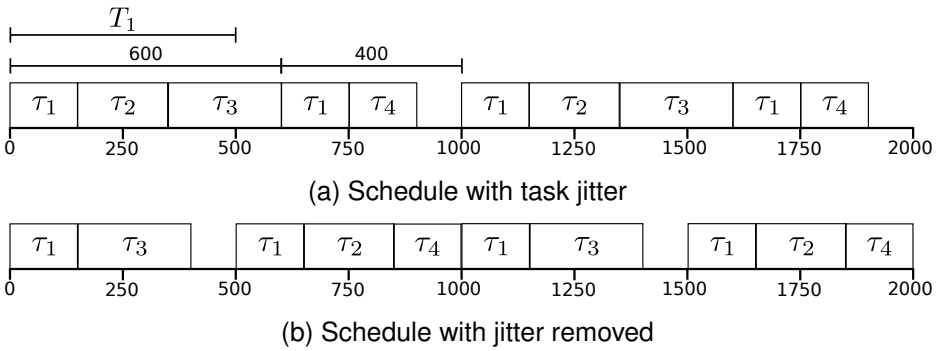


Figure 2.1: Possible static cyclic schedules for task set in table 2.2

approach used by the APEX standard in IMA systems. The tasks within each partition are scheduled using a *fixed priority* policy and each partition is allocated time within a *static cyclic* schedule. There are three major categories of scheduling policy: static cyclic scheduling, fixed priority scheduling and dynamic priority scheduling. All three are now explained in the context of single processor systems.

2.2.2.1 Static Cyclic Scheduling

A static cyclic schedule is one where each task executes at a pre-determined time relative to a time chosen to be time zero. If all tasks have the same period then tasks can be laid end-to-end and then the sequence is executed at times $0, T_1, 2T_1, 3T_1, \ldots$ where T_1 is the period of the first and every other task. System requirements usually dictate that there are tasks which need to run at different rates. For example, an engine temperature sensor does not need to sample as frequently as a rotation speed sensor [40]. Enforcing all tasks to run at the rate of the task with the shortest period requirement is highly inefficient. Therefore, a cycle of length equal to the lowest common multiple (LCM) of the task periods is created with each task repeated the correct number of times within each cycle.

Table 2.2 shows a task set containing four tasks where the LCM of the periods is 1000. Figure 2.1a shows two repetitions of a possible cyclic schedule for this task set. Since task τ_1 has a period half the length of the cycle, it features twice within one repetition of the cycle. As shown in figure 2.1a, the gap between two executions of this task is never exactly equal to its period of 500 but alternates between 400 and 600. This feature of a task's schedule is called *jitter*. Tasks can exhibit both input and output jitter [64], defined as:

Task	Figure 2.1a		Figure 2.1b	
	WCRT	Jitter	WCRT	Jitter
τ_1	250	100	150	0
τ_2	350	0	850	0
τ_3	600	0	400	0
τ_4	900	0	1000	0

Table 2.3: Worst case response time calculations for schedules in figure 2.1

- *input jitter* — variation in the interval between start times of successive jobs
- *output jitter* — variation in the interval between end times of successive jobs

Due to the assumptions of control algorithms, jitter can cause degradation in control performance and possibly instability [65]. Therefore, static schedules should be constructed to minimise jitter for tasks related to jitter sensitive applications.

An alternative schedule is shown in figure 2.1b where the gap between start time of all tasks in exactly equal to their periods. In general, it is not always possible to achieve this but, at the same time, not all tasks will be part of an application where jitter is of paramount importance.

Once a static cyclic schedule is constructed, each job of a task within the schedule can be checked to find the worst case response time (WCRT) of the task. The worst case is given by the job which has the longest duration between the start of the period and the end of its execution. For the example in figure 2.1, this results in examining two instances of task τ_1 and a single instance of tasks τ_2, τ_3 and τ_4. The WCRTs for both of the possible schedules shown in figure 2.1 are given in table 2.3. This shows that the reduction in jitter for task τ_1 comes at the expense of an increase in WCRT for some tasks with task τ_4 only just meeting its deadline assuming $D_4 = T_4$.

Architectural decisions for hard real-time systems are driven by timing predictability over and above absolute performance. Static cyclic scheduling excels in this regard since the execution of each task is pre-determined. The schedule can also be optimised for tasks related to jitter sensitive applications.

There are drawbacks to a static cyclic scheduling policy. The static-by-design nature of this policy can lead to fragile schedules which need to undergo a large amount of redesign and retesting when new tasks are added or task execution times change [66, 67].

As in the example previously given with a temperature sensor and rotation speed sensor, periods can vary widely within a system. Locke [66] identifies a number of problems of engineering static cyclic schedules containing tasks with different periods. If the periods of the highest rate and lowest rate tasks are different orders of magnitude then a very long schedule must be created with several repetitions of some of the tasks. Slow running tasks can be split into several parts, with a different part running in successive cycles. However, this increases the number of tasks and their dependencies and hence shifts rather than reduces the engineering overheads.

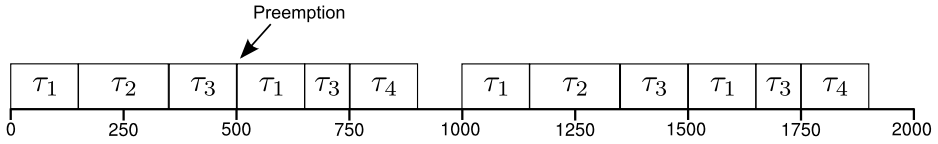


Figure 2.2: Fixed priority scheduling with preemption

A long cycle can also be generated if the task periods are not harmonic. This can lead to task periods being selected according concerns regarding the engineering of the schedule as opposed to the best design for the control application.

2.2.2.2 Fixed Priority Preemptive Scheduling

Fixed priority preemptive scheduling is a commonly used scheduling policy in hard real-time systems. Under this policy, each task is assigned a priority as part of the design process and this priority remains constant while the system is running. When more than one job is active, the highest priority one is chosen to execute. If a job becomes active while a lower priority job is executing, then the lower priority job is suspended until there are no higher priority jobs currently active. Figure 2.2 shows the task set in table 2.2 scheduled using a preemptive fixed priority scheduling policy. Task τ_i is assigned priority i with the task at priority 1 being the highest priority task. After 500 time units, a job of task τ_3 is executing but is preempted by τ_1 when it becomes active.

Unlike the static cyclic schedule, where tasks release a job at a predetermined time, a job of τ_j is released at every multiple of its period $0, T_j, 2T_j, 3T_j, \ldots$. Execution of the job will begin when all higher priority jobs have completed their execution. This means that the time at which higher priority jobs complete and a lower priority job begins its execution will vary from one period to the next possibly leading to higher levels of jitter than a static cyclic schedule, especially for lower priority tasks. This can be mitigated by the use of offsets [58].

Compared to static cyclic scheduling, fixed priority preemptive scheduling has reduced engineering overheads whilst retaining predictable timing properties. In particular, fixed priority scheduling is more readily able to handle tasks with a wide range of periods [66].

Static cyclic scheduling takes a correct-by-construction approach to ensuring all tasks meet their deadline. This cannot be said of fixed priority scheduling since there are many possible interleavings of tasks which must be verified for any particular chosen priority ordering. The seminal work of Liu and Layland [60] established two important results regarding single processor fixed priority preemptive scheduling. Firstly, the optimal priority assignment is a rate monotonic assignment where higher rate tasks are assigned a higher period. The term “optimal” is used in the sense that if there exists a priority ordering which allows the task set to be scheduled, then it will also be schedulable using a rate monotonic ordering. Liu and Layland [60] also

proved that, using a rate monotonic priority ordering, any task set with the property

$$\sum_{i=1}^{K} \frac{C_i}{T_i} \leq K(2^{1/K} - 1) \tag{2.1}$$

will be schedulable where $K = |\mathcal{T}|$, the number of tasks in the task set.

If deadlines are not equal to periods, then a rate monotonic priority ordering is no longer optimal [67]. Audsley [67], citing Leung and Whitehead [68], states that Deadline Monotonic Priority Ordering (DMPO) is optimal when all task deadlines are less than or equal to the task period.

Inequality (2.1) gives a least upper bound on the utilisation of schedulable task sets. Task sets with higher utilisations still have the potential to be schedulable but require further schedulability tests.

An upper bound of the WCRT of each task can be calculated and compared to its deadline. Since the pattern of task interferences will repeat after a duration equal to the lowest common multiple of the task periods, tasks can be laid out using the rules of fixed priority preemptive scheduling policy as shown in figure 2.2 and then each instance of each task can be examined to find the WCRT. This method does not scale well to systems with several tasks running with different periods. A more efficient form of analysis for calculating R_i, the WCRT of task τ_i, is given by Joseph and Pandya [69] based on the following equation

$$R_i = C_i + \sum_{j \in hp(\tau_i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j \tag{2.2}$$

where $hp(\tau_i)$ is the set of values which index tasks with a higher priority than τ_i. R_i is on both sides of this equation. Joseph and Pandya proved that the lowest valued solution to the following recurrence relation will solve equation (2.2) and provide the correct value for the response time of task τ_i.

$$R_i^{(q+1)} = C_i + \sum_{j \in hp(\tau_i)} \left\lceil \frac{R_i^{(q)}}{T_j} \right\rceil C_j \tag{2.3}$$

The following value for $R_i^{(0)}$ can be used since the duration of the WCRT has to include at least one execution of all higher priority tasks as well as the task itself.

$$R_i^{(0)} = \sum_{j \in hp(\tau_i) \cup \{i\}} C_j \tag{2.4}$$

Equation (2.3) is calculated for $q = 1, 2, \ldots$ until $R_i^{(q+1)} = R_i^{(q)}$ or $R_i^{(q)}$ is greater than the deadline D_i in which case the task set is not schedulable. This equation only holds for independent tasks on a single processor with task deadlines less than or equal to periods. Table 2.4 gives the WCRT values for the task set previously

Task	C_i	T_i	π_i	R_i
τ_1	150	500	1	150
τ_2	200	1000	2	350
τ_3	250	1000	3	750
τ_4	150	1000	4	900

Table 2.4: Worst case response time calculations for fixed priority preemptive scheduling

given in table 2.2 with the priority of task i, π_i, set to i.

Other elements can also be introduced into the response time analysis. A more general form of equation (2.3) is

$$R_i = J_i + B_i + C_i + \sum_{j \in hp(\tau_i)} \left\lceil \frac{R_i + J_j}{T_j} \right\rceil C_j \tag{2.5}$$

This equation includes the term J_i to represent the release jitter of task i and B_i which is the longest possible time a task can be blocked by a lower priority task when competing for a shared resource. Jitter terms can also be used to model delays caused by precedence constraints. This is the case for the multiprocessor scheduling analysis given in section 2.2.3.5.

It is also possible to use a non-preemptive fixed priority scheme where, once a job begins to execute, it will always continue to completion, potentially causing higher priority tasks to be blocked. A good example is scheduling messages on a CAN network where communications cannot preempt each other. Davis et al. [70] provide schedulability analysis for this case. The longest amount of time a task can be blocked is equal to the maximum WCET of lower priority tasks

$$B_i = \max_{j \in lp(\tau_i)} C_j \tag{2.6}$$

Using equation 2.5 to calculate WCRT values for non-preemptive systems is not valid in all circumstances. It can give a result which is overly optimistic and not a true worst case. The situation which causes this is when a lower priority task blocks the release of a higher priority one which in turn delays the start of the next job of the lower priority task. In effect, jobs released by the same task are simultaneously active and interfere with each other. To overcome this, the analysis must be run for multiple instances of each task and the maximum response time can then be used for the worst case. Analysis which achieves this is presented in section 2.2.3.5. Similar reasoning can also be used for analysis of systems with deadlines greater than periods and the analysis is also valid for this situation.

2.2.2.3 Dynamic Priority Scheduling Policies

Another class of scheduling policies are dynamic priority based policies which allow priorities to change at run-time. Liu and Layland described a deadline-driven approach [60] whereby the job currently executing is always the one with the earliest deadline from the current point in time or the processor is idle. That is, the job with the earliest deadline is given the highest priority. Using this scheme allows all tasks to be scheduled as long as the condition

$$\sum_{i=1}^{K} \frac{C_i}{T_i} \leq 1 \tag{2.7}$$

holds. This policy is optimal in terms of scheduling task sets since no policy can schedule a task set which requires over 100% over the processor's resources.

Earliest Deadline First (EDF) scheduling is far less commonly found in automotive and avionics systems than static cyclic and fixed priority scheduling. It is a more complex policy to implement, requiring more kernel support than fixed priority policy implementations [71]. In overload situations, it becomes difficult to predict which task will be running and is likely to miss a deadline [72] though improvements in analysis of EDF have given cause to dispute this [71]. In comparison, it is known in advance that the lower priority tasks of a fixed priority system are more likely to miss deadlines. Although some work in the automotive domain has considered the use of EDF scheduling for engine management [73], this thesis will concentrate on the historically more dominant static priority and static cyclic policies.

2.2.3 Distributed Scheduling

For uniprocessor systems, the major real-time architectural components are tasks and the single processor. When this is extended to a distributed multiprocessor platform then networks and messages as well as further processors must also be modelled.

Messages can be scheduled using static cyclic, fixed priority or hybrid policies [74, 75]. The commonly used CAN bus [45] schedules messages using a fixed priority non-preemptive policy [70]. It is also possible to use preemptive policies for real-time communications [76]. For example, the ARINC IMA standard assigns messages to channels and allows preemptions [77] within channels.

The time taken to transfer a message on a CAN bus is [70]

$$(k + 10s_i)\sigma_{j_{bit}} \tag{2.8}$$

where $\sigma_{j_{bit}}$ is the time taken to transfer an individual bit on network σ_j, s_i is the size of the message in bytes and k is a constant which depends on whether standard or extended frame identifiers are being used. The frame identifiers also contain the message priority in order to perform bus arbitration [70]. By setting a latency value

$\sigma_{j_{lat}} = k\sigma_{j_{bit}}$ and a bandwidth value $\sigma_{j_{ban}} = 1/(10\sigma_{j_{bit}})$, equation (2.8) can be rewritten as

$$\sigma_{j_{lat}} + \frac{s_i}{\sigma_{j_{ban}}} \tag{2.9}$$

In the case of CAN, $\sigma_{j_{lat}}$ represents an overhead for frame headers and bit stuffing which enables error detection [70]. More generally, this formula expresses a transmission time dependent upon the size of the message with some constant overhead which could be for data overheads or generation and delivery delays [76]. If s_i is substituted for S_i, the maximum size of a message, then equation (2.9) becomes an expression for the message worst case communication time (WCCT) C_i.

$$C_i = \sigma_{j_{lat}} + \left\lceil \frac{S_i}{\sigma_{j_{ban}}} \right\rceil \tag{2.10}$$

Values of C_i are usually specified as integers. Equation (2.10) uses a ceiling function to always round values up since C_i represents a worst case value.

Static cyclic policies are also used in automotive systems. Flexray, a more modern standard than CAN, uses a hybrid policy with two transmission windows, one containing a static cyclic schedule for control tasks requiring low jitter and the other intended for scheduling sporadic event driven tasks [75]. Similarly Time Triggered Controller Area Network (TTCAN) [78] is an evolution of the CAN protocol which allows a hybrid scheduling approach to be used. Search optimization methods, specifically genetic algorithms, have been used to optimise how resources are divided between different policies within TTCAN [79]. Note that purely time triggered system are poor at handling sporadic events [66] since the all tasks must be included in a pre-defined schedule and run continuously.

2.2.3.1 Messages And Task Dependencies

Tasks may need to receive data from multiple tasks, for example a task which compares or combines values from separate sensors. Similarly a task may generate information which needs to be processed by multiple tasks as in the case of a speed sensor being used for both engine management and cruise control. Such dependencies between tasks can described by a directed graph, as shown in figure 2.3, with tasks represented as graph nodes and messages forming the edges connecting the nodes.

The restrictions on the structure of the graph depend on the ability to analyse the model and verify that timing requirements are met. Figure 2.3 shows an acyclic graph with a single initiating task which allows tasks to both send and receive multiple messages. Dependencies in task graphs can also have different semantics. Two important variations are [40]:

- AND / OR relationships. When a job receives multiple messages, does it depend on all of them (AND) or any one of them (OR)?

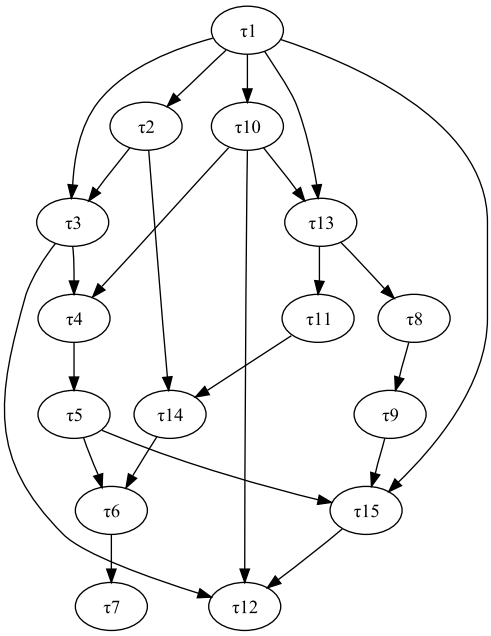


Figure 2.3: Example acyclic task graph

- Temporal precedence constraints. Does a job need to wait until a predecessor completes or can it start as soon as the predecessor produces some output?

Once again, in the context of a hard real-time system, it is the ability to analyse the model which impacts implementation decisions. A common assumption is that a job does not produce any output until it completes and successors cannot use any of the information until they have received a message from the previous job. A sequence of dependencies forming a task graph is called a *transaction*.

2.2.3.2 Event Triggered Versus Time Triggered Architectures

Periodically activated tasks are described as *time triggered* since they release a job according to a timer event. Sporadic tasks, activated by an external event are *event triggered*. In a distributed system, where information must pass through a chain of tasks across several processors, the release of a job or the sending of a message can also be time triggered or event triggered [74, 43, 75]. Each approach has advantages and disadvantages, and combinations of the two approaches are possible [74, 75].

A completely time triggered system with asynchronous communication is simple to implement and easy to analyse. All tasks are periodically activated reading from and writing to shared buffers [80, 81]. Periodically activated messages transfer the most recently written data from a task's output buffer to the input buffer of the destination task. In such a system, changing one schedule does not directly effect when tasks or messages will run on other scheduling resources.

The time triggered approach goes hand-in-hand with a static cyclic schedule resulting in a system with a small amount of jitter [75]. However, the end-to-end

response time referring to how long it takes a piece of information to be processed by the entire chain of tasks can be affected by any change to a schedule on any processor or network. Asynchronous buffered systems tend to have typically poor end-to-end response times [75] since it is possible for a task to run just prior to new data arriving resulting in it having to wait nearly a whole cycle to be processed.

It is difficult to support sporadic tasks in a purely time triggered system. If time triggered messaging is used to support event triggered tasks then the messages must be sent continuously regardless of whether an event has occurred. This results in a lot of wasted communication but also makes it far easier to detect a failed node. If messages are event triggered then it is not known whether an absence of messages is due to no event occurring or a failure [43].

An event driven system with a priority based scheduling policy is better suited to incremental design [43]. However, all tasks and messages dependencies also become response time calculation dependencies and so changes to schedules propagate throughout the system [75].

2.2.3.3 Analysis Of Time Triggered Multiprocessor Static Cyclic Schedules

Multiprocessor systems using asynchronous communication can be analysed with simple extensions to uniprocessor response time calculations in order to calculate end-to-end response times. The longest time that a piece of data will take to be processed by a task or transferred by a message is its response time, R_i, as calculated from uniprocessor scheduling analysis plus its period, T_i. The worst case situation occurs when every task throughout a chain of tasks takes this amount of time [81]. Therefore the worst case end-to-end time is the longest path through the task graph with the cost of each node and edge being $R_i + T_i$. If, however, the system is assumed to be synchronised with a global clock then this can be reduced [64].

2.2.3.4 Multiprocessor Schedulability Tests For Priority Based Policies

Constructing simple sufficient schedulability tests with a reasonable degree of pessimism, as in equations (2.1) and (2.7) for uniprocessor systems, is much more difficult in the multiprocessor case. This stems from the fact that any schedulability test must be in the context of a task to processor allocation as well as priority assignment policy. Leading work on multiprocessor schedulability tests assumes the sporadic task model but without dependencies [82, 83]. Schedulability tests can be divided into four categories based upon the priority assignment and allocation policies:

1. static priority assignment / partitioned scheduling
2. dynamic priority assignment / partitioned scheduling
3. static priority assignment / global scheduling
4. dynamic priority assignment / global scheduling

With partitioned scheduling, each task is pre-assigned to a processor at design time and then each processor can use any uniprocessor scheduling policy. Global scheduling policies maintain a single job queue and each job is dynamically assigned to a processor according to a particular policy [83]. Global scheduling policies may also allow *job migration*, whereby a job can migrate between processors during its execution [83] though this typically introduces overheads, especially when cache is not shared between processors [84].

With a combination of partitioned scheduling and EDF dynamic priority assignment on a homogeneous platform, the task allocation problem becomes a bin packing problem [82] since it is known that each processing resource provides 1 unit of utilisation. This type of problem is NP-hard. Baruah and Fisher give a polynomial time partitioning algorithm [82] which will schedule any feasible task set on processors which are $3 - 1/|\mathcal{P}|$ times as fast as would be needed using an optimal allocation policy. This result applies to task sets with all periods less than or equal to deadlines.

2.2.3.5 Analysis For Event Triggered Multiprocessor Fixed Priority Scheduling

Palencia and Harbour [59], building on the work of Tindell and Clark [85], developed schedulability analysis for fixed priority scheduling with static and dynamic offsets. This is used to model an event triggered multiprocessor system where the static offset is the minimum amount of time an object has to wait for preceding objects to complete and the dynamic offset is the maximum amount of additional waiting time. This analysis is a schedulability test for a system using partitioned allocation / fixed priority assignment scheduling policies for tasks and messages with dependencies. As long as WCETs are known for the chosen allocation, there is no restriction on the heterogeneity of the hardware platform. The analysis is suitable for a sporadic task system with both $D_i < T_i$ and $D_i > T_i$ as well as $D_i = T_i$.

The objects in the system are grouped into transactions so that τ_{ab} is object b within transaction a. The dynamic offset for τ_{ab} is represented as a jitter term, J_{ab}. The static offset, Φ_{ab}, is the minimum delay a task can cause to others. This can be any lower bound on the best case response time of the task. A suitable estimate is the sum of the task's best case execution time (BCET) and the shortest path leading to that task where the cost of each node and edge is the BCET of each task and message. The calculations rely on the period of all objects within a transaction being the same. T_a is the period of all objects within transaction a.

Equation (2.12) calculates W_{ik}, the interference on τ_{ab} due to objects in transaction i up to time t. This is in the context of a continuous period of execution, initiated by τ_{ik}, of objects on the same processor and with a higher priority than τ_{ab}. τ_{ik} is called the critical object. Equation (2.13) finds the maximum value of W_{ik} over different values of k to calculate an upper bound on the worst case interference.

$$\phi_{ijk} = T_i - (\Phi_{ik} + J_{ik} - \Phi_{ij}) \mod T_i \tag{2.11}$$

$$W_{ik}(\tau_{ab}, t) = \sum_{j \in hp_i(\tau_{ab})} \left(\left\lfloor \frac{J_{ij} + \phi_{ijk}}{T_i} \right\rfloor + \left\lceil \frac{t - \phi_{ijk}}{T_i} \right\rceil \right) C_{ij} \tag{2.12}$$

$$W_i{}^*(\tau_{ab}, w) = \max_{k \in hp_i(\tau_{ab})} W_{ik}(\tau_{ab}, w) \tag{2.13}$$

ϕ_{ijk} gives the time between the critical instant defined by τ_{ik} and the last activation of τ_{ij} before it. The situation for worst case interference is when the critical object is delayed as much as possible and then activations of tasks after the critical instant have 0 delay [59].

Since this analysis covers cases with deadlines and jitter larger than periods, a task can have multiple jobs active at one time. Equation (2.14) calculates $w_{abc}(p)$ which is the longest period of time where instance p of τ_{ab} is delayed by instances of higher priority tasks and other instances of τ_{ab} for a period of continuous execution initiated by τ_{ac}.

$$w_{abc}(p) = B_{ab} + (p - p_{0,abc} + 1)C_{ab} + W_{ac}(\tau_{ab}, w_{abc}(p)) + \sum_{i \neq a} W_i{}^*(\tau_{ab}, w_{abc}(p)) \tag{2.14}$$

Equation (2.14) is a recurrence relation which is solved when the value of w_{abc} converges. To obtain an initial value, at the beginning of the analysis calculation, all response times and jitter values are set to 0. Afterwards, the jitter value for a task can be taken as the response time of its incoming message. If there are multiple messages, the one with the highest response time is used. Similarly the jitter for a message is set by the WCRT of the task sending it.

To obtain a response time $R_{abc}(p)$, $w_{abc}(p)$ is adjusted according to the instance number and the phase relationship with the critical object τ_{ac}.

$$R_{abc}(p) = w_{abc}(p) - \phi_{abc} - (p - 1)T_a + \Phi_{ab} \tag{2.15}$$

The WCRT is then found by checking all instance values between $p_{0,abc}$ and $p_{L,abc}$ for possible critical objects in the same transaction as the object being studied. The first possible activation is

$$p_{0,abc} = -\left\lfloor \frac{J_{ab} - \phi_{abc}}{T_a} \right\rfloor \tag{2.16}$$

The final task activation which needs to be checked is initialised to $p_{0,abc}$ and, in future iterations, updated to be

$$p_{L,abc} = \left\lceil \frac{w_{abc} - \phi_{ac}}{T_a} \right\rceil \tag{2.17}$$

Finally, the WCRT of object τ_{ab} is

$$R_{ab} = \max_{c \in hp_a(\tau_{ab}) \cup b} \left(\max_{p = p_{0,abc}, \ldots, p_{L,abc}} R_{abc}(p) \right) \tag{2.18}$$

2.2.3.6 Pessimism and Efficiency Of Scheduling Analysis

Scheduling analysis that is not based on constructing a static schedule is generally not able to calculate exact WCRTs. Since scheduling analysis is typically used in hard real-time systems where there must be certainty that deadlines will be met, most analyses, including that in section 2.2.3.5, produce a pessimistic result. This results in some feasible schedules being declared infeasible. If pessimism is too high, the only task sets which will be accepted will use only a small amount of the available hardware resources leading to inefficient system designs.

For automated task allocation algorithms, the schedulability analysis is at the heart of the objective function. More pessimistic analysis will generally be more efficient to calculate so more configurations can be evaluated in a certain duration. However, more accurate analysis has the potential to increase the number of valid configurations in the design space.

More accurate versions of the Worst Case Dynamic Offset (WCDO) analysis in section 2.2.3.5 are available. Palencia and Harbour improved upon it with Worst Case Dynamic Offset with Priority Schemes (WCDOPS) analysis [86] but this did not allow branching in the task graphs. Redell introduced WCDOPS+ [87] which reduced pessimism further and also allowed for branching in the task dependency graph. Kany and Madsen made further adjustments to take account of task dependency graphs which had tasks receiving more than one message with analysis they call WCDOPS++ [88]. All versions of WCDOPS rely on the concept of H-segments which are sequences of dependent objects running on the same processor and preceded by the same lower priority object. None of the WCDO variants are able to cope with cycles in task graphs since the delay caused by a task waiting for an incoming message to trigger it could be dependent upon the response time of the same task creating a never-ending loop in the analysis. Loops can be broken by splitting a task into separate instances [59] but all task graphs in this thesis will be acyclic.

2.2.4 Task Allocation

The task allocation problem is the problem of allocating tasks and messages to a hardware platform and constructing a suitable schedule which ensures all tasks and messages meet specified deadlines. As is evident from the different possible real-time models, this statement alone is not sufficient to precisely define a problem which can be solved algorithmically. Some of the questions which need to be answered to define the problem are: Are there task dependencies? Are communications synchronous or asynchronous? Are the processors homogeneous? What is the scheduling policy and which schedulability test should be used? Are there constraints other than timing?

Fixed priority scheduling with DMPO and without dependencies is an allocation only problem since the priority ordering is decided efficiently by a deterministic algorithm. However, the utilisation based schedulability test given in section 2.2.2.2

is a sufficient but not necessary test. The problem is more complex than a simple bin packing problem because schedulability is not determined by resource usage alone.

If a static cyclic scheduling policy is to be used, then the schedule as well as the allocation must be constructed. This can be done with the use of an ordering attribute [53] in which case the algorithm for converting a task ordering into time slots becomes part of the problem.

When dependencies are introduced, then constructing a schedule to meet end-to-end deadlines becomes an issue. Use of fixed priority scheduling requires a suitable priority ordering as well as allocation since DMPO is no longer necessarily optimal. With asynchronous communications, changing priority of tasks on one processor will not affect the schedulability of tasks and messages with respect to local deadlines. However end-to-end deadlines can be affected by any change which affects the schedule of objects in the transaction. For an event triggered synchronous system, changing priority of a task can affect schedulability of tasks and messages on other processing resources with respect to both local and end-to-end deadlines.

Even when two task allocation problems are constructed from the same system model, the difficulty can still vary depending on the schedulability test being used. For example the admissible set of solutions will vary between the WCDO test [59] and WCDOPS+ test [87] for certain task sets.

In light of these variations in the nature of task allocation problems, this thesis defines a task allocation problem as follows.

Definition 2.1. *The task allocation problem is the discovery of a mapping of tasks and messages to a well defined hardware platform and selection of task and message attributes required by a chosen scheduling policy so that all tasks and messages are deemed schedulable by a schedulability test appropriate to the scheduling policy.*

A well defined hardware platform is a collection of processors and networks for which task and message execution times are known for all potentially valid software to hardware mappings. If a mapping assigns a message to a network in a way that does not allow it to be transmitted between its source and destination tasks according to the network topology then it has an infinite execution time.

This definition is applicable to all hard real-time task allocation problems. Particular problem instances may place additional constraints on the solution such as allowing a particular task to only be assigned to a subset of the processors. The above formulation is a constraint satisfaction problem since all valid solutions are considered equivalent in terms of quality. It can be converted into a constrained optimisation problem by the addition of a metric which differentiates between solution qualities. The system model, scheduling policies and schedulability tests used in task allocation problems in this thesis are stated in chapter 4.

2.2.5 Modes And Mode Changes

2.2.5.1 Definitions Of Modes

For many hard real-time systems, the design of task allocations and schedules is further complicated by mutually exclusive phases of operation called *modes* where the system will need to provide different functionality or perform differently (e.g. provide same functionality slower but with fewer errors). A commonly given example of a multi-moded system is an aircraft taking off, moving into level flight and then landing [30, 89, 90, 91]. Real and Crespo identify some generic operational modes which can be found in complex hard real-time systems: initialisation mode, maintenance related modes (running diagnostics, data downloads), low power mode, emergency mode (running with reduced resources), alarm mode (reacting to a high risk situation), high reliability mode and fault recovery mode. Another more subtle and more specific example is found in the analysis of an automotive electronic height control system by Stauner et al. [92]. This system automatically adjusts chassis height via pneumatic suspension so that the chassis remains in the same position relative to the wheels within specified tolerance levels. Three modes are analysed: stopped, driving normal and driving in a bend. For safety reasons, the height adjustment must be suspended whilst driving in a bend.

As discussed in section 2.2.3, purely time triggered systems with pre-defined schedules are unable to adapt to changes in environment since tasks cannot be inserted into the schedule once the system is running. Creating a set of pre-defined time triggered schedules corresponding to different modes and allowing events to trigger changes between modes can make a time triggered architectures more adaptable [90].

A change in the behaviour of the system does not necessarily imply a change to the schedule. For a time triggered system, a slot can be maintained for all tasks in all modes whether needed or not. This could be the case in the electronic height control example where the height adjustment actuation task could be made to cause no physical effect when driving in a bend. In general, however, the "all modes in one schedule" approach is impractical since it results in severe under utilisation of resources [91, 93]. Fohler [91] defines a mode as "an operational phase which is performed by a single schedule", referring to a uniprocessor multi-moded system. This is a view echoed by Pedro who also emphasises the mapping between a schedule and the set of functions and performance the system can provide using that schedule.

> *"A mode is defined by the behaviour of the system, described by a set of allowable functions and their performance attributes, and hence by a single schedule, containing a set of processes and their timing parameters."*
>
> – Paulo Pedro [93]

To define the modes for a distributed multiprocessor system, Pedro uses all combinations of uniprocessor modes which are a valid system configuration [93]. Whilst

this view is sensible from the perspective of analysing timing requirements of multi-moded systems, the requirements for operational phases from which modes will be derived will arise prior to task allocation in the design process. Therefore it seems preferable to define a mode in terms of the set of tasks and messages and their attributes involved in the operational phase which will need to be allocated and scheduled rather than the schedule itself.

Giotto is a programming language for time triggered real-time systems which includes semantics to support several modes within a system [89]. Within Giotto, a mode is defined as follows.

> *"Formally, a mode consists of a period, a set of mode ports, a set of task invocations, a set of actuator updates, and a set of mode switches."*
>
> – Henzinger et al. [89]

The period of the mode is the length of the schedule associated with the mode. Mode ports provide a way of communicating data from one mode to the next. The set of mode switches is the set of possible mode transitions from the current mode. A mode will periodically check exit conditions triggered by sensors which will cause it to perform a switch to a different mode. Giotto requires a compiler to assign tasks to processors and construct schedules after the completion of the system design at the Giotto level of abstraction which includes modes.

2.2.5.2 Mode Changes

During a mode change, a task can stop, start or continue to run. If it continues to run, some of its attributes, (C_i, D_i, T_i), may change or it can continue as before. If the execution time, C_i, changes then it may be easier to model the task as two separate tasks with one stopping in the old mode and the other starting in the new mode [30]. If the task set changes, then the messages sent across the bus are also likely to change. As with other scheduling issues in a hard real-time system, it must be analysable and predictable [91, 93, 30]. Real and Crespo outline four requirements for a mode change [30]:

1. Schedulability — during the mode transition, task deadlines must continue to be met for tasks in the task sets of both modes
2. Periodicity — periodic control tasks must continue to run with the same activation pattern, i.e. not suffer excess jitter during the mode change
3. Promptness — there may be a deadline on completing the mode change, especially when transitioning into an emergency mode
4. Consistency — steps must be taken to keep all shared data consistent

Something which is given little attention in the literature on analysing mode changes is the possibility of a task continuing to run but migrating to another processor during the mode change. Of course, this can be modelled as a task stopping

on one processor and a new one starting on another though this ignores the overhead of migration. An area where task migration becomes useful is that of reconfigurable real-time systems which is usually discussed at a more abstract architectural level than mode changes. In IMA systems, run-time reconfiguration is a way of improving system dependability [94, 50]. If a processor shows signs of a fault then there is the possibility of migrating tasks away from it which may require other non-critical tasks to be stopped. Therefore, the concepts of mode changes for providing different functionality and reconfiguration to improve dependability are closely linked. Both are key to making real-time systems adaptable.

If task migrations are not supported, then a task allocation must be found which allows tasks which are common between modes to remain on the same processor. If tasks are continuing to run between modes, then maintaining the same priority level will make the system easier to analyse and is less likely to affect the amount of jitter the task typically experiences. If however, the system is reconfigurable then task migration should be taken advantage of to maximise functionality. At the same time, unnecessary migration should be minimised due to the overheads involved. This introduces extra difficulty to the basic task allocation problem defined in section 2.2.4.

2.2.6 Fault Tolerance

Given the close relationship between hard real-time systems and safety-critical systems, it is, perhaps, not surprising that there is an emphasis on fault tolerance in hard real-time systems' architectures. To construct a fault tolerance mechanism that will ensure the system remains in a safe state, it is important to understand the types of faults that can occur and how they can be handled.

2.2.6.1 Faults And Failures

If a system deviates from its specification, then it is said to have failed. A fault, such as a software bug, can be present in a system without causing a failure if the system does not receive inputs which cause the fault to produce an undesired effect. In this case, it is said to be dormant [95]. The intermediate state between a fault and a failure is an error [39, 95]. An error is an unintended internal state [39] that still may be corrected before it manifests itself as a failure. In a hierarchically decomposed system, a failure of one component can cause a fault in another [95].

Faults can be *transient* or *permanent*. A transient fault is one which is temporary and only present in the system for a short period of time. Transient faults are common in communications channels which are subject to electromagnetic interference [96]. A permanent fault cannot be removed from the system without repair [95].

Since the correct system behaviour depends on both the value of a result and the time it is delivered, there are two possible types of failure: a timing failure and a value failure [39, 95]. A timing error has occurred if a resultant output is too early,

too late or never arrives [95].

When a component fails, there are a number of ways it can behave. If a component is *fail silent* [96, 39] then it produces no output once it has failed. This is preferable to an uncontrolled failure such as the babbling idiot problem where a node on a network continues to send data possibly causing further failures or overloading the system [39]. However, a fail silent node can be hard to detect, especially if the system is event triggered and not otherwise sending data [97]. The *fail stop* paradigm [98] acts like fail silent except that the failure is detectable. A possible implementation is given by Schlichting et al. [98] based on a set k of processors acting as a single fail stop processor which can detect up to k failures.

2.2.6.2 Fault Tolerance Techniques

Transient faults where, for example, a task produces an incorrect output, can be overcome by rerunning the task. This relies on there being sufficient slack in the schedule to be able to do this without causing other tasks to miss their deadlines [99]. This is a form of time redundancy.

N-Modular Redundancy, for example Triple Modular Redundancy (TMR), replicates subcomponents and then uses voting mechanisms to detect and correct errors [95]. This is a form of *static redundancy* which masks subcomponent errors [95, 100] such that the component as a whole still produces valid output. Static redundancy requires a large amount of resources [99] since all components run continuously. In contrast, *dynamic redundancy* allows an error to occur and then takes steps to recover from it, assuming it can be detected.

The MARS (MAintainable Real-Time Systems) architecture [97] proposes redundant shadow components as a compromise between resource usage and fault tolerance. A shadow component receives the same data as the component it is shadowing but does not send any data [96] so the communications bus is not replicated. If the main component fails, then the shadow component is in a suitable state to take over from it. Because the shadow component is not sending data, it is hard to detect if it itself has failed. For this reason, the shadow components occasionally send a heart beat message.

For safety-critical systems which must run for a long period of time with high availability, there comes a point at which errors can no longer be masked or recovered from [95]. At this point, it is necessary for the system to *gracefully degrade*. A gracefully degrading system is one which is still able to operate after some components have failed, albeit with reduced functionality or performance [95, 101]. This is closely linked with the previously introduced idea of reconfigurable systems. Architectures such as IMA which use standardised hardware are particularly well suited to reconfiguration [94]. Allowing some degradation of non safety-critical services combined with reconfiguration allows hardware to be used much more efficiently compared with traditional static redundancy [94].

Redundancy not only enlarges the task allocation problem by introducing new tasks and messages but also introduces the additional constraint of not allocating replica tasks to the same processor. Whilst it is possible to find an allocation for the original set of tasks and then replicate the entire system, this may not be the most efficient use of hardware. Graceful degradation introduces the need for modes containing subsets of tasks which provide critical functionality. Allowing task migration during these mode changes raises the possibility of moving critical tasks away from failing processors to improve the fault tolerance of the system.

2.3 Approaches To Solving Task Allocation Problems

In this section previous work on task allocation is collated according to the type of algorithm used to solve the problem. Each optimisation method is briefly explained and followed by a review of work on task allocation which has used that technique.

2.3.1 Problem Specific Heuristics

A *heuristic*, deriving from the Greek word ευρισκειν meaning "to find", is a guide to solving a problem. The guide can be a sequence of steps expected to lead to a good solution or a function estimating the distance from the current solution to an optimal one. Reeves and Beasley define a heuristic in the context of an algorithm:

> *A heuristic is a technique which seeks good (i.e. near optimal) solutions at a reasonable computational cost without being able to guarantee either feasibility or optimality, or even in many cases to state how close to optimality a particular feasible solutions is.*
>
> – Reeves and Beasley [102]

Another use of the word "heuristic" is to describe functions which are an inexact guide to the quality of a solution, or its distance from being an optimal solution [103]. A heuristic function can be combined with a search algorithm to guide the search towards good solutions. Search algorithms which can solve a range of problems but require a problem specific heuristic function are called *metaheuristics*. Examples of metaheuristics are hill descent, simulated annealing and genetic algorithms. These are covered in sections 2.3.2 and 2.3.3.

The heuristic function used with a metaheuristic algorithm is called an *objective function*. Objective functions to be maximised are usually referred to as fitness functions whereas those to be minimised are called cost functions [104]. In this work, unless otherwise specified, a minimisation problem is assumed and the cost function terminology is preferred.

2.3.1.1 Examples Of Task Allocation Specific Heuristics

A good heuristic that is well matched to the problem can offer extremely good performance. Researchers working on task allocation in the 1980s had limited access to compute power compared to modern (2009) standards and hence spent time analysing the problem and developing problem specific heuristics.

Many authors [105, 106, 51, 107] have developed heuristics for soft real-time allocation problems. The aims being to minimise response time [106, 51, 107] or balance load [105] rather than find a solution which ensures deadlines are met. The problem formulation is that of an optimisation rather than constraint satisfaction problem as given in the definition in section 2.2.4. Valuable insight can still be gained from the design of the heuristics.

Chu and Lan [105] present an algorithm separated into two stages: clustering of tasks and assignment of clusters. This is a strategy to reduce the size of the assignment space [106, 3, 107]. The clustering algorithm tries to group pairs of tasks which transfer a large amount of data between themselves to reduce communication overheads and balance this utilising the available processors efficiently.

Lo [106] uses heuristics based on graph theory to reduce the overall processing of tasks and messages that is required. Other than the heuristic, Lo's work differentiates itself from Chu and Lan by allowing tasks to have processor dependent execution times, i.e. a heterogeneous hardware platform. The method, published in 1988, was shown to be effective on task sets of 4 to 35 tasks on a VAX 11/780.

A more recent piece of work with a similar model to Chu and Lan was published by Peng et al. in 1997 [51]. The aim is to minimise the maximum task response time. This uses a branch and bound [108] approach with an appropriate heuristic for exploring high quality parts of the solution space first and allowing much of the search tree to be bounded. The exhaustive nature of branch and bound algorithms mean an optimal solution is guaranteed. The algorithm was shown to be feasible on task sets of up to 140 tasks, requiring only a small fraction of the entire possible solution space to be checked in order to find an optimal solution.

One of the first pieces of work on task allocation specifically aimed at safety-critical hard real-time systems was published by Ramamritham in 1990 [109] with an extended version of the work later published in 1995 [3]. Like Chu and Lan, the algorithm takes a two stage approach of first clustering tasks and then assigning clusters to processors. In the assignment stage, the algorithm also constructs a static cyclic schedule with deadlines assumed equal to task periods. The model includes varying numbers of replicas for the different tasks and a valid solution must also separate replicas onto different processors. The clustering stage of the algorithm is based on a heuristic which clusters tasks depending on whether the ratio of task computation times to the communication time of messages passing between the tasks exceeds a threshold value. The best value of this threshold is highly problem depen-

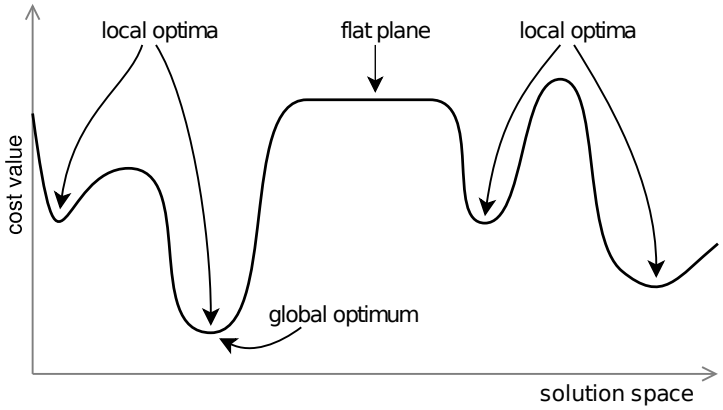


Figure 2.4: One dimensional cost function landscape

dent and so the algorithm tries several values. This heuristic relies on computation and communication times being known prior to allocation. This makes it unsuitable for a heterogeneous hardware platform where times are processor dependent. The algorithm was shown to be efficient on systems with 24 tasks taking only 12s for the main assignment and scheduling portion of the algorithm on an Intel 386 processor.

Work also falling into the category of specific heuristic solutions to task allocation problems are those employed for EDF schedulability tests [82]. These are generally efficient deterministic heuristics which benefit from the simple EDF schedulability test given in section 2.2.2.3. These algorithms take no account of precedence constraints. As previously stated in section 2.2.2.3 static scheduling policies are still dominant in safety-critical systems and allocation algorithms related to partitioned EDF policies are considered outside the scope of this thesis.

2.3.2 Local Search Algorithms

Local search algorithms are a class of metaheuristic search algorithms which explore the solution space by moving between solutions which are considered neighbours of each other [103]. The set of neighbours of a solution is its *neighbourhood*. In addition to a cost function, the definition of a solution's neighbourhood is a way of providing problem specific information to a local search metaheuristic since it defines the set of possible next steps.

If a neighbour is evaluated and is in some sense considered a poor move for the algorithm, it is rejected and a different neighbour is tried. The criteria for accepting or rejecting a neighbour is the key differentiator between different types of local search algorithm.

Figure 2.4 shows an imaginary landscape generated by the cost function of a one dimensional optimisation problem. Features which are considered important to local search algorithms are labelled. The aim of the search is find a local optimum of sufficient quality which is not necessarily the global optimum. Flat areas of a landscape can cause problems for a search since it provides no information as to

whether the search is moving towards a high quality solution. A landscape may also be deceptive if there is a steep downward gradient leading to a poor local optimum.

The problem trying to be overcome by all search algorithms is achieving the correct balance between exploration; covering a sufficiently large proportion of the solution space; and exploitation; intensively searching a small, but promising area of the search space [110]. If an algorithm is over explorative then it is likely to bypass good solutions. If an algorithm is too exploitative, then it will not move far from its starting point before becoming trapped in a local optimum.

2.3.2.1 Hill Descent Algorithms

A first-choice hill descent algorithm selects a solution at random from the neighbourhood and only accepts it if it is better than the current one [103]. A steepest descent hill descender will evaluate every possible neighbour and select the best one. This is only possible if the neighbourhood can be enumerated and isn't too large. Steepest hill descent is a very exploitative search. Variations are possible where a smaller sample is taken.

To stop hill descent algorithms from becoming trapped in local optima, the search can restart from a randomly generated solution. Criteria need to be set for doing this. A sensible strategy is to restart after there has been no improvement in cost value for a certain number of moves.

2.3.2.2 Simulated Annealing

Simulated annealing [111, 102] is a popular local search method inspired by the physical annealing process. Like hill descent, any chosen neighbours which are better than the current solution are always accepted. In other cases, a random value $r \sim U(0,1)$ is generated and a poorer solution is accepted if

$$r < e^{\frac{-\Delta C}{t}} \tag{2.19}$$

where t is a temperature parameter and ΔC is the difference between the cost of the neighbour and the last accepted solution. As the search proceeds, the temperature is decreased, making it less likely for worse solutions to be accepted. Therefore the search is more explorative at the beginning. There are several possible cooling strategies [112]. One of the most common methods is to decrease the temperature by a factor α so after a particular number of cooling steps c, $t = t_0\alpha^c$. α is known as the cooling rate and t_0 is the initial temperature. The choice of values for α and t_0 change the exploration and exploitation characteristics of the algorithm. Ingber [113] notes that slower cooling schedules give statistical guarantees of convergence to a global optimum but are often not used in real applications because convergence can be extremely slow and impractical.

Modifications to the standard algorithm such as random restarts or reheating are

sometimes used to escape local optima [102]. If the simulated annealing algorithm is implemented where the temperature remains constant ($\alpha = 1$) then it is called the Metropolis algorithm.

2.3.2.3 Tabu Search

Hill descent and simulated annealing are both memoryless. That is, the neighbourhood construction and acceptance criteria do not make use of the information gained from evaluating previous solutions. It is possible to return to the same solution several times. Tabu search [114] keeps a history either of previous solutions or information about those solutions. It uses this information to prevent the search returning to solutions or classes of solutions it has previously visited. The history is normally of limited size so that after some time the search can return to a previously investigated area of the solution space. A search can be prevented from revisiting solutions because it has recently been there or because solutions with a particular characteristic have already been evaluated a certain number of times [114]. The design of the history structure and acceptance criteria will depend on the problem.

Since it is a local search method, Tabu search also requires a neighbourhood to be designed. It is often applied to problems where the entire neighbourhood can be enumerated. The neighbour is then chosen by steepest descent, except that moves which are considered taboo; those excluded by the history; are omitted from the neighbourhood. Tabu search of this form, with a number of extensions, has been extremely successful in solving job shop scheduling problems [115]. Job shop scheduling is not within the scope of this thesis but, being an allocation and scheduling problem, has a number of aspects in common with real-time task allocation.

2.3.2.4 Use Of Local Search For Task Allocation

Tindell et al. [4] apply simulated annealing to a real-time task allocation problem. The system model uses fixed priority scheduling with event triggered messages (sent upon task completion) and time triggered tasks. Priority assignment is decided by DMPO so the problem is reduced to an allocation only problem. The model also includes memory constraints, task replicas which must be separated, and other allocation constraints. The objective function is a weighted sum of functions which penalise each type of missed constraint as well as a function which increases the cost for solutions with more network bus usage. The algorithm was successfully applied to a problem with 42 tasks. It was also demonstrated that a balanced processor utilisation could be achieved by modifying the cost function.

Tindell et al. state, “Simulated annealing is not a heuristic algorithm — it is sufficient to state what makes a good solution not how to get one”. This is also the view given in the original simulated annealing paper by Kirkpatrick et al. [111]. However, there is evidence which disputes this claim for practical applications of

simulated annealing. The design of the cost function and neighbourhood generally require problem specific heuristic information in order to get good performance. Indeed, Tindell et al. themselves include a heuristic for reducing network usage within their cost function. Their aim is to differentiate between feasible solutions in order to produce a higher quality solution but omit to mention that this element of the cost function may in fact force the algorithm to find feasible solutions more efficiently in the first place. A similar point is made by Dowsland [102] commenting on cost function design for graph coloring algorithms. She says that authors have found cost functions, whose minimisation does not necessarily lead to a global optimum, but give guidance to the search greatly enhance performance for practical applications.

Cheng and Agrawala [53] apply simulated annealing to a task allocation problem with static cyclic scheduling. Their model allows jitter limitations to be specified for tasks and for the task graph to contain circular dependencies. The simulated annealing algorithm uses different neighbourhoods at different stages of the search depending on properties of the solution. For example, if the processor utilisation is very unbalanced, the neighbourhood is constructed to only contain moves which improve load balancing. Therefore heuristic information is being encoded into the neighbourhood structure rather than the cost function. The scheduling order of tasks on a processor is a deterministic algorithm based on the latest possible start time of a task calculated from its relative deadline and jitter properties. The simulated annealing algorithm is therefore solving an allocation only problem. The method was applied to an avionics platform with 155 tasks and 951 messages. The total task utilisation was 5.14. A valid allocation was found for a hardware platform with 6 processors, the minimum number possible, connected with a single network bus.

Attiya and Hamam [54] use simulated annealing to solve a non real-time task allocation problem with a reliability objective. The cost function measure reliability by combining the time tasks are executing on a particular processor with an assumed known failure rate of that processor.

2.3.3 Genetic Algorithms

Local search algorithms trace a path through the solution space by making small modifications to a single solution. In comparison, population based solutions move from one set of solutions to another. A genetic algorithm is one of the most popular forms of population based metaheuristics. Inspired by the theory of evolution, Holland [116] developed an algorithm which would improve the fitness of a set of solutions by mirroring the natural processes of mating, selection (survival of the fittest) and mutation.

A set of solutions, known as *individuals*, are created, usually at random. This set is the initial population. The following steps are then repeatedly applied to the population [102]:

1. select fittest individuals according to a fitness function and selection mechanism
2. combine fittest individuals with other individuals to create a new generation using a crossover mechanism
3. mutate random selection of new generation
4. replace some or all of existing population with new generation depending on number of children produced by step 2.

The above steps illustrate that genetic algorithms require *genetic operators* to be defined for selection, crossover, mutation and replacement. These all provide ways of using problem specific information within the metaheuristic. It also means that there are several feasible implementations of a genetic algorithm for a particular problem and knowing which one to choose can be difficult.

Crossover functions use part of the solution from both parents in order to create one or more children. It is possible that the resulting children will not be valid solutions in the sense that they fall outside of a constrained solution space which the fitness function can evaluate. There are different ways of handling this [117]. The fitness function can be adapted to recognise every possible child which can result from crossover and apply an appropriate penalty for an invalid solution. The alternative is to enforce crossover to produce only valid solutions. Whilst it may seem preferable to avoid evaluating several invalid solutions, the former is often preferred as the crossover function is usually much less computationally intensive.

Selection strategies are usually probabilistic with probabilities related to fitness values. Fonseca and Fleming [118] point out that the value of the fitness function for an individual does not equate to its fitness. Instead, its fitness is the ability of the individual to survive and reproduce. Blickle and Thiele [119] provide a comprehensive survey of selection strategies.

Mutations are small random changes made to the solution similar to those that would be made to generate neighbours in a local search algorithm.

There are many further variants of genetic algorithms. Elitism ensures that the fittest individuals are not replaced by weaker individuals from the next generation [110]. Genetic algorithms are a subset of evolutionary algorithms. Evolutionary strategies [120] are another type of evolutionary algorithm. Some forms of evolutionary strategy allow children to be produced by a single parent without using crossover relying on selection and mutation alone.

As well as the design of genetic operators, the behaviour of genetic algorithms is affected by several parameters such as population size and mutation rate [102].

2.3.3.1 Use Of Genetic Algorithms For Task Allocation

Bicking et al. [52] use a genetic algorithm to solve a non real-time task allocation problem with an emphasis on dependability. They use a weighted sum cost function including metrics for availability and reliability. Reliability block diagrams are used

to describe functional dependencies so individual component availabilities can be combined into an overall system measure.

Costs for different aspects of the design problem are combined using a hierarchy of weightings to achieve an overall cost. Components for reliability and availability are combined into a cost value for dependability whilst a weighted sum of memory, throughput and communications costs give another cost value. These two values are then combined with another weighting to obtain the final result.

Yin et al. [55] solve a problem with the same objectives as Bicking et al. They use a particle swarm optimisation technique, not detailed here, and a weighted sum objective function.

Axelsson [121] evaluated simulated annealing, tabu search and a genetic algorithm on an architecture synthesis problem. The target is real-time systems with fixed priority scheduling though the scheduling semantics are not described in detail. The algorithm is not only concerned with task allocation but also choice of components. This includes the number and type of processing units with a choice between ASICs (Application Specific Integrated Circuits) and regular general purpose processors. Axelsson's cost function includes timing constraint penalties and also a minimal required processor speedup heuristic to guide the search. This latter cost calculates the minimum increase in processor speed required in order to reduce execution times sufficiently to make the tasks on it schedulable.

Axelsson concludes that the local search methods outperform the genetic algorithm with tabu search being the best overall method. However, it is hard to extrapolate from algorithm comparisons such as these. No only do task allocation problem formulations vary widely but there are many design choices to be made in the implementation of metaheuristic which can drastically change their behaviour.

Nicholson [122] also studied task allocation within a broader architecture synthesis problem for hard real-time systems. He decided upon the use of simulated annealing from the choice of simulated annealing, tabu search and genetic algorithms. However, this was based on reasoned argument rather than any empirical investigation.

2.3.4 Optimisation Via Alternative Formulations

There are a range of off the shelf tools for certain formulations of constraint satisfaction and optimisation problems. The algorithms used by these tools are beyond the scope of this thesis and in some cases may be proprietary. A number of recent task allocation research papers [56, 81, 123] have suggested reformulating task allocation problems so that they can be solved by off the shelf tools.

Metzner et al. [56] formulate task allocation as a satisfiability problem. The scheduling policy is fixed priority with time triggered tasks and messages and asynchronous communications. Zheng et al. [81] use a very similar model but transform

the problem into a mixed integer linear programming (MILP) problem. Their work also includes the packing of signals sent between tasks into messages rather than assuming one message per communication.

Hladik et al. [123] use constraint programming tools to solve task allocation problems. They identify the problem of converting schedulability tests into the appropriate format. They develop two separate algorithms. The first converts the schedulability test into the appropriate format but with increased in pessimism. The second algorithm uses an external schedulability test which feeds information back to the constraint solver which is able to make deductions based on the result.

Bannister and Trivedi [124] formulated task allocation, with task replicas for fault-tolerance, as a constrained linear programming optimisation problem. The scheduling policy is either pre-emptive fixed priority scheduling or EDF dynamic priority scheduling. The mathematical formulation relies upon a simple utilisation based scheduling test for each processor. This is effective for the EDF case since, by equation (2.7), any task set with a total utilisation less than or equal to 1 is schedulable. The limit used for pre-emptive fixed priority scheduling is $\log 2$ since this is the limit of equation (2.1) as the number of tasks tends to infinity. This is obviously pessimistic for small task sets. Since the condition in equation (2.1) is a sufficient but not necessary test, any utilisation based test for pre-emptive fixed priority scheduling will often be pessimistic. The system model does not include task dependencies.

2.3.5 Multi-Objective Optimisation

Local search and genetic algorithms decide quality of solution based on a single value returned from a cost or fitness function. If there are multiple competing objectives, then metrics for those objectives are combined together, usually using a weighted sum approach [125].

A multi-objective algorithm does not combine metrics for each objective into a single value but uses the concept of *pareto-dominance* to decide whether one solution is better than another [126]. If a solution is better in at least one objective and better than or equal to another solution for all other objectives then it is said to dominate that solution. If it is better in some objectives but worse in others then neither solution is dominant. Rather than seek a single best solution, a multi-objective algorithm builds up a set of non-dominated solutions and it is up to the engineer to select a final solution from that set. The *pareto front* is the set of solutions which are not dominated by any others in the solution space. For a multi-objective algorithm, the global optimum solution is the *pareto front* or a subset of it.

The first recognised multi-objective metaheuristic search algorithm was VEGA (VEctor Evaluated Genetic Algorithm) by Schaffer [127]. More recent algorithms are NSGA, NSGA-II, SPEA and SPEA2 [126]. The algorithms differ in how scores

are assigned to solutions based on pareto dominance. Multi-objective algorithms also distinguish themselves on how they ensure that the returned set of solutions is evenly spread across the pareto front rather than emphasising one objective in particular.

The multi-objective algorithms listed above are examples of population based Multi-Objective Evolutionary Algorithms (MOEAs). This is the largest class of multi-objective algorithms. An exception is the Pareto Archive Evolutionary Strategy (PAES) [128] which is an evolutionary strategy which, in its simplest form, uses a population with a single individual, effectively making it a local search strategy.

The weighted sum approach can also be used to generate a non-dominated set of solutions. To achieve this, the algorithm must be run several times with different weightings. A downside to this approach is that it is not able to find all possible points on the pareto front if it is non-convex [125].

2.3.5.1 Use Of Multi-Objective Optimisation For System Design

Dick and Jha [129] use a purposefully designed multi-objective genetic algorithm for embedded systems synthesis. The hardware platform, task allocation and schedule are all chosen by the algorithm. The objectives are price and power usage. Scheduling is done by constructing static cyclic schedules.

Hamann et al. [130] use the SPEA2 algorithm to optimise multiple robustness criteria by varying task priority assignments though not actual task allocations. These robustness metrics use sensitivity analysis to find the limits of task WCETs in the system with the maximisation of the largest feasible WCET of each task being a separate objective. They state other objectives could be minimising CPU clock rates or maximising input data rates.

2.3.6 Discussion Of Techniques For Solving Task Allocation Problems

Making performance comparisons between task allocation solvers is a futile exercise because there is no standard set of benchmark problems and no standard definition of the system model, schedulability test and other problem constraints which define a particular type of task allocation problem.

Of the methods covered, only simulated annealing, problem specific heuristics and the method of reformulating the problem to a standardised form have been applied to hard real-time task allocation problems.

Designing a tailor made heuristic for a problem intuitively seems most likely to achieve best performance but this comes at the price of a reduction in flexibility. The more problem specific information which is used (e.g. Ramamritham assumes a homogeneous platform to design a heuristic), the better performance is likely to be but the smaller the class of problem to which the algorithm can be applied.

A similar argument can be made regarding algorithms which reformulate the problem into a standard form such as a satisfiability or MILP problem. The examples

of work in this area [56, 81] assume a system model and schedulability test which is possible to put in the correct form. Changing the schedulability test requires significant work and it is not clear whether it is even feasible for a more complex platform that, for example, uses a hybrid scheduling policy.

Simulated annealing has been applied with some success to hard real-time task allocation [4, 53, 122]. It is also used as the basis of comparison for other algorithms [56, 123] as it is simple to implement. In common with other metaheuristic searches, it is easy to include any type of schedulability test as it can be directly implemented with no reformulation. There is plenty of scope to include problem specific information in a simulated annealing algorithm via cost function penalties and the neighbourhood structure. The problem specific heuristic approaches provide suggestions for heuristics such as load balancing and grouping of dependent tasks. If these heuristics are included in the cost function they need to be weighted correctly relative to the main constraint satisfaction measure.

Genetic algorithms are harder to implement and configure, mainly due to the large number of possible choices in the design of genetic operators. Bicking's work with genetic algorithms is notable for including dependability as an objective and using a hierarchical objective function which helps manage the problem of weighting configuration.

There is a reasonable amount of work relating task allocation with dependability and related qualities [52, 55]. Other work is less explicit but recognises that task replicas may exist within the system and need to be allocated to separate processors [4, 3]. Work in this regard is related to the property of adaptability since it is giving the system the ability to adapt to faults and continue to function.

No work has been found which explicitly considers design time flexibility as a key objective of task allocation. The research which comes closest is that by Hamann et al. [130] who use sensitivity analysis as a robustness metric but only allow task priorities to vary.

2.4 Software Architectures

2.4.1 Why Study Architectures?

Software engineering is the development of software to meet cost and quality constraints. In a piece of work which was fundamental in establishing software architectures as an artefact of software engineering, Perry and Wolf give the following benefits of software architectures.

> *"Some of the benefits we expect to gain from the emergence of software architecture as a major discipline are:*
>
> 1. *architecture as the framework for satisfying requirements*

> *2. architecture as the technical basis for design and the managerial basis for cost estimation and process management*
> *3. architecture as an effective basis for reuse*
> *4. architecture as the basis for dependency and consistency analysis"*
>
> – Perry and Wolf [131]

Understanding the flexibility of software necessitates a good understanding of the dependencies involved in the system and how changes to certain modules will ripple through the system [132]. Ideally as many modules as possible should be reused with no changes. When new features are required, the changes should be minimal and cost efficient. It is a widely held belief [133, 134, 25] that the software architecture is a suitable level of design at which to measure quality. Clements states

> *"architecture can either allow or preclude the achievement of most of a system's targeted quality attributes"*
>
> – Paul Clements [135]

In an idealised top-down software development model [13], the architecture is the first step in translating requirements into an implementation [135, 136]. In the context of change, a modification to an architectural level component will potentially affect many lower level sub-components. Choices made at an architectural level are far reaching and poor decisions can be costly [137].

2.4.2 Architecture Definitions

There is no agreed single definition for a software architecture. What constitutes the architecture may vary from system to system or change to coincide with a particular development model. That is, the stage of development which is considered architecture and the stage which is considered lower level design is not always clearly separated.

The earliest, widely accepted definition of architectural level elements was given by Perry and Wolf [131]. They state "we use the term 'architecture', in contrast to 'design', to evoke notions of codification, of abstraction, of standards, of formal training (of software architects), and of style". Their definition of an architecture is

$$\text{Software Architecture} = \{\text{Elements}, \text{Form}, \text{Rationale}\}$$

Elements are said to be one of processing elements, data elements or connecting elements. The form of an architecture is the way in which the components fit together. For example, the topology of a network. Perry and Wolf also define form as including properties to constrain the choice of elements such as whether an architecture is permitted to contain layers, pipes, etc. Finally, the rationale describes the reasoning behind choices made. The rationale behind design choices may link to requirements

for certain non-functional properties which will constrain the choice of architectural elements and form.

Newer definitions have given emphasis to different subtleties of software architecture but none have deviated far from Perry and Wolf. Bass et al. [136] define an architecture as follows.

> *"The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them."*
>
> – Bass, Clements, Kazman [136]

This definition places more emphasis on element interfaces. The interface of an element separates what is internal, and must be considered at a later design stage, to what is visible to other architectural elements.

Soni, Nord and Hofmeister clearly and concisely sum up an architecture in the following way.

> *"Software architectures describe how a system is decomposed into components, how these components are interconnected, and how they communicate and interact with each other."*
>
> – Soni, Nord, Hofmeister [134]

This definition makes no mention of "high level" design and can be applied to several levels of system design as long as a suitable abstraction can be made to define the components. However, it is important not to omit the rationale behind the choices made so that the architecture remains a means by which to communicate information between stakeholders.

2.4.3 Architecture Modeling And Analysis

To quantitatively assess an architecture prior to system development, it is necessary to create a suitable model of the architecture. A significant proportion of work on architecture analysis has a background in real-time systems. One of the reasons for this, as noted in Barbacci et al.'s taxonomy of quality attributes [33] is that schedulability tests provide a good basis for quantitative analysis.

2.4.3.1 Model Views

Kruchten [138] considers four possible views from which to model and analyse an architecture: logical, process, physical and development.

The logical view is intended to bring out functional requirements. It has to consider what the data items are and how they are acted upon. It is often the first

view to be considered in meeting a set of requirements as it answers, "What does the system need to do and what does it need to do it to?".

The process view considers how the functionality should be divided into concurrent tasks and which tasks need to communicate with each other. The view helps to answer, "How will the system interact with its external environment and its users?". For example, it may help to see the effects of multiple users simultaneously trying to access a shared resource. This view will be important in assessing timing properties and concurrency issues.

For a full understanding of the process view, it must be considered alongside the physical view. This view describes the underlying hardware architecture. Tasks which are shown as needing to communicate in the process view may be running on the same processor or on two separate nodes of a network. This will obviously have an impact on communication time. While the process view can describe how tasks interact, the actual system behaviour will be dependent upon the physical view and how the process view is mapped onto it.

Finally, the development view answers, "How will the system be built?". It shows how the architecture is divided up into programming modules, what interfaces are required and dependencies such as library and compilation dependencies.

Kruchten ties these four views together with the use of scenarios. If the impact of a scenario can be assessed for each view of the architecture model, it should give a firm idea of the impact of the scenario on the system.

2.4.3.2 Quality Attributes

Before any notion of software architecture was formalised, Boehm et al. [24] discussed quality attributes in terms of program code. This work analyses some of the qualities in terms how easily they can be measured autonomously from the code. In particular, it is pointed out that metrics which take hours of manual inspection are unlikely to be used. It is also argued that it is impossible to combine the quality of a system into a single value since it is likely that systems with similar quality values would have vastly different properties [24].

Nearly two decades after Boehm et al.'s work, Barbacci et al. [33] give a taxonomy of quality attributes with more emphasis at the architectural level. They have the same aspirations as Boehm with regards to quality; "The ultimate goal is the ability to quantitatively evaluate and trade-off multiple software quality attributes to arrive at a better overall system".

Despite Boehm's warning that quality cannot be expressed as a single value, Daniels et al. [139] look at a number of systematic and methodical ways of doing this. There are three aspects to consider in constructing an overall quality metric. Stakeholders find it difficult to consider more than seven quality metrics at one time [139]. Therefore, the first aspect to creating a quality value is to break down qualities into a hierarchy [140]. This allows higher level decisions, such as choosing

whether performance or dependability is most important, to be made before deciding how to trade-off individual performance characteristics.

The second aspect to creating a quality value is to decide on the shape of the quality metric functions. For example, does the perceived quality benefit increase linearly with the metric.

The final aspect is the method of combining individual scoring functions. Weighted linear combination is commonly found in literature but a number of alternatives do exist [139].

2.4.3.3 Scenario Based Analysis

A *scenario* describes a possible change that might occur to the system in the future. There are a variety of analysis techniques based on applying scenarios to architecture models and evaluating the effects of those scenarios with quality metrics provided by the model.

In 1994, Kazman et al. [25] introduced the Scenario Based Architecture Analysis Method (SAAM) where the method is described mainly by way of example. This paper presents some classes for modifiability which are relevant to understanding flexibility.

> - *"Extension of capabilities*
> - *Deletion of unwanted capabilities*
> - *Adaption to new operating environment*
> - *Restructuring"*
>
> – Kazman et al. [25]

A later paper by Kazman et al. [141] adds more detail to SAAM. A scenario is described as a "brief narrative of expected or anticipated use of a system from both development and end-user viewpoints.". By applying scenarios to an architecture model, it is possible to count how many components each scenario affects and how many scenarios apply to each component. This can be aligned to coupling and cohesion. If a scenario affects many components, then there is high coupling between components. If many scenarios affect a component then there is low cohesion.

SAAM is principally a method for assessing modifiability. It is stated that asking, "How modifiable is this architecture?" is of little use but instead the question that should be asked is "How will this architecture accommodate a change of the following class?". Work by Lassing et al. [132] shows that even this question is too broad. A system may react differently to two changes from the same class depending on the complexity of the components involved. Therefore, scenarios which are too broad are of less use, while precise future changes are harder to predict:

> *"Our ability to design for change depends on our ability to predict the future. We can do so only approximately and imperfectly."*

– David Parnas [1]

The benefits of SAAM are said to be largely social [141]. It requires an architect to be able to apply the scenarios to the model and understand their effect. This style of analysis is in conflict with Boehm et al.'s point (from section 2.4.3.2) that quality metrics which require lengthy manual inspection are not likely to be useful. It also leaves the question of the relevance of methods like SAAM to automated architecture improvement. There is nothing explicit in the method, however, that states architecture analysis should be manual. In particular, the relationships between scenarios, the components they effect and the system's modifiability is valid outside of the SAAM process.

2.4.3.4 Tradeoff Techniques

The quality of an architecture is often predicted based on previous architectures built in a similar style [35]. To assess new styles of architecture, Kazman et al. propose the Attribute-Based Architecture Style (ABAS) [35]. An ABAS has four parts: problem description, stimuli/responses, architectural style and analysis. The ABAS associates a section of the requirements with a part of the architecture and provides a way to analyse that part of the architecture. In a sense it defined an architecture view in terms of the quality attributes it makes accessible.

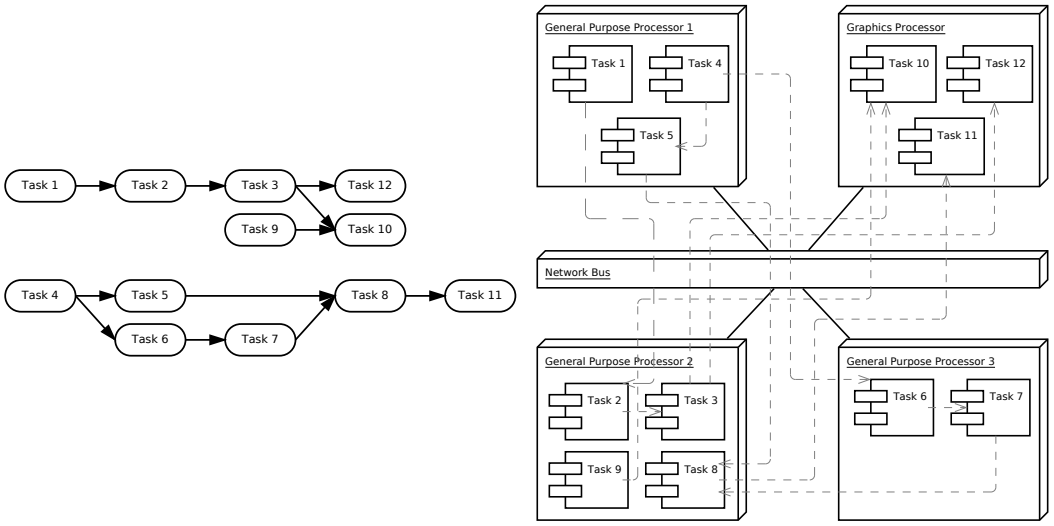
By using ABASs, the Architecture Tradeoff Analysis Method (ATAM) expands on the SAAM to include qualities other than modifiability and make trade-offs between them. It is acknowledged that metrics will not be exact but are expected to identify trends [142] which allow trade-offs to be calculated.

Like the SAAM, mapping a scenario to the architecture requires an architect familiar with the design and is done in an ad-hoc manner. Scenarios are separated into abstract and specific scenarios [143]. An abstract scenario can be applied to many systems. For example, "response times for users must be bounded". An instantiation of that scenario for a particular system would describe the maximum number of users, the required response time and possibly map this onto a response for a particular architectural component.

A survey by Dobrica and Niemelä [144] on architecture assessment methods includes the SAAM and ATAM as well as some extensions to these methods and other alternatives. The conclusion drawn is that the ATAM is the dominant technique and fulfills most criteria for an analysis method. It is also noted that modifiability / flexibility is the primary aim for many methods.

2.4.4 Task Allocation As An Architecture Selection Problem

To put task allocation in a software architecture context, it is best to return to the concept of architectural views previously discussed in section 2.4.3. A software task graph, shown in figure 2.5a, falls clearly into the process view showing interactions



(a) Process view (b) Deployment view

Figure 2.5: Process and deployment architecture views

between separate threads of execution. The deployment or physical view, shown in figure 2.5b, shows both the hardware platform and the mapping of task and messages on to the hardware. The physical view includes information about component types such as specialist graphics hardware. This assumes a static task allocation chosen at design time as opposed to runtime allocation as in the case of global EDF scheduling.

Scenarios are used to tie views together, evaluate architecture qualities and perform trade-offs. Change scenarios for a task allocation problem would be additions of tasks, changes to task attributes such as periods or a hardware platform upgrade.

Scenario based analysis and trade-off methods need set of qualities of interest and metrics to measure them with. The two leading quality metrics used in previous task allocation work are timing, an essential part of the problem, and dependability [52, 55]. Flexibility can be measured directly with sensitivity analysis [130]. The indirect method used in the SAAM of counting how many components are affected by a scenario is also applicable. In the context of task allocation, this could be interpreted as counting processors with an unchanged schedule or tasks whose allocation does not need to change when the scenario is applied.

2.4.5 Automating Architecture Optimisation

This section reviews previous work which has used automated optimisation as a means to selecting an architecture and improving architecture qualities.

Architecture development requires selection of components, connection topologies and component attributes. Automated tools do not develop a complete architecture from scratch but usually concentrate on a smaller selection problem for chosen quality constraints and objectives. With reference to section 2.4.2 the rationale behind an architecture is also part of its definition. The reasoning behind the choices made by

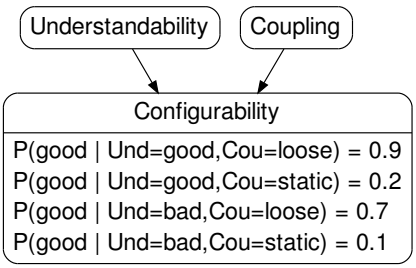


Figure 2.6: Bayesian network section from van Gurp and Bosch [146]

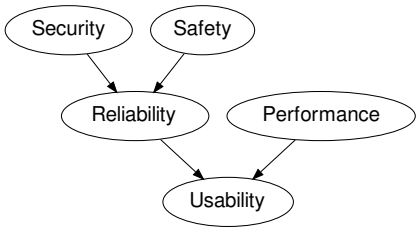


Figure 2.7: Bayesian network of some architectural qualities from Parakhine et al. [147]

a tool should be well understood just as they would be for manual design processes. This is essential for easy maintenance of the system.

2.4.5.1 Bayesian Belief Networks

The subject of Bayesian statistics and decision theory are closely intertwined. The theory behind Bayesian networks is that of *conditional probability*. According to Jensen [145] a conditional probability statement is one which takes the pattern: "Given the event B, the probability of the event A is x". By gathering data from previous experience and simulation, researchers hypothesize that similar statements can be made relating to decisions made about architecture development. For example, "If we choose a network of type T, the probability that fewer than 5% of packets are lost or corrupted is p".

A Bayesian network allows several such statements to be combined into a knowledge base so that probabilities of system quality constraints being met can be calculated. For example the previous conditional probability statement along with several others could be used to calculate a probability for the system having acceptable reliability. Jensen [145] gives further information on how these probabilities can be efficiently calculated. Figure 2.6 shows a section of a hypothetical Bayesian network taken from van Gurp and Bosch [146]. This structure of this network indicates that the quality of configurability is dependent upon the qualities of understandability and component coupling. The probability values show that systems with loosely coupled, well understood components are easier to configure than those which are either poorly understood or have static coupling. If (unconditional) probabilities were assigned to the system for its understandability and type of coupling, then the probability of the configurability being good could be calculated.

Constructing a Bayesian network and assigning conditional probability values to nodes is not something that can be calculated precisely. Both the structure of the network and the probability values will be estimates based on experience and collected data. For this reason van Gurp and Bosch recommend the Bayesian network approach to architecture optimisation as qualitative rather than quantitative. They specifically say that it would be useful in an iterative architecture trade-off method

like ATAM [146]. Scenario based analysis requires a model of the architecture to exist in order to detect effects of scenarios. A Bayesian network model of architecture qualities is complementary to this since it can be built prior to and during the architecture model construction process.

Parakhine et al. [147] also use a Bayesian network as a tool to trade-off architecture qualities. They use simulation as a means to setting probability values within the network. Figure 2.7 is taken from Parakhine et al. [147] and shows the probability of the system being usable to be conditional on reliability and performance. Agent based modelling is used to decide whether a system with different reliability and performance characteristics is considered usable or not.

2.4.5.2 Heuristic Search

Thompson et al. [148] construct a multi-objective real-time architecture design problem. The parameters which are adjustable by the algorithm are the choice of components, number of components and bus topology. The algorithm can choose between smart sensors which are all-in-one modules which can connect straight to the bus and dumb sensors whose output requires processing by a so called smart module. The choice affects system cost, weight (mainly due to wiring required), development risk and maintainability. A MOEA was able to find a number of interesting solutions which could the be further analysed by an engineer.

The work of Axelsson [121] and Nicholson [122], which were both previously discussed in section 2.3 in the context task allocation algorithms, looked more broadly at the topic of architecture synthesis. Both favour single objective local search methods. Axelsson's objective is to minimise financial cost and Nicholson combines both cost and reliability.

2.5 Summary

This chapter has surveyed work on analysis of real-time systems, task allocation and software architecture development. The key findings of this survey which are relevant to the remaining chapters are described below.

Section 2.2 included the sporadic task model and fixed priority and static cyclic scheduling policies. Both of these policies are used in hard real-time systems. The distributed event driven scheduling model produces difficult design problems since changing the scheduling attributes can have far reaching consequences. A task allocation problem centred on this policy is one of the most challenging and is chosen for the majority of work in this thesis. However, it is clear that a task allocation algorithm needs to be flexible to changes in scheduling policy which could be made throughout the lifetime of a system.

Real-time systems adapt to environmental changes and the requests of users with mode changes. Mode changes are also part of a larger set of fault tolerance

mechanisms which use redundancy to enable graceful degradation.

Section 2.3 reviewed existing approaches to solving task allocation problems. Comparing performance is difficult due to differences in problem formulation. Use of problem specific heuristics and certain reformulations of the problem appear to provide good performance but with a reduction in flexibility. Local search methods such as simulated annealing are more flexible and are widely used.

Fault tolerance related metrics are common in work on task allocation. However the aim is always to produce a single fault tolerant design. There does not appear to be any work which produces a series of task allocation solution which allows a system to gracefully degrade as the number of faults increases. Bicking et al. [52] recognise the need to measure system functionality so that the impact of a processor fault on the overall system can be understood. No work has looked at solving task allocation problems for multi-moded systems whether modes are used for fault tolerance for just for increasing functionality.

No task allocation work has included an explicit measure of the flexibility of the design with respect to future changes. An unavoidable problem with attempting to do this is correctly predicting future changes. Section 2.4 explored the way this has been traditionally done in the field of software architecture design with the use of of scenarios and quality attributes.

Tools have been developed to support scenario based architecture analysis. In particular, the work of van Gurp and Bosch [146] and Parakhine et al. [147] uses Bayesian Belief Networks to model the system's qualities and understand the effect of scenarios. An interesting suggestion would be use a Bayesian model as a quality metric within the objective function of a search algorithm. However, the current models are not aimed at producing quantitative quality measures simply classing systems as "good" or "bad". These models are also aimed at a considerably broader problem than real-time task allocation. This thesis remains focussed on the more restricted problem of flexibility in the context of task allocation as opposed to flexible architectures in general. Constructing a tool which could automatically solve the latter problem seems overly ambitious at this stage.

It is hoped that the following areas, though currently unexplored, will be a profitable direction in which to build on existing task allocation work, specifically with the aim of producing systems with good flexibility and adaptability:

- support for mode changes
- support for the use of scenarios to decide between solutions
- support for producing gracefully degrading systems

In order to ensure that the method itself is flexible, it should be able to support different real-time platforms and scheduling policies. Meta-heuristic search is the most applicable optimisation technique for meeting this requirement.

3

Requirements And Hypothesis

It is proposed that existing work on task allocation can be extended to support a range of changeability requirements. Section 1.4.2 stated that good task allocation solutions:

1. re-use parts of allocations and schedules from previous development iterations;
2. are flexible — they have the ability to withstand requirements changes without severe disruption to the design;
3. are adaptable — they require few changes to the allocation and schedule when a system reconfigures itself;
4. aid system robustness in the presence of faults by suitable arrangement of redundant tasks.

This chapter lists the requirements for a task allocation tool to provide these benefits. This motivates a hypothesis on the ability of chosen techniques to meet the requirements. These chosen techniques are motivated by the review of existing work in the previous chapter.

3.1 Requirements

A *problem specification* is defined as the section of the system requirements which contains input data for a task allocation problem. A *configuration* is a mapping of tasks and messages to a hardware platform which is a solution to a particular problem specification. The precise contents of problem specifications and configurations are provided in chapter 4.

3.1.1 Reuse Of Existing Solutions (Req. 1)

Assume a configuration, $\mathcal{C}_1$, satisfies all constraints for a problem specification, $\mathcal{S}_1$, which was refined from a past set of requirements. $\mathcal{C}_1$ is called the *baseline* config-

uration. A new solution is now needed for a problem, $\mathcal{S}_2$ which is defined by an evolution of those requirements. To reuse $\mathcal{C}_1$, the following optimisation problem must be solved.

> Find a new configuration $\mathcal{C}_2$
> which minimises the difference between $\mathcal{C}_2$ and $\mathcal{C}_1$
> such that $\mathcal{C}_2$ meets all constraints defined by problem $\mathcal{S}_2$.

A metric to measure the difference between two configurations must be devised. This must take into account changes in allocations of tasks to processors and also the schedule within a processor. If allocation changes are necessary, then some may have a smaller impact others. The difference metric should also include this information.

3.1.1.1 Rationale For Requirement 1

Reuse of a section of a system design has benefits for perfective maintenance. Even though an automated optimisation method can make it easy to generate a new configuration which meets all constraints for a new specification, there are still reasons for this new configuration to be similar to the old one. Firstly, timing issues often emerge during integration. If the previous system worked well, then the fewer tasks whose timing properties change, the less chance there is of unexpected consequences. Secondly, engineers become familiar with and possess knowledge about a particular design. In section 1.1, it was described how understandability is a key part of maintainability. Decisions may have been made either consciously or subconsciously which depend on a particular configuration. The more a design changes, the more knowledge which must be relearnt and the greater the chance of mistakes being made due to unfamiliarity.

The rationale for different allocation changes having a different impact is as follows. Processing units are often grouped together. A good example is a set of LRMs within the same cabinet in an IMA architecture. Moving a task between modules within the same cabinet is not as big a change to the design as moving a task to a different cabinet located elsewhere in the aircraft. This latter case has larger potential for ripple effects to cause undesired functional changes as well as an increase in recertification costs.

3.1.2 Implementation Of Systems With Multiple Configurations (Req. 2)

Some system requirements contain multiple related task allocation problem specifications for which a set of similar configurations must be found. This can be expressed as the following optimisation problem:

> Given a set of specifications, $\{\mathcal{S}_1, \ldots, \mathcal{S}_n\}$,
> find a set of configurations, $\{\mathcal{C}_1, \ldots, \mathcal{C}_n\}$,
> which minimises the differences between each pair of configurations
> such that each $\mathcal{C}_i$ meets the constraints of problem $\mathcal{S}_i$

Some problems may require certain pairings of configurations to be emphasised over others when minimising differences.

3.1.2.1 Rationale For Requirement 2

There are two situations when solving the problem above aids system design. The first is for systems which have product line variations. Each specification contains a common set of tasks providing basic functionality but vary in additional features. A set of similar configurations will be easier to maintain than several disparate ones. Secondly, some systems use more than one mode of operation at run-time to perform different functions and also to adapt to environmental changes. Some systems will dynamically decide how to configure themselves. However, for hard real-time systems where timing and safety guarantees are required the configuration for each mode is chosen during the design process. Using similar configurations for each mode reduces the overheads required to perform a reconfiguration as well as making the system easier to maintain.

The key difference between this requirement and that in section 3.1.1 is that here several problem specifications are presented to the engineer at the same time and a group of configurations must be generated which solve them. In section 3.1.1, problem specifications are given in a sequence and a configuration must be found for each one in turn.

3.1.3 Consideration Of Future Changes (Req. 3)

When a configuration is generated, it should consider future requirements changes. The method should acknowledge that information on how requirements will change will be imperfect and that some parts of the problem specification will be more likely to change than others.

The optimisation problem is the same as that given in section 3.1.2. In this situation, the set of specifications will contain both current and future requirements. The configurations generated for current requirements should then require a minimal amount of changes to satisfy the specifications derived from future requirements.

3.1.3.1 Rationale For Requirement 3

The requirement in section 3.1.1 is about reusing as much of an existing design as possible. This will be made easier if, at the time previous configuration was generated, allowances were made for the problem specification changes now taking place. Although all future requirements cannot be predicted perfectly, some information should be available based on preplanned phased development and requirements assessment methods which give an indication of where to target flexibility [149].

3.1.4 Robustness In The Presence Of Faults (Req. 4)

A hardware fault within a system causes a change in environment for the software running on it. Use of task replicas, reconfiguration and graceful degradation of functionality are all ways in which a system can tolerate a fault without catastrophic failure. These methods raise further optimisation problems related to task allocation.

- Minimise the effect of a fault by varying the number of replicas of each task, their allocation and schedule.
- Select subsets of tasks which maximise the level of service provided by a system when insufficient resources are available to support all tasks.

These optimisation problems require the functionality of the system to be measured and this must be balanced with other non-functional constraints.

3.1.4.1 Rationale For Requirement 4

In previous requirements, it was assumed that the task sets in each problem specification were completely predefined and static. Relaxing this assumption significantly increases the difficulty of the related optimisation problems but also allows more of the development process to be automated. This requirement does not address the general problem of assigning functionality to tasks but does allow the number of replicas of each task to be varied in order to improve robustness of the system. The first optimisation problem considers this issue in the context of a single fault. The second optimisation generalises the problem to selecting a subset of tasks, including replicas, for a given level of available resources. This is asking the design tool to generate alternative problem specifications for changing environments as well as finding configurations which solve them.

Since the objective is to keep levels of functionality high, the functionality of a particular configuration must be measured. There can be tension between the level of functionality provided and non-functional requirements. When the number of processors is reduced due to faults, it may be necessary to remove non-critical tasks from the system so that remaining tasks can continue to meet their deadlines. The aim is to remove a set of tasks so that the minimum amount of functionality is lost and deadlines are still met.

3.1.5 Task Allocation With Multiple Objectives

It is eminently possible that all requirements set out above will be relevant to a single system. This would be the case in following scenario. A system has multiple modes of operation. This makes the requirement in section 3.1.2 relevant. Following an enhancement request, as much of the existing design should be used as possible. The requirement in section 3.1.1 is now appropriate with the key difference that the

minimisation problem now involves sets of problem specifications and configurations instead of individual ones. At the same time as using past design choices, the new design should be flexible in case of more enhancements and therefore the requirement of section 3.1.3 is applicable. Finally, since the target for these requirements are hard real-time systems, it is likely that there will be fault tolerance requirements as in section 3.1.4. When selecting the number of task replicas to use, and selecting subsets to use as the system degenerates, it may be necessary to consider all modes of operation, previous designs and possible future changes.

Combining these optimisation problems increases the size of the solution space exponentially and introduces multiple objectives which may be in tension with each other. This makes solving such a problem a formidable challenge. It is left as a secondary requirement which will be revisited after each of the previous requirements has been addressed individually.

3.1.6 Guiding Principles

This section contains some non-functional requirements which will guide choices made in the design of the optimisation tool.

3.1.6.1 Performance

Performance in terms of speed and memory usage should be sufficient for the tool to be of practical use within an industrial software engineering process. The tool must also support research involving a large number of experiments. Improvements in performance over and above these requirements is not a primary concern.

3.1.6.2 Solution Quality

Solution quality must be balanced with performance. Obtaining a globally optimal solution (or solution set for a multi-objective problem) is not necessary. Indeed, the optimality of a solution cannot be known unless the design space has been exhaustively searched or an efficient mathematical optimisation method exists for the problem.

A solution's flexibility will be measured by how many changes to a solution are needed to support a change in the problem specification. This requires a suitable metric to measure the difference between solutions for each axis of variation in the task allocation problem, e.g. task assignment and task priorities.

There are no benchmarks available to compare solution quality with for the specific flexibility and adaptability issues which will be included within the optimisation methods. The quality of solution should be shown to be significantly better, according to statistical tests, than an alternative algorithm which does not consider these issues.

3.1.6.3 Repeatability

If the algorithm used to produce a solution is non-deterministic then there can be variations in the quality of solution produced from one run to another. If quality variance can be reduced by running the algorithm for longer or running it multiple times then this may be done as long as performance requirements are still met.

3.1.6.4 Flexibility

Functional and quality requirements should be met for a wide range of problem specifications with different characteristics. The tool should be evaluated against a range of problems whose characteristics are relevant to current systems and those in the foreseeable future.

3.2 Hypothesis

From the requirements listed above, it is evident that generating task configurations with good flexibility and adaptability involves dealing with multiple problem specifications and the configurations which are solutions to them. This includes generating configurations which are:

- similar to configurations from the past for reuse (Req. 1)
- similar to configurations from the present for product line variations and multi-moded systems for maintainability and performance (Req. 2)
- similar to configurations which solve predicted problem specifications from the future for design flexibility (Req. 3) and run-time fault tolerance (Req. 4)

3.2.1 Hypothesis Statements

The following statements form the hypothesis of this thesis:

1. A local search algorithm, extended in appropriate ways, is an efficient and effective method for producing feasible task configurations which are similar to previous ones and for finding sets of similar configurations for multiple task allocation problem specifications representing alternatives for present and future system designs.
2. A configurable hierarchical cost function allows the method to be easily modified for problems with a wide range of characteristics.
3. Using scenarios which represent predictions of future problem specifications within the search problem can produce more flexible configurations, even when predictions are imperfect.

3.3 Scope

These requirements and hypothesis are set in the context of a hardware architecture model which contains a set of general purpose processors connected with bidirectional networks. It is possible that not all processors will be interconnected. Communication between tasks is by message passing only. The methods chosen for satisfying these requirements should allow processors and networks to be heterogeneous and also not be dependent upon scheduling tasks and messages in any particular way. With such general hardware and software models, experiments can only be conducted on a limited set of examples. The restrictions put on the hardware and software models for experimentation are given in chapter 4. However the algorithm design in chapter 5 purposefully does not make optimisations dependent on these restrictions.

For the work on fault tolerance relating to requirement Req. 4, faults are limited to permanent processor faults which have a known probability of occurring in a chosen time frame. The fault model is expanded on in chapter 8.

4

Architecture Model And Problem Generation

4.1 Introduction

The real-time system architecture design process consists of a sequence of decisions which refine natural language requirements into a precise architecture with sufficient detail to be implemented. Some decisions may need to be re-evaluated as requirements evolve throughout the lifetime of the system. To be in a position where some decisions are taken by automated tools, a prototype architecture model must exist which can be further configured via parameters which are amenable to automated optimisation. Also, there must exist a way to quantitatively analyse the model to assess any particular settings of those parameters.

The prototype architecture model contains sets of components and connections whose attribute values are decided prior to automated optimisation. Figure 4.1 shows

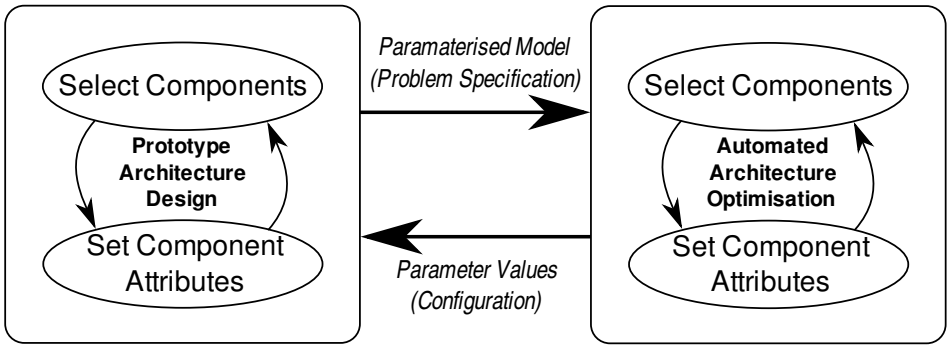


Figure 4.1: Architecture development design iterations

design iteration loops contained within the development of the architecture. The first design loop is the selection of components and their attributes to create the prototype architecture model. This includes design trade-offs such as, for example, selecting the number of processors and processor speeds based on cost, energy consumption and heat dissipation. Some attribute values such as task execution times are a consequence of other design decisions and are measured or calculated rather than chosen explicitly.

The prototype architecture model is then refined by an automated optimisation tool. Depending on the nature of the problem, this can involve adding and removing components as well as altering their configurable attributes. If the automated optimisation tool is not able to achieve an architecture with the qualities required then changes must be made to the prototype architecture outside of the configurable parameter domain of the automated optimisation tool.

Each design optimisation problem has four major elements:

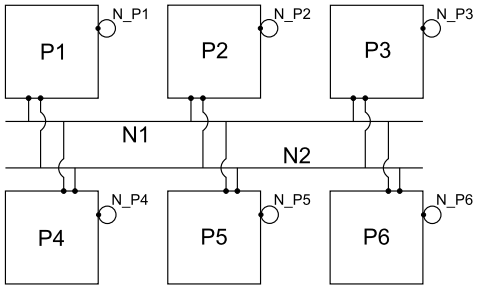
> Given an architecture model (i.e. a *problem specification*) $\mathcal{S}$,
> find a set of values for the configurable parameters (a *configuration*) $\mathcal{C}$
> which minimises a *cost function* $f(\mathcal{C};\mathcal{S})$
> subject to a *constraint function* $\Lambda(\mathcal{C};\mathcal{S})$ indicating that $\mathcal{C}$ is valid.

The task allocation problem fits into this pattern. This chapter describes a form for the problem specification, configuration and constraint function which defines the task allocation problem to be solved in this thesis. The cost function is dealt with in chapter 5.

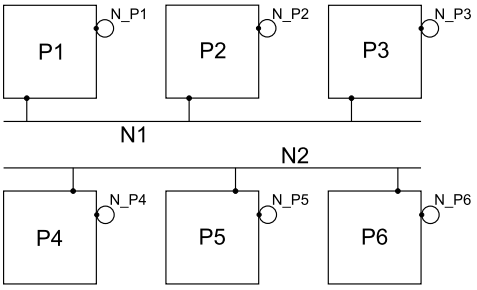
Given the choice of real-time architectures given in section 2.2, section 4.2 describes a set of components and attributes and schedulability tests which define a task allocation problem. Section 4.3 explains possible variations in the configurable parameters of the model which depend on the nature of the task allocation problem. To gain a good understanding of a task allocation algorithm, it must be experimentally evaluated on a range of problem instances with different characteristics. The usefulness of conclusions drawn from experimental results is highly dependent upon how test problems are generated. This is the subject of section 4.4.

4.2 Architecture Model And Problem Specification Definition

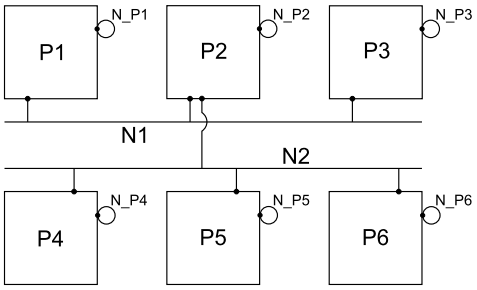
The structure of the architecture model for each task allocation problem is simple yet easily extendible. The intention is for the task allocation algorithm to only be dependent upon a small core model and easily adjustable for specific project or domain features. Design and evaluation of the algorithm with respect to this aim is the main subject of chapter 5. The model components are processor, networks, tasks and messages.



(a) Completely interconnected processors with redundant network



(b) Two processor clusters



(c) Two clusters connected via a single processor

Figure 4.2: Example network topologies

4.2.1 Hardware Components

Networks are connected to processors to form a communications topology. Each network is assumed to allow bidirectional communication between tasks allocated to any of the processors the network is connected to. Three example topologies for 6 processors and 2 networks are shown in figures 4.2a, 4.2b and 4.2c.

Figure 4.2a shows a set of completely interconnected processors. Networks N1 and N2 are both connected to every processor. Network N2 may act as a backup for N1 to improve fault tolerance or, in another case, a high volume of messages may need to be spread between the two networks. Decisions on how to use available resources are part of the task allocation problem.

Figure 4.2b is an example of two separate clusters of processors for different subsystems. In terms of the task allocation problems, these types of topologies present the challenge of ensuring all tasks which need to communicate are allocated within the same cluster.

Figure 4.2c shows two clusters with processor P2 being a member of both. In this situation tasks on processor P2 can send messages to tasks on any other processor. However, tasks on processors P1 and P3 cannot send messages to tasks on P4, P5 and P6 or vice-versa. Automatic relaying of messages via processor P2 is assumed not to be possible. This would add a routing element to the task allocation problem which is not considered in this thesis. For two tasks on separate processors to be able to

Object	Attributes
Processor	Id
Network	Id, Bandwidth, Latency
Interface	Processor Id, Network Id
Task	Id, WCET, Period, Deadline
Message	Id, From Task Id, To Task Id, Maximum Size

Table 4.1: Problem specification objects and attributes

communicate, there must exist at least one network connected to both processors.

Each processor Px is also joined to network labelled N_Px. This is used to model any intra-processor communication overheads rather than ignoring communications sent between tasks allocated to the same processor. Networks are used to represent any communication mechanism whether it be a communication bus, such as CAN, or shared memory message passing. Network bandwidth and latency attributes, introduced in section 2.2.3, affect the time required to transfer a message on a particular network. No broadcast ability is assumed. If a task sends the same message to multiple tasks on different processors, then this is modelled as separate messages.

4.2.2 Software Components

Task dependencies are implied by the passing of messages which creates a graph with tasks at the nodes and messages on the edges. See figure 2.3 for an example.

Each task τ_i has WCET C_i, deadline D_i and period T_i for $i = 1, \ldots, K$. Each message has a maximum size S_i, which is converted into a WCCT C_i for $i = K + 1, \ldots, K + L$. The WCCT C_i of a message assigned to a network σ_j is calculated with equation (2.10) except that a floor rather than ceiling function was used to round to an integer. This allows intra-processor communications to have 0 time if the desired behaviour is to assume they are negligible. Note that these times are based on synthesised message sizes so they cannot be said to be either optimistic or pessimistic. If undertaking a real world case study with non-negligible intra-processor communication, more care is needed on the rounding of values to ensure no optimism is introduced into the worst case.

Throughout this work all tasks are given a BCET of 0 which leads to the highest possible levels of jitter. Offsets for all tasks are also set to 0. The objects and attributes contained within a problem specification are summarised in table 4.1. Interface objects are used to connect networks to processors. An XML format which can describe a problem specification is given in appendix A.

4.2.3 Schedulability Tests

The architecture model also contains a system wide schedulability test attribute to indicate how to check for a valid solution and which scheduling policy to assume. In

chapter 5, experiments will be conducted using both static cyclic and fixed priority scheduling models. This will demonstrate how metaheuristic local search algorithms can solve problems with different scheduling models without a significant reformulation of the problem as previously discussed in section 2.4.5. The core part of this work which is concerned with evaluating and improving the flexibility of solutions will only use the fixed priority model.

The main fixed priority schedulability test used is the WCDO analysis by Palencia and Harbour [59] which was given in section 2.2.3.5. This assumes an event triggered synchronous communications model. Under this model, changing the priority of an object anywhere in the system has the potential to affect objects on other processors. This makes it one of the most interesting and difficult forms of the task allocation problem. A small number of experiments in chapter 5 will use the WCDOPS++ schedulability test by Redell [87] and extended by Kany and Madsen [88]. This will show the trade-off between the more computationally efficient WCDO analysis and less pessimistic WCDOPS++ analysis. Since all communications between tasks, even when on the same processor, is modelled as a message sent on an external network, some of the benefits of WCDOPS++ will be lost since it relies on the concept of H-segments which are directly dependent objects executing on the same processor.

Experiments conducted with a static cyclic schedule use the algorithm of Cheng and Agrawala [53] to convert a task ordering attribute into a slot position. This algorithm assumes each task has a low jitter and high jitter attribute so that the separation between consecutive jobs of a task τ_i is in $[T_i - \text{low jitter}, T_i + \text{high jitter}]$. This constrains the movement of tasks within the schedule to accommodate other tasks. Since jitter requirements are not currently included in the model, both low jitter and high jitter are set to 10% of the task's period.

4.2.4 Omitted Component Attributes

There are component attributes often found in architectural models for task allocation problems which have been omitted from this model. In particular, processors have no attributes such as clock speed or available memory. There are no technical reasons why these cannot be incorporated into the model. The decision to maintain a homogeneous set of processors is linked to decisions on which set of problem characteristics to investigate with a limited amount of compute power and time with which to run experiments. Little is learnt from increasing generality of the architecture model if the effects of changing attribute values is not then fully evaluated.

4.3 System Configuration

The system configuration is a table of object attributes which are left as free variables whose values are to be found by the optimisation method. The layout of this table is shown in table 4.2. It contains configuration information for both tasks and messages.

Object	**Attr**$_1$ **(= Allocation)**	**Attr**$_2$	**Attr**$_3$	...
τ_1	v_{11}	v_{12}	v_{13}	
$\vdots$	$\vdots$	$\vdots$	$\vdots$	
τ_K	v_{K1}	v_{K2}	v_{K3}	
τ_{K+1}	$v_{(K+1)1}$	$v_{(K+1)2}$	$v_{(K+1)3}$	
$\vdots$	$\vdots$	$\vdots$	$\vdots$	
τ_{K+L}	$v_{(K+L)1}$	$v_{(K+L)2}$	$v_{(K+L)3}$	

Table 4.2: Configuration table

v_{ij} is the value of attribute j for object i. For the problem to be classed as a task allocation problem, one attribute must be the allocation of the object. One or more further attributes are used to define the schedule of objects on each processor or network. These will depend on which scheduling policy used. For fixed priority policies, the attribute is the priority number and for static cyclic scheduling, the attribute is an ordering value which is converted into a slot position. In chapter 8 a further attribute is used to allow variations in which tasks are present in the system, looking particularly at task replicas to improve fault tolerance. An XML format for system configurations is given in appendix B.

4.4 Problem Generation

Evaluation of a task allocation algorithm is performed in the context of a set of problem instances in the form of system specifications as described in section 4.2. Data must either be obtained from industrial case studies or synthesised. Both approaches have advantages and disadvantages. A case study gives a true test of the algorithm's performance in an industrial environment. Since it only provides a single data point, however, it does not allow the performance of the algorithm to be extrapolated. In comparison, using a set of randomly generated problem instances can evaluate an algorithm over a range of different characteristics. Even so, the problem space is far larger than can be covered by any set of experiments and so the characteristics of synthetic problems must be chosen so that results are relevant to real world situations.

Since no fully detailed industrial data was made available for the work in this thesis, all evaluation is based on synthesised problems. It is important therefore that the method for generating these problems allows several problem characteristics to be independently controlled so that patterns of behaviour of the task allocation algorithm can be established. These characteristics can then be used to classify problems. The algorithms for generating problems include random elements but are also constrained so that problems have similarities to those found in industry. An example is the algorithm described in section 4.4.6 which samples values for task utilisations at random from a distribution which is chosen based on some real

problem instances.

The problem generator described throughout the remainder of this chapter is called Gentap. The parameters of Gentap define the set of problem characteristics available with which to classify problems. To the author's knowledge, the only similar tool available in the real-time systems domain is Task Graphs For Free (TGFF) [150, 151]. The Gentap tool includes more sophisticated algorithms for sampling execution times and periods and is able to produce a variety of task graphs with exact control over the number of tasks in the system which isn't the case for the task graph generation algorithms used by TGFF. Also, there are currently no tools which generate a variety of hardware platforms in combination with a software model for distributed scheduling and allocation problems.

Some previous task allocation work describes the methods by which they generate problems. Hladik et al. [123] use a fixed hardware platform and only allow linear task graphs. Task utilisations are generated with the UUniFast algorithm [152] even though this is not suitable for multiprocessor systems as discussed in section 4.4.6. It is assumed that they simply discarded invalid task sets.

Ucar et al. [107] create task graphs by randomly adding edges to a matrix of points. This creates a variety of graphs but does not provide the user any control over the shape of the graph for the purpose of linking problem characteristics to results of evaluations.

Attiya and Hamam [54] randomly generate task graphs but do not give details of the algorithm. Execution times are assigned using a simple uniform distribution. However, the systems are not hard real-time and the tasks are not periodic so utilisation is not considered in the problem generation.

Yin et al. [55] ensure their test problems have a wide range of problem characteristics. Both processor/network connections and task graphs are constructed to have different topologies with different degrees of connectivity. However, there is no indication of being able to automatically generate a large variety of problems. Also the problems are not aimed at hard real-time systems and so issues of task periods and total task utilisation are not dealt with.

Baker [153] and Bertogna et al. [154] have evaluated multiprocessor schedulability tests on synthetic problems. They generate task sets of a particular utilisation by continually adding tasks until a utilisation level is reached. This makes comparisons of an algorithm on systems of the same utilisation difficult since they can have different numbers of tasks. This is avoided by the algorithm proposed in section 4.4.6 which changes the distribution utilisations are sampled from instead of changing the number of tasks to achieve different utilisation levels.

4.4.1 Overview Of Problem Generation Algorithm

The architecture model defined in section 4.2 requires the following problem characteristics to be chosen for each specification generated.

- number of tasks, messages, processors and networks
- task WCETs, periods and deadlines
- task dependencies
- message sizes
- network topology
- network bandwidths and latencies

These are chosen by Gentap using one of the following three methods:

1. setting a characteristic directly from an input parameter
2. deriving a characteristic from multiple input parameters
3. random generation of the characteristic influenced by input parameters

A summary of all Gentap parameters is given in table 4.3. Each parameter has a main element with a number of attributes which mirrors an XML format for Gentap input files given in appendix C. Most parameters have min and max attributes which gives Gentap scope to uniformly select from a range of values. Their precise effect varies depending on the parameter. Parameters such as number-of-tasks and tasks-per-processor only require a single value to be sampled per problem generated. Repeated runs of Gentap would produce problems with different numbers of tasks in the specified range. Both min and max can be set to the same value to ensure that all problems generated have the same value for a parameter. Other parameters such as period and network-bandwidth may need to be used more than once within the generation of a single problem. In this case multiple values will be sampled from the parameter range so that, for example, not every task period is the same if min and max for period are different.

The number-of-tasks parameter is used to set a variable, K, which acts as a base size parameter for the problem being generated. Using this value, the following steps are taken:

1. the number of processors is decided by K and the tasks-per-processor parameter
2. the network topology is generated according to the number-of-networks and processor-connectivity
3. tasks are separated into transactions using the tasks-per-transaction parameter and then joined with messages to create task dependency graphs using transaction-length and messages-per-task parameters
4. task periods are randomly generated based on period-range and period-granularity parameters
5. WCETs are decided by sampling utilisation values from a distribution to achieve a total utilisation which meets a utilisation-per-processor parameter value

Parameter	Attributes	Description
number-of-tasks	min, max	Sets number of tasks.
period	min, max, granularity	Selects period values for tasks and messages. Attributes influence and value range and LCM (section 4.4.5).
utilisation-per-processor	min, max	Sets the total task utilisation when averaged over the number of processors.
distribution	type, alpha	Describes task utilisation sampling distribution (section 4.4.6.
utilisation-per-network	min, max	Sets total message utilisation based on number and type of networks. *Not* a straightforward mean average because of network heterogeneity (section 4.4.7)
distribution	type, alpha	Describes message utilisation sampling distribution.
tasks-per-transaction	min, max	Mean tasks per transaction. Each transaction has as close to same number of tasks as possible.
transaction-length	min, max	Changes transaction shape by setting number of tasks in longest path through transaction as a percentage of tasks in transaction (section 4.4.4).
messages-per-task	min, max	Sets number of messages in system as a multiple of number of tasks.
transaction-utilisation-distribution	equal	Boolean value indicating whether task utilisations are set globally or per transaction to even out utilisation per transaction. (section 4.4.6.7).
tasks-per-processor	min, max	Sets number of processors according to mean number of tasks per processor (section 4.4.2
number-of-networks	min, max	Sets number of inter-processor networks.
processor-connectivity	min, max	Affects how many processors each inter-processor network is connected to (section 4.4.3)
network-bandwidth	min, max	Sets bandwidth of inter-processor networks.
network-latency	min, max	Sets latency of inter-processor networks.
processor-network-bandwidth	min, max	Sets bandwidth of intra-processor networks.
processor-network-latency	min, max	Sets latency of intra-processor networks.

Table 4.3: Gentap parameters for controlling problem characteristics

6. message sizes are assigned based on an estimate of transmission rates and a distribution of utilisation values to achieve a total utilisation which meets a utilisation-per-network value

Each of these steps are now described in more detail.

4.4.2 Step 1 — Processors

In keeping with the strategy of having a single parameter, K = number-of-tasks, able to change the size of the system while preserving other characteristics, the number of processors is decided by the parameter tasks-per-processor using the formula $\lceil K/\text{tasks-per-processor} \rceil$. Hardware generation needs to be performed first since the number of processors influences the setting of other properties such as average

utilisation per processor. Processors are assumed to be homogeneous and have no attributes. There is some discussion of attributes not currently generated by Gentap, such as processor speeds, in section 4.4.8. The set of processors is $\mathcal{P}$ and the number of processors calculated by the above formula is $|\mathcal{P}|$.

4.4.3 Step 2 — Network Topology

For a given number of processors and networks, there are a large number of possible network topologies, the choice of which could have a significant effect on the difficulty of the task allocation problem. Using an algorithm with a large degree of randomness may allow several different topologies to be generated but the problems generated for the same input parameters could have very different properties. Instead, Gentap uses a mostly deterministic algorithm controlled by a number-of-networks and processor-connectivity which restricts the set of possible topologies which can be generated. The three topologies in figures 4.2a, 4.2b and 4.2c are all contained within this set.

$\mathcal{N}$ is the set of networks produced and the size of this set, $|\mathcal{N}|$, is chosen by the number-of-networks parameter. The connectivity of a processor is defined as the number of other processors it is connected to per network and has values in the range $[0, |\mathcal{P}| - 1]$. The maximum total processor connectivity achievable per network is $|\mathcal{P}|(|\mathcal{P}| - 1)$. The processor-connectivity parameter is specified as a proportion of this value. For example, a system with 6 processors has a maximum processor connectivity of $6 \cdot 5 = 30$. In figure 4.2a the processor connectivity for both networks is also 30 and so this topology would be generated by specifying a value of 1 for the processor-connectivity parameter. The topology in figure 4.2b would be generated by specifying a value of 0.2 since the processor connectivity of each network is $3 \cdot 2 = 6$. In figure 4.2c, network N1 has a connectivity of 6, N2 has a connectivity of 12 and the mean connectivity is 9. To obtain this topology a processor-connectivity value of $9/30 \approx 0.33$ should be given. Of course, other topologies with the same connectivity values are possible and repeatedly running the Gentap tool with the same parameters can produce different topologies in each problem generated.

The set of possible topologies which can be generated for given parameter values is explained by the following network generation algorithm:

1. take an array of references to the set of networks which is indexed by k
2. take a randomly shuffled array of references to the set of processors which is indexed by l
3. initialise k and l to 0
4. connect the network referenced at k to the processor referenced at l
5. increment k and l
6. if $k = |\mathcal{N}|$ then reset k to 0
7. if $l = |\mathcal{P}|$ then reset l to 0
8. if connectivity requirements are met or exceeded then end

9. else return to step 4

The connectivity requirements also ensure that each network is connected to at least two processors so the minimum connectivity value of the generated topology will be $\frac{2}{|\mathcal{P}|(|\mathcal{P}|-1)}$ regardless of the processor-connectivity parameter. If $|\mathcal{P}| = 1$ then no networks are generated.

The communication time of a message is calculated based on the bandwidth and latency attributes of the network on which it resides. The parameters network-bandwidth and network-latency are used to guide the problem generator in the attribute values it assigns to networks. Each parameter is actually composed of three values: a minimum, maximum and granularity. The attribute value is chosen uniformly from the set of multiples of the granularity between the minimum and maximum values inclusive. The minimum and maximum should be multiples of the granularity and are rounded if not. This style of random number generation is used for several parameters which are selected from a range.

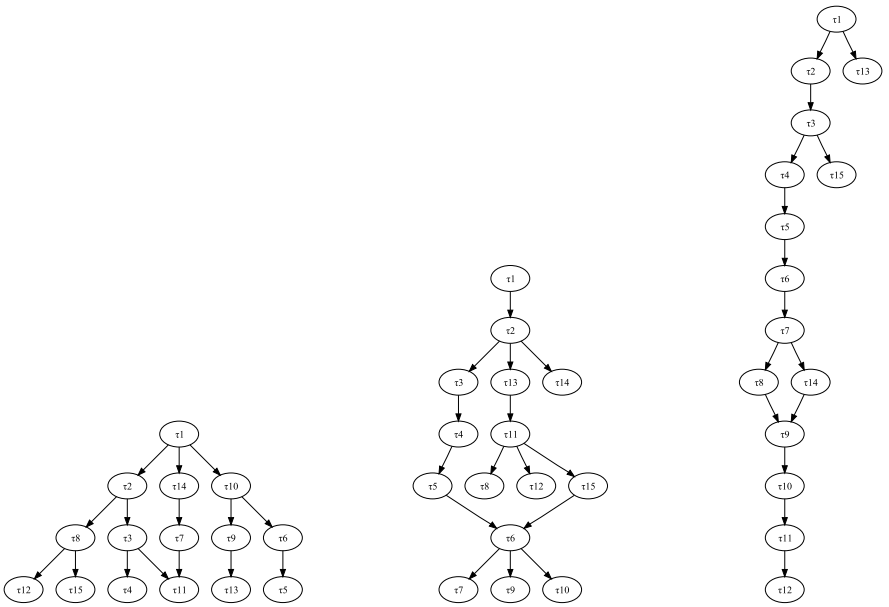
An additional network is added to each processor for intra-processor communication. Separate parameters are used to specify the bandwidth and latency of these networks.

4.4.4 Step 3 — Task Graphs

The most prominent random task graph generator for real-time tasks is the Task Graphs For Free (TGFF) tool [150] by Robert Dick et al. The original TGFF task graph generation algorithm concentrated on generating a graph such that each node met specific in-degree and out-degree criteria. More recently, as described in the April 2008 version of the TGFF manual [151], a new algorithm has been developed based on recursively generating multiple linear chains of tasks called *parallel series*. Each recursive step is able to create a branch with tasks which can run in parallel with the original chain. Additional messages can then connect chains together to create more complex task graphs. Neither TGFF algorithm produces graphs with an exact number of tasks but rather continually adds tasks to meet other criteria until an upper limit on the number of tasks has been reached or surpassed.

The approach used in Gentap is also to generate parallel series of tasks and then insert additional dependencies for systems which have a high ratio of messages to tasks. The most significant difference is that an exact number of tasks per task graph can be specified using the Gentap algorithm, allowing for precise control of this variable when running experiments.

The shape of task graphs is affected by the transaction-length and messages-per-task parameters. Some example task graphs generated by Gentap are shown in figures 4.3 and 4.4. These parameters can be set based on characteristics of the type of system being studied. For example, the AIMS Boeing 777 avionics platform is said to have 951 messages and 155 tasks [53] which suggests that messages-per-task

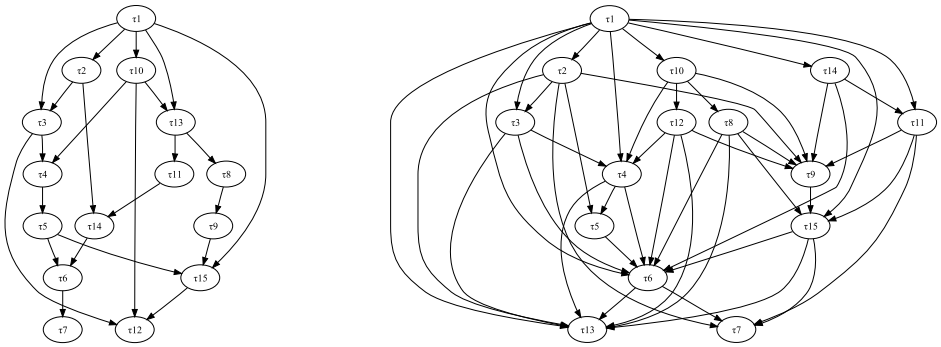


(a) transaction-length = 0.3, messages-per-task = 1

(b) transaction-length = 0.5, messages-per-task = 1

(c) transaction-length = 0.8, messages-per-task = 1

Figure 4.3: Variations in transaction length



(a) transaction-length = 0.5, messages-per-task = 1.5

(b) transaction-length = 0.5, messages-per-task = 3

Figure 4.4: Variations in messages per transaction

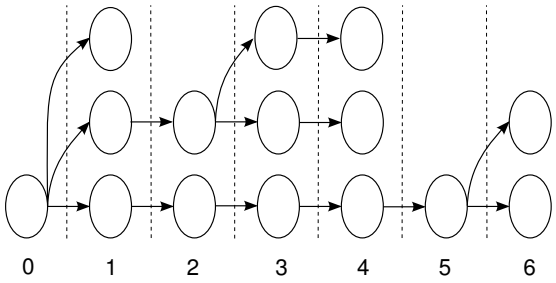


Figure 4.5: Task graph stages

should be set significantly above 1 if trying to randomly generate similar systems.

4.4.4.1 Task Graph Generation Algorithm

Tasks are divided into transactions according to the tasks-per-transaction parameter which, along with the value of K, decides how many transactions need to be generated. Gentap distributes tasks as evenly as possible between transactions. Once tasks have been grouped into transactions, the algorithm for generating each task graph proceeds as follows. The length of a transaction, defined as the longest path from the triggering task to a task with no dependents is given by a transaction-length parameter. The parameter is specified as a proportion of the tasks in the transaction. Values close to 1 will produce very linear transactions with little branching, whereas lower values produce shorter transactions with more tasks able to operate in parallel. The parameter can be specified with a minimum and maximum value to allow different values for different transactions to be sampled randomly from a uniform distribution. With the length of the transaction decided, a number of *stages* are generated equal to the length of the transaction. These are shown by numbered columns in figure 4.5. Following this, these steps are taken:

1. A single task is placed in each stage.
2. The remaining tasks are randomly distributed between stages other than stage 0 which contains the initial triggering task. Tasks are placed as if being dropped into slots so that each task is one row above the previous task to be placed in the same stage.
3. A dependency is created between all horizontally adjacent tasks from the lower numbered stage number to the higher.
4. A dependency is added to all tasks which don't have a predecessor (other than the initial triggering task) from a randomly selected task in the previous stage.

At this point, the task graph has a dependency structure similar to that in figure 4.5 which has just sufficient messages to ensure that every task is connected to at least one other. The algorithm now proceeds according to the messages-per-task parameter. If there are x tasks in a task graph, the current number of messages per task will be $(x-1)/x$. If the messages-per-task parameter has a value greater than

this, further messages must be added.

The next group of messages which are added reconnect any dangling task chains into another chain. This is similar to the behaviour of the TGFF parallel series algorithm. To do this, a list of tasks which aren't in the final stage and have no dependents is created. A task is selected at random from this list and a dependency is added from this task to a randomly chosen task in the following stage. The task is then removed from the list. This is repeated until the correct ratio of messages per task has been reached or the list is empty.

If, at this point, further messages still need to be added, pairs of tasks from different stages are randomly selected and a dependency is added from the task in the lower numbered stage to the other task. If a dependency already exists between the two tasks, another pair is selected. Before these final messages are added, the maximum number of dependencies possible is calculated to ensure that the desired number of messages per task is theoretically possible and the parameter is lowered automatically if necessary.

4.4.5 Step 4 — Task Period Selection

Periods are assigned on a per transaction basis so that all tasks and messages within a transaction will have the same period. Davis et al. showed that the distribution of periods can bias the performance of schedulability tests [155] and, in particular, the importance of the number of orders of magnitude in the sample set. It is quite common for a system to have tasks operating in different time bands [156], e.g. 1 – 10ms, 10 – 100ms, and 100ms – 1s, and hence have sets of tasks with significantly different periods. For example, a task set used in an industrial case study by Bate [157] has a sets of tasks with periods ranging from $25 \cdot 10^3$ to 10^6 (time units not given). Another important characteristic of the sample of period values is their lowest common multiple (LCM) [53] which defines the length of a scheduling cycle. Both of these are taken into account by Gentap when sampling periods.

The sampling is guided by minimum (T_a), maximum (T_b) and granularity (T_g) parameters. Period values are selected from a log uniform distribution and rounded according to the granularity.

$$r_i \sim U(\log T_a, \log(T_b + T_g)) \tag{4.1}$$

$$T_i = \left\lfloor \frac{\exp(r_i)}{T_g} \right\rfloor T_g \tag{4.2}$$

The random values r_i produced lie in $[\log T_a, \log(T_b + T_g))$. T_a and T_b must be multiples of T_g.

The maximum possible LCM of the sampled periods will decrease as the granularity increases since the number of selectable values between T_a and T_b will decrease as T_g increases. A lower bound on the maximum possible LCM can be found from

the case where just two tasks values sampled:

$$\max \mathrm{LCM} \geq \frac{T_b(T_b - T_g)}{T_g} \tag{4.3}$$

The values produced will follow a similar distribution to the method of Davis et al. [155] who suggested sampling from separate bins such as $[10, 100)$, $[100, 1000)$, $[1000, 10000)$ but removes the need to define bins for different ranges. In comparison, the TGFF tool selects periods from a finite list given by the user [151]. These are given as a base period and a set of multipliers. Although TGFF can be set up to produce similar results, Gentap allows a wide variety of realistic task periods to be generated with three just intuitive parameters.

4.4.6 Step 5 — Task WCETs

One of the most important characteristics in real-time system design problems is the total task utilisation per processor. Therefore, Gentap has several parameters to control this value and also the distribution of task utilisations. The utilisation of each task along with its period, assigned using the method from section 4.4.5, can then be used to determine the WCET of each task.

Let s be the target utilisation per processor value. The total target utilisation for all tasks is then

$$u = |\mathcal{P}|s \tag{4.4}$$

A utilisation value, U_i needs to be chosen for each of n tasks such that

$$\sum_{i=1}^{n} U_i = u \tag{4.5}$$

where u is the total target utilisation value. n may equal K, the number of tasks in the system, but the method may also be applied to smaller subsets of tasks as described in section 4.4.6.7. The WCET C_i for task i with period T_i can then be calculated using

$$C_i = U_i T_i \tag{4.6}$$

4.4.6.1 Overview Of UUniFast

The most prominent piece of work in the area of generating WCETs for synthetic task sets is the UUniFast algorithm by Bini and Buttazzo [152]. The utilisations of several sets of n tasks which adhere to the constraint in equation (4.5) can be plotted in an $n - 1$ dimensional space. The UUniFast algorithm produces task sets which are evenly distributed in this space. This is intended to allow schedulability tests to be compared on a set of task sets without undue bias towards any particular test. No claims are made about UUniFast regarding how representative the produced utilisation distributions are of real task sets.

The distribution of each U_i in the sets of utilisations generated by UUniFast is defined by $u - x_i$ where x_i has distribution equivalent to the sum of $n-1$ independent uniform random variables over $[0, 1]$. The probability density function for this distribution restricted to the domain $[0, 1]$ is proportional to x^{n-2} [158]. (It is not true over the whole domain which is between 0 and $n-1$). With appropriate normalisation, the density function for the distribution of each x_i is

$$DensSum(x; n, u) = \frac{n-1}{u^{(n-1)}} x^{n-2} \quad x \in [0, u] \tag{4.7}$$

with the condition $u < 1$. The distribution each U_i generated by UUniFast is now seen to be

$$DensUUF(x; n, u) = \frac{n-1}{u^{(n-1)}} (u - x)^{n-2} \quad x \in [0, u] \tag{4.8}$$

UUniFast is an efficient algorithm for generating utilisations values which conform to this distribution. However, UUniFast was designed for generating uniprocessor task sets. When the total utilisation $u > 1$, as is possible for multiprocessor task sets, UUniFast can generate invalid utilisation values, that is values greater than 1.

An alternative algorithm which would produce equivalently distributed task sets as UUniFast can be constructed which involves sampling values from the distribution in equation (4.8). This is the basis of the utilisation generation algorithm in Gentap.

4.4.6.2 Overview Of Gentap Utilisation Sampling

Gentap allows values to be sampled from a more general distribution, the beta distribution. This is a flexible distribution with many applications [159]. The probability density function for this distribution is 0 outside of the $[0, 1]$ domain. This means that sampling from it will never produce individual task utilisations greater than 1.

The probability density function of a beta distribution is

$$DensBeta(x; \alpha, \beta) = \frac{(1-x)^{\beta-1} x^{\alpha-1}}{B(\alpha, \beta)} \tag{4.9}$$

where B is the beta function [159]. Equation (4.9) includes two parameters, α and β, which change the shape of the distribution.

If $\alpha = 1$, $\beta = n - 1$ and the target utilisation $u = 1$ then this is the same as the distribution generated by the UUniFast algorithm. Tasks utilisations which are distributed in the same way as those generated by UUniFast can be obtained by sampling from this distribution and then scaling all values by u. In that sense, the methods presented here are a generalisation of UUniFast.

The parameters of the beta distribution, α and β, can be selected by fitting a beta distribution to real task set data. This means that task sets sampled from it are purposefully representative of situations where the algorithm will be applied irrespective of biases toward or against any schedulability test. Also, unlike UUniFast,

the distribution enables task sets to be sampled with a total utilisation greater than 1 while the individual utilisation of any task is always less than 1.

Distributions other than the beta distribution could be used. However, most would need to be truncated to ensure individual task utilisations were valid. For example, Baker [153] and Bertogna et al. [154] both use a truncated exponential distribution for sampling utilisation values. Neither give a method of parameterising the distribution and arbitrarily choose a distribution with a mean of 0.25.

The fitted distribution is intended to be an approximation of the underlying real data. If an exact fit to the original data is desired then it makes more sense to base evaluations on the data itself rather than approximately fitted distribution.

The procedure for generating a distribution of task utilisations with Gentap is as follows:

1. Obtain a set of task utilisations from an industrial case study and then use the method of maximum likelihood estimation to fit a beta distribution to this task set (section 4.4.6.3).

2. Generate further task sets by sampling from this distribution (section 4.4.6.4).

Initially the target utilisation per task, u/n, is assumed to be the same as that of the mean of the distribution fitted in step 1. In section 4.4.6.5, the distribution fitting method is changed for arbitrary values of n and u.

4.4.6.3 Fitting a Beta Distribution To A Task Set

The steps above are expanded upon by way of example using data from a case study on a Rolls-Royce BR715 Engine Controller by Bate [157] as the initial representative task set. This task set has 70 tasks with a total utilisation of 0.816. The bar chart in figure 4.6 shows a histogram of task utilisation from this task set. The histogram has been normalised so that its total area is 1 which allows it to be compared visually with other probability density functions.

The parameters α and β for the beta distribution are chosen by the method of maximum likelihood estimation [159] so that the distribution best fits the original task set. Maximum likelihood estimation optimises a log likelihood function which is a metric for how well the data fits the distribution. The log likelihood function for the beta distribution is [159]

$$\begin{aligned} LL(\alpha,\beta;\boldsymbol{V}) = m(\log\Gamma(\alpha+\beta) - \log\Gamma(\alpha) - \log\Gamma(\beta)) \\ + (\alpha-1)\sum_{i=1}^{m}\log V_i + (\beta-1)\sum_{i=1}^{m}\log(1-V_i) \end{aligned} \tag{4.10}$$

where $\boldsymbol{V}$ is the vector of task utilisations to fit the distribution to, m is the number of elements in $\boldsymbol{V}$, and α and β are the parameters to the beta distribution. Finding values, $\hat{\alpha}$ and $\hat{\beta}$ which maximise equation (4.10) can be done using the `mle` function

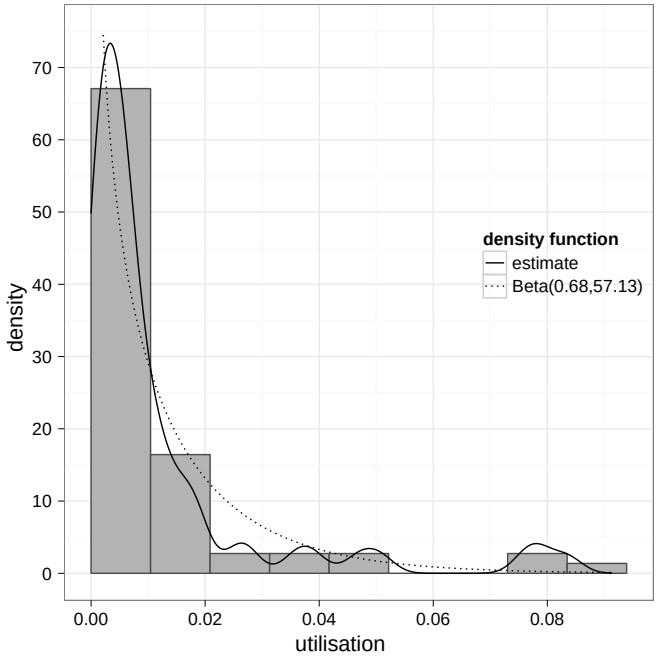


Figure 4.6: Utilisation density of case study task set from Bate [157]

found in the R language [160]. Using this method, values of $\hat{\alpha} = 0.68$ and $\hat{\beta} = 57.13$ were obtained. The probability density function from equation (4.9) is plotted on figure 4.6. Also plotted is a smoothed version of the histogram which is an estimate of the density function generated by the R `density` function.

A similar exercise was undertaken for a task set taken from Tindell and Clark [85]. This task set has 32 tasks and a total utilisation of 1.53 spread over 3 processors. The results of fitting a beta distribution are shown in figure 4.7.

4.4.6.4 Sampling From The Distribution

Once a probability density estimate has been obtained, task set utilisations can be sampled from it. One of the key features of the UUniFast algorithm is that it guarantees that the sum of the task utilisations is equal to a target utilisation value. The approach taken in Gentap is to repeatedly sample the distribution until the obtained task set gives the required utilisation within a certain error bound. This is compared with a sampling method for obtaining an exact total utilisation value. In section 4.4.6.6, it is shown how estimates of the number of iterations for both methods can be obtained and that either is feasible even for large ($\approx$ 1500 tasks) task sets.

The exact method samples $n - 1$ values and then the final task is allocated the correct amount of utilisation to make up the total. If the sum of $n - 1$ utilisation values is either less than $u - 1$ or greater than u then the sampling is repeated until this condition is met.

The approximate method does not guarantee the precise total utilisation u will

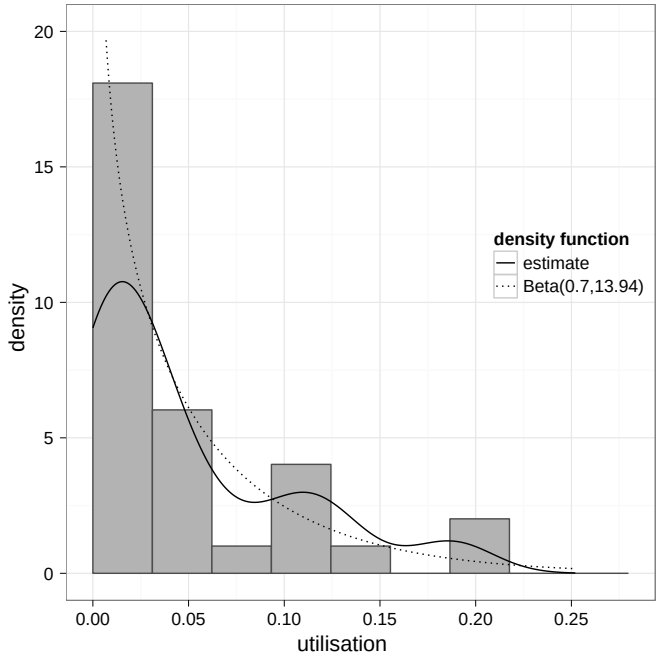


Figure 4.7: Utilisation density of case study task set from Tindell and Clark [85]

be achieved. Instead, it repeatedly samples n values until the difference between the sum of the utilisations in the sample and u is within a specified error bound. Since both methods discard some sampled values, the distribution of all accepted utilisation values across multiple task sets will not be identical to the fitted distribution. An exact match to the original distribution is not required since the aim is to produce similar rather than identical task sets. The size of the discrepancy will depend on the number of discarded values which is studied in section 4.4.6.6.

Figure 4.8 gives listings in the R language [160] for both approaches. The function in figure 4.8a samples n utilisations in each iteration. The `relerr` parameter specifies the relative error allowable for the sum of the sample utilisations in order for the sample to be accepted. For example, a value of 0.01 requires all accepted task sets to have a total utilisation within 1% of u. The function returns both the set of task utilisations and the number of iterations required for the total utilisation to fall within the acceptable limit.

Figure 4.8b gives a listing of a function which can be used to generate a task set so that the exact utilisation target is achieved. It generates a series of task sets containing $n - 1$ tasks until an iteration is reached where the remaining utilisation for the final task is valid.

The parameter values corresponding to the best fitting beta distribution are $\hat{\alpha}$ and $\hat{\beta}$. The population mean of a beta distribution in the general case is [159]

$$\mu_{\alpha,\beta} = \frac{\alpha}{(\alpha + \beta)} \tag{4.11}$$

```
GenTaskUtilsErr <- function(n, alpha, beta, relerr) {
    mu <- alpha / (alpha+beta)
    u <- (n * mu);
    err <- u * relerr
    iter <- 0
    repeat {
        iter <- iter + 1
        utils <- rbeta(n, alpha, beta);
        if (abs(u-sum(utils)) <= err) {
            break;
        }
    }
    list(utils=utils, iter=iter);
}
```

(a) GenTaskUtilsErr function

```
GenTaskUtils <- function(n, alpha, beta) {
    mu <- alpha / (alpha+beta)
    u <- (n * mu);
    iter <- 0
    repeat {
        iter <- iter + 1
        utils <- rbeta(n-1, alpha, beta);
        last <- u - sum(utils)
        if (last > 0 && last < 1) {
            utils <- append(utils, last)
            break;
        }
    }
    list(utils=utils, iter=iter);
}
```

(b) GenTaskUtils function

Figure 4.8: Functions for sampling task utilisations from beta distribution

This is also the expected per task utilisation when values are sampled from the distribution using either algorithm. If it is also the case that this is the desired per task utilisation when sampling for n tasks, then the desired total utilisation is

$$u = n\mu \tag{4.12}$$

where $\mu = \mu_{\hat{\alpha},\hat{\beta}}$.

4.4.6.5 Setting The Target Utilisation

Section 4.4.6.4 assumes that the desired per-task utilisation, u/n is μ, the mean of the fitted distribution. This value should be extremely close to the per task utilisation of the original data. For the Tindell task set it is approximately 0.048 in both cases.

However, to create task sets for experiments with total utilisations set at say, 1.2, 1.8 and 2.4 and each with 40 tasks then the per task utilisations would be $0.030, 0.045$ and 0.060 so the fitted distribution is no longer suitable.

If $u/n \neq \mu$, then combining equations (4.11) and (4.12) shows that values of $\alpha = \hat{\alpha}'$ and $\beta = \hat{\beta}'$ need to be chosen such that $u/n = \alpha/(\alpha + \beta)$ which implies

$$\beta = \frac{\alpha(n-u)}{u} \tag{4.13}$$

By using equation (4.13) to substitute β in equation (4.10), the maximum likelihood optimisation can be repeated with just a single variable, α, to find $\hat{\alpha}'$. $\hat{\beta}'$ can then be set from equation (4.13) in order to specify the distribution. This distribution will be the beta distribution which best fits the data (within the limits of the optimisation method) and also meets the additional constraint of having the desired total utilisation for a system with n tasks.

There may be occasions where values of $\hat{\alpha}$ and $\hat{\beta}$ are known but the underlying data these values were obtained from is unavailable and so it is not possible to re-fit for different u/n. In these circumstances, a reasonable fit can be obtained by setting $\hat{\alpha}' = \hat{\alpha}$ and then obtaining $\hat{\beta}'$ from equation (4.13). For the purposes of using Gentap, if both $\hat{\alpha}$ and $\hat{\beta}$ are given, they are used to parameterise the beta distribution. If only $\hat{\alpha}$ is given, then $\hat{\alpha}'$ and $\hat{\beta}'$ are calculated using this method and these are used instead. When using this method of calculating $\hat{\beta}'$, it should be noted that the shape of the beta distribution changes more significantly when either parameter moves from less than 1 to greater than 1 or vice-versa.

4.4.6.6 Estimating number of iterations

To ensure that the sampling methods of the `GenTaskUtilsErr` function in figure 4.8a and the `GenTaskUtils` function in figure 4.8b are practical, an estimate is needed for the number of iterations required by each to produce a valid task set.

The `GenTaskUtils` method which produces a total utilisation of exactly u will be assessed first. Let t_j be the total task utilisation for the sample of $m = n - 1$ utilisation values on iteration j. For sufficiently large m, the central limit theorem [161] applies and the distribution of t_j is approximated by a normal distribution. Since the estimation relies on this, it will be less accurate when generating task sets with a low number of tasks, say fewer than 20. The population variance for a beta distribution is given by [159] as

$$\sigma^2_{\alpha,\beta} = \frac{\alpha\beta}{(\alpha+\beta)^2(\alpha+\beta+1)} \tag{4.14}$$

Therefore all t_j are normally distributed with mean $m\mu$ and variance $m\sigma^2$ where $\sigma^2 = \sigma^2_{\hat{\alpha},\hat{\beta}}$.

$$t_j \sim N(m\mu, m\sigma^2) \tag{4.15}$$

For the task set to be valid, t_j must be between $u-1$ and u. The probability of this is given by

$$P(t_j \text{ is valid}) = \Phi_{0,1}\left(\frac{u-m\mu}{\sqrt{m}\sigma}\right) - \Phi_{0,1}\left(\frac{(u-1)-m\mu}{\sqrt{m}\sigma}\right) \tag{4.16}$$

where $\Phi_{0,1}$ is the cumulative distribution function for the Normal distribution with mean 0 and variance 1.

For the `GenTaskUtilsErr` function, let u_j be the sum of the n task utilisations sampled on iteration j. Each value of u_j has a corresponding error value, err_j which is the difference between u and u_j. Using a similar argument to that which obtained equation (4.15), these error values are normally distributed with mean 0 and variance $n\sigma^2$.

$$err_j = u - u_j \tag{4.17}$$

$$err_j \sim N(0, n\sigma^2) \tag{4.18}$$

Let $rerr$ be the value of the `relerr` parameter and err the absolute value calculated from it. The probability of the task set being accepted is the probability that the value of err_j is between $-err$ and $+err$.

$$err = u \cdot rerr \tag{4.19}$$

$$P(|err_j| < err) = 2\Phi_{0,1}\left(\frac{err}{\sqrt{n\sigma^2}}\right) - 1 \tag{4.20}$$

For both methods, the expected number of iterations is given by

$$E(iter) = \frac{1}{P(\text{ sample is valid })} \tag{4.21}$$

where the denominator on the right hand side is replaced by the value of either equation (4.16) or equation (4.20) depending on the sampling method being used.

A simple experiment was performed to validate these estimates. For varying numbers of tasks from 10 to 1500, 1000 valid task sets were generated and the mean number of iterations was calculated. This value was compared to the estimate given by equation (4.21). The parameter values were taken from the distribution fitted to the Tindell and Clark case study [85] such that $\hat{\alpha} = 0.699856$, $\hat{\beta} = 13.94309$ and the values of μ and σ^2 were 0.0478 and 0.002909 respectively. Experiments were conducted using both `GenTaskUtils` which achieves the exact value of u and `GenTaskUtilsErr` with relative error values of 1% and 5%. The results are shown in figure 4.9. In each case the solid line gives the estimated number of iterations and the plotted points are the mean number of iterations required for the 1000 task sets generated at each point. In all cases, the estimated number of iterations provides a very good fit, even for smaller task sets where the normal approximation is less accurate.

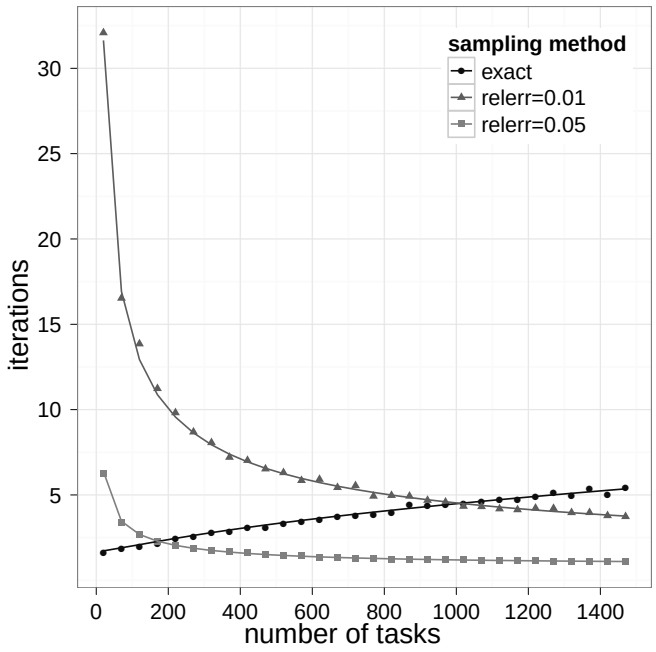


Figure 4.9: Iterations to find suitable task set

The key difference is that, for the method using a relative error, the number of iterations falls as the number of tasks increases whereas it rises for the exact method. In this case, the number of iterations is small so both methods are practical. However, for distributions with a larger variance, more iterations would be needed in both cases. If a large number of iterations are needed to generate large task sets, then the exact method could become less practical. Gentap uses the approximate method with a 1% error.

The graph in figure 4.9 suggests a hybrid method should exist so that performance is as good as the exact method when using the approximate method for small task sets. This is indeed the case. For the exact method, $n-1$ values are sampled and accepted when their sum is within 1 of the desired total so that the final task has a valid utilisation. By allowing the total utilisation to be approximate but bounded by an error, the acceptance criteria could be loosened so that the sum of the $n-1$ utilisations could be up to $1+err$ below the target or err above it. The last value must then be sampled from a truncated beta distribution. This method is not expanded further since the previous algorithms perform adequately.

4.4.6.7 Assigning WCETs To Tasks

There are two ways in which utilisation values can be assigned to tasks. A set of utilisation values can be sampled as described throughout this section with $n = K$, the number of tasks in the system, and then assigned randomly to tasks. Once utilisation values have been assigned, WCETs can be calculated using equation (4.6). Time values, both periods and execution times, are integers and so the WCET must

be rounded. This is a further source of error. Selecting a higher numerical period values (by changing the unit of time) will reduce this rounding error.

The other method of assigning utilisation values is to sample values on a per transaction basis. It is conceivable that the way in which system utilisation is shared out between transactions could have an effect on problem difficulty. Therefore, utilisations can be sampled for each transaction where n is set to the number of tasks in the transaction and the target per-processor utilisation is divided by the number of transactions in the system. Gentap allows a separate sample to be generated for each transaction so that utilisation is divided evenly between transactions.

Using either method, within transactions utilisation values are assigned randomly to tasks. A significant problem characteristic is the correlation between the stage number of the task used in the transaction construction and its utilisation. If using a jitter based distributed scheduling analysis such as WCDO analysis [59] then a transaction with a bias of higher utilisation tasks at lower stage numbers will make the transaction more difficult to schedule since the accumulation of jitter throughout the transaction will be greater. This provides an example of a factor which Gentap does not explicitly control and so must be taken account of using repetitions of experiments.

4.4.7 Step 6 — Sampling Message Sizes

A sample of utilisation values for messages is constructed in exactly the same way as that for tasks which can then be converted into a WCCT. However, converting communication times to message sizes presents an additional difficulty since this conversion will depend on where the message is allocated which is not known at the time of problem generation.

An initial idea was to use mean bandwidth and latency values which would achieve the correct utilisation if message utilisation was allocated evenly between networks. However, systems typically have a small number of inter-processor networks compared to the number of much faster intra-processor networks which is equal to the number of processors. Therefore, systems using mean bandwidth and latency values required most messages to be sent between tasks allocated to the same processor which often isn't possible. Whilst it would be possible to supply Gentap with a very low total message utilisation to make generated problems easier, it is somewhat counterintuitive.

The chosen approach uses double the mean bandwidth and latency of inter-processor networks only. This means that if half of the messages (in terms of utilisation) are evenly distributed between inter-processor networks then the inter-processor network utilisation will be as given in the utilisation-per-network parameter. Messages will very rarely be distributed between networks in this way but choosing a sensible network utilisation value, e.g. 50%, produced problems which could often be solved

which was not the case when a simple mean was used.

Let *ban* be twice the mean inter-processor network bandwidth and *lat* be twice the inter-processor network latency. Using these values, the following equation, where C_i is the WCCT, can be used to calculate the worst case size, S_i, of the message.

$$S_i = \max(0, ban(C_i - lat)) \tag{4.22}$$

The *max* term is required since it is possible that the sampling of utilisations may result in WCCTs which are smaller than *lat* and it must be ensured that all message sizes are non-negative.

4.4.8 Additional Characteristics

There are a number of other problem characteristics and possible additions to the architecture model. The number of parameters present in Gentap already exceeds the number of problem characteristics which can be studied with thorough experimentation, especially when interactions between multiple characteristics are considered. It is, however, worth noting additional characteristics which are likely to impact task allocation algorithms but are not being considered.

The deadline of a task is currently set equal to its period. This is not an uncommon situation but enforcing tighter deadlines will certainly reduce the number of feasible solutions. This value is also used as the end-to-end deadline for a transaction since all tasks in the transaction have the same period and hence the same deadline. Timing requirements are only specified in terms of WCRTs less than deadlines. No restrictions are placed on task or message jitter.

4.5 Summary

This chapter has defined the system model for the task allocation problem instances in this thesis. The model provides a framework for processors, networks, tasks and messages and their attributes. Values for the majority of attributes are contained in a system specification. The remaining ones are used to define the system configuration. One of these configuration attributes is the allocation to a scheduling resource. Others are used to define the schedule on each scheduling resource and must be compatible with the scheduling policy and schedulability test used. Finding a feasible configuration which passes the schedulability test is the basic task allocation problem to which further quality objectives may be added.

For single processor scheduling problems, the characteristics which describe a scheduling problem are well studied. For multiprocessor task allocation problems with different task dependency structures and network topologies, there is a large increase in the number of possible characteristic measures for describing a problem.

Since there is little industrial data available, problem specifications must be gen-

erated synthetically. The Gentap tool has been created for this purpose. It addresses both software and hardware aspects of problem generation in greater detail than any previous work in the field of real-time task allocation.

Gentap is controlled by a number of parameters which correspond to characteristics of the problem specifications generated. This facilitates the classification of problems so that interaction between task allocation algorithms and different types of problem can be easily evaluated for either performance or solution quality. The parameters are carefully chosen to give a good balance between having sufficient control over problem generation and not having to set so many parameters that the tool is awkward to use. For example many different shapes and sizes of task graphs can be produced by adjusting just three parameters.

Particular emphasis has been placed on generating task utilisations which are distributed in a similar fashion to those of real systems. This is based on fitting a beta distribution to existing data and then sampling from the distribution. Algorithms for sampling were demonstrated to be efficient and accurate estimation of the number of samples required to achieve a particular target utilisation were given. The utilisation level can be adjusted independently of the number of tasks by adjusting the distribution.

5

Algorithm Design And Analysis

5.1 Introduction

This chapter is concerned with the engineering of a search algorithm for task allocation. In the review of task allocation solution methods in section 2.3, metaheuristic local search was revealed to have been successfully used by a number of previous authors [4, 53, 121, 122]. Simulated annealing, in particular, is often used as a baseline comparison for development of new methods [56, 123].

A hierarchical cost function is used, which includes a configurable guidance heuristic. The guidance heuristic includes a number of strategies for improving the performance of the search algorithm. The No Free Lunch Theorem [162] establishes that improved performance may be gained by configuring an optimisation algorithm specifically for different classes of problem. This chapter describes an experimental method for classifying problems and simultaneously assigning suitable biases to the strategies in the guidance heuristic for each problem class. The algorithm is applied to problems with different characteristics and the effects of changing the policy for scheduling the system are also examined.

A need to retune the algorithm quickly for different classes of problem is motivated by the fact that software engineering tools must be readily adaptable to new situations. The time taken to setup the tool for use with a particular problem should be kept low. This was one aspect of changeability previously discussed in section 1.1.3. If results show that it is possible to readily reconfigure the algorithm for different problem types, then these results also support the claim made in hypothesis statement 2 in section 3.2.1.

An efficient, systematic experimental approach is used for configuring the algorithm. The efficiency of the method comes from the ability to analyse performance based on prematurely terminated runs of the search algorithm using *survival anal-*

ysis. Survival analysis was first applied to the study of heuristic search algorithms performance by Poulding motivated by the need to configure an earlier version of the algorithm presented in this chapter. This work was previously published in a paper by Poulding and this author [163].

The cost function used in the work of Poulding et al. [163] used a single large experiment for tuning all cost function weightings. The objectives in the cost function used here are instead combined using a hierarchy of weightings. This gives a natural way to group weightings and use separate experiments to tune each group. When additional objectives are added in future chapters, the higher level weighings of the hierarchy are then rebalanced without re-tuning all weighting values. This puts an emphasis on engineering practicality over optimality of algorithm performance.

The techniques of Poulding et al. [163] are used to discover problem characteristics which are most significant to the performance of the task allocation algorithm presented. This information is used to create problem classes for which the algorithm should be separately tuned. Previous work by this author [164] also tuned a task allocation algorithm for different problem classes. However, the systematic method of choosing the problem classes has not been published previously.

The main motivation for choosing search over other optimisation techniques is flexibility. The other main categories of solution method are problem specific heuristics and the reformulation of task allocation into a well studied optimisation problem such as integer linear programming [81] or satisfiability [56]. When the solution method is built around a particular test, changing the test becomes non-trivial. It is not clear that all schedulability tests are amenable to being reformulated, especially ones based on the construction of a static schedule where worst case response times do not have any obvious mathematical formulation. Reformulating the schedulability test can also introduce more pessimism as previously discussed in section 2.3.4.

Many problem specific heuristics reduce the solution space by pre-clustering tasks together and then assigning clusters of tasks to processors [3, 107]. This can be done in combination with other task allocation problem solving strategies by including multiple heuristics within the objective function of a metaheuristic search.

There are three major sections in the chapter. Section 5.2 introduces the local search algorithm variations and cost function which will be evaluated and configured for use on task allocation problems. Section 5.3 describes the experimental method and analysis techniques that are used for configuring algorithm parameters. Section 5.4 describes the sequence of experiments which were performed with discussion of the results obtained. A summary of the work carried out in these sections is given in section 5.5

```
   input  : init ;                                   /* the initial solution */
   input  : ssize ;                                  /* sample size parameter */
   input  : inittemp ;                               /* initial temperature */
   input  : maxinner ;                               /* inner loop samples */
   input  : stop ;                                   /* stopping condition function */
   input  : randrestart ;                            /* random restart samples */
   output: bestconf ;                                /* best configuration found */
 1 begin
 2   curconf = init;
 3   bestconf = init;
 4   curcost = costfn(curconf) ;
 5   bestcost = curcost;
 6   t = inittemp;
 7   randrestartcount = 0 ;
 8   repeat
 9     i = 0 ;
10     repeat
11       if randrestart ≥ 0 and randrestartcount ≥ randrestart then
12         (newconf, newcost) = randsolution() ;
13         randrestartcount = 0 ;
14         curconf = newconf;
15         curcost = newcost;
16       else
17         (newconf, newcost) = samplenei(curconf, ssize) ;
18         randrestartcount = randrestartcount + 1 ;
19         delta = newcost − curcost;
20         if (delta ≤ 0) or (t > 0 and randuniform() < exp(−delta/t)) then
21           curconf = newconf;
22           curcost = newcost;
23         endif
24       endif
25       i = i + 1 ;
26       if newcost < bestcost then
27         bestconf = newconf;
28         bestcost = newcost;
29         randrestartcount = 0 ;
30       endif
31     until i == maxinner or stop() ;
32     t = t ∗ 0.99 ;
33   until stop() ;
34   return bestconf
35 end
```

Figure 5.1: Local search algorithm for hill descent or simulated annealing

Algorithm	Relevant Parameters
Simulated annealing	randrestart $= -1$, inittemp > 0, ssize > 0, maxinner > 0
Hill descent with random restart	randrestart > 0, inittemp $= 0$, ssize > 0
Random search	randrestart $= 0$

Table 5.1: Input parameter configurations for algorithm in figure 5.1

5.2 Search Algorithm

Two forms of local search and a random search are used in experiments in this chapter. All three types of search are implemented by the algorithm in figure 5.1 with input parameters deciding which variation is used. The two local search algorithms are simulated annealing and hill descent with random restarts. The parameters relevant to each algorithm are shown in table 5.1. These parameters and others are provided to the search tool using the XML file format given in appendix D.

For the hill descent algorithm, the initial temperature (inittemp) should be set to 0 and the random restart parameter (randrestart) defines how many solutions are evaluated without any improvement in cost value before a restart occurs. The sample size (ssize) is the number of solutions from the neighbourhood which are generated and evaluated per iteration. The best solution from the sample set is compared with the last accepted solution. Therefore, increasing the sample size makes the algorithm more greedy, giving the possibility of improved performance at the increased risk of becoming stuck in poor local optima.

The simulated annealing algorithm works similarly to hill descent but with different acceptance criteria. Although it needn't be the case, random restart is switched off whenever simulated annealing is used, achieved by setting the parameter to -1. An initial temperature needs to be set for simulated annealing. Assuming geometric cooling, there are two factors affecting temperature changes: the cooling rate and the number of solutions evaluated between changes in temperature. The algorithm in figure 5.1 fixes the cooling rate to 0.99 and the number of inner loop iterations is left as an input parameter (maxinner).

Setting the random restart parameter (randrestart) to 0 causes a new random solution to be generated on every loop iteration. This is an unguided random search.

The `stop` parameter is actually a function which evaluates a stopping condition and returns a boolean value. It can use data external to the main search routine shown in figure 5.1, such as a limit on the number of solutions evaluated. The stopping conditions used are given alongside the description of each experiment.

The search algorithm requires two further functions, `samplenei` and `costfn` which define the neighbourhood and cost function.

```
   input  : config ;                                   /* the current configuration */
   output: config_delta ;                  /* a change to the current configuration */
 1 begin
 2   object = rand_schedulable_object(config) ;
 3   r = randuniform() ;                                          /* value in [0,1) */
 4   if r < 0.5 then
 5     config_delta = modify_alloc(object, config)
 6   else
 7     config_delta = modify_priority(object, config)
 8   endif
 9   return config_delta
10 end
```

Figure 5.2: The `modify_config` function which generates a configuration change

```
   input  : config ;                                   /* the current configuration */
   input  : object ;                                    /* the object to change */
   output: config_delta ;                  /* a change to the current configuration */
 1 begin
 2   scheduler_list = avail_scheduler_list(object) ;
 3   if (object_type(object) == message) and
 4        (length(scheduler_list) == 0) then
       /* If object is a message and no suitable schedulers to move it to, reallocate
          source or destination task instead                                        */
 5     r = randuniform() ;                                        /* value in [0,1) */
 6     if r < 0.5 then
 7       config_delta = modify_alloc(object.from_task);
 8     else
 9       config_delta = modify_alloc(object.to_task);
10     endif
11   else
12     config_delta = new_configuration() ;
13     set_alloc(config_delta, object, rand_object(scheduler_list)) ;
14   endif
15   return config_delta
16 end
```

Figure 5.3: The `modify_alloc` function which changes the allocation of an object

5.2.1 Local Search Neighbourhood Design

The `samplenei` function generates ssize possible changes to the current solution and returns the one with the lowest cost. The `modify_config` function generates modifications to an existing solution. Each change is applied, evaluated and then undone so that the next change can be tested. The `modify_config` function is shown in figure 5.2. It flips a fair coin in order to decide whether a change to an object allocation or object priority will be made. An object to be modified is also selected at random. It is chosen from the set of all tasks and messages treated together as a set of schedulable objects. Every object has an equal chance of being chosen. The way in which either the allocation or priority of an object is modified is now explained.

Figure 5.3 shows the function for making a change to the object's allocation. This function's first step is to generate a list of possible scheduling resources to reallocate

```
   input  : object ;                          /* schedulable object to calculate list for */
   output: scheduler_list ;                   /* list of possible schedulers to allocate to */
 1 begin
 2   scheduler_list = {} ;
 3   current_scheduler = object.scheduler;
 4   if object_type(object) == task then
 5     scheduler_list.add(all_processors()) ;
 6     scheduler_list.remove(current_scheduler) ;
 7   else
 8     from_scheduler = object.from_task.scheduler;
 9     to_scheduler = object.to_task.scheduler;
10     if from_scheduler == to_scheduler then
         /* both source and destination on same processor                          */
11       if from_scheduler != current_scheduler then
12         scheduler_list.add(intra_proc_net(from_scheduler)) ;
13       endif
14     else
         /* source and destination on different processors                         */
         /* add any inter-processor networks connected to both processors         */
15       scheduler_list.add( inter_proc_net(from_scheduler) ∩
16                           inter_proc_net(to_scheduler) ) ;
17       if length(scheduler_list) == 0 then
           /* no networks connected to both processors                            */
           /* add networks connected to one or other                              */
18         scheduler_list.add( inter_proc_net(from_scheduler) ∪
19                             inter_proc_net(to_scheduler) ) ;
20         if length(scheduler_list) == 0 then
             /* both processors must be lone processors                           */
             /* add all inter-processor networks                                  */
21           scheduler_list.add(all_inter_proc_net()) ;
22         endif
23       endif
24       scheduler_list.remove(current_scheduler) ;
25     endif
26   endif
27   return scheduler_list
28 end
```

Figure 5.4: The `avail_scheduler_list` function for calculating possible allocation changes for an object

the object to. If the object is a message and no suitable alternative schedulers exist, then either the sending or receiving task is changed instead. The function which generates a list of suitable schedulers for an object to move to is given in figure 5.4.

If the object is a task, then a list of all processors except the one the task is currently allocated to is returned. If allocation restrictions are defined for an object then the list is adjusted appropriately. For simplicity this is not shown in figure 5.4 since this feature is not used prior to the work in chapter 8.

The generation of possible networks for message reallocation uses a heuristic to favour networks which provide a path between dependent tasks over those that don't and intra-processor networks over inter-processor networks.

If both the sending and receiving task are allocated to the same processor then a

```
   input  : config ;                                   /* the current configuration */
   input  : object ;                                   /* the object to change */
   output: config_delta ;                  /* a change to the current configuration */
 1 begin
 2   sort_by_priority(config) ;
 3   config_delta = new_configuration() ;
     /* select another random object from the set of objects in the configuration
        minus the object to be changed                                              */
 4   shuffle_object = rand_schedulable_object(config \ {object}) ;
     /* find index of objects within configuration                                 */
 5   shuffle_index = config_index(config, shuffle_object) ;
 6   object_index = config_index(config, object) ;
     /* set object priority to be that of randomly chosen shuffle object          */
 7   set_priority(config_delta, object, shuffle_object.priority) ;
     /* adjust priority of all objects in between input object and shuffle object
        according to their relative positions within the configuration             */
 8   if shuffle_index > object_index then
 9     for i from (object_index + 1) to shuffle_index do
10       config_object = get_config_object(config, i)
         set_priority(config_delta, config_object,
11                    get_config_object(config, i − 1).priority) ;
12     endfor
13   else
14     for i from shuffle_index to (object_index − 1) do
15       config_object = get_config_object(config, i)
         set_priority(config_delta, config_object,
16                    get_config_object(config, i + 1).priority) ;
17     endfor
18   endif
19   return config_delta
20 end
```

Figure 5.5: The `modify_priority` function

list containing just the intra-processor network for that processor is returned. If the message is already allocated to this network then an empty list is returned.

If the source and destination task of the message are allocated to separate processors then all networks connecting these processors are returned. If this list is empty, then networks connected to either one of the processors is returned instead. As a last resort all inter-processor networks are returned.

The `modify_config` function may choose to make a priority change instead of an allocation change. The function which creates a configuration which applies priority changes is given in figure 5.5. All object priorities within the system are kept unique. This ensures that when an allocation change is made, all priorities on each processor are unique and there is no ambiguity in the ordering. To maintain a unique set of priorities, priority changes are made by exchanging priorities between objects. To change the priority of the random object provided to the `modify_priority` function, another object is selected and its priority is used as the new priority value. The priorities of all objects with a priority between the two chosen ones are shifted one slot to fill the gap left by the initial priority change. This is preferred over simply

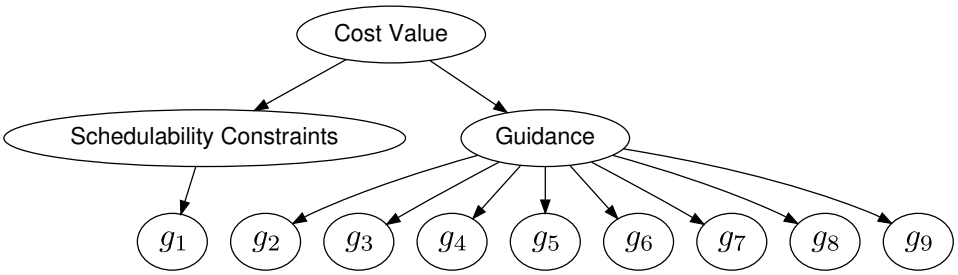


Figure 5.6: Cost function hierarchy

swapping the priorities of the two objects, since the aim is to cause minimal disruption to the ordering of objects other than the one chosen to be modified. This aids the extended version of this algorithm presented in chapter 6 which tries to solve a new problem with as few changes as possible to the configuration.

This algorithm changes priorities of objects globally throughout the entire system. A priority change will only have an immediate effect if it changes the relative priorities of objects on the same processor or network. However, any priority change can have an affect on subsequent object allocation changes.

5.2.2 Generating A Random Solution

A way of generating random solutions is needed for choosing an initial solution and also for performing random restarts within the algorithm in figure 5.1.

To set allocations, the `modify_alloc` function in figure 5.3 is applied to each object at in turn. Since the heuristic for allocating messages in the `avail_scheduler_list` in figure 5.4 depends on the positioning of the tasks, all tasks are allocated first.

Priority values from 1 to $K + L$, where $K + L$ is the combined number of tasks and messages, are assigned to each object in turn. To give some randomness to the assignment the priority values are first shuffled using a Knuth shuffle [165].

5.2.3 Cost Function

The cost value is a weighted sum of the results of lower level functions, called *cost subfunctions*. Subfunctions are grouped and then composed in a hierarchical fashion. This allows the problem of assigning weights to subfunctions to also be decomposed as is common when deciding between objectives in decision theory [140]. Bicking et al. [52], whose work was previously described in section 2.3.3.1, used this approach to combine penalties for unsatisfied constraints with a dependability objective.

A 3 level function hierarchy is used as shown in figure 5.6. The root node is the final cost value. Below that there is a value for unsatisfied schedulability constraints and a value for a guidance heuristic. The schedulability value is a simple measure which is required to know when the constraint satisfaction problem has been solved. For the experiments in this chapter, a search will terminate when this value reaches

0 since there is no additional quality optimisation goal. All schedulable solutions are deemed equal quality. The guidance heuristic provides a way of introducing problem specific knowledge into the search.

The bottom level of the function hierarchy contains the measurable attributes. The guidance heuristic subfunctions were developed based on experience and the work of previous authors which was described in section 2.3.1. Examples are the use of load balancing by Chu and Lan [105] and the grouping of communicating tasks by Ramamritham [3]. The effectiveness of each subfunction is assessed in section 5.4.

Normalisation is applied at all levels of the hierarchy so that the final cost and intermediate values lie in the range $[0, 1]$. The bottom level subfunctions are labelled $g_1, \ldots, g_9$. g_1 is used to measure how many objects meet their timing requirements. $g_2, \ldots, g_9$ are combined into the value for the guidance heuristic as follows.

$$guidance(\mathcal{C}) = \frac{\sum_{i=2}^{9} w_i g_i(\mathcal{C})}{\sum_{i=2}^{9} w_i} \tag{5.1}$$

where $\mathcal{C}$ is the configuration being assessed and $w_2, \ldots, w_9$ are weightings used to affect the size of the contribution of each subfunction to the guidance heuristic.

Since only a single value makes up the schedulability subfunction, no weighting values are needed and it is simply defined as:

$$sched(\mathcal{C}) = g_1(\mathcal{C}) \tag{5.2}$$

Equations (5.1) and (5.2) are combined into the single cost value as follows

$$f(\mathcal{C}) = \frac{w_{sched} sched(\mathcal{C}) + w_{guidance} guidance(\mathcal{C})}{w_{sched} + w_{guidance}} \tag{5.3}$$

where further weighting values are used to balance the contribution of the guidance heuristic against testing for schedulability constraints. If the guidance heuristic is well designed then configurations which give low values of *guidance* should coincide with those which give low values of *sched*.

5.2.3.1 Cost Subfunction Definitions

This section defines the cost subfunctions $g_1, \ldots, g_9$. The notation used is the same as that previously given in table 2.1. The sets of tasks, messages, processors and network links are denoted as $\mathcal{T}$, $\mathcal{M}$, $\mathcal{P}$, and $\mathcal{N}$ respectively. $\mathcal{S} = \mathcal{T} \cup \mathcal{M}$ is the set of all schedulable objects. The notation for the cardinality of a set X is $|X|$. Directly dependent (DD) tasks are a pair of tasks which have a message sent between them. Indirectly dependent (ID) tasks appear in the same transaction but are not necessarily adjacent in the task graph. Functions labelled g_i are the cost subfunctions and functions labelled h_i are helper functions for g_i.

Subfunction g_1 assesses the number of unschedulable objects, by comparing the

calculated response time for each schedulable object, R_τ, with its deadline D_τ.

$$h_1(\tau) = 1 \text{ if } R_\tau > D_\tau \text{ else } 0 \tag{5.4}$$

$$g_1 = \frac{1}{|\mathcal{S}|} \sum_{\tau \in \mathcal{S}} h_1(\tau) \tag{5.5}$$

The method of obtaining response times depends on the system model and the schedulability test used.

Subfunction g_2 counts how many DD tasks are allocated to processors not connected by an inter-processor network. Let *al* map a task to its allocated processor and V map a processor to the set of processors to which it is connected by a network. The functions *src* and *dest* give the sending and receiving task of a message.

$$c(\tau, \upsilon) = 1 \text{ if } V(\text{al}(\tau)) \cap V(\text{al}(\upsilon)) = \emptyset \text{ else } 0 \tag{5.6}$$

$$g_2 = \frac{1}{|\mathcal{M}|} \sum_{\rho \in \mathcal{M}} c(\text{src}(\rho), \text{dest}(\rho)) \tag{5.7}$$

The next subfunction penalises objects which cannot receive their input or send their output due to their allocation. For example, there may exist a network connecting the allocated processors of a pair of DD tasks but the message isn't allocated to it. This situation is not penalised by g_2 but is by g_3. However all situations penalised by g_2 are also penalised by g_3 so there is a strong dependency between the two subfunctions. Interactions between subfunctions are accounted for when deciding the values of subfunction weightings in section 5.4.

Firstly, two functions are defined which give objects connected to the input and output of a schedulable object in the task graph. The definitions are conditional on whether the object is a task or message.

$$\text{in}(\tau) = \begin{cases} \{\rho \in \mathcal{M} : \text{dest}(\rho) = \tau\} \text{ if } \tau \in \mathcal{T} \\ \{\text{src}(\tau)\} \text{ if } \tau \in \mathcal{M} \end{cases} \tag{5.8}$$

$$\text{out}(\tau) = \begin{cases} \{\rho \in \mathcal{M} : \text{src}(\rho) = \tau\} \text{ if } \tau \in \mathcal{T} \\ \{\text{dest}(\tau)\} \text{ if } \tau \in \mathcal{M} \end{cases} \tag{5.9}$$

The helper function h_3 counts how many objects which are connected in the task graph are not allocated to connected scheduling resources. g_3 normalises h_3 to give the cost subfunction value.

$$\begin{aligned} h_3(\tau) = |\{\upsilon \in \text{out}(\tau) : \text{al}(\upsilon) \notin V(\text{al}(\tau))\}| + \\ |\{\upsilon \in \text{in}(\tau) : \text{al}(\upsilon) \notin V(\text{al}(\tau))\}| \end{aligned} \tag{5.10}$$

$$g_3 = \frac{\sum_{\tau \in \mathcal{S}} h_3(\tau)}{\sum_{\tau \in \mathcal{S}} \left[|\text{in}(\tau)| + |\text{out}(\tau)| \right]} \tag{5.11}$$

Subfunctions such as this can be normalised in more than one way. g_3 is normalised over the total number of inputs and outputs of all objects. This means that each invalid connection contributes the same amount to the cost. An alternative would have been to normalise the value for each object and then divide through by the number of objects. This means that, for example, two objects with half of their connections invalid would contribute the same amount to the cost though one could have one input and one output whereas the other has three inputs and one output. The decisions on how to normalise cost subfunctions are taken based upon what this author believes makes the behaviour of each function easiest to comprehend. No claims are made regarding the performance of the overall search algorithm with relation to these decisions.

Grouping communicating objects onto the same scheduler to reduce communication overheads was the most popular heuristic of the work reviewed in section 2.3. Both subfunctions g_4 and g_5 are designed to achieve this aim in slightly different ways. g_4 encourages objects within the same transaction to be allocated to a smaller number of scheduling resources. However, this does not stop DD tasks exchanging communications back and forth between processors. g_5 penalises messages sent between DD tasks on different processors. Function g_5 tries to get an immediate gain by moving communicating tasks to the same processor. g_4 is more forward looking since moving tasks which are not DD may not immediately improve the solution but should hopefully lead to a good solution after a number of such moves.

Let the set of all transactions be TRANS and the set of schedulable objects contained in transaction r be TRANS_r. V_r is the set of tasks in TRANS_r. For each $\tau_i \in V_r$, the number of tasks allocated to the same scheduler as τ_i and also in V_r is a_{ri}. A grouping value and its maximum for each transaction is:

$$\gamma_r = |V_r| - \sum_{i=0}^{|V_r|-1} \frac{a_{ri}}{|V_r|} \qquad \gamma_{r_{\text{MAX}}} = \frac{|V_r|(|\mathcal{P}|-1)}{|\mathcal{P}|}$$

The theoretical maximum value occurs when tasks are equally spread among processors and $a_{ri} = |V_r|/|\mathcal{P}|$ for all i. Similar formulae can be defined for messages with W_r being all messages in TRANS_r. Using $\gamma_{r_{\text{MAX}}}$ to normalise γ_r and then summing over all transactions, the subfunction formula is

$$\begin{aligned} g_4 = &\frac{|\mathcal{P}|}{2|\text{TRANS}|(|\mathcal{P}|-1)}\left[|\text{TRANS}| - \sum\nolimits_r \sum\nolimits_i \frac{a_{ri}}{|V_r|^2}\right] \\ &+\frac{|\mathcal{N}|}{2|\text{TRANS}|(|\mathcal{N}|-1)}\left[|\text{TRANS}| - \sum\nolimits_r \sum\nolimits_i \frac{a_{ri}}{|W_r|^2}\right] \end{aligned} \tag{5.12}$$

Function g_5 penalises any DD tasks which aren't allocated to the same processor.

This is calculated by considering the source and destination task of each message.

$$g_5 = \frac{1}{|\mathcal{M}|}|\{\rho \in \mathcal{M} : \text{al}(\text{src}(\rho)) \neq \text{al}(\text{dest}(\rho))\}| \tag{5.13}$$

g_6 is a subfunction which performs sensitivity analysis on the WCETs of schedulable objects. A value, $\text{SCAL}_{\mathcal{S}}$, is calculated that is the largest factor by which WCETs can be scaled while keeping the system schedulable. This value can be found using a binary search and will be less than 1 when the system is unschedulable under the current configuration or greater than 1 when it is schedulable. This is converted into a cost value as follows.

$$g_6 = e^{-\alpha \text{SCAL}_{\mathcal{S}}} \tag{5.14}$$

where α is a parameter for changing the shape of the function which is set to $\frac{2}{3}$. If the system is not schedulable for any value of $\text{SCAL}_{\mathcal{S}}$ then $g_6 = 1$. This will be the case when the allocation does not allow objects to communicate as required.

A load balancing subfunction g_7 is based upon the variance of the utilisations of processors:

$$g_7 = \sqrt{\frac{\sum_{\sigma \in \mathcal{P}} (U_\sigma - \mu)^2}{(|\mathcal{P}| - 1)\mu^2 + (u - \mu)^2}} \tag{5.15}$$

where U_σ is the utilisation of processor σ, $u = \sum_{\sigma \in \mathcal{P}} U_\sigma$ is the total utilisation, and μ is the mean utilisation. The normalising denominator is the maximum variance which occurs when the utilisation of all but one processors is 0. If the load is perfectly balanced so that $U_\sigma = \mu$ for all σ then the value of g_7 is also 0. Extremely unbalanced solutions are usually avoided by the search as many task deadlines will be missed. This makes high values of g_7 rare. Applying the square root operator gives larger differentials between cost values for reasonably well balanced solutions.

Note that this formula is encouraging the spreading of tasks between processors putting it in direct conflict with the aims of the grouping subfunctions g_4 and g_5. Which should be given the most weighting within the guidance heuristic function will be problem dependent.

It is known that any processor with over 100% utilisation will never be schedulable. Subfunction g_8 supplements the load balancing function and counteracts grouping subfunctions g_6 and g_7 by giving an additional penalty to processors with greater than 100% utilisation.

$$g_8 = \frac{|\{l \in \mathcal{P} \cup \mathcal{N} : U_l > 100\}|}{|\mathcal{P} \cup \mathcal{N}|} \tag{5.16}$$

Subfunction g_9 measures priority ordering which does not correlate with the order of objects in a transaction. This generally leads to poorer response times since the object waiting to receive data can cause interference to the object it is receiving data from. In static cyclic scheduling, the priority value is used to influence the objects

Subfunction	Parent	Reason for cost penalty
g_1	Schedulability	Objects failing schedulability test
g_2	Guidance	Directly dependent tasks with no communication path between them
g_3	Guidance	Directly dependent tasks with message not allocated to network connecting tasks' scheduling resources
g_4	Guidance	Tasks in same transaction allocated to different processors
g_5	Guidance	Directly dependent tasks allocated to different resources
g_6	Guidance	Fragility of schedulability tests to changes in execution / communication times (sensitivity analysis)
g_7	Guidance	Unbalanced processor loads
g_8	Guidance	Schedulers with utilisation greater than 1
g_9	Guidance	Schedule ordering attribute not correlated with dependency ordering

Table 5.2: Summary of cost subfunctions

position in the schedule so lower values (higher priorities) appear earlier.

$post(\tau)$ is the set of all objects that follow τ.

$$g_9 = \frac{\sum_{\tau \in \mathcal{S}} |\{\upsilon \in \text{post}(\tau) \text{ and } P_\upsilon < P_\tau\}|}{\sum_{\tau \in \mathcal{S}} |\text{post}(\tau)|} \tag{5.17}$$

A summary of all the cost subfunctions in this section is given in table 5.2

5.2.4 Implementation Details

The software within which the search algorithm is implemented is named Toast (loosely based on an acronym for Task Ordering and Allocation Search Tool). Toast is implemented in the C language.

Experiments with Toast are conducted on a large network of machines connected using the BOINC [166] infrastructure for grid computing. There is a range of platforms within the network: 32 bit and 64 bit Linux as well as Windows machines. It is a requirement that an experimental result does not depend on which machine a job of the experiment is sent to. Decisions within the search algorithm are sensitive to pseudo-random number generation and to comparisons of floating point values. Both of these are platform / compiler dependent. Therefore, instead of using the default library pseudo-random number generator, a Mersenne Twister implementation [167] is used instead. This also provides access to the internal state of the random number generator which can be used to checkpoint and restart jobs without affecting the result. To deal with differences between floating point implementations and register sizes, an error tolerance is used in comparisons. If values are within 10^{-10} of each other, they are considered equal.

5.3 Experimental Aims And Design

The reason for conducting an experiment is to increase understanding in the relationship between factors, also known as *covariates*, which are inputs to the system and one or more responses which can be measured. The aim of the experiments in this chapter is to discover how both algorithm parameters and problem characteristics affect the performance of the search algorithm.

Setting up a search algorithm is non-trivial. One point of view is that parameters can be set manually by trial and error until performance is considered acceptable. However, for a large number of parameters whose effect is not independent of each other, this is infeasible. A one-factor-at-a-time (OFAT) approach where each parameter is tuned in turn is a systematic method but a poor one [168]. This does not take account of dependencies between parameters which may have a significant effect on how the algorithm operates in terms of performance or quality.

This section covers a suite of techniques which will be used to decide which experiments to run and how to analyse the data generated. Further information on all the topics in sections 5.3.1 to 5.3.3 can be found in Montgomery's book [169] which covers experimental design and response surface methodology in depth. Section 5.3.5 is covered by Lee and Wang's book [170] on survival data analysis.

5.3.1 Response Surface Modeling

The method used to find the relationship between factors of interest and response is response surface modelling, described at length by Montgomery [169] and advocated by Ridge [171] as a good means for studying heuristic search algorithms.

Response surface modelling constructs a model relating factors to the response from the data generated by a set of experiments. The type of model needs to be chosen prior to the data being fitted.

The simplest model is a linear model which links the response, y, to each factor $x_1, \ldots, x_q$ as follows

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_q x_q + \epsilon \tag{5.18}$$

where ϵ is an error term which captures variations due to factors other than $x_1, \ldots, x_q$.

It is common for the influence of a factor on the response to be dependent on the values of other factors. This can be modeled with the following interaction model.

$$y = \beta_0 + \sum_{i=1}^{q} \beta_i x_i + \sum_{i=1}^{q-1} \sum_{j=i+1}^{q} \beta_{ij} x_i x_j + \epsilon \tag{5.19}$$

The above equation only includes interaction effects for combinations of two factors. It is also possible to include higher order interaction effects as long as there is sufficient experimental data to fit the model to. This is discussed further in section 5.3.2.

Some factors can have a quadratic relationship with the response. That is, there is a value of the factor which minimises the response and either increasing or decreasing the factor will increase it or vice-versa. The following is the quadratic effects model

$$y = \beta_0 + \sum_{i=1}^{q} \beta_i x_i + \sum_{i=1}^{q-1} \sum_{j=i+1}^{q} \beta_{ij} x_i x_j + \sum_{i=1}^{q} \beta_{ii} x_i^2 + \epsilon \tag{5.20}$$

The interaction and quadratic effects models are still linear models in the sense that a new factor x_{ij} can be used to substitute $x_i x_j$ and put in the form

$$y = \boldsymbol{\beta}^T \boldsymbol{x} + \epsilon \tag{5.21}$$

where $\boldsymbol{\beta}$ is the vector of *regression coefficients* to be found and $\boldsymbol{x}$ contains the factor values. Assuming the first value of $\boldsymbol{\beta}$ is β_0, then the corresponding first element of $\boldsymbol{x}$ is 1 to create the intercept term.

When an experiment is conducted, each run r in the experiment uses a particular vector of factor values $\boldsymbol{x}_{(r)} = \{1, x_{(r)1}, \ldots, x_{(r)q}\}$ and has a corresponding observed response $y_{(r)}$. This results in a set of equations:

$$y_{(r)} = \boldsymbol{\beta}^T \boldsymbol{x}_{(r)} + \epsilon_{(r)} \tag{5.22}$$

where r ranges from 1 to the number of runs in the experiment. The method of least squares regression finds a vector of values, say $\hat{\boldsymbol{\beta}}$, which minimises the sum of the squares of the error values. Details of how to do this are found in Montgomery [169] and also well supported by the R language for statistical computing [160]. Many experiments in this thesis generate censored data for which more advanced techniques, summarised in section 5.3.5, are required. However, the principle of finding coefficients to minimise error between the assumed model and real data is the same.

Once a response surface has been constructed, the point on the surface which minimises the response (e.g. time to complete, requirements not met, etc.) can be found. Since the surfaces are relatively simple and mainly convex (depending on how dominant interaction terms are), standard optimisation packages such as LINDO [172], are suitable for finding minimal points on the surface.

5.3.2 Factorial Designs

Each experiment consists of a number of runs of the search algorithm with different input factor values. An *experimental design* provides a framework for deciding how many runs are needed and which combination of factor levels to use for each run. A full factorial design runs all combinations of each level of each factor. Therefore, an experiment with q factors at d levels requires d^q runs. For example, a full factorial

design for 3 factors at 2 levels can be described by the following matrix:

$$\begin{array}{ccc} x_1 & x_2 & x_3 \\ -1 & -1 & -1 \\ +1 & -1 & -1 \\ -1 & +1 & -1 \\ +1 & +1 & -1 \\ -1 & -1 & +1 \\ +1 & -1 & +1 \\ -1 & +1 & +1 \\ +1 & +1 & +1 \end{array} \tag{5.23}$$

where -1 represents the low value for a factor and $+1$ is the high value. The actual values should be chosen based on experience around an area of interest in the design space. Often a series of *screening experiments* are performed in order to to build up knowledge of the factor space before more in depth experiments are run [173].

A full factorial design, though not the most efficient in terms of the number of runs, is able to fit a model with any number of interactions between factors. In order to fit a quadratic effects model, at least 3 factor levels should be used.

5.3.3 Mixture Designs

There are times at which an experimenter is not interested in absolute factor values but in ratios of factors which optimise the response. This topic is covered since it applies to finding values for normalised cost function weightings as in equation (5.1) where the result is independent of the sum of the weightings.

Let there be q factors which are known to sum to a constant value. This constant is chosen to be 1 without loss of generality.

$$\sum_{i=1}^{q} x_i = 1 \tag{5.24}$$

Using equation (5.24), the linear model from equation (5.18) can be written as

$$y = \beta_0 \left(\sum_{i=1}^{q} x_i \right) + \beta_1 x_1 + \cdots + \beta_q x_q + \epsilon \tag{5.25}$$

Therefore, a suitable linear model for a mixture experiment is

$$\begin{aligned} y &= (\beta_0 + \beta_1)x_1 + \cdots + (\beta_0 + \beta_q)x_q + \epsilon \\ &= \beta_1' x_1 + \cdots + \beta_q' x_q + \epsilon \end{aligned} \tag{5.26}$$

which notably lacks the intercept term of equation (5.18).

The form for a mixture model with first order interactions is

$$y = \sum_{i=1}^{q} \beta_i x_i + \sum_{i=1}^{q-1} \sum_{j=i+1}^{q} \beta_{ij} x_i x_j + \epsilon \tag{5.27}$$

Smith [174] reformulates equation (5.27) as follows

$$x_q = 1 - (x_1 + \cdots + x_{q-1}) \tag{5.28}$$

$$y = \beta_q + \sum_{i=1}^{q-1} (\beta_i - \beta_q + \beta_{iq}) x_i + \sum_{i=1}^{q-2} \sum_{j=i+1}^{q-1} (\beta_{ij} - \beta_{iq}) x_i x_j - \sum_{i=1}^{q-1} \beta_{iq} x_i^2 + \epsilon \tag{5.29}$$

$$y = \beta'_0 + \sum_{i=1}^{q-1} \beta'_i x_i + \sum_{i=1}^{q-2} \sum_{j=i+1}^{q-1} \beta'_{ij} x_i x_j + \sum_{i=1}^{q-1} \beta'_{iq} x_i^2 + \epsilon \tag{5.30}$$

which is now in the same form as the quadratic effects model in equation (5.20) for $q-1$ factors. This formulation is used for estimating regression coefficients in Cox's proportional hazard model in section 5.3.5 which requires factor values not to be linear combinations of each other in order to uniquely determine the coefficients.

A suitable experimental design for fitting data to mixture models is the *simplex lattice* design. Assuming the sum of the factors is 1, the factor values are set at $0, \frac{1}{m}, \frac{2}{m}, \ldots, 1$ in order to achieve a $m+1$ level design.

For 3 factors at 3 levels, the values used would be:

$$\begin{array}{ccc} x_1 & x_2 & x_3 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ \frac{1}{2} & \frac{1}{2} & 0 \\ \frac{1}{2} & 0 & \frac{1}{2} \\ 0 & \frac{1}{2} & \frac{1}{2} \end{array} \tag{5.31}$$

In general a simplex lattice design for q factors at $m+1$ levels requires $\frac{(q+m-1)!}{m!(q-1)!}$ runs [169].

Sometimes, a centre point is added to the design since all of the above only include mixtures where at least one element of the mixture is unused. For q factors, the centre point is at $(\frac{1}{q}, \frac{1}{q}, \ldots, \frac{1}{q})$. This would be $(\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$ for the above example.

In order to focus on a smaller area of the design space, a constrained mixture can be used. Let the minimum value of factor i be l_i. The values x'_i are chosen according to a simplex lattice. The factor levels used in the experiment, x_i are then set according to

$$x_i = \frac{x'_i - l_i}{1 - \sum_{j=1}^{q} l_j} \tag{5.32}$$

If all lower bounds are the same so that $l = l_1 = \cdots = l_q$ then this reduces to

$$x_i = \frac{x'_i - l}{1 - ql} \tag{5.33}$$

5.3.4 Factors And Response For Algorithm Configuration

The aim of the experiments in this chapter is to configure the parameters and cost function weightings for the search algorithm in figure 5.1. The response of interest for these experiments will be the number of solutions evaluated by the search before a valid solution is found. This is a response corresponding to the performance of the algorithm. Factors which can affect this result are:

1. algorithm parameter values and cost subfunction weightings
2. controlled problem characteristics
3. uncontrolled problem characteristics
4. the pseudo-random number generator initialisation seed

Algorithm parameters and weightings are the main factors of interest for the experiments in this chapter. These are easily controlled since they are specified for each experiment in an XML configuration file, whose format is given in appendix D.

Each individual problem has the potential to generate a different response. The Gentap tool described in section 4.4 accepts a number of parameters such as for the number of tasks and mean processor utilisation. These characteristics can therefore be controlled and their effects can be studied with the appropriate experimental setup. Other aspects of the problem will still vary even if the problem generation parameters are set in the exact same way. These are factors which are uncontrolled for these experiments. There are good reasons for not controlling more characteristics:

- As the number of problem characteristics increases, the size of a problem class with particular characteristic values diminishes. There is a risk that parameters become overfitted to a small group of problems.
- There are already more controllable problem characteristics than can be fully investigated in a reasonable amount of time given the available compute resources.

The technique of blocking [169] is used to reduce the impact of problem instance on results. Under this scheme each combination of factor values are run on the same set of problem instances. The ability to easily make exact replicas of problem instances is one of the advantages experimental computer scientists have over other experimentalists in physical sciences.

The pseudo-random number generator seed influences the generation of the random starting solution and subsequent decisions made by the search. The random seed is set as one of the input parameters to the algorithm. The seed itself is generated

by another random number generator using time based initialisation. It is assumed that there is no correlation between random seed assignment and runs for particular factor levels or problem characteristics so that over the course of an experiment the results are not biased by this factor.

Since experiments are run on a variety of platforms, the time taken will be dependent on the machine a job is sent to and whether other processes are running on that machine as well as the factors listed above. Using the number of solutions evaluated removes these nuisance factors from the experiment. Mean running times will be given at appropriate points as a guide to how long a problem takes to be solved. Machines in the BOINC connected grid typically have Intel Xeon processors ranging from 1.6GHz to 2.66GHz.

Another response of interest is the robustness of particular parameter settings. This can be measured using the variance of the performance response over a number of runs. The robustness response will not be explicitly optimised for but can be used to test whether a highly performant algorithm is robust.

The aim of these experiments is to find good parameter values for problems with certain characteristics. These values are found based on the model of the response surface. This model will be imperfect and so there is immediately some inaccuracy in the method. No claims regarding optimality are warranted or even of much use.

Parameters fitted to a single problem instance may well produce a very exploitative search which works well for that problem but fail to find solutions for other problems which require more exploration. On the other hand, parameters found for a very large test set of problems may perform poorly on a majority of problem instances since the search is unable to exploit problem specific information. For this reason, part of the investigation will include the separation of problems into classes based on problem characteristics to see how much, if any, benefit is gained from tuning parameters for specific characteristics.

There are dependencies between different cost function weightings, algorithm parameters and problem characteristics. A global approach, running experiments which varies all of these factors simultaneously would allow these interactions to be captured and analysed. Unfortunately, since the number of runs in an experiment typically grows exponentially with the number of factors, investigating all factors simultaneously is not feasible. To attempt to do so would also be a very high risk strategy since any small mistake found in the experimental procedure could result in several weeks of computation time being wasted.

Experiments are broken down into a sequence of manageable chunks. Whilst some interactions between factors will be lost, it is hoped that parameters can still be found which enable the search algorithm to solve task allocation problems in a moderate amount of time, which, after all, is the main objective. The hierarchical design of the cost function gives a natural way to break down experiments. Also, algorithm parameters will be tuned separately from cost function weightings.

5.3.5 Survival Analysis

Survival analysis [170] is an area of statistics most commonly associated with medical research but has a wide range of applications such as analysing failure of components in engineering. Survival analysis is concerned with the prediction of the point in time an event will occur based on data which has been previously collected.

Consider the following experiment. A researcher wants to gain an understanding of how diet affects the chances of someone getting a particular disease. The factors of interest could be age, the amount of fish eaten and total calories consumed per week. If the experiment is conducted over a period of 5 years, it is expected that some patients will get the disease while others will not. For those who do not, some may get the disease after the trial has ended and some may never suffer from it in their lifetimes. The data collected is *right censored* since for some participants the event will occur after the trial has ended but exactly when is unknown.

The statistician wants to be able to answer the following types of questions:

1. What is the probability of someone getting the disease in the next n years?
2. At what time in the future is someone most likely to get the disease?

The above situation directly translates to being able to make predictions on the performance of search algorithms. Input factors are algorithm parameter values, etc. The event of interest is the termination of the search because a valid solution has been found. However, since there is finite time available to perform experiments, a run of the search algorithm is terminated if a maximum number of configurations is evaluated and no valid solution has been found. In this case, the search may have found a valid solution after a larger number of configurations. Therefore, the data is right censored.

When dealing with censored data, it is easier to consider median rather than mean response values. If a sample is taken, as long as fewer than 50% of values are censored, the median is not affected by censoring. However, no accurate value for the mean of a sample is available if some values are censored. This makes it difficult to compare predictions made by models to actual values taken from samples.

The following terminology is used in survival analysis. $f(t)$ is a density function for the probability of an event occurring at a time w in the interval $(t, t + \Delta t]$.

$$f(t) = \lim_{\Delta t \to 0} \frac{1}{\Delta t} P(t < w \leq (t + \Delta t)) \tag{5.34}$$

The cumulative distribution $F(t)$ gives the probability of an event occurring any time before t, $P(w \leq t)$.

$$F(t) = \int_0^t f(t)dt = P(w \leq t) \tag{5.35}$$

The survival function gives the probability that an event happens after time t.

$$S(t) = 1 - F(t) = P(w > t) \tag{5.36}$$

$S(t)$ is a non-increasing function with $S(0) = 1$. The value of w that solves $S(w) = 0.5$ gives the median survival time [170].

The hazard function gives the probability of an event occurring in the next instant of time given that the individual has survived up to time t. It is defined as

$$\lambda(t) = \lim_{\Delta t \to 0} \frac{1}{\Delta t} P(w \leq (t + \Delta t) | w > t) \tag{5.37}$$

Whereas the survival function decreases with time, the shape of the hazard function can be much more variable. Imagine the case of the hill descent algorithm with randomised restart. Just after a restart, the probability of the search terminating might be quite low and then increase as it approaches a local optima before decreasing again at the next restart.

The hazard function can be related to the survival function as follows

$$\lambda(t) = \frac{f(t)}{S(t)} = \frac{-S'(t)}{S(t)} \tag{5.38}$$

This allows estimations of one to be derived from the other.

5.3.5.1 Cox's Proportional Hazard Model

It is not possible to use standard regression techniques to fit data to the models listed in section 5.3.1 since the data is censored. The proportional hazard model of Cox [175], designed for modelling survival data, is used instead. This model can be used to find factor values which should make the search terminate, because a valid solution has been found, as early as possible.

The Cox proportional hazard model assumes a hazard function of the following form

$$\lambda(t|\boldsymbol{x}) = \lambda_0(t) \exp(\boldsymbol{\beta}^T \boldsymbol{x}) \tag{5.39}$$

It is written as a function of time, t, for particular factor values $\boldsymbol{x}$. λ_0 is the baseline hazard function, dependent on t, which captures hazard due to factors not included in $\boldsymbol{x}$. It is assumed that the contribution to the hazard by the factors of interest is not dependent on time and that the baseline hazard λ_0 is the same for all values of $\boldsymbol{x}$. The vector $\boldsymbol{\beta}$ is the vector of regression coefficients as explained in section 5.3.1. Note that the vector $\boldsymbol{x}$ does *not* include a 1 as its first entry in this case since any intercept term is subsumed into the baseline hazard function, λ_0.

Let $\boldsymbol{x}$ and $\boldsymbol{x}'$ be two possible vectors of factor values. The ratio of their hazards

is a value independent of λ_0.

$$\frac{\lambda(t|\boldsymbol{x}')}{\lambda(t|\boldsymbol{x})} = \frac{\exp(\boldsymbol{\beta}^T\boldsymbol{x}')}{\exp(\boldsymbol{\beta}^T\boldsymbol{x})} \tag{5.40}$$

This property of the proportional hazards model is used in the estimation of the regression coefficients, $\boldsymbol{\beta}$. This is now briefly explained for the case where every uncensored run of the search within a single experiment completes at a unique time.

Let $t_{(r)}$ be the time at which experimental run r terminated and $\boldsymbol{x}_{(r)}$ be the set of factor values used for that run. Let there be k uncensored runs. Runs are ordered by increasing termination time. Since termination times are unique, $t_{(1)} < t_{(2)} < \cdots < t_{(k)}$. The at risk set $R(t_{(r)})$ gives the set of runs which are still live at time $t_{(r)}$, including the run which did in fact terminate at that time.

The data gathered from the experiment informs us that from the at risk set $R(t_{(i)})$, it was in fact individual i which terminated at time $t_{(i)}$. If a proportional hazard model represents this data then the probability of this occurring is

$$\begin{aligned} &P(i \text{ ended at } t_{(i)}|1 \text{ member of } R(t_{(i)}) \text{ ended at } t_{(i)}) \\ &= \frac{\exp(\boldsymbol{\beta}^T\boldsymbol{x}_{(i)})}{\sum_{j\in R(t_{(i)})}\exp(\boldsymbol{\beta}^T\boldsymbol{x}_{(j)})} \end{aligned} \tag{5.41}$$

To obtain the best fitting model, a $\boldsymbol{\beta}$ must be found which maximises the probability for all of the termination times being as observed. This creates a likelihood function to be maximised.

$$L(\boldsymbol{\beta}) = \prod_{i=1}^{k} \frac{\exp(\boldsymbol{\beta}^T\boldsymbol{x}_{(i)})}{\sum_{j\in R(t_{(i)})}\exp(\boldsymbol{\beta}^T\boldsymbol{x}_{(j)})} \tag{5.42}$$

The method of doing this and for handling tied termination times is left to other texts [170, 176]. However, given that this equation is the basis for model fitting to the proportional hazard model, it demonstrates two important points:

1. As intuition would suggest, the amount of data available on which to fit the data depends on the number of uncensored values
2. Equation (5.42) relies on the proportional hazard assumption to cancel out λ_0 terms and the fitted model will be less accurate if this is not the case.

All survival analysis carried out in this thesis uses the R Survival package written by Terry Therneau [177]. This includes analysis for data with tied survival times. It also provides estimations of the survival function and median survival time.

Once a model has been fitted, factor values must be found which maximise the hazard function, hence increasing the probability of the search terminating earlier. This is achieved using the LINDO optimisation software [172].

5.3.5.2 Identification Of Significant Factors

The value of the regression coefficient β_i indicates the significance of factor x_i. If a factor j does not effect the hazard then $\beta_j = 0$. The Wald statistic [170] gives a test of the null hypothesis that $\beta_j = 0$. The R survival package provides a value for this test for each j.

5.3.5.3 Safety Of Proportional Hazard Model Assumptions For Relevant Factors

The proportional hazard model assumes that any change in the value of a factor will increase (or decrease) the value of the hazard function function by the same proportion for all times t. For different values of cost function weightings, this is, perhaps, a reasonable assumption. However, for algorithm parameters such as the number of configurations evaluated between cooling and size of neighbourhood sample, this condition is unlikely to hold. Any changes which make the search more exploitative may improve the chances of finding a solution early in the search but also make the search more likely to become trapped in a local optima and so no more or even less likely to terminate in the later stages of the search. Of course, behaviour will be dependent upon the problems in the test set.

If the proportional hazard condition does not hold, estimations of median survival time are certainly likely to be inaccurate. However, it is still possible that finding values of factors which improve performance according to the model correlates with an improvement in real performance if the model is not wildly inaccurate. It should be emphasised once again, that it is good, rather than optimal, parameter values which are sought. Any predictions made by a model will be tested with further experiments. Also, attention should always be paid to the actual data obtained from the experiment where it would be expected that the best performing data point included in the experiment should not be too dissimilar to the best factor values predicted by a model.

5.4 Algorithm Configuration Experiments

This section describes a series of experiments which were conducted to configure and then evaluate the algorithm given in figure 5.1. Nine experiments were carried out. They can divided into four groups according to their purpose:

1. Algorithm configuration — Experiment 5.1 finds weightings for the subfunctions in bottom level of the hierarchy in figure 5.6. Experiment 5.2 finds weightings at the next level up. Experiment 5.3 tunes parameters specific to simulated annealing. Experiment 5.4 compares simulated annealing to random search and a tuned hill descent algorithm to decide which local search variant works best.
2. Investigate schedulability test variations — One of the main motivations given in the introduction of this chapter for using heuristic search was the ability to

Parameter	Attributes
number-of-tasks	min=max=48
period	min=10^4, max=10^7, granularity=10^4
utilisation-per-processor	min=max=0.6
distribution	type=beta, alpha=0.7
utilisation-per-network	min=max=0.5
distribution	type=beta, alpha=0.7
tasks-per-transaction	min=max=8
transaction-length	min=max=0.6
messages-per-task	min=max=1.5
transaction-utilisation-distribution	equal=true
tasks-per-processor	min=max=8
number-of-networks	min=max=2
processor-connectivity	min=max=1
network-bandwidth	min=max=1
network-latency	min=max=0
processor-network-bandwidth	min=max=1024
processor-network-latency	min=max=0

Table 5.3: Gentap parameters for problem generation in initial configuration experiments

easily change between schedulability tests. Experiment 5.5 tests whether this is possible.

3. Investigate problem characteristics variations — The experiments in this group both investigate the effectiveness of the cost subfunctions for problems with different characteristics and also the relationship between certain characteristics and problem difficulty. Experiments 5.6 and 5.7 find the best way of retuning weightings at each level of the cost subfunction hierarchy for selected problem characteristic combinations.
4. Check algorithm robustness and scalability — Experiment 5.8 checks whether performance of the algorithm is consistent across a set of problems with the same characteristics. Experiment 5.9 increases problem sizes to find the limits of the algorithm.

For experiments 5.1 to 5.4, 50 problems were generated, all with the same controllable characteristics. These characteristics are described by the parameter values given to the Gentap task allocation problem generator described in chapter 4. The parameter values, shown in table 5.3, were chosen so that the problems were non-trivial but within bounds that would be unlikely to generate unsolvable problems.

5.4.1 Experiment 5.1 — Guidance Heuristic Weightings

The first experiment was designed to find some good weightings for the cost subfunctions linked to the guidance heuristic. These are the functions labelled $g_2, \ldots, g_9$ in section 5.2.3.1. The search algorithm was configured to perform simulated annealing with inittemp $= 0.05$, ssize $= 1$, maxinner $= 3000$, randrestart $= -1$. A reminder of the meaning of each parameter is given in table 5.4. w_{sched} and $w_{guidance}$ were

Parameter	Description
randrestart	Number of loop iterations before restarting at random solution. -1 implies never restart
ssize	Number of solutions sampled from neighbourhood per iteration
inittemp	Initial temperature used for simulated annealing
maxinner	Number of loop iterations between temperature reductions

Table 5.4: Parameters for search algorithm in figure 5.1

both set to 100 so that the guidance heuristic and schedulability constraints were equally weighted. These values were chosen based on experience so that the search would not be over exploitative and be likely to become trapped in local optima. Subsequent experiments endeavour to modify these values to improve performance further. The response used to measure performance is the number of configurations evaluated before a solution meeting all schedulability constraints is found.

Values for the weightings $w_2, \ldots, w_9$ were generated according to a four level mixture design for 8 factors. Weighting values are set to integers for convenience with $\sum_{i=2}^{9} w_i = 90000$. The mixture was constrained so that the minimum value of any weighting was 3. If 0 was used it would mean that the cost subfunction was effectively removed and would no longer provide guidance to the search. The behaviour at the edge of the factor value design space may be quite different to that predicted by behavioural trends of interior points and therefore need to be modelled separately. Elimination of cost subfunctions allows an overall cost value to be computed more quickly. This introduces the possibility that by removing some subfunctions, a worse performing search in terms of solutions evaluated may find a solution faster because of a reduction in time per evaluation. This is another reason that 0 valued weightings should be investigated separately. It is not done in these experiments.

The mixture design for 8 factors at 4 levels requires 120 runs. A blocked design [169] was used so that the 120 factor value combinations were run for the same 50 problems requiring 6000 runs in total.

The initial data analysis was based on studying the raw data rather than a statistical analysis. The 120 weighting combinations were labelled param000, ..., param119. The frequency with which a particular parameter combination gave the best response for a problem is shown in figure 5.7. Only 7 of the 120 weighting combinations were ever the best performing. According to the formula of Korwar and Serfling [178], 41 unique combinations would be expected if 50 random samples were taken from a set of 120 with replacement. This strongly suggests that a change in weightings has an effect on the search performance.

The leading weightings combination was param116 which was best for 22 of the 50 problems with param082 the second most frequent best combination.

Figure 5.8 shows how many times a weighting combination was any one of the

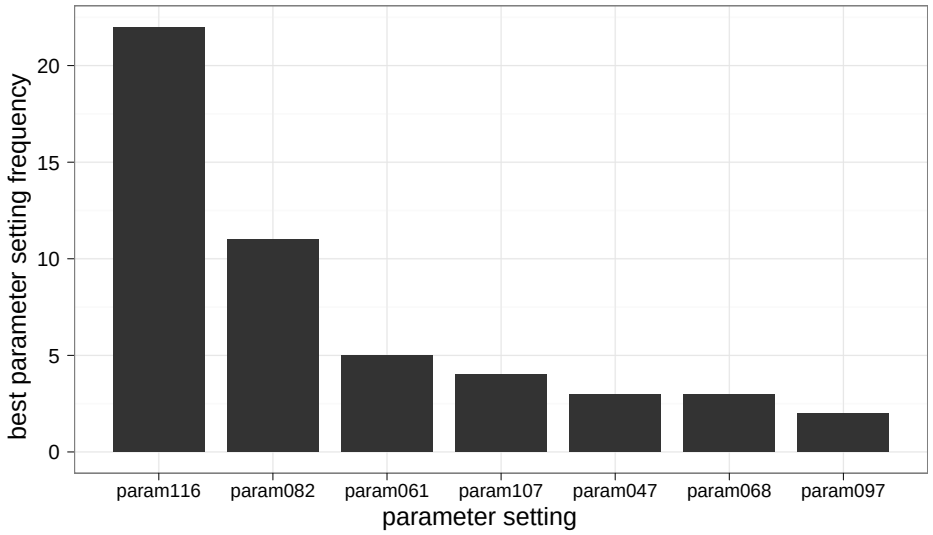


Figure 5.7: Frequency of parameter combination giving best median performance

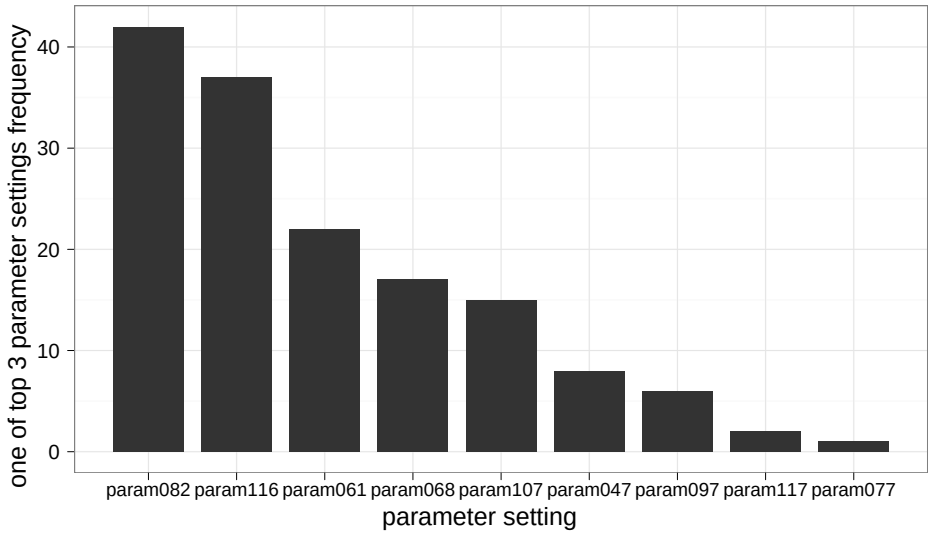


Figure 5.8: Frequency of parameter combination giving top 3 median performance

Name	Weight Values $w_2, \ldots, w_9$	Median	I.Q.R.	Censored	Predicted Median
param082	3, 3, 29995, 59887, 3, 3, 3, 3	41516.5	21065.25	0/50	78624.0
param116	3, 3, 3, 89979, 3, 3, 3, 3	42040.0	33057.25	7/50	148420.0
param061	3, 29995, 3, 59987, 3, 3, 3, 3	56109.5	46452.75	3/50	137401.0
param068	3, 3, 59987, 29995, 3, 3, 3, 3	59453.0	23762.75	0/50	63615.0
param107	3, 3, 3, 59987, 3, 29995, 3, 3	62223.0	19812.25	0/50	80044.0
param097	3, 3, 3, 59987, 29995, 3, 3, 3	72203.0	27129.75	0/50	130582.0
tuned	3, 3, 40849, 37881, 3, 11255, 3, 3	64357.0	24256.00	0/50	58002.0

Table 5.5: Best performing weightings by median and tuned weightings

best three combinations for a problem. In this graph, the order of param082 and param116 is reversed with param082 being one of the best 3 parameter combinations for 42 of the 50 problems. This shows that it is fairly robust to the differences in problems due to uncontrolled characteristics.

Table 5.5 shows the top 6 weighting combinations according to median response. The interquartile range and number of censored responses are also shown to give an idea of the robustness of each weighting combination. Recall that a censored response is one where the search terminated after the maximum number of evaluations ($5 \cdot 10^5$ in this case) without finding a valid solution. param082 is as good or better than other weighting combinations on measures of performance and robustness. It has high values for weightings w_4 and w_5 which are related to the cost subfunctions for grouping communicating tasks.

A Cox's proportional hazard model was fitted to all of the data and the weightings which maximised the hazard were also tested by running them on all 50 problems. Other algorithm parameters kept the same values as in the experiment which collected data for the model. These weightings are labelled tuned in table 5.5. The results utilised the grouping subfunctions and also brought in an element of the load balancing subfunction. The result of using these weightings are good but did not outperform the best performing data points from the experiment demonstrating the imperfection fitted model.

The survival curve predicted by the model is shown in figure 5.9. It predicts that the probability of a search lasting longer than $1.5 \cdot 10^5$ is close to 0. The predicted median survival time is 58002 evaluations which is less than the tested one of 64357. Table 5.5 also lists the median predicted by the model for each weighting combination.

A boxplot of results from weighting combinations in table 5.5 is shown in figure 5.10. The extents of each box show the interquartile range and the median is marked inside each box. The lines protruding from the boxes show the range of values up to 1.5 times the I.Q.R. away from the median. Any points outside of this range are outliers, marked with a dot. The plot clearly shows how param116 leads to an aggressive but fragile heuristic working very well for some problems but poorly on

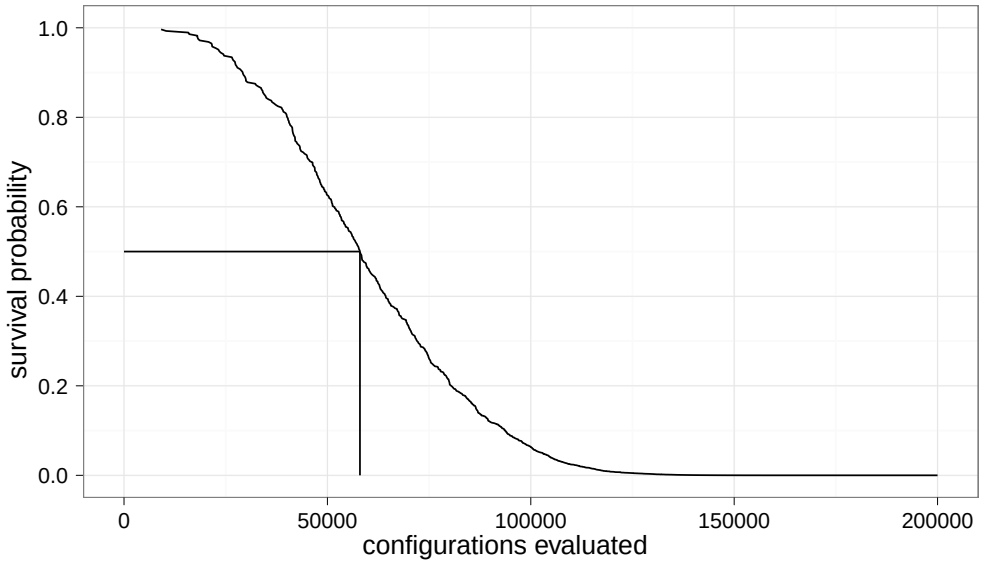


Figure 5.9: Survival curve at predicted best weighting combination

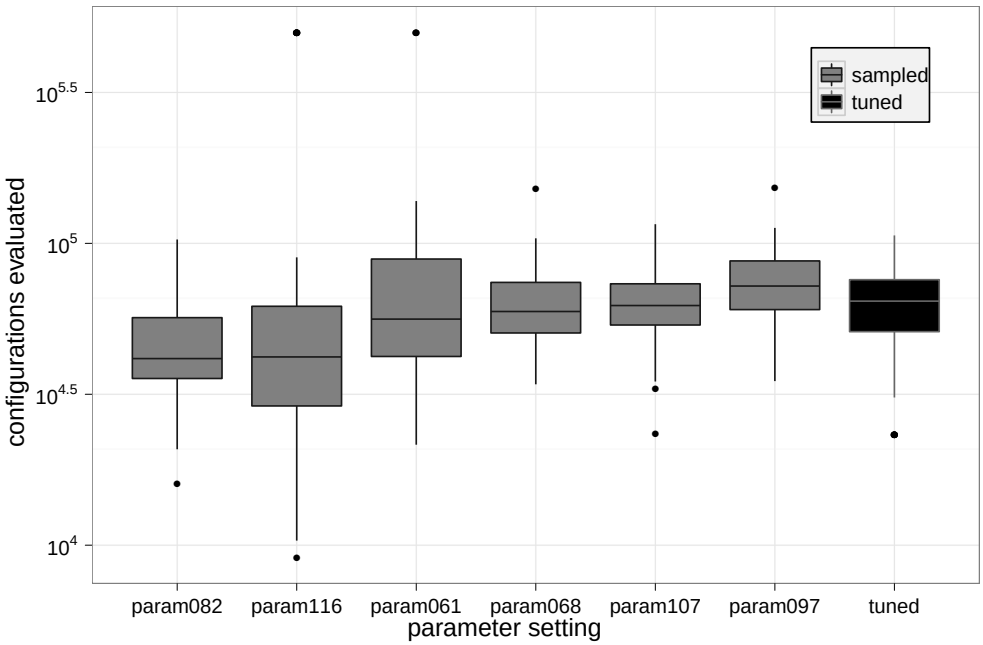


Figure 5.10: Box plot of best performing sampled weightings and tuned weightings

others. The tuned parameters appear robust with a narrow range of response values but are dominated by param082. A paired Wilcox test rejects the null hypothesis that the mean response from param082 weightings is the same as tuned weightings at the 99% confidence level. The p-value returned is approximately $2 \cdot 10^{-5}$. A further experiment was conducted which evaluated both the tuned weightings and param082 weightings over the 50 problems for 3 repetitions resulting in 150 runs of each weighting combination. This confirmed the initial result that the performance improvement given by param082 is statistically significant at the 1% level. In this latter experiment the median for param082 was 43944.5 and 60565 for the tuned weightings with similar interquartile ranges as before. The similarity to previous results with these weightings goes some way to demonstrate that differences due to the pseudo-random number generator seed do not greatly change the performance of the algorithm.

This experiment has shown that a response surface methodology with a Cox's proportional hazard model is an effective way of obtaining good cost function weightings but at the same time has shown that better weighting combinations may exist due to imperfections in the model. A large number of factors requires a model to be fitted to high dimensional data and limitations in the shape of a quadratic mixture model will not be a perfect fit.

However, there are a number of reasons which suggest that the method is effective. Firstly, the tuned weighting combination is comparable or better than nearly all sampled design points in the original experiment. Only one is significantly better.

Simply selecting the best design point from the experiment is not a recommended strategy since each run only has at most 3 above-minimum weighting values whereas the model uses information across all of the runs. A review of the fitted regression coefficients (values contained in $\boldsymbol{\beta}$ in equation (5.42)) gives evidence that the model has captured some of the algorithm's behaviour. The two most significant factors correspond to weightings for the two grouping factors with w_4 having a higher coefficient ($\beta_{w_4} = 1.27 \cdot 10^{-4}$) than w_5 ($\beta_{w_5} = 7.23 \cdot 10^{-5}$). The reason that this isn't made evident in the tuned weighting combination is due to the effects of interactions between factors. If a model without interaction effects is used, then the weighting combination suggested by the model is in fact identical to param116 which is highly performant but not robust. The interaction model has been successful in recognising some of the poorer runs from using param116 and similar weighting combinations and produced weightings which have both good performance and robustness characteristics.

5.4.2 Experiment 5.2 — Schedulability Versus Guidance

This experiment is concerned with finding good values for w_{sched} and $w_{guidance}$ which balance the effect of the guidance heuristic with penalising the search for unsatisfied schedulability constraints. It is a continuation of experiment 5.1 and uses the same

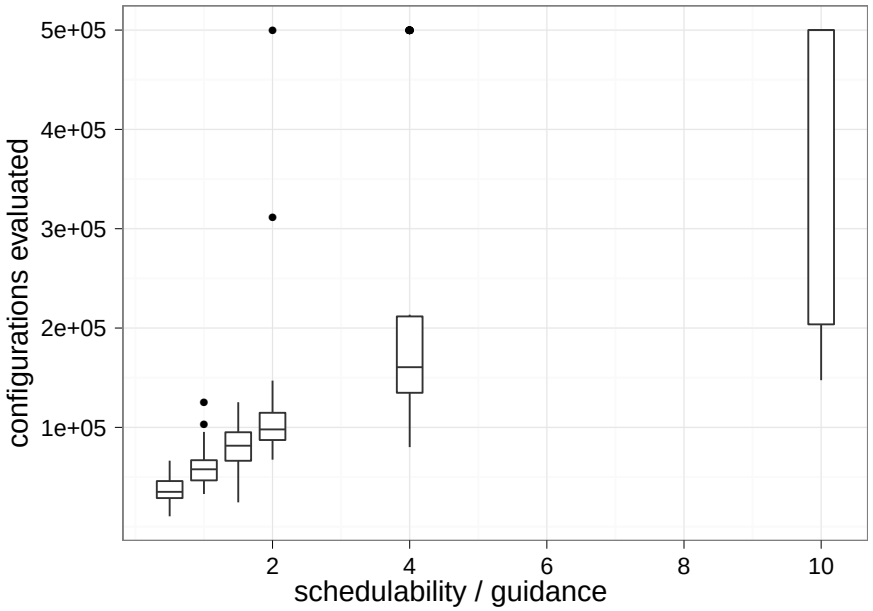


Figure 5.11: Results of screening experiment changing balance between schedulability penalty and guidance heuristic

set of problems as a test set. The guidance subfunction weightings are set to the tuned weightings combination found in experiment 5.1. Since there are only two weightings at the second level of the weightings hierarchy in figure 5.6, it can be treated as a single factor; $w_{guidance}$ is fixed at 100 and w_{sched} is varied. The chosen factor of interest is in fact $\frac{w_{sched}}{w_{guidance}}$. Within this section b will represent this factor which balances the guidance heuristic against schedulability constraints.

The first set of runs for experiment 5.2 varied b between 0.5 and 10. A bias was placed towards favouring schedulability over the guidance heuristic since it was expected that the heuristic would be imperfect and weighting it too highly would lead to poor performance. The results in figure 5.11 show that this is not in fact the case. The performance continued to increase for values of b as low as 0.5, the smallest value tested, showing the heuristic to work better than expected.

Since it was still assumed that the guidance heuristic was imperfect, this first set of runs was treated as a screening experiment and further runs with values of b as low as 0.05 were used to supplement the data. Table 5.6 shows the results from runs with b set between 0.05 and 2. It is not until the value drops below 0.3 that performance begins to fall off. This shows that the guidance heuristic tuned in the previous experiment is very well suited to this particular set of test problems.

The data is also plotted in figure 5.12 with box plots showing the experimental data and the line showing the predicted median response from fitting a Cox's proportional hazard model. With only a single factor and a large amount of data (7 factor levels over 50 problems), the model looks to be a good fit to the data. The predicted best value for b is close to 0.6 and so 60 was chosen as the value for w_{sched}

$w_{sched}/w_{guidance}$	Median	I.Q.R.	Censored
0.05	71558.5	46839.75	0/50
0.15	42566.5	22128.75	0/50
0.30	35046.5	23190.75	0/50
0.50	35076.5	17127.00	0/50
1.00	57781.0	57781.00	0/50
1.50	81538.0	81538.00	0/50
2.00	97946.5	97946.00	1/50

Table 5.6: Results of experiment to tune balance between constraints and guidance heuristic

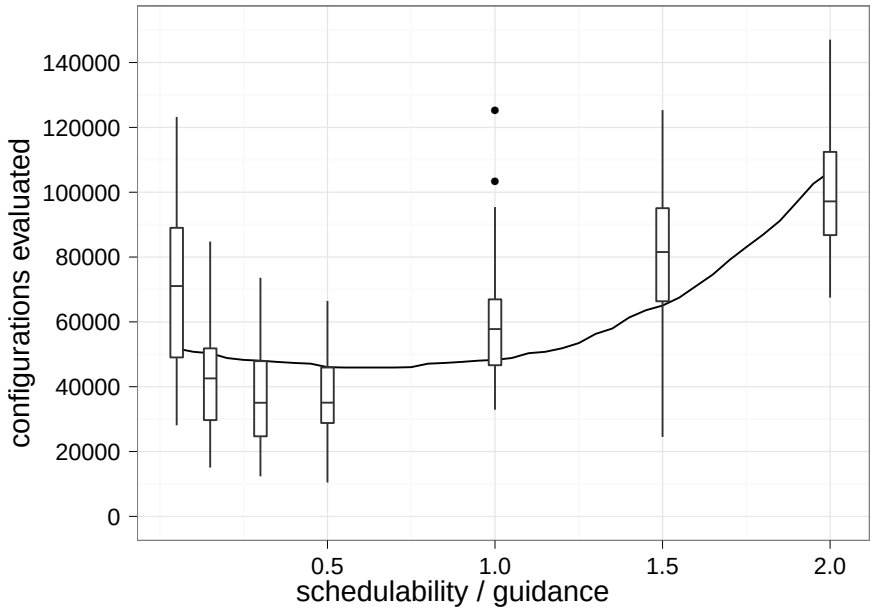


Figure 5.12: Box plot schedulability / guidance balance experiment data with median response predicted by model

in the next experiment.

5.4.3 Experiment 5.3 — Simulated Annealing Parameters

Experiment 5.1 and experiment 5.2 have found weightings for the cost function which already enable simulated annealing to solve task allocation problems in the test set with adequate efficiency. In these previous experiments, algorithm parameters (as opposed to weighting values) were chosen based on experience with other task allocation problems. Experiment 5.3 tries to further improve performance of the simulated annealing algorithm by tuning algorithm parameters specifically for the problem test set and the weighting values found in previous experiments.

There are three factors, maxinner, ssize and inittemp. Refer back to figure 5.1 and table 5.4 for an explanation of these parameters. A 3 level full factorial experimental design was used. The values of ssize used were 1, 4 and 8. Once again, the censoring level was set at $5 \cdot 10^5$ evaluations. Note that this implies fewer search loop iterations

maxinner	inittemp	ssize	Median	I.Q.R.	Censored
1000	0.005	1	24202.5	8344.00	1/50
1000	0.005	4	85782.0	*	24/50
1000	0.010	1	93672.5	11210.75	0/50
1000	0.010	4	28006.0	54985.00	4/50
3000	0.005	1	40660.5	20623.25	0/50
3000	0.005	4	443684.0	*	24/50
3000	0.010	1	246447.5	25021.50	0/50
3000	0.010	4	31372.0	49827.00	1/50
6000	0.005	1	54764.0	36449.75	0/50
6000	0.005	4	140804.0	*	16/50
6000	0.010	1	462828.5	37918.00	10/50
6000	0.010	4	27632.0	49883.00	1/50

Table 5.7: Data from screening experiment to tune simulated annealing parameters

for ssize > 1.

With ssize=8, all combinations of the other two factors had more than half the runs censored resulting in a median at the censoring level of $5 \cdot 10^5$. This was also the case for all runs with inittemp=0.002 which was tried at 0.002, 0.005 and 0.010.

The data for the remaining factor levels are shown in table 5.7. No interquartile range values are given where the entire upper quartile is censored. The data show a large interaction effect between inittemp and ssize. For each maxinner level, the median responses when inittemp and ssize are either both high or both low are better than when they oppose each other. A low value of inittemp and high value of ssize make the search too exploitative and the reverse is over explorative.

The best design point on all criteria is the first one with maxinner $= 1000$, inittemp $= 0.005$, ssize $= 1$. There does not seem to be any advantage in using both temperature and sample size to control exploitation versus exploration.

Further experiments were conducted in the region of this good design point with sample size fixed at 1. Four levels of each factor were used with maxinner ranging between 500 and 2000 and inittemp ranging between 0.003 and 0.009.

The results of this experiment are shown in table 5.8 and a Cox's proportional hazard model fitted to the data is displayed in figure 5.13. The model appears to be more accurate for high values of inittemp and somewhat pessimistic at lower values. This is due to the points with a initial temperature of 0.003 having many censored points though still maintaining a low median response. This is especially true for the point at $(500, 0.003)$. This is a highly exploitative search, the low median showing many problems were solved quickly, but when a solution was not found early on in the search it was unable to escape from local optima.

A minimum point was found on the fitted surface at maxinner $= 1906$, inittemp $= 0.0035$. Running a test with these parameters on the 50 problems gave a median response of 14730 and interquartile range of 9947.25 with a single censored value. This is a substantial improvement on the median response value of 58002 found before experiments 5.2 and 5.3 tuned values to w_{sched} and the simulated annealing

maxinner	inittemp	Median	I.Q.R.	Censored
500	0.0030	18512.0	*	22/50
500	0.0050	18283.5	6716.75	1/50
500	0.0075	40463.5	9950.00	5/50
500	0.0100	51732.0	7318.25	1/50
1000	0.0030	15114.0	19301.50	9/50
1000	0.0050	25466.5	8135.75	0/50
1000	0.0075	64230.0	6799.50	1/50
1000	0.0100	94980.0	9128.25	0/50
1500	0.0030	12730.0	14494.00	6/50
1500	0.0050	31295.5	14122.25	0/50
1500	0.0075	88649.0	8529.00	0/50
1500	0.0100	131213.5	13098.75	0/50
2000	0.0030	13600.0	16401.75	7/50
2000	0.0050	36555.5	10117.25	0/50
2000	0.0075	112493.5	15988.75	0/50
2000	0.0100	168108.5	12606.75	0/50

Table 5.8: Data from experiment to refine simulated annealing parameters

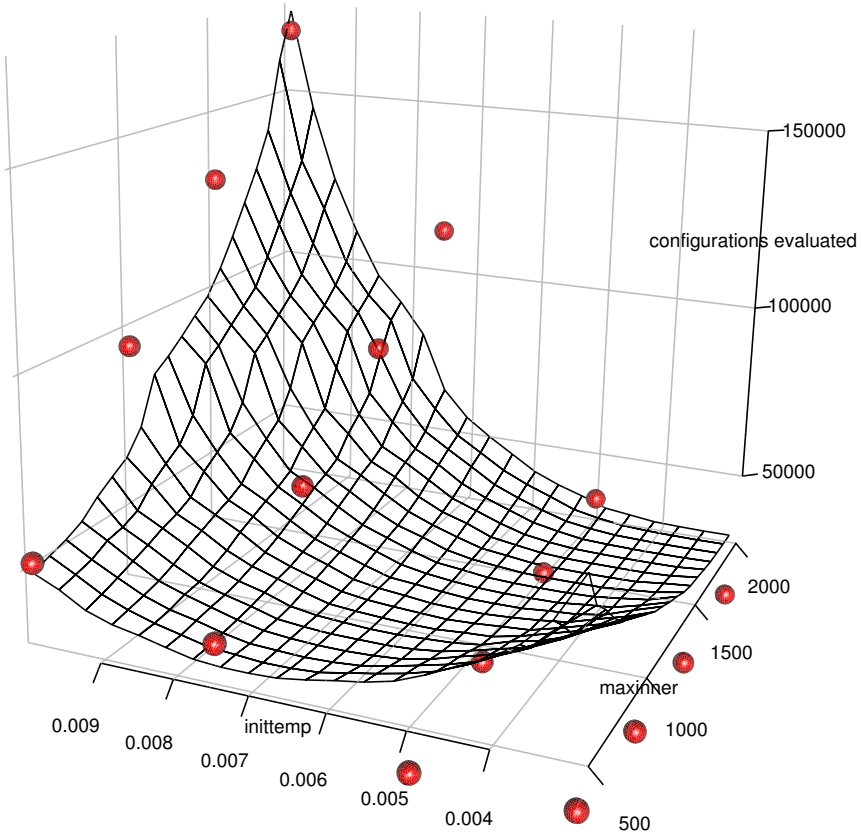


Figure 5.13: Model fitted from experiment to refine simulated annealing parameters

parameters.

Repeated tests with these tuned parameters always resulted in 1 or 2 censored values. Moving from this tuned design point along the diagonal valley of the surface in figure 5.13 reaches the tested design point at $(1500, 0.005)$ with a median response of 31295.5. This point was found to be more reliable in terms of solving all 50 problems within the censoring limit at the expense of some performance.

5.4.4 Experiment 5.4 — Algorithm Variations

Experiment 5.4 tested the other search algorithm variants implemented by the algorithm in figure 5.1 so that they could be compared with the tuned simulated annealing algorithm.

By selecting randrestart=0, the algorithm performs a random search, generating a new solution on every iteration according to the method in section 5.2.2 for generating a random solution. This type of search has the advantage of being simple. It is undirected so there is no benefit gained from taking the extra time to calculate guidance heuristics making it more efficient per configuration evaluated. However, within a censoring limit of $5 \cdot 10^5$ evaluations, the search failed to find one valid solution for any of the 50 problems. Therefore further experiments with random search were abandoned.

The alternative form of guided local search which is implemented is hill descent with random restart. To be fairly compared to simulated annealing, it was necessary to first tune the algorithm's parameters. It is assumed that the weightings found previously for simulated annealing are also good for hill descent since it is a similar local search algorithm and the neighbourhood definition is also unchanged.

It is not possible to exhaustively evaluate all configurations in a neighbourhood to detect a local optimum. The randrestart parameter sets the number of configurations which can be evaluated without any improvement to the best solution before the search restarts from a new random solution. This is therefore acting as a heuristic for detecting local optima. If randrestart is set too high then the search wastes time in a local optima when it could be exploring a new area of the solution space. If it is set too low then the search will restart too soon, possibly missing good solutions.

The first set of runs used a 3 level full factorial experimental design with ssize set to 1, 4 and 8 and randrestart set to 1000, 3000 and 5000. There was a continuous pattern of improvement towards lower values of randrestart. Given the previous explanation, it was expected that there would be a low level of randrestart where performance would begin to worsen. Further runs of the algorithm were conducted with randrestart set at 750, 500 and 250 with the same ssize levels as before. The results for all factor combinations are shown in table 5.9.

Viewing the summary statistics in table 5.9, no obvious pattern emerges. With ssize set to 1, then the behaviour with relation to randrestart is as expected. Perfor-

ssize	randrestart	Median	I.Q.R.	Censored
1	250	169147.0	237283.00	5/50
1	500	71110.5	79170.75	0/50
1	750	52242.5	79581.75	1/50
1	1000	81872.0	115153.75	0/50
1	3000	73307.0	176858.00	2/50
1	5000	87324.5	220277.00	3/50
4	250	75638.0	97700.50	1/50
4	500	98348.0	155899.75	2/50
4	750	82910.0	112580.75	2/50
4	1000	143350.0	229969.75	3/50
4	3000	266107.5	393101.25	12/50
4	5000	360691.0	*	17/50
8	250	105072.0	138597.75	0/50
8	500	97225.0	176015.25	3/50
8	750	128844.5	288461.00	2/50
8	1000	106956.0	233376.75	5/50
8	3000	271893.5	394872.75	13/50
8	5000	460138.0	*	24/50

Table 5.9: Data generated for investigation of hill descent with random restart

mance is best for randrestart set to 750 but degrades either side of this value.

For larger sample sizes, this pattern does not emerge. Given that more solutions are being evaluated per iteration, it is perhaps the case that fewer iterations are required to detect a local optimum. Therefore, performance may improve further for even lower values of randrestart at larger sample sizes though this has not been tested.

The median response values in table 5.9 are competitive with the previous result for simulated annealing but the interquartile range values are much higher. This suggests that while simple hill descent with restarts can perform well, it is more fragile to differences between problems and / or the choice of random number generator initialisation seeds.

An attempt was made to a fit a model to the data. This is shown in figure 5.14. Due to the large amount of variance at each factor combination, it is difficult to fit a model that captures the behaviour. It can be seen that the median data points do not lie near the surface of predicted values.

While it may be possible to achieve better results with higher values of ssize and lower values of randrestart, the algorithm was rejected at this stage because its performance is less predictable than simulated annealing.

5.4.5 Experiment 5.5 — System Model Variations And Time Required

The aim of experiment 5.5 was to test the simulated annealing algorithm tuned in experiments 5.1 to 5.3 on systems using different scheduling models and schedulability tests. The weightings were set to those labelled tuned in table 5.5, ssize was set to 1, inittemp to 0.005 and maxinner to 1500. w_{sched} was set to 60 and $w_{guidance}$ to 100.

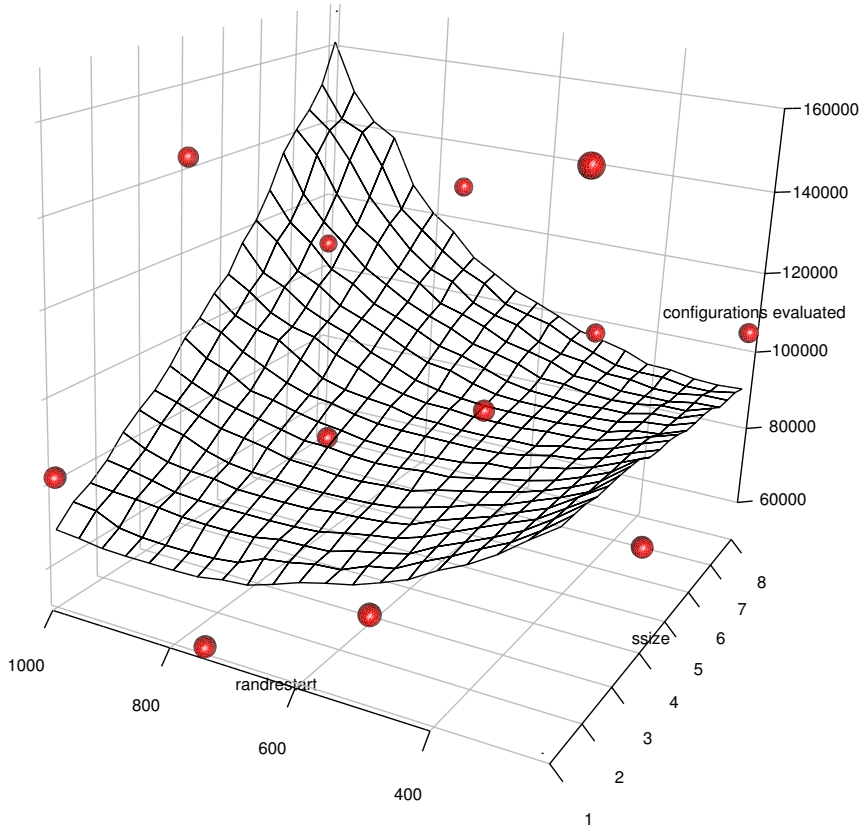


Figure 5.14: Surface fitted to responses from hill descent with random restart algorithm

The algorithm was run once again on the same set of test problems which specified WCDO as a schedulability test. All problems were solved with no censored values. The median number of configurations evaluated was 30820 and the mean was 30567. This median value matches closely with the value of 31295.5 obtained at the end of experiment 5.3 supplementing the evidence that the performance is robust to changes in random number seeds. Over all the problems, the algorithm evaluated 491 evaluations per second on average. This equates to 63 seconds at the median response value.

WCDOPS++ is a less pessimistic but more compute intensive form of the WCDO analysis. They were previously compared in sections 2.2.3.6 and 4.2.3. The problem test set was replicated except that the schedulability test was changed from WCDO to WCDOPS++. The algorithm was run on this replicated test set with identical parameters. All 50 problems were solved with a median response of 23236.5 and mean of 23048. The mean rate of configurations evaluated was 37.406 evaluations per second. The is equivalent to 621 seconds at the median response.

These results clearly show the effects of changing the schedulability test. Using the less pessimistic test, the algorithm required approximately 10 times longer to find a valid solution even though fewer configurations needed to be evaluated. The more pessimistic analysis is therefore preferable purely on the basis of the performance of

the simulated annealing algorithm.

There are, however, good reasons why an engineer would prefer to use a less pessimistic test. It could be the case that no schedulable solution can be found with the WCDO test even when one exists which would be detected by WCDOPS++. Furthermore, a more accurate schedulability test may allow a solution to be found on a hardware platform with fewer processors saving costs and reducing energy usage. WCDO is still the preferred choice throughout this work.

The final set of runs in this experiment tested the algorithm on problems using a schedulability test based on the construction of a static cyclic schedule. The range of period values in the test set used up to this point has a theoretical maximum LCM of at least $9.99 \cdot 10^9$. This value comes from the data in table 5.3 and equation (4.3). Even though the LCMs of period values in the actual test set are below this, constructing schedules whose length is this order of magnitude is not feasible within a search algorithm which will need to evaluate many thousands of solutions. For this reason, a new test set was generated. The parameters provided to Gentap were the same as those in table 5.3 except period was given attributes min $= 2.5 \cdot 10^5$, max $= 10^6$ and granularity $= 2.5 \cdot 10^5$.

The algorithm was run on this newly generated set of problems using static cyclic scheduling. 42 of 50 problems were solved within the censoring limit. The median number of configurations evaluated was 42961.5. This was done at a rate of 313 evaluations per second.

Experiment 5.5 has shown the flexibility of using a metaheuristic search based approach to task allocation. If a new scheduling mechanism is used and a schedulability test implementation is available then the same algorithm can be applied with limited additional effort. For this experiment, the algorithm was not reconfigured for each specific test and doing so may avail better performance.

The efficiency of a schedulability test is key to the overall performance of the algorithm as was demonstrated by the comparison between WCDO and WCDOPS++. A very complex or inefficiently implemented schedulability test is likely to push the boundaries of what is considered acceptable performance.

The time taken to execute a test will not only depend on the test but also the system it is being applied to. It is expected that larger systems will not only require more configurations to be evaluated before a valid solution is found but the time to evaluate each configuration will also increase. Experiments 5.6 to 5.9 look at the effects problem characteristics have on algorithm performance.

5.4.6 Experiment 5.6 — Problem Characteristic Investigation

The aim of experiment 5.6 is to relate the performance of the simulated annealing search algorithm to different task allocation problem characteristics. The configurable weights in the guidance heuristic are intended give the algorithm flexibility

Parameter	Attributes
number-of-tasks	min=max=48
period	min=10^4, max=10^7, granularity=10^4
utilisation-per-processor	min=max=0.5
messages-per-task	min=max=1.5
transaction-utilisation-distribution	equal=true
number-of-networks	min=max=2
network-bandwidth	min=max=1
network-latency	min=max=0
processor-network-bandwidth	min=max=1024
processor-network-latency	min=max=0

Table 5.10: Problem characteristics kept constant throughout experiment 5.6

Parameter	High Value	Low Value	Factor Label
tasks-per-processor	min=max=12	min=max=6	x1
processor-connectivity	min=max=1	min=max=0.5	x2
utilisation-per-network	min=max=0.90	min=max=0.45	x3
tasks-per-transaction	min=max=12	min=max=6	x4
transaction-length	min=max=0.6	min=max=0.3	x5

Table 5.11: Problem characteristics investigated in experiment 5.6

for problems with different characteristics. In the second part of the experiment, the problems are classified according to the most dominant characteristics and parameters are tuned for each class of problem. The hypothesis is that tuning the algorithm separately for smaller classes of problem should increase overall performance.

Tables 5.10 and 5.11 list the Gentap parameter values used for these experiments. Table 5.10 lists the problem characteristics whose values were kept constant throughout the investigation. Table 5.11 lists characteristics which were varied between two levels in the experiment. Refer to section 4.4.7 for an explanation of how the utilisation-per-network parameter is translated into message sizes. The algorithm for generating a network topology from the processor-connectivity parameter was given in section 4.4.3.

A concious decision was made not to vary the number-of-tasks and utilisation-per-processor for this experiment but rather investigate the range of difficulty problems of the same size with the same per-processor utilisation can present. The limitations of the search algorithm with regards to problem size are studied in experiment 5.9. It was not possible to vary all parameters simultaneously due to the limited resources available for each experiment. Five problem characteristic parameters were varied between two levels, creating $2^5 = 32$ classes of problem. Three instances of each class of problem were generated so the test set included 96 problems in total.

An 8 factor, 4 level mixture design could have been used for setting the guidance heuristic weightings as in experiment 5.1. However the $120 * 96 = 11520$ runs required were judged to be too many to be completed in a reasonable amount of time. A compromise was made to use a 3 level mixture design with additional interior

Name	Weight Values $w_2, \ldots, w_9$	Median	I.Q.R.	Censored	Predicted Median
param043	4283, 4283, 4283, 56238, 4283, 4283, 4283, 4284	28685.0	48383.75	21/96	109624.0
param024	3, 3, 3, 44991, 44991, 3, 3, 3	36497.0	51723.50	19/96	75855.0
param028	3, 3, 3, 44991, 3, 44991, 3, 3	51660.0	122124.25	22/96	94173.0
param015	3, 3, 89979, 3, 3, 3, 3, 3	54021.5	320970.00	26/96	113361.0
param039	4283, 4283, 56238, 4283, 4283, 4283, 4283, 4284	55190.5	103565.00	23/96	109038.0
param013	3, 44991, 3, 44991, 3, 3, 3, 3	55581.0	322196.75	31/96	118577.0
group0	3, 3, 3, 38894, 33482, 4067, 13545, 3	37889.0	40140.75	16/96	72873.0

Table 5.12: Best performing weightings by median and tuned weightings

design points. A regular 3 level mixture experimental design of the type described in section 5.3.3 was used for the first 36 runs for each problem. To this, a centroid design point was added. Further design points were then generated as follows. For q factors, a possible mixture is to have one factor level at $\frac{1}{q} + \frac{1}{2}$ and the other factors set to $\frac{1}{q} - \frac{1}{2(q-1)}$. The matrix below shows these extra design points for 8 factors along with the centroid point.

$$\begin{array}{cccccccc} w_2 & w_3 & w_4 & w_5 & w_6 & w_7 & w_8 & w_9 \\ \frac{5}{8} & \frac{3}{56} & \frac{3}{56} & \frac{3}{56} & \frac{3}{56} & \frac{3}{56} & \frac{3}{56} & \frac{3}{56} \\ \frac{3}{56} & \frac{5}{8} & \frac{3}{56} & \frac{3}{56} & \frac{3}{56} & \frac{3}{56} & \frac{3}{56} & \frac{3}{56} \\ \frac{3}{56} & \frac{3}{56} & \frac{5}{8} & \frac{3}{56} & \frac{3}{56} & \frac{3}{56} & \frac{3}{56} & \frac{3}{56} \\ \frac{3}{56} & \frac{3}{56} & \frac{3}{56} & \frac{5}{8} & \frac{3}{56} & \frac{3}{56} & \frac{3}{56} & \frac{3}{56} \\ \frac{3}{56} & \frac{3}{56} & \frac{3}{56} & \frac{3}{56} & \frac{5}{8} & \frac{3}{56} & \frac{3}{56} & \frac{3}{56} \\ \frac{3}{56} & \frac{3}{56} & \frac{3}{56} & \frac{3}{56} & \frac{3}{56} & \frac{5}{8} & \frac{3}{56} & \frac{3}{56} \\ \frac{3}{56} & \frac{3}{56} & \frac{3}{56} & \frac{3}{56} & \frac{3}{56} & \frac{3}{56} & \frac{5}{8} & \frac{3}{56} \\ \frac{3}{56} & \frac{3}{56} & \frac{3}{56} & \frac{3}{56} & \frac{3}{56} & \frac{3}{56} & \frac{3}{56} & \frac{5}{8} \\ \frac{1}{8} & \frac{1}{8} & \frac{1}{8} & \frac{1}{8} & \frac{1}{8} & \frac{1}{8} & \frac{1}{8} & \frac{1}{8} \end{array} \tag{5.43}$$

These 9 design points along with the previous 36 result in a total of $45 * 96 = 4320$ runs in total.

Table 5.12 lists the best design points from the experimental runs and an additional point generated from maximising the hazard based on a fitted Cox's proportional hazard model. For this experiment, this point is labelled group0. The quality of the model is consistent with that obtained in experiment 5.1; the best couple of design points have a lower median than the group0 design point but this latter point appears more robust with a lower interquartile range and fewer censored results. Actual medians were once again lower than those predicted by the model.

The weighting combination labelled group0 in table 5.12 was obtained by fitting a model to the entire set of problems which had characteristic variations as listed in table 5.11. The next step of this experiment divided problems into smaller groups, know as problem classes, according to their characteristics. The aim was to improve performance by finding weightings which are a good fit for certain characteristics.

factor	coef	z	p
x1	0.44121	3.9446	8.0e-05
x2	0.38951	3.5929	3.3e-04
x3	-0.14543	-1.3176	1.9e-01
x4	-1.13087	-8.9173	0.0e+00
x5	-0.60615	-5.1608	2.5e-07
x1:x2	-0.12162	-0.8837	3.8e-01
x1:x3	-0.11073	-0.7904	4.3e-01
x1:x4	0.44212	2.8961	3.8e-03
x1:x5	0.03653	0.2500	8.0e-01
x2:x3	-0.19725	-1.4953	1.3e-01
x2:x4	0.35177	2.3591	1.8e-02
x2:x5	-0.00380	-0.0267	9.8e-01
x3:x4	0.17192	1.1255	2.6e-01
x3:x5	0.17714	1.2250	2.2e-01
x4:x5	-1.82425	-8.9311	0.0e+00
x1:x2:x3	0.33862	2.3140	2.1e-02
x1:x2:x4	0.04626	0.2837	7.8e-01
x1:x2:x5	-0.18052	-1.1393	2.5e-01
x1:x3:x4	-0.28687	-1.7640	7.8e-02
x1:x3:x5	-0.13563	-0.8564	3.9e-01
x1:x4:x5	2.06867	11.1710	0.0e+00
x2:x3:x4	-0.09313	-0.6268	5.3e-01
x2:x3:x5	0.00426	0.0293	9.8e-01
x2:x4:x5	-0.31933	-1.9695	4.9e-02
x3:x4:x5	0.29784	1.8393	6.6e-02

Table 5.13: Output from R `coxph` function fitting model with problem characteristic factors

The issue of how to create problem classes from combinations of characteristic values is now addressed. The issue of which characteristics to use and how many are dealt with separately in the next two sections.

5.4.6.1 Choosing Characteristics For Problem Classes

The data obtained in the previous set of runs can also be viewed as performing a full factorial over the five problem characteristics listed in table 5.11. In order to decide how to classify the problems, an interaction model was fitted to the data where problem characteristics were used as factors rather than weightings.

Table 5.13 lists partial output from the R `coxph` function used with the problem characteristic interaction model. The first column labels match the factor labels in table 5.11. `x:y` is the term for the interaction between `x` and `y`. The remaining table columns are the regression coefficients, the Wald statistic, a measure of factor significance, and a p-value. The p-value is related to the null hypothesis that the regression coefficient for that factor is in fact 0.

The most significant individual factor is `x4` followed by `x5` and `x1`. These relate to the tasks-per-transaction, transaction-length and tasks-per-processor characteristics respectively. The most significant factor overall is the interaction between these three factors. This was used as the basis to classify the problems.

Table 5.14 gives a different view of the data classified by combinations of the

Problems	Median	I.Q.R.	Censored
low tasks / proc., low tasks / trans., short trans.	110604.5	104148.25	41/540
low tasks / proc., low tasks / trans., long trans.	151019.5	182733.25	94/540
low tasks / proc., high tasks / trans., short trans.	201881.0	*	175/540
low tasks / proc., high tasks / trans., long trans.	*	*	485/540
high tasks / proc., low tasks / trans., short trans.	78730.5	90505.25	16/540
high tasks / proc., low tasks / trans., long trans.	123557.0	153931.75	56/540
high tasks / proc., high tasks / trans., short trans.	105709.5	173914.50	83/540
high tasks / proc., high tasks / trans., long trans.	131671.5	*	152/540

Table 5.14: Result data split by according to problem characteristic

Name	Problems	Weight Values $w_2, \ldots, w_9$
group0	all	3, 3, 3, 38894, 33482, 4067, 13545, 3
group1	low tasks / trans.	3, 3, 3, 49407, 3, 40575, 3, 3
	high tasks / trans.	3, 3, 3, 34682, 24264, 3, 31039, 3
group2	low tasks / trans., short trans.	3, 56619, 3, 33363, 3, 3, 3, 3
	low tasks / trans., long trans.	3, 3, 3, 49600, 3, 40382, 3, 3
	high tasks / trans., short trans.	3, 89979, 3, 3, 3, 3, 3, 3
	high tasks / trans., long trans.	3, 3, 69400, 3, 3, 20582, 3, 3
group3	low tasks / proc., low tasks / trans., short trans.	3, 57393, 3, 32589, 3, 3, 3, 3
	low tasks / proc., low tasks / trans., long trans.	3, 3, 3, 47009, 3, 42973, 3, 3
	low tasks / proc., high tasks / trans., short trans.	3, 3, 3, 31582, 4393, 3, 54010, 3
	low tasks / proc., high tasks / trans., long trans.	11137, 25323, 17713, 3, 3, 9614, 26204, 3
	high tasks / proc., low tasks / trans., short trans.	3, 3, 3, 45993, 43989, 3, 3, 3
	high tasks / proc., low tasks / trans., long trans.	3, 3, 3720, 43805, 42460, 3, 3, 3
	high tasks / proc., high tasks / trans., short trans.	3, 3, 82794, 3, 3, 7188, 3, 3
	high tasks / proc., high tasks / trans., long trans	3, 3, 72829, 3, 3, 17153, 3, 3

Table 5.15: Weightings derived from models fitted to classified problems

tasks-per-transaction, transaction-length and tasks-per-processor characteristics. Three behavioural patterns which relate to these characteristics are evident. Problems with more tasks per processor are easier to solve than those with few. This is explained by the fact that a problem with the same utilisation per processor and more processors results in a higher mean task utilisation and hence is similar to a coarser grained bin packing problem. Problems with more tasks per transaction are harder to schedule since it is more difficult to partition tasks in a way that removes inter-processor communication. Long transactions are more difficult to schedule than shorter ones. A shorter transaction allows more of the tasks to take advantage of the parallelism offered by a multiprocessor system.

The strength of the interaction effect between these three characteristics is remarkable in that it is larger than the effect of any of them acting alone.

Name	Median	I.Q.R.	Censored
group0	37889.0	40140.75	16/96
group1	44368.0	49057.75	13/96
group2	47874.0	61021.25	14/96
group3	31728.5	75781.25	14/96

Table 5.16: Results from weightings fitted to problem subsets

5.4.6.2 Choosing Problem Class Size

Three further sets of weightings were generated by fitting models to problems grouped by problem characteristic. Three groupings were generated using the most significant single characteristic (tasks-per-transaction), the most significant two way interaction (tasks-per-transaction:transaction-length) and the most significant three way interaction (tasks-per-transaction:transaction-length:tasks-per-processor). The weightings obtained from the different problem groupings are given in table 5.15. The group0 weights derived from a model fitted to all problems are repeated here for comparison.

The weighting values listed in table 5.15 gives some insight into the way in which the algorithm is working. The weightings are listed in the same order as the corresponding guidance heuristics in table 5.2. Little attention should be paid to the values in the fourth problem class of group3 since, as shown in table 5.14, there were few uncensored responses in the results which the model was fitted to.

Apart from this one weighting combination in group3, the first weighting value is never above the lowest possible value. This corresponds to g_2, the subfunction for penalising dependent tasks which cannot communicate. It was thought this heuristic would be useful for some problem instances since some problems had hardware platforms which were not completely connected. However, one of the two subfunctions for grouping tasks, g_4 and g_5, have non-minimal values in all cases. It appears that this does a sufficiently good job of placing tasks on the same processor that there is little use for g_2. Also, all tasks penalised by g_2 are also penalised by g_3, the subfunction which penalises incorrectly allocated messages even though a valid allocation which connects the two relevant tasks may be possible. Subfunction g_3 was predicted to be useful for a number of the problem subsets.

The other subfunction with a consistently minimal weighting is g_9 which penalises tasks whose priority order does not match the dependency order within a transaction. The other guidance subfunctions: sensitivity analysis, load balancing, and penalising over utilised processors, were all predicted to be useful for solving some of the problems.

Each of these weighting combinations were tested by performing one run of the algorithm per problem. In each case weightings were mapped to problem subsets as in table 5.15 and then summary statistics were calculated over the entire set of results. These are shown in table 5.16. The result for the group0 weighting combination from table 5.12 is repeated for comparison. The group1 and group2 weighting

Name	Problems	Median	I.Q.R.	Censored
group0	low tasks / proc., low tasks / trans., short trans.	42565.5	34453.75	1/12
	low tasks / proc., low tasks / trans., long trans.	44049.0	8894.75	0/12
	low tasks / proc., high tasks / trans., short trans.	61095.0	70508.75	2/12
	low tasks / proc., high tasks / trans., long trans.	*	*	10/12
	high tasks / proc., low tasks / trans., short trans.	12800.0	12218.50	0/12
	high tasks / proc., low tasks / trans., long trans.	28898.0	21466.00	0/12
	high tasks / proc., high tasks / trans., short trans.	30060.0	31441.25	2/12
	high tasks / proc., high tasks / trans., long trans.	20471.0	14074.50	1/12
group1	low tasks / proc., low tasks / trans., short trans.	36656.0	6882.75	0/12
	low tasks / proc., low tasks / trans., long trans.	28558.5	8221.00	0/12
	low tasks / proc., high tasks / trans., short trans.	78053.0	22315.75	1/12
	low tasks / proc., high tasks / trans., long trans.	*	*	9/12
	high tasks / proc., low tasks / trans., short trans.	40656.5	23067.75	0/12
	high tasks / proc., low tasks / trans., long trans.	35215.0	24900.25	0/12
	high tasks / proc., high tasks / trans., short trans.	36323.0	28336.00	1/12
	high tasks / proc., high tasks / trans., long trans.	47880.5	52157.00	2/12
group2	low tasks / proc., low tasks / trans., short trans.	53189.5	37247.00	1/12
	low tasks / proc., low tasks / trans., long trans.	31586.0	3841.50	0/12
	low tasks / proc., high tasks / trans., short trans.	99947.5	26498.50	0/12
	low tasks / proc., high tasks / trans., long trans.	*	*	12/12
	high tasks / proc., low tasks / trans., short trans.	52698.0	23891.75	0/12
	high tasks / proc., low tasks / trans., long trans.	41107.5	29315.25	1/12
	high tasks / proc., high tasks / trans., short trans.	47611.0	49153.00	0/12
	high tasks / proc., high tasks / trans., long trans.	18911.5	15947.75	0/12
group3	low tasks / proc., low tasks / trans., short trans.	40611.5	43921.50	2/12
	low tasks / proc., low tasks / trans., long trans.	37342.0	16401.00	0/12
	low tasks / proc., high tasks / trans., short trans.	93931.5	25195.00	0/12
	low tasks / proc., high tasks / trans., long trans.	*	*	9/12
	high tasks / proc., low tasks / trans., short trans.	10892.5	13721.50	0/12
	high tasks / proc., low tasks / trans., long trans.	24071.0	14626.25	0/12
	high tasks / proc., high tasks / trans., short trans.	11746.0	11997.50	2/12
	high tasks / proc., high tasks / trans., long trans	18755.0	25336.25	1/12

Table 5.17: Results from test of weighting combinations separated by problem characteristic

combinations gave surprisingly poor performance with worse median response than the group0 weightings. In these cases, use of smaller problem classes is not improving performance. The only possible explanation is a lack of fit in the models. The group3 weightings do give an improved median. Contrary to expectation, these results do not give strong evidence that tuning the algorithm separately for smaller problem classes leads to improved overall performance.

In an attempt to gain further understanding of the results, summary statistics were calculated in each case for the problems separated by characteristic. This is shown in table 5.17. Recall that only group3 used separate weightings for each of these problem classes. For the other weighting combinations, problems mapped to the same set of weightings were separated as required. The table shows that group1 and group2 do perform well for some classes of problem but are not consistent leading to lower overall median values. It is not clear that the group3 weightings are sufficiently better than the group0 weightings to warrant the additional complexity of

$w_{sched}/w_{guidance}$	Median	I.Q.R.	Censored
0.25	42751.0	202933.25	24/96
0.60	41188.0	51733.75	19/96
1.00	42868.0	64557.50	19/96
2.00	49376.0	67563.50	20/96

Table 5.18: Results of changing balance between schedulability and guidance heuristic for *group0* weightings

$w_{sched}/w_{guidance}$	Median	I.Q.R.	Censored
0.25	28706.5	99070.25	18/96
0.60	34348.0	69667.25	14/96
1.00	40959.0	68703.00	13/96
2.00	50094.5	74064.50	17/96

Table 5.19: Results of changing balance between schedulability and guidance heuristic for *group3* weightings

using more problem classes. However, median values for group3 are better in all but one case. Weightings tuned to more specific problem classes would be expected to be more sensitive to problems with characteristics which deviate slightly away from the specific class. However, since the problems used for tuning and testing are the same, this is not a significant issue. There is no clear pattern in the I.Q.R. values, which represent the algorithm's robustness, between class sizes. The group0 and group3 weightings are evaluated further in the next experiment.

5.4.7 Experiment 5.7 — Rebalancing Schedulability Versus Guidance

Experiment 5.7 evaluates changes in the balance between schedulability and guidance using the same problem test set as experiment 5.6. Experiment 5.6 used a value of 0.6 for $w_{sched}/w_{guidance}$. The two weighting combinations: group0 and group3, which were found to work well in experiment 5.6 are re-evaluated at this level along with both higher and lower levels.

The results are given in tables 5.18 and 5.19. The results at the 0.6 level are repeats of the tests carried out in experiment 5.6 and confirm that the group3 weightings outperform the group0 weightings at this level. For both sets of weightings, the performance worsens with higher values of $w_{sched}/w_{guidance}$, i.e. less use of the guidance heuristic. The most interesting result is at the 0.25 level. Using the group0 weightings, the median and interquartile range both increase showing worse performance. Using the group3 weightings, there is a smaller increase in interquartile range and a decrease in median response. This is showing that increasing the bias towards the guidance heuristic improves median performance when using the group3 weightings suggesting it to be a better heuristic than the group0 weightings.

Given that the 0.6 level for $w_{sched}/w_{guidance}$ continues to work well and appears robust, this was maintained for future experiments and the group3 weightings were

Name	Problems	Median	I.Q.R.	Censored
group3	low tasks / proc., low tasks / trans., short trans.	38397.5	39035.75	6/40
	low tasks / proc., low tasks / trans., long trans.	38090.0	23874.25	0/40
	low tasks / proc., high tasks / trans., short trans.	100833.0	16947.50	1/40
	low tasks / proc., high tasks / trans., long trans.	*	*	33/40
	high tasks / proc., low tasks / trans., short trans.	8447.0	12031.75	0/40
	high tasks / proc., low tasks / trans., long trans.	20189.5	14341.50	0/40
	high tasks / proc., high tasks / trans., short trans.	9809.0	4296.00	0/40
	high tasks / proc., high tasks / trans., long trans.	14533.0	12049.50	2/40

Table 5.20: Evaluation of search algorithm on new problem test set

preferred over the group0 weightings.

5.4.8 Experiment 5.8 — Algorithm Performance Consistency Check

In all previous experiments, tests of the search algorithm have been conducted on the same set of problems that it was previously tuned for. In this experiment, new problems are generated to check the consistency of using the guidance heuristic with the group3 sets of weightings which were obtained in experiment 5.6.

The Gentap parameters used were the same as those in tables 5.10 and 5.11 to create 32 different classes of problem. Therefore, the problems had the same characteristics as those in experiment 5.6 but are newly generated problems. 10 problems were generated in each class giving 320 problems in total.

The results were separated into the same problem subsets as those used for each of the group3 weighting combinations. The summary statistics are shown in table 5.20 and can be directly compared to those for group3 in table 5.17. The performance is very similar for both sets of problems. This is pleasing because it not only shows that the performance of the algorithm is consistent but also that the chosen problem characteristics are good indicators of problem difficulty with respect to the algorithm configured in this way.

5.4.9 Experiment 5.9 — Algorithm Scalability

In experiment 5.6, the effects of a selection of five problem characteristics on problem difficulty with respect to the search algorithm were investigated. Compute resources did not allow all characteristics to be investigated.

Two characteristics not chosen for experiment 5.6 were those controlled by the number-of-tasks and utilisation-per-processor Gentap parameters. The effects of these parameters should be predictable. Changing the number-of-tasks parameters scales the problem. The problem characterisation was designed in terms of parameters such as tasks-per-processor, tasks-per-transaction and messages-per-task. Therefore, the number of processors, transactions and messages will increase proportionally with the number of tasks. Increasing the size of the problem should make it harder to solve in terms of the number of configuration evaluations since it is expected that

number-of-tasks	utilisation-per-processor	Median	I.Q.R.	Censored
48	30	28309.5	15689.25	0/10
48	45	31103.0	42282.50	0/10
48	60	37651.5	8822.00	0/10
48	75	50722.5	9013.25	1/10
96	30	197951.5	19268.25	0/10
96	45	232268.0	51549.50	2/10
96	60	224374.0	232394.25	3/10
96	75	381065.0	*	5/10
144	30	311315.5	25134.75	0/10
144	45	428108.5	*	5/10
144	60	*	*	8/10
144	75	*	*	10/10
192	30	398191.0	*	0/10
192	45	466438.0	*	5/10
192	60	*	*	10/10
192	75	*	*	10/10

Table 5.21: Results showing effects of increasing problem difficulty via the number-of-tasks and utilisation-per-processor parameters

the number of valid solutions will grow slower than the size of the design space. Larger problems will also require more time to analyse per evaluation.

Increasing the utilisation per processor will reduce the number of solutions with a valid schedule and therefore should increase the number of configuration evaluations required. Changing the utilisation shouldn't have any effect on the time required to run a schedulability test on the problem.

Experiments were run using the parameters from table 5.3 except the number-of-tasks and utilisation-per-processor were varied through 4 different levels each. The guidance heuristics weightings used were the tuned weightings found in experiment 5.1. Given the increased difficulty of larger problems in particular, the censoring level was raised to 500000 evaluations and the initial temperature was raised from the 0.005 level used in previous experiments to 0.006.

Table 5.21 gives the median, interquartile ranges and number of censored results for different numbers of tasks and utilisation levels. The pattern of results is as expected with both increases in problem size and utilisation level increasing the number of evaluations required. It is clear that increasing the problem size has a larger effect. For large problems with high per-processor utilisations, it was not possible to find a solution within the censoring limit in most cases. However, the algorithm could solve larger problems with lower utilisation levels. Figure 5.15 shows the median number of evaluations required for these lower utilisation levels. At these lower utilisation levels, the number of evaluations required increases at an approximately linear rate with the problem size though the solution space grows much faster. It is not known however, how the proportion of valid solution changes in relation to the size of the problem space.

These results show that problem difficulty should never be estimated by size alone

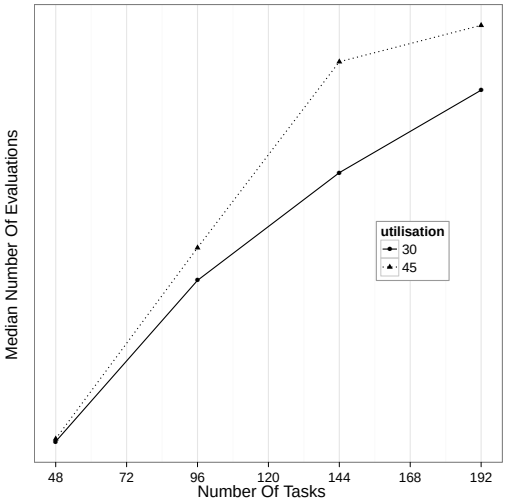


Figure 5.15: Median number of evaluations required to solve problems of increasing size

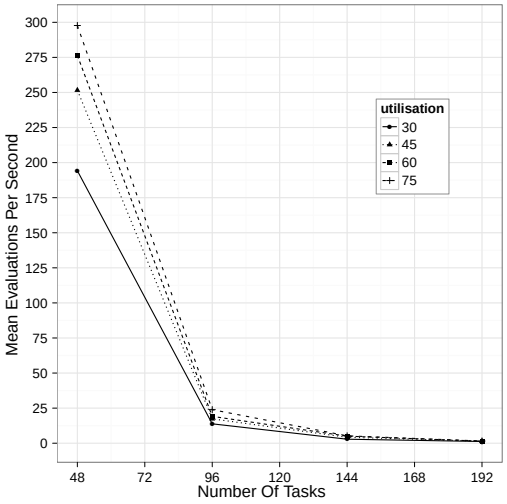


Figure 5.16: Rate of configuration evaluation for increasing problem sizes

but also take other characteristics into account such as the utilisation per processor and those found to be significant in experiment 5.6.

For an overall view of algorithm performance and scalability, the time taken to evaluate each configuration must also be taken into account. Figure 5.16 shows the mean rate of configuration evaluation for the different combinations of characteristics studied in this experiment.

As expected, performance decreases sharply as the size of the problem increases. For the largest problem sizes, the rate of configuration falls close to one evaluation per second. At this rate, the algorithm is running too slow to be of practical use, requiring several days to complete.

Since overall performance is dominated by the rate of configuration evaluation, it is interesting to analyse the schedulability test at a lower level. The schedulability test used, given in section 2.2.3.5, calculates the interference on each task from each of the other tasks in the system. Further to this, analysis is done with each task as a possible initiator of the worst case busy period. This means that performance is expected to be $O(K^{-3})$ at best. The set of equations for the WCDO test is also a recurrence relation and the number of iterations required to solve it will depend on the attributes of each task and the dependencies between them. The number of jobs of each task which need to be checked also varies between task sets. For these reasons, calculating a precise performance relationship between the number of tasks and rate of schedulability test is not possible.

From the plot in figure 5.16, it appears that problems with higher per-processor utilisation are evaluated more quickly. It should be remembered that the plot is based on mean evaluation rates over a whole algorithm run. The WCDO schedulability test iterates until it converges or task deadlines are missed. One hypothesis for

this behaviour is that searches trying to solve problems with higher per-processor utilisations spend more time in areas of the configuration space where several tasks are missing deadlines and the schedulability test is performing fewer iterations. No further evaluation was done to test this hypothesis.

The smallest problems in this experiment are of the scale of an automotive subsystem according to Zheng et al. [81]. These problems are solved in a matter of minutes. Problems moving towards the size of a whole system are not currently within reach of this algorithm. These results must be placed in the context of the WCDO schedulability test being used. This is assuming a synchronous event triggered system. Using an asynchronous time triggered system would likely have a simpler and more efficient schedulability test. Similarly a more accurate test with less pessimism is likely to reduce the rate of evaluation as was demonstrated in experiment 5.5.

The sensitivity analysis cost subfunction (g_6) uses a binary search involving repeated application of the schedulability test and is therefore the most expensive part of the guidance heuristic. As was stated in section 5.4.1, no test of the effects of removing subfunctions from the guidance heuristic (giving them a weighting of 0) was conducted in these experiments.

This experiment and previous ones have shown how problem characteristics effect both the number of configurations which need to be evaluated before a solution is found and the time taken per evaluation. If the effectiveness of removing subfunctions was to be properly tested, experiments would need to take account of the trade-off between possible increases in number of evaluations against increased rate of evaluation. This optimal trade-off point will be problem characteristic dependent. There could also be interaction with algorithm parameters as this has been shown to significantly effect the number of evaluations required throughout these experiments.

It is reasonable to put forward the position that the sheer number of factors involved is prohibitive to optimising algorithm performance in general. The fact that changes to the per-processor utilisation characteristic appear to affect both problem difficulty in terms of number of evaluations required and time per evaluation is subtle. Such complexities in the relationships between factors affecting problem difficulty and performance responses continue to be an issue in the use of metaheuristic search algorithms.

5.5 Summary

A large amount of material has been covered in this chapter. This section recounts the topics which were covered and underlines the main contributions made.

5.5.1 Overview Of Work Done

Section 5.2 introduced a local search algorithm. Parameters allowed it to run as a random search, a pure hill descent algorithm with random restarts or a simulated an-

nealing search. The neighbourhood definition in section 5.2.1 was used to instantiate them specifically for a task allocation problem using the system model presented in section 4.2. The neighbourhood uses a heuristic to favour sensible message allocations which is also used for generating random solutions as described in section 5.2.2.

Section 5.2.3 introduced a hierarchical cost function. The cost value is a weighted sum of cost functions for penalising broken schedulability constraints and for guiding the search towards good solutions more quickly. The guidance heuristic function is broken down into eight further cost subfunctions which are combined using a weighted sum.

In order to determine values for these weightings and other search algorithm parameters, a response surface modelling approach was taken. Models were fitted to data generated from experimental runs and then the response surface function was minimised to find the parameters which were predicted to give the lowest response value. The response measure used was the number of evaluations needed to find a valid solution.

The experiments required search runs to be stopped after a chosen number of evaluations regardless of whether a valid solution had been found. This was needed to conduct a large number of runs in a reasonable amount of time. The problem generation algorithm may also generate problems with no valid solution so a search for a solution would never complete. Stopping an experiment before the event of interest, i.e. finding a valid solution, occurs is known as censoring. Survival analysis, described in section 5.3.5, is a technique for analysing censored data and this was used throughout the experiments.

A sequence of experiments were performed to investigate the relationship between algorithm performance, algorithm parameters including cost function weightings, and problem characteristics.

5.5.2 Overview Of Results

Following the tuning of cost function weightings for simulated annealing in experiments 5.1 to 5.3, each of the search algorithms was tested in experiment 5.4. Random search was found not to be a viable option. A hill descent with random restart algorithm was able to find solutions to the problems it was tested with. However, the median performance was below that of simulated annealing. Importantly, the variation in performance was larger than that for simulated annealing suggesting it was less robust to the uncontrolled factors in the experiment which were random number generator seeds and uncontrolled problem characteristics. Simulated annealing was used in the remaining experiments.

During the tuning of guidance heuristic weightings in experiment 5.1 and experiment 5.6, the best weightings predicted by the fitted response surface gave poorer performance than the best design point used in the experiment. However, the weight-

ings predicted by the model appeared more robust based on lower interquartile range values and solved more problems before searches were terminated by the censoring limit. This suggests that the fitted models use information from all experimental runs as is intended by a response surface methodology and, though there are inaccuracies in the model, are successful in finding good weighting values.

In experiment 5.5, problems were solved using different schedulability tests. The tuned simulated annealing search algorithm was able to cope well with a test that constructed a static cyclic schedule as well as the previously used fixed priority event triggered test. Different fixed priority tests showed the trade-off between test accuracy and time required to run a test.

The response surface methodology was used with problem characteristics as factors in experiment 5.6. Of the five tested, the three most significant characteristics were mean tasks per transaction, mean transaction length and mean tasks per processor. The interaction between tasks per transaction and transaction length was more significant than any individual characteristic as was the interaction between all three of the problem characteristic factors. The differences in median values between problem classes show that several problem characteristics can affect difficulty, not only those related to utilisation or problem size.

Experiment 5.7 showed that a guidance heuristic tuned for specific problem classes was able to outperform one tuned over the entire set of problems. In particular, the search performance improved as weighting given to the guidance heuristic increased. This was not the case for the heuristic which used a single set of weightings for all problems.

Throughout the experimentation, the performance of the tuned simulated annealing algorithm was found to be robust to changes in random number generator seeds and uncontrolled problem characteristics. For example, the same algorithm parameters were used for one design point in experiment 5.2 as were used in experiment 5.1 and performance was extremely similar. An explicit test was done in experiment 5.8. In this experiment, the performance was found to be consistent when the algorithm was applied to different problems having the same characteristics.

The scalability of the algorithm was tested in experiment 5.9. Problems were scaled by the number-of-tasks parameter and made more difficult by increasing the utilisation-per-processor parameter. The number of evaluations required increases approximately linearly with the number of tasks. However, the speed of a schedulability test decreases at a much faster rate as the number of tasks increased. This turns out to be the main limiting factor with regards to the problem size which can be solved in a reasonable amount of time.

An interesting side effect of increased processor utilisation was that the rate of evaluation increased. This was put down to the fact that tasks were known to miss their deadlines after fewer iterations of the schedulability test so that tests could run more quickly. Valid solutions for systems with 96 tasks and 12 processors could be

found in under 12 hours.

5.5.3 Aspects Of Hypothesis Satisfied

This chapter has clearly shown the benefits of using a hierarchical cost function which can be tuned for problems with different characteristics and even allows switching between scheduling policies and associated tests. This supports statement 2 of the hypothesis in section 3.2.1.

It has been shown that benefit can be gained from using different guidance heuristics for problems with different characteristics. A systematic experimental method was given for configuring the guidance heuristic.

5.5.4 Additional Contributions

The primary aim of the work in this chapter was to support statement 2 of the hypothesis as previously mentioned. In the process of achieving this, other contributions have also been made.

In order for a particular set of weights to be associated with a class of problems, a suitable set of characteristics needed to be devised so that the algorithm would give consistent performance with problems in the same class. The characterisation was actually done in chapter 4 by the design of the parameters for the Gentap problem generation tool. In this chapter, it was shown that the characterisation was suitable, made evident by the consistent performance of the search algorithm on problems within the same class. It was notable that problems of the same size with the same mean processor utilisation can vary in difficulty quite substantially depending on the structure of the task graph.

The work in this chapter also raises broader issues in the fields of task allocation and search based software engineering. There is an issue of whether a method should be made flexible or highly optimised for a specific problem instance. Clearly, this author prefers a flexible approach believing that an automated algorithm needs to be adaptable to changing requirements. A search based approach can, for example, accommodate a change to the schedulability test since an implementation can be used directly in the cost function without change.

The results of this chapter back up comments made in section 2.3.2.4 stating that meta-heuristic search is ineffective if used without any guidance heuristic. Heuristics should be incorporated into search algorithms to improve performance but the heuristic must be flexible enough that it can be adjusted, e.g. with the use of weightings. Such a heuristic was presented in this chapter for task allocation.

In the field of SBSE, there is often emphasis on finding software engineering problems where search may be applied and setting out criteria which make problems suitable [104]. While this addresses the search aspects of SBSE, there is little work on the software engineering criteria for the introduction of a search tool. A full

discussion of software engineering processes is somewhat beyond the scope of this thesis yet has a large influence on its aims. The direction of software engineering processes is towards iterative [12] and agile [16] processes which accept uncertain requirements rather than making expensive but futile attempts to perfectly define the system. Software engineering tools must be flexible and recognise that the nature of a problem does not remain static throughout the lifetime of a project.

To meet the needs of modern software engineering processes, both the tools and the solutions they produce need to be flexible. This chapter has tackled the flexibility of a search tool for task allocation. Chapters 6 and 7 will now concentrate on producing flexible solutions and taking advantage of flexibility in future development iterations.

6

Task Allocation For Multi-Moded Systems

6.1 Introduction

There are many real-time applications where the set of running tasks changes throughout the lifetime of the application. When a system moves from one set of tasks to another, it undergoes a *mode change*. There are typically two incentives for doing this: a change in the mode of operation of the system or to adapt to a change of environment. An example of a change in mode of operation is a flight control system which has different modes for take-off, in-flight cruising and landing [91, 30]. Other general operational modes can be identified including initialisation, maintenance, low power, fault recovery and emergency [30]. More information on modes and mode changes can be found in section 2.2.5.

Given requirements for two different modes, one of four things can happen to a task when the system moves from the first mode to the second: it continues running as before, it stops running, it starts running or it continues running but with a change to its attributes (period, deadline, etc.). A change to the attributes of a task may come from a task being required to read data from a sensor more quickly [179]. It is also possible for messages to be removed from or introduced into the system as a result of changes to the task set.

On occasion, it is necessary to run a task on different processors in different modes. There are good reasons to try to select a task mapping for each mode that minimises occurrences of this situation. Depending on the system model, the task either has to be migrated between processors or a copy is maintained on each processor. If the processors are of different types, e.g. one is an application specific integrated circuit (ASIC) and the other is general purpose, then multiple implementations [180]

are needed.

There is a considerable overhead to task migration, usually associated with transferring a large amount of state information [181]. Mode changes should be prompt as tasks running in the new mode may be required to complete before a deadline following the mode change request [30].

For systems where a large proportion of tasks are present in multiple modes, configuring these tasks in as similar way as possible in all modes leads to a more flexible system. In particular, the system is easier to maintain and enhance since the impact of a change can be assessed for a group of modes simultaneously rather than treating each one as a separate system. At run-time, adaptability is improved by virtue of being able to transition between modes more quickly.

An alternative view should also be noted. A mode with fewer tasks may be able to be scheduled on fewer processors allowing processors to be shut down, hence saving power [180]. This creates a trade-off between moving a task in order to shut down a processor and the speed of the mode change. The work in this chapter does not consider power usage and solely concentrates on reducing mode change overheads.

There are also benefits to reducing the number of priority changes between modes. Ideally, each task will have a single unique priority. Otherwise, during a mode change, more priority levels may be needed to maintain the required priority ordering before during and after the mode change. Significant changes to priority ordering are also more likely to change levels of task jitter.

The focus of this chapter is to try to configure tasks and messages within a system in such a way that task allocations and priorities remain constant throughout the configurations for each mode.

6.1.1 Challenges

Finding similar configurations to a set of problem specifications is much more challenging than finding a single solution for a set of problems. If only a single solution is needed then each configuration can simply be evaluated against each of the different problems until one is found which meets the requirements of all problems. More generally, however, each problem specification can be mapped to a set of valid configurations in the solution space. A visualisation of this with two problem specifications is shown in figure 6.1. For each specification, there exists a number of valid configurations. The number of differences between the configurations is visualised as a distance in figure 6.1.

The optimal solution is the pair of valid solutions which have the shortest distance between them. Finding this solution would require every pair of valid solutions to be checked unless a pair of solutions are identical in which case they are known to be an optimal solution. For more problem specifications, i.e. a system with more than two modes, a measure of solution similarity for each possible set of solution needs to

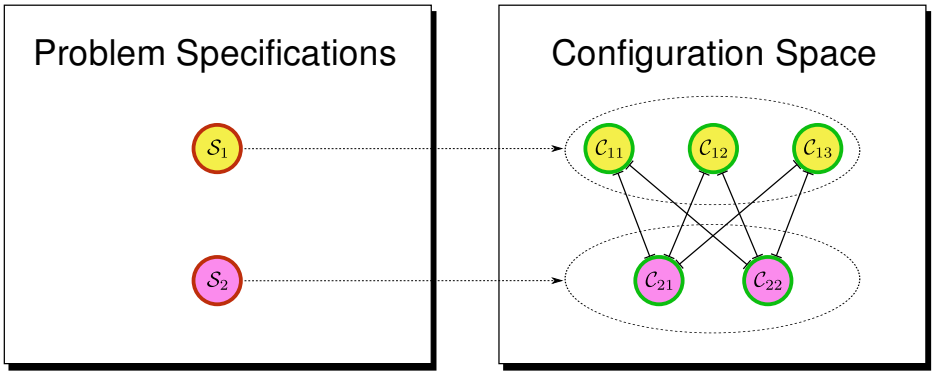


Figure 6.1: Depiction of mapping from problem specifications to valid configurations with distances between configurations

be constructed from individual distances.

The work in the previous chapter has shown that finding a single valid solution for a problem specification is already challenging. Finding several solutions for each problem and checking for similar ones is not practical. Instead, an algorithm needs to be developed which uses knowledge of the fact that a set of similar solutions is needed and will move towards suitable areas of the solution space where such sets exist. Such an algorithm is engineered and evaluated in this chapter. The features developed address Req. 2, from section 3.1.2, on the implementation of systems with multiple configurations.

To find configurations which are similar to each other, further cost subfunctions are used to measure the difference between two configurations. These can also be used for finding configurations which are similar to existing ones. This allows the Toast task allocation tool to be used to evolve a design with few changes rather than find a completely different one for each requirements change. Therefore, the work in this chapter also addresses Req. 1, given in section 3.1.1, on the reuse of solutions. This theme is also the main focus of chapter 7 which will make further use of the algorithms developed in this chapter.

6.1.2 Goals And Chapter Structure

Statement 1 of the hypothesis in section 3.2.1 says that local search is an efficient and effective method for meeting requirements Req. 1 and Req. 2. The evaluations in this chapter and the next will test this statement.

Further examples which illustrate the issues of minimising change between configurations are presented in section 6.2. These examples provide simple test cases which are used to analyse the behaviour of algorithms suggested in section 6.3. An evaluation of the proposed algorithm is performed in section 6.4. A summary of results is given in section 6.5.

Task	Utilisation	
	Mode 1	*Mode 2*
A	60	80
B	40	40
C	50	50
D	50	0

Table 6.1: Multi-mode task sets example 1

Task	Configuration		
	Mode 1	*Mode 2*	*Unchanged*
A	P1	P1	✓
B	P1	P2	✗
C	P2	P2	✓
D	P2	–	✓

Table 6.2: Possible configuration for example 1 task sets

6.2 Motivating Examples

Two small examples are used as simple test cases which guide development of an algorithm which finds a set of similar solutions for a set of problem specifications. To maintain simplicity of the example, task attributes are reduced to just the utilisation of the task and it is assumed that the tasks on a processor may be scheduled if the total task utilisation for that processor is less than 100%.

Table 6.1 shows the percentage utilisation requirements of four tasks in two modes. In the transition from *Mode 1* to *Mode 2*, task A requires an increase in utilisation (e.g. from an increase in frequency or execution time) and task D stops running. The platform on which these tasks must run has two processors available. Assuming homogeneity of processors, so that processor labels can be swapped without loss of generality, there is only a single possible configuration for *Mode 1* that will allow all tasks to be scheduled. Task A cannot be allocated to the same processor as tasks C or D as this would require over 100% utilisation of the processor. This results in task A being paired with task B in *Mode 1*. When the transition is made to *Mode 2*, task B can no longer run alongside task A and must migrate to the other processor where capacity has become available after task D terminated.

There are two ways in which the tasks may be allocated to be schedulable in *Mode 2*. Either, as shown in table 6.2, with A on P1 and B and C on P2 or with A on P2 and B and C on P1. Note that when selecting a configuration for the second mode, the processors may no longer be considered homogeneous since the number of migrations must be measured relative to the configuration chosen for *Mode 1*. These two possible solutions for *Mode 2* require 1 migration and 2 migrations respectively with the former being identified in the right hand column of table 6.2.

A slightly modified version of the first example is shown in table 6.3. In this

Task	Utilisation	
	Mode 1	*Mode 2*
A	50	80
B	50	50
C	50	50
D	50	0

Table 6.3: Multi-mode task sets example 2

Task	Configuration X			Configuration Y		
	Mode 1	*Mode 2*	*Unchanged*	*Mode 1*	*Mode 2*	*Unchanged*
A	P1	P1	✓	P1	P1	✓
B	P1	P2	✗	P2	P2	✓
C	P2	P2	✓	P2	P2	✓
D	P2	–	✓	P1	–	✓

Table 6.4: Possible configurations for example 2 task sets

example, all tasks have a utilisation of 50% in *Mode 1*. This means that there are three possible ways of grouping the tasks in the configuration for *Mode 1* so that they all can be scheduled. Two of these possible solutions are shown in table 6.4.

If configuration X is selected then at least one migration is required for the tasks to also be schedulable in *Mode 2*. However, if configuration Y is selected then it is possible to move from *Mode 1* to *Mode 2* with no task migrations. This illustrates the fact that, when selecting a configuration for a particular mode, information regarding transitions to other possible modes is required to achieve minimal task migrations. Note that there is only a single valid task grouping for *Mode 2*. If the configuration for *Mode 2* had been selected first, then this would have lead more naturally to using configuration Y as a solution for both modes.

6.3 Solution Methods

Three approaches to minimising changes in mode transitions are identified for consideration. All make use of the simulated annealing algorithm previously described in chapter 5. Each of the methods are described in terms of changing from *Mode 1* to to *Mode 2*. Extensions of the methods to cope with systems with more than two modes are considered in section 6.3.5. The three methods are named *sequential method*, *simultaneous method* and *parallel method*. The idea behind each method is summarised below.

6.3.1 Proposed Methods

Sequential Method. The sequential method finds a solution for *Mode 1* without any consideration for *Mode 2*. It then tries to find a solution for *Mode 2* using

the solution found for *Mode 1* as a starting point and in addition to finding a valid schedulable solution for *Mode 2*, attempts to minimise the number of changes between the solutions.

Simultaneous Method. The simultaneous method attempts to find a single configuration which meets the requirements of all modes. Tasks and messages found in both modes are configured in the same way in every mode. If such a configuration cannot be found then, after a certain number of evaluations, the search is terminated and the best solution found so far is used as a starting point for further searches. In these subsequent searches, a valid solution for each mode is found which is as similar as possible to the starting configuration.

Parallel Method. The parallel method, like the simultaneous method, tries to find a configuration for all modes at the same time. However, a separate configuration for each mode is maintained in separate searches. The searches exchange information about their current best configurations according to a protocol described in section 6.3.6. Each search can then try to minimise differences between its own solutions and those of other searches.

The parallel method described in this chapter is an evolution of that found in a previous paper by the author [182] on which the work in this chapter is based. More details of the differences are explained in section 6.3.6. It is the same algorithm as used in a more recent paper by the author [164].

6.3.2 Configuration Change Cost Subfunctions

All of the methods outlined above need a way of measuring by how much two configurations differ. This can then be used as part of a cost function which penalises change. Four new cost subfunctions are introduced: task allocation changes, message allocation changes, task priority changes and message priority changes.

Let there be two configurations, $\mathcal{C}$ and $\mathcal{C}'$. The sets of tasks and messages contained in each configuration may not be the same. A change is only counted for objects present in both configurations. $\mathcal{T}$, $\mathcal{M}$, $\mathcal{P}$, $\mathcal{N}$ is the set of tasks, messages, processors and networks in $\mathcal{C}$ and $\mathcal{T}'$, $\mathcal{M}'$, $\mathcal{P}'$, $\mathcal{N}'$ are the equivalent sets contained in $\mathcal{C}'$.

The function used to measure task allocation changes (h_{tac}) and message allocation changes (h_{mac}) are as follows:

$$h_{tac}(\mathcal{C}, \mathcal{C}') = \frac{|\{\tau \in (\mathcal{T} \cap \mathcal{T}') : \mathrm{al}_{\mathcal{C}}(\tau) \neq \mathrm{al}_{\mathcal{C}'}(\tau)\}|}{|\mathcal{T} \cap \mathcal{T}'|} \tag{6.1}$$

$$h_{mac}(\mathcal{C}, \mathcal{C}') = \frac{|\{\rho \in (\mathcal{M} \cap \mathcal{M}') : \mathrm{al}_{\mathcal{C}}(\rho) \neq \mathrm{al}_{\mathcal{C}'}(\rho)\}|}{|\mathcal{M} \cap \mathcal{M}'|} \tag{6.2}$$

where $\mathrm{al}_{\mathcal{C}}(\tau)$ is the current allocation of the object and $\mathrm{al}_{\mathcal{C}'}(\tau)$ is the allocation in

the baseline configuration. These simply give the proportion of schedulable objects common to both configurations which are allocated to a different scheduler.

There are two factors which prevent the use of a priority change metric based on directly comparing priority values of equivalent objects. The assigned priority values may be different between two configurations with very few differences between priority order. For example, two sets of tasks with the same priority order but different numbering schemes should be considered equivalent as one can easily be remapped to the other. Comparison of priorities between two configurations is therefore based on the rank of a priority in an ordered list rather than the absolute value. The priority difference metric is based on Spearman's rank correlation coefficient [183].

Comparing priorities only make sense for objects allocated to the same scheduler and so the priority difference metrics only involve objects which have the same allocations in both configurations.

Let X and Y be sets of schedulable objects and $n = |X \cap Y|$ is the number of objects in their intersection. For each object $\tau \in (X \cap Y)$, $\mathrm{RANK}_X(\tau)$ is the rank of object τ in X and $\mathrm{RANK}_Y(\tau)$ is the rank of the same object in Y. The basis of the priority comparison metric is

$$q(X,Y) = \begin{cases} \frac{3}{n(n^2-1)} \sum_{\tau \in (X \cap Y)} (\mathrm{RANK}_X(\tau) - \mathrm{RANK}_Y(\tau))^2 \text{ if } n > 1 \\ 0 \text{ otherwise} \end{cases} \tag{6.3}$$

For a given scheduler, σ, let $O_{\mathcal{C}}(\sigma)$ be the set of objects on that scheduler in the current configuration and $O_{\mathcal{C}'}(\sigma)$ be the equivalent set in the baseline configuration. The functions for measuring task priority changes (h_{tpc}) and message priority changes (h_{mpc}) are

$$h_{tpc}(\mathcal{C}, \mathcal{C}') = \frac{1}{|\mathcal{P} \cap \mathcal{P}'|} \sum_{\sigma \in (\mathcal{P} \cap \mathcal{P}')} q(O_{\mathcal{C}}(\sigma), O_{\mathcal{C}'}(\sigma)) \tag{6.4}$$

$$h_{mpc}(\mathcal{C}, \mathcal{C}') = \frac{1}{|\mathcal{N} \cap \mathcal{N}'|} \sum_{\sigma \in (\mathcal{N} \cap \mathcal{N}')} q(O_{\mathcal{C}}(\sigma), O_{\mathcal{C}'}(\sigma)) \tag{6.5}$$

Note that all of h_{tac}, h_{mac}, h_{tpc} and h_{mpc} are commutative.

The actual cost subfunctions can be applied to a set of configurations rather than just a pair. They use the functions just described above and apply them to every

pair of configurations in the given set.

$$g_{tac}(\{\mathcal{C}_1,\ldots,\mathcal{C}_l\}) = \frac{2}{l(l-1)}\sum_{i=1}^{l-1}\sum_{j=i+1}^{l} h_{tac}(\mathcal{C}_i,\mathcal{C}_j) \tag{6.6}$$

$$g_{mac}(\{\mathcal{C}_1,\ldots,\mathcal{C}_l\}) = \frac{2}{l(l-1)}\sum_{i=1}^{l-1}\sum_{j=i+1}^{l} h_{mac}(\mathcal{C}_i,\mathcal{C}_j) \tag{6.7}$$

$$g_{tpc}(\{\mathcal{C}_1,\ldots,\mathcal{C}_l\}) = \frac{2}{l(l-1)}\sum_{i=1}^{l-1}\sum_{j=i+1}^{l} h_{tpc}(\mathcal{C}_i,\mathcal{C}_j) \tag{6.8}$$

$$g_{mpc}(\{\mathcal{C}_1,\ldots,\mathcal{C}_l\}) = \frac{2}{l(l-1)}\sum_{i=1}^{l-1}\sum_{j=i+1}^{l} h_{mpc}(\mathcal{C}_i,\mathcal{C}_j) \tag{6.9}$$

Separate subfunctions are defined for tasks and messages allowing the importance of minimising change for each to be set as desired. The number of configurations in the input set, l, is dependent upon the context in which these subfunctions are used.

As with previous bottom level subfunctions, they are combined using a normalised weighted sum:

$$change(\{\mathcal{C}_1,\ldots,\mathcal{C}_l\}) = \frac{w_{tac}g_{tac} + w_{mac}g_{mac} + w_{tpc}g_{tpc} + w_{mpc}g_{mpc}}{w_{tac} + w_{mac} + w_{tpc} + w_{mpc}} \tag{6.10}$$

Each of the methods proposed in section 6.3.1 use the function in equation (6.10) in a slightly different way. This is now explained along with further analysis of each of the proposed methods.

6.3.3 Sequential Method

The sequential method assumes a solution, $\mathcal{C}_1$, for ***Mode 1*** is found using a task allocation algorithm such as that in the previous chapter. A second search then finds a solution for ***Mode 2*** in the vicinity of the solution for ***Mode 1***. This search uses $\mathcal{C}_1$ as its initial solution and looks for a solution which meets the requirements of ***Mode 2*** but remains as close as possible to the starting point according to the *change* function in equation (6.10). Note that the idea of the *change* function as a distance metric is only used in a loose sense, not a formal mathematical one, to give a diagrammatic explanation of the problem.

To encourage the simulated annealing algorithm to intensively search the area surrounding $\mathcal{C}_1$, lower starting temperatures and slower cooling schedules are used compared to the search used which found $\mathcal{C}_1$. In addition to this, the *change* function is added to the cost function hierarchy to penalise configurations with more differences. This new function hierarchy is shown in figure 6.2. It is an extension of the hierarchy first shown in figure 5.6. The cost function f (equation (5.3)) is replaced

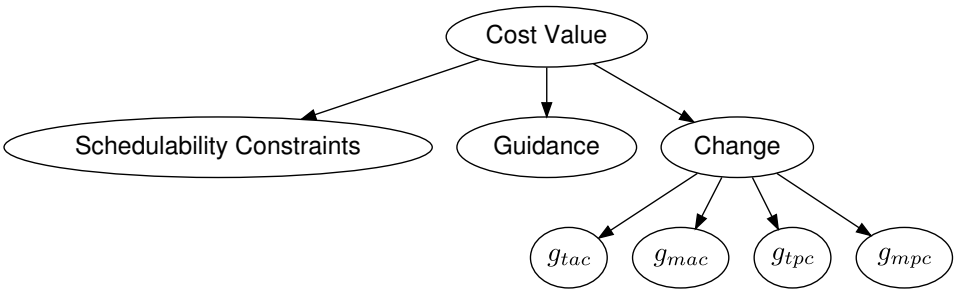


Figure 6.2: Cost function hierarchy extended with subfunctions for measuring changes between configurations

by one which represents the hierarchy in figure 6.2

$$f_{seq}(\mathcal{C}, D) = \frac{w_{sched}\, sched(\mathcal{C}) + w_{guidance}\, guidance(\mathcal{C}) + w_{change}\, change(\{\mathcal{C}\} \cup D)}{w_{sched} + w_{guidance} + w_{change}} \tag{6.11}$$

D is a set of configurations to use as baselines to measure differences from. For two mode problems, $D = \{\mathcal{C}_1\}$.

f_{seq} includes one more weighting value, w_{change}, than the cost function in equation (6.10). This must be balanced with the other subfunctions at the same hierarchical level. If solution $\mathcal{C}_1$ is in fact near to a solution for the second mode then the search should need little guidance and concentrate on finding a schedulable solution with few changes. If no such solution exists, the weightings must be adjusted to reflect this. Since it cannot be known in advance how many changes will be needed, it is likely some experimentation will be needed to find weightings which give good quality results and such values will almost certainly be problem specific.

The earlier examples and their solutions shown in tables 6.1, 6.2, 6.3 and 6.4 show that the success of this method may depend on the order in which the configurations are generated. For the example in table 6.1, both modes only have a single possible grouping under which all objects are schedulable. In this case, the method would be expected to achieve the optimal result of a single allocation change regardless of whether a configuration for *Mode 1* or *Mode 2* is generated first. However, in the example shown in table 6.3, *Mode 1* has multiple solutions under which all objects are schedulable. If configuration X for *Mode 1* from table 6.4 is generated then it is impossible to achieve the optimal solution in the second step.

This indicates that the quality of the overall solution is dependent on the order in which individual configurations are generated. For the task sets in table 6.3, if *Mode 2* is generated first, then as in the first example, there is only a single possible feasible grouping and it is expected that the sequential method will obtain the optimal result. This observation implies that trying all orderings of the modes may improve solution quality. However, it also adds complexity and increases compute resources, especially if the method is extended to more than two modes.

This method has the benefit of simplicity over the subsequent two methods anal-

ysed in this section. However, it is expected that better results can be obtained using an algorithm which considers all of the problem specifications of all modes simultaneously.

6.3.4 Simultaneous Method

The first step of this method tries to produce a *super-configuration* which meets the requirements of all modes. This configuration contains all objects from all modes. Objects present in more than mode must therefore be configured identically in each mode.

A new cost function is created which takes the mean cost value of each specification evaluated with the super-configuration. Assuming two modes still this cost function is

$$\bar{f}(\mathcal{C}) = \frac{f(\mathcal{C};\mathcal{S}_1) + f(\mathcal{C};\mathcal{S}_2)}{2} \tag{6.12}$$

where $f(\mathcal{C};\mathcal{S}_i)$ is the cost of configuration $\mathcal{C}$ in the context of specification $\mathcal{S}_i$. A configuration is schedulable if and only if $sched(\mathcal{C};\mathcal{S}_1)$ and $sched(\mathcal{C};\mathcal{S}_2)$ are both 0.

It is simple to see that this method should successfully find an optimal solution to the problem in table 6.3 since there exists a single configuration for which all tasks are schedulable in both modes. In this instance, it has an advantage over the sequential method whose result depends on which mode is processed first.

If no single configuration can be found which satisfies the requirements of both specifications, then a secondary search must be performed with the f_{seq} cost function using the best super-configuration found as a baseline configuration to minimise differences from.

Using this scheme, it can also be shown that the simultaneous method would find an optimal solution for the problem in table 6.1. In this example, there is no single configuration that can schedule both modes so the search should find a solution which minimises the number of unschedulable tasks over both modes. The search may conceivably produce one of two solutions. If it chooses to schedule all tasks under *Mode 1*, then only task B will be unschedulable in *Mode 2*. Alternatively, it may configure tasks A, B and C to be schedulable in *Mode 2*. With this configuration, these tasks will also be schedulable in *Mode 1*, leaving only D unschedulable in *Mode 1*. The super-configuration can then be used as a baseline for the mode which it does not solve to work from. If either of the two solutions are found, a feasible solution can be found for the other mode with a single allocation change.

The task sets in table 6.5 give an example where this method may be less successful. Once again, there is no single configuration where both modes can be scheduled. Configuration W in table 6.6 shows how the modes can be scheduled with a single task migration. However, if any of the *Mode 2* configurations in table 6.6 are taken and applied to *Mode 1*, all of tasks A, B, C and D will be schedulable leaving a single unschedulable task, E. Ignoring the guidance heuristic, the *Mode 2* configurations X,

Task	Utilisation	
	Mode 1	*Mode 2*
A	60	80
B	40	40
C	5	5
D	5	5
E	80	0

Table 6.5: Multi-mode task sets example 3

Task	Configuration W			Configuration X		
	Mode 1	*Mode 2*	Unchanged	*Mode 1*	*Mode 2*	Unchanged
A	P1	P1	✓	P1	P1	✓
B	P1	P2	✗	P1	P2	✗
C	P2	P2	✓	P2	P1	✗
D	P2	P2	✓	P2	P2	✓
E	P2	-	✓	P2	-	✓
Task	**Configuration Y**			**Configuration Z**		
	Mode 1	*Mode 2*	Unchanged	*Mode 1*	*Mode 2*	Unchanged
A	P1	P1	✓	P1	P1	✓
B	P1	P2	✗	P1	P2	✗
C	P2	P2	✓	P2	P1	✗
D	P2	P1	✗	P2	P1	✗
E	P2	-	✓	P2	-	✓

Table 6.6: Possible configurations for example 3 task sets

Y and Z, have the same cost as configuration W but do not allow a transition to *Mode 1* with only a single migration.

There is only a single schedulable configuration for *Mode 1*. On some occasions the search will output this configuration, which when applied to *Mode 2*, also leaves only a single task, B, unschedulable. If this is the case then the second step of the method should be able to find the *Mode 2* allocation in configuration W which requires only a single migration. This indicates that for example 3, it would be better to concentrate on producing a schedulable solution for *Mode 1* rather than *Mode 2* in the initial step.

Previous work by this author [182] suggested using weightings in equation (6.12) to favour one of the modes when generating the initial super-configuration. Similar to trying different orderings of modes in the sequential method, several super-configurations could be generated favouring different modes. However, this idea is dismissed here since, as in the sequential method, there is no way of knowing which mode to favour and the method becomes impractical.

In terms of processing required, the simultaneous method is more expensive than the sequential method. The generation of the initial super-configuration requires an evaluation of the cost function for each mode. The second step which finds a

configuration for the second mode is equivalent.

By considering both modes in the initial step, the simultaneous method is more reliable than the sequential method for finding identical solutions to both modes. However, when this is not the case, it is more computationally expensive and still leaves open questions of whether to favour certain modes to be configured first. The parallel method, described in section 6.3.6, uses concepts of the simultaneous method which make it good at finding valid identical configurations for multiple modes but tackle some of its problems in cases where this is not possible.

Before the parallel method is described in detail some issues surrounding applying the sequential and simultaneous methods to systems with more than two modes are discussed. This will identify further issues which need to be tackled in the design of a task allocation algorithm for multi-moded systems.

6.3.5 Extensions To Several Modes

Most multi-moded systems have more than two modes. The scalability of each method with respect to the number of modes must also be considered. A system with N modes has $N(N-1)/2$ distinct pairs of modes. Therefore, the number of mode transitions that must be dealt with increases quadratically with the number of modes. In most systems, only a subset of these transitions will take place.

There are two conceivable ways of extending the sequential method for more than two modes:

1. Configurations are generated one by one, minimising changes to all previous configurations within which a mode transition may occur. This increases the problem of knowing in which order the modes should be tackled. There are $N!$ possible sequences and it will become increasingly likely that picking a single ordering at random will give poor quality solutions.
2. The first configuration is generated then all others are generated minimising changes to this one configuration. If all configurations are near to the first one, it could be hoped they were also similar to each other. However, this is likely to be an optimistic assumption and results would probably vary quite considerably between search runs. This method does have the advantage that the second stage could easily be performed concurrently unlike a purely sequential one.

The first of these extensions is reliant on making an arbitrary choice for the mode ordering or trying every possible order. Neither of these is satisfactory. The second is also reliant on an arbitrary choice for the first mode and relies somewhat on the randomness of the search producing similar solutions in the second stage.

The parallel method detailed in the following section overcomes the problem of ordering the modes since all configurations in a created single step. Unlike the first stage of the simultaneous method, these configurations are allowed to differ but are increasingly penalised for doing so.

6.3.6 Parallel Method

The concept of the parallel method is to run simulated annealing searches in parallel with each search assigned a separate problem specification to solve. The search threads* share information in an attempt to keep their solutions as similar as possible.

6.3.6.1 Requirements

The requirements for the parallel simulated annealing algorithm are as follows:

1. to allow there to be differences between configurations produced for each problem specification
2. to minimise the differences between the final configurations for each of the input specifications
3. to be scalable in terms of the additional computation required for increasing numbers of problem specifications
4. to produce repeatable results for a fixed set of inputs (including a random number generator seed)

The first requirement is accomplished by assigning responsibility for each problem specification to a separate search thread. Threads are penalised more as their current solutions become increasingly different from the best solution of other threads but configurations are not enforced to be the same. This mechanism is used to meet the second requirement.

The third requirement is achieved by running each search thread in parallel. This allows the algorithm to scale as long as the hardware platform provides sufficient processing cores for the number of scenarios used. The implementation of the algorithm for the Toast task allocation tool uses a shared memory multi-threading model. Since 8 core machines are becoming commonplace, it is reasonable to state that the algorithm scales to a problem with 8 problem specifications. An alternative implementation, allowing distribution over networked machines, would provide further scalability. Since a problem with more specifications must minimise differences between more pairs of configurations, the number of evaluations required by each search thread to obtain high quality solutions may also be larger.

Section 5.3.4 outlined the factors affecting search algorithm results for the single threaded simulated annealing algorithm used previously. When a concurrent algorithm is used, an additional uncontrollable nuisance factor, namely the interleaving of threads, must potentially be added to this list. If the result of the algorithm is dependent on the times at which threads communicate, then the results of the algorithm can vary on subsequent executions for an identical problem using identical

*The term *threads* is used here to mean separate threads of execution. The implementation could use any mechanism of concurrency that can support the algorithm described.

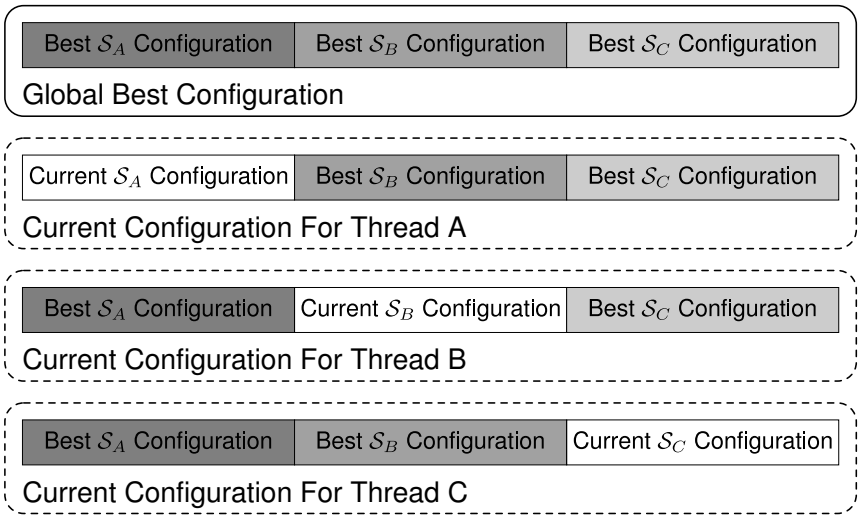


Figure 6.3: Solution representation for multi-problem search

parameters. The final requirement dictates that the algorithm is repeatable. This is of benefit to researchers and practitioners. There can be more confidence in experimental results if nuisance factors are removed. For practitioners, the tool can be ported between platforms without affecting results and the sequence of decisions that led to a solution can be reproduced.

This final requirement is dealt with by the design of a mechanism which defines synchronisation points in terms of the sequence of cost values produced by each search thread rather than time based synchronisation which was used previously [182]. This mechanism is described later in this section.

6.3.6.2 Representation And Cost Evaluation

Figure 6.3 shows the representation of the solution which would be held by three search threads, labelled A, B and C, solving three specifications, $\mathcal{S}_A$, $\mathcal{S}_B$ and $\mathcal{S}_C$. The search algorithm which initiates the search threads considers itself to be working on finding a single large configuration with sections for each specification. This is different from the super-configuration used in the simultaneous method since objects can appear multiple times and be configured differently in different sections of the configuration. It is this configuration, labelled Global Best Configuration in figure 6.3, which is output at the end of the search.

Each search thread maintains a configuration for all specifications but only operates on one section of it for its specification. The other parts of the configuration are filled with the best configuration found by other search threads for their specifications. The global configuration is made up of the best configurations found for each specification.

The cost function used by each search thread operates on the whole of its locally stored configuration, not just the part pertaining to the specification it has been

assigned to. Within each thread, the cost of the whole configuration is obtained by applying f_{seq} from equation (6.11) to each section of the configuration and taking a mean value. The function which does this is f_{par}.

$$f_{par}(\{\mathcal{C}_1, \ldots, \mathcal{C}_N\}) = \frac{1}{N} \sum_{i=1}^{N} f_{seq}(\mathcal{C}_i, \{\mathcal{C}_1, \ldots, \mathcal{C}_N\} \setminus \{C_i\}; \mathcal{S}_i) \tag{6.13}$$

f_{par} applies f_{seq} to each part of the configuration in the context of the relevant specification. The difference set passed to f_{seq} is the input set minus the configuration currently being evaluated.

The results of the schedulability and guidance heuristic subfunctions will not change for parts of the configuration assigned to other threads between synchronisation points. This allows many values to be cached to speed up the calculation of f_{par}.

For clarity, the cost function which thread A in figure 6.3 would use is

$$\begin{aligned} f_{par}(\{\mathcal{C}_{A_{cur}}, \mathcal{C}_{B_{best}}, \mathcal{C}_{C_{best}}\}) = \\ \frac{1}{3}\big[& f_{seq}(\mathcal{C}_{A_{cur}}, \{\mathcal{C}_{B_{best}}, \mathcal{C}_{C_{best}}\}; \mathcal{S}_A) + \\ & f_{seq}(\mathcal{C}_{B_{best}}, \{\mathcal{C}_{A_{cur}}, \mathcal{C}_{C_{best}}\}; \mathcal{S}_B) + \\ & f_{seq}(\mathcal{C}_{C_{best}}, \{\mathcal{C}_{A_{cur}}, \mathcal{C}_{B_{best}}\}; \mathcal{S}_C)\big] \end{aligned} \tag{6.14}$$

When a thread using f_{par} for cost evaluation makes a change to a single configuration, the size of the cost change is going to be affected by the number of specifications involved. The probability of accepting a configuration within simulated annealing is affected by the absolute magnitude of cost changes and parameters which set the initial temperature and rate of cooling. To make the behaviour of certain values of the temperature related parameters somewhat independent of the number of specifications involved, line 19 of figure 5.1 is changed from

$$\mathsf{delta} = \mathsf{newcost} - \mathsf{curcost}$$

to

$$\mathsf{delta} = (\mathsf{newcost} - \mathsf{curcost}) * N$$

for N specifications.

The reason for each search thread using a cost value for all of the configuration, not just the part it is modifying, is to enable comparisons between cost values from different search threads and with the cost of the solution currently held as the global best solution. This is essential to the working of the synchronising mechanism which is now described.

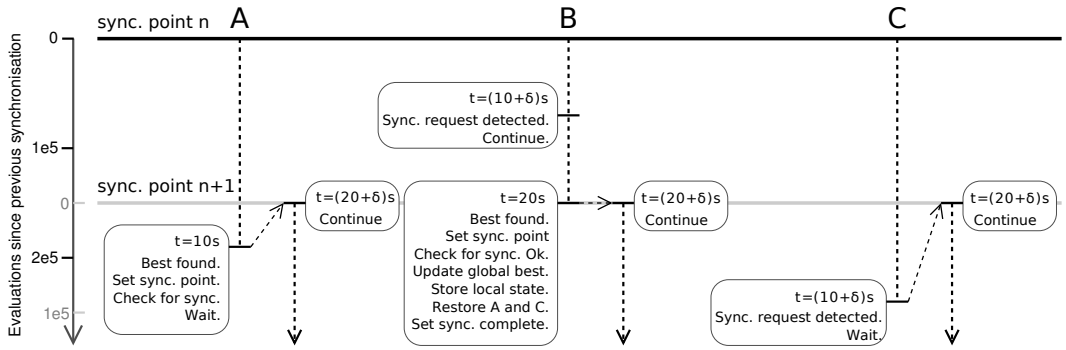


Figure 6.4: Synchronisation mechanism

6.3.6.3 Synchronisation Mechanism

The strategy for synchronising threads is to exchange information whenever a search thread finds a solution better than the currently held global best solution. For a given set of inputs, a search will find its first improvement on the initial solution after the same number of evaluations on every run. This can be used as a basis for creating a repeatable sequence of interactions.

When each search thread is first spawned from the main application, each thread locally stores a copy of all its state variables which can affect the decisions it takes. For simulated annealing, this includes the current temperature and inner loop count. In order to achieve repeatability, the state of the pseudo-random number generator must also be controlled. The implementation in Toast uses a separate instance of the Mersenne Twister [167] in each search thread. These are seeded by the random number generator in the main application which is seeded by a controllable parameter.

The steps taken to synchronise the searches are best explained by way of the example shown in figure 6.4. At the top of the diagram, the searches have just passed a synchronisation point, which could be the start of the search. The next synchronisation point is based on which search finds a solution better than the global best solution in the fewest evaluations since the last synchronisation point. If more than one search finds a better solution after the same number of evaluations, then the one with the lower cost is preferred. After this, ties are broken based on a deterministic ordering of the search threads. Figure 6.4 shows the following steps.

1. After 10 seconds, search A finds a new best solution and sets a global data flag requesting synchronisation. It now waits for other searches to reach the same number of evaluations.

2. Just after 10 seconds searches B and C read this synchronisation request. Search B hasn't performed enough evaluations and continues. Search C is past the synchronisation point so stops and waits.

3. After 20 seconds, search B still hasn't reached the synchronisation point but

has found a new best solution. It sets the synchronisation point to a lower number of evaluations. All searches are now at or past the synchronisation point. Search B is the only search exactly at the synchronisation point so it updates the global best with its solution. It then makes a copy of its local state. The other searches restore their local state from the last copy made. B sets a flag indicating synchronisation is complete and continues.

4. Searches A and C continue using the new best solution found by B as their starting point.

Since each specification is expected to have a similar solution to the others, immediately following a synchronisation, each search thread will try applying the current best solution for other specifications to their specification. This solution can be accepted or rejected using the usual simulated annealing acceptance criteria. This optimisation means that the parallel method can often find a solution which has identical configurations for each specification as is the case for the initial stage of the simultaneous method.

A simpler repeatable parallel algorithm exists whereby all search threads run in lock step so a thread cannot evaluate a new configuration until all other threads are also ready to proceed. This requires threads to synchronise after every evaluation and this communication would become more expensive if the implementation was scaled up across several machines. In contrast, the synchronisation mechanism described in this section only requires threads to synchronise when a new best solution is found. This event becomes increasingly rare as the search proceeds.

There is, however, the possibility of the parallel method wasting computation if one thread evaluates solutions more quickly than another but it is not the thread that finds a new best solution after fewest evaluations. Therefore, it is most efficient if the hardware platform provides all threads with roughly equal resources. This wastage could be removed at the expense of losing repeatability but not necessarily reducing solution quality. This is not evaluated in this thesis; only the repeatable form of the algorithm is used.

When implementing the parallel method's synchronisation mechanism, care must be taken with the handling of stopping conditions to ensure perfect repeatability. If a thread meets its stopping conditions and some other threads have not, then it cannot simply exit. Another thread may improve on the global best solution and request that threads synchronise. Threads which have met stopping conditions must wait and listen for signals from other threads indicating a synchronisation is required or that all threads have met stopping conditions. Stopping conditions based on a maximum number of evaluations use the synchronisation point counter. The actual number of evaluations performed by a thread can be larger than the maximum number requested if, at some point during the search, the thread evaluates solutions more quickly than the thread initiating the next synchronisation. The Toast implementation of this

Class	Problems
Class 1	low tasks / proc., low tasks / trans., short trans.
Class 2	low tasks / proc., low tasks / trans., long trans.
Class 3	low tasks / proc., high tasks / trans., short trans.
Class 4	low tasks / proc., high tasks / trans., long trans.
Class 5	high tasks / proc., low tasks / trans., short trans.
Class 6	high tasks / proc., low tasks / trans., long trans.
Class 7	high tasks / proc., high tasks / trans., short trans.
Class 8	high tasks / proc., high tasks / trans., long trans.

Table 6.7: Problem classes used in experiment 5.8

algorithm uses the POSIX pthread conditional synchronisation mechanism.

6.4 Evaluation

The evaluation concentrates on testing whether the parallel method is an efficient and effective method for finding sets of similar solutions for a set of related problem specifications. If this is the case, then the parallel method can be used to support hypothesis statement 1 from section 3.2.1 and meet requirement Req. 2 of section 3.1.2 which states the need for a task allocation tool to support multi-moded systems.

The parallel method is compared to the sequential method for two mode systems. The sequential method can also be applied in situations requiring the minimisation of changes to an existing solution in order to meet new requirements. The need for this was set out in requirement Req. 1 of section 3.1.1. The suitability of search for this requirement is another aspect of hypothesis statement 1.

The simultaneous method is not included in this evaluation based on the analysis in section 6.3.4. Previous work [182] contains some results which confirm that the simultaneous method is good at finding a single solution for all modes but is not practical in cases where this is not possible.

6.4.1 Selection Of Problem Specifications

The problem specifications used were taken from a previous experiment, namely experiment 5.8. This experiment contained 32 classes of problem with 10 instances in each class. The problem characteristic parameter values which were provided to the Gentap tool to generate these problems are given in tables 5.10 and 5.11. All of the problems contain 48 tasks.

These 32 classes were then grouped into 8 higher level classes each containing 4 subclasses. A separate guidance heuristic was tuned for each of the 8 classes in experiment 5.6. These classes are shown in table 6.7. From the results in experiment 5.8, valid configurations were found for the majority of problems in each class other than Class 4. The problems used in this evaluation were taken from one subclass of each of the remaining 7 classes. With reference to table 5.11, the subclass chosen from

each class had low values of both the processor-connectivity and utilisation-per-network parameters.

6.4.2 Experiment 6.1 — Balancing Objectives

Since the guidance heuristic was tuned for each problem class in previous experiments this is not repeated. However, the cost function hierarchy, shown in figure 6.2 now has three top level objectives: schedulability, guidance and change. These must be balanced appropriately. There is no guarantee that the lower level weightings of the guidance heuristic subfunctions won't interact with these higher level weightings. It has been a theme of this thesis, however, that the method should take a practical approach to SBSE. Using the hierarchical weighting scheme and only retuning top level weightings is consistent with this approach.

As described in section 5.3.4, previous experiments aimed to improve performance of the algorithm and used the number of configurations evaluated as a response variable. The situation in this experiment is different. The aim is to select a balance of objectives which best minimises the differences between configurations produced at the end of the search. The response variable used is the total proportion of allocation changes as given by $g_{tac} + g_{mac}$. Priority changes are not included since they only become significant once allocation changes are removed. If a search is successful at reducing allocation changes, the reduction of priority changes should follow.

Note that the response value is used for analysis of results only. Within the *change* function itself, weightings need to be chosen for balancing allocation changes and priority changes. The values chosen were $w_{tac} = w_{mac} = 100$ and $w_{tpc} = w_{mpc} = 10$. If corresponding schedulers only have 0 or 1 allocations in common then the priority change functions will return lower values due to equation (6.3). Allocation changes need to be penalised more than priority changes so that allocation changes are not used as a device for reducing priority changes.

A slight alteration was made to the calculation for measuring priority changes. In equations (6.4) and (6.5), q was replaced with $q' = 1 - \exp(-5 \cdot q)$. The reason for doing this was to provide a greater differential in cost values when there were few changes to priority ordering. This happens much more often during the search than the opposite extreme where the priority ordering of one configurations is the reverse of the other.

Separate experiments were conducted for balancing objectives in the sequential and parallel methods since there is no reason why weighting values chosen for one should work well with the other.

6.4.2.1 Balancing Objectives For Sequential Method

As previously stated, problems were selected from 1 subclass of the 7 problem classes used for this evaluation. 10 problems were available in each class and valid configu-

rations had been found for the majority of these in experiment (5.8). The sequential method was tested by using one of the existing valid configurations as a baseline and evolving it into a solution for a different problem within the same class. To follow the terminology of section 6.3.3, the existing valid configuration represents configuration $\mathcal{C}_1$ for *Mode 1* and the other problem picked from the class represents the specification for *Mode 2*.

As stated in section 6.3.3, $\mathcal{C}_1$ is expected to be a good starting point for the solution to *Mode 2* and therefore the search is run with a lower initial temperature and is cooled slowly. This causes more intensive investigation of configurations similar to $\mathcal{C}_1$. In these experiments, the following parameters were used: inittemp $= 0.001$, maxinner $= 20000$. The maximum number of moves allowed for the search was 500000.

A 4 level, 3 factor mixture design with an additional centroid point was used to set the levels of the weightings w_{sched}, $w_{guidance}$ and w_{change}. The constant sum of these weightings was arbitrarily chosen to be 1200 and the minimum level allowed was 10. This resulted in 16 runs for each of the 7 problems used; 112 in total. See section 5.3.3 for an explanation of mixture designs.

As already stated, the response used for analysis was the value of $g_{tac} + g_{mac}$ at the end of a run. If no valid configuration was found for *Mode 2* then the response value was set to 2.

Unlike previous experiments in section 5.4, no censoring of the response is taking place and so there is no need to use survival analysis. Standard linear regression techniques are appropriate for building a model of the data. After building such a model and finding an optimal point on it, the suggested balance of weighting values was $w_{sched} = 559$, $w_{guidance} = 631$ and $w_{change} = 10$.

After learning that this point lay in a good area of the design space, a second set of runs were performed around this point to refine the result further. This used a 3 level, 3 factor mixture with additional interior points (similar to experiment 5.6 without the centroid) requiring 13 runs per problem. In this experiment, the following minimum values were used to constrain the mixture: $w_{sched} \geq 500$, $w_{guidance} \geq 500$ and $w_{change} \geq 10$. Refer to section 5.3.3 for information on constrained mixture designs. Analysis of results from this second set of runs recommended a weighting combination of $w_{sched} = 690$, $w_{guidance} = 500$ and $w_{change} = 10$.

In contrast to the results of experiment 5.7 the guidance heuristic is given a lower weighting than the schedulability objective. This is expected; the initial configuration, $\mathcal{C}_1$, should be close to good solutions and so there is less need for a guidance heuristic. This result is also in agreement with previous results published by this author [164].

6.4.2.2 Balancing Objectives For Parallel Method

The same pairs of problem specifications are used for the parallel method as with the sequential method. The parallel method is simultaneously finding new configurations for both modes. No configurations found in previous experiments are involved.

The search parameters found in experiment 5.3 are reused with inittemp = 0.005 and maxinner = 1500.

As with the experiments for the sequential method, 2 sets of runs using mixture designs were carried to first find a good area of the design space and then refine the result.

The results of the first set of runs recommended the design point $w_{sched} = 620$, $w_{guidance} = 570$, $w_{change} = 10$. After refinement, this was adjusted to $w_{sched} = 500$, $w_{guidance} = 580$, $w_{change} = 120$. Consistent with the results of experiment 5.7, more emphasis is put on guidance over penalising missed schedulability constraints when finding new configurations from a random starting point.

These results recommend a much higher weighting for the change subfunction in the parallel method than in the sequential method. In the sequential method, increasing this weight leads to the search becoming trapped near the initial configuration and not able to find a valid configuration. The use of low temperatures also makes it more difficult for the sequential method search to move away from the starting point which complements the aims of the change subfunction. Neither of these factors are relevant to the parallel method so the higher weighting value for the change subfunction is sensible.

6.4.3 Experiment 6.2 — Comparison Of Methods For Two Mode Problems

Based on the analysis in section 6.3.3, it is suspected that the parallel method will outperform the sequential method in finding a pair of configurations with fewer differences for a pair of problem specifications. This hypothesis is tested in this section.

Since a fair comparison of priority ordering changes is only possible when all allocation changes are removed, the hypothesis will only be tested with regards to allocation changes. For a sequence of test runs, let $ac_i = g_{tac_i} + g_{mac_i}$ where g_{tac_i} and g_{mac_i} are the values of g_{tac} and g_{mac} for the final result produced by run i. The null hypothesis is

$$H_0 : \mu_{seq} = \mu_{par} \tag{6.15}$$

where μ_{seq} is the population mean for values of ac_i produced by the sequential method and μ_{par} is the equivalent population mean for the parallel method.

This experiment used the same specifications taken from 7 problem classes as were used in experiment 6.1. Each method was applied for 3 repetitions and the result with the lowest total of allocation changes was selected. The sequential method used the same baseline configuration in all 3 repetitions.

Class	g_{tac}	g_{mac}	g_{tpc}	g_{mpc}	$g_{tac} + g_{mac}$
Class 1	0.0417	0.0750	0.5168	0.2896	0.1167
Class 2	0.1458	0.2453	0.4594	0.4612	0.3911
Class 3	0.1042	0.0968	0.3738	0.1912	0.2009
Class 5	0.0833	0.1250	0.9571	0.5737	0.2083
Class 6	0.0208	0.0612	0.9436	0.5013	0.0821
Class 7	0.0208	0.0256	0.5794	0.5057	0.0465
Class 8	0.0000	0.0667	0.8034	0.3209	0.0667

Table 6.8: Values of change metrics for configuration pairs produced by sequential method

Class	g_{tac}	g_{mac}	g_{tpc}	g_{mpc}	$g_{tac} + g_{mac}$
Class 1	0.0000	0.0000	0.0000	0.0000	0.0000
Class 2	0.0000	0.0000	0.0000	0.0000	0.0000
Class 3	0.1250	0.0323	0.2700	0.2226	0.1573
Class 5	0.0000	0.0000	0.6032	0.3559	0.0000
Class 6	0.0000	0.0000	0.8087	0.3114	0.0000
Class 7	0.0000	0.0000	0.0000	0.0000	0.0000
Class 8	0.0000	0.0000	0.0000	0.0000	0.0000

Table 6.9: Values of change metrics for configuration pairs produced by parallel method

The sequential method produced a valid configuration for *Mode 2* on 20 of the 21 runs. The values of the change subfunctions for repetitions with fewest allocation changes are given in table 6.8.

The equivalent set of results for the parallel method is shown in table 6.9. Using this method, only 18 of the 21 repetitions produced a valid configuration for both modes. However, a valid pair of configurations was found for every problem class within the 3 repetitions.

As shown in table 6.9, the parallel method was able to find a perfect solution with no changes for 4 of the 7 problem classes. Only problem class 3 required any allocation changes. This problem class also required the greatest number of evaluations in experiment 5.8 (table 5.20). This suggests that there are fewer valid configurations in the solution space for problems in this class which could also be why it is also more difficult to find a pair of nearby solutions.

Applying a paired exact Wilcox test to the final columns of tables 6.8 and 6.9 returns a p-value of 0.000156 given the assumption that the null hypothesis in equation (6.15) is true. The null hypothesis is rejected at the 1% level indicating that the performance of the two methods is different.

Table 6.10 shows the allocation differences between the configurations previously produced for the same specifications in experiment 5.8. No attempt was made to minimise differences between the configurations in this experiment. A comparison with other results in this section is unfair since they were taken from the best of 3 repetitions and these are from a single repetition. However, the results do suggest that not attempting to minimise differences between configurations will lead to a

Class	g_{tac}	g_{mac}
Class 1	0.8542	0.8500
Class 2	0.7708	0.8491
Class 3	0.9792	0.9355
Class 5	0.6250	0.7250
Class 6	0.8542	0.8980
Class 7	0.5833	0.7179
Class 8	1.0000	1.0000

Table 6.10: Allocation changes when no attempt is made to minimise differences. Results from selected pairs of configurations found in experiment 5.8

large amount of change being required.

6.4.3.1 Performance Of Parallel And Sequential Methods

During the test runs, the sequential method averaged a rate of 253 evaluations per second. This is consistent with the results for task sets containing 48 tasks shown in figure 5.16 from experiment 5.9. Therefore, the schedulability test is still dominating the rate of evaluation and the added cost subfunctions for minimising change have little effect on performance.

$5 \cdot 10^5$ evaluations were used per repetition and three repetitions took approximately 1.5 hours on average. The time taken for the sequential method should also take account of the time needed to generate the initial solution. From the results in table 5.20, this requires in the region of 10^4 to 10^5 evaluations. Therefore, the total time needed to apply the sequential method to these problems is ≈ 2 hours.

Using a dual core machine, the parallel method with two threads operated at an average rate of 166 evaluations per second within each thread. The reduction in speed is due to the fact that each f_{par} evaluation needs two f_{seq} evaluations. The reduction isn't greater since many values can be cached as previously explained in section 6.3.6.2.

To perform $5 \cdot 10^5$ evaluations in each thread requires $\approx$ 1 hour or $\approx$ 3 hours for 3 repetitions. Obviously, this time would approximately double for a single core machine. If a valid solution with no changes between configurations was found, the method terminated early since no better solution could be found.

Section 6.3.6.3 explained how the repeatable synchronisation mechanism of the parallel method can waste some configuration evaluations. Of the 21 runs of the parallel method, 12 did not terminate early. These 12 runs should have performed $2 \cdot 5 \cdot 10^5$ evaluations each. On average they actually performed 1039821. This means that approximately 4% of evaluations were wasted.

For two mode systems, the sequential and parallel method take a similar amount of time with the sequential method being slightly quicker. The parallel method evaluates solutions slower and relies on the availability of multi-core computing for its performance. The sequential method is unable to take advantage of more cores.

Class	g_{tac}	g_{mac}	g_{tpc}	g_{mpc}
Class 1	0.0000	0.0000	0.0000	0.0000
Class 2	0.0000	0.0000	0.2962	0.2826
Class 3	*	*	*	*
Class 5	0.0000	0.0000	0.6977	0.5123
Class 6	0.2708	0.2628	0.7455	0.5570
Class 7	0.0000	0.0000	0.0000	0.0000
Class 8	0.0000	0.0000	0.1917	0.1857

Table 6.11: Values of change metrics for 4 mode configurations produced by parallel method

6.4.4 Experiment 6.3 — Parallel Method With More Than Two Modes

This experiment applies the parallel method to 4 mode systems and an 8 mode example. The problem specifications were taken from the same problem classes selecting the required number of specifications from the 10 available.

The first attempt at applying the parallel method to 4 mode systems used the same weightings as for the 2 mode experiment. Valid schedulable solutions were not found on several occasions indicating the w_{change} weighting value was too high. A new set of values was calculated by reducing this value and scaling the others up in proportion. This resulted in a new set of weightings for balancing objectives: $w_{sched} = 546$, $w_{guidance} = 634$, $w_{change} = 20$.

Once again, 3 repetitions for each of the 7 problem classes were run. Using the new weightings, 16 of the 21 runs found a valid solution where all 4 modes were schedulable. The best result from each of the three repetitions is shown in table 6.11. For class 3, none of the 3 repetitions found a valid solution. Only the solution for class 6 required any allocation changes between the 4 modes.

A repeat of the experiment for class 6 was performed with more emphasis on minimising change. The weightings used were: $w_{sched} = 463$, $w_{guidance} = 537$, $w_{change} = 200$. From three repetitions, this produced a best result of $g_{tac} = 0.0000$, $g_{mac} = 0.0000$, $g_{tpc} = 0.4896$, $g_{mpc} = 0.4691$.

Efforts were also made to find a solution for class 3 by lowering the weighting for minimising change further. A solution was eventually found with the weightings $w_{sched} = 555$, $w_{guidance} = 644$, $w_{change} = 1$. With these weightings, the algorithm is close to solving each mode completely independently without any minimisation of change. The best resulting values for the change subfunctions were $g_{tac} = 0.7847$, $g_{mac} = 0.8174$, $g_{tpc} = 0.0735$, $g_{mpc} = 0.0460$.

A single repetition of an 8 mode problem taken from class 1 was also solved using the parallel algorithm. This used the same parameter setup as that which produced the results for 4 modes in table 6.11. The results from this run were $g_{tac} = 0.0000$, $g_{mac} = 0.0214$, $g_{tpc} = 0.2284$, $g_{mpc} = 0.2262$.

6.4.4.1 Performance Of Parallel Method With More Modes

The experiments for 4 and 8 mode systems were run on an 8 core machine so all threads could run in parallel. For the set of 21 runs of the parallel method on 4 mode systems, the algorithm averaged 87 evaluations per second per thread. At this rate, $5 \cdot 10^5$ evaluations takes 1.6 hours or ≈ 5 hours for 3 repetitions. This is a significant decrease on the rate of evaluation for the two mode system which averaged 166 evaluations per second. After some investigation, it was found that the implementation was not caching differences between unchanged configurations in the calculation of f_{seq}. This is compounded by the quadratic increase in the number of pairs of modes to measure differences between.

The single run of the 8 mode system ran at a speed of only 15 evaluations per second, taking over 9 hours to complete. The need to evaluate increasing numbers of configurations inside each thread clearly has an impact on performance for systems with many modes. As mentioned above, better caching of intermediate results should improve this situation.

For the 4 mode systems, 19 of the 21 runs did not terminate early. These should have used $2 \cdot 10^6$ evaluations in total. The mean actually used was 2251985, about 13% wastage. This is higher than the value found for 2 mode systems which could be due to the machine running the experiments not giving equal resources to each thread because of contention with other processes.

The 8 mode system required 4203168 evaluations though the maximum set was $8 \cdot 5 \cdot 10^5 = 4 \cdot 10^6$. Therefore $\approx 5\%$ of evaluations were wasted.

6.5 Summary And Further Work

The main aim of the work in this chapter is to meet requirements Req. 1 and Req. 2 from section 3.1 and to test hypothesis statement 1 from section 3.2.1. While carrying out the work other contributions were made and questions were raised which prompt suggestions for further work.

6.5.1 Achievement Of Goals

Req. 1 stated that a method was needed to minimise differences between an existing configuration and one produced for new requirements. The sequential method, described in section 6.3.3, was developed for this purpose. It is based on adding a new subfunction to the cost function which penalises changes from an existing configuration.

Req. 2 stated that a method was needed for implementing systems with multiple modes where the differences between configurations for each mode must be minimised. In section 6.3.1, three algorithms were proposed for this purpose: the sequential method, simultaneous method and parallel method. Analysis of each

method suggested that the parallel method was most likely to succeed at minimising change and most practical for systems with several modes.

Experiment 6.2 was performed to test the hypothesis that the parallel method would outperform the sequential method with respect to minimising change between configurations for two mode systems. The results confirmed the hypothesis. Experiment 6.3 ran the parallel method on 4 and 8 mode problems where it was once again successfully able to reduce, and in some cases totally remove, allocation and priority ordering changes.

Hypothesis statement 1 questioned the efficiency and effectiveness of local search for achieving requirements Req. 1 and Req. 2. Experiments 6.2 and 6.3 demonstrated the effectiveness of the sequential method for achieving Req. 1 and the parallel method for achieving Req. 2.

In experiment 6.2, the sequential method was shown to evaluate solutions at the same rate as the standard task allocation local search algorithm used in experiment 5.9 where minimising change was not an objective. The time taken to calculate the new cost subfunctions for penalising change between a single pair of configurations is negligible compared to the time required to run the schedulability test.

The additional constraint of remaining near to an existing solution does mean that the searches using the sequential method require more evaluations to find a solution that has few changes and is schedulable. Previously, in experiment 5.8, the search took in the region of 10^4 to 10^5 evaluations to find a valid solution. In these experiments $5 \cdot 10^5$ evaluations were required. The additional constraints also made the search less reliable at finding a valid schedulable solution and so the best result from 3 was used. In total, a good solution that was schedulable and with few changes from the original could be found in 2 hours on machines clocked at 2.66GHz. This is well within the amount of time an engineer could expect a tool to take to produce a solution and therefore hypothesis statement 1 is supported with regards to local search being suitable to meet Req. 1.

The parallel method requires more resources than the sequential method and relies on the availability of multi-core machines for its efficiency. On an 8 core machine, 8 very similar configurations for 48 task systems were produced comfortably within 12 hours. For 4 mode systems, 3 repetitions of the algorithm were possible in under 5 hours. These durations are felt to be within bounds of what is reasonable and so hypothesis statement 1 is also supported with regards to local search being able to achieve Req. 2. Experiment 6.3 uncovered possible improvements to the implementation of the parallel method which may substantially improve performance with several modes.

As shown previously in experiments 5.6 and 5.9 the number of evaluations required and speed of the schedulability test are both heavily dependent on the characteristics of the task allocation problem. This is still very much the case with regards

to finding solutions with few changes between them. Reflecting on experiment 5.8, the results of experiments 6.2 and 6.3 suggest that problem classes whose instances are harder to solve also contain problem instances for which it is harder to find valid configurations which are similar to each other.

6.5.2 Other Contributions

The repeatable parallel algorithm used in the parallel method in this chapter could have more general applicability. The framework could be applied to any problem where it is necessary to find similar solutions to a set of related problems. An example is product line design where several similar products have slight feature variations.

The parallel method has been built around a simulated annealing algorithm. The same framework could be used with other heuristic search algorithms, including population based algorithms. There are three requirements for using the parallel method framework: being able to store and restore state, maintaining a solution which is currently considered to be the best and being able to measure the difference between a pair of solutions.

6.5.3 Further Work

It is unlikely that all mode transitions will be possible when a system is operational or some may occur more frequently than others. When measuring differences between pairs of solutions using equations (6.6) to (6.9) it would be possible to use a weighted mean to place more bias on some transitions. However, this may not always be a useful feature to have. If transitions *Mode 1* $\rightarrow$ *Mode 2* and *Mode 1* $\rightarrow$ *Mode 3* are important then this will easier to achieve if the configurations for *Mode 2* and *Mode 3* are similar even if *Mode 2* $\rightarrow$ *Mode 3* is not an important transition.

The parallel method applies the same cost function (f_{seq}) to each part of the configuration with the assumption that the characteristics of each problem specification are similar. If each mode had very different characteristics then it would be desirable to use different guidance heuristics for finding each section of the overall configuration for all modes. However, for situations where related configurations are required, it is likely that there will also be a degree of similarity between problem specifications.

7

Producing Flexible Solutions Using Scenarios

7.1 Introduction

Flexibility is an important architectural quality for software based systems because they rarely remain the same if they continue to be used. Section 1.1, which covered the subject of software maintenance, gave some reasons why this is the case:

- correction of implementation errors,
- iterative development driven by changing customer requirements,
- obsolescence of hardware or software dependencies,
- change of operating environment.

Other than the first of these, the need to change the system is being caused by another external change, termed a change agent in section 1.1.3.

The technique of scenario based analysis, described in section 2.4.3, is used in the development of systems' architectures. Scenario based analysis asks questions of the form, "If X changes, what will happen to Y?". The answers to these questions are discovered by applying scenario X to a model of the architecture and then using simulation or theoretical analysis to learn how Y will react. If the answer is not satisfactory, then the architecture must be modified.

It is hard to predict which scenarios will actually occur. Often, many scenarios will be suggested and it will be impossible to accommodate them all given the available resources of the project at the time. In this situation, attempts are made to design an architecture which can be easily changed to continue to meet requirements if scenario X does come about. Scenarios can also be wrong. An external change may occur which was not predicted. However, the modifications made to the archi-

tecture to accommodate scenarios which were suggested may simplify adjusting the architecture in future for other external changes.

The ease with which an architecture can change to meet the needs of an external change, predicted by a scenario or otherwise, is its flexibility by the definition given in section 1.1.3. By using scenarios in the process of selecting a task allocation, it is hoped that the selected configuration will be more flexible to both changes predicted by the scenarios as well as those which are not.

Requirement Req. 3 of section 3.1.3 states that task allocation tools should take account of possible future changes, i.e. scenarios, when selecting a configuration for a set of tasks and messages. This requirement is covered by the work in this chapter.

Hypothesis statement 3 from section 3.2.1 says that configurations which have been generated with a set of scenarios will also be more flexible with respect to unpredicted changes which are different from though have some similarities with the scenarios. This statement is tested during the evaluation in this chapter.

This chapter now proceeds as follows. Section 7.2 describes how scenarios can be involved in the selection of a configuration with the use of the parallel method developed in chapter 6. Section 7.3 describes a series of experiments to test hypothesis statement 3 above. Section 7.4 summarises the findings and suggests avenues for further work.

7.2 Incorporating Scenarios Into The Search Algorithm

When using scenarios to produce a task and message configuration, the configuration should:

- meet the requirements of as many scenarios as possible,
- need as few adjustments as possible to meet the requirements of other scenarios.

All of the algorithm features required to meet these requirements were developed in chapters 5 and 6. In particular, the parallel method, described in section 6.3.6, can be applied in this context.

The parallel method algorithm was originally designed to accept a set of problem specifications which represented the run-time modes of a system and produce a corresponding set of configurations which met the needs of each mode but had as few differences between them as possible. This allowed the system to transition between modes efficiently. As recognised in section 3.1.3 the efficient mode transition optimisation problem is of the same form as the one which needs to be solved to meet the requirements listed above.

The algorithm can be supplied with a set of problem specifications which contain the current requirements of the system and additional requirements which represent possible change scenarios. The algorithm will then search for a set of valid configurations which have as few differences between them as possible and reuse the same

configuration where possible. One significant difference is that a valid schedulable solution is only needed for the current problem specification. An extreme scenario specification may be impossible to satisfy yet still guide the search to produce a more flexible configuration for the current specification.

The configuration produced for the current problem specification as opposed to scenario specifications is referred to as the *baseline configuration*. At some point in the future, it is assumed that new requirements will cause the baseline configuration to be reevaluated and possibly changed. These new requirements are referred to as an *upgrade specification*.

The fewer changes which need to be made to the baseline configuration to be valid in the context of an upgrade specification, the more flexible the baseline is said to be in the context of that upgrade specification. Finding a new configuration which differs from a previous one as little as possible can be done by applying the sequential method, described in section 6.3.3. The application of the sequential method for this purpose satisfies requirement Req. 1 of section 3.1.1 on the reuse of solutions.

7.3 Evaluation

The search algorithm for generating flexible baseline solutions makes the assumption that if a configuration for a problem specification meets all or nearly all of the requirements of a scenario, the solution produced will be more flexible with respect to changes of a similar nature to the scenario. This assumption is tested in this evaluation.

The results presented in this section are taken from a previous paper [164]. The set of problem specifications used are different from those in chapters 5 and 6. They were generated with a slightly older version of the Gentap tool than that described in chapter 4. The only major difference is that message sizes were a constant value rather than being sampled from a distribution for a particular network utilisation level.

Parameter tuning was conducted in the same manner as was done in chapters 5 and 6 with different weighting values being used for different problem characteristics. The low level cost subfunctions were the same though the hierarchical organisation has since been improved to the structures which were given in figures 5.6 and 6.2. Full details of the parameter tuning undertaken for this evaluation are available in the original paper [164].

7.3.1 Problem Generation

Three types of problem specification need to be generated for the evaluation: baseline problem specifications, scenario specifications and upgrade specifications.

Problem Specification	Parameter Values			
	utilisation-per-processor	tasks-per-processor	messages-per-task	period max / period min
01	0.40	5	1	10 (5.9)
02	0.65	5	1	10 (5.0)
03	0.40	8	1	10 (2.3)
04	0.65	8	1	10 (6.4)
05	0.40	5	2	10 (5.9)
06	0.65	5	2	10 (8.6)
07	0.40	8	2	10 (9.6)
08	0.65	8	2	10 (6.0)
09	0.40	5	1	1000 (30.8)
10	0.65	5	1	1000 (62.0)
11	0.40	8	1	1000 (39.3)
12	0.65	8	1	1000 (81.3)
13	0.40	5	2	1000 (110.6)
14	0.65	5	2	1000 (367.8)
15	0.40	8	2	1000 (14.9)
16	0.65	8	2	1000 (127.2)

Table 7.1: Values of varied problem characteristics

7.3.1.1 Baseline Problem Specifications

Baseline specifications were generated using Gentap, the problem generator described in chapter 4. The parameters used are shown in tables 7.1 and 7.2. Four problem characteristics were varied between high and low values to create the 16 problem classes listed in table 7.1. These problem characteristics were the mean utilisation per processor, mean tasks per processor, number of messages per task and period range. The final column of table 7.1 gives the maximum possible ratio of periods for the parameter values used and the actual ratio for that problem in parentheses. The remaining Gentap parameter values, which were kept constant, are shown in table 7.2.

Parameter	Fixed Value
number-of-tasks	40
number-of-networks	1
processor-connectivity	1 (broadcast network)
network-bandwidth	2048
network-latency	0
processor-network-bandwidth	102400
processor-network-latency	0
utilisation-per-network	approx 0.2 (not a Gentap parameter at time of evaluation)
tasks-per-transaction	10 (implies 4 transactions)
transaction-length	0.4
transaction-utilisation-distribution	equal

Table 7.2: Fixed problem characteristic values

Scenario	Num Scenarios	Percentage Tasks Changed	Utilisation Increase
noscen	no scenarios used	–	–
scen1	1 scenario	40%	4%
scen2	1 scenario	40%	11%
scen3	1 scenario	40%	18%
scen4	1 scenario	40%	25%
scen5	1 scenario	20%	25%
scen6	1 scenario	60%	25%
scen7	3 scenarios	40%	11%

Table 7.3: Scenario profiles

A single problem instance from each class was generated. Each of these problems were solved without using any scenarios to check the baselines were feasible.

The scenario and upgrade specifications were created by making random but controlled alterations to these baseline specifications. The nature of these alterations is described in sections 7.3.1.2 and 7.3.1.3.

7.3.1.2 Scenario Specifications

The profiles of the scenarios are shown in table 7.3. For each scenario profile, problem specifications were generated by selecting a certain percentage of tasks in the baseline at random and increasing their utilisation to cause the total utilisation of all tasks to increase by the amount indicated.

The profile labels have the following meanings. *noscen* corresponds to generating a baseline without the use of scenarios. Scenarios in profiles *scen1*, . . . , *scen4* were generated by randomly selecting an equal number of tasks from each transaction so that 40% of the systems' tasks were changed to achieve the utilisation increase levels shown in table 7.3. Scenario profiles *scen5* and *scen6* change different proportions of tasks and *scen7* uses three separate scenarios, each generated in the same manner as those from *scen2*.

Three instances of each scenario profile were generated for each baseline which needed to be generated, i.e. three instances of a single scenario for *scen1*, . . . , *scen6* and three sets of three scenarios for *scen7*.

7.3.1.3 Upgrade Specifications

Upgrade specifications were also generated with an element of randomness and were different from the predicted scenarios which were generated in section 7.3.1.2. The upgrades were limited to utilisation increases where the increase was spread evenly between transactions.

The 5 levels of utilisation increase are given in table 7.4. Initially the utilisation increases used were the same for all problems. However, when large utilisation

Utilisation Increase Level	Actual Utilisation Increase	
	Low Starting Utilisation	High Starting Utilisation
1	4%	2%
2	8%	4%
3	12%	6%
4	16%	8%
5	20%	10%

Table 7.4: Utilisation increase levels for upgrade specifications

increases were applied to some problems which already had a higher starting utilisation, it was not possible to find valid configurations for the upgrade specifications. Therefore, smaller utilisation increases were made to the baselines which have a higher initial utilisation according to table 7.1. For each utilisation increase level and each problem, 3 separate upgrades were generated to be used for repetitions in experiments. There were $3 \cdot 5 \cdot 16 = 240$ upgrade specifications generated in total.

Upgrade specifications were checked to be feasible by searching for a valid configuration for each of them without regard for any existing baseline. As a reference for future experiments, the configurations found during these checks had a mean proportion of task allocation changes of 0.842 from the configurations found when checking the feasibility of the baselines. Given that there are several valid configurations for each baseline and upgrade specification, this high value is not surprising.

7.3.2 Experiment 7.1 — Using A Single Utilisation Increase Scenario

Testing the effectiveness of the different scenario profiles is a 2 stage procedure:

1. Generate baselines configurations for each problem using the different scenario profiles
2. Use the sequential method to upgrade these baselines. The resulting number of changes required is used as a measurement of the baseline's flexibility.

In these experiments only task, and not message, allocation and priority changes were considered.

In this experiment, 3 baselines were generated for each problem using each of the 5 scenario profiles *noscen*, *scen1*, *scen2*, *scen3* and *scen4*. Each baseline was upgraded 5 times for the different types of upgrade specification listed in section 7.3.1.3. Therefore, across all 16 problems, 240 upgrade specifications were generated per scenario profile.

The combined task allocation changes for all problems are shown for each of the scenario profiles in figure 7.1. In the graph sections of each bar are shaded differently to show the proportion of changes contributed by each upgrade type.

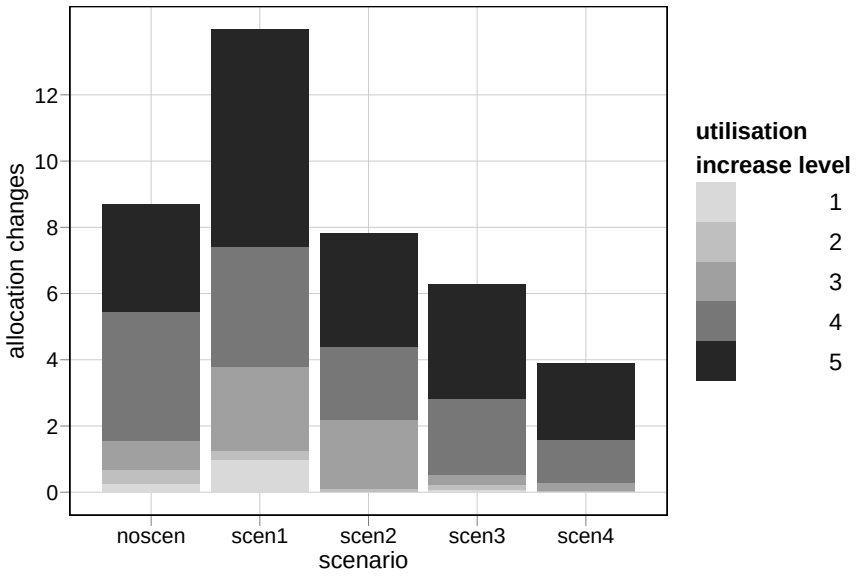


Figure 7.1: Allocation changes required to meet upgrade requirements

The first point of note is that the baselines generated without using scenarios are more flexible than may be expected. This can be explained by the fact that the sequential method parameter tuning experiments to find weightings which best minimise change between baseline and upgrade configurations used these baselines. This puts the new baselines generated with scenarios at a slight disadvantage. For the other solutions, increasing the size of utilisation increase in the scenario gradually improves the flexibility of the system. For *scen4*, nearly all allocation changes for upgrades with lower level utilisation increases have been eliminated and allocation changes required for higher level utilisation increases have been reduced.

Figure 7.2 breaks down the results of figure 7.1 by problem characteristic. This shows a very clear pattern. The problems which have the least flexible solutions by far are those with a high utilisation and a low number of tasks per processor. In a problem with a lower number of tasks per processor, each task is using a larger chunk of utilisation on average and so it is more difficult to fully utilise the available resources of each processor. One problem in particular requires more changes but there is insufficient data within a single cell to draw conclusions about its problem class.

Because of the way cost subfunctions which penalise change, given in section 6.3.2, are calculated, priority changes can only be compared once allocation changes have been removed. Figure 7.3 shows the priority changes required to meet the upgrade specifications but only for the 12 systems which were shown in figure 7.2 to have negligible allocation changes. This graph once again shows that using scenarios which stress the baseline configuration with larger utilisation increases reduce the number of changes more than those with smaller increases.

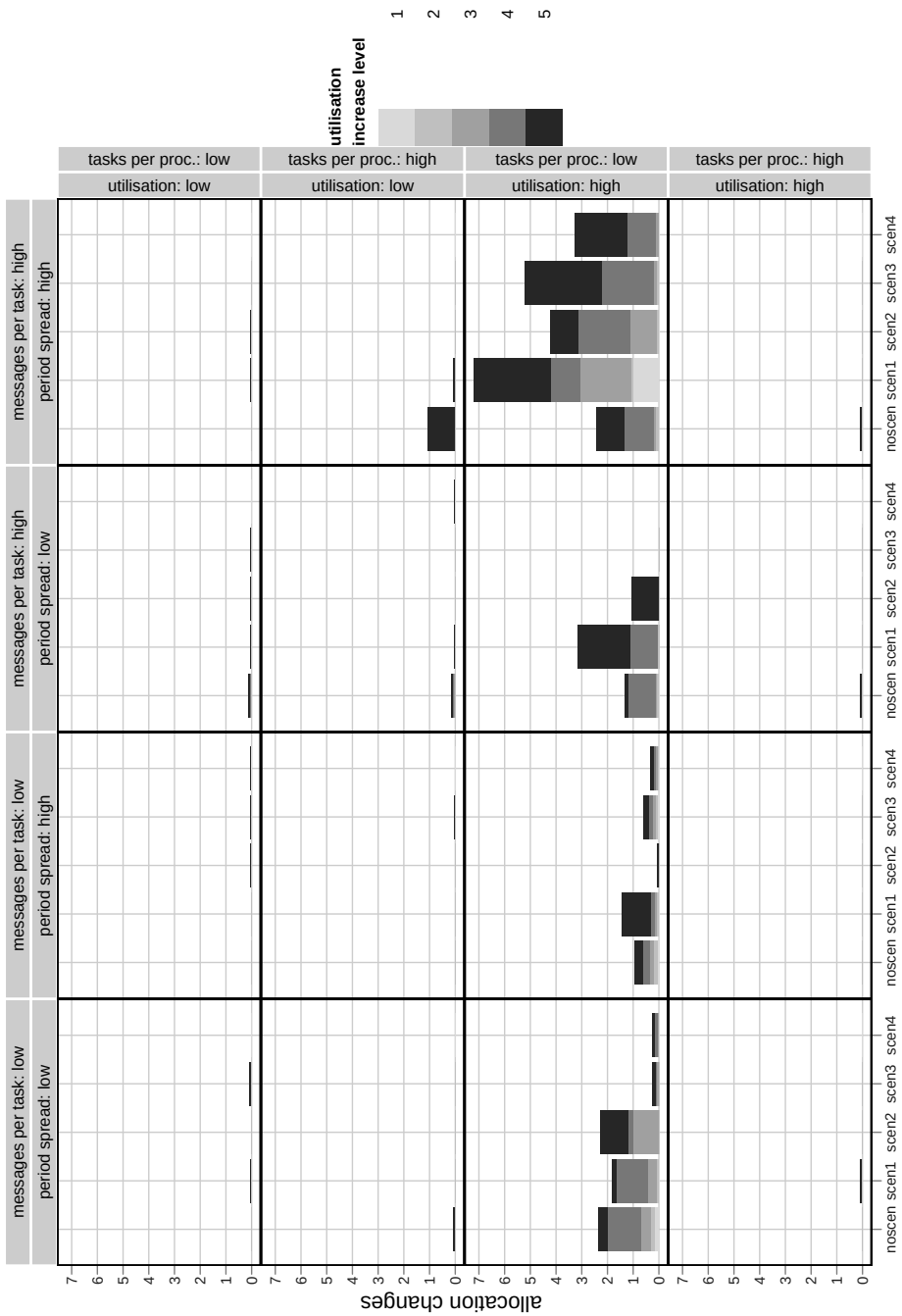


Figure 7.2: Changes required for upgrades for each problem class

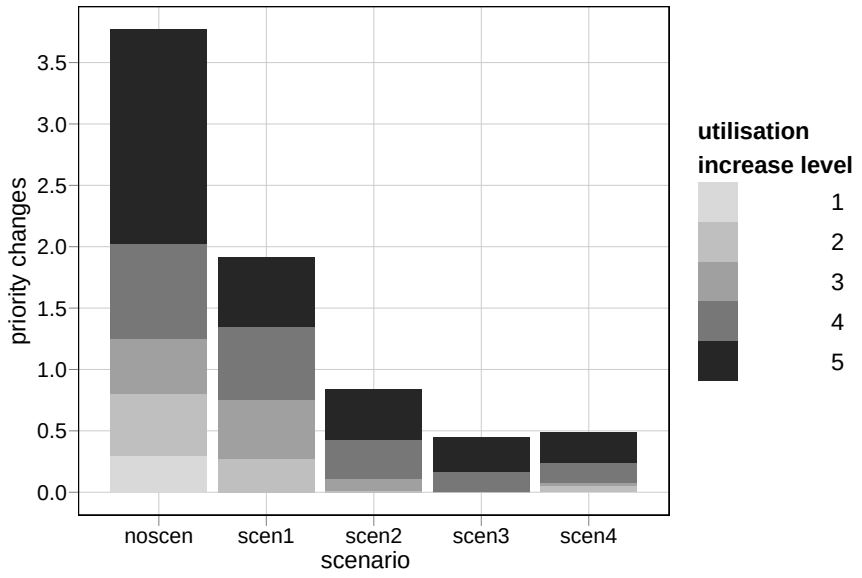


Figure 7.3: Priority changes required for upgrades

7.3.3 Experiment 7.2 — Using Scenarios With Other Stressing Patterns

For the four systems which had a high per processor utilisation and low number of tasks per processor, further baselines were created using scenario profiles *scen5*, *scen6*, and *scen7*. The flexibility of the baselines generated for just these four systems is compared in figure 7.4.

Scenario profile *scen5* increased utilisation per processor by the same amount as *scen4* but concentrated the change amongst fewer tasks. In general, if it were known which tasks would change, then a more targeted scenario makes sense but for the style of upgrade tested here, these scenarios do not perform well.

Scenario profile *scen6* does the opposite, spreading the utilisation increase over a larger number of tasks. This style of scenario actually generates the most flexible baselines for these upgrades showing that diluting the utilisation increase over more tasks does not have a negative effect. Finally, *scen7* represents a combination of 3 scenarios each with the same characteristics as *scen2*. It has been shown that increasing the amount of stress a scenario applies via a larger utilisation increase improves flexibility. An alternative way of increasing the stress is to use multiple scenarios with smaller utilisation increases. A comparison between baselines generated with *scen2* and *scen7* validates this statement.

7.4 Summary And Further Work

In this chapter, the parallel method, developed in chapter 6, was used to generate valid baseline configurations which were similar to configurations for possible change scenarios. By doing this, when an upgrade is required, the baseline configuration should require fewer changes if there is some similarity between the upgrade and

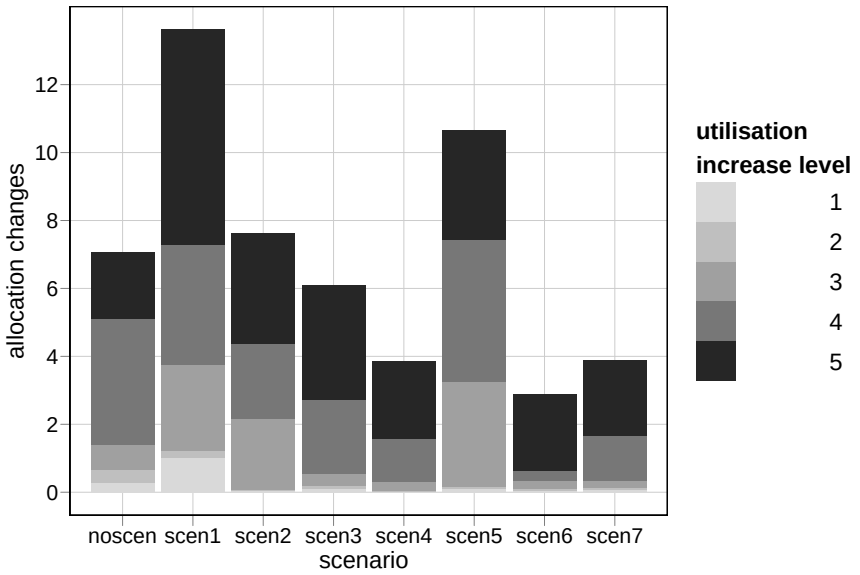


Figure 7.4: Results of additional scenario evaluations

scenarios used.

When an upgrade needs to be performed, the sequential method, also developed in chapter 6, can be used to find a new configuration as close to the baseline as possible.

The primary aim of the work in this chapter was to test hypothesis statement 3 from section 3.2.1. This said that baseline configurations generated with scenarios would be more flexible than those that weren't even when the scenarios did not correctly predict changes. The results of experiments 7.1 and 7.2 have clearly shown that baselines generated with increasingly more stressful scenarios require fewer allocation and priority ordering changes when they are upgraded.

Four problem characteristics were studied throughout this chapter. Problems with a high mean processor utilisation and low mean number of tasks per processor proved to be the most difficult to generate flexible baselines for. The range of task periods and the number of messages per task did not have a significant impact.

7.4.1 Further Work

Since the same algorithms are used in this chapter as in chapter 6, issues raised in section 6.5.3 are still relevant. The weightings used for solving the current problem specification and scenario specifications are the same. Allowing these to differ is an implementation, rather than theoretical, issue and this issue did not cause any problems for these experiments.

As in the application of the parallel method to mode changes, all transitions between configurations are treated equally. This means that differences between configurations for scenarios are minimised as much as differences between the cur-

rent problem specification and the configuration for a scenario. There is perhaps a stronger argument for allowing the importance of these transitions to differ than there was in the application to mode changes. However, the same argument regarding finding groups of solutions given in section 6.5.3 is also still relevant.

Requirement Req. 1 stated that moving some tasks may have a higher cost penalty than others. The sequential method used to satisfy this requirement does not currently address this. The Toast task allocation tool can take a constraint file as input which restricts the allocation of a task to a single processor or group of processors. The format of this file is given in appendix E. This would completely remove the possibility of moving a task.

If values were assigned to the importance of not moving a task or message, then equations (6.1) and (6.2) could be modified to use a weighted mean. This has not been implemented or evaluated.

8

Extensions For Fault Tolerance And Graceful Degradation

8.1 Introduction

The reliability of computer based systems has become more important as their use as key components of critical systems has escalated. In particular, in section 1.3, this was identified as one of the important trends in automotive and avionic control system application development. Due to the severe consequences of these systems failing, it is essential that they can provide an acceptable level of service and system safety even when components have a permanent fault. A system which can continue to run safely in the presence of faults but with a reduced level of service is referred to as a gracefully degrading system [184].

There are a number of strategies which can be used to allow a system to continue to function in the presence of faults. The most prevalent is replication. A system with replication runs redundant versions of some or all of its tasks and will usually require additional processors to do so. Systems using replication can use cold, warm or hot back up [100] strategies. The difference lies in whether all task replicas are always running and being kept up to date to allow an almost instantaneous fail-over or if they are left dormant until needed. This work assumes a hot backup strategy, commonly used in avionics [185].

The choice between dynamic or static redundancy also distinguishes fault-tolerance mechanisms [100]. Static redundancy uses redundant tasks to mask faults whereas a dynamically redundant system waits for the system to begin to error or give an indication an error is about to occur before taking steps to recover. This chapter is only concerned with static redundancy which is commonly used in critical systems [185].

In this chapter, the Toast tool is extended to support fault-tolerance. By al-

lowing the algorithm to vary the number of task replicas and their allocations, it is able to alter the fault-tolerance properties of the system. Additionally, this should find a better trade-off between hardware utilisation and fault-tolerance compared to replicating each task the same number of times. There are a number of challenges in doing this. When task and message replicas are added to the system, they must also be allocated and scheduled, increasing the size of the problem. In addition to this, the optimisation algorithm now has an additional axis of variation in determining how many replicas should be used. Pure task allocation for hard real-time systems is a constraint satisfaction problem where any solution in which all deadlines are met is acceptable. In order to assess fault-tolerance qualities of a system, a fault-tolerance cost function needs to be developed. This is non-trivial since there are many ways in which a system can fail. In particular, a system can degrade in different ways for increasing numbers of faults and the preferred degradation behaviour will differ according to system requirements. Therefore, cost functions which can be adapted to different situations are needed.

The chosen method of evaluation of fault-tolerance quality uses a measure of system utility designed by Shelton [186]. This is a value assigned to the level of service the system is currently providing based on which components are working and which have failed. The system may be able to continue to work safely when all versions of some tasks have failed if critical tasks are still running. Shelton's metric allows a task allocation algorithm to make decisions as to which combinations of tasks must be kept running so that processor faults do not cause catastrophic failures and the system degrades gracefully instead. How levels of service are mapped to system utility values and safe thresholds are decided is considered outside the scope of this work though a possible method is briefly discussed.

Fault-tolerant systems are designed with the aim of being t-fault-tolerant and/or having a certain mean time between failure (MTBF) [187]. A t-fault-tolerant system is one that can withstand t faults before failure. Objective functions are built upon the system utility measure to support both the t-fault-tolerant and MTBF paradigms.

This chapter proceeds as follows. Section 8.2 briefly surveys related work which brings together task allocation and fault tolerance. Section 8.3 extends the task allocation algorithm to accomodate fault tolerance objectives. This includes changing the search neighbourhood to allow the number of task replicas to vary and the development of new cost subfunctions to achieve the pattern of degradation required. Section 8.4 evaluates the modifications made to the algorithm and the results are summarised in section 8.5.

Much of the work in this chapter was previously published in a paper by Emberson and Bate [188]. In relation to this paper, this chapter further develops the expected utility loss cost subfunction. Details of the changes are given in section 8.3.4.2. The evaluation in section 8.4.2 reflects these changes.

8.2 Related Work

Previous work combining automated real-time task allocation with fault-tolerance has tended to concentrate on modifying the scheduling analysis to accommodate the overheads needed for fault tolerance [189, 190, 191]. Often the work is specific to scheduling policy such as static cyclic scheduling and optimises this schedule so that the system can handle a single processor failure [189, 191]. The emphasis of this work is in making higher level architectural decisions, in particular the number of replicas to use and where they will be allocated. Previous work on static redundancy and task allocation has been concerned with a fixed number of processor faults, and often just a single fault. This work deals with task allocation in the context of a gracefully degrading system and has the ability to differentiate between systems which can handle the same numbers of faults before complete failure but degrade differently.

Oh and Son [190] discuss the need to consider schedulability and fault-tolerance simultaneously. They prove that finding a schedule to handle a single processor failure is NP-hard and give an algorithm to solve this problem. The system model does not include precedence constraints.

Girault et al. [189] adapt an algorithm which generates distributed static schedules to handle processor failures with fail-stop behaviour. However, the number of replicas is pre-determined and allocated by making a copy of task sets on existing processors to redundant processors. Qin and Hong [191] build on the work of Girault et al. The system model includes precedence constraints and a more heterogeneous environment. They introduce the concept of performability which is a combination of schedulability and fault-tolerance. They allow for reliability heterogeneity by including a failure rate for processors in their model.

Echtle and Eusgeld [192] use search, specifically a genetic algorithm, to find fault-tolerant system designs. However, the approach is not aimed at real-time systems and schedulability is not taken into account. Of some interest is the fitness function used by the search algorithm. It considers how combinations of faults lead to failures and in this sense has some commonality with the utility model introduced in section 8.3. Bicking et al. [52] take a similar approach to [192], also using a genetic algorithm, but once again the system model is not targeted at real-time.

Attiya and Hamam [54] also solve a non real-time task allocation problem using simulated annealing with an emphasis on maximising reliability. System reliability is measured based on the failure rate of processors and networks combined with the utilisation of the scheduling resource. This approach takes no account of the importance of each task to the level of service provided by the system.

Kany and Madsen have written a high quality study on design optimisation for fault-tolerant real-time systems [88]. Their fault-tolerance design choice is the decision between trying to re-execute a task or using a single task replica to mask faults. This is in addition to selecting an allocation. Task priorities are pre-defined,

unlike Toast which searches for a priority ordering in addition to allocation and fault-tolerance decisions.

Izosimov et al. [193] look at a similar problem to that of Kany and Madsen where decisions must be made on whether tasks will be re-executed or replicated to mask transient faults. They look at a larger optimisation problem than Kany and Madsen, considering task mapping, task scheduling and bus scheduling also. The algorithm used is a combination of a greedy heuristic followed by tabu search, an optimization method often used for job shop scheduling [115]. The work of Kany and Madsen and Izosimov et al. differs from this work which is concerned with system behaviour over an increasing number of permanent failures.

Ejlali et al. [194] present work which combines fault-tolerance with power management. They point out that a full hot backup mechanism uses a lot of additional power. Instead a backup is used which consumes less power by taking advantage of slack in the schedule. This suggest an alternative way to gracefully degrade systems; allow the system to have worse performance in order to mask faults. In contrast, this chapter presents a metric which measures functional degradation.

Nace, Koopman and Shelton [184, 186], as part of the RoSES (Robust Self-configuring Embedded Systems) project, outline a framework for providing graceful degradation using a combination of feature subsets, utility model and task allocation.

However the main results from the project provide only the utility model of Shelton [186], and it is generally assumed that each feature subset is resident on its own processor. The main aim of this chapter is to combine the task allocation algorithm developed in chapter 5 with Shelton's utility model and thereby contributing to the overall framework envisioned by Nace et al. [184].

8.3 Extensions For Fault Tolerance

In order to embrace fault-tolerance as a core part of the automated architecture design process, it is necessary to extend the system model and schedulability test as well as both the neighbourhood and cost functions used in the search algorithm.

8.3.1 Extensions To System Model

An extra attribute is added to the problem specification which indicates the maximum number of replicas for each task. Since a hot backup strategy is being assumed, if a message passes between two tasks, then an equivalent message must be passed between all replicas of those tasks. Figure 8.1 shows a task with two replicas which sends a message to a task with a single replica. If at least one version of each task is on a processor which has not failed, then the functionality provided by these tasks will still be present. When multiple versions of a task are present in the system other tasks will receive messages from each of the replicas. In this chapter, the computational model assumes that the first message received is used. Other models are

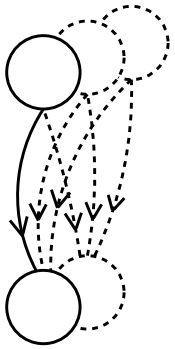


Figure 8.1: Messages sent by task replicas

feasible. For instance, tasks could wait for all messages of working replicas to arrive in order to compare results.

8.3.2 Extensions To Neighbourhood

In addition to configuring the original set of tasks and messages in the system, allocations and priorities must be found for all replica tasks and their messages. For this purpose, replicas are simply treated as extra schedulable objects in the system. The replicas increase the size of the neighbourhood making the possible solution space much larger. In order for the algorithm to decide how many task replicas should be used, a third axis of variation (in addition to allocations and priorities) must be added. The neighbourhood function is given the option of enabling and disabling replica tasks. A disabled task is effectively removed from the system for the purposes of evaluating the system's schedulability and fault-tolerance qualities. When a task is disabled, all of the messages it sends and receives are also disabled as these will not be needed as part of a design with fewer replicas. On completion of the search, any disabled objects present in the solution judged to be the best will not be included in the output.

During the evaluation it was found that it is better to favour enabling a disabled task as opposed to disabling a working one. Therefore, the function chooses to change the status of a currently working task with probability 0.1 and changes the status of a currently disabled replica with probability 0.9. It should be noted that this parameter's optimal value is dependent upon the problem to be solved. Without this bias, the search has a tendency to find local optima where all tasks are schedulable but too few replicas are included in the solution.

8.3.3 The System Utility Metric

The system utility metric is taken from Shelton [186]. This section explains how it is implemented efficiently and used as part of the cost function.

For a system, where some components may have failed, the *utility* of the system is a measure of the functionality that the system is still providing. Calculating such a

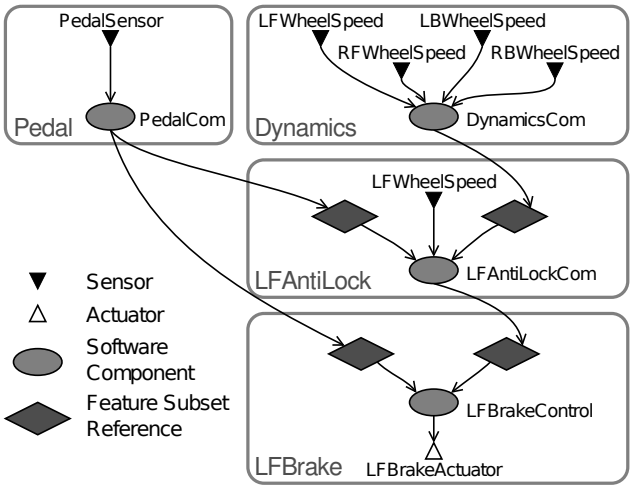


Figure 8.2: Example feature subsets from a braking system (based on Shelton [186])

value is difficult since failures are generally not independent. For example, consider an automotive braking system with a brake on each of the four wheels. The loss in utility can be considered equivalent for any single brake failure. If two brakes fail, configurations where both failed brakes are on the same side of the car can be considered to have less utility since this will cause the car to swerve when braking.

Assuming a fail-fast, fail-stop model [98], where each component can only have a status of *working* or *failed*, there are 2^N failure states for a system with N components. Assigning a utility value to every one of these states is not a scalable solution. To overcome this, Shelton [186] developed a method which uses hierarchical decomposition to reduce the number of utility values required. Shelton's method takes advantage of the existing design decomposition already present in the system to group individual system components into feature subsets. These feature subsets containing system components are grouped to form higher level feature subsets.

Figure 8.2, a simplified version of Shelton's diagram [186], shows four feature subsets for the left front brake subsystem of an automotive braking system. The utility of each feature subset depends on the status of the components in it and that of any feature subsets it references. Shelton's example example utility values for the LFAntiLock feature subset are shown in table 8.1. The utility value in each row of the table corresponds to one or more failure configurations of the components in the feature subset. All components listed in the configuration column must have a working status in order to use the corresponding utility value. The utility value for a configuration is written as a formula which can refer to the utility value of another feature subset using the function $U(\cdot)$. This is shown in the first row of table 8.1 which uses the utility value of the Dynamics feature subset.

In order to involve utility as part of the cost function, it is necessary to create both an internal representation for efficiently calculating system utility and also

Feature Subset	Configuration	Utility
LFAntiLock	LFAntiLockCom, LFWheelSpeed, Pedal, Dynamics	$0.7 + 0.3 * U(Dynamics)$
	LFAntiLockCom, LFWheelSpeed, Pedal	0.7
	Others	0

Table 8.1: Example utility values for the *LFAntiLock* feature subset

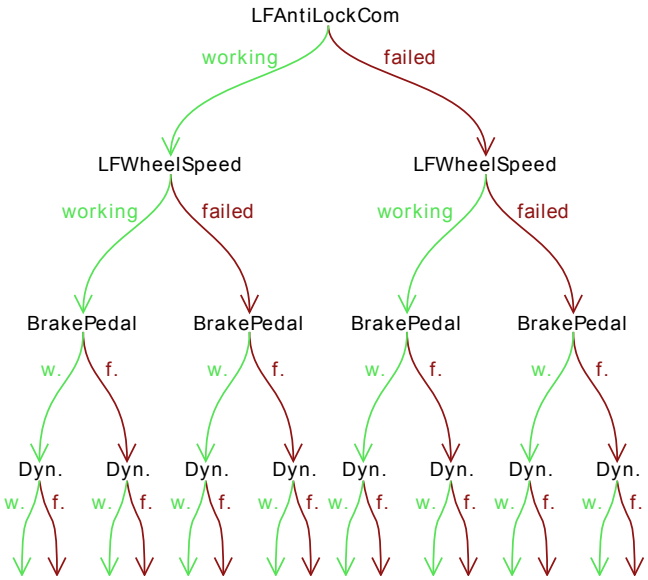


Figure 8.3: Fully expanded utility decision tree

an external representation for conveniently providing input data which represents a system utility model.

The internal representation for the utility of a particular feature subset is a decision tree. Each node of the tree is one of the components of the feature subset. At the leaves of the tree are the utility values. In order to calculate the utility value for the feature subset, the tree is descended from the root selecting either the *working* or *failed* branch from each node depending on the failure status of the node. For low level components, the failure status is decided by whether the component has failed or not. For higher level feature subset components, the failure status is *failed* if the utility value of the referenced feature subset is 0 else the failure status is *working*.

In general, for a feature subset with k components, a utility tree will have a depth of $k + 1$ and have 2^k utility values on the leaves of the tree. An example of such a tree for the LFAntiLock feature subset is shown in figure 8.3. However, it is clear from table 8.1 that many configurations have the same utility value. In particular, for configurations dependent on multiple components it is not necessary to check the status of every component if any have failed. This allows the tree to be pruned as shown in figure 8.4.

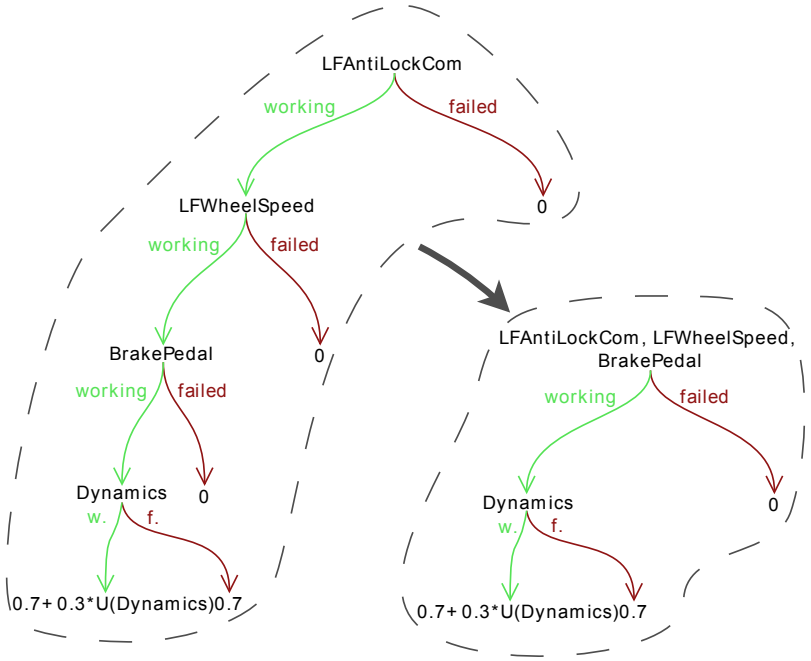


Figure 8.4: Pruned and compacted trees

The tree can be compacted further by allowing a decision node to depend upon multiple components as shown by the transformation in figure 8.4. This form has the advantage of being a more direct mapping from Shelton's table of utility values in table 8.1. It also forms the basis of an XML format for utility decision trees which is not overly verbose. Appendix F provides further details of this format. All of the lowest level components are linked to a task in the system specification. Each decision node lists feature subsets on which to base the decision between following the *working* or *failed* branch. Every node also has a require attribute which can be set to any or all which indicates how many of the features must be working in order to take the *working* branch. Every utility model input must include a feature subset named System. This is assumed to be the root of the feature subset hierarchy and its utility value is used to calculate the utility value for the system under a particular failure configuration.

The derivation of utility values for each feature subset is not dealt with by this work. One way in which to interpret these values is to map a loss in utility to the expected monetary value [195] which will be the cost incurred by any failures caused by the faults. When combined with probabilities of the faults occurring expected monetary value is one method of quantifying risk.

Two cost subfunctions which both use the utility metric are described in the next section. The first is based upon keeping the system utility above an acceptable threshold for as many faults as possible. The other is motivated by the concept of expected monetary value and uses probabilities of processor faults to minimise the expected loss of utility.

Task	WCET	Period	Max Replicas
A	7	10	2
B	7	10	2
C	2	10	2
D	2	10	2

Table 8.2: Example problem

Feature Subset	Utility
System	0.5 * U(A) + 0.3 * U(B) + 0.15 * U(C) + 0.05 * U(D)

Table 8.3: Utility model for example problem

8.3.4 Extensions To Cost Function

The system utility model can generate several values for all numbers and combinations of faults. The challenge of using this model in a search algorithm is to create a function which can map all of these values to a quality metric for fault-tolerance. Two approaches are introduced here. The choice of which to use will depend on the fault-tolerance requirements for the system.

8.3.4.1 Maximum Utility Loss Subfunction

The first cost subfunction is based on the worst case system utility value for an increasing numbers of faults. This is clarified with the following example. Table 8.2 shows a small system with four tasks and no messages. Deadlines are set equal to the period of the task. The timing requirements of tasks A and B dictate that they cannot be allocated to the same processor. Each task can be replicated up to two times so there may be up to three versions of each task in the solution. The utility model for this system, given in table 8.3, uses a simple additive model such that the utility provided by any particular task is independent of whether other tasks have failed. In this instance, the amount of utility provided to the system decreases from task A to task D.

Two feasible solutions, where all schedulability constraints are met, are shown in figure 8.5. For Solution 1, the worst case single processor fault which can occur is when P1 fails since there are no versions of task B left in the system. The worst combination of two faults is when processors P1 and P3 fail which removes all versions of both tasks B and D from the system. The worst case system utility values for increasing numbers of faults for both Solution 1 and Solution 2 are plotted

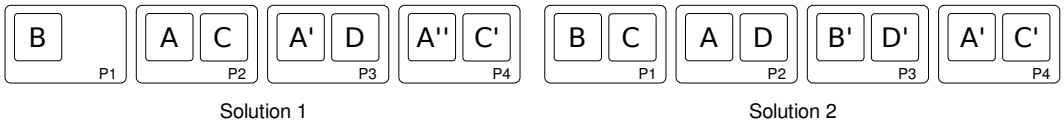


Figure 8.5: Two solutions to example problem

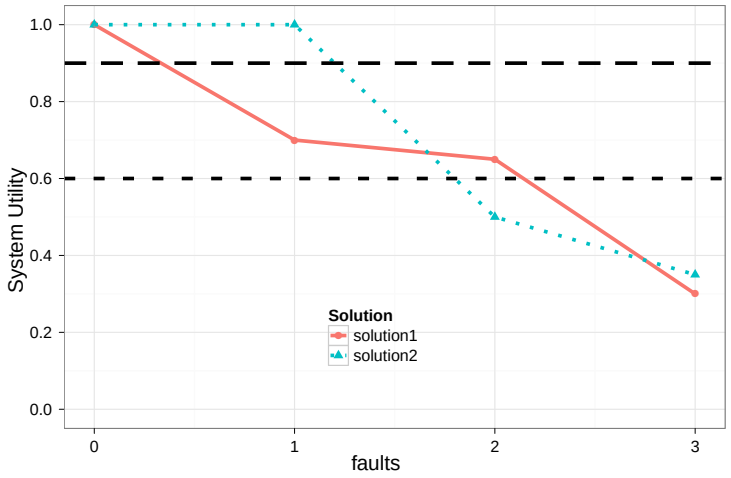


Figure 8.6: Worst case utility degradation

in figure 8.6. The answer to the question of which degradation profile is preferable will depend on requirements. In order for the cost subfunction to be able to select which solution is best, a threshold parameter, which is specified as part of the utility model, is introduced. It is assumed that once utility falls below this threshold the system in some sense becomes unreliable or unsafe. Therefore, the aim is to withstand as many faults as possible before the utility falls below this level. This is akin to trying to maximise the value to t when designing a t-fault-tolerant system. Using this model, Solution 1 is preferable for the low threshold value marked in figure 8.6 whereas Solution 2 is better for the high threshold. A cost subfunction which will order solutions according to this scheme is given in equation (8.3).

$$M_t = \min_{F \in combs(\mathcal{P},t)} U(System|Ffailed) \tag{8.1}$$

$$v_t = \begin{cases} M_t \text{ if } M_t \geq V \\ -V \text{ otherwise} \end{cases} \tag{8.2}$$

$$g_{maxloss} = 1 - \frac{\sum_{t=0}^{|\mathcal{P}|-1} 2^t (v_t + V)}{(1+V)(2^{|\mathcal{P}|}-1)} \tag{8.3}$$

In these equations, t is the number of processor faults. The set $combs(X, i)$ is the set of all combinations of i items chosen from X. M_t is the worst case utility value for t faults and V is the threshold utility value. In order not to differentiate between solutions once utility has fallen below the threshold, if $M_t < V$ then a value v_t is set to $-V$ else it is equal to M_t. The 2^t term ensures that systems which can withstand more faults will always have a lower value of $g_{maxloss}$. Note that the initial value of t is 0. The utility is not necessarily maximal for 0 faults since the search algorithm has the design choice of not including any versions of a particular task in the system.

Given an arbitrary utility model, calculating v_t requires every combination of

possible processor faults to be evaluated. For systems with more processors, the number of combinations will grow rapidly. One way of limiting this is to only iterate up to a maximum number of faults. A suggested topic for future work is to investigate approximations for v_t for particular restricted utility models.

8.3.4.2 Expected Utility Loss Subfunction

An alternative cost subfunction for assessing the fault-tolerance quality of a system is based on the system's expected loss of utility.

Let $P(F)$ be the probability that processors in the set F have permanent faults.

$$h_{exploss} = 1 - \sum_{i=0}^{|\mathcal{P}|-1} \sum_{F \in combs(\mathcal{P},i)} P(F)U(System|Ffailed) \tag{8.4}$$

Equation (8.4) calculates a value for the expected utility loss over a unit of time. The system utility for each possible fault combination is calculated and summed. $P(F)$ is calculated by combining the probabilities of individual processor faults. Processor faults are assumed to be independent though any probability function which maps a set of faults, F, to the probability of them occurring could be used.

$h_{exploss}$ lies in the range $[0, 1]$ and could be used as a cost subfunction. However, the probability of a processor failure is small, say 0.001, and so the values of $h_{exploss}$ for possible solutions are typically very small in comparison to other cost subfunction values. Further to this, the resulting value is dominated by terms for a single failure since the probability of multiple faults is very low. Setting a weighting value for a cost subfunction set to the value of $h_{exploss}$ is difficult since other subfunctions return values which are orders of magnitude larger and have more variation.

In a previous paper [188] on which this work is based, the following transformation was suggested

$$g'_{exploss} = {h_{exploss}}^{0.075} \tag{8.5}$$

to create a cost subfunction, here labelled $g'_{exploss}$, which has larger values for solutions typically encountered during a search. The mapping from $h_{exploss}$ to $g'_{exploss}$ is shown in figure 8.7. This transformation does indeed map values of $h_{exploss}$ to values whose magnitude is more comparable with other cost subfunctions. However, during experimentation, it was found that the expected utility loss of most solutions fell in a very small range between 0.0021 and 0.0022. As seen in figure 8.8, the plot of $g'_{exploss}$ over this small range is nearly horizontal. This gives the search little guidance in moving towards higher quality solutions. While experiments in previous work [188] were able to reduce the expected utility loss using this cost subfunction, an attempt was made to improve upon it using a function designed specifically designed for the problem used in the evaluation in section 8.4. It is the same problem as was used in the previous paper [188] by this author.

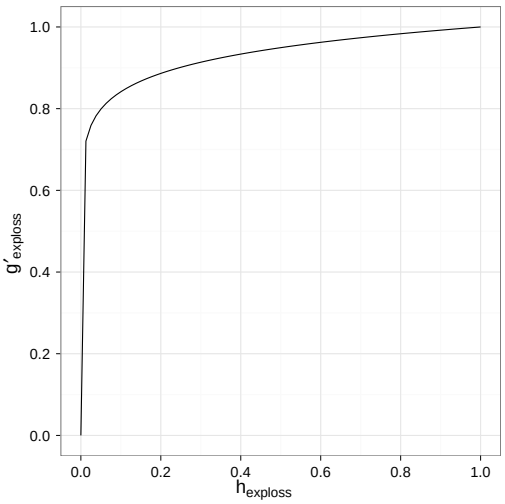


Figure 8.7: Transformation of expected utility loss to cost subfunction using $h_{exploss}{}^{0.075}$

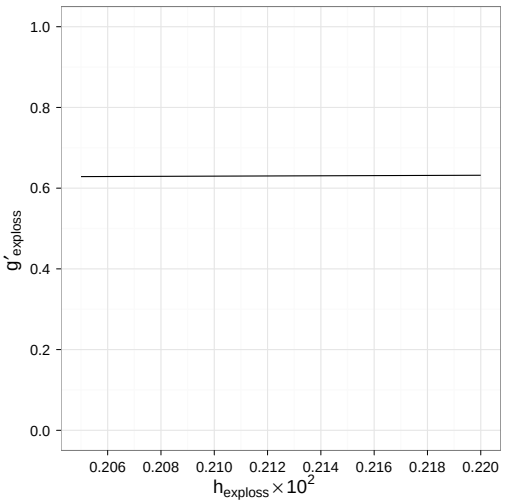


Figure 8.8: Function from figure 8.7 zoomed to region where most solutions lie

The function used, $g_{exploss}$, is given in equation (8.6).

$$g_{exploss} = \frac{1}{1 + \exp(-3 \cdot 10^5 (h_{exploss} - 0.002135))} \tag{8.6}$$

A plot of how $g_{exploss}$ changes with $h_{exploss}$ is shown in figure 8.9. It is displayed over the same domain as in figure 8.8 for comparison. The form of equation (8.6) and the coefficients used were chosen to achieve a steep gradient in the domain of interest as viewed in figure 8.9. Alternatives which achieved the same effect would be equally valid.

It should be noted that the cost subfunction in equation (8.6) has been designed specifically for the problem in the evaluation assuming 0.001 to be the probability of processor failure. In order to be able to design such a function, some screening experiments need to be run to know the range of expected loss values which are of interest. This can be determined by attempting to minimise a simpler cost subfunction such as that in equation (8.5) or even the expected loss value itself from equation (8.4).

This approach is consistent with the conclusions of chapter 5 in section 5.5.4. The use of search in SBSE does not remove the engineering aspect of the problem but moves it into the design of the search algorithm. However, by providing a framework with a heuristic that can easily be adjusted, this problem should be easier than the original one. In this case, the measure for expected loss is established, but a suitable transform still needs to be found to turn it into a cost subfunction which will guide the search effectively.

The final issue in the design of the extended cost function is the balancing of schedulability and fault-tolerance. Since schedulability is being treated as a constraint rather than an objective, the optimisation problem is a single objective prob-

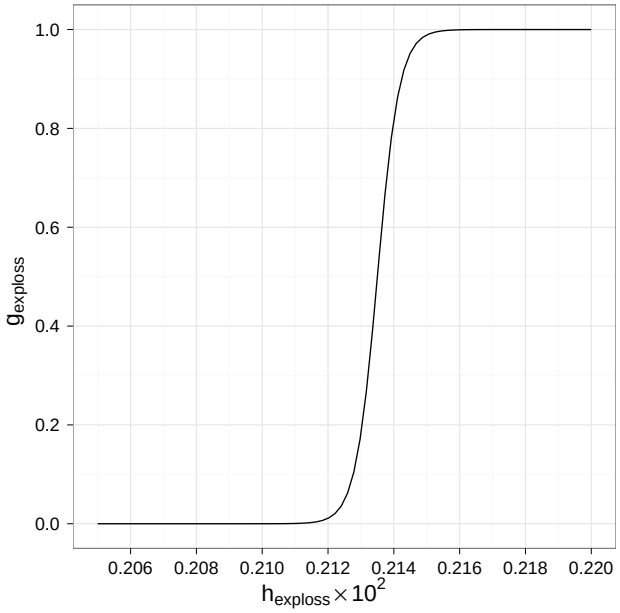


Figure 8.9: Cost subfunction for expected loss designed specifically for problem in evaluation

System	Replicas	Tasks	Messages
evalsys	0	8	12
evalsys	1	16	48
evalsys	2	24	108
brakes	0	19	29
brakes	1	29	90
brakes	2	39	183

Table 8.4: Change in problem size as replicas are added

lem which can be stated as: *optimise fault-tolerance such that all schedulability constraints are met*. Weightings in the cost function must be set to achieve a good balance between meeting all scheduling constraints, maximising fault tolerance and making good use of the guidance heuristic.

8.4 Evaluation

The evaluation tests the effectiveness of the two cost subfunctions for fault-tolerance given in equations (8.3) and (8.6). Two systems are used during the course of the evaluation. The first is a randomly generated 8 task example called *evalsys* and the other is a brake-by-wire example taken from Shelton [186] labelled *brakes*.

The increase in problem size as replicas are added is shown in table 8.4. Of particular note is the number of extra dependencies which are generated. In the *brakes* problem, sensors, software components and actuators are represented but only software components are replicated. Separation constraints were also enforced

Feature Subset	Configuration	Utility
System	τ_1, τ_2, τ_3	$0.25 * U(\tau_4) + 0.25 * U(\tau_6) + 0.25 * U(\tau_7) + 0.25 * U(\tau_8)$
	τ_1, τ_5	$0.25 * U(\tau_7) + 0.25 * U(\tau_8)$
	Others	0

Table 8.5: Utility values for *evalsys*

Num Replicas	Procs	V=0.8	V=0.4
1	3	1 (1.00)	1 (1.00)
2	3	unschedulable	unschedulable
2	4	1 (1.00)	2 (0.75)
2	5	2 (1.00)	2 (1.00)

Table 8.6: Fixed number of replicas

for this problem with the assumption that wheel speed sensors and brake actuators for each wheel would be on separate processors.

In order to conduct experiments using the fault-tolerance cost subfunctions, utility models were needed for both systems. The model for the *brakes* system is taken from Shelton [186]. The utility model for the *evalsys* system was constructed as shown in table 8.5. It shows that task τ_1 is critical to the system and that the system can only run at full utility if tasks τ_2 and τ_3 are also present.

8.4.1 Experiment 8.1 — Evaluation Of Maximum Loss Subfunction

Experiment 8.1 used the *evalsys* problem to evaluate the worst case loss fault-tolerance cost subfunction, $g_{maxloss}$. It compared two strategies of replication. The first used a fixed number of replicas whilst the second allowed the algorithm to vary the number of replicas used. Separation of replicas is not enforced but solutions where replicas are allocated to the same processor should be heavily penalised for poor fault-tolerance quality.

Table 8.6 shows results for systems with a fixed number of replicas. The number of processors and threshold values were varied across different runs. The values in the final two columns give the number of faults which could be tolerated before the system utility fell below the threshold. The values in parentheses are the worst case system utility after that many failures. The original problem requires two processors to schedule all tasks. Therefore, a solution which duplicates this would require 4 processors and handle a single fault. However, the results show that it is possible to find a solution which achieves the same degree of fault-tolerance with only 3 processors. These runs took about 25 minutes to complete. Although there is some overhead from having to calculate worst case system utility values, this was insignificant compared to the additional time spent by the search in finding a schedulable solution

Max Replicas	Procs	V=0.8	V=0.4
2	3	1(1.00)	2 (0.50)
2	4	1(1.00)	2 (0.75)
2	5	2(1.00)	2 (0.75)

Table 8.7: Variable number of replicas

Task	0.8,3	0.8,4	0.8,5	0.4,3	0.4,4	0.4,5
τ_1	1	2	2	2	2	2
τ_2	1	1	2	1	2	2
τ_3	1	1	2	1	2	2
τ_4	1	1	2	1	1	2
τ_5	0	1	0	1	2	1
τ_6	2	1	2	1	1	1
τ_7	2	1	2	1	1	1
τ_8	1	1	2	2	1	1
Total	9	9	14	10	12	12

Table 8.8: Number of replicas used

since it had to balance this constraint with the objective of improving fault-tolerance.

Table 8.7 shows results when the search was able to vary the number of replicas. These runs took longer still, taking up to an hour to find good solutions. Finding a schedulable solution is now easier for the search because it is able to remove replicas. To compensate for this and maintain a good level of utility, it was necessary to adjust the balance of weights away from the group of schedulability cost subfunctions and in favour of the worst case loss cost subfunction. For fixed numbers of replicas, schedulability was weighted 10 times higher than fault tolerance but for this latter table of results, fault tolerance was weighted more highly in a ratio of 2 to 1. This difficulty is emphasised by the fact that the result achieved for a threshold of 0.4 and 5 processors is slightly worse than that of the equivalent result with a fixed number of replicas. However, the benefit can be seen in that it was able to withstand an additional fault with only 3 processors when the threshold was set at 0.4. Table 8.8 shows the number of replicas included in each solution for the results in table 8.7. This shows that the algorithm correctly favoured the critical task, τ_1, and increased the number of replicas used when extra processors were available.

8.4.2 Experiment 8.2 — Evaluation Of Expected Loss Subfunction

The expected utility loss cost subfunction was tested with the *brakes* system. Since this system did not have redundant sensors included in the example, it will not withstand any faults in the worst case but the expected utility loss can still be improved. The probability of each processor failing in a given time frame was set to 0.001. The search was limited to 50000 evaluations in all runs. Experimental runs were conducted using both fixed numbers of task replicas and allowing the search to

Minimise Utility Loss	Replicas	Vary Replicas	Procs	Expected Utility Loss
No	=1	No	5	0.002606158134324
			6	0.002273133081091
			7	0.002134274843946
			8	0.002384026767044
No	=2	No	5	*
			6	*
			7	0.002134744579720
			8	0.002134413256702
Yes	=1	No	5	0.002134274843945
			6	0.002134274843946
			7	0.002134274843946
			8	0.002134275561537
Yes	=2	No	5	*
			6	*
			7	0.002134274888824
			8	0.002134275007039
Yes	≤2	Yes	5	0.002134301776499
			6	0.002134274850682
			7	0.002134274870878
			8	0.002134275027201

Table 8.9: Expected utility loss results for different parameters and platforms

vary the number.

The results are shown in table 8.9. The top two sets of runs solved the problem without the use of the cost subfunction for minimising utility loss. These results provide values for expected utility loss for comparison with other runs which hope to improve upon them. A schedulable solution could not be found using 2 replicas on platforms with 5 or 6 processors.

With 1 replica of each task and the use of the cost subfunction in equation (8.6), the expected utility loss was as good or better in all cases. By adding another replica of each task, it was hoped that the expected utility loss could be reduced further. The search was not able to do this for the 7 processor case however. Also, using 8 processors gave a value slightly worse than using 7 in each case showing that no solution could be found which benefited from the use of more hardware. In fact the smallest expected loss value achieved throughout the whole experiment was using 1 task replica on a 5 processor system.

For the final set of runs, the search was allowed to vary the number of replicas of each task. This was able to find good solutions but could not improve on the runs which used a fixed number of replicas.

8.5 Summary

This chapter has presented extensions to the Toast task allocation tool to make architectural decisions influenced by fault-tolerance requirements. These decisions are made using a utility model which assigns quantitative values to the system's utility in different failure states. The utility model is general enough to allow different utility values to be assigned to every possible combination of task failures. However, following the work of Shelton [186], hierarchical decomposition of the system design makes this assignment of values a tractable problem. The use of such a general model of system utility as part of an objective function for task allocation is a novel contribution of the work in this chapter.

The neighbourhood for local search was extended to allow the number of task replicas to be varied. Objective functions were designed based on the utility value of a system after tasks have been eliminated due to permanent processor failures.

Suitable cost subfunctions were presented which allowed for gracefully degrading systems to be generated. The pattern of degradation can be changed by setting a threshold parameter for a worst case utility loss cost subfunction. A cost subfunction for expected utility loss was also given assuming a known probability of processor failure for each combination of processors.

For each of the new cost subfunctions, the search was run on problems with a fixed number of task replicas and also on equivalent problems where the search could vary the number of replicas.

Using the maximum utility loss cost subfunction, the search was able to make more efficient use of the hardware platform when allowed to vary the number of replicas. The search algorithm was not able to achieve the same result for the expected loss utility function within the maximum number of evaluations allowed. The expected utility loss cost subfunction also needed to be engineered specifically for the problem due to the very small changes in expected utility loss between different solutions. From a search engineering point of view, the maximum utility loss cost subfunction could achieve better results with less effort.

No hypothesis statement was made with regard to the results of this chapter. The work performed is, however, directly aimed at meeting Req. 4 from section 3.1.4 of chapter 3. This requirement stated that the search should select subsets of tasks which minimised loss of service when processors fail. The mechanism of enabling and disabling tasks was not limited to changing the number of task replicas. All copies of a task could be removed for a system with few resources so that critical functionality is still provided. When using the maximum utility loss, tasks were arranged so that the subsets remaining when a processor failed provided as much functionality as possible.

The problems used for the evaluation in this section are relatively small. There is a large amount of overhead in allocating scheduling the additional task replicas and

messages. An alternate model which does not require all replicas to be concurrently running may increase the rate at which systems can be analysed. The calculation of the utility related cost subfunctions is also time consuming. This is due to the need to consider all combinations of processor failures. As stated in section 8.3, this could be improved by only considering up to a certain number of faults.

Communication failures have not been accounted for in this chapter. The utility model approach is still valid if it is assumed a task fails when it does not receive a message due to a network failure. Therefore, the cost function extensions could be changed to measure utility loss for network rather than processor failures. This is left as future work.

9

Conclusions and Future Work

The motivation for this work originates from the need to build hard real-time systems, particularly automotive and avionics systems, which are flexible and easy to maintain. One of the ways this is being done is through the use of standards such as AUTOSAR and IMA which provide layers of abstraction between software and hardware. This means that software can be:

- developed without full details of the hardware platform,
- bound to the hardware at a later stage of development,
- be upgraded more easily since the software can be remapped to the hardware to accommodate new applications

One of the open questions in the design of these systems is how to map the software onto the hardware platform and, in particular, how to map it in such a way that new requirements do not cause severe disruption to the chosen mapping. This is the task allocation problem which was studied in depth throughout this thesis. Contributions were made in the field of creating flexible solutions to task allocation problems and related areas.

9.1 Overview Of Work Done And Contributions

Chapter 1 surveyed issues relating to general software maintenance, changeability and trends in avionics and automotive systems. It was found that these systems are becoming larger and at the same time there is increasing integration of functionality which increases complexity. This makes finding a task mapping and schedule a very challenging task to perform manually. When a mapping is established, the high costs of change in safety-critical hard real-time systems mean it is desirable to evolve the existing solution rather than create a new one.

Another trend in avionics and automotive system is the reliance on software for safety-critical functionality. Systems must be fault-tolerant and adapt to environmental changes at run-time. The ability to change the set of running tasks and provide redundancy are relevant to task allocation. These topics, as well as design-time flexibility, were also covered in this thesis.

Chapter 2 reviewed existing literature on task allocation and current approaches to creating flexible architectures. While there is plenty of work on task allocation algorithms, there is little in the area of hard real-time task allocation with support for design-time flexibility or run-time adaptability. It was suggested that scenario based analysis, a technique for improving architecture flexibility, could be linked with algorithms for task allocation. Heuristic search was the optimisation approach chosen because it itself is flexible and can work with task allocation problems using different schedulability tests and system models. In this chapter, a new definition of task allocation was given which emphasised the need to put the problem in the context of the system model and schedulability test.

Chapter 3 gave requirements for a task allocation tool to support flexibility, adaptability and fault-tolerance. A hypothesis was proposed as to whether meta-heuristic local search algorithms would be able to meet these requirements. This is addressed further in sections 9.2 and 9.3.

Chapter 4 described the hardware and software models which would be used throughout this thesis. A task allocation problem generation tool, Gentap, was developed in this chapter.

> *Contribution 1. — Multiprocessor allocation and scheduling problems have many characteristics. The parameter set for the Gentap tool provides a way of characterising task allocation problems and generating problems with those characteristics. This characterisation can also be used to classify problems when tuning search algorithm parameters.*

> *Contribution 2. — The problem generation technique included a novel way of producing task sets with a given size and total utilisation. The method fitted a distribution to task utilisations from an industrial case study and then sampled from it to produce similar task sets.*

> *Contribution 3. — A new algorithm for generating task graphs with different properties is also included in Gentap.*

Chapter 5 described the design and configuration of a simulated annealing based search algorithm for task allocation. The algorithm was shown to work well on a range of problems with different characteristics and could also be easily modified for different schedulability tests. The number of solution evaluations increased approximately linearly with problem size but the time taken to perform a schedulability test

decreased much faster. This was the limiting performance factor for large problems.

> *Contribution 4. — A hierarchical cost function was given which can be tuned to work well on problems with different characteristics using a systematic experimental method.*

> *Contribution 5. — A Cox's Proportional Hazard model was fitted using problem characteristic covariates to find the characteristics most significant to algorithm performance. The most significant factors were then used as a means to classify problems and tune the algorithm separately for different problem classes. The classification was judged to be good based on the algorithm giving very consistent performance on problems within a class.*

Chapter 6 presented a parallel simulated annealing algorithm for finding sets of solutions with minimal differences between them for multi-moded task allocation problems. This allows efficient mode transitions for run-time adaptability. The parallel algorithm was compared with and shown to be better than an alternative algorithm which found solutions for each mode sequentially.

> *Contribution 6. — The parallel algorithm used in this chapter is a new algorithm for finding sets of similar solutions to related task allocation problems. Its parallelism makes it scalable yet it is still perfectly repeatable, with only a small percentage of evaluations being wasted. It was also shown to be highly effective at minimising changes between the solutions produced.*

Chapter 7 reapplied the parallel algorithm from the previous chapter to produce solutions which tried to meet the needs of scenarios as well as a current problem specification. It then used the sequential method to upgrade existing solutions with as few changes as possible.

> *Contribution 7. — Solutions produced by the parallel algorithm with scenarios were more flexible than solutions produced by the standard simulated annealing search algorithm. The flexibility was measured with respect to upgrade specifications which differed from the scenarios.*

Chapter 8 tackled task allocation from the aspect of fault-tolerance. The algorithm was given the task of deciding how many replicas of each task to include to maximise fault-tolerance given limited hardware resources. This new axis of variation, i.e. deciding how many replicas to include as well as allocating and scheduling tasks, increased the difficulty of the optimisation problem and limited the size of problems which could be solved.

Contribution 8. — A new worst case utility loss cost subfunction was designed based on previous work which assigned utility values to a system based on which tasks had failed. The pattern of degradation with increasing numbers of processor failures could be changed with a utility threshold parameter. This subfunction was used to decide how many task replicas to include in the system and where to allocate them.

9.2 Support For Hypothesis Statements

In chapter 3, three hypothesis statements were made regarding the work in this thesis. These were:

1. A local search algorithm, extended in appropriate ways, is an efficient and effective method for producing feasible task configurations which are similar to previous ones and for finding sets of similar configurations for multiple task allocation problem specifications representing alternatives for present and future system designs.
2. A configurable hierarchical cost function allows the method to be easily modified for problems with a wide range of characteristics.
3. Using scenarios which represent predictions of future problem specifications within the search problem can produce more flexible configurations, even when predictions are imperfect.

The first of these was supported by the work in chapters 6 and 7. The second statement was shown to be true based on the work in chapter 5. The results of chapter 7 support the final statement.

The most important caveat which needs to be placed on these results is that the hypothesis statements were only shown to be true for the problems used in the evaluations within each chapter as is true of most empirical results. The work in chapters 4 and 5 mitigates against this somewhat by placing the foundations of the work on systematically classifying problems and tuning the algorithm for particular problem classes. This means that the methods should be able to adapt to problems with different characteristics.

It was shown in chapter 5, however, that characteristics relating to problem size can limit the algorithm performance as the speed of the schedulability test decreases. Therefore, with regards to statement 1, the algorithm is only efficient for systems of around 50–90 tasks depending on other characteristics. This is large enough to cover the needs of an automotive subsystem but not an entire system with several hundred tasks. Performance is also affected by the number of specifications, representing modes or scenarios, being solved simultaneously. The method was tested, and shown to be efficient, with up to 8 specifications.

9.3 Satisfaction Of Requirements

Four primary requirements were listed in chapter 3 for the development of the task allocation tool. These were:

- Reuse Of Existing Solutions (Req. 1)
- Implementation Of Systems With Multiple Configurations (Req. 2)
- Consideration Of Future Changes (Req. 3)
- Robustness In The Presence Of Faults (Req. 4)

Req. 1 was mostly satisfied by the sequential method in chapter 6. Req. 2 was satisfied by the parallel method in chapter 6. Req. 3 was satisfied by the parallel method in chapter 6 and the application of it in chapter 7. Req. 4 was satisfied by work done in chapter 8.

One aspect of Req. 1 was not satisfied as described in section 7.4.1. This was the placing of more emphasis on minimising changes for some tasks than others. Some suggestions were made in section 7.4.1 on how this could be achieved. A method of constraining allocation changes has already been implemented for situations where a task cannot be moved whatever the cost.

A secondary requirement was specified which tied all of the above requirements into a single optimisation problem. Each individual optimisation problem was found to be challenging and no attempt was made at solving a single problem which included all aspects of the primary requirements. This is still seen as a long term goal for task allocation algorithms.

9.4 Further Work

Minor aspects of further work such as implementation improvements were suggested at the end of individual chapters. This section covers broader long term goals.

A much larger study of scenario profiles could be conducted. The scenarios used in chapter 7 applied changes throughout the system. It would be interesting to see whether scenarios which only affect a small subset of tasks cause the flexibility of baselines to be negatively affected with respect to changing tasks not included in the subset. The problem of scenario selection certainly exists in traditional scenario based architecture analysis and it would not be surprising if it was also relevant to the use of scenarios with task allocation.

One of the great advantages of using scenarios is that a scenario is just another problem specification and can capture any possible type of change. This includes changes to the hardware platform and changes to the structure of task dependencies. Indeed, these are just some of the types of scenario that might be included in a longer study of scenario profiles. Each type of scenario can also be investigated in the context of baseline and upgrade specifications with different characteristics.

Some issues have been raised with respect to the scalability of the algorithms used. By far the largest limiting factor has been found to be the speed of the schedulability test. Using tests with different accuracy versus efficiency trade-offs at different stages of a single search may make it possible to improve performance yet still find solutions to tightly constrained problems. This would require finding suitable schedulability tests and designing heuristics which make decisions as to when they should be used.

Each of the requirements in this thesis was described in terms of a single objective optimisation problem with constraints. It was noted in the work on multi-moded systems that another objective could be to move all tasks away from a processor when possible so it could be shut down to save power. This links more generally with problems which involve a trade-off between flexibility and resource usage. A multi-objective optimisation search algorithm would be applicable in this situation. The nature of these algorithms is quite different from local search algorithms with weighted cost functions. A first step for work in this area would be to investigate which multi-objective algorithm is best suited to finding valid solutions to difficult task allocation problems with tight deadline constraints and several dependencies between tasks.

The development of cost functions which measure system utility in the presence of failed processors could also be used to make trade-offs between system utility and power usage. For instance, it could be used to decide which tasks should be included in lower power modes where the number of available processors is reduced and the system is only required to provide a basic level of service.

It is well known that the parameter settings which make a heuristic search algorithm work well are highly dependent on the characteristics of the problem it is solving. Time must be spent on finding a suitable problem characterisation, dividing problems into classes and tuning algorithms for classes of most interest. Systematic ways of doing this will help with the deployment of search based tools into industry. Using meta-heuristic search on a software engineering problem does not remove the need for engineering, regardless of the meta-heuristic chosen.

9.5 Concluding Remarks

Task allocation algorithms for hard real-time systems have been studied for 20 years but are still not in widespread use. Only now are systems: a) becoming large enough to need automated task allocation and scheduling, and b) have suitable hardware abstraction layers which makes the mapping easy to change. Flexibility is important to all software systems' engineering and task allocation is no different. Algorithms cannot continue to merely find valid solutions but also must take other qualities like flexibility, fault-tolerance and power usage into account.

The motto adopted at the start of this thesis was "designing for change is designing for success". Agile development methods and tools to automate software

engineering are moving some of the responsibility for dealing with change away from those designing software to those that manage projects and select tools. Designing for change is necessary for success though not always sufficient as the changes predicted during the design many not come to fruition. Automated software engineering tools must produce flexible designs but they themselves are pieces of software which must anticipate change.

A

Problem Specification XML Representation

A.1 Document Type Definition

```
<!ENTITY % system-object
          "id ID #REQUIRED
           name CDATA #REQUIRED">

<!ELEMENT tat:system (tat:hardware-module*, tat:network*, tat:interface*, tat:task*, tat:
    message*, tat:dependency*)>
<!ATTLIST tat:system
         xmlns:tat CDATA #FIXED "http://www.cs.york.ac.uk/atu/tat"
         %system-object;
         scheduling-model CDATA #IMPLIED>

<!ELEMENT tat:hardware-module (tat:object-details?, tat:available-resources?)>
<!ATTLIST tat:hardware-module %system-object;>

<!ELEMENT tat:network (tat:object-details?, tat:available-resources?)>
<!ATTLIST tat:network %system-object;>

<!ELEMENT tat:interface EMPTY>
<!ATTLIST tat:interface
         module-id IDREF #REQUIRED
         network-id IDREF #REQUIRED>

<!ELEMENT tat:task (tat:object-details, tat:required-resources?)>
<!ATTLIST tat:task
         %system-object;
         replicas CDATA #IMPLIED>

<!ELEMENT tat:message (tat:object-details, tat:required-resources?)>
<!ATTLIST tat:message
         %system-object;
         from-task-id IDREF #REQUIRED
```

```
         to-task-id IDREF #REQUIRED>

<!ELEMENT tat:dependency EMPTY>
<!ATTLIST tat:dependency
         from-id IDREF #REQUIRED
         to-id IDREF #REQUIRED
         type CDATA #IMPLIED>

<!ELEMENT tat:object-details (tat:attribute+)>
<!ELEMENT tat:available-resources (tat:attribute+)>
<!ELEMENT tat:required-resources (tat:attribute+)>

<!ELEMENT tat:attribute EMPTY>
<!ATTLIST tat:attribute
         name CDATA #REQUIRED
         value CDATA #REQUIRED>
```

A.2 Example

```
<?xml version="1.0" ?>
<tat:system id="Test_Case" name="Test_Case" xmlns:tat="http://www.cs.york.ac.uk/atu/tat"
     scheduling-model="fp">
    <tat:hardware-module id="P1" name="Processor 1"/>
    <tat:hardware-module id="P2" name="Processor 2"/>
    <tat:network id="N_1" name="Network N_1">
        <tat:object-details>
            <tat:attribute name="bandwidth" value="1"/>
            <tat:attribute name="latency" value="0"/>
        </tat:object-details>
    </tat:network>
    <tat:network id="N_P1" name="Network N_P1">
        <tat:object-details>
            <tat:attribute name="bandwidth" value="1024"/>
            <tat:attribute name="latency" value="0"/>
        </tat:object-details>
    </tat:network>
    <tat:network id="N_P2" name="Network N_P2">
        <tat:object-details>
            <tat:attribute name="bandwidth" value="1024"/>
            <tat:attribute name="latency" value="0"/>
        </tat:object-details>
    </tat:network>
    <tat:interface module-id="P1" network-id="N_1"/>
    <tat:interface module-id="P2" network-id="N_1"/>
    <tat:interface module-id="P1" network-id="N_P1"/>
    <tat:interface module-id="P2" network-id="N_P2"/>
    <tat:task id="T_1_t0" name="Random Task 1">
        <tat:object-details>
            <tat:attribute name="period" value="30000"/>
            <tat:attribute name="deadline" value="30000"/>
            <tat:attribute name="wcet" value="1996"/>
        </tat:object-details>
    </tat:task>
    <tat:task id="T_2_t0" name="Random Task 2">
        <tat:object-details>
            <tat:attribute name="period" value="30000"/>
            <tat:attribute name="deadline" value="30000"/>
```

```
        <tat:attribute name="wcet" value="25"/>
    </tat:object-details>
</tat:task>
<tat:task id="T_3_t0" name="Random Task 3">
    <tat:object-details>
        <tat:attribute name="period" value="30000"/>
        <tat:attribute name="deadline" value="30000"/>
        <tat:attribute name="wcet" value="586"/>
    </tat:object-details>
</tat:task>
<tat:task id="T_4_t0" name="Random Task 4">
    <tat:object-details>
        <tat:attribute name="period" value="30000"/>
        <tat:attribute name="deadline" value="30000"/>
        <tat:attribute name="wcet" value="2075"/>
    </tat:object-details>
</tat:task>
<tat:task id="T_5_t0" name="Random Task 5">
    <tat:object-details>
        <tat:attribute name="period" value="30000"/>
        <tat:attribute name="deadline" value="30000"/>
        <tat:attribute name="wcet" value="5701"/>
    </tat:object-details>
</tat:task>
<tat:task id="T_6_t0" name="Random Task 6">
    <tat:object-details>
        <tat:attribute name="period" value="30000"/>
        <tat:attribute name="deadline" value="30000"/>
        <tat:attribute name="wcet" value="835"/>
    </tat:object-details>
</tat:task>
<tat:task id="T_7_t0" name="Random Task 7">
    <tat:object-details>
        <tat:attribute name="period" value="30000"/>
        <tat:attribute name="deadline" value="30000"/>
        <tat:attribute name="wcet" value="15173"/>
    </tat:object-details>
</tat:task>
<tat:task id="T_8_t0" name="Random Task 8">
    <tat:object-details>
        <tat:attribute name="period" value="30000"/>
        <tat:attribute name="deadline" value="30000"/>
        <tat:attribute name="wcet" value="3480"/>
    </tat:object-details>
</tat:task>
<tat:message from-task-id="T_1_t0" id="M_1_t0_2_t0" name="Message M_1_t0_2_t0" to-task-id
     ="T_2_t0">
    <tat:object-details>
        <tat:attribute name="size" value="5198"/>
    </tat:object-details>
</tat:message>
<tat:message from-task-id="T_1_t0" id="M_1_t0_3_t0" name="Message M_1_t0_3_t0" to-task-id
     ="T_3_t0">
    <tat:object-details>
        <tat:attribute name="size" value="2653"/>
    </tat:object-details>
</tat:message>
<tat:message from-task-id="T_1_t0" id="M_1_t0_4_t0" name="Message M_1_t0_4_t0" to-task-id
```

```
        ="T_4_t0">
        <tat:object-details>
            <tat:attribute name="size" value="3611"/>
        </tat:object-details>
    </tat:message>
    <tat:message from-task-id="T_1_t0" id="M_1_t0_5_t0" name="Message M_1_t0_5_t0" to-task-id
        ="T_5_t0">
        <tat:object-details>
            <tat:attribute name="size" value="7586"/>
        </tat:object-details>
    </tat:message>
    <tat:message from-task-id="T_1_t0" id="M_1_t0_6_t0" name="Message M_1_t0_6_t0" to-task-id
        ="T_6_t0">
        <tat:object-details>
            <tat:attribute name="size" value="5570"/>
        </tat:object-details>
    </tat:message>
    <tat:message from-task-id="T_1_t0" id="M_1_t0_7_t0" name="Message M_1_t0_7_t0" to-task-id
        ="T_7_t0">
        <tat:object-details>
            <tat:attribute name="size" value="1409"/>
        </tat:object-details>
    </tat:message>
    <tat:message from-task-id="T_1_t0" id="M_1_t0_8_t0" name="Message M_1_t0_8_t0" to-task-id
        ="T_8_t0">
        <tat:object-details>
            <tat:attribute name="size" value="936"/>
        </tat:object-details>
    </tat:message>
</tat:system>
```

B

Configuration XML Representation

B.1 Document Type Definition

```
<!ELEMENT tat:configuration (tat:system-configuration+)>
<!ATTLIST tat:configuration
         xmlns:tat CDATA #FIXED "http://www.cs.york.ac.uk/atu/tat">

<!ELEMENT tat:system-configuration (tat:object-configuration+)>
<!ATTLIST tat:system-configuration
         id ID #REQUIRED>

<!ELEMENT tat:object-configuration (tat:config-attribute+)>
<!ATTLIST tat:object-configuration
         id CDATA #REQUIRED>

<!ELEMENT tat:config-attribute EMPTY>
<!ATTLIST tat:config-attribute
         name CDATA #REQUIRED
         value CDATA #REQUIRED>
```

B.2 Example

```
<?xml version="1.0"?>
<tat:configuration xmlns:tat="http://www.cs.york.ac.uk/atu/tat">
    <tat:system-configuration id="Test_Case">
        <tat:object-configuration id="T_1_t0">
            <tat:config-attribute name="allocation_id" value="P1"/>
            <tat:config-attribute name="priority" value="2"/>
        </tat:object-configuration>
        <tat:object-configuration id="T_2_t0">
            <tat:config-attribute name="allocation_id" value="P1"/>
            <tat:config-attribute name="priority" value="5"/>
        </tat:object-configuration>
```

```
        <tat:object-configuration id="T_3_t0">
            <tat:config-attribute name="allocation_id" value="P1"/>
            <tat:config-attribute name="priority" value="10"/>
        </tat:object-configuration>
        <tat:object-configuration id="T_4_t0">
            <tat:config-attribute name="allocation_id" value="P2"/>
            <tat:config-attribute name="priority" value="15"/>
        </tat:object-configuration>
        <tat:object-configuration id="T_5_t0">
            <tat:config-attribute name="allocation_id" value="P2"/>
            <tat:config-attribute name="priority" value="6"/>
        </tat:object-configuration>
        <tat:object-configuration id="T_6_t0">
            <tat:config-attribute name="allocation_id" value="P2"/>
            <tat:config-attribute name="priority" value="3"/>
        </tat:object-configuration>
        <tat:object-configuration id="T_7_t0">
            <tat:config-attribute name="allocation_id" value="P1"/>
            <tat:config-attribute name="priority" value="13"/>
        </tat:object-configuration>
        <tat:object-configuration id="T_8_t0">
            <tat:config-attribute name="allocation_id" value="P2"/>
            <tat:config-attribute name="priority" value="14"/>
        </tat:object-configuration>
        <tat:object-configuration id="M_1_t0_2_t0">
            <tat:config-attribute name="allocation_id" value="N_P1"/>
            <tat:config-attribute name="priority" value="4"/>
        </tat:object-configuration>
        <tat:object-configuration id="M_1_t0_3_t0">
            <tat:config-attribute name="allocation_id" value="N_P1"/>
            <tat:config-attribute name="priority" value="1"/>
        </tat:object-configuration>
        <tat:object-configuration id="M_1_t0_4_t0">
            <tat:config-attribute name="allocation_id" value="N_1"/>
            <tat:config-attribute name="priority" value="9"/>
        </tat:object-configuration>
        <tat:object-configuration id="M_1_t0_5_t0">
            <tat:config-attribute name="allocation_id" value="N_1"/>
            <tat:config-attribute name="priority" value="7"/>
        </tat:object-configuration>
        <tat:object-configuration id="M_1_t0_6_t0">
            <tat:config-attribute name="allocation_id" value="N_1"/>
            <tat:config-attribute name="priority" value="8"/>
        </tat:object-configuration>
        <tat:object-configuration id="M_1_t0_7_t0">
            <tat:config-attribute name="allocation_id" value="N_P1"/>
            <tat:config-attribute name="priority" value="12"/>
        </tat:object-configuration>
        <tat:object-configuration id="M_1_t0_8_t0">
            <tat:config-attribute name="allocation_id" value="N_1"/>
            <tat:config-attribute name="priority" value="11"/>
        </tat:object-configuration>
    </tat:system-configuration>
</tat:configuration>
```

C

Gentap XML Parameter File

C.1 Example

```
<?xml version='1.0'?>
<settings>
    <hardware generate-hardware="yes">
        <tasks-per-processor>
            <range min="6.0" max="6.0"/><!-- Number, not percentage -->
        </tasks-per-processor>
        <number-of-networks>
            <range min="2.0" max="2.0"/>
        </number-of-networks>
        <processor-connectivity>
            <range min="50.0" max="50.0" granularity="1"/>
        </processor-connectivity>
        <network-bandwidth>
            <range min="1.0" max="1.0" granularity="1"/>
        </network-bandwidth>
        <network-latency>
            <range min="0.0" max="0.0" granularity="1"/>
        </network-latency>
        <processor-network-bandwidth>
            <range min="1024" max="1024" granularity="1"/>
        </processor-network-bandwidth>
        <processor-network-latency>
            <range min="0" max="0" granularity="1"/>
        </processor-network-latency>
    </hardware>
    <tasks>
        <number-of-tasks>
            <range min="48.0" max="48.0"/>
        </number-of-tasks>
        <period>
            <range min="10000.0" max="10000000.0" granularity="10000.0"/>
        </period>
        <utilisation-per-processor>
            <range min="50.0" max="50.0"/><!-- Utilisation -->
```

```
            <distribution type="beta" alpha="0.7"/>
        </utilisation-per-processor>
    </tasks>
    <messages>
        <utilisation-per-network>
            <range min="45.0" max="45.0"/>
            <distribution type="beta" alpha="0.7"/>
        </utilisation-per-network>
    </messages>
   <transactions>
        <tasks-per-transaction>
            <range min="6.0" max="6.0"/> <!-- Number of tasks involved in each transaction -->
        </tasks-per-transaction>
        <transaction-length>
            <range min="30.0" max="30.0"/> <!-- Percentage of number of tasks in transaction
                 -->
        </transaction-length>
        <messages-per-task>
            <range min="150.0" max="150.0"/> <!-- within each transaction -->
        </messages-per-task>
        <transaction-utilisation-distribution equal="true"/>
    </transactions>
</settings>
```

D

Search Parameters XML Representation

D.1 Document Type Definition

```
<!ELEMENT tat:search-control (tat:search-parameter-list|tat:cost-component-list|tat:objective-
    list)+>
<!ATTLIST tat:search-control
        xmlns:tat CDATA #FIXED "http://www.cs.york.ac.uk/atu/tat">
<!ELEMENT tat:search-parameter-list (tat:search-parameter*)>
<!ATTLIST tat:search-parameter-list
        table CDATA #IMPLIED>
<!ELEMENT tat:search-parameter EMPTY>
<!ATTLIST tat:search-parameter
        type CDATA #REQUIRED
        name CDATA #REQUIRED
        value CDATA #REQUIRED>
<!ELEMENT tat:cost-component-list (tat:cost-component*)>
<!ELEMENT tat:cost-component EMPTY>
<!ATTLIST tat:cost-component
        id ID #REQUIRED
        name CDATA #IMPLIED
        function CDATA #REQUIRED>
<!ELEMENT tat:objective-list (tat:objective+)>
<!ATTLIST tat:objective-list
        constraint-bias CDATA #IMPLIED>
<!ELEMENT tat:objective (tat:cost-component-ref+)>
<!ATTLIST tat:objective
        name CDATA #REQUIRED
        weight CDATA #REQUIRED
        type (objective|constraint) #IMPLIED
        threshold CDATA #IMPLIED>
<!ELEMENT tat:cost-component-ref EMPTY>
<!ATTLIST tat:cost-component-ref
        idref CDATA #REQUIRED
        weight CDATA #REQUIRED>
```

D.2 Example

```
<?xml version='1.0'?>
<tat:search-control xmlns:tat='http://www.cs.york.ac.uk/atu/tat'>
    <tat:search-parameter-list>
    <tat:search-parameter type="integer" name="seed" value="595455204"/>
        <tat:search-parameter type="string" name="search_method" value="local"/>
        <tat:search-parameter type="string" name="hill_climb_method" value="sa"/>
        <tat:search-parameter type="integer" name="max_moves" value="500000.0"/>
        <tat:search-parameter type="integer" name="moves_modify_param" value="1500.0"/>
        <tat:search-parameter type="real" name="temperature" value="0.005"/>
        <tat:search-parameter type="integer" name="sample_size" value="1.0"/>
        <tat:search-parameter type="integer" name="init_sample_size" value="3.0"/>
        <tat:search-parameter type="string" name="neighbourhood" value="allocation_and_priority
            "/>
        <tat:search-parameter type="string" name="multiple_system_method" value="
            parallelrepeatable"/>
        <tat:search-parameter type="boolean" name="try_diff_configs" value="true"/>
    </tat:search-parameter-list>
    <tat:cost-component-list>
        <tat:cost-component id="unrt" function="cost_unreachable_tasks"/>
        <tat:cost-component id="unio" function="cost_unconnected_io"/>
        <tat:cost-component id="unso" function="cost_unschedulable_objects" />
        <tat:cost-component id="aos" function="cost_all_object_sensitivity"/>
        <tat:cost-component id="lb" function="cost_load_balance"/>
        <tat:cost-component id="ous" function="cost_over_utilised_schedulers"/>
        <tat:cost-component id="dt" function="cost_distant_tasks"/>
        <tat:cost-component id="inco" function="cost_incorrect_transaction_order"/>
        <tat:cost-component id="ungo" function="cost_ungrouped_objects"/>
        <tat:cost-component id="tacsd" function="cost_task_alloc_system_diffs"/>
        <tat:cost-component id="macsd" function="cost_message_alloc_system_diffs"/>
        <tat:cost-component id="tpcsd" function="cost_task_priority_system_diffs"/>
        <tat:cost-component id="mpcsd" function="cost_message_priority_system_diffs"/>
    </tat:cost-component-list>
    <tat:objective-list>
        <tat:objective name="schedulability" weight="546" type="constraint" threshold="0">
            <tat:cost-component-ref idref="unso" weight="1" />
        </tat:objective>
        <tat:objective name="schedobj" weight="634" type="constraint">
            <tat:cost-component-ref idref="unrt" weight="3.0" />
            <tat:cost-component-ref idref="unio" weight="57393.0" />
            <tat:cost-component-ref idref="ungo" weight="3.0" />
            <tat:cost-component-ref idref="aos" weight="3.0" />
            <tat:cost-component-ref idref="lb" weight="3.0" />
            <tat:cost-component-ref idref="ous" weight="3.0" />
            <tat:cost-component-ref idref="dt" weight="32589.0" />
            <tat:cost-component-ref idref="inco" weight="3.0" />
        </tat:objective>
        <tat:objective name="change" weight="20.0" threshold="0">
            <tat:cost-component-ref idref="tacsd" weight="100" />
            <tat:cost-component-ref idref="macsd" weight="100" />
            <tat:cost-component-ref idref="tpcsd" weight="10" />
            <tat:cost-component-ref idref="mpcsd" weight="10" />
        </tat:objective>
    </tat:objective-list>
</tat:search-control>
```

E

Constraint File XML Representation

E.1 Document Type Definition

```
<!ELEMENT tat:config-constraints (tat:allocation-constraint|tat:status-constraint)*>
<!ATTLIST tat:config-constraints
         xmlns:tat CDATA #FIXED "http://www.cs.york.ac.uk/atu/tat">

<!ELEMENT tat:allocation-constraint (tat:allow-allocation|tat:forbid-allocation|tat:separate-
    from)+>
<!ATTLIST tat:allocation-constraint
         object-id CDATA #REQUIRED
         forbid-all CDATA #IMPLIED>

<!ELEMENT tat:allow-allocation EMPTY>
<!ATTLIST tat:allow-allocation
         allocation-id CDATA #REQUIRED>

<!ELEMENT tat:forbid-allocation EMPTY>
<!ATTLIST tat:forbid-allocation
         allocation-id CDATA #REQUIRED>

<!ELEMENT tat:separate-from EMPTY>
<!ATTLIST tat:separate-from
         object-id CDATA #REQUIRED>

<!ELEMENT tat:status-constraint EMPTY>
<!ATTLIST tat:status-constraint
         variable CDATA #REQUIRED
         set-all CDATA #IMPLIED
         object-id CDATA #IMPLIED>
```

E.2 Example

```
<?xml version='1.0'?>
<tat:config-constraints xmlns:tat="http://www.cs.york.ac.uk/atu/tat">
    <tat:allocation-constraint forbid-all="true" object-id="T_LFWheelSpeed">
        <tat:allow-allocation allocation-id="P1"/>
    </tat:allocation-constraint>
    <tat:allocation-constraint forbid-all="true" object-id="T_LBWheelSpeed">
        <tat:allow-allocation allocation-id="P2"/>
    </tat:allocation-constraint>
    <tat:allocation-constraint forbid-all="true" object-id="T_RFWheelSpeed">
        <tat:allow-allocation allocation-id="P3"/>
    </tat:allocation-constraint>
    <tat:allocation-constraint forbid-all="true" object-id="T_RBWheelSpeed">
        <tat:allow-allocation allocation-id="P4"/>
    </tat:allocation-constraint>
    <tat:allocation-constraint forbid-all="true" object-id="T_LFBrakeAct">
        <tat:allow-allocation allocation-id="P1"/>
    </tat:allocation-constraint>
    <tat:allocation-constraint forbid-all="true" object-id="T_LBBrakeAct">
        <tat:allow-allocation allocation-id="P2"/>
    </tat:allocation-constraint>
    <tat:allocation-constraint forbid-all="true" object-id="T_RFBrakeAct">
        <tat:allow-allocation allocation-id="P3"/>
    </tat:allocation-constraint>
    <tat:allocation-constraint forbid-all="true" object-id="T_RBBrakeAct">
        <tat:allow-allocation allocation-id="P4"/>
    </tat:allocation-constraint>
    <tat:allocation-constraint forbid-all="true" object-id="T_PedalSensor">
        <tat:allow-allocation allocation-id="P5"/>
    </tat:allocation-constraint>
</tat:config-constraints>
```

F

Utility Model XML Representation

F.1 Document Type Definition

```
<!ELEMENT tat:utilities (tat:feature+, tat:utility-tree+)>
<!ATTLIST tat:utilities
         xmlns:tat CDATA #FIXED "http://www.cs.york.ac.uk/atu/tat">

<!ELEMENT tat:feature (tat:feature-ref+)>
<!ATTLIST tat:feature
         id ID #REQUIRED
         type (component|subset) #REQUIRED
         threshold CDATA #IMPLIED>

<!ELEMENT tat:feature-ref EMPTY>
<!ATTLIST tat:feature-ref
         idref CDATA #REQUIRED>

<!ELEMENT tat:utility-tree (tat:utility-node)>
<!ATTLIST tat:utility-tree
         feature-id IDREF #REQUIRED>

<!ELEMENT tat:utility-node (tat:utility-value, tat:utility-value)>
<!ATTLIST tat:utility-node
         feature-ids IDREFS #REQUIRED
         require (all|any) #IMPLIED>

<!ELEMENT tat:utility-value (tat:utility-expr|tat:utility-node)>
<!ATTLIST tat:utility-value
         status (failed|working) #REQUIRED>

<!ELEMENT tat:utility-expr (#PCDATA)>
```

F.2 Example

```
<?xml version="1.0" ?>
<tat:utilities xmlns:tat="http://www.cs.york.ac.uk/atu/tat">
```

```
<tat:feature id="T1" type="component">
    <tat:feature-ref idref="T_1"/>
</tat:feature>
<tat:feature id="T2" type="component">
    <tat:feature-ref idref="T_2"/>
</tat:feature>
<tat:feature id="T3" type="component">
    <tat:feature-ref idref="T_3"/>
</tat:feature>
<tat:feature id="T4" type="component">
    <tat:feature-ref idref="T_4"/>
</tat:feature>
<tat:feature id="T5" type="component">
    <tat:feature-ref idref="T_5"/>
</tat:feature>
<tat:feature id="T6" type="component">
    <tat:feature-ref idref="T_6"/>
</tat:feature>
<tat:feature id="T7" type="component">
    <tat:feature-ref idref="T_7"/>
</tat:feature>
<tat:feature id="T8" type="component">
    <tat:feature-ref idref="T_8"/>
</tat:feature>
<tat:feature id="System" type="subset" threshold="0.4">
    <tat:feature-ref idref="T1"/>
    <tat:feature-ref idref="T2"/>
    <tat:feature-ref idref="T3"/>
    <tat:feature-ref idref="T4"/>
    <tat:feature-ref idref="T5"/>
    <tat:feature-ref idref="T6"/>
    <tat:feature-ref idref="T7"/>
    <tat:feature-ref idref="T8"/>
</tat:feature>
<tat:utility-tree feature-id="T1">
    <tat:utility-node feature-ids="T1">
        <tat:utility-value status="failed">
            <tat:utility-expr>0</tat:utility-expr>
        </tat:utility-value>
        <tat:utility-value status="working">
            <tat:utility-expr>1</tat:utility-expr>
        </tat:utility-value>
    </tat:utility-node>
</tat:utility-tree>
<tat:utility-tree feature-id="T2">
    <tat:utility-node feature-ids="T2">
        <tat:utility-value status="failed">
            <tat:utility-expr>0</tat:utility-expr>
        </tat:utility-value>
        <tat:utility-value status="working">
            <tat:utility-expr>1</tat:utility-expr>
        </tat:utility-value>
    </tat:utility-node>
</tat:utility-tree>
<tat:utility-tree feature-id="T3">
    <tat:utility-node feature-ids="T3">
        <tat:utility-value status="failed">
            <tat:utility-expr>0</tat:utility-expr>
```

```
        </tat:utility-value>
        <tat:utility-value status="working">
            <tat:utility-expr>1</tat:utility-expr>
        </tat:utility-value>
    </tat:utility-node>
</tat:utility-tree>
<tat:utility-tree feature-id="T4">
    <tat:utility-node feature-ids="T4">
        <tat:utility-value status="failed">
            <tat:utility-expr>0</tat:utility-expr>
        </tat:utility-value>
        <tat:utility-value status="working">
            <tat:utility-expr>1</tat:utility-expr>
        </tat:utility-value>
    </tat:utility-node>
</tat:utility-tree>
<tat:utility-tree feature-id="T5">
    <tat:utility-node feature-ids="T5">
        <tat:utility-value status="failed">
            <tat:utility-expr>0</tat:utility-expr>
        </tat:utility-value>
        <tat:utility-value status="working">
            <tat:utility-expr>1</tat:utility-expr>
        </tat:utility-value>
    </tat:utility-node>
</tat:utility-tree>
<tat:utility-tree feature-id="T6">
    <tat:utility-node feature-ids="T6">
        <tat:utility-value status="failed">
            <tat:utility-expr>0</tat:utility-expr>
        </tat:utility-value>
        <tat:utility-value status="working">
            <tat:utility-expr>1</tat:utility-expr>
        </tat:utility-value>
    </tat:utility-node>
</tat:utility-tree>
<tat:utility-tree feature-id="T7">
    <tat:utility-node feature-ids="T7">
        <tat:utility-value status="failed">
            <tat:utility-expr>0</tat:utility-expr>
        </tat:utility-value>
        <tat:utility-value status="working">
            <tat:utility-expr>1</tat:utility-expr>
        </tat:utility-value>
    </tat:utility-node>
</tat:utility-tree>
<tat:utility-tree feature-id="T8">
    <tat:utility-node feature-ids="T8">
        <tat:utility-value status="failed">
            <tat:utility-expr>0</tat:utility-expr>
        </tat:utility-value>
        <tat:utility-value status="working">
            <tat:utility-expr>1</tat:utility-expr>
        </tat:utility-value>
    </tat:utility-node>
</tat:utility-tree>
<tat:utility-tree feature-id="System">
    <tat:utility-node feature-ids="T1">
```

```
            <tat:utility-value status="failed">
                <tat:utility-expr>0</tat:utility-expr>
            </tat:utility-value>
            <tat:utility-value status="working">
                <tat:utility-node feature-ids="T2">
                    <tat:utility-value status="failed">
                        <tat:utility-node feature-ids="T5">
                            <tat:utility-value status="failed">
                                <tat:utility-expr>0</tat:utility-expr>
                            </tat:utility-value>
                            <tat:utility-value status="working">
                                <tat:utility-node require="any" feature-ids="T7 T8">
                                    <tat:utility-value status="failed">
                                        <tat:utility-expr>0</tat:utility-expr>
                                    </tat:utility-value>
                                    <tat:utility-value status="working">
                                        <tat:utility-expr>sum(mul(0.25,U(T8)),mul(0.25,U(T7)))</
                                            tat:utility-expr>
                                    </tat:utility-value>
                                </tat:utility-node>
                            </tat:utility-value>
                        </tat:utility-node>
                    </tat:utility-value>
                    <tat:utility-value status="working">
                        <tat:utility-node feature-ids="T3">
                            <tat:utility-value status="failed">
                                <tat:utility-expr>0</tat:utility-expr>
                            </tat:utility-value>
                            <tat:utility-value status="working">
                                <tat:utility-node require="any" feature-ids="T4 T6 T7 T8">
                                    <tat:utility-value status="failed">
                                        <tat:utility-expr>0</tat:utility-expr>
                                    </tat:utility-value>
                                    <tat:utility-value status="working">
                                        <tat:utility-expr>sum(mul(0.25,U(T8)),mul(0.25,U(T7)),mul
                                            (0.25,U(T4)),mul(0.25,U(T6)))</tat:utility-expr>
                                    </tat:utility-value>
                                </tat:utility-node>
                            </tat:utility-value>
                        </tat:utility-node>
                    </tat:utility-value>
                </tat:utility-node>
            </tat:utility-value>
        </tat:utility-node>
    </tat:utility-tree>
</tat:utilities>
```

References

[1] D. L. Parnas, "Software aging," in *Proceedings of the 16th international conference on Software engineering (ICSE 94)*, pp. 279–287, 1994.

[2] C. C. Price, "Task allocation in distributed systems: A survey of practical strategies," in *ACM 82: Proceedings of the ACM '82 conference*, (New York, NY, USA), pp. 176–181, ACM Press, 1982.

[3] K. Ramamritham, "Allocation and scheduling of precedence-related periodic tasks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 6, no. 4, pp. 412–420, 1995.

[4] K. Tindell, A. Burns, and A. Wellings, "Allocating hard real-time tasks: An NP-hard problem made easy," *Real-Time Systems*, vol. 4, no. 2, pp. 145–165, 1992.

[5] P. Naur and B. Randell, eds., *Software Engineering: Report on a conference sponsored by the NATO Science Committee*, October 1968.

[6] T. M. Pigoski, *Practical Software Maintenance*. Wiley, 1997.

[7] S. Rosenberg, *Dreaming In Code*, ch. Epilogue, pp. 346–349. Crown Publishers, 2007.

[8] K. H. Bennett and V. T. Rajlich, "Software maintenance and evolution: a roadmap," in *Proceedings of the Conference on The Future of Software Engineering (ICSE 00)*, pp. 73–87, 2000.

[9] B. P. Lientz and B. E. Swanson, *Software Maintenance Management*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1980.

[10] J. Mchale, "It's not your father's space shuttle any more," *Military & Aerospace Electronics*, April 2001. `http://mae.pennnet.com/display_article/97983/32/ARTCL/none/none/1/It%27s-not-your-father%27s-Space-Shuttle-any-more/`.

[11] W. W. Royce, "Managing the development of large software systems," in *IEEE WESCON*, pp. 328–339, 1970.

[12] B. W. Boehm, "A spiral model of software development and enhancement," *Computer*, vol. 21, pp. 61–72, May 1988.

[13] I. Sommerville, *Software Engineering*. Addison Wesley, eighth ed., 2006.

[14] N. F. Schneidewind, "The state of software maintenance," *IEEE Trans. Softw. Eng.*, vol. 13, no. 3, pp. 303–310, 1987.

[15] B. W. Boehm, *Software Engineering Economics*. Prentice Hall PTR, 1981.

[16] K. Schwaber and M. Beedle, *Agile Software Development with SCRUM*. Prentice Hall, February 2002.

[17] A. Cockburn, *Agile Software Development*. Addison-Wesley Professional, December 2001.

[18] K. Beck, *Extreme Programming Explained: Embrace Change*. Addison-Wesley Professional, October 1999.

[19] D. C. Schmidt, "Model-driven engineering," *Computer*, vol. 39, no. 2, pp. 25–31, 2006.

[20] B. Selic, "The pragmatics of model-driven development," *Software, IEEE*, vol. 20, no. 5, pp. 19–25, 2003.

[21] A. Fuggetta, "A classification of CASE technology," *Computer*, vol. 26, no. 12, pp. 25–38, 1993.

[22] M. Harman, "The current state and future of search based software engineering," in *Future of Software Engineering (FOSE '07)*, pp. 342–357, 2007.

[23] M. Bowman, L. C. Briand, and Y. Labiche, "Multi-objective genetic algorithm to support class responsibility assignment," in *IEEE International Conference on Software Maintenance (ICSM)*, pp. 124–133, 2007.

[24] B. W. Boehm, J. R. Brown, and M. Lipow, "Quantitative evaluation of software quality," in *Proceedings of the 2nd International Conference on Software engineering (ICSE 76)*, pp. 592–605, 1976.

[25] R. Kazman, L. J. Bass, M. Webb, and G. D. Abowd, "SAAM: A method for analyzing the properties of software architectures," in *International Conference on Software Engineering*, pp. 81–90, 1994.

[26] A. Sangiovanni-Vincentelli and M. Di Natale, "Embedded system design for automotive applications," *Computer*, vol. 40, no. 10, pp. 42–51, 2007.

[27] P. Popp, M. Di Natale, P. Giusto, S. Kanajan, and C. Pinello, "Towards a methodology for the quantitative evaluation of automotive architectures," in *Design, Automation & Test in Europe Conference & Exhibition (DATE '07)*, pp. 1–6, 2007.

[28] E. Fricke and A. P. Schulz, "Design for changeability (dfc): Principles to enable changes in systems throughout their entire lifecycle," *Systems Engineering*, vol. 8, no. 4, 2005.

[29] A. M. Ross, D. H. Rhodes, and D. E. Hastings, "Defining changeability: Reconciling flexibility, adaptability, scalability, modifiability, and robustness for maintaining system lifecycle value," *Systems Engineering*, vol. 11, no. 3, pp. 246–262, 2008.

[30] J. Real and A. Crespo, "Mode change protocols for real-time systems: A survey and a new proposal," *Real-Time Systems*, vol. 26, no. 2, pp. 161–197, 2004.

[31] S. Mitra, N. Seifert, M. Zhang, Q. Shi, and K. S. Kim, "Robust system design with built-in soft-error resilience," *Computer*, vol. 38, no. 2, pp. 43–52, 2005.

[32] D. Garlan, S.-W. Cheng, and B. Schmerl, "Increasing system dependability through architecture-based self-repair," in *Architecting Dependable Systems*, pp. 61–89, Springer, 2003.

[33] M. Barbacci, M. H. Klein, T. A. Longstaff, and C. B. Weinstock, "Quality attributes," Tech. Rep. CMU/SEI-95-TR-021, Carnegie Mellon Software Engineering Institute, 1995.

[34] M. A. Babar, L. Zhu, and R. Jeffery, "A framework for classifying and comparing software architecture evaluation methods," in *Proceedings of the 2004 Australian Software Engineering Conference (ASWEC'04)*, pp. 309–318, 2004.

[35] R. Kazman, M. Klein, and P. Clements, "Evaluating software architectures for real-time systems," *Annals of Software Engineering*, vol. 7, no. 1-4, pp. 71–93, 1999.

[36] A. Brown and J. McDermid, "The art and science of software architecture," in *Software Architecture*, pp. 237–256, Springer, 2007.

[37] N. Storey, *Safety Critical Computer Systems*. Addison Wesley, August 1996.

[38] N. Leveson, *Safeware : System Safety and Computers*. Addison-Wesley, 1995.

[39] H. Kopetz, *Real-Time Systems : Design Principles for Distributed Embedded Applications*. Springer, April 1997.

[40] J. W. Liu, *Real-Time Systems*. Prentice Hall, 2000.

[41] J. A. Cook, I. V. Kolmanovsky, D. Mcnamara, E. C. Nelson, and K. V. Prasad, "Control, computing and communications: Technologies for the twenty-first century model t," *Proceedings of the IEEE*, vol. 95, no. 2, pp. 334–355, 2007.

[42] I. Moir and A. Seabridge, *Civil Avionics Systems*. Wiley, April 2006.

[43] N. Navet, Y. Song, F. Simonot-Lion, and C. Wilwert, "Trends in automotive communication systems," *Proceedings of the IEEE*, vol. 93, no. 6, pp. 1204–1223, 2005.

[44] M. Broy, I. H. Kruger, A. Pretschner, and C. Salzmann, "Engineering automotive software," *Proceedings of the IEEE*, vol. 95, no. 2, pp. 356–373, 2007.

[45] R. Bosch, "CAN specification version 2.0," September 1991.

[46] "Automotive Open System Architecture (AUTOSAR)." `http://www.autosar.org/`.

[47] C. R. Spitzer, *Digital Avionics Systems: Principles and Practices*. McGraw-Hill Companies, second ed., 1993.

[48] L. Sha, "Real-time virtual machines for avionics software porting and development," in *Real-Time and Embedded Computing Systems and Applications*, vol. 2968, pp. 123–135, Springer, 2004.

[49] M. J. Morgan, "Integrated modular avionics for next generation commercial airplanes," *IEEE Aerospace and Electronic Systems Magazine*, vol. 6, no. 8, pp. 9–12, 1991.

[50] G. Jolliffe, "Producing a safety case for IMA blueprints," in *The 24th Digital Avionics Systems Conference (DASC 05)*, vol. 2, pp. 8.C.1–1–8.C.1–14, 2005.

[51] D. T. Peng, K. Shin, and T. Abdelzaher, "Assignment and scheduling communicating periodic tasks in distributed real-time systems," *Software Engineering*, vol. 23, no. 12, pp. 745–758, 1997.

[52] F. Bicking, B. Conrard, and J. M. Thiriet, "Integration of dependability in a task allocation problem," *IEEE Transactions on Instrumentation and Measurement*, vol. 53, no. 6, pp. 1455–1463, 2004.

[53] S. T. Cheng and A. K. Agrawala, "Allocation and scheduling of real-time periodic tasks with relative timing constraints," Tech. Rep. CS-TR-3402, Department of Computer Science, University of Maryland, 1995.

[54] G. Attiya and Y. Hamam, "Task allocation for maximizing reliability of distributed systems: A simulated annealing approach," *Journal of Parallel and Distributed Computing*, vol. 66, pp. 1259–1266, October 2006.

[55] P.-Y. Yin, S.-S. Yu, P.-P. Wang, and Y.-T. Wang, "Multi-objective task allocation in distributed computing systems by hybrid particle swarm optimization," *Applied Mathematics and Computation*, vol. 184, pp. 407–420, January 2007.

[56] A. Metzner and C. Herde, "RTSAT – an optimal and efficient approach to the task allocation problem in distributed architectures," in *Proceedings of the 27th IEEE International Real-Time Systems Symposium (RTSS 06)*, pp. 147–158, 2006.

[57] S. K. Baruah, A. K. Mok, and L. E. Rosier, "Preemptively scheduling hard-real-time sporadic tasks on one processor," in *Proceedings of the 11th Real-Time Systems Symposium*, pp. 182–190, 1990.

[58] K. Tindell, "Adding time-offsets to schedulability analysis," Tech. Rep. YCS-1994-221, Department of Computer Science, University of York, 1994.

[59] J. Palencia and M. G. Harbour, "Schedulability analysis for tasks with static and dynamic offsets," in *Proceedings of the IEEE Real-Time Systems Symposium*, pp. 26–37, 1998.

[60] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *Journal of the ACM*, vol. 20, no. 1, pp. 46–61, 1973.

[61] J. P. Lehoczky, "Fixed priority scheduling of periodic task sets with arbitrary deadlines," in *Proceedings of the 11th Real-Time Systems Symposium*, pp. 201–209, 1990.

[62] N. C. Audsley, A. Burns, M. F. Richardson, and A. J. Wellings, "Hard real-time scheduling: The deadline-monotonic approach," in *Proceedings of the 8th IEEE Workshop on Real-Time Operating Systems*, May 1991.

[63] R. I. Davis and A. Burns, "Hierarchical fixed priority pre-emptive scheduling," in *Proceedings of 26th IEEE International Real-Time Systems Symposium*, pp. 389–398, 2005.

[64] H. Lonn and J. Axelsson, "A comparison of fixed-priority and static cyclic scheduling for distributed automotive control applications," in *Proceedings of the 11th Euromicro Conference on Real-Time Systems*, pp. 142–149, 1999.

[65] P. Marti, J. M. Fuertes, G. Fohler, and K. Ramamritham, "Jitter compensation for real-time control systems," in *Proceedings of 22nd IEEE Real-Time Systems Symposium*, pp. 39–48, 2001.

[66] C. D. Locke, "Software architecture for hard real-time applications: cyclic executives vs. fixed - priority executives," *Real-Time Systems*, vol. 4, pp. 37–53, March 1992.

[67] N. Audsley, K. Tindell, and A. Burns, "The end of the line for static cyclic scheduling?," in *Proceedings of the 5th Euromicro Workshop on Real-Time Systems*, pp. 36–41, 1993.

[68] J. Y. T. Leung and J. Whitehead, "On the complexity of fixed-priority scheduling of periodic, real-time tasks," *Performance Evaluation*, vol. 2, pp. 237–250, 1982.

[69] M. Joseph and P. Pandya, "Finding response times in a real-time system," *The Computer Journal*, vol. 29, pp. 390–395, May 1986.

[70] R. Davis, A. Burns, R. Bril, and J. Lukkien, "Controller area network (can) schedulability analysis: Refuted, revisited and revised," *Real-Time Systems*, vol. 35, pp. 239–272, April 2007.

[71] G. C. Buttazzo, "Rate monotonic vs. EDF: Judgment day," *Real-Time Systems*, vol. 29, no. 1, pp. 5–26, 2005.

[72] R. K. Abbott and H. G. Molina, "Scheduling real-time transactions: a performance evaluation," *ACM Trans. Database Syst.*, vol. 17, no. 3, pp. 513–560, 1992.

[73] C. Diederichs, U. Margull, F. Slomka, and G. Wirrer, "An application-based EDF scheduler for OSEK/VDX," in *DATE '08: Proceedings of the conference on Design, automation and test in Europe*, pp. 1045–1050, 2008.

[74] T. Pop, P. Eles, and Z. Peng, "Holistic scheduling and analysis of mixed time/event-triggered distributed embedded systems," in *CODES '02: Proceedings of the tenth international symposium on Hardware/software codesign*, pp. 187–192, 2002.

[75] M. Di Natale, W. Zheng, C. Pinello, P. Giusto, and A. S. Vincentelli, "Optimizing end-to-end latencies by adaptation of the activation events in distributed automotive systems," in *Proceedings of 13th IEEE Real Time and Embedded Technology and Applications Symposium (RTAS 07)*, pp. 293–302, 2007.

[76] K. Tindell, A. Burns, and A. J. Wellings, "Analysis of hard real-time communications," *Real-Time Systems*, vol. 9, pp. 147–171, September 1995.

[77] Y.-H. Lee, D. Kim, M. Younis, and J. Zhou, "Scheduling tool and algorithm for integrated modular avionics systems," in *Proceedings of the 19th Digital Avionics Systems Conferences (DASC)*, vol. 1, pp. 1C2/1–1C2/8, 2000.

[78] G. Leen and D. Hefferman, "TTCAN: a new time-triggered controller area network," *Microprocessors and Microsystems*, vol. 26, pp. 77–94, March 2002.

[79] X. Qiao, K.-F. Wang, Y. Sun, W.-L. Huang, and F.-Y. Wang, "A genetic algorithms based optimization for TTCAN," in *IEEE International Conference on Vehicular Electronics and Safety (ICVES)*, pp. 1–7, 2007.

[80] S. Kodase, S. Wang, Z. Gu, and K. G. Shin, "Improving scalability of task allocation and scheduling in large distributed real-time systems using shared buffers.," in *Proceedings of 9th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS '03)*, pp. 181–188, 2003.

[81] W. Zheng, Q. Zhu, M. Di Natale, and A. S. Vincentelli, "Definition of task allocation and priority assignment in hard real-time distributed systems," in *Proceedings 28th IEEE International Real-Time Systems Symposium (RTSS 2007)*, pp. 161–170, 2007.

[82] S. Baruah and N. Fisher, "The partitioned multiprocessor scheduling of deadline-constrained sporadic task systems," *IEEE Transactions on Computers*, vol. 55, no. 7, pp. 918–923, 2006.

[83] M. Bertogna, M. Cirinei, and G. Lipari, "Schedulability analysis of global scheduling algorithms on multiprocessor platforms," *IEEE Transactions on Parallel Distributed Systems*, vol. 20, no. 4, pp. 553–566, 2009.

[84] M. Bertogna, M. Cirinei, and G. Lipari, "New schedulability tests for real-time task sets scheduled by deadline monotonic on multiprocessors," in *Principles of Distributed Systems*, pp. 306–321, Springer Berlin / Heidelberg, 2006.

[85] K. Tindell and J. Clark, "Holistic schedulability analysis for distributed hard real-time systems," *Microprocessing and Microprogramming - Euromicro Journal*, vol. 40, no. 2-3, pp. 117–134, 1994.

[86] J. C. Palencia and M. G. Harbour, "Exploiting precedence relations in the schedulability analysis of distributed real-time systems," in *Proceedings of the 20th IEEE Real-Time Systems Symposium (RTSS 99)*, 1999.

[87] O. Redell, "Analysis of tree-shaped transactions in distributed real time systems," in *Proceedings of the 16th Euromicro Conference on Real-Time Systems (ECRTS 2004)*, pp. 239–248, 2004.

[88] J. P. Kany and S. H. Madsen, "Design optimisation of fault-tolerant event-triggered embedded systems," Master's thesis, Informatics and Mathematical Modelling, Technical University of Denmark, DTU, 2007.

[89] T. Henzinger, B. Horowitz, and C. Kirsch, "Giotto: A time-triggered language for embedded programming," in *Embedded Software*, pp. 166–184, Springer, 2001.

[90] H. Kopetz, R. Nossal, R. Hexel, A. Krüger, D. Millinger, R. Pallierer, C. Temple, and M. Krug, "Mode handling in the time-triggered architecture," *Control Engineering Practice*, vol. 6, pp. 61–66, January 1998.

[91] G. Fohler, "Realizing changes of operational modes with pre run-time scheduled hard real-time systems," in *Proceedings of Second International Workshop on Responsive Computer Systems*, Saitama, Japan, October 1992.

[92] T. Stauner, O. Müller, and M. Fuchs, "Using hytech to verify an automative control system," in *Proceedings of the International Workshop on Hybrid and Real-Time Systems*, (London, UK), pp. 139–153, Springer-Verlag, 1997.

[93] P. Pedro, *Scheduling of Mode Changes in Flexible Real-Time Distributed Systems*. PhD thesis, Department of Computer Scheduling, University of York, 1999.

[94] E. A. Strunk, J. C. Knight, and M. A. Aiello, "Distributed reconfigurable avionics architectures," in *The 23rd Digital Avionics Systems Conference (DASC 04)*, vol. 2, pp. 10.B.4–1–10.B.4–10, 2004.

[95] A. Burns and A. Wellings, *Real-Time Systems and Programming Languages: Ada, Real-Time Java and C/Real-Time POSIX*. Addison Wesley, fourth ed., May 2009.

[96] H. Kopetz, H. Kantz, G. Grunsteidl, P. Puschner, and J. Reisinger, "Tolerating transient faults in mars," in *20th International Symposium on Fault-Tolerant Computing (FTCS-20)*, pp. 466–473, 1990.

[97] H. Kopetz, A. Damm, C. Koza, M. Mulazzani, W. Schwabl, C. Senft, and R. Zainlinger, "Distributed fault-tolerant real-time systems: The mars approach," *IEEE Micro*, vol. 9, no. 1, pp. 25–40, 1989.

[98] R. D. Schlichting and F. B. Schneider, "Fail-stop processors: An approach to designing fault-tolerant computing systems," *Computer Systems*, vol. 1, no. 3, pp. 222–238, 1983.

[99] A. Burns, R. Davis, and S. Punnekkat, "Feasibility analysis of fault-tolerant real-time task sets," in *Proceedings of the Eighth Euromicro Workshop on Real-Time Systems*, pp. 29–33, 1996.

[100] W. Jia and W. Zhou, "Reliability and replication techniques," in *Distributed Network Systems*, Network Theory and Applications, pp. 213–254, Springer US, 2005.

[101] C. P. Shelton, P. Koopman, and W. Nace, "A framework for scalable analysis and design of system-wide graceful degradation in distributed embedded systems," in *Proceedings of the Eighth International Workshop on Object-Oriented Real-Time Dependable Systems (WORDS 2003)*, pp. 156–163, 2003.

[102] C. Reeves, ed., *Modern Heuristic Techniques for Combinatorial Problems*. Oxford, England: Blackwell Scientific Publishing, 1993.

[103] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*. Pearson Education, 2003.

[104] J. Clark, J. J. Dolado, M. Harman, R. Hierons, B. Jones, Lumkin, S. Mancoridis, K. Rees, M. Roper, and M. Shepperd, "Reformulating software engineering as a search problem," *IEE Proceedings - Software*, vol. 150, pp. 161–175, June 2003.

[105] W. W. Chu and M. T. Lan, "Task allocation and precedence relations for distributed real-time systems," *IEEE Transactions on Computers*, vol. 36, no. 6, pp. 667–679, 1987.

[106] V. M. Lo, "Heuristic algorithms for task assignment in distributed systems," *IEEE Transactions on Computers*, vol. 37, no. 11, pp. 1384–1397, 1988.

[107] B. Ucar, C. Aykanat, K. Kaya, and M. Ikinci, "Task assignment in heterogeneous computing systems," *Journal of Parallel and Distributed Computing*, vol. 66, pp. 32–46, January 2006.

[108] E. Lawler and D. Wood, "Branch-and-bound methods: A survey," *Operations Research*, vol. 14, no. 4, pp. 699–719, 1966.

[109] K. Ramamritham, "Allocation and scheduling of complex periodic tasks," in *Proceedings of 10th International Conference on Distributed Computing Systems*, pp. 108–115, 1990.

[110] D. Thierens, "Selection schemes, elitist recombination, and selection intensity," in *Proceedings of the 7th International Conference on Genetic Algorithms*, pp. 152–159, 1998.

[111] S. Kirkpatrick, C. Gelatt, and M. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 1983.

[112] Y. Nourani and B. Andresen, "A comparison of simulated annealing cooling strategies," *Journal of Physics A: Mathematical and General*, pp. 8373–8385, 1998.

[113] L. Ingber, "Simulated annealing: Practice versus theory," *Mathematical And Computer Modelling*, vol. 18, no. 11, pp. 29–57, 1993.

[114] F. Glover and M. Laguna, "Tabu search," in *Modern Heuristic Techniques for Combinatorial Problems* (C. Reeves, ed.), Blackwell Scientific Publishing, 1993.

[115] E. Nowicki and C. Smutnicki, "An advanced tabu search algorithm for the job shop problem," *Journal of Scheduling*, vol. 8, pp. 145–159, April 2005.

[116] J. H. Holland, *Adaptation in Natural and Artificial Systems.* The University of Michigan Press, 1975.

[117] C. A. C. Coello, "Constraint-handling using an evolutionary multiobjective optimization technique," *Civil Engineering and Environmental Systems*, vol. 17, pp. 319–346, 2000.

[118] M. C. Fonseca and P. J. Fleming, "Multi-objective genetic algorithms made easy: Selection, sharing and mating restrictions," in *Proceedings of the 1st International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications*, pp. 45–52, 1995.

[119] T. Blickle and L. Thiele, "A comparison of selection schemes used in genetic algorithms," Tech. Rep. TIK-Report-11, Computer Engineering and Communication Networks Lab (TIK), Swiss Federal Institute of Technology (ETH), Gloriastrasse 35, 8092 Zurich, Switzerland, December 1995.

[120] H.-G. Beyer and H.-P. Schwefel, "Evolution strategies – a comprehensive introduction," *Natural Computing*, vol. 1, pp. 3–52, March 2002.

[121] J. Axelsson, "Three search strategies for architecture synthesis and partitioning of real-time systems," Tech. Rep. R-96-32, Department of Computer and Information Science, Linköping University, Sweden, 1996.

[122] M. Nicholson, *Selecting a topology for safety-critical real-time control systems.* PhD thesis, University of York, 1998.

[123] P.-E. Hladik, H. Cambazard, A.-M. Deplanche, and N. Jussien, "Solving a real-time allocation problem with constraint programming," *Journal of Systems and Software*, vol. 81, pp. 132–149, January 2008.

[124] J. A. Bannister and K. S. Trivedi, "Task allocation in fault-tolerant distributed systems," *Acta Informatica*, vol. 20, pp. 261–281, September 1983.

[125] K. Deb, *Multi-Objective Optimization Using Evolutionary Algorithms.* Wiley, first ed., June 2001.

[126] E. Zitzler, M. Laumanns, and S. Bleuler, "A tutorial on evolutionary multiobjective optimization," in *Metaheuristics for Multiobjective Optimisation* (X. Gandibleux, M. Sevaux, K. Sörensen, and V. T'kindt, eds.), vol. 535 of

Lecture Notes in Economics and Mathematical Systems, pp. 3–37, Springer, 2004.

[127] D. J. Schaffer, *Multiple Objective Optimization with Vector Evaluated Genetic Algorithms*. PhD thesis, Vanderbilt University, 1984.

[128] J. D. Knowles and D. Corne, "Approximating the nondominated front using the pareto archived evolution strategy," *Evolutionary Computation*, vol. 8, no. 2, pp. 149–172, 2000.

[129] R. P. Dick and N. K. Jha, "Mogac: a multiobjective genetic algorithm for hardware-software cosynthesis of distributed embedded systems," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 17, no. 10, pp. 920–935, 1998.

[130] A. Hamann, R. Racu, and R. Ernst, "Multi-dimensional robustness optimization in heterogeneous distributed embedded systems," in *Proceedings of the 13th IEEE Real Time and Embedded Technology and Applications Symposium (RTAS '07)*, (Los Alamitos, CA, USA), pp. 269–280, IEEE Computer Society, 2007.

[131] D. E. Perry and A. L. Wolf, "Foundations for the study of software architecture," *ACM SIGSOFT Software Engineering Notes*, vol. 17, no. 4, pp. 40–52, 1992.

[132] N. Lassing, D. Rijsenbrij, and H. van Vliet, "How well can we predict changes at architecture design time?," *Journal of Systems and Software*, vol. 65, no. 2, pp. 141–153, 2003.

[133] N. H. Lassing, D. B. B. Rijsenbrij, and J. C. van Vliet, "Viewpoints on modifiability," *International Journal of Software Engineering and Knowledge Engineering*, vol. 11, no. 4, pp. 453–478, 2001.

[134] D. Soni, R. L. Nord, and C. Hofmeister, "Software architecture in industrial applications," in *International Conference on Software Engineering*, pp. 196–207, 1995.

[135] P. C. Clements, "A survey of architecture description languages," in *Proceedings of the 8th International Workshop on Software Specification and Design*, 1996.

[136] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*. Addison-Wesley Longman Publishing Co., Inc., second ed., 2003.

[137] W. E. Beregi, "Architecture prototyping in the software engineering environment," *IBM Systems Journal*, vol. 23, no. 1, pp. 4–18, 1984.

[138] P. Kruchten, "The 4+1 view model of architecture," *IEEE Software*, vol. 12, no. 6, pp. 42–50, 1995.

[139] J. Daniels, P. W. Werner, and A. T. Bahill, "Quantitative methods for tradeoff analyses," *Systems Engineering*, vol. 4, no. 3, pp. 190–212, 2001.

[140] R. L. Keeney and H. Raiffa, *Decisions with Multiple Objectives: Preferences and Value Trade-offs*. Cambridge University Press, Cambridge, 1993.

[141] R. Kazman, G. Abowd, L. Bass, and P. Clements, "Scenario-based analysis of software architecture," *IEEE Software*, vol. 13, pp. 47–55, November 1996.

[142] R. Kazman, M. Barbacci, M. Klein, S. J. Carrière, and S. G. Woods, "Experience with performing architecture tradeoff analysis," in *Proceedings of the 21st international conference on Software Engineering (ICSE '99)*, pp. 54–63, IEEE Computer Society Press, 1999.

[143] R. Kazman, J. S. Carrière, and S. G. Woods, "Toward a discipline of scenario-based architectural engineering," *Annals of Software Engineering*, vol. 9, no. 1-4, pp. 5–33, 2000.

[144] L. Dobrica and E. Niemelä, "A survey on software architecture analysis methods," *IEEE Transactions on Software Engineering*, vol. 28, no. 7, pp. 638–653, 2002.

[145] F. Jensen, *An introduction to Bayesian networks*. UCL Press, 1996.

[146] J. van Gurp and J. Bosch, "Saabnet: Managing qualitative knowledge in software architecture assessment.," in *ECBS*, 2000.

[147] A. Parakhine, T. O'Neill, and J. Leaney, "Application of bayesian networks to architectural optimisation," in *14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS '07)*, pp. 37–44, 2007.

[148] H. A. Thompson, A. J. Chipperfield, P. J. Fleming, and C. Legge, "Distributed aero-engine control systems architecture selection using multi-objective optimisation," *Control Engineering Practice*, vol. 7, pp. 655–664, May 1999.

[149] Z. Stephenson and J. McDermid, "Deriving architectural flexibility requirements in safety-critical systems," *IEE Proceedings - Software*, vol. 152, no. 4, pp. 143–152, 2005.

[150] R. P. Dick, D. L. Rhodes, and W. Wolf, "TGFF: task graphs for free," in *Proceedings of the Sixth International Workshop on Hardware/Software Codesign (CODES/CASHE '98)*, pp. 97–101, March 1998.

[151] K. Vallerio, *Task Graphs for Free (TGFF v3.0)*, April 2008. `http://ziyang.eecs.northwestern.edu/~dickrp/tgff/manual.pdf`.

[152] E. Bini and G. C. Buttazzo, "Biasing effects in schedulability measures," in *Proceedings of the 16th Euromicro Conference on Real-Time Systems (ECRTS 04)*, pp. 196–203, 2004.

[153] T. P. Baker, "A comparison of global and partitioned EDF schedulability tests for multiprocessors," in *Proceedings of International Conference on Real-Time and Network Systems*, pp. 119–127, 2006.

[154] M. Bertogna and M. Cirinei, "Response-time analysis for globally scheduled symmetric multiprocessor platforms," in *RTSS '07: Proceedings of the 28th IEEE International Real-Time Systems Symposium*, (Washington, DC, USA), pp. 149–160, 2007.

[155] R. I. Davis, A. Zabos, and A. Burns, "Efficient exact schedulability tests for fixed priority real-time systems," *IEEE Transactions on Computers*, vol. 57, no. 9, pp. 1261–1276, 2008.

[156] A. Burns and G. Baxter, "Time bands in systems structure," in *Structure for Dependability: Computer-Based Systems from an Interdisciplinary Perspective*, pp. 74–88, Springer London, 2006.

[157] I. Bate, *Scheduling and Timing Analysis for Safety Critical Real-Time Systems*. PhD thesis, Department of Computer Science, University of York, 1999.

[158] P. Hall, "The distribution of means for samples of size n drawn from a population in which the variate takes values between 0 and 1, all such values being equally probable," *Biometrika*, vol. 19, no. 3/4, pp. 240–245, 1927.

[159] A. K. Gupta and S. Nadarajah, eds., *Handbook of Beta Distribution and Its Applications (Statistics: a Series of Textbooks and Monographs)*. Dekker, June 2004.

[160] R Development Core Team, *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2009. ISBN 3-900051-07-0.

[161] N. L. Johnson, S. Kotz, and N. Balakrishnan, *Continuous Univariate Distributions*, vol. 1. Wiley, second ed., 1994.

[162] D. H. Wolpert and W. G. Macready, "No free lunch theorems for optimization," *IEEE Transactions On Evolutionary Computation*, vol. 1, no. 1, pp. 67–82, 1997.

[163] S. Poulding, P. Emberson, I. Bate, and J. Clark, "An efficient experimental methodology for configuring search-based design algorithms," in *Proceedings of 10th IEEE High Assurance System Engineering Symposium (HASE 2007)*, pp. 53–62, 2007.

[164] P. Emberson and I. Bate, "Stressing search with scenarios for flexible solutions to real-time task allocation problems," *IEEE Transactions on Software Engineering*, To Appear. `http://dx.doi.org/10.1109/TSE.2009.58`.

[165] D. E. Knuth, *Art of Computer Programming, Volume 2: Seminumerical Algorithms*. Addison-Wesley Professional, third ed., November 1997.

[166] "Berkeley Open Infrastructure for Network Computing." `http://boinc.berkeley.edu/`. Accessed August, 2009.

[167] M. Matsumoto and T. Nishimura, "Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator," *ACM Trans. Model. Comput. Simul.*, vol. 8, pp. 3–30, January 1998.

[168] E. Ridge and D. Kudenko, "Screening the parameters affecting heuristic performance," in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '07)*, vol. 1, 2007.

[169] D. C. Montgomery, *Design and Analysis of Experiments*. Wiley, sixth ed., December 2004.

[170] E. T. Lee and J. W. Wang, *Statistical Methods for Survival Data Analysis*. Wiley Series in Probability and Statistics, Wiley, third ed., 2003.

[171] E. Ridge and D. Kudenko, "Analyzing heuristic performance with response surface models: prediction, optimization and robustness," in *Proceedings of the 9th annual conference on Genetic and evolutionary computation (GECCO '07)*, (New York, NY, USA), pp. 150–157, ACM, 2007.

[172] "LINDO Systems." `http://www.lindo.com/`. Accessed August, 2009.

[173] D. E. Smith and C. A. Mauro, "Factor screening in computer simulation," *SIMULATION*, vol. 38, pp. 49–54, February 1982.

[174] W. F. Smith, *Experimental Design for Formulation*. Statistics and Applied Probability, SIAM, 2005.

[175] D. R. Cox, "Regression models and life-tables," *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 34, no. 2, pp. 187–220, 1972.

[176] J. P. Klein and M. L. Moeschberger, *Survival Analysis: Techniques for Censored and Truncated Data*. Springer, second ed., March 2003.

[177] T. Therneau and original R port by Thomas Lumley, *survival: Survival analysis, including penalised likelihood.*, 2009. R package version 2.35-4.

[178] R. M. Korwar and R. J. Serfling, "On averaging over distinct units in sampling with replacement," *The Annals of Mathematical Statistics*, vol. 41, no. 6, pp. 2132–2134, 1970.

[179] L. Sha, R. Rajkumar, J. Lehoczky, and K. Ramamritham, "Mode change protocols for priority-driven preemptive scheduling," Tech. Rep. UM-CS-1989-060, University of Massachusetts, 1989.

[180] M. T. Schmitz, B. M. Al-Hashimi, and P. Eles, "Cosynthesis of energy-efficient multimode embedded systems with consideration of mode-execution probabilities," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 24, pp. 153–169, January 2005.

[181] D. Milojicic, F. Douglis, Y. Paindaveine, R. Wheeler, and S. Zhou, "Process migration," *ACM Computing Surveys*, vol. 32, no. 3, pp. 241–299, 2000.

[182] P. Emberson and I. Bate, "Minimising task migration and priority changes in mode transitions," in *RTAS '07: Proceedings of the 13th Real Time and Embedded Technology and Applications Symposium*, pp. 158–167, 2007.

[183] C. Spearman, "The proof and measurement of association between two things," *The American Journal of Psychology*, vol. 100, no. 3/4, pp. 441–471, 1987.

[184] W. Nace and P. Koopman, "A product family approach to graceful degradation," in *Proceedings of the IFIP WG10.3/WG10.4/WG10.5 International Workshop on Distributed and Parallel Embedded Systems*, 2000.

[185] Y. C. Yeh, "Safety critical avionics for the 777 primary flight controls system," in *Digital Avionics Systems, 2001. DASC. The 20th Conference*, vol. 1, 2001.

[186] C. Shelton, *Scalable Graceful Degradation For Distributed Embedded Systems.* PhD thesis, Carnegie Mellon University, June 2003.

[187] F. B. Schneider, "Implementing fault-tolerant services using the state machine approach: a tutorial," *ACM Comput. Surv.*, vol. 22, pp. 299–319, December 1990.

[188] P. Emberson and I. Bate, "Extending a task allocation algorithm for graceful degradation of real-time distributed embedded systems," in *Proceedings 29th Real-Time Systems Symposium (RTSS 08)*, pp. 270–279, November 2008.

[189] A. Girault, C. Lavarenne, M. Sighireanu, and Y. Sorel, "Generation of fault-tolerant static scheduling for real-time distributed embedded systems with

multi-point links," in *IEEE Workshop on Fault-Tolerant Parallel and Distributed Systems*, pp. 125–125, April 2001.

[190] Y. Oh and S. H. Son, "Scheduling real-time tasks for dependability," *The Journal of the Operational Research Society*, vol. 48, no. 6, pp. 629–639, 1997.

[191] X. Qin and H. Jiang, "A novel fault-tolerant scheduling algorithm for precedence constrained tasks in real-time heterogeneous systems," *Parallel Computing*, vol. 32, pp. 331–356, June 2006.

[192] K. Echtle and I. Eusgeld, "A genetic algorithm for fault-tolerant system design," in *Dependable Computing*, Lecture Notes In Computer Science, pp. 197–213, Springer, 2003.

[193] V. Izosimov, P. Pop, P. Eles, and Z. Peng, "Design optimization of time- and cost-constrained fault-tolerant distributed embedded systems," in *DATE '05: Proceedings of the conference on Design, Automation and Test in Europe*, pp. 864–869, 2005.

[194] A. Ejlali, B. M. Al Hashimi, and P. Eles, "A standby-sparing technique with low energy-overhead for fault-tolerant hard real-time systems," in *CODES+ISSS '09: Proceedings of the 7th IEEE/ACM international conference on Hardware/-software codesign and system synthesis*, pp. 193–202, 2009.

[195] J. Abrams and R. Cone, "Implementing expected monetary value analysis into risk metrics and assessment criteria," in *Proceedings of the 24th International System Safety Conference (ISSC)*, August 2006.