Enhancing real-time embedded systems development using artificial immune systems

Nicholas Christopher Lay

PhD

The University of York
Computer Science

October 2009

Abstract

The Consumer Electronics industry produces a large number of systems which exhibit both real-time and embedded properties (RTES). Frequently problems are encountered during the development of these systems due to a mismatch between the requirements of traditional real-time development techniques and the restrictions imposed on the development process by the retail market. There is therefore a requirement for a method which can be used to enhance the development process and improve the reliability of these systems, yet which does not require the intensive formal analysis often required by established real-time development techniques.

Observing that organisms in the natural world often exhibit characteristics which would be desirable in computer systems, such as the ability to adapt to changing environments and protect themselves against attack from external agents, this thesis examines the potential for techniques inspired by biological observations to be employed as a means to detect common problems encountered in the development of RTES.

The Dendritic Cell Algorithm (DCA), an algorithm derived from the observed operation of the innate immune system which has previously been applied to a number of anomaly-detection problems, is included in a simulated RTES to detect deadline overruns in a task scheduling system, and its effectiveness investigated over a variety of sample systems of varying sizes and complexities.

In conjunction with this, the issues surrounding the incorporation of an immune-inspired technique into embedded systems are examined in detail, with focus on the optimisation of the DCA for maximum effectiveness, whilst still providing flexibility. Attention is paid towards ensuring that the DCA is able to function satisfactorily in a

constrained environment, by establishing the level of system resources required and proposing further enhancements to the algorithm to enhance its suitability for constrained-resource implementation.

The work makes contributions to the real-time community by proposing an original method by which software development for real-time embedded systems can be enhanced, to enable the production of reliable software without the need for the formal analysis procedures normally needed.

There are also significant contributions made to the AIS community, by way of the introduction of an application domain in which the application of AIS techniques has not previously been proposed, and the enhancements made to the DCA to make it better suited to this domain than previously.

Table of Contents

List of Figures

List of Tables

Acknowledgements

The work which has made up the content of this thesis would not have been possible without the support of my supervisor, Dr. Iain Bate.

I would like to thank current and former members of the Real-Time Systems Group in the Department of Computer Science at the University of York for their assistance and friendship over the past four years, especially Dr. Neil Audsley, Prof. Alan Burns, Sue Helliwell, Prof. Andy Wellings, and of course Ian Gray. I would also like to thank all those in the informal "AIS group" at the University of York for their support, and particularly Prof. Jon Timmis for his help and encouragement.

I would also like to thank my family and friends for their help and encouragement over the course of my PhD work. Special thanks go to my wife Helen, without whose love and support I would never have made it to this point.

The work presented in this thesis was funded by the Engineering and Physical Sciences Research Council and Philips Research through a CASE award, reference CASE/CNA/04/78.

Declaration

The contents of this thesis have not previously been submitted for any degree, and is not submitted concurrently in candidature for any degree other than that of Doctor of Philosophy at the University of York. Unless otherwise stated all work presented in this thesis is the result of my own research and experimentation. Other sources are acknowledged by appropriate references.

Parts of the literature survey in chapters 1, 2 and 3 have been adapted from my Qualifying Dissertation text. In addition, some of the material in this thesis has been previously published. Early work on the DCA implementation in RTES has been published in one conference paper [100], and parts of chapter 5 have been published as a journal paper [101]. These publications were written by me with assistance from my supervisor.

1

Introduction

The adoption of computer systems has increased dramatically throughout the last few decades. In particular, in recent decades there has been a shift away from the concept of computers as separate devices towards the encapsulation of computer systems within a wide variety of devices. These systems are referred to as "embedded systems". The presence of embedded computer systems inside many electrical or electronic devices is now commonplace; indeed, new devices which do not contain some form of embedded microprocessor are significantly in the minority [17]. Classical computer systems are now significantly outnumbered by embedded systems, largely due to the large number of application domains in which embedded systems are now used, and the number is increasing steadily. Each year, billions of new embedded systems are produced, versus millions of "normal" computer systems. Estimates suggest that while a typical household will only contain two or three devices which would typically be referred to as "computers", there are also

likely to be in excess of 30 embedded computer systems of varying levels of complexity [27].

The rapid rise in embedded systems is due to their increased proliferation on a number of fronts. Firstly, embedded computer technology allows new classes of devices to be created which previously did not exist, such as the mobile telephone. In addition to this, embedded systems are also used extensively in established device classes, either replacing existing control logic [102], or supplementing it to allow additional functionality to be provided. Often, embedded systems exhibit several distinct advantages over traditional control systems. By utilising mass-produced devices such as microcontrollers, embedded systems can be cheaper and simpler to produce than the systems they replace. However, embedded systems often provide a greater level of control, or the scope to permit the inclusion of additional functionality than is possible with an analogue control system, sometimes with little further cost or development time.

Figure 1.1: Chips with everything - embedded systems are used in many different application areas

It is has been observed that the market for these embedded computer systems is increasing by approximately an order of magnitude over each ten-year period [66], [47] – both due to a larger number of systems being produced in existing application

areas, but also due to the spread of embedded systems technology into domains where currently traditional control systems are still used. However, it has been observed that the complexity of embedded systems is increasing: one observer in 1994 noted that the size of embedded software, when measured in terms of the number of lines of code produced, was doubling approximately every two years [65]. This trend appears to have continued throughout the 1990s [147].

A large number of embedded systems are intended to operate with little or no human interaction, for example as part of a control system, and therefore are not required to be interactive and consequently require little in the way of a user interface – perhaps only a few buttons and a simple numeric display. However, one area where interactive embedded systems are particularly prevalent is the Consumer Electronics (CE) business [17], where the trends for increasing complexity discussed above have been particularly apparent. As CE devices become increasingly more complex, so do the control systems which are contained within them, with many devices now containing multiple microprocessors and application-specific ICs, performing a wide variety of different tasks.

The most obvious example of a CE device is probably a television set, and as with virtually all CE devices, the television has been subject to considerable development over the last 20 years. Early models of television set used relatively simple discrete electronic components to receive and display a picture, which was broadcast via a modulated radio signal. However, over the last ten years, there has been a transition towards broadcasts using digital encoding methods as opposed to the traditional analogue modulation techniques.

Although digital broadcasting brings about a number of benefits, such as improved reception quality and more efficient use of radio spectrum allowing a larger number of services to be broadcast in the same space, it is more complex to implement. While the analogue signals only required relatively simple electronic components to allow them to be received and displayed, the reception of digital television requires a

system of much greater complexity, in order to accommodate the additional processing and decoding stages required.

Modern television sets therefore contain a number of microprocessors and a large amount of software, allowing them to receive and decode digitally-encoded broadcasts, perform processing on the decoded picture and sound – for example, upscaling the picture to fill the available screen area – and to allow access to interactive services, such as on-demand channels or information services. It was estimated that, in 1998, a state-of-the-art television set or similar CE device contained software consisting of somewhere in the region of 10^6 lines of code [145]. It has also been estimated that embedded software in CE devices increases in complexity approximately according to Moore's Law (i.e. doubling every two years) [148], therefore it can be estimated that a contemporary high-end television at the time of writing contains somewhere in the region of 32 – 64 X 10^6 – i.e. approaching 10^8 lines of code.

The development of more complex embedded devices such digital TV sets has led to a blurring of the boundaries between "normal" and embedded systems, and at the same time has brought about new challenges for the designers and developers of these systems.

1.1 Real-time embedded systems

In the field of systems engineering, systems are often considered to possess so-called "real-time" requirements. In these systems, the correctness of an operation depends not just on the result of the computation, but also on the time at which that result is produced [24]. These systems are broadly known as "Real-Time Systems" (RTS).

RTS are traditionally associated with safety-critical or high-integrity applications, where incorrect behaviour cannot be tolerated as it may result in catastrophic consequences. Such systems, where it is imperative that all real-time constraints are met and where the failure to meet those constraints may lead to severe consequences

in the system's environment, are referred to as being "hard" real-time. Typical examples of hard real-time systems include fly-by-wire avionics systems, and on a smaller scale, the drive-by-wire systems found in modern automobiles [24], [17].

Systems where real-time deadlines exist, but where the system is able to continue functioning where they are occasionally be missed or delivered late, are referred to as "soft" real-time. Typically, soft real-time systems are not safety-critical, and therefore the occasional failure to meet deadlines will not have a catastrophic effect in the system's environment, although its overall performance may be degraded [24]. There can also be considered to be a third class of systems in which deadline failures can be tolerated, but once the deadline has occurred there is no advantage in the result being delivered late. These systems are generally referred to as "firm" real-time [24].

A typical RTS will be a hybrid, containing some aspects which are "hard" real-time, for which all real-time requirements must be met in order for the system to function safely and/or correctly, and other aspects which are "firm" or "soft" real-time, where the occasional failure can be tolerated [24].

In addition to the "hard", "soft" and "firm" definitions, which are normally associated with "traditional" real-time control systems, there can also be considered to be a class of systems which may be known as "interactive" real-time. These are systems which are required to respond to user input as part of their operation. The operation of these systems is time-dependent: the users of interactive systems expect the system to respond in a timely fashion to their inputs, and so a user's perception of "system speed" can be considered as analogous to a process "deadline". Although not catastrophic by any means if these "deadlines" are missed, a perceived slowness or lack of responsiveness in an interactive system can make it particularly frustrating to use. These systems can therefore be considered to be similar to soft real-time systems.

Throughout this thesis, the term "real-time embedded system" (RTES) is used to denote a class of systems which exhibits both real-time and embedded properties. The class of systems which can be thought of as RTES is broad, due to there being a large degree of overlap between the two domains: a large number of RTS are to some degree embedded, and a large number of embedded systems exhibit real-time properties to a greater or lesser extent.

Increasingly, systems with definite real-time properties are being utilised in the consumer electronics domain. These systems can experience a variety of problems if their real-time requirements are not met, ranging from small issues, such a perception of unresponsiveness or "lag" having a negative effect on usability, through to more severe issues such as complete failure of the system.

This is particularly true for digital broadcasting platforms, which transmit digitally-encoded audio and video signals. For example, in a digital television receiver, the audio and video streams must be processed in a way that ensures a high quality output, with no dropped video frames or gaps in the audio. Furthermore the decoded video and audio signals must be synchronised when they arrive at their respective outputs [46]. The system has no control over the data stream it receives, and it has no way of predicting the contents before it arrives. Regardless of this, it must be able to process that data within a specific timeframe: for example, any processing which must be carried out on a video frame must be completed by the time that frame is to be displayed [46]. The system must also be able to deal with errors in the received data. Such a system may therefore be considered to be firm real-time: it is not safety-critical, but if the frame is delivered late or not at all then the overall operation of the system is impaired.

Digital video systems, therefore, have clear real-time requirements, which must be satisfied if the final implementation is to function correctly [46]. In addition to the video processing components, a large number of systems also have interactive components, and these need to respond appropriately to user input without a

significant delay. High end systems may also have the ability to run third-party software, either on removal media or installed onto the device itself. This places further requirements on the system which cannot be known at the time the system is designed. Such systems can be considered to be typical examples of RTES as used in the CE domain.

1.2 RTES development: current practice

The development of RTES makes use of a mix of methods, incorporating aspects of both traditional real-time systems engineering and embedded design. Frequently, the exact development procedure used is influenced by the nature of the system being developed. However, the need for the system to satisfy real-time constraints in addition to any time, cost or resource limitations often leads to conflicting requirements in the development process.

1.2.1 Embedded systems development

Traditionally, embedded systems were functionally much simpler than normal computer systems. Until recently the majority of embedded systems were function-specific: they always performed a single function or set of functions, and were never required to operate outside of that domain. Latterly, as the complexity of devices incorporating embedded software has increased, many embedded systems have become increasingly general-purpose, offering a range of functionality which can often be customised and extended by the end user.

Frequently, the developers of embedded systems are required to deal with a number of conflicting issues, which requires trade-offs to be made to optimise the system as far as possible [144]. Although not all issues necessarily apply to all systems, they may include:

- Unit cost: where many systems are being produced (i.e. tens or hundreds of thousands), the main factor on the cost of a system is the combined cost

of its constituent parts, referred to as the bill of materials (BoM). To maximise the profit potential of a system, it is therefore necessary to keep these costs to a minimum [146].

- Development cost: the cost of designing the system must be considered carefully, in the context of the number of systems being produced. Any costs incurred in developing the system must be recovered out of the sales of the finished product. If many systems are sold, the development cost is a smaller proportion of the total price than in situations where relatively few systems are sold [146]. It may also be possible to spread the overall development across a number of similar systems.

- Resource constraints: this is often the biggest issue facing the designers of embedded systems. The physical hardware may be physically constrained, maybe by size or its power usage (and the associated need to dissipate any heat created), or the available processing power or data storage may be restricted by the financial cost of the hardware. Designers of embedded systems need to ensure that they are able to implement the desired functionality within the physical and financial constraints of the project.

- The design of embedded systems is often multidisciplinary: designers must be able to work closely on the hardware and software components of the system, to exploit the available resources to the maximum extent possible. For example, knowing the features and limitations of the underlying hardware, such as the presence of hardware accelerators, can allow designers to use data structures and algorithms which run more efficiently on the target platform.

Attempting to meet these and other design requirements often leads to challenges and problems [48]. Designers are often placed in a position where they must attempt

to produce systems which conform with multiple sets of competing design objectives. This has led to the emergence of various techniques to assist in the development of embedded systems [52].

1.2.2 CE development factors

As well as being subject to the development trade-offs highlighted above, the development of CE devices is subject to further constraints dictated by the nature of the marketplace.

CE devices are generally intended to be mass-market products, which may sell hundreds of thousands or even millions of units: as a result, it is desirable to keep the manufacturing costs down, as a small saving on the cost of each unit combines to produce a considerable overall saving [144]. It is also important to consider the costs associated with developing the product, as these must also be recovered through sales of the final product. Although for CE devices, the development costs are amortised over a large number of devices, a device with significantly higher development costs than its competitors will be less profitable overall. There is of course a trade-off to be made between unit cost and development cost: for example, it may be possible to reduce the unit cost by replacing hardware components with software, but for this to be economical the cost of developing the software must be less than the amount saved on the additional hardware over the total number of devices produced.

A major factor in the development of CE devices is competition in the marketplace. CE manufacturers compete for sales in a market where there is a particularly high product turnover. In order to maximise potential sales, a product must enter the marketplace at the correct time: if it is late, it may lose sales to rival manufacturers.

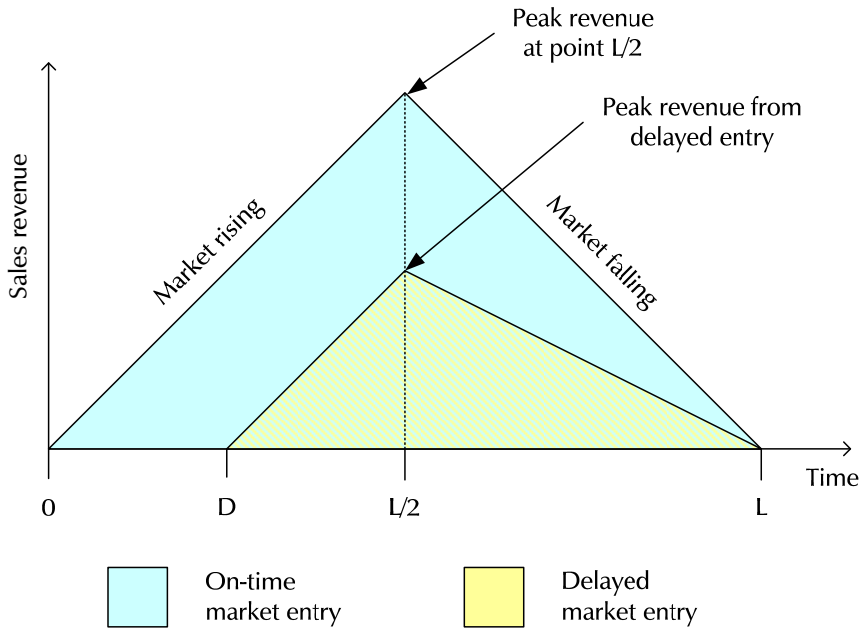


Figure 1.2: Simplified sales model showing effects of delayed market entry (from [144])

A simplified sales model [144] showing the effects of delayed market entry is shown in Figure 1.2. For the purposes of the model, the product is assumed to have a fixed lifespan ranging from 0 to L, with the point of peak sales at the point L/2. Between the start of the lifespan and the point of peak revenue, sales increase at a constant rate with respect to time. Between the point of peak revenue and the end of the product lifespan, sales decrease. The total sales are represented by the area under the graph, coloured blue for a product which enters the market at the beginning of its lifecycle. If a product is delayed entering the marketplace, its sales rise at the same rate from the point at which it is released. In a competitive marketplace where similar products are also available from other manufacturers, the market peak occurs at the same time as it would if the product was released on time: consequently, a delayed product has less time to attain sales before the market begins to decline. The size of the total sales where the product is delayed is shown in yellow, and can be seen to be significantly smaller than if the product was not delayed.

However, it is also important to ensure that a product is not rushed to market at the expense of development and testing. The implications of rushing a product to market before it is ready can include customer complaints, loss of consumer confidence and potential bad reviews in the press.

Flaws in a product which are not found until after it has gone on sale can lead to large costs being incurred, associated with, for example, the need to conduct a higher than anticipated number of warranty claims. If the system in question sells in significant numbers, these costs can quickly mount up: for example, the cost of warranty repairs following hardware problems with Microsoft's Xbox 360 games console is estimated to have reached $1.1 billion [1].

Even if the problems can be corrected, there is potential for huge losses in sales due to the product having a bad reputation. Worse is the potential for a bad product to do long-term damage to the reputation of an entire brand.

1.2.3 RTS development methods

The properties of real-time systems, and the various complexities with their development, have been the subject of active research over many years and consequently they are relatively well understood, with a wide range of specialist development and analysis tools available. In the safety-critical domain, software must frequently be analysed and verified to ensure that it is correct and that it meets its timing requirements [118]. However, these techniques are time-consuming and often require specialist knowledge: most methods are based around worst-case analysis of the system scheduling [24]. This approach is often pessimistic, leading to a lower than optimal resource utilisation. Also, there is no way to predict the occurrence of problems which are not governed by worst-case parameters [67], [105]. A particular problem with worst-case analysis methods is that it is extremely difficult to obtain the Worst-Case Execution Times (WCET), on which the scheduling and timing analysis relies.

There are two main approaches to derive WCETs of tasks: measurement-based and analysis-based. Analysis-based methods are problematic as they require in depth knowledge of the structure of the CPU on which the system will run. This is problematic even for the simplest of processors [50], and for complex modern processors employing advanced caching, pipelining and branch-prediction

technologies, is virtually impossible. To date there are only two tools that attempt to statically obtain the WCET of tasks (Bound-T [85] and AiT [134]). These tools are only able to produce results for a small percentage of the microprocessors available to the embedded systems market, and these results are still not exact due to the presence of caches, the operation of which cannot be predicted.

Measurement-based techniques are more straightforward: the behaviour of a system is monitored while running on its target architecture, and the WCET is taken to be the longest observed during the execution. The problem with measurement-based methods is therefore the proof of correctness. It is impossible to know, without monitoring a system for an infinite period of time, whether the worst-case time encountered is actually the maximum worst case which it is possible to obtain: if it is not, then any proof using that value as the actual WCET does not hold. A common approach with many measurement-based techniques is to use the longest measured time with the addition of a safety factor: this goes some way towards ensuring that all observed occurrences are within the worst-case time, although it cannot guarantee this [76].

Regardless of whether the technique is analysis or measurement based, there are some key problems with all formal analysis techniques. Firstly, the results obtained are likely to be pessimistic: in the case of analysis-based methods, this is generally due to the need to discount any advanced hardware technologies such as caching; for measurement-based techniques, it is the addition of safety factors which are the most significant cause of added pessimism.

In addition, the results obtained by formal analysis are specific to the exact system being analysed. Any changes to the system software, or to the underlying hardware, render the analysis invalid and therefore if a new version must be proven correct it must be completely re-analysed. The need to separately analyse each slightly different version of a system makes the formal analysis process time-consuming and

expensive, particularly where there is a high likelihood that changes to the software will be required.

Finally, as with any technique, the results obtained through formal analysis are only as correct as the underlying models and assumptions. Should these prove not to be correct, the correctness of the overall results can no longer be guaranteed.

1.3 Complexity and Emergence

With any system involving a large number of interactions between a number of individual elements, we start to encounter problems associated with the complexity of these operations. Frequently, the combined operations of a number of apparently simple components leads to the overall system operating in a way which is far more sophisticated than would normally be expected simply by analysing the individual components [20].

In general, the behaviour of the individual elements making up a system can be analysed and understood. It is only when combined in large numbers that the system as a whole begins to exhibit properties which cannot easily be related to the functionality of its elements. The sophistication which is displayed by these so-called "complex systems" is known as "emergent behaviour", and it is visible in a wide variety of systems, both artificial and natural [20]. A large number of natural complex systems have been observed to display adaptive properties, allowing them to react to changes in their circumstances particularly effectively, although they do not possess any particular locus of control.

There are many examples of complex and emergent behaviour in the natural world [68]. The flocking behaviour of birds, or the swarming behaviour of insects, can be considered emergent. Each bird or insect follows a specific set of rules to establish how it should move: the combination of many individuals (known in the field of complex systems as "agents") all following the same rules independently leads to the overall flocking behaviour.

Emergent behaviour shows patterns similar to those which occur as a result of a similar phenomenon called stigmergy. An example of stigmergy in the natural world is the use of pheromones by ants as a part of foraging behaviour [15]. An ant returning to its colony with food lays a chemical trail which other ants then follow, when they return they reinforce the trail by laying additional chemicals along it. The result is that more ants follow the trail, which is then even further reinforced. Repeated reinforcement eventually results in a majority of ants following the trail.

Although emergence and stigmergy result in similar effects, stigmergy achieves this through the reinforcement of "good" behaviour by signals which allow agents to communicate with each other. Pure emergence has no such stimulation mechanism: the emergent behaviour is a result of the interactions between agents acting independently of one another.

It is often difficult to define exactly what is meant by emergent behaviour or emergent properties and there are a large number of similar – but subtly different – informal definitions, although effort has been made to try and produce a formal definition [99]. Generally, an emergent property is considered to be one which cannot be easily predicted based solely on examination or modelling of the system, or of an agent in isolation: consequently, emergence can be closely related to "surprisingness" or "unexpectedness".

1.3.1 Emergence in systems development

Emergent properties can be important during the development of computer systems, as they may result in undesirable side-effects or problems with the operation of the system [13]. These emergent properties are normally connected with the non-functional characteristics of the system, for example, processes in a real-time system may miss their deadlines.

In the context of computer systems development, emergent properties are considered as those which are difficult to predict by abstract modelling of a system [113]. These

properties often arise as a consequence of integration: that is, the mapping of a system design onto its target implementation platform. The existence of these previously unforeseen properties may result in the system not functioning correctly, and in turn this may lead to delays while the problems are fixed, with the potential for associated financial penalties [120].

In particular, it has been noted that the effects of emergent properties in a system, and the need to manage them, increase as the complexity of the system increases [20]. It has also been observed that, in many ICT and computing systems, the effects of emergence are already being felt, and that the development of techniques allowing the understanding and management of emergence must be considered as a matter of urgency [113].

There are various different categories of emergent properties in computer systems: some appear during the development of the system, while others are a consequence of faults or failures while the system is in use [62]. The reasons for the manifestation of emergent properties are unclear, however it is likely that some are due to mismatches between models used during development, and the physical implementation of the system.

It is important to make a distinction between the management of emergent properties and their elimination from a system. While the obvious conclusion regarding emergence is simply that emergent properties are bad, this is not always the most appropriate response: indeed, there have been cases where emergent properties have in fact been beneficial to the system as a whole. Consequently, an effective strategy for dealing with emergence should be able to exploit emergence where it is helpful, and to eliminate it where it causes problems. This concept is known as “controlled emergence” [113], although as yet its feasibility is unknown.

1.4 Challenges in RTES development

The development of RTES in the CE domain is subject to a number of conflicts, which must be resolved in order to produce systems which meet all their requirements. With a large number of CE devices, it is important that real-time constraints are satisfied, in order for the product to operate correctly. Traditional real-time techniques could of course be used in order to guarantee that the system meets its constraints. Indeed, in other application domains, such as the use of RTES in the automotive industry, there are safety-critical requirements and RTES are therefore developed using traditional real-time systems techniques, despite the time-consuming and expensive nature of the process, and of course these techniques are only as accurate as their underlying models and assumptions.

In the case of the automotive industry, safety-critical components are sourced from a few specialised suppliers: these suppliers are then able to spread the development costs over a larger number of units by supplying the same system to multiple manufacturers; in addition safety-critical components are isolated from non-safety-critical ones, ensuring that the failure or malfunctioning of non-essential systems does not affect safety [17].

The CE development process, however, is particularly driven by the prevailing market situation, and in the vast majority of cases there are no safety-critical elements associated with the development of a CE device. It is not therefore possible for CE manufacturers to justify the use of traditional real-time development techniques from a commercial perspective: in a marketplace which is primarily driven by price, there is not scope for the additional costs associated with real-time development to be recouped, even if the finished product is guaranteed to be 100% reliable.

1.4.1 RTES development problems

Problems and anomalies can be encountered during the development and operation of RTES, often resulting in failure of that system. Although there are a large number of potential problems which can arise, the overall effect from the user's perspective is partial or even total non-responsiveness of the system.

Often, these failures are associated with the real-time nature of the systems in question. Typical problems encountered during the development of RTES may include:

- Bandwidth bottleneck problems: these can occur when a number of different components within a system are required to transfer data to other components via a shared bus. Although the bus may appear to have a high enough bandwidth to allow all the require data to be transferred, problems can arise if multiple components attempt to transfer data on the shared bus simultaneously, resulting in a situation where there is insufficient bandwidth available for all the desired transfers to complete.

- Deadline overrun problems: these can occur in any system containing real-time tasks with a deadline by which they must be completed. A deadline overrun occurs when for some reason a task fails to complete before its specified deadline, perhaps because it is being blocked from executing by another task, or if it is waiting for external devices to provide it with data.

- Deadlock problems: deadlock occurs when two or more different components of a system each require access to shared resources currently held by another component. Generally, these resources can only be accessed by a single task at once, perhaps because access by multiple components simultaneously would lead to errors, and so are protected by locking mechanisms to guarantee mutual exclusivity. At least two tasks

and at least two external resources must be present in a system before a deadlock situation can arise, a typical situation being where Task A holds a lock on Resource 1, and Task B holds a lock on Resource 2. If Task A requires access to Resource 2 in order to complete, while simultaneously Task B requires access to Resource 1, then a deadlock situation will occur as neither task can obtain access to all the resources it needs to be able to complete successfully.

These problems are frequently encountered during the development of real-time systems and as such they are well understood from a traditional real-time perspective. With all of these examples, a key commonality is contention over shared resources: either CPU time, bus bandwidth or the use of external devices, and within the real-time systems community there exist a number of techniques for ensuring systems are not affected.

The fact that these problems are encountered during the development of RTES in the CE industry can be attributed, at least in part, to the constraints within which the development of these systems is conducted. It is apparent that techniques and methods exist within the RTS domain which go some way towards allowing many of these problems to be solved; however the circumstances in which CE development occurs preclude their use. There is therefore a need for methods and techniques which help to ensure that RTES are reliable and robust, but which is also compatible with the rapid product development cycles which are characteristic of the CE industry.

This following chapters of this thesis investigate the problem area and examine a number of potential novel solutions. Chapter 2 examines the development of RTES and the methods and tools which are currently used in industry, and establishes a number of problems with these methods which cause the reliability of RTES to be reduced. Chapter 3 examines the desired properties of RTES and draws comparisons with the properties desired by RTES developers, and those seen to be possessed by

many biological systems, and investigates the potential use of techniques inspired from aspects of biology to improve the reliability of RTES. Chapter 4 formulates a hypothesis and a number of research questions which must be answered in order to derive a successful solution which can be applied to RTES development problems. Chapters 5 - 7 detail the application of a biologically-inspired method to a typical example of a real-time anomaly, considering how the effectiveness is affected by the characteristics of the problem systems as well as of the techniques applied, and also the implications of the deployment of methods inspired by biology in the constrained resource environment of RTES. Finally, chapter 8 draws conclusions and suggests areas where further investigation could be conducted.

2

Survey of RTES development methods and issues

In order to determine how new techniques can be applied as a part of the RTES development cycle, it is first necessary to examine the methods currently used in the design of these systems. By formulating an understanding of the deficiencies with existing techniques, new methods can be developed which go some way to solving the problems encountered during the development and operation of RTES in CE devices. Therefore this chapter examines the state of the art in systems development in general, and focussing on CE development in particular. This knowledge can then be used to guide the formulation of new methods.

2.1 Component-based software engineering

A large number of software systems developed today make use of a development strategy widely know as Component-based Software Engineering (CBSE). CBSE

methods confer a number of advantages to software developers, most significantly allowing a software system to be assembled from a selection of pre-existing components. This allows software to be developed in a shorter time. Efforts have also been made to allow individual components to be pre-tested and certified: the use of certified components in a system contributing to increased reliability in the system as a whole.

2.1.1 Development of CBSE

CBSE is a refinement of software engineering concepts over many years. Its origins lie in work carried out by Dijkstra, and others, who first observed the need to apply structure to the creation of computer programs in the late 1960s [42].

Dijkstra originally outlined his ideas in two documents ([44] and [45]) which were circulated amongst the scientific community informally, and one of which was included in the proceedings of a NATO Science Committee conference in 1969 [43]. He was concerned with the development of computer programs which were large, due to the level of complexity in the problem they were solving. He had noticed that programmers were starting to reach the limits of their ability in terms of the size of the programs they were creating. He hoped that his structured programming techniques would enable programmers to increase their programming ability by "an order of magnitude". He advocated a number of (then novel) principles, all of which were intended to make a program more understandable, such as the use of subprocedures and other high-level control structures rather than jumps or goto statements, and the use of abstraction allowing layered implementation.

It is clear from reading Dijkstra's notes that he never envisaged software systems as large and complex as those which exist today. As the complexity of systems has increased further, refinements of his techniques have been employed to allow additional levels of decomposition and abstraction in order to simplify the development process and increase understanding. In particular, the concepts of modularisation and information hiding were brought about by the realisation that

overall system complexity could be reduced by dividing a system's functionality according to data flow and storage rather than by its functional steps as seen in a flowchart [117]. Modularisation allows all low-level aspects of one particular process (for example disk I/O) to be located in one module rather than spread throughout the system, making development and maintenance simpler, and allowing the reuse of code across different systems.

Although code reuse has been practised since the beginnings of software development, thereby avoiding unnecessary re-invention of technology [14], initially this was through low-level "code scavenging" [98]. The practice of modularisation allowed the reuse of larger code blocks, each with well-defined inputs, outputs and functionality [111], and resulted in an increased level of abstraction, allowing the design of systems to occur largely at the architectural level rather than directly at the source code level.

Most modern high-level languages support the concepts of reuse through "source code components" [98], for example, Ada/Java packages, or C's `#include` system, and the use of these systems is well documented and widespread. The growth of the Internet has helped to create large-scale repositories of packages which can be easily searched to locate components with appropriate functionality.

2.1.2 Architectural style

The use of software-architecture level development has not been without problems. In particular, problems have been encountered when there is a mismatch between the "architectural styles" of different components. "Architectural style" refers to the overall top-level organisation of a system, and will often represent the flow of data around it. There are number of different architectural styles, each of which can be used to create workable systems, providing that suitable components are available which match both functionally and stylistically [64].

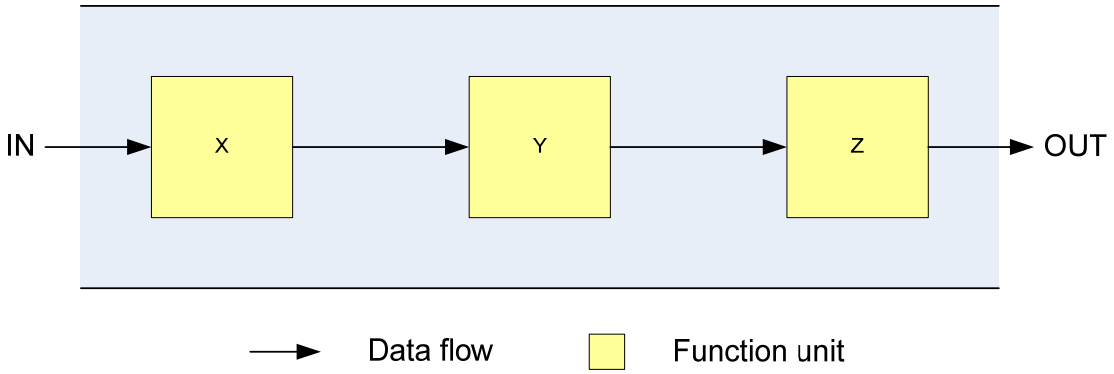


Figure 2.1: Pipelined system architecture

All architectural styles share a number of common features. They all feature identifiable components, which can be used to compose systems. There is a mechanism to allow components to communicate with each other, i.e. connectors, and the overall execution of the system is governed by a control structure and a system model [124]. Two possible architectures for a simple system are shown in Figure 2.1 (pipelined architecture) and Figure 2.2 (object-oriented architecture).

Importantly, components created for use in one particular architectural style may not be appropriate for use in another architectural style. For example, a component intended for use in a pipelined architecture will be packaged as a filter, with distinct input and output interfaces, while one intended for use in an object-oriented style will possess a different type of interface, able to support parameterised procedure calls. Although the two components may support exactly the same functionality, they cannot be used interchangeably: the choice of component depends on the underlying architecture of the system as well as on the component's functionality.

Often, the choice of architectural style will be imposed, either by the development environment used, or else simply by convention. For example, many programming languages support object-oriented (OO) concepts, either as a fundamental part of their design – such as Java, which was object-oriented from its inception [5] – or have been extended to provide OO support, as with Ada95/Ada2005, which support OO concepts through the addition of tagged types [12]. The concepts of OO and the principles it promotes, including the concepts of modularisation and information

hiding, are well understood and appreciated: consequently, object-oriented design is probably the most frequently used of the architectural styles identified in [64].

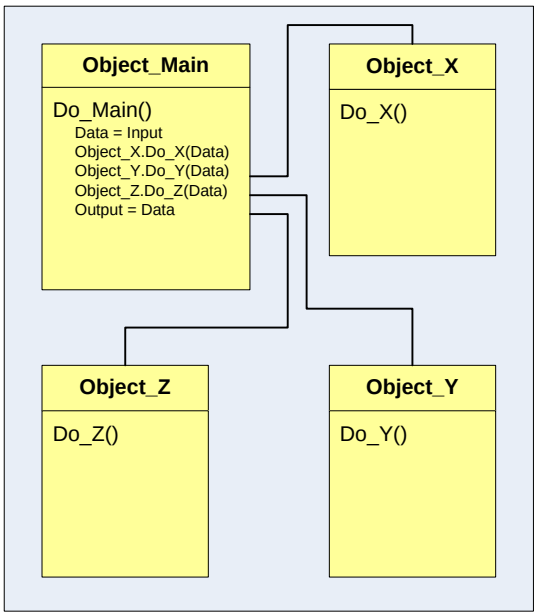


Figure 2.2: Object-oriented system architecture

2.2 CBSE in embedded systems: Koala

CBSE is employed in the majority of software engineering disciplines, including the embedded systems domain [146]. As discussed in chapter 1, the development of embedded systems gives rise to a number of issues in addition to those encountered during the development of a more conventional system. However, the majority of CBSE development methods are not specifically designed to engineer embedded software. In particular, embedded software must be able to run with limited resources; however the use of component-based development systems often results in additional overheads, associated with the interfacing between different components. These resource overheads must be kept to a minimum if the system is to work effectively.

A toolset which has been developed specifically for the engineering of embedded systems is Koala. This was created by Philips Research in the late 1990s to assist with the creation of embedded software in CE devices (specifically in television sets) [145],

[149]. Koala was developed to address several issues which were being encountered during the development of embedded software.

As such, Koala represents the state of the art in embedded software development for CE, and therefore it is evaluated here in order to provide an insight into the current methods and tools used to develop software for CE devices.

2.2.1 Aims of Koala

Most significantly, Koala was intended to allow the creation of more complex software. It has been observed that the size of embedded software in consumer electronics devices increases in complexity according to Moore's Law, roughly doubling in size every two years [147]. Koala was intended to support this increase in complexity for a number of years into the future. In conjunction with this, Koala was intended to simultaneously reduce the development time of embedded software in order that the total time to bring a product to market could be reduced, giving a competitive advantage in the marketplace [146].

A key aspect of Koala is the ability to manage diversity amongst a range of similar products. The creators of Koala observed that within the CE device range, devices can be classified into a relatively small number of so-called "product families": each member of a product family offers similar overall functionality, but will differ from the other members of that family in terms of the exact features it offers, for example due to differences in regional markets or standards [146]. These product families may be able to share a common software base, incorporating different additional components depending on the desired overall feature set, and Koala was designed with support for the creation of such software in mind.

2.2.2 Koala architectures

Koala is based around the concepts of component-based development. Functional units are packaged into blocks referred to as "components", an example of which is

shown in Figure 2.3. These components are intended to be heavily reusable, and are designed to communicate with the outside world (and so with other components) solely by means of strictly defined interfaces. A Koala component can offer functionality to other components by "providing" one (or more) interfaces. If a component needs access to functionality provided by another component, it does this by "requiring" a particular interface. The actual implementation of a component's functionality is therefore entirely hidden from the system architect.

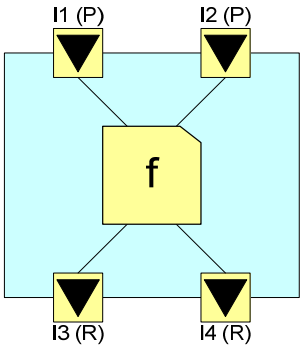


Figure 2.3: Example Koala component

This notion of "requiring" and "providing" interfaces removes any knowledge of specific components from the inter-component bindings [149]. The development tool is able to import the most appropriate component which "provides" a particular interface. For a system to be valid, all of its components' "requires" interfaces must be bound to a corresponding "provides" interface (although components may "provide" functionality which is not "required" by any other component).

Considerable emphasis in the Koala system is put on the importance of the interfaces between components: it is these interfaces which define the structure of a system, and the data flow through it. All interfaces are explicitly defined according to the functions they provide, using an interface definition language. Once defined, an interface cannot be changed under any circumstances [149], in order to preserve the integrity of systems built using that interface.

There are a number of methods by which Koala interfaces can be bound. Some of these are shown in Figure 2.4. The simplest of all bindings is the direct binding, shown in Figure 2.4a, where two interfaces are directly connected.

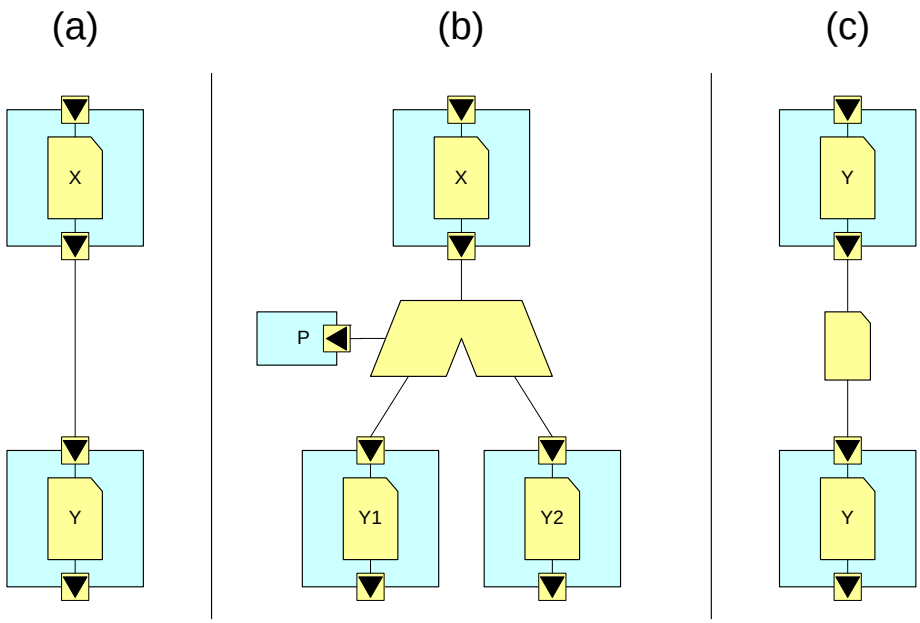


Figure 2.4: Koala interface bindings

More complex bindings are available: Figure 2.4b shows a switched binding, allowing the introduction of diversity into a system model. In this instance, the component X is bound to either component Y1 or Y2, depending on the configuration of the diversity component P. Such an arrangement allows the same system model to support, for example, different regional standards, which can be simply selected by altering a single component. The use of these diversity mechanisms is a means of satisfying Koala's aim of supporting product families: the same overall software structure can be applied in a number of similar systems, with the exact functionality controlled by the use of a number of switches.

Koala interfaces can also be bound through the use of "glue modules" – these are intended for joining subtly different interfaces without the need to create a separate glue component. This is shown in Figure 2.4c.

Koala makes use of explicit architectures, allowing the developer to specify exactly the underlying structure of their system. This method is used because not all applications can be made to fit into a particular architecture or set of architectures. Using an explicit architecture also makes it simpler to visualise the relationships between components, and helps ensure that the final generated code matches the underlying architectural design [149]. A simple Koala architecture is shown in Figure 2.5.

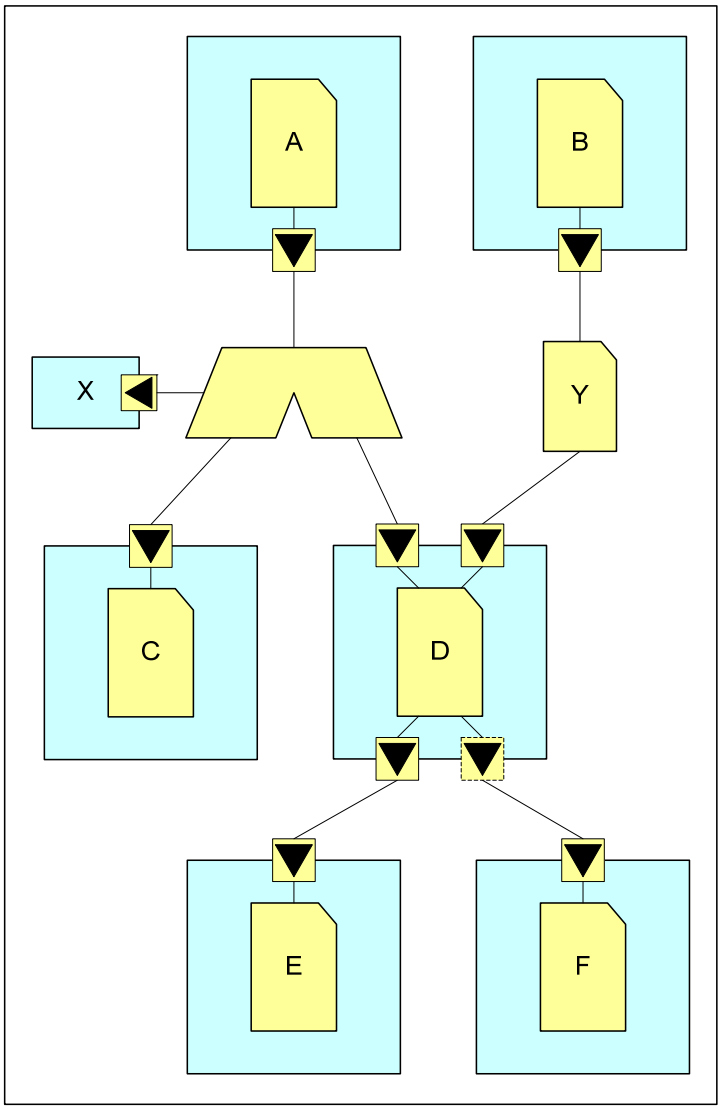


Figure 2.5: Simple Koala architecture

Architectures in Koala are described using an architectural description language (ADL). An important distinction between Koala and other industry standard tools such as UML (unified modelling language) is that when developing using Koala, the

implementation is derived directly from the component design, whereas UML is primarily a design language: the implementation must be carried out in a different toolset [141]. Although UML can be a powerful tool, the separation between the design and implementation of a system often results in a system's design being reverse-engineered from the implementation, rather than being carefully conceived as an integral part of the development process.

Interestingly for a system developed for use in embedded systems engineering, Koala's interfaces only support functional properties: there is no direct support for the specification of non-functional properties in the Koala model.

Koala builds in some important principles to make it particularly suited to the development of embedded systems. The majority of connections between components are bound statically during the system's configuration: this reduces the overheads at run-time. The exception to this is components which are intended to be upgradeable – these use dynamic bindings allowing them to be easily replaced without upgrading the entirety of the embedded software.

The final major aspect of the Koala system are two central repositories: one containing interfaces; the other, components. The repositories are maintained in accordance with some simple but important rules, to ensure that compatibility between components and interfaces in existing systems is not broken. For example: an interface, once added to the repository, cannot be changed. If an update to an existing interface is required, the update must be added to the repository as a new interface in its own right, so maintaining the integrity of existing components which make use of that interface.

There are also restrictions on the alteration of components, in order to preserve the integrity of systems which already make use of those components:

- A “provides” interface cannot be deleted, although new ones can be added. This restriction ensures that any systems which make use of the functionality provided through that component’s interface are not broken by the removal of the interface.

- A “requires” interface can be added to an existing component, but it must be made optional, such that any systems already using that component are not left with unresolved requirements when built with the replacement component.

- A “requires” interface cannot be deleted, but must be made optional [149]. This ensures that systems where the interface is already bound to a corresponding “provides” interface do not have to be altered to remove that binding.

These restrictions can all be considered analogous to the pre- and post-conditions employed in formal systems specification: to maintain the integrity of such a system, pre- and post-conditions can be weakened, but not strengthened. The exception to this is the third restriction listed above (forbidding the deletion of “requires” interfaces). This restriction is imposed solely for practical reasons, to eliminate the need to redesign existing systems (or subsystems) in which the interface in question has a binding.

2.2.3 Koala and RTES

Although it has been noted that the use of organised reuse strategies in embedded software development is not yet commonplace [66], Koala has been used to develop embedded software in a number of different classes of CE device. The effectiveness of Koala in real-world usage is difficult to judge, as it is a proprietary system and there is little information about its usage. The majority of, if not all, papers concerning Koala have been written by its developers, and are decidedly non-critical of the system.

Koala's main focus, like many CBSE techniques, is very much on providing mechanisms by which a system's functional complexity can be increased. When considering the development of software for CE devices with real-time requirements, it is necessary for the development methods and techniques used to take these requirements into account.

However, Koala is notable for its lack of support for non-functional or real-time properties. This can be partly attributed to the reasoning behind its development, which was to allow the creation of more complex software in a shorter time period: consequently the need to evaluate non-functional properties was not considered necessary.

Consequently, therefore, Koala is currently of limited use when designing a system where non-functional properties are an important consideration. However, this does not necessarily mean that Koala cannot provide a suitable solution for the non-functional emergence problem, as there may be ways in which it can be extended to cover these properties.

2.3 Component-based development and real-time properties

The development of CBSE techniques has been largely brought about by the need to support more complex functionality during the software creation process. Because of this, the emphasis in the techniques and the tools which support them is primarily on functionality, and the understanding of it. While this allows the creation of functionally larger and more complex systems, it does not help when considering those aspects of the system which are not related to its functionality, such as the constraints associated with real-time operation. This makes many software development techniques not well suited for the development of embedded software, where consideration of non-functional properties is important [66].

The effects of component integration in complex systems often give rise to emergent properties [120] which can have significant effects on the operation of an embedded

system. These emergent properties are normally related to non-functional characteristics which manifest themselves when the system runs. They may include execution times and power usage at the simplest level, through to higher-level system-specific properties, such as, in a video processing system, the number of dropped frames. Some effects may cause multiple emergent properties to occur simultaneously: a timing error in a video processor may cause both instability in the control loop, and also result in image errors.

While temporal and power usage requirements can be measured quantitatively, as can some system-specific properties which can be represented numerically, some properties, such as (in the case of a video processing system) visual properties – compression artefacts, quality, etc – cannot be measured directly and are instead more subjective in the way they manifest themselves. Although not easily measurable as an output of the system they can clearly be associated with its design and functionality, and in many cases it can be seen that changing the design or structure of the system, even subtly, can significantly affect its non-functional properties.

It is generally difficult to test the effects of non-functional properties. The functionality of software can easily be tested, either by simulation or by execution of the binary code. This can be done using an arbitrary platform, or in the case of simulation, using a platform-independent model. However, it is much more difficult to analyse non-functional properties, the effects of which can only be observed when executing the complete system on its target hardware, in order to ascertain the combined effects of any interactions between hardware and software. Even minor changes to the underlying platform, or to the system's composition, may result in significant changes in the system's non-functional behaviour, making the results of such analysis highly platform-specific.

This dependence on platform and the particular combination of software components makes the prediction of emergent properties particularly difficult: it is

often as a result of the interactions between the combination of hardware and software components that emergent behaviour occurs.

2.3.1 Evaluation of non-functional properties

The ability to predict the non-functional properties of a component-based system is currently one of the most active research areas in software development [55]. Although there are a number of research groups looking to develop tools and methods which allow the reliable and accurate prediction of non-functional properties, as yet, none are sufficiently mature as to have made it into regular usage.

Researchers have looked into various approaches for the evaluation of non-functional properties. These can be broadly categorised into two main approaches: techniques involving analysis of the structure of the system, combining this with known information about the individual components; and techniques involving simulation or measurement.

In general, the aim of research in this area is to be able to predict the interactions of a component assembly's non-functional properties at the design stage. Although there are a wide variety of possible non-functional properties which could be investigated, the most widely researched non-functional property has been execution time and overall system performance.

Where time and performance are the properties examined, the techniques are broadly known as "performance analysis". Work on performance analysis techniques is more advanced than techniques allowing the prediction of other properties, but in general performance analysis is still immature. The current state of the domain is [10]:

> *"Although some [work has] been successfully applied, we are still far from seeing performance analysis integrated into ordinary software development"*

Balsamo et al note that in order to achieve the desired goal of *a priori* evaluation of a system's performance, techniques involving measurement or simulation (for example as outlined in [16]) are not appropriate, because they require the presence of the actual system implementation or at least a detailed working knowledge of it [10]. It is therefore necessary to use analytical techniques to attempt to calculate the system's properties.

Broadly speaking, the various different analysis techniques follow the same basic principles, and all require at least the following:

- Knowledge of performance characteristics for each of the components present in a system. This may be derived from measurements, taken from the operation of components in isolation or as part of other systems, or calculated theoretically.

- A model of the operation and structure of the system, from which it can be calculated how the performance characteristics should be combined.

A similarity can therefore be drawn between "performance analysis" in the software engineering domain, and the WCET used in the real-time systems domain as outlined in section 1.2.3. Both techniques involve the same principles and are applied in similar ways, although the application of performance analysis techniques is normally at a higher abstraction level than most applications of WCET analysis.

There are several significant difficulties with performing *a priori* evaluation.

- It is necessary to obtain accurate performance data for individual components, without interference from other components or background processes, which is still accurate when combined with similar data from other components.

- The system model must be sufficiently accurate to allow the data to be combined, but not so complicated as to make the calculations computationally infeasible.

- Many systems are dynamic or adaptable, so their behaviour changes depending on their circumstances, many of which will be affected by the state of the system's inputs and the environment in which the system runs. These systems cannot be evaluated effectively at the design stage, as it is not possible to accurately model every aspect of the environment in which they will run.

There are many examples of performance evaluation work, such as [121], [83] and [32]. The majority of the work focuses on automated extraction of the system timing/performance model from the system's design, so allowing the automatic determination of a system's performance. Work has been carried out towards trying to establish the performance of parameterised systems [103] although the techniques developed are still relatively immature.

There appears to have been little focus on accurately determining individual component properties: most techniques rely on inaccurate values derived either from measurements, which are likely to be optimistic; or component analysis, which are likely to be pessimistic. The inaccuracy of these values for component properties obviously has a direct bearing on the overall accuracy of the analysis procedure, and therefore on the number of errors which are successfully detected.

There are fewer examples of analysis techniques dealing with other non-functional properties. A notable example of work which considers non-functional properties other than performance is [55], which considers the static memory requirements of a component-based system. This example is particularly interesting in the context of this thesis because it utilises the Koala component model as its underlying component architecture, and therefore is useful when considering the development of embedded systems.

The primary assumption behind these analysis techniques is that the non-functional properties of different components can be combined predictably, i.e. that it is possible to predict the performance of a given system, given the performance exhibited by the components in isolation, and the way in which the components are combined in the system. In practice, the way in which components interact with each other may not be straightforward, due to interference from other components, or the need to communicate with hardware devices. This can lead to effects which cannot be easily predicted – consequently reducing the effectiveness of any such evaluation scheme. There are a number of different real-time concepts which behave in this way, such as certain problems associated with task scheduling: these will be examined in more detail in a later section.

It is difficult to evaluate how well these techniques scale when dealing with large component-based systems. Frequently, papers in this area will outline their technique in terms of a case study and evaluate the results purely in terms of this, for example [121]. Although sometimes based around real-world problems, often the case studies involve small "toy" systems which are orders of magnitude simpler than a typical system. The overall complexity of a system increases significantly as the number of components and connections increases, and so it is possible that a technique which performed satisfactorily when evaluating a simple case study system may not cope with the increased complexity of a full-size system.

In addition, it can often be difficult to verify the accuracy of these techniques: indeed, at least one example [83] suggests that the provision of empirical evidence is adequate to demonstrate that their method is correct.

2.4 Hardware consideration

A significant difference between the development of embedded systems and the development of regular computer systems is the need to carefully consider the hardware requirements of an embedded system. In particular, the hardware available may be constrained by non-functional requirements such as power or space, or of course by the need to keep its monetary value as low as possible. Designers must therefore make trade-offs to achieve the best balance of all possible factors [144].

This is an issue when considering the suitability of performance analysis techniques in embedded software development. Virtually all the analysis techniques covered in the literature rely on the availability of accurate execution time models for each software component. However, these models are fundamentally dependent on the hardware platform on which the system runs, and the behaviour of software components is likely to change if the underlying hardware is altered. These changes in behaviour may be significant even if the underlying changes are subtle. For example, it is commonly assumed that if a software component functions on one platform, it will continue to function on a faster version of that same platform; however this is not always the case. In particular, problems may be encountered when the timing characteristics of a component are closely tied to the speed of the processor on which it runs.

There is little consideration of hardware issues in the literature. Only a few papers in the performance analysis domain acknowledge that the hardware platform is an issue, and even fewer propose any kind of solution to the problem of changing hardware platforms – the majority simply state that the underlying hardware is fixed. One technique which is aimed at real-time embedded systems and does consider hardware properties is that proposed in [121], however unfortunately this

does so at the expense of "fixing" the system software! As far as can be established from the published literature in the domain, there appears to have been little work carried out regarding the performance analysis of component based systems where there is variable hardware/software mapping.

2.5 Summary: CBSE in real-time/embedded systems

CBSE is widely used as a software engineering technique, including in the development of embedded systems, and there exist a number of methods and toolkits to aid the development of such systems. However, the primary driving force behind the development of CBSE methods is to increase the understanding of a software system's functionality, therefore allowing corresponding increases in correctness or complexity to follow. This has left non-functional system properties largely overlooked, with the result often being that, although a system's functionality may be correct it may not satisfy non-functional requirements.

Methods of evaluating the non-functional properties of a component-based system are now being investigated (e.g. [16, 121]), but there are as yet no mature technologies suitable for use in the development of real-time embedded systems. In particular, most proposed techniques have not yet been shown to be scalable for dealing with more complex systems, and they also make the assumption that the non-functional properties of individual components combine predictably, which particularly with complex systems involving a large number of component interactions, is not necessarily the case. Although there has been some progress made towards reliable and efficient performance analysis of component-based systems at the design stage, this has not yet seen widespread adoption, and does not incorporate many features which would make it suitable for the development of embedded systems.

3

Biologically-inspired computation methods

Biologically-inspired computation, i.e. the derivation of computational processes from observations of biological processes in nature, is a significant field of computer science. From observations of a wide variety of behaviours and characteristics which occur in nature, it is possible to derive solutions which can then be applied to problems in electronic systems.

The principle of observing a natural process in order to derive computational models is well established. However, the idea of applying biological principles to computational processes is still considered to be relatively unconventional, although the popularity of this idea is increasing [129]. Some of the more established bio-inspired techniques, such as genetic algorithms, are now accepted problem solving methods, particularly with respect to search or optimisation problems.

Bio-*inspired* systems, are not the same as biological computation, with which they are sometimes confused. Bio-inspired systems make use of ideas and concepts from biological processes in the natural world, but are implemented *in silico* (i.e. on standard silicon-based computing devices) [122]. On the other hand, biological computation is where the actual computation is performed by a biological substance, such as neurons or DNA.

This chapter examines the properties possessed by living organisms, and examines some of the systems and structures which afford these organisms the characteristics of robustness and adaptability, and how inspiration can be taken from these biological systems to help produce computing systems which are equipped with similar properties.

3.1 Exploitation of biological systems in silico

Bio-inspired computation is now relatively well-established. The first work involving ideas inspired from biology was artificial evolution, the origins of which can be traced back to pioneering work in the late 1950s and early 1960s, such as [19], [61], and [18]. Although the potential of evolutionary techniques was realised early on, their practical applications were significantly limited by a lack of available computing power [9].

As computing power has increased and evolutionary-inspired approaches have been shown to be effective [9], attention has turned to further areas of biology for inspiration of new methods. The field of bio-inspired computation (sometimes referred to as natural computation) now encompasses a wide variety of different sub-fields [41], some of which are becoming established as separate fields in their own right.

There has been considerable research into the use of bio-inspired techniques in a wide variety of different application areas [41]. Traditionally, bio-inspired methods have been developed from the fields of evolution and learning. Evolutionary systems

are particularly employed to solve search or optimisation problems, often making use of population-based techniques such as genetic algorithms. Learning systems, such as neural networks, are used extensively in artificial intelligence applications [41].

However, as yet there has been little exploration of the use of biologically-inspired concepts in the real-time systems domain. This is largely due to the requirement for real-time systems to be predictable, whereas a large number of biologically-inspired methods are derived from concepts which are by their very nature unpredictable. Although this at first appears to be a fundamental mismatch between the two domains (and indeed this is likely to be true for safety-critical systems which must undergo rigorous certification procedures), there is still potential for the application of biologically-inspired ideas in systems with soft real-time requirements.

3.2 Properties of biological systems

A key property of many biological systems is their ability to adapt to their surroundings, in order to survive. In many cases, organisms have the ability to recover and rebuild following a catastrophic event. This ability to react to environmental changes and maintain a stable state is known as homeostasis, defined as [2]:

> *"a relatively stable state of equilibrium or a tendency toward such a state between the different but interdependent elements or groups of elements of an organism, population, or group"*

The properties exhibited by homeostatic systems are similar to those desired in many artificial systems: that they are robust and reliable, able to cope with changes in their environment [116]. These characteristics are difficult to achieve with traditional methods, but by studying the mechanisms by which homeostasis is achieved in

living organisms, new methods of enhancing reliability *in silico* can be developed [116]. It has been observed that, in order to achieve these goals of robustness and flexibility, systems will need to be produced which can act in a similar way to natural systems [113]. The term "organic" has been used to describe a system which exhibits some of these life-like properties. An "organic computer" is defined in [113] as:

> *"a technical system which adapts dynamically to the current conditions of its environment. It is self-organising, self-configuring, self-healing, self-protecting, self-explaining and context-aware"*

The concept of organic computing as outlined in [113] goes beyond trying to achieve homeostasis, incorporating methods such as emergence and self-organisation. A key aspect of organic computing is that of "controlled emergence": as it is the interactions between individual agents in a system which cause the system to exhibit its overall behaviour, therefore the creation of complex adaptive systems which are useful relies on being able to design autonomous agents which, by interaction with other agents, will produce a desired emergent effect [113]. Although the concept of controlled emergence is an attractive one, it is currently unrealised: research is ongoing and small systems have been produced which exhibit some of the characteristics desired in an organic computer [122], but as yet it is not known whether the idea is feasible when scaled up to larger systems.

Although the concepts behind organic computing are not yet practical, there has been success in trying to achieve homeostasis within computer systems [116]. In living organisms, homeostasis appears to be the result of a series of interactions in and between a number of systems. In vertebrates, the most complex type of organisms, homeostasis is thought to be achieved by a combination of the nervous system, endocrine system and the immune system. Simpler organisms may lack some of these systems (for example, plants lack nervous or complex immune

systems) but are still able to achieve homeostasis through the subset of systems they do possess. A number of *in silico* applications have taken inspiration from these different systems, both independently and in combination with other similar systems, in order to attempt to produce a homeostatic effect *in silico*.

3.3 The immune system

When considering the need for robust systems incorporating mechanisms such as homeostasis, the immune system in particular can be considered to exhibit characteristics which would be desirable in such a system. Biologically, the immune system is complex, and its operation is not completely understood. Indeed, its actual function is subject to debate amongst immunology experts. There is considerable debate amongst immunologists and there are various, sometimes conflicting, theories as to exactly how the immune system functions (for example: [89]; [108]) and exactly what its function within the body is.

The generally understood and accepted function of the immune system is to protect the host organism against attack by invading pathogens, and that the immune system is comprised of several interacting subsystems which are closely linked with the endocrine and the central nervous systems [153]. More radical views suggest that the immune system is part of a larger cognitive system that has a fundamental role in the maintenance of the body and the preservation of homeostasis [30], [31].

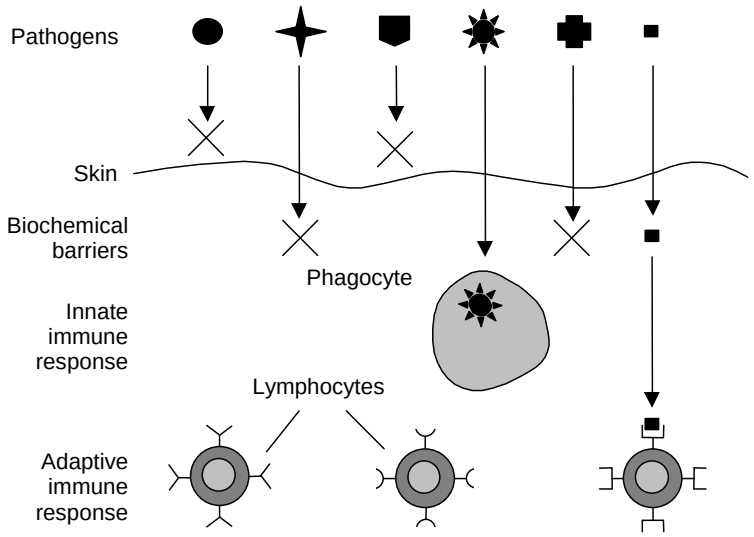


Figure 3.1: Layered immune response (from [135])

From a "traditional" viewpoint, the immune system is thought to comprise a series of layers, each providing protection against a specific type of pathogenic attack (Figure 3.1). The first layer of defence consists of barriers which prevent the entry of pathogens: either physical barriers preventing their entry (i.e. the skin); or biochemical substances such as mucus or stomach acid, intended to immobilise pathogens before they can cause damage. Should a pathogenic substance breach the barrier defences, the next line of defence against it is the innate immune system. It has the ability to detect certain classes of pathogens effectively, and mechanisms by which they can be removed. Finally, the adaptive immune system exists as a further line of defence to deal with pathogens against which the innate immune system is ineffective.

Depending on the complexity of the organism in question, not all the layers described here will necessarily be present: for example, only vertebrates possess an adaptive immune system (indeed in some literature, the term "vertebrate immune system" is used synonymously with "adaptive immune system"), while plants lack both innate and adaptive immune systems, but in many cases possess the ability to produce biochemical barriers (e.g. saps and resins) to protect against damage.

A brief description of the generally accepted operation of the innate and adaptive immune system follows. It is intended to give a general overview of the concepts on which problem-solving algorithms can be based. A fuller explanation is given in [37].

3.3.1 Innate immunity

The innate immune system is fundamental in the detection of invading pathogens, and in controlling the initial immune response to them. All multicellular organisms possess some form of innate immunity [109], and it can be considered the first line of defence against pathogenic attack once the physical barriers have been breached.

The cells which make up the innate immune system contain receptors which are able to detect specific molecular patterns. These patterns are associated specifically with external pathogens, and are never produced by the host organism. Consequently, these so-called "pathogen associated molecular patterns" (known almost universally in immunology literature as PAMPs) are analogous to a pathogenic "signature": their presence signifies that a pathogen must be present [110].

The detection of PAMPs by a receptor causes the stimulation of further components of the immune system by the release of chemical signals, known as "cytokines". Depending on the type of PAMP detected, and the location of its detection, the release of these cytokines may trigger a further innate response, such as the stimulation of phagocytic cells which engulf pathogens, or alternatively may cause activation of the adaptive immune system [110].

As a part of inducing an immune response, the detection of PAMPs will cause the secretion of chemical signals which increase the blood flow to the area where pathogenic activity is detected [109]. This effect is known as inflammation, and its purpose is simply to increase the number of immune cells in that area; this increases the likelihood that pathogens will be encountered by immune cells and immobilised.

The innate immune system is effective and has the ability to initiate a rapid immune response, but it is only able to completely defend against a relatively small number of specific pathogens, particularly given the high levels of mutation amongst bacteria and viruses which make up most pathogens [109]. In order to provide protection against all pathogens, in vertebrates the innate immune system is supported by an additional layer comprising the adaptive immune system, and is able, through the release of cytokines, to initiate an adaptive immune response.

3.3.2 Adaptive immunity

The adaptive immune system provides a further line of defence against infection by pathogens. The primary mechanism by which it operates is by the generation of a specific class of lymphocyte known as B-cells, which circulate in the bloodstream in large numbers. Each B-cell incorporates a protein structure capable of recognising a particular protein pattern associated with pathogenic behaviour, and can produce antibodies which act against that pathogen [22].

These B-cells are generated at random and consequently throughout the whole immune system are able to recognise a large number of different pathogenic patterns. When a receptor on a B-cell is stimulated by an antigen, a response is induced within that B-cell which causes it to undergo rapid cloning [23]. The clones are mutated as part of the cloning process, giving a large number of B-cells which are similar to the original but which have receptors for slightly different antigens. As well as undergoing cloning, stimulated cells begin to secrete antibodies which bind to pathogens and neutralise them.

This clonal response is repeated, ensuring a rapid production of antigen-specific B-cells, which are able to then produce antibodies to fight the invading pathogen. This process is able to produce a primary immune response, even where there is no previous knowledge of that particular pathogen. However, as well as this, the adaptive immune system is able to acquire an immunological memory, which enables it to produce a rapid response should the same pathogen (or one similar to it)

be encountered again [106]. This is achieved through the promotion of highly-activated B-cells to memory cells: these cells remain in the bloodstream for long periods of time and so are able to detect a resurgence of their specific pathogen quickly and stimulate a rapid immune response against it [155].

It has long been thought that the adaptive immune system operated completely independently of the innate immune system, however there is increasing evidence to suggest that the two are intrinsically linked and that the adaptive immune response is in fact controlled by the innate immune system [109]. In particular, it has been observed that the cloning of lymphocytes does not occur in the presence of antigens alone, but where both antigens and cytokines – released by the innate immune system on the detection of PAMPs – are present.

3.3.3 Mechanisms for detecting invaders

In order to protect an organism from attack by foreign invaders, an important function of the immune system is the ability of its components to be able to distinguish between "normal" operations and agents – those that are an integral part of the organism being protected – and "abnormal" activities, caused by invading pathogens [21]. The way in which the immune system goes about this has long been the subject of much debate amongst immunologists, and the exact mechanisms are still disputed [125].

The longest-standing and most widely accepted theory of immunology is the "infectious non-self" model [87]. In this model, the immune system is understood to have an inbuilt ability to differentiate between actions and agents which are a part of its normal operation, and those which are abnormal. The model associates normal actions with "self" and the remainder (i.e. those which are abnormal) with "non-self". The premise behind the infectious non-self model is that all agents or actions which do not match with the "self" space are considered harmful to the organism: therefore an immune response is initiated against "non-self" agents [88].

The primary mechanism for the initiation of an immune response in the infectious non-self model is B-cells: on generation, B-cells are subjected to a "self-screening" process during which they are tested against all known examples of self, and those which are stimulated by self tissues or agents are eliminated in order to prevent the immune system from attacking parts of the organism that it is intended to protect. The remaining B-cells are released into the bloodstream where they are then able to detect invading pathogens. The sheer number of B-cells present in the bloodstream, and the random nature of their generation, leads to a wide range of pathogens being recognisable.

The infectious non-self model is considered by the majority of immunologists to be an accurate representation of how the immune system functions [125], however observations have been made which are inconsistent with this model, such as the immune system's ability to become tolerant to certain types of transplanted tissues [107]. Various alternatives have been suggested: the best known (and arguably most controversial) is the danger model, initially proposed by Matzinger [107]. In contrast with accepted thinking, the danger model postulates that it is not reaction to non-self which promotes an immune response, rather that the response is caused by reaction to distress signals, produced by cells dying unexpectedly. Consequently, the theory states, the immune system is not concerned with protection against all agents identified as "non-self", but is instead stimulated to protect bodily tissues from harm, and contains mechanisms allowing it to become tolerant of agents which cause no harm.

As well as providing mechanisms and agents by which an immune response can be initiated, the danger model is able to provide explanations for a variety of issues which were not satisfactorily addressed by previous models, such as the need to provide an adjuvant with certain vaccines, without which the vaccine would not cause an immune response to be initiated [108]. Although at the time the theory was first conceived the ideas were unproven, it has since been established that cells which

die under duress do give off a chemical signal indicating that harm has been caused to them [63].

Initially, the danger model was only concerned with how the immune system maintains tolerance towards non-invading tissues, however it has since been extended and now is able to offer explanations for the choice between different immune responses, based on the type of tissue producing the danger signal [108].

3.3.4 The immune system and homeostasis

The emergence of new immune theories has given rise to the notion that, rather than simply being a mechanism for defence against invading pathogens, the immune system is part of a larger mechanism allowing the maintenance of homeostasis within the host organism.

The danger model goes some way towards this by suggesting that immune responses are brought about in response to damage to body tissues rather than simply in response to foreign agents, and it has been observed that necrotic cells (i.e. those which die under duress) emit different signals from those which are apoptotic (i.e. those which die naturally). It has been suggested that, in addition to the primary level of immune response induced by the presence of necrotic cells, that immune system components are also involved in the processes associated with apoptosis; i.e. that the immune system also plays a fundamental role in body maintenance [30]. One theory is that, rather than being a separate system, the immune system is in fact part of a much larger, closely interlinked series of systems which are responsible for body maintenance and protection [31]. It has been postulated that this system, referred to as the "immune homunculus" exhibits many properties of a cognitive system and is responsible for all aspects of maintenance and homeostasis within that organism.

3.4 Application of immune principles in computer systems: Artificial Immune Systems

The domain of theoretical immunology provides a large number of interesting models and processes which can be observed. Many of the aspects of the immune system give rise to characteristics which could be considered to be desirable in systems implemented *in silico*. As has occurred with other principles from the natural world, the behaviour exhibited by the immune system can used as a basis for problem solving in *in silico* computer systems.

Drawing on inspiration from the immune system, the last decade has seen the proliferation of a variety of immune-inspired techniques grouped together in a domain known as "Artificial Immune Systems" (AIS). These are a range of biologically-inspired techniques which can be applied to various problems. The AIS domain encompasses a wide range of work from immune system modelling through to engineered applications using immune principles, and consequently it has become difficult to formalise exactly what the domain represents. Perhaps one of the more commonly cited definitions of AIS is that they are:

> *"adaptive systems, inspired by theoretical immunology and observed immune functions, principles and models, which are applied to problem solving"[37]*

The idea of using immune system principles in computer science applications first arose in the middle of the 1980s [53]. The potential for use in the fields of machine learning and artificial intelligence was noted, but the idea was not investigated fully until the 1990s. By 1998, immune-inspired methods were the subject of special sessions at a number of computational intelligence conferences, and the first edited book on the subject [33] was published.

Until relatively recently the majority of AIS research has focused on concepts and principles inspired by the adaptive immune system [80]. In particular, the majority of work in the field has focused on a small number of adaptive immune methods: primarily the principles of negative selection, clonal selection and immune networks [136], and techniques based on these principles have been subject to refinement in order to produce viable methods by which problems can be solved. Recently, mirroring work in the field of immunology, considerable research has been carried out examining the potential of techniques based around the principles of innate immunity [137]. This work is beginning to show potential in a number of application areas.

The existence of various competing immunological theories does not pose a problem when deriving algorithms and processes based on the immune system – in fact, the wide variety of different viewpoints provides an extensive range of principles on which to base immune-inspired algorithms. For the purposes of a computer system, it does not matter if the principle on which it is based is unproven, (or even that a principle may have been proven to be scientifically implausible): it is that particular algorithm's effectiveness in solving relevant problems which is of interest [6].

Examples of this from other biologically-inspired methods include the use of Lamarckian principles in the design of evolutionary algorithms: their use can enhance the efficiency of an algorithm, even though the principles of Lamarckian evolution have been scientifically disproven as a biological mechanism [143]. In the context of AIS, this is especially relevant to Matzinger's danger model. Although controversial and as-yet unproven, it has been noted that the principles of the danger model may provide useful foundations on which to base AIS, and consequently should be investigated [4].

3.5 A framework for AIS development

As stated in section 3.4, it is important to remember when considering an algorithm inspired by a natural process – including AIS – that the primary purpose of that

algorithm is to solve a specific problem, rather than to be an accurate representation of the underlying natural process [59]. It is therefore important to design an algorithm which solves the problem, rather than trying to shoehorn the problem into a specific implementation which, while being exactly faithful to the theory, is not appropriate for that particular problem [59]. This requires a methodical approach to ensure that all aspects of the problem are covered and that the finished system is effective in solving the target problem.

de Castro and Timmis suggest that, for the development of any system based around a biological metaphor, a development framework must incorporate a small set of fundamental elements [37]:

- *A representation of the components of the system*
- *A set of mechanisms to evaluate the interaction of individuals with the environment and each other*
- *Procedures of adaptation*

This framework can be considered as a layered approach to solving problems using AIS (see Figure 3.2). All systems have their foundations in the application domain, and it must be considered how the components of the system are to be represented, in the context of the components of the immune system. Once the representation has been chosen, the interaction of the components can then be quantified. Generally, this requires some means of classifying the "closeness" of a particular element to a number of other elements, referred to as an affinity measure. These choice of affinity measures is often partially (or in some cases totally) dependant on the representation chosen. Finally, an immune algorithm can be applied, which defines the behaviour of the system. By following these stages, an engineered solution is reached.

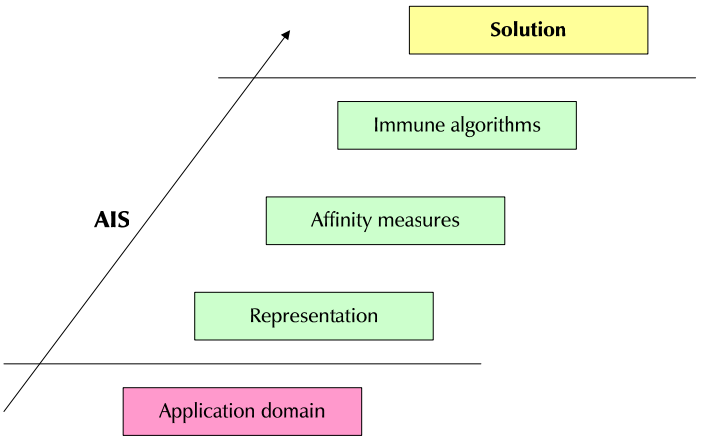


Figure 3.2: Layered framework for AIS development (from [37])

3.5.1 Representation

When designing an AIS, the representation defines the relationship between the components of the original problem, and the corresponding components present in the immune system. For example, this may include the definitions of the B-cells and antigens, in terms of the structure of the problem system, and indicate how they are to be encoded in the final system.

There are a large number of ways in which any particular problem can be encoded. In particular, de Castro and Timmis consider binary representations and integer representations. They note, however, that all representations can be generalised as either discrete or continuous [37].

Freitas and Timmis [59] note that it is important to choose the representation carefully such that it encompasses all the relevant data. In particular, they note that it is not desirable to "adapt" data to force it to fit into a particular representation: this approach often leads to potentially important data being disregarded simply because it does not fit into the desired representation. They observe that, in a large number of cases, picking a representation which is entirely continuous or entirely discrete often leads to attributes being lost, or else converted inappropriately. They suggest the use of hybrid representations appropriate to the problem data and recommend their use to ensure that all relevant data is utilised [59].

3.5.2 Affinity

In the context of the immune system, the term "affinity" defines the closeness of the match between an antigen and a particular antibody. This concept is carried through into the field of AIS. The choice of affinity function is closely related to the representation being used: different types of data must be compared in different ways.

The simplest affinity measures, in terms of computation, occur when a binary representation is used. When considering antibodies' affinities with antigens, the closeness of matches is determined by the closeness of the physical shapes of the molecules. The most natural translation of this idea, when a binary representation is used, is simply to compute the Hamming distance between two bit strings: the lesser the Hamming distance, the greater the similarity.

Freitas and Timmis note that other affinity measures have been used with binary representations. Some, such as the r-contiguous bits rule, have been justified as being superior on the grounds that they are more "immunologically plausible" than straightforward Hamming distances [84]. It is noted, however, that this approach introduces an ordering to the set of attributes which make up the antigen/antibody: an ordering which is not necessarily present in the original data, particularly if the attributes being considered are independent of each other [59]. The implications of any such bias introduced by such functions must be carefully considered, and it should be noted that, while an algorithm may be based on a biological principle, it does not need to be an exact copy of it but should be carefully engineered to fit the problem data. Therefore, the key characteristic of any affinity measure is that it is appropriate for the data and its chosen representation, particularly where a hybrid representation is used.

Continuous or numeric data often calculates an affinity based on the Euclidean (straight line) distance between attribute sets. In n-dimensional Euclidean space, the

distance between points $P = (p_1, p_2 \ldots p_n)$ and $Q = (q_1, q_2 \ldots q_n)$ is given by the equation

$$\sqrt{\sum_{i=1}^{n} (p_i - q_i)^2} \tag{1}$$

The Euclidean distance is often perceived as the most natural way of measuring distance. It is of moderate computational complexity, but has the advantage that it is scalable when dealing with multi-dimensional attribute sets.

de Castro and Timmis note that there are other viable alternatives to Euclidean distance, in particular the Manhattan distance (sometimes known as "taxicab distance") [37]. This is given by the equation

$$\sum_{i=1}^{n} |p_i - q_i| \tag{2}$$

de Castro and Timmis felt that the potential of the Manhattan distance had not been widely explored in the literature [37]. In particular, it is useful to note that the computational complexity is less than that of the classical Euclidean distance. In certain circumstances, the two measures can give different orderings for the same values (Figure 3.3).

Freitas and Timmis explore the differences between Euclidean and Manhattan distances in some detail, and they note that the Manhattan distance is less sensitive to noisy data (particularly where the noise is not present in all dimensions) [59]. As with discrete data, they state that the choice of affinity measure for continuous data is, again, dependent on the problem data.

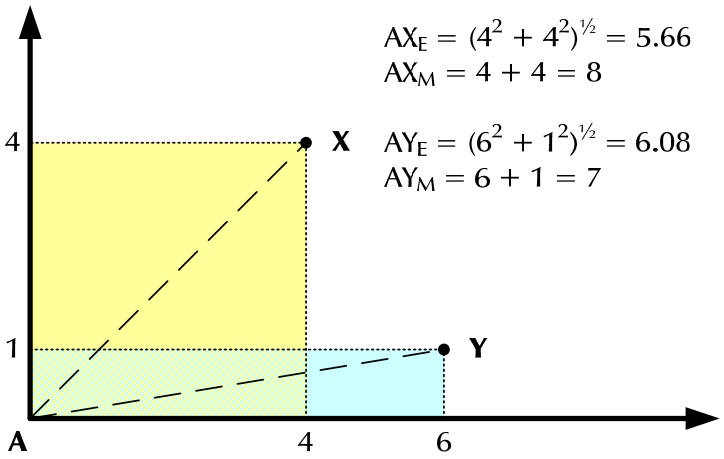


Figure 3.3: Euclidean and Manhattan distances (from [59]). Using Euclidean measurements, X is closer to A; using Manhattan measurements, Y is closer to A.

3.5.3 Immune algorithms

Once a suitable representation and corresponding affinity measure have been chosen, an immune algorithm can be selected. There are a variety of immune algorithms which can be used in the development of an AIS. Different algorithms model different aspects of the operation of the natural immune system, or make use of alternative immune theories: consequently there are a number of approaches which can be used to solve any particular problem. These will be discussed in more detail in the next section.

As mentioned in section 3.4, AIS have been applied in a wide range of problem areas. Most of these problems fall into one of two broad categories: classification problems require that a particular value be classified as a member of a particular class, generally based on its similarities with examples known to belong to that class; while optimisation problems are concerned with trying to optimise a particular attribute or quality.

The majority of problems solved using AIS techniques ultimately reduce to classification problems (for example: anomaly detection can be reduced to classification, as situations are either “anomalous” or “not anomalous”), and so most AIS algorithms and techniques are intended to solve classification problems. In many

cases, these algorithms can be extended or altered such that they can also be used to solve optimisation problems.

The components of classification systems can be easily mapped to the components of an AIS. The data being classified is normally considered as the antigen, and the available classes of data are represented as members of a lymphocyte population. The relationship between the antigen and each of the lymphocytes can then be established in terms of an affinity measure: the lymphocyte which has the highest affinity with the antigen will be the one representing the most appropriate classification.

The earliest algorithms, used in the pioneering work by Forrest [58], [56], are negative selection algorithms, the purpose of which is to detect patterns which do not belong to a defined set. The process by which antibodies are generated in a negative selection algorithm involves censorship of those antibodies which are stimulated by normal operation: a set of initial antibodies are generated at random, and to determine their suitability, their affinity with all members of the self space is computed. If an antibody has too high an affinity towards the self space, it is eliminated; otherwise it is added to the detector set. The process is repeated until an appropriate number of detectors has been generated, which will vary depending on the problem in question and the coverage of the search space required [97].

There are a number of important points which can be made with respect to these algorithms. Firstly, there is a significant issue when using negative selection algorithms for classification. Traditionally, classification algorithms are trained using samples from all classes (in this case, both self and non-self examples), however for the negative selection algorithm, the training phase of the algorithm only evaluates the antibodies with respect to one classification (i.e. self). However in use, the negative selection algorithm is required to classify antigens as either self or non-self, despite the algorithm having no prior experience of any actual examples of non-self data [59].

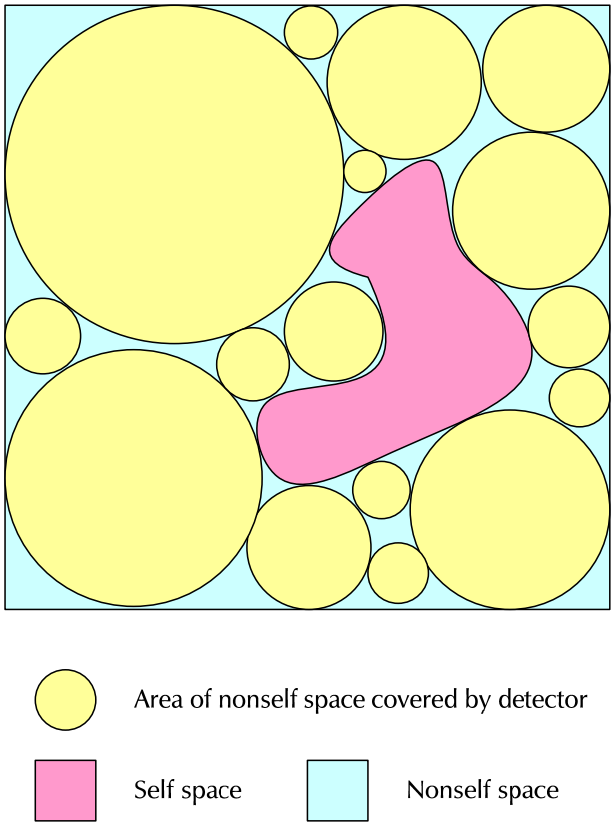


Figure 3.4: The negative selection algorithm in 2D, showing detectors covering nonself space

Secondly, the way by which the negative selection algorithm represents and generates detectors results in it being impossible to determine whether there is complete coverage of the nonself space [132]. The algorithm assumes that each detector is able to cover a significant proportion of the total shape space, and that random generation of detectors will produce a range of detectors which cover the majority of the non-self space. However, unless a large number of detectors are used, there is a tendency for the coverage of the non-self space to not be complete, as can be seen in Figure 3.4 [132].

The coverage issue becomes increasingly relevant as the complexity of problems increases. The detectors generated by the negative selection algorithm are normally represented as a hyperspherical object in a multidimensional Euclidean space, with a known centre and diameter [77]. The advantage of this is computational simplicity: using Euclidean geometry, the affinity between any point in the shape space to any particular detector can be found according to equation 1. However, it has been observed that as the dimensionality of the shape space increases, the amount of

volume covered by each individual detector decreases as a proportion of the overall shape space [133], [131], resulting in poor coverage unless a very large number of very specific detectors are used. Freitas and Timmis suggest that negative selection is not a particularly suitable immune algorithm for classification problems [59], and further work suggests that negative selection algorithms are also deficient for anomaly detection purposes [130], [131]. Despite this, it is still argued by many that negative selection principles can be of use [91], [92], and there is still a significant amount of work being published which makes use of negative selection methods. However for most in silico applications negative selection can be considered to be unsuitable [81].

A significant class of immune algorithms are those based on the principle of clonal selection. This is thought to be the means by which the immune system produces antibodies when an unknown pathogen is detected [38]. Initially, generated antibodies are screened to remove those which detect self patterns. The remaining antibodies are then presented to the invading pathogen and their affinities established. Each antibody is then cloned proportionally with its affinity, and mutated inversely with affinity: consequently, antibodies with high affinity are extensively cloned, with low levels of mutation; those antibodies with low affinity undergo high levels of mutation and little cloning. This behaviour allows for rapid generation of high-affinity antibodies. The original clonal selection algorithm following these principles was CLONALG [38].

Clonal selection algorithms share many characteristics with genetic algorithms, the key difference between them being that there is no crossover involved in clonal selection [57]. Features of clonal selection not provided by standard GAs are the cloning and mutation being proportional to the affinity. When used for classification, clonal selection algorithms are superior to negative selection, as the training stage of the algorithm involves examples of both self (in the initial screening stage) and non-self (in the cloning stage) classes [59].

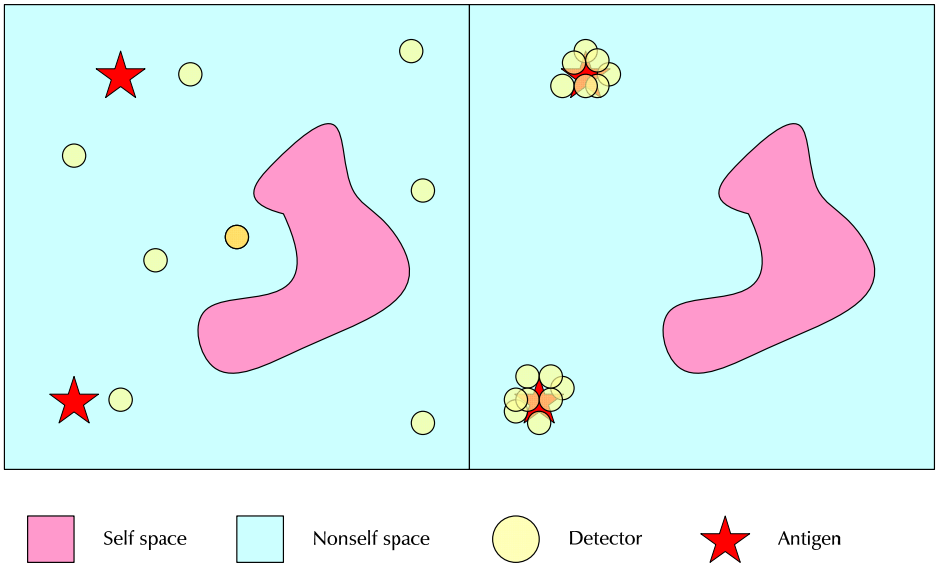


Figure 3.5: Detector population before (left) and after (right) clonal selection procedure

Another significant class of immune algorithms is those based on immune network principles, first proposed by Jerne [90]. The fundamental difference between immune networks and other immunological principles is the interaction between the components of the immune system. The clonal selection theory proposes that immune cells are normally at rest, and become active only when stimulated by a foreign antigen. The immune network concept proposes that the immune system is formed of a set of components which can recognise each other even in the absence of foreign antigens.

To achieve this, it has been suggested that antibodies contain multiple active sites: firstly, a paratope – a part responsible for recognising the patterns presented by antigens and other molecules (these patterns are known as epitopes); and an idiotope – an epitope presented by an antibody – which can be recognised by another body. The presence of these idiotopes causes the immune system to continually self-stimulate itself into producing antibodies: this constant self-stimulation promotes immunological memory.

Various immune algorithms have been proposed which draw on the principles of immune network theory to varying degrees. One such "artificial immune network" is the AINET algorithm [39] and its variants [36], [28]. AINET employs an affinity

between individual B-cells to maintain diversity in the search space being covered, this helps to improve the efficiency of the overall system by removing members of the B-cell population which are similar to each other in addition to removing those which are of low affinity to the antigen being presented. The effect of this behaviour is a rapid clustering in the search space around the most effective results.

3.5.4 Immune algorithms for optimisation

Although the same basic methods can be used for both classification and optimisation problems, optimisation problems are solved in a slightly different way, and so techniques designed for use in classification systems must be modified to allow them to solve optimisation problems. The most significant difference between the two types of problem is that optimisation problems have no concept of "antigen": there is no pre-determined "optimal" solution with which to compare the output of the AIS, and thus no direct measure of affinity. Instead, the suitability of candidate solutions (antibodies) must be evaluated by means of a fitness function [37].

The effective operation of an optimising AIS (or indeed any optimisation algorithm) is highly dependent on the choice of fitness function. The value of the fitness function may at first appear to be unrelated to the attributes being altered, and is often not easy to predict based solely on examination of the attributes: in this case, the value being measured can be considered an emergent property [7]. Often there can be several fitness measures which must be combined according to a specific hierarchy to calculate the overall fitness of any particular solution: it is necessary to consider the relative importance of these individual measures carefully, such that they can be weighted accordingly in the combined fitness function. Slightly changing the weightings involved in the fitness function calculation can result in significant changes to the results obtained [11].

There exist versions of several immune algorithms which are designed for optimisation. These are mostly based on the clonal selection and immune network principles [151]. In particular, there is an optimisation-oriented version of the

CLONALG clonal selection algorithm, which has been commonly used [40]. The diversity in cell populations brought about by the use of immune network inspired methods has brought about a number of immune network algorithms for optimisation, including an optimisation-oriented version of the AINET immune network algorithm called opt-AINET, which has been applied to a range of problems [36].

The use of AIS for optimisation has been examined in the literature [151]. However, it has been observed that although these optimisation techniques give reasonable results [94], there is little benefit in using immune algorithms over any other evolutionary algorithm for static optimisation, either in terms of the quality of the results achieved or the time taken for the solutions to be found [138]. A comparison of two AIS optimisation methods and a hybrid genetic algorithm found that, while in individual cases the AIS methods outperformed the genetic algorithms, when the results were compared across a variety of trial problems there was no overall benefit in using the AIS methods [138].

3.6 Applications of AIS

Some of the earliest work which employed immune principles in computer systems was the work carried out by Stephanie Forrest, in the field of computer security [58]. This work aimed to detect unauthorised changes in a computer, by creating a system which was able to distinguish between authorised uses from unauthorised attacks. Around the same time (but independently of Forrest's work), Kephart proposed the idea of creating an immune system for computers based around biological principles [95].

Both of these early works were concerned with the detection and removal of viruses from computer systems, and were aiming to utilise the principles of the natural immune system to provide an adaptive protection system, allowing computers to detect abnormal behaviour. Forrest's work was based on the concept of self-nonself discrimination, categorising authorised use as "self" and unauthorised processes as

"nonself". Detection was accomplished by negative selection, allowing the computer to formulate an idea of "normal" operation, and therefore allowing it to detect operations occurring outside these normal parameters, in much the same way as the immune system detects invading pathogens. The ideas proposed by Kephart noted this need to be able to distinguish "self" from "non-self", but proposed a mechanism of "decoy" applications which would deliberately attempt to get infected by viruses to detect their presence.

The properties of the natural immune system seem to make it a particularly appropriate metaphor on which to base anti-virus and other security systems, however, the underlying principles of the immune systems, that is learning and adaptation, make the concept of AIS suitable for solving a far wider range of problems [37].

The ability of lymphocytes and other immune components to detect specific protein patterns can be applied in the domain of pattern recognition, allowing a specific pattern to be classified according to a series of previously examined pattern [35], [26]. Similar techniques may be applied to data mining and classification problems [123]. Applications such as data mining could benefit from an immune-inspired approach, by training an "immune system" to identify classes of data, then using this system to classify previously unseen data [60].

The ability of the immune system to identify and destroy invaders is of relevance to engineers attempting to produce reliable or fault-tolerant systems, where the ability to detect and remove abnormalities can be used to increase the dependability of systems [8]. In these systems, there are obvious parallels between the functionality of the natural immune system, and the desired functionality of a fault-tolerant or fault detection system. A key focus here is the area of computer security [95], [58]. Such systems may make use of methods allowing the discrimination of self from non-self [56]; alternatively could be based around alternative theories such as the danger

model [4], [73]; or the mechanisms by which it is proposed that the immune system aids in the maintenance of homeostasis [126].

Other work has focussed on the principles by which the immune system develops immunological memory to inspire methods for machine learning, both supervised [86] and unsupervised [139]. Taking inspiration from parts of the immune system has seen various immune algorithms used for function optimisation [49], [140].

One particular implementation which is relevant to the RTES reliability problem is the application of AIS techniques to scheduling problems. Task scheduling is a fundamental part of the operation of real-time systems, as the very definition of RTS requires that the operation of the system must meet specific deadlines. In particular there have been a number of applications of AIS techniques to job-shop scheduling, such as [79],[78], [29] and [51], with promising results, although there does not appear to have been any comparison of the effectiveness of AIS techniques with other similar bio-inspired methods, as there has been with the optimisation algorithms outlined in section 3.5.4.

The fact that AIS has been successfully employed as a solution to job-shop scheduling problems indicates that it has good potential for application in other areas of scheduling, although there are differences between job-shop scheduling and the task scheduling employed in RTS. A key difference is that job-shop schedules are generally finite and absolute (i.e. job 3 follows job 2, which follows job 1) whereas the scheduling of tasks in a system has no discernable endpoint, and is generally controlled by some form of overall scheduling policy.

In addition, a number of RTES problems, such as the deadline overrun problem or the bandwidth bottleneck problem, can be considered as anomaly detection problems, and as such they exhibit similar characteristics to some of the anomaly detection problems to which AIS has been successfully employed.

3.6.1 Efficiency and resource issues

One significant issue with the use of AIS (or indeed, any evolutionary techniques) is the efficiency of the algorithms and procedures involved, and the computational resources that they require. Most evolutionary techniques are derived from natural processes which occur over long periods of time (in the case of evolution, millions of years), and it is often the case that to achieve effective solutions requires considerable computational resources [9], not just in terms of processing power but also for the storage of large amounts of data (in the case of AIS, for B-cell populations, memory cells, gene libraries etc).

While this is of minor relevance in the academic world, and in many applications where large amounts of processing power and data storage are readily available, it becomes a significant issue in the world of embedded systems. The constrained resource availability in the majority of embedded systems conflicts with the requirements of many immune-inspired algorithms such as those based on clonal selection or immune network theories. These algorithms make use of a large population of elements, requiring a significant amount of memory for storage [37]. In addition, there is often a need to calculate affinity values, either between a specific antigen and all members of the cell population such as in clonal selection algorithms [38]; or between the members of the cell population itself as in immune network algorithms [39].

These factors are likely to make adaptive immune-inspired methods impractical for use in embedded systems. However, the simpler behaviour of the innate immune system compared with the adaptive immune systems is of particular interest when considering methods of anomaly detection for use in embedded systems. The next section will examine recent research into innate-immune inspired methods, and assess their suitability for use in resource-constrained environments.

3.7 Innate-immune derived AIS methods

At the time de Castro and Timmis defined their framework in [37] as outlined above, virtually all the published work in the AIS domain had taken inspiration from the principles of adaptive immunity . More recently, there has been an increased focus on the innate immune system in the field of immunology [142]. Mirroring this increased focus on innate immunity amongst immunologists, the AIS community has also been investigating the potential of techniques inspired by observations of the innate immune system [34]. This has given rise to a new range of ideas, algorithms and applications inspired by the principles of innate immunity.

Innate immune-inspired methods and algorithms are based around the structures and principles of the innate immune system, such as the class of cells known as antigen presenting cells (APCs), or the mechanisms of inflammation. As innate immune inspired methods are a comparatively new area of AIS research – at the time the research for this thesis was commenced, research into innate immune-inspired methods had only just begun - there are relatively few papers in the published literature compared with the more traditional adaptive AIS methods. At this stage the majority of the published work has examined applications of innate immune methods to anomaly detection problems in a similar vein to the earliest work based around adaptive immune principles.

A significant amount of research into innate immune methods has focussed on aspects of the danger model. Although acknowledged that the danger model is controversial amongst immunologists, this does not prevent it from having great potential as inspiration for problem-solving techniques [4]. In particular, the danger model has been examined in the context of intrusion detection systems [3]: the concept of danger outlined in the danger model can be considered to be similar to the type of attacks which can occur against computer systems.

3.7.1 Inspiration from Dendritic Cells

A recent focus amongst immunologists studying the effects of innate immunity is the observation of APCs responsible for the collection of antigens present in the bloodstream of the host organism. One class of these APCs are Dendritic Cells (DCs). DCs collect antigens, which are then presented to T-cells in a lymph node, along with information about the concentrations of cytokine signals associated with cell death, either necrotic or apoptotic. The information provided by DCs is used to determine whether the initiation of an immune response against those antigens is necessary.

In order to go about this, DCs exist in a number of different states throughout their lifecycle. All DCs start out in an "immature" state. Immature DCs exist within the body tissues, and collect samples of antigen and cytokine signals within those tissues. A DC which experiences high levels of chemical signals associated with "danger", caused for example by cells dying through necrosis, or by the DC encountering high concentrations of PAMPs, undergoes transformation into a "mature" state once the number of danger signals encountered exceeds a threshold value: it then travels to a lymph node where it presents its antigen. If a DC detects high levels of signals associated with apoptosis, and low levels of danger signals, then it enters a different "semi-mature" state before travelling to the lymph node to present its antigen in the same way as a mature DC.

Within the lymph node, mature DCs secrete inflammatory cytokines in addition to presenting their antigen to T-cells: these inflammatory cytokines serve to activate the T-cells and thus initiate an immune response against the presented antigens. However, semi-mature DCs secrete an anti-inflammatory cytokine when they present their antigens: the presence of semi-mature DCs therefore serves to temper any induced immune response.

The method by which this antigen presentation method operates is essentially based on guilt by association: an antigen observed in the presence of danger signals is assumed to be the cause of that danger. In order to avoid the initiation of an immune

response against benign antigens, the T-cells in the lymph node rely on the costimulatory cytokines produced by a number of DCs. Consequently, the relative ratio of mature to semi-mature DCs presenting the same antigens will determine whether an immune response is induced. A high ratio of mature DCs presenting a particular antigen will result in an immune response being initiated against it, whereas a high ratio of semi-mature DCs will lead to tolerance of that antigen. Although in some cases individual DCs may falsely present benign antigens, the combined effect of many DCs presenting the same antigen can be taken to indicate that that particular antigen is indeed dangerous and requires the initiation of an immune response against it.

3.7.2 DCs in silico: the Dendritic Cell Algorithm

The observed behaviour of DCs *in vivo* can be readily mapped to a number of problems observed in *in silico* systems, providing them with a mechanism by which anomalies can be detected.

The Dendritic Cell Algorithm (DCA) is an algorithm which is derived from the operation of DCs as described above, intended for use in *in silico* systems. The idea of basing an immune-inspired system on DCs was first outlined in [72], and further clarified in [74].

In its canonical form [74], the algorithm's exact operation is based on a series of *in vitro* observations of the behaviour of individual living DCs. The broad function of the DCA is to provide an indication of danger levels associated with different parts of the system. Based on the idea of a DC detecting chemical signals, the DCA makes use of virtual DCs and virtual signals derived from various aspects of the system being monitored [74].

The combination of these input signals causes the virtual DC to mature in the same way as its biological counterpart. By using a population of DCs monitoring different

components of the system, the output from a number of DCs can be combined to deduce information about the overall state of the system.

The canonical implementation of the DCA, as outlined in [74], makes use of a framework (known as libtissue) which provides support structures permitting interactions between a set of signals, a set of specified antigen and a population of DCs. Within this framework, each DC takes the form of a data structure, consisting of a matrix of input signals, a matrix of output signals, an antigen store and a migration threshold. The structure of such a DC is shown in Figure 3.6.

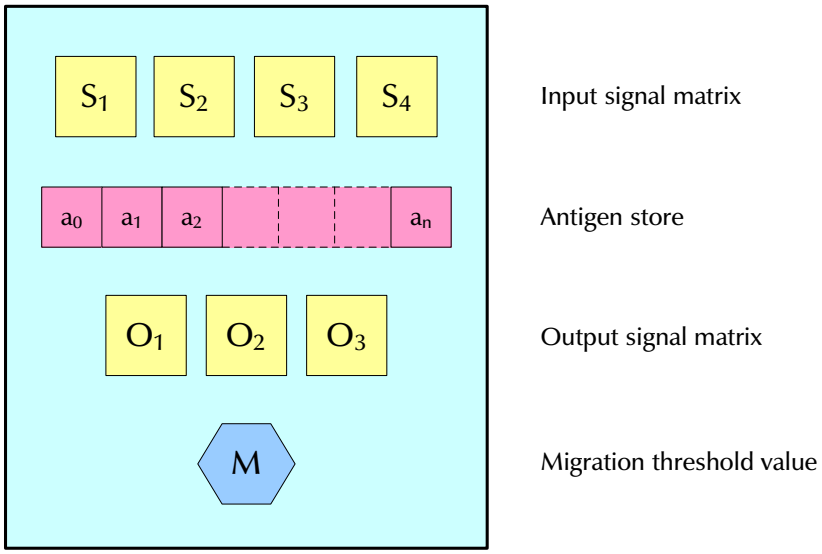


Figure 3.6: Data structure of single DC

The operation of the algorithm, as defined in [74] and illustrated in pseudocode in Figure 3.7, is based around a regular cell update cycle. During each update cycle, each member of the DC population updates its input signal matrix and antigen store by evaluating the values passed to it from the environment.

```
while dc cycle count < max dc cycle count loop
  update antigen and signal levels from environment;

  for all DCs in population loop
    sample associated antigen;
    sample associated signals;
    compute cycle output signal;
    compute cumulative output signal;

    if cumulative output signal > migration threshold then
      DC becomes mature;
      remove DC from population;
      migrate DC to lymph node;
    end if;

  end loop;

  dc cycle count = dc cycle count + 1;

end loop;

remaining immature DCs become semi-mature;
remove remaining DCs from population;
migrate DCs to lymph node;

analyse DC antigen and output signal levels;
```

Figure 3.7: Pseudocode representation of the DCA (from [73])

Each cell then computes values for each of its required output signals. This operation takes some or all of the input signals and combines them to produce the appropriate output value. In determining the value of each output signal, the values of each of the available input signals are weighted according to a set of weighting values: therefore, the algorithm can be tuned to give preference to specific signals or categories of signals.

For example, if the presence of a particular category of signal always indicates the presence of danger in the system, that signal can be assigned a high weighting value in order that the presence of that signal is highly likely to cause a DC to become mature. Conversely, if there are signals which when present indicate the absence of danger in the system, these can be weighted to counteract the effect of any danger signals present. The procedure by which a DC calculates its output values using the weighting values is shown in Figure 3.8.

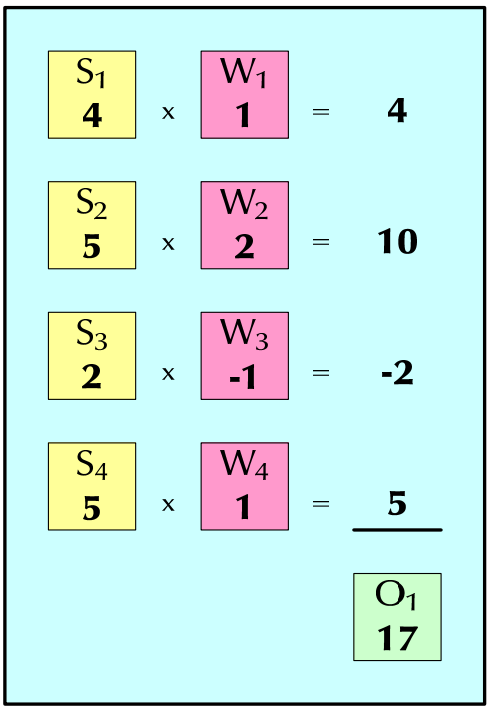


Figure 3.8: Deriving DC output signal values from input signal and weighting values

When DCs are added to the population, each immature cell has a maximum lifespan defined as a number of cell update cycles. If at the end of this lifecycle a DC has not reached maturity, it will undergo transformation into the semi-mature state. However, if at any point in its lifecycle the value of a DC's output signal passes a specific migration threshold, it is transformed into the mature state, whereupon it presents its antigen and signal values to a lymph node structure. This then analyses the ratio of mature to semi-mature DCs presenting each antigen to derive a measure of danger for that antigen, referred to in [74] as the "mature context antigen value" or MCAV – the closer this value is to 1, the greater the level of danger associated with that antigen.

The purpose of the DCA as designed is to provide information pertaining to the levels of danger associated with the various components of the system to which it is applied. This information can then be passed to other mechanisms which allow the particular problem to be resolved, but this is outside the scope of the DCA itself.

3.7.3 Applications of the DCA

The DCA was created with a view to solving anomaly detection problems, specifically in the domain of intrusion detection [73]. Applications in this area have included the detection of port scan attacks [74] and SYN scan attacks [70]. These applications have been successful, although it is worth noting that these applications have been carried out on pre-prepared datasets with known attack properties with the detection algorithm run on the dataset "offline", rather than being implemented on an actual online system.

In addition to the applications by the algorithm's creators in intrusion detection, the DCA has also been applied to problems in other domains. These applications include anomaly detection in sensor networks [96] and classification on robots [115]. Its particular advantage is that it is generally a lightweight solution, with little on-line adaptation involved. This requires fewer resources than AIS techniques which are based around adaptive immune principles, and is therefore more likely to be usable in a resource-constrained system.

3.8 Summary: biologically-inspired methods in RTES

The concept of AIS has the potential to help solve problems in the real-time systems domain. Of particular interest is the use of AIS for anomaly detection or fault tolerance. Traditionally, real-time systems developers make use of formal analysis techniques at the design stage to ensure that the system will run correctly. As systems become more complex, it becomes increasingly impractical to perform full formal analysis.

Instead of using static analysis to prove that a system contains no faults, an AIS-based technique could be employed to detect and fix faults as they occur. In addition to this, the system could learn the particular conditions which are associated with faults, such that when these conditions are detected, the system's behaviour can be altered before the problem occurs, so preventing the problem from arising again.

There are a number of approaches which could be taken when applying AIS to solve a problem. In majority of applications of AIS encountered in the literature, it seems commonplace to make use of existing concepts and models, and then use AIS methods to tweak those models to produce a more effective outcome [37]. An alternative approach which seems to be less popular in the literature is the direct implementation of a solution using AIS.

As an example from an RTES perspective, consider the use of a task scheduler which controls the execution of tasks in a system. Taking the first approach, a standard real-time scheduling policy would be used, incorporating AIS to monitor the system and alter the task properties or the behaviour of the scheduler as necessary. In contrast, is would also be possible to base the entire scheduler on an AIS. The benefits of each approach do not yet appear to have been fully evaluated.

The next chapter will consider the options for the use of biologically-inspired methods as a means to solve problems in RTES development, and propose a research strategy and hypothesis which will guide further investigation.

4

Thesis aims

In chapter 1, it was established that the development of RTES often encounters problems, due in part to a mismatch between conflicting requirements in the development process. As discussed in chapter 2, traditional software development has established a number methods which allow an increase the complexity of the systems which can be produced, however these techniques are in general only focussed on functionality. Consequently non-functional characteristics such as real-time properties can to date only be guaranteed by the use of formal analysis techniques which are time consuming, expensive and inflexible.

Chapter 3 investigated the use of techniques based on observations of biological systems, and established that there is potential for a new method of real-time systems development, which incorporates aspects of living systems (such as adaptation and homeostasis) into *in silico* implementations. The use of these methods

may allow a system to detect faults within itself as they occur: these faults and their characteristics can then be highlighted and the information fed back to the system's designers, allowing them to focus their debugging efforts on the parts of systems where faults arise. In the case of emergent properties, the parts of the system which exhibit symptoms of faults or failure may not necessarily be the source of the problem, but a mechanism for identifying these symptoms can provide a useful starting point for investigation.

In particular, chapter 3 highlighted the potential for the use of AIS in solving the problems associated with RTES development, although it also noted that there were significant issues, particularly with respect to resource usage, which would need to be overcome in order for immune-inspired techniques to be useful to RTES designers.

When considering the application of these methods in embedded systems where resources are often constrained, there is a distinct possibility that methods based around adaptive immunity are unsuitable due to the complexity of the population-based algorithms used. Recent research into techniques inspired by the operation of the innate immune system may provide a viable solution. In particular the DCA has been employed in a number of anomaly detection problems, particularly in the field of intrusion detection. Many of the problems which affect real-time embedded systems give rise to specific symptoms associated with that problem, and can therefore be regarded as anomaly-detection problems. For example, the failure of tasks to meet their deadlines gives rise to a clear system anomaly (a deadline overrun) which can be characterised and detected in the same way as an attack in an intrusion-detection system.

Therefore, these innate-immune inspired techniques may provide a viable solution to these problems encountered during the development of real-time embedded systems, for example by providing a mechanism by which real-time anomalies can be

detected or predicted, either as a standalone detection mechanism or else as part of a larger fault-tolerance system permitting the detection of and recovery from faults.

4.1 Thesis Hypothesis

The thesis will investigate the application of AIS to the development of real-time embedded systems, based on the following hypothesis:

> *Artificial immune systems can provide an accurate, responsive method to detect anomalies across a range of real-time embedded systems*

The aim behind this hypothesis is to devise an original method for enhancing the development of real-time embedded systems to be employed in non-safety-critical situations. The incorporation of anomaly detection into a system as part of the development process allows errors and potential problems to be flagged to the developers while the system is tested: these can then be rectified before the system is deployed.

The principles used for anomaly detection draw their inspiration from the natural immune system, specifically those of innate immunity. The function of the immune system is much debated amongst immunologists, but one theory suggests that the immune system is able to respond to the presence of danger, allowing it to detect and eliminate agents which cause that danger. These principles have been taken as the inspiration for algorithms which allow the detection of danger in computer systems, such as the DCA outlined in section 3.7.2.

Any technique to provide anomaly detection in RTES will need to be analysed according to a number of criteria, to ensure that it achieves the desired results, and that it is comparable with other techniques with similar aims.

- Correctness: any proposed solution must be able to successfully identify any anomalies present in a RTES – of all the metrics discussed here, the correctness of a solution is the most fundamental. When considering correctness, it is particularly important to evaluate the solution's overall accuracy, taking into account not just successful identifications, but also the presence of false positives, where the system incorrectly detects an anomaly where none is present, and false negatives where the system fails to detect an anomaly which is present. A solution which upholds the hypothesis will therefore be able to correctly identify anomalies present in a system.

- Responsiveness: particularly with solutions applied to in-service systems, it is important that the identification of any anomalies present in the system are identified quickly whenever they occur. A system which is slow to detect anomalies is potentially less effective than one which is highly responsive, especially at detecting transient anomalies where the period for which the fault is present is shorter than the response time.

- Flexibility: as a major problem with traditional real-time development techniques is that they are inflexible, an important goal of any improved development technique is that it can be applied to a variety of problems with little or no effort on the part of the system designer. This will allow the technique to be used with families of similar products, or in updated versions of products, without the need to perform large amounts of time-consuming analysis for each one.

- Scalability: for a solution to be successfully applied in real-world systems, it must be capable of being scalable to work with systems of the size and complexity of those being produced today. In addition, it should be able to cope with projected increases in system complexity, making the solution of continued use in the longer term.

- Resource efficiency: the need for a solution to make efficient use of resource is of significance when considering systems where resources are limited, as is the case with RTES in the consumer electronics domain. Any solution intended to be integrated into an in-service RTES must therefore be able to function effectively within the available system resources for it to be useful.

In analysing the solution against these properties, comparisons will be made against other potential solutions, both immune-inspired and traditional, allowing the overall effectiveness of the new approach to be accurately gauged.

5

Application of AIS to a real-time problem

In order to demonstrate that immune-inspired techniques can be applied in RTES, this chapter proposes a typical real-time problem, upon which an immune-inspired solution can be built and evaluated. The tools and simulations developed can then be evaluated further, to demonstrate the effectiveness of the solution in a variety of scenarios according to the hypothesis.

5.1 Formulation of problem area

There are a number of different problems which are commonplace in the RTES domain, including:

- Deadline overruns
- Bandwidth bottlenecks

- Deadlock/livelock

A method for reducing any of these problems would be highly beneficial to the RTES community. For the purposes of this work solutions for reducing the problem of deadline overruns will be examined in more detail. The presence of deadline overruns in a system is a typical example of a problem encountered by the designers of real-time systems, which can lead to undesirable behaviour or even total system failure if allowed to remain in the system.

5.2 Task scheduling

Task scheduling is a significant problem in the development of RTES, as the ability of tasks to meet their deadlines is important in guaranteeing the system's real-time properties and thus ensuring its reliability.

A variety of scheduling approaches are examined in the real-time literature, each of which has a differing range of properties and characteristics. For the purposes of this work, fixed-priority scheduling is used. The characteristics of this are such that static analysis can be readily applied for small problems, however the run-time ordering is determined dynamically, so giving rise to interesting properties during the course of system operation. In addition, fixed-priority scheduling is one of the scheduling strategies most frequently used in the industrial design of real-time systems [24]. This section explains the fixed-priority scheduling model, along with the static analysis techniques traditionally used for its verification. It also examines the model for deadline overruns and explains the quality attributes for a deadline overrun detection mechanism.

5.2.1 Fixed Priority Scheduling

Fixed-priority scheduling was initially examined in [104], in the context of a simple conceptual framework where the executing task at any point in time is always the

highest priority task which is runnable, such that the tasks execute in a pre-emptive manner.

A typical implementation of fixed-priority scheduling features three queues: a run queue, which contains tasks which are released but which have not yet commenced execution; a suspend queue, containing those tasks which have commenced execution but have then been suspended, for example due to pre-emption by a higher-priority task; finally the waiting queue contains all tasks which are neither released nor executing.

In [104], tasks are assigned priorities based on their periods, such that the task with the shortest period is that with the highest priority. This policy is known as rate-monotonic priority assignment, and has been proven optimal for task sets where each tasks deadline is equal to its period, and where there are zero offsets (where at some point in the execution of the system, there exists a point where all tasks are released simultaneously, known as a *critical instant*) [104].

5.2.2 Static Analysis of Fixed Priority Scheduling

The purpose of performing static analysis on a scheduling scheme is to determine formally, with the knowledge of the scheme's properties, whether it meets certain real-time requirements. A fundamental aspect of this is schedulability analysis, the purpose of which is to determine whether all tasks in a task set will meet their deadlines when a specific scheduling scheme is employed. For this purpose, there exist a number of schedulability tests which can be conducted. These tests fall into three categories, based on the accuracy of the test results:

- *Sufficient and Necessary* – The analysis always indicates correctly when a task set is schedulable. In addition, the analysis always indicates correctly when a task set is not schedulable.

- *Sufficient and Not Necessary* – The analysis always indicates correctly when a task set is schedulable. However, there are cases when the task set is schedulable contrary to the results of the analysis.

- *Not Sufficient* – The analysis indicates a schedulable solution when the task set is in fact not schedulable.

Clearly, the most desirable tests are those which fall into the *sufficient and necessary* category. However, in cases where no *sufficient and necessary* analysis technique is available, or where it is considered computationally infeasible, a *sufficient and not necessary* test may be regarded as acceptable. Tests which are *not sufficient* are undesirable as they give incorrect assurances regarding the schedulability of task sets.

The schedulability of a task set using fixed-priority scheduling can be verified by the application of a simple utilisation-based test [104] given in equation (3):

$$U_{\max} = \sum_{i=1}^{n} \frac{W_i}{T_i} \leq n(2^{\frac{1}{n}} - 1) \tag{3}$$

where: n *is the number of tasks*
i *is a task in the set of tasks*
W_i *is the worst-case execution of task* i
T_i *is the period of task* i
U_{max} *is the maximum processor utilisation*

A system is always schedulable provided that the combined utilisation of all tasks in that system is less than the value determined by equation (3), according to the number of tasks in the system. Consequently, if *n* is 1 and U_{max} is less than or equal to 100%, then the task set is schedulable. As *n* tends to infinity, then the maximum utilisation guaranteed to be schedulable, U_{max}, tends to 69.31%.

The results provided by this test, however, are pessimistic. An example of this is a task set with two tasks whose period and deadline are both equal to *T*, and with worst-case execution times of *T/2*. The test implies that the combined utilisation of these tasks (100%) is not schedulable since, according to equation (3), $U_{max} = 82.82\%$ where n=2. However, it can be demonstrated that this task set is schedulable: as the tasks are simultaneously released, one task is dispatched immediately and executes for time *T/2*. On its completion the second is dispatched and executes for a further time *T/2*, finishing at time *T*. Both tasks complete execution within their deadlines, and so the task set is schedulable. Consequently, this test can be considered to be *sufficient and not necessary*.

This pessimism was observed in [93], where it was also noted that rate-monotonic priority assignment is sub-optimal for task sets where task deadlines are not equal to task periods, or where tasks have non-zero offsets (i.e. a critical instant does not exist). The pessimism in this test indicates that a better schedulability analysis method is needed.

An alternative schedulability test was derived in [82], and is valid for task sets where there is a critical instant. The analysis assumes that all tasks have a fixed unique priority, and that the task deadline is not greater than the period.

For each individual task, the response time can be computed using equation (4) [82]:

$$R_i = W_i + I_i + B_i \tag{4}$$

where i *is a task in the set of tasks for a given node*
R_i *is the worst-case response time of task* i
W_i *is the worst-case execution time of task* i
B_i *is the blocking time of task* i
I_i *is the interference of task* i

In equation (4), the blocking time B_i is the longest time that a runnable task can be prevented from executing by a lower-priority task. This is dependent on the computational model being used. For an ideal pre-emptive system, this blocking time is zero. A non pre-emptive system, or a pre-emptive system where tasks access shared resources, will involve some degree of task blocking which must be accounted for.

When executing in a priority-based system, each task suffers interference from other tasks which are higher in priority than itself. For any task, the maximum interference is suffered at a critical instant, where all tasks higher in priority than itself are released simultaneously. The interference over a period of interest, (in this case the response time of the task being analysed), is derived from the higher priority tasks' periods and worst-case execution times, as shown in equation (5).

$$I_i = \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil W_j \qquad (5)$$

where hp(i) *is the set of higher priority tasks than task* i
R_i *is the worst-case response time of task* i
T_j *is the period of task* j
W_j *is the worst-case execution time of task* j

Equations (4) and (5) can be combined to form a recurrence relation as shown in equation (6), to give the worst-case response time for each task in the system:

$$R_i^{n+1} = W_i + B_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i^n}{T_j} \right\rceil W_i \qquad (6)$$

with $R_i^0 = W_i$
which terminates when $R_i^{n+1`} = R_i^n$, *or* $R_i^{n+1} > D_i$
where D_i *is the deadline of task* i

If this recurrence successfully indicates that, for each task in the system, $R_i < D_i$, then the task set is schedulable. This test has been found to be *sufficient and necessary* providing the task set meets the assumptions outlined above.

5.2.3 Deadline Overrun Model

In general, work on scheduling and timing analysis only considers systems which meet their timing requirements, or providing tests to ensure that requirements are met. Although some papers consider soft real-time requirements, such as [25] this is generally only in the context of providing best effort scheduling. There appears to be little work in the RTS literature which attempts to formalise an understanding of what happens when tasks fail to meet their requirements.

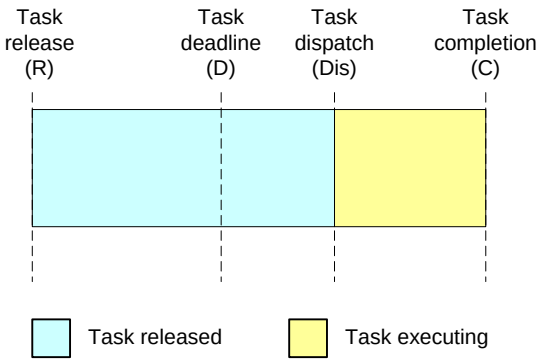


Figure 5.1: Typical Deadline Overrun

Figure 5.1 shows a typical deadline overrun, with four variables. The task's deadline is denoted by *D*. *C* indicated the time at which the task completes: in the case of a deadline overrun, the value of *C* will be greater than *D*. The value R represents the time at which the task is released, i.e. the point at which it is transferred into the run queue from the waiting queue. Finally, the task dispatch time, shown as *Dis*, is the time at which the task begins execution following its release. Although in Figure 5.1, *R* and *Dis* are shown as occurring before and after the deadline *D* respectively, these points can in fact occur at any point in time providing that $R \leq Dis$, and therefore either can be less than, equal to or greater than *D*.

One key property of real-world systems is that execution times of tasks are not constant, caused by differences in task behaviour between cycles, and often also by external factors such as user interaction. As a result, the times at which tasks are dispatched and completed relative to their release and deadlines will vary between execution instances. This has a number of implications for this work.

The likelihood that a task will overrun varies between the task's release and its completion. It is quite possible that, although the potential for an overrun to occur is identified at some point during a task's execution cycle, that the task will not actually overrun. In some cases, this is due to errors in the overrun prediction model, or with the information supplied to it by the system. Generally, however, the likelihood of an overrun is reduced as other tasks in the system complete before their worst-case time. Conversely, the release of higher priority tasks into the system during a task's execution cycle may result in an overrun being predicted where previously no risk had been identified. Frequently, these two behaviours will combine, such that the likelihood of a task overrunning is low on its release, increases as other higher priority tasks are released into the system, and decreases again as those tasks complete ahead of their worst-case times.

In addition, in non pre-emptive systems it is possible that overruns can occur when tasks execute for durations less than their worst-case execution times – this is a form of timing anomaly [67]. An example of this is shown in Figure 5.2. In the first scenario on the left, task A completes after 4 clock cycles, permitting the execution of task C. When task B is released in cycle 5 it is then blocked from executing until task C has completed at cycle 7. If the deadline of task B is 7, the task misses its deadline.

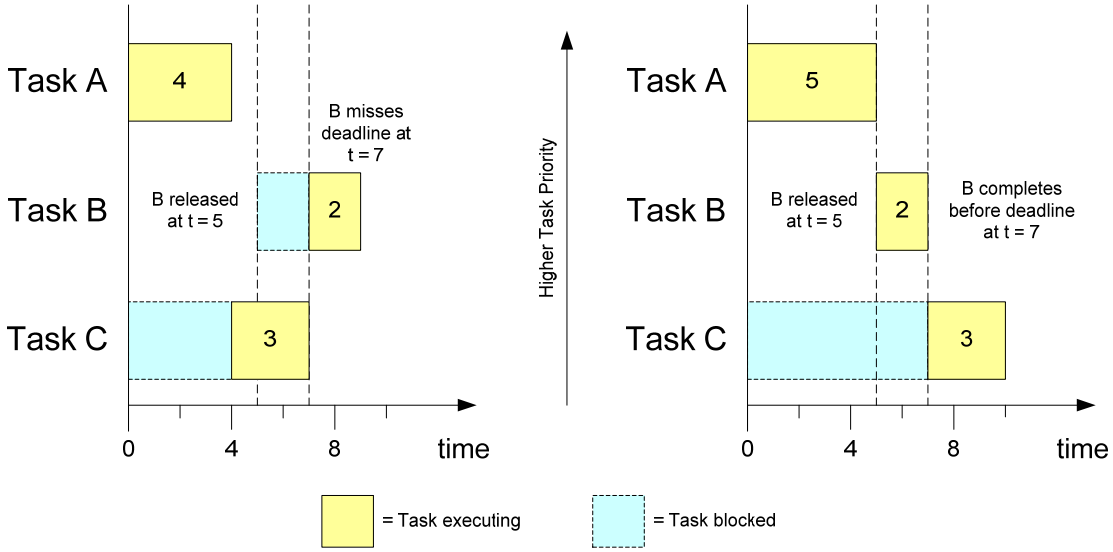


Figure 5.2: Overrun caused by task executing for less than worst case time

However, in the second scenario, task A executes for 5 cycles. By this point, task B has already been released, and so it is the next to execute after task A completes. The execution of task B is then completed before its deadline at cycle 7.

This complexity makes the analysis and detection of overruns particularly difficult. In a small system, it is possible to exhaustively search through all combinations of task properties to establish all the possible conditions which lead to overruns. However, this quickly becomes infeasible as the complexity of the system increases.

5.3 Using AIS to solve deadline overruns

As established, there are a variety of techniques already in existence which can be employed to prove the schedulability of a system *a priori* (and, therefore, the absence of deadline overruns). However, these techniques are not suitable for many classes of systems, particularly those in the CE industry where cost and development time are significant issues. Schedulability analysis *a priori* assumes the availability of detailed knowledge of various system characteristics – information such as worst-case execution times of tasks. In many cases this knowledge is difficult to obtain, and for systems with many different tasks or modes of operation the required analysis is complex and time-consuming. The results of any analysis obtained are specific to the system as analysed – any alterations to the system, for example through the

connection of additional hardware or the desire to run third-party software applications, cannot be considered in such analysis.

In order to solve the problem of deadline overruns, a monitoring system is envisaged which employs a closed feedback loop, to maintain the system in a state where there are no deadline overruns:

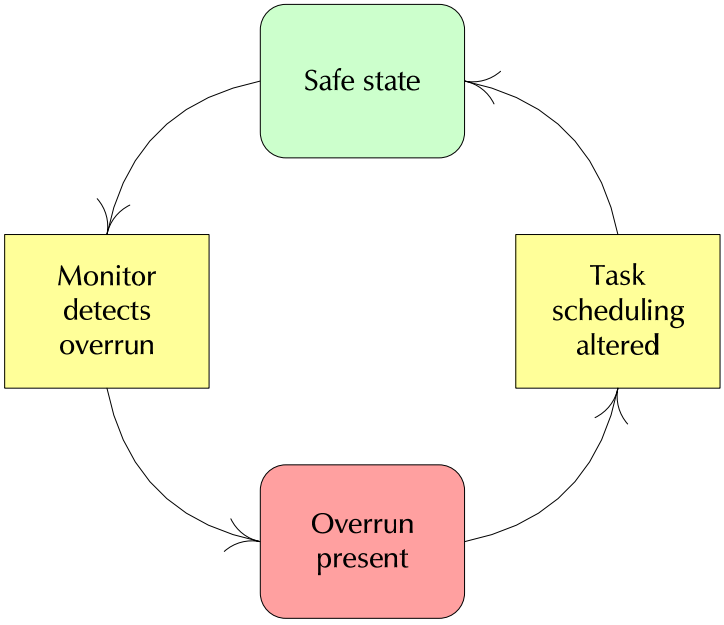


Figure 5.3: Closed loop monitoring

In such a system, there are two major functions:

- A monitoring function, which continually examines the state of the system, in order to detect situations in which an overrun may occur

- A fault prevention function which uses this information to alter the execution characteristics of the system in order to return the system to a safe state

An ideal solution to the problem of deadline overruns, therefore, will be able to monitor the system and detect when a task may overrun. When such a situation

occurs it will be able to alter the scheduling of the system, returning the system to a safe state.

It is important that an ideal solution should be able to detect the presence of overruns with no a priori knowledge of the system in which it is incorporated, ensuring that time-consuming system analysis is not necessary and allowing the solution to be as generic as possible. In addition, it is highly desirable that a solution is able to adapt to changing system characteristics – for example, the addition of new tasks to the system, or a change in execution characteristics of a particular task or set of tasks.

Previous work involving the development of CE devices has either made use of established real-time techniques to guarantee reliability, or else has focused on the software architecture aspects of the development process and not considered the real-time aspects in great detail. As far as can be established, this work is the first application of a "non-standard", biologically-inspired technique as a solution to a traditional real-time problem. In addition, this is one of only few applications of an immune-inspired technique which has been required to produce results in real-time. Although similar methods have been employed successfully in a variety of anomaly detection schemes, and have been applied to certain types of scheduling problems, the use of immune-inspired techniques in real-time anomaly detection has not been significantly explored.

Given the unproven nature of biologically-inspired methods in the RTES domain, and the lack of prior implementations of AIS in real-time scenarios, this work initially makes use of a small example in which the real-time aspects are necessarily basic, however despite the apparently simple nature of the problem the conditions under which an overrun will occur are not always easy to determine. The use of simple problems allows the underlying issues to be easily understood, affording the opportunity to compare the effectiveness of immune-inspired solutions against common RTES techniques. Although it does not necessarily hold that a solution to a simple problem will scale up to more complex situations without problems, a good

solution should provide a solid basis on which the solutions to more complex problems can be built.

5.4 Immune methods: RTES considerations

In order to satisfy the hypothesis (see section 4.1), artificial immune techniques must be employed in a device with real-time and embedded characteristics. As established in chapter 1, embedded computer systems are likely to be subject to resource constraints when compared with non-embedded computer systems. In the case of CE devices, there is a particular need to keep the required resources to a minimum, as small increases in unit cost add up to considerable amounts when multiplied over the many thousands or even millions of devices which may be produced.

In RTES, and particularly in CE devices, it is likely that the resource overheads associated with adaptive immune techniques cannot be justified, even though the characteristics exhibited by the adaptive immune system would be desirable in RTES. It is therefore necessary to consider AIS methods derived from the innate immune system, which are unable to adapt to changes in environment but generally require fewer resources than a fully adaptive system and still offer effective solutions.

As detailed in section 3.7.2, the DCA is an innate immune concept derived from innate immunity and the danger model, which has already been applied to anomaly detection [114], [154]. The concepts of the danger model map particularly well onto problems such as the deadline overrun problem: the presence of potential overruns in the system being equivalent to the presence of danger in a living organism. In the context of the deadline overrun problem, therefore, the DCA appears to be particularly suitable.

However, in the context of RTES, the most significant advantage of the DCA is that, particularly when compared with other immune-inspired algorithms, its resource requirements are generally low. Although the algorithm makes use of a population of DCs, it is not "population-based" in the same sense that other algorithms such as

GAs are: the population size does not fluctuate as the algorithm runs, and there is no requirement for affinity calculations to be performed across the entire population.

Of course, methods derived from innate immunity lack the ability to adapt to changes in their environment. Although in a number of applications it is desirable that there should be some ability to respond to previously unseen behaviour, this is unlikely to be the case with the deadline overrun problem, as the characteristics which lead to the occurrence of overruns are well understood and documented.

When considering implementation in a resource-constrained environment as encountered in RTES, the apparently low resource usage of the DCA is a notable advantage.

5.5 Applying the DCA to the deadline overrun problem

In order to test the effectiveness of the DCA in RTES, it is applied to the deadline overrun problem. The initial aim is to establish the mappings between the problem and the DCA components, and to determine whether the DCA can be effective in the detection of overruns in a simple task-based system.

Although it would be desirable to develop a complete monitoring system, at this stage, the primary goal is the successful identification of overruns, rather than on deriving a method to mitigate their effects. There are a variety of methods which can be used to reschedule systems, but most of these are reliant on the output of a detection scheme such as that provided by the DCA. There is also scope for the anomaly detection element of such a system to be employed in its own right, either as part of the development process incorporated in some form of external monitoring device which can be connected to a prototype system, or simply to gather data in a finished system which can then be fed back to the developers, allowing problems to be investigated at a later point in time.

The problem of tasks overrunning deadlines is well known amongst the RTS community, and there already exist a number of methods by which a running system can indicate that it has missed a deadline, such as the raising and handling of exceptions. However, the RTS community lacks a method of predicting the potential for overruns to occur *a priori* without the use of static analysis techniques such as schedulability analysis. Therefore, an important feature of any new solution to the deadline overrun problem is the ability to predict overruns before they occur, or to highlight the situations where, had the system run to worst-case, an overrun would have occurred. The ability to predict potential overruns in this way provides scope for the situation to be remedied before that task actually suffers an overrun, or allows the system designers to be made aware of which parts of the system may experience problems.

5.5.1 Task simulation environment

To simplify the application of the DCA to task scheduling the experiments will be conducted in a simulation environment, allowing for detailed monitoring of the entire task scheduling process to take place, and also guaranteeing repeatability should it be necessary. Mechanisms for such detailed monitoring are difficult or impossible to achieve in a real system without the use of expensive equipment, and due to the architecture of many processors which include caches and advanced branch-prediction environments, it is difficult to ensure a consistent and repeatable execution environment.

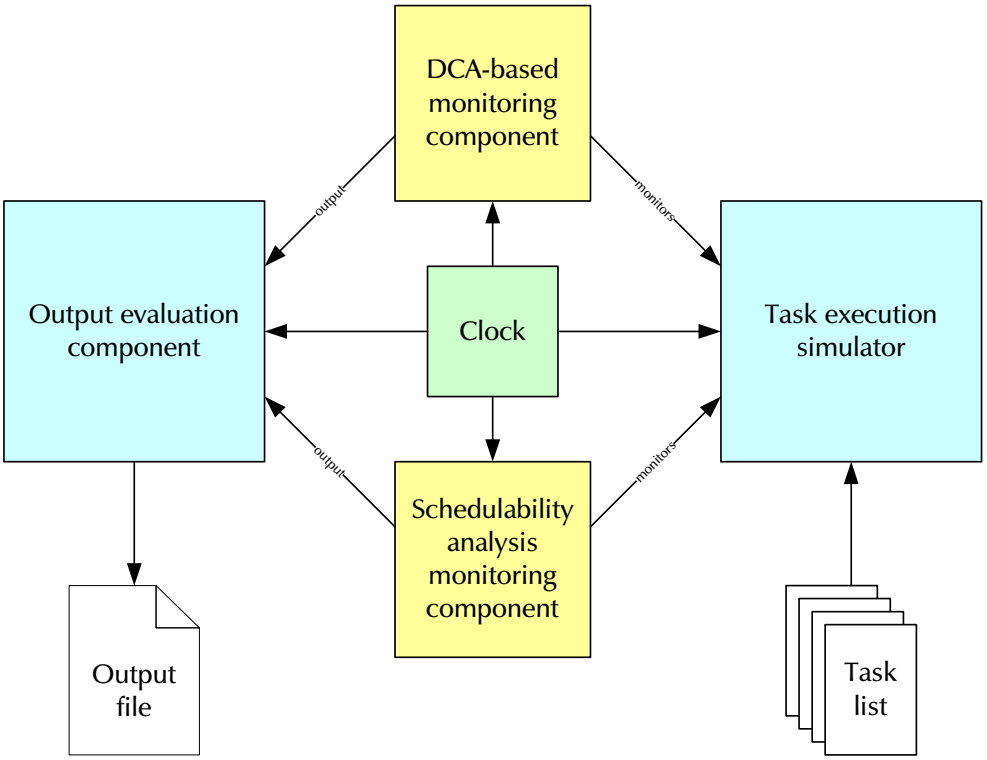


Figure 5.4: Simulation environment - architectural block diagram

The architecture of the simulation environment is shown in Figure 5.4, and is made up of a number of components. The primary component is a task execution simulator. This takes as input a set of task parameters, and simulates the execution of those tasks according to a defined scheduling policy – a set of rules which controls the order in which tasks are executed. It is possible to vary the scheduling policy used by the scheduler, to allow the use of a variety of different scheduling policies. For the purposes of this work, however, the scheduler is run according to a "fixed priority" policy, as outlined in section 5.2.1. In order to create a higher incidence of overruns in the system the execution of tasks is conducted in a non-preemptive manner, such that once a task begins executing, it will not be interrupted before its completion. The use of non-preemptive scheduling causes anomalies such as that shown in Figure 5.2 to occur, increasing the likelihood of overruns in situations other than the worst-case.

The task execution simulator includes functions which allow it to pass information about the state of the simulation to a number of monitoring modules. These modules

can be used to implement different monitoring methods in parallel, allowing comparisons to be made as to the effectiveness of the different methods.

Two monitoring components are employed in this work. The first is an implementation of the DCA, which is executed in parallel with the task scheduler, allowing a population of DCs to monitor the properties of the tasks as they are released, dispatched and executed by the scheduler. The operation of the DCA component is explained in more detail in section 5.5.2. The DCA input signals, which are described in more detail in section 5.5.2.2, are derived through interrogation of the task execution state from the task scheduler.

In addition to the DCA monitoring component, an online schedulability analysis monitor is provided, which runs in parallel with both the task simulator and the DCA. This schedulability analysis makes use of the static analysis techniques described in 5.2.2, and is intended to function as a baseline against which the effectiveness of the DCA-based monitoring solution can be evaluated. Given that the schedulability analysis methods used are guaranteed to detect all instances of overruns within a system, it can be assumed that the static analysis monitor will act as a perfect predictor, and the effectiveness of any other technique can therefore be evaluated by comparison.

The whole simulation environment is controlled by a clock component, which keeps time in arbitrary units referred to as “simulation cycles” and is able to pass information about the current simulation time to all other components in the system. Task properties, such as periods and deadlines, are specified in these units, and the simulation runs with a granularity of a single simulation cycle.

Although it is not computationally feasible to run such an analysis in a real system, the use of the “simulation cycle” as the a fundamental unit of time means that it is possible to make use of these methods, as the actual amount of time taken to execute an arbitrary simulation cycle can be extended to allow the completion of complex

evaluations, whereas in a real system the maximum complexity would be constrained by the number of processor cycles available between two consecutive evaluations.

From the perspective of evaluating the effectiveness of the DCA, the fact that the DCA is not run in simulated time allows a broader range of parameters to be investigated, as there is no upper bound on the complexity of an evaluation which can take place, other than the length of the overall simulation run itself.

The output of all the monitoring methods incorporated into the simulation environment is directed into an output aggregation component, the purpose of which is to compare the operation of the various methods for the purposes of calculating their effectiveness. The output of this component is directed to a number of output log files which can be further analysed across a number of simulation runs to give the final output results either in numerical or graphical formats.

5.5.2 DCA operation

The DCA applied to the deadline overrun problem is based on the algorithm first outlined in [74], as discussed in 3.7.2 and defined by the pseudocode in Figure 3.7. Due to the differences between the intrusion detection problems on which Greensmith's work is focussed, and the class of real-time embedded systems which are to be investigated here, there are a number of alterations which need to be made to the algorithm outlined in [74] to make it suitable for application to the deadline overrun problem.

In the majority of applications detailed in the literature, the DCA has been employed in a batch-processing mode working on stored sample data. For application to the deadline overrun problem, it is necessary that the DCA be executed in parallel with the system which it is monitoring, such that it is able to predict or detect overruns as the system operates. Therefore, the operation of the algorithm must be modified slightly from the original in order to take the desired output mode into account.

Therefore, in order to allow continuous execution in real-time, the algorithm is altered to operate according to the pseudocode shown in Figure 5.5.

```
initialise cell population;
for each clock cycle loop
  select x DCs from DC population

  for each task in run queue loop
    compute input signals;
    for each selected cell loop
      update signal matrix;
      update antigen matrix;
    end loop;
  end loop;

  for each selected cell loop
    increment cycle count;
    compute output signal;
    if output signal > migration threshold then
      register danger output;
      reinitialise cell;
    else if cycle count >= lifecycle threshold then
      register safe output;
      reinitialise cell;
    end if;
  end loop;

end loop;
```

Figure 5.5: DCA pseudocode, amended for application to the deadline overrun problem

The key difference between the two applications are that, in the version applied to the deadline overrun problem, once a DC has entered the mature or semi-mature state, it signals the according output and is then instantly reinitialised and returned to the population, rather than the analysis being held until the entire population has been allowed to either become mature or semi-mature. This method allows for continuous overrun detection coverage as the task simulation runs, rather than subdividing the simulation cycle space into a number of separately defined windows each of which is analysed individually. This is an important consideration when applied to the deadline overrun problem, as overruns may only exist in the system for a short amount of time which may be considerably smaller than any time window used, however it is important that the algorithm is able to detect and signal these overruns as they occur.

As applied to the deadline overrun problem, the core time unit around which the algorithm's main cycle is based is the arbitrary "simulation cycle" as described in section 5.5.1. Each cycle, the DCA monitoring component selects a random pool of DCs from the overall pool which are to be evaluated in that clock cycle. It then interrogates the task scheduler, and for each task currently in the run queue (i.e. those tasks which are released or executing at that point in time), it determines the values for the DC input signals based on the state of the task at that point in the simulation, and updates the signal and antigen matrices of each selected DC. The exact derivation of the antigen and input signals will be explained in sections 5.5.2.1 and 5.5.2.2.

Once the input signals have been processed for all the tasks in the run queue, the output signals for all the selected DCs are then computed, and the lifecycle counter for each selected DC is incremented. For each DC, the output signal value is compared against the DC's maturation threshold, and the DC becomes mature if the output signal value is greater than the maturation threshold. If the lifecycle counter is greater than the lifecycle threshold, the DC becomes semi-mature.

Once a DC has reached maturity or semi-maturity, its antigen is evaluated, and the DC is reset and returned to the population. The random selection of a subset of DCs each cycle, together with the non-deterministic characteristics of the task execution, ensures a continuous evaluation of the status of the run queue throughout the simulation.

5.5.2.1 Antigen

In addition to its output signal, each DC also provides a list of associated antigen on reaching maturity or semi-maturity. The antigen in this implementation represents the tasks examined by that particular DC. By correlating the output signals of a number of DCs with the antigen observed by those DCs, it is possible to deduce exactly where in the system problems are occurring.

According to the DCA as outlined in section 3.7.2, the assignment of antigen to DCs occurs each time a DC is evaluated. In this implementation, the antigen assigned to a DC is simply the contents of the run queue at the point when that DC is evaluated. As a DC undergoes multiple evaluation cycles, it builds up a library of all the tasks which it has observed during its maturation phase. Upon reaching maturity or semi-maturity, this information is evaluated along with the DC's output signal.

5.5.2.2 Signals

The choice of input signals is fundamental to the operation of the DCA. In its canonical form, the DCA supports multiple categories of input signal, and multiple individual signals within each category. The input signals are assigned values, which are derived from easily quantifiable characteristics of the system. Although in theory there is no reason why a DC's input signals should not be real-valued (indeed, in the biological system from which the DCA is derived, a higher concentration of cytokines often leads to a greater level of immune response), in the majority of applications to date DC input signals are limited to a Boolean value for simplicity. Likewise the output of each DC takes the form of a binary signal, indicating the presence or absence of "danger" in the system.

For application to the deadline overrun problem, the DCA is applied with three signal categories, which can be considered analogous to the PAMP, danger and safe categories used in previous applications, each of which consists of one signal. These signals, shown in Table 5.1 are derived from measures based on the response-time analysis methods detailed in section 5.2.2.

Table 5.1: Derivation of DC input signals

Event	Signal category	Derivation
Actual overrun	PAMP	Task completion time > task deadline (Ci > Di)
Potential overrun	Danger	At any point from task release to completion, worst-case response time is greater than time to deadline (Ri > Di)
No projected overrun	Safe	At all points from task release to completion, worst case response time is less than time to deadline (Ri < Di)

5.5.2.3 Weights and thresholds

The operation of the DCA relies on the combination of the input signals as determined by the set of weighting values, in order to produce an overall output value. Consequently, the choice of these weighting values will have a significant effect on the overall output of the algorithm. The output of a DC is determined by the comparison of its output signal value with the migration threshold: if the signal value is greater than the threshold value, then the DCA becomes mature as detailed in section 3.7.2. In addition, each DC also has a value determining the maximum number of DC cycles it will undergo: if a DC reaches this number of cycles without becoming mature, then it is considered to have reached a state of semi-maturity.

The choice of threshold and weighting values, and the relationship between them, has a significant bearing on how the algorithm behaves. By choosing threshold values which are similar or lower than the weighting values for a particular signal, the algorithm can be engineered for rapid response to a particular condition. However by choosing threshold values significantly higher than any weighting value, the algorithm will instead respond to sustained levels of danger over a period of time. It is however likely that too high a level of sensitivity will lead to a greater incidence of false positives, while a sensitivity level which is too low will result in a higher number of false negatives, particularly where overrun conditions are short-lived, for example where a task is only active for a short amount of time.

Because of this, two sets of weight and threshold values are used: one where the signal weighting values are higher than the output threshold, giving a high level of sensitivity, and a second where the threshold value is higher than the signal weighting producing a lower level of sensitivity. The exact values of these weights and thresholds will be given later in this chapter.

As stated previously, in the context of the deadline overrun problem a key area of interest is the prediction of overruns before they occur. Therefore, it can be considered that the danger signal (which signifies the presence of a potential overrun in the system) is of at least equal importance to the PAMP signal. The weighting values have been chosen such that the presence of either PAMP or danger signals will result in the same level of DCA response.

As detailed in section 3.7.2, the DCA has a large number of parameters which affect its operation. Initially, to determine the applicability of the DCA to RTES problems in its current state, the DCA is employed in a manner similar to how it has been applied to intrusion detection in [74] and [70], with the set of parameters fixed for the given scenario. The use of a fixed algorithm over a wide variety of task sets allows the general effectiveness of the algorithm to be established over a range of problems which are all similar, yet due to differing characteristics may behave very differently from each other. From the results obtained, potentially interesting task sets can be identified as candidates for further experimentation and analysis.

5.5.3 DCA Effectiveness

In order to analyse the effectiveness of the DCA in this context, its output is compared with that of a schedulability analysis as detailed in section 5.2.2, which for the purposes of investigation is executed in parallel with the DCA within the task simulation framework. This schedulability analysis is provided with all details of the task parameters, and can therefore be considered to be both a perfect predictor and the earliest possible predictor. The output of this analysis can therefore be used as a benchmark against which the effectiveness of any other solution can be compared.

The purpose of the schedulability analysis is solely to allow the effectiveness of the DCA-based solution to be established: in an actual implementation, the schedulability analysis would not be carried out.

The effectiveness of the DCA is here considered in terms of two measures: the accuracy and responsiveness, both calculated by comparison with the output of the schedulability analysis. Accuracy is chosen due to the importance of the solution being able to predict overruns correctly. In addition to the correctness, however, it is important that the solution is as responsive as possible: earlier prediction of an overrun gives any corrective system greater time in which to prevent that overrun from occurring.

5.5.3.1 Effectiveness measure: Accuracy

For the purposes of calculating the accuracy of the DCA-based solution, each individual execution is considered, as determined by the task simulation.

Figure 5.6 shows the three possible modes of execution of a task, from the point at which the task is released (i.e. the start of its execution) at the point labelled R, through its dispatch point (where it begins executing) at DS, its completion at C, to its deadline DL. It is assumed that an online schedulability analysis will detect a potential overrun at the earliest possible opportunity, (point O). This point is therefore assumed to be the earliest that a DCA-based solution can correctly identify the presence of a potential overrun in the system.

For the DCA to correctly predict an overrun, at least one DC must report the presence of danger in the system at some point in the time window between O and the point C where the task completes. In the case of a task which successfully completes before its deadline, as in Figure 5.6a, a report of danger in this window indicates successful prediction of an overrun which, although possible, did not actually occur. These overruns are considered to be “potential overruns”.

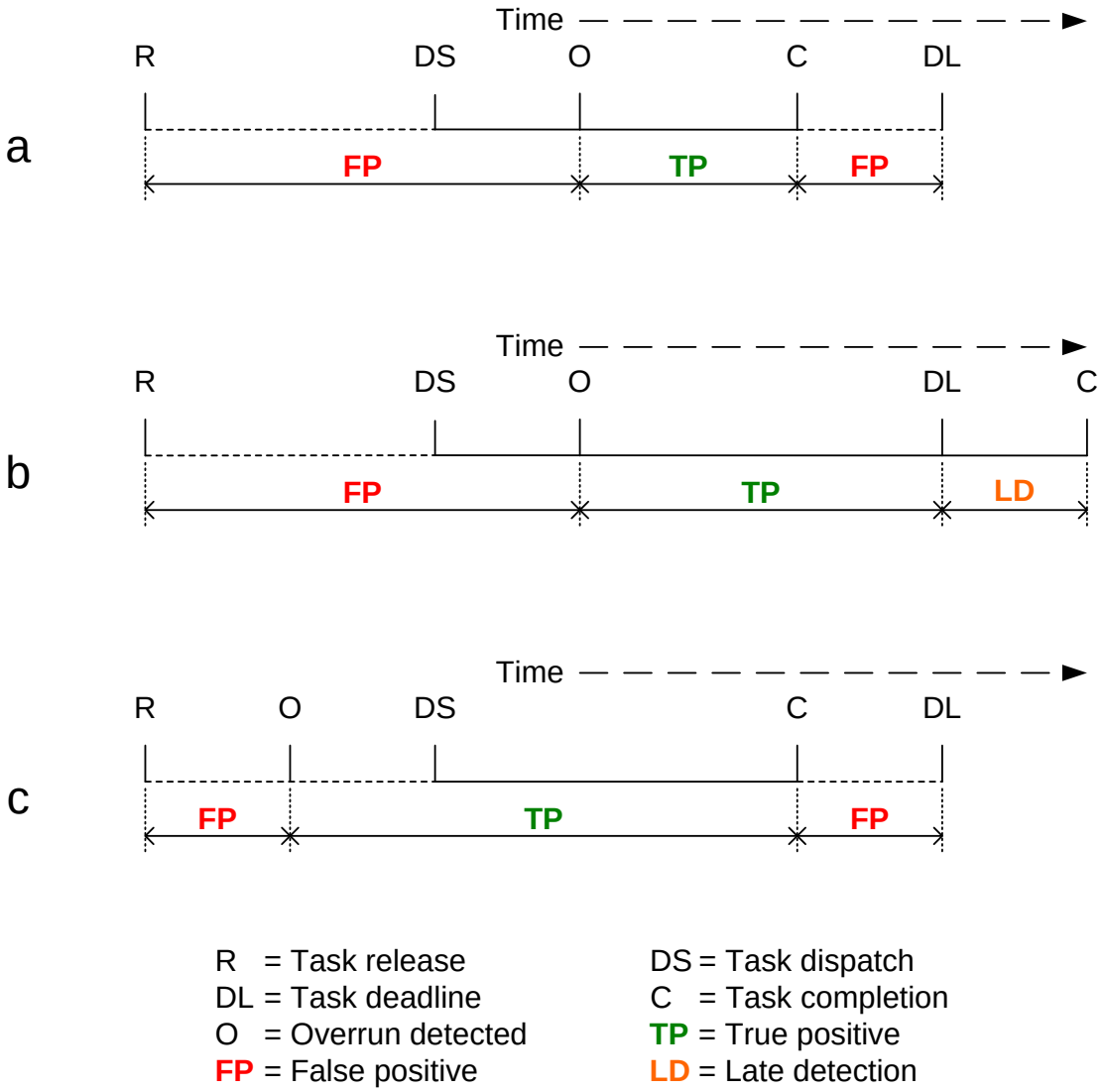


Figure 5.6: Determining accuracy of DCA-based solution

Where the task overruns, its deadline DL occurs before the completion of its execution (Figure 5.6b). Any report of danger in the time period between O and DL indicates successful prediction of an overrun which then did occur: overruns in this class are referred to as "actual overruns". Prediction of a potential or actual overrun is regarded as a true positive.

A report between DL and C indicates successful *detection* of an overrun which has already occurred by the time the detection occurs. This situation is considered to be a late detection. Although a late detection can be considered correct, and therefore not a false positive, the detection of overruns after the fact is something which can already be accomplished in real-time systems using well-established exception

raising and handling methods. When establishing the effectiveness of the DCA, the prediction of overruns before their occurrence is the primary objective, as the prediction of overruns is not something which can be easily achieved in real-time. For the purposes of the evaluation in this chapter, incidences where late detection occur are classified as true positives; however the evaluation mechanism incorporates the ability to classify them as false negatives should there be a desire to place additional emphasis on the prediction rather than the detection of overruns in the evaluation.

It is possible that, if the schedulability analysis is able to allow for the effects of tasks which have not yet been released, that the point SA may occur before the task dispatch point DS. This does not affect the correctness of the results, which are still determined by the timing of point O rather than whether the task is actually executing. This shown in Figure 5.6c.

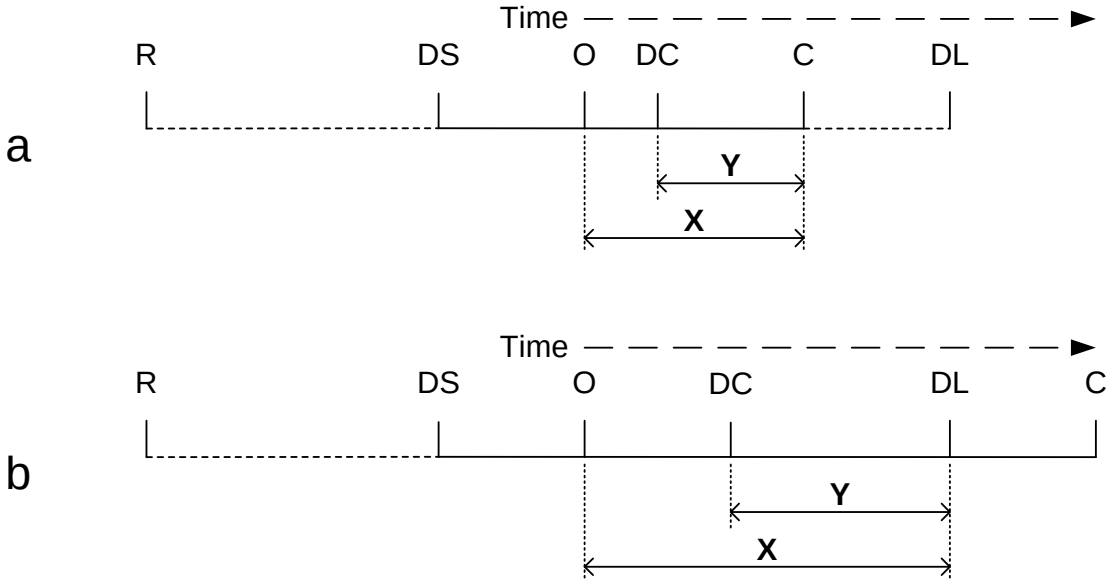
In situations where the schedulability analysis detects the presence of an overrun, but no DC reports the presence of danger, then the DCA is presumed to have failed to detect the overrun, regardless of whether that overrun went on to occur or not. Such an instance is considered to be a false negative. Situations where a DC reports danger at any point where the schedulability analysis has not detected the presence of an overrun are considered to be false positives. A true negative occurs when a task executes with neither the schedulability analysis detecting the presence of an overrun, nor any DC reporting danger at any point during its execution.

The simulator records these statistics throughout each simulation. From these, the true positive rate, true negative rate and the overall accuracy of the system can then be calculated.

5.5.3.2 Effectiveness measure: Responsiveness

We consider the responsiveness of the DCA-based solution for all task execution instances where the schedulability analysis indicates the presence of an overrun in the system.

Figure 5.7a shows the execution of a task, where the schedulability analysis has determined that an overrun is present at point O. As previously mentioned, the task has deadline DL and completion time C. The point DC indicates the first report of danger from any DC. This point is the earliest response to danger by the DCA.



R = Task release
DS = Task dispatch
DL = Task deadline
C = Task completion
O = Overrun detected
DC = First DC danger report

Figure 5.7: Calculating responsiveness of DCA-based solution

The responsiveness is calculated by first recording the time values for O, DC and C (the task simulation tool makes use of arbitrary simulation cycles rather than in actual units of time). The responsiveness is given by equation (7):

$$Rsp = \frac{C - DC}{C - O} \quad (7)$$

This responsiveness of the DCA-based solution is expressed as a proportion of the responsiveness of the schedulability analysis, with the assumption as already stated that the schedulability analysis reports at the earliest possible time. The durations used for the calculation of the responsiveness can be seen in Figure 5.7, Y being equal to C – DC and X being equal to C – O. The value for responsiveness for any valid task instance must lie between 0 and 1, where 0 is the least responsive and 1 the most. For any execution instance where the DCA-based solution responds at the same time as the schedulability analysis, the responsiveness is 1. Where there is no DCA response at all (i.e. the accuracy for that instance is 0) the responsiveness value is assumed to be 0.

Where the task actually overruns, such that the task deadline DL comes before the point of completion C, the responsiveness is calculated in terms of the deadline rather than the completion time. This is shown in Figure 5.7b. Consequently, only successful prediction of overruns has a responsiveness greater than 0.

5.5.4 Task set classification measures

As outlined in section 5.2.2, there are a huge variety of different measures from which the status of a particular task can be derived. For the DCA to be effective it is important to consider which of these measures should be used, and how they will be mapped onto the input signals for each individual DC. Some of these measurements relate to the system as a whole, while others are linked only to individual tasks.

System-wide measurements include the total utilisation of all currently-running tasks in the system, giving an indication of the system's workload. Although it is possible for systems to function correctly in overload situations, it is inevitable that a system with a worst-case utilisation level greater than 100% will eventually

experience problems. Utilisation is a useful measure of the overall health of a system, but it can be difficult to attribute system failures to any one task.

When considering execution properties of individual tasks, a useful concept is that of slack time. This is the interval between the completion of a task and its deadline. Due to it not always being possible to determine the execution time of a task *a priori*, it is also impossible to determine the actual slack time for any given execution of a task until that task is completed. However, by making use of worst-case properties, it is possible to calculate the minimum possible slack time at any point in a task's execution, by comparing the task's deadline with the worst-case remaining execution time. If this worst-case slack time is less than zero, then the task will complete before its deadline. A worst-case slack time equal to zero indicates that, at worst, the task will complete at the same time as its deadline. If at any point during the execution cycle the worst-case slack time is negative, there is the possibility that the task may overrun. The value given by the calculation is the worst-case scenario: due to tasks having variable execution times and not always executing to their worst-case times, it is still possible that the task may complete before its deadline.

Although a task's slack time is a useful measure, it is not particularly intuitive: the greater the value for a task's slack, the earlier it completes, and negative values of slack indicate that the task will overrun. In many cases it is preferable to think in terms of overrun time: this is simply the opposite of the slack time. A process which has a positive overrun time is one which does not complete before its deadline; a negative overrun time indicates successful completion before the deadline.

5.5.5 Learning system behaviour

One of the most significant problems with existing real-time techniques is that they require specific knowledge of the characteristics of the system they are applied to. To ensure that this solution is generally applicable to a large number of problems, it is important that the DCA should not require any prior knowledge of the system. This is accomplished by including support for the DCA components to learn the execution

properties of each individual task as the system is running. These parameters learned are used by the DCA in deriving the input signals.

Incorporating learning behaviour also allows the DCA solution to go some way towards supporting systems with dynamic run-time properties, for example where additional tasks are added to the system, or where the task properties change as the system runs. Given the increasing complexity of many classes of RTES, it is important for any devised technique to be able to support such systems.

In order that the learning behaviour does not affect the results while it is learning the system properties, a configurable initial learning period is included: for the specified number of cycles after the simulation starts, the system characteristics will be learned. The DCA will then be initialised with the learned information and commence reporting, and the learned information continues to be refined as the execution commences. The simulation allows the learning behaviour to be overridden if necessary, bypassing the learning phase and initialising the DCA with the worst-case parameters: this is useful if repeatability is desired.

By necessity, the online schedulability analysis against which the DCA is compared must be provided with full details of the task set properties. This is possible as the task sets used by the simulator are generated with known properties prior to the start of the simulation.

5.6 Evaluation of DCA-based overrun prediction

DCA-based solution is run in conjunction with the task simulator with a number of task sets of varying characteristics. From the results obtained, the accuracy and responsiveness for each different task set can be calculated.

5.6.1 Initial testing – engineered four-task example

In order to test its applicability to this problem, the DCA is applied to a set of four purely periodic tasks. The properties of these tasks, shown in Table 5.2, have been engineered such that in most instances they complete normally, but that the relatively infrequent release of a task with a long execution time (task 4) can, in certain circumstances, cause other tasks in the system to overrun.

Table 5.2: Task properties of four-task example

Task ID	WCET	BCET	Deadline	Period
1	5	4	25	25
2	20	10	50	50
3	30	20	100	100
4	40	25	775	775

The actual occurrence of an overrun is determined by a number of additional factors, such as the execution times and release rates of other tasks. Figure 5.8 shows three possible execution patterns for this task set following a critical instant (where the critical instant occurs at time T=0). In scenario a, all four tasks are assumed to execute for their worst-case values. The execution of task 4 and the non pre-emptive scheduling method used leads to task 1 missing its deadline at time T=125. In scenario b, task 3 is assumed to execute only for its best-case execution time, and all other tasks for their worst-case values. In this scenario, the ordering of the tasks and their phasing relative to each other permits task 4 to complete execution without any of the four tasks experiencing an overrun.

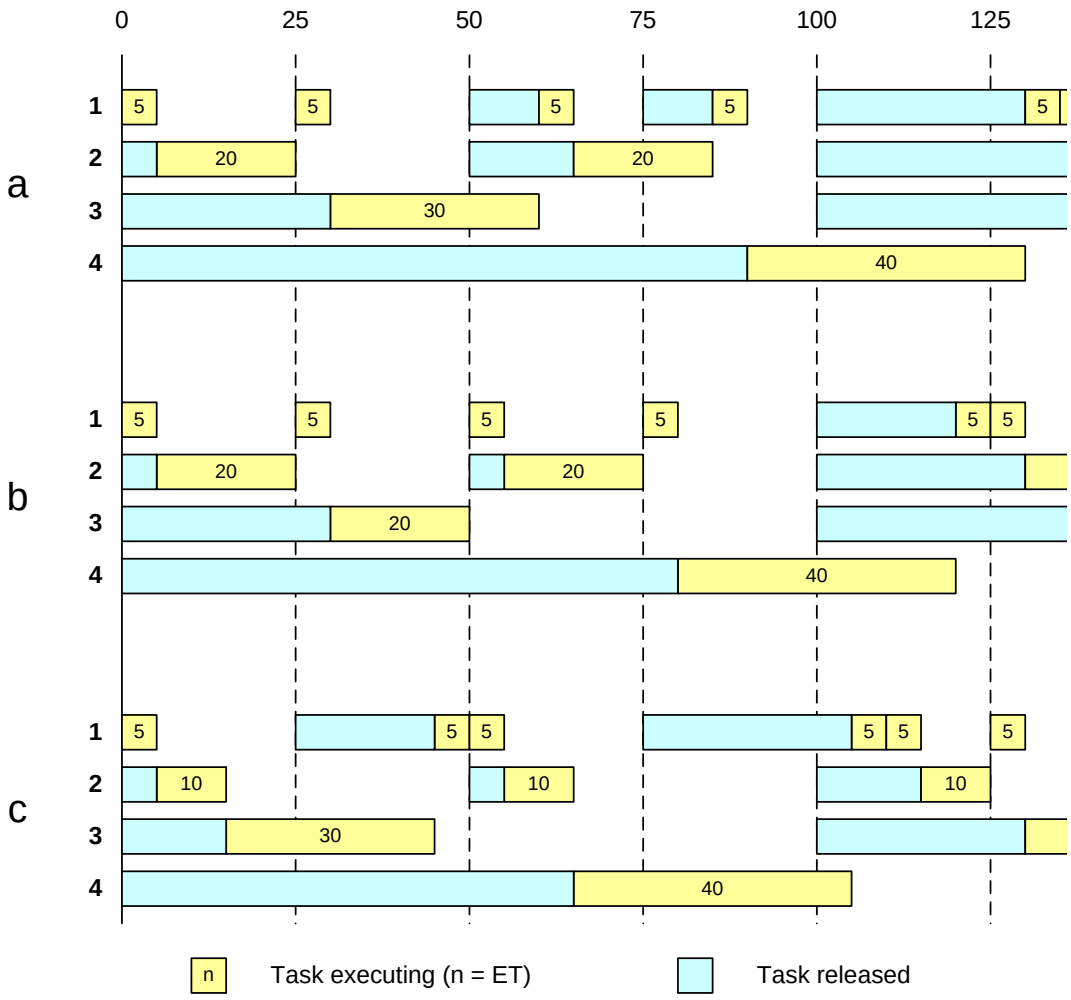


Figure 5.8: Example execution timelines for four-task example

However, in scenario c, task 2 now executes for its best-case time and all other tasks for their worst-case values, and once again the execution of task 4 leads to task 1 missing its deadline at time T=100.

These scenarios are contrived for simplicity, but when this task set is executed (or simulated) the variation of execution times and the unpredictability of their phasing makes this apparently simple problem difficult to analyse completely *a priori*, due to the large number of circumstances which can lead to errors. Consequently, it is a good candidate with which to test a DCA-based overrun detection system.

The task simulator is configured to run the example task set. Each time a task is executed, its execution time is generated randomly, lying between its BCET and

WCET values. For this test, the generation of execution times between the BCET and WCET follows a beta distribution with parameters α=2 and β=7. These parameters give the distribution shown in Figure 5.9, which is skewed towards the best case value but which has a long tail, giving a low probability of values closer to the worst case.

This distribution mirrors the behaviour of a large number of tasks which exist in RTES, which generally complete in a time significantly quicker than the worst-case execution time, but where occasionally the execution time is significantly higher, perhaps due to an intermittent need to wait for an external device to respond, or the requirement for a significantly more complex execution for certain inputs. From an experimental perspective, it also serves to bias the actual execution time for any particular instance of a task towards the best-case value, reducing the likelihood that an overrun will actually occur, but providing plenty of scope for potential overruns to be predicted.

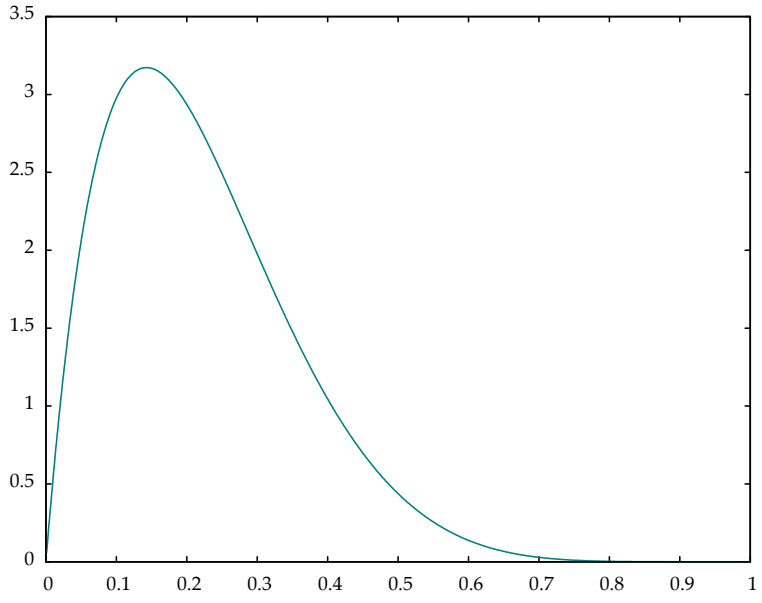


Figure 5.9: Beta distribution - probability distribution function (=2, =7)

The DCA is implemented as a part of the task simulator software, allowing the execution of the scenario system to be monitored in parallel with its simulated execution. In addition, an online schedulability analysis is also executed as the

simulation runs, and the output of this and the DCA compared as described in section 5.5.3.

For the initial tests, a population of 20 DCs is used to monitor the four tasks. This population size is perhaps larger than would be expected for such a simple system: the relationship between task set sizes and DC population size will be examined in more depth in a later chapter.

Each DC makes use of three categories of input signals: PAMP, Danger and Safe. The DCA has in previous applications made use of an inflammatory signal: this is an issue which can be examined later. Tests are run with two different sets of weightings and thresholds for these signals: a "low sensitivity" set where the maturation threshold is significantly higher than that which would be obtained from a single instance of a PAMP or danger signal, and a "high sensitivity" variant where only one instance of PAMP or danger is sufficient to cause the output signal to be over the maturation threshold. These weightings and thresholds are shown in Table 5.3. The use of these different sets of parameters gives some indication as to the effects of using different parameter sets to solve the same problem: the results will be used to guide further research into this area later.

Table 5.3: DC weightings and thresholds for initial application

	High sensitivity mode	**Low sensitivity mode**
PAMP weighting	6	6
Danger weighting	6	6
Safe weighting	-6	-6
Maturation threshold	5	10
DC cycle threshold	40	40

In addition, each test is run twice: once with the learning mode off, and then with the learning mode on, using a suitable learning period at the start of the execution. Each

scenario is repeated five times, seeding the random number generator differently each time.

5.6.1.1 Results: initial testing

The results for detection accuracy are shown in Table 5.4 through Table 5.7 for the four different scenarios. The results for each run are shown individually, and also combined to give an average value for all five runs for each scenario.

Table 5.4: Accuracy results (low sensitivity, learning mode off)

	Overruns		Correct detections		% Correct	
Run	**Potential**	**Actual**	**Potential**	**Actual**	**Potential**	**Actual**
0	1234	23	168	23	13.61	100.00
1	1234	24	174	24	14.10	100.00
2	1232	23	169	23	13.70	100.00
3	1236	23	176	23	14.20	100.00
4	1227	31	172	31	14.02	100.00
				Average	13.93	100.00
				Std Dev	0.23	0.00

Table 5.5: Accuracy results (high sensitivity, learning mode off)

	Overruns		Correct detections		% Correct	
Run	**Potential**	**Actual**	**Potential**	**Actual**	**Potential**	**Actual**
0	1231	25	187	25	15.19	100.00
1	1227	30	185	30	15.08	100.00
2	1240	17	219	17	17.66	100.00
3	1234	19	177	19	14.34	100.00
4	1237	21	189	21	15.30	100.00
				Average	15.51	100.00
				Std Dev	1.12	0.00

Table 5.6: Accuracy results (low sensitivity, learning mode on)

	Overruns		Correct detections		% Correct	
Run	Potential	Actual	Potential	Actual	Potential	Actual
0	1220	24	155	24	12.70	100.00
1	1216	23	160	23	13.20	100.00
2	1212	28	161	28	13.30	100.00
3	1215	26	170	26	14.00	100.00
4	1217	28	168	28	13.80	100.00
				Average	13.40	100.00
				Std Dev	0.46	0.00

Table 5.7: Accuracy results (high sensitivity, learning mode on)

	Overruns		Correct detections		% Correct	
Run	Potential	Actual	Potential	Actual	Potential	Actual
0	1221	22	194	22	15.89	100.00
1	1230	17	196	17	15.90	100.00
2	1225	20	189	20	15.40	100.00
3	1213	26	185	26	15.30	100.00
4	1223	23	180	23	14.70	100.00
				Average	15.44	100.00
				Std Dev	0.44	0.00

5.6.1.2 Evaluation of initial test results

The results of the initial test runs are generally positive. Firstly, it should be observed that, for all the variations of the algorithm tested, that there is successful detection of 100% of actual the actual overruns which occur. Given that there already exist a number of well-established techniques which are capable of detecting actual overruns accurately, it is encouraging to observe that the DCA-inspired technique performs as well as these methods. Although it is not anticipated that the DCA method would necessarily replace techniques such as exception-handling, there is little point in employing an additional system which is intended to enhance the detection of overruns but that is inferior in performance than existing methods.

A key requirement of this work is that, in addition to detecting overruns as they occur, the DCA-based system should be able to predict overruns which may occur as the system runs, before they actually happen. This is not possible using current techniques, so a method which is able to predict overruns would be particularly desirable.

It can be seen from the results shown above that the DCA-based solution is capable of predicting a number of overruns before they occur. The number of overruns successfully predicted is relatively low, ranging from around 13.5% to 15.5% in the test systems evaluated so far, but given that there are currently no techniques allowing any overruns to be predicted, the fact that even a small number can be successfully predicted is a significant result.

The results for each of the four different scenarios show little variation between the individual runs executed, as indicated by the low standard deviation values. This suggests that the behaviour of the DCA with respect to the detection of overruns is consistent between executions of the same scenario.

Considering the difference between the results across the range of scenarios evaluated, those runs executed with the learning mode turned off (i.e. where the DCA is pre-seeded with knowledge of the system properties) show a slightly higher rate of prediction than those runs where learning mode was enabled. This is not surprising, as the learned characteristics can only be based on the observed behaviour of the system, and consequently there may be instances where overruns are missed because a particular task executes for longer than has previously been observed.

The significant difference in the results is between the two different sensitivity configurations. The high-sensitivity configuration successfully predicts a larger number of overruns than the lower-sensitivity variant. This difference could be

caused by the interference of safe tasks in the low-sensitivity system, the presence of which would possibly mask the effects of overruns in the system.

Overall, the results of the initial tests suggest that the DCA has potential for application to the deadline overrun problem, although further investigation is clearly necessary.

5.6.2 Further testing: randomly-generated ten-task examples

To further verify the applicability of the DCA to this problem, the next phase of testing employs the DCA to detect overruns in more complex task sets, using the same set of parameters as before. Although these task sets are larger than the small engineered example used in the previous section, each containing ten tasks, they are still simple enough that online schedulability analysis can be easily conducted.

For this stage of testing, the operation of the algorithm is evaluated using a larger number of task sets each with randomly-generated task properties. The aim is to ensure that the algorithm works effectively across a range of similar yet slightly different problems.

As with the four-task example, the task sets are generated with rate-monotonic priority ordering. In order to guarantee the presence of potential overruns in the system, the most infrequent task is engineered to have a worst-case execution time which is greater than the period of the most frequent: in this way, the execution of the least frequent task may lead to an overrun in the most frequent. In order to ensure that the incidence of actual overruns is relatively low, the utilisation of the entire task set is bounded by a minimum best-case utilisation value and a maximum worst-case utilisation value. The actual execution times of tasks are again determined by a beta distribution, again with the parameters α=2 and β=7. This further reduces the likelihood of actual overruns by biasing the execution time towards the best-case, as with the four task example.

The DCA is evaluated as before, making use of a comparison between its output and that of a schedulability analysis running in parallel. To allow comparison across a number of task sets, the output for each simulation run is processed to give aggregated values for accuracy and responsiveness throughout that run. These are then compared with the results produced by a number of other task sets with similar properties.

In order to compare the output of the DCA across multiple task sets, it is necessary to devise a method by which task sets can be classified. Initially, the overall utilisation of the task set is taken as an indication of this. The utilisation of each individual task, which can be combined to give the overall system utilisation, is a classic measure of the complexity of a task set used by many of the traditional static-analysis techniques [1], for example the schedulability test from [20] shown in equation (3) (section 4.2.2). It can therefore be considered as a good base measure of "problem difficulty" in the context of this work.

Each task in the randomly-generated task sets have a known BCET and WCET, period and deadline, so allowing schedulability analysis to be carried out. From these values, the best-case and worst-case utilisation values for each task can be derived: these can then be combined to produce an overall utilisation for the entire task set.

5.6.3 DCA accuracy with respect to system utilisation

With the DCA configured as outlined above, the accuracy of the implementation is shown in Figure 5.10 for actual overrun detection; and Figure 5.11 for potential overrun detection. Each graph shows the results for two simulation runs: the red crosses indicate the results obtained with the DCA in high-sensitivity mode; the green crosses indicate low-sensitivity mode.

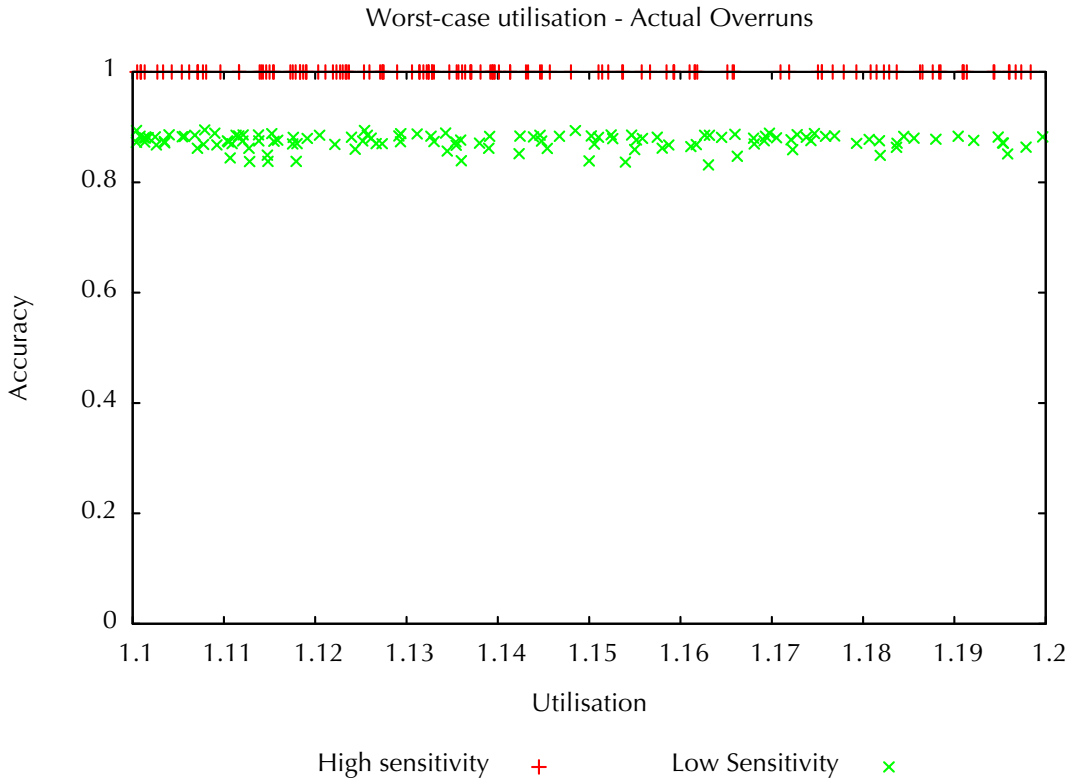


Figure 5.10: DCA accuracy with respect to WC system utilisation - actual overruns

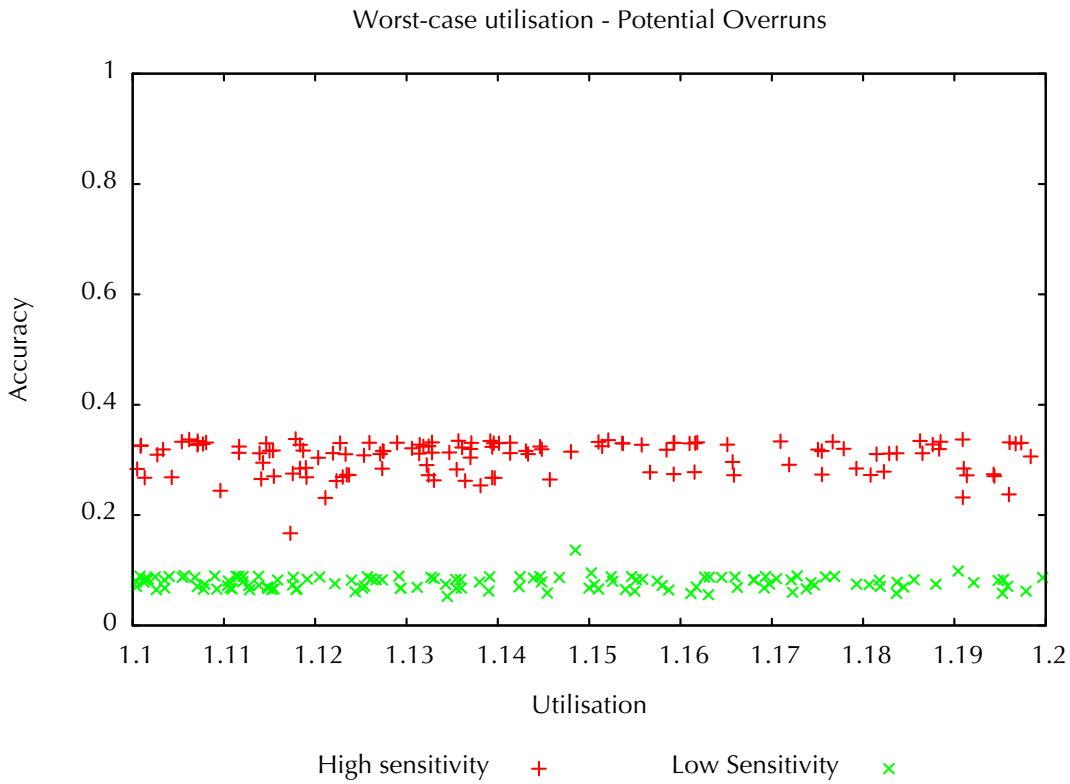


Figure 5.11: DCA accuracy with respect to WC system utilisation - potential overruns

Table 5.8: Averages and standard deviations for DCA accuracy

	Actuals		Potentials	
	LS	**HS**	**LS**	**HS**
Average	0.874076	1	0.077459	0.305128
Std Dev	0.01351	0	0.01127	0.029522

From these graphs it can be noted that the high-sensitivity version of the DCA consistently achieves a higher level of accuracy than the low-sensitivity version, both for the detection of actual overruns and the prediction of potential overruns. In the case of potential overrun prediction, this matches the results obtained for the engineered four-task example, although the difference between high-sensitivity and low-sensitivity accuracy levels is markedly greater than seen before.

The high-sensitivity version of the DCA correctly detects all incidences of actual overruns across the whole range of task sets, mirroring the results obtained for the four-task example. However, in this case the low-sensitivity version appears to only attain a maximum of around 90% accuracy. Although the detection of overruns is something which can already be accomplished in many systems by the use of traditional exception-handling techniques, the fact that the DCA is able to detect these overruns correctly indicates that the technique has potential for application in this area.

Table 5.8 shows the average and standard deviation values for DCA accuracy. In all cases, the standard deviation is very low indicating no significant variation in the accuracy values across the range of trials.

The differences between the results obtained here and those for the engineered four task example above can be attributed to an increase in complexity, caused by the increased size of the task sets used. The DCA monitors the run queue in order to generate its signal values, and an increase in task set size leads to a higher number of tasks being present in the run queue on average. Consequently, the effects of

potential or actual overruns may be masked by the presence of several tasks in the run queue which are not in danger of overrunning.

As with previous work, these results suggest that the choice of signals and weighting values is of paramount importance to the effective application of the DCA. As can be seen from Table 5.3, the difference between the high and low sensitivity versions of the DCA is a matter of variation of a single value, yet the effect on the overall results is clear to see. In addition, different values appear to have different effects on the results depending on the characteristics of the problem in question. The investigation of these effects is an area where further work is likely to be required.

System utilisation is considered to lie somewhere between the worst-case value and the best-case value. It is therefore prudent to examine the results with respect to the overall best-case utilisation in addition to the overall worst-case. The accuracy of the DCA with respect to best-case utilisation is therefore shown in Figure 5.12 for actual overrun detection, and Figure 5.13 for potential overrun prediction.

From the graphs, it is apparent that there is no correlation between the accuracy of the DCA and either the best-case or worst-case system utilisation. This can be confirmed by calculating the covariance of the dataset and the correlation coefficient between the two variables. The correlation coefficient is a value in the range -1 to +1. Although there are no definite correlation brackets, the nearer the correlation coefficient is to 1, the stronger the relationship is between the two variables. A correlation coefficient which is very close to zero on either side indicates that there is no discernable relationship between the two variables: their distribution can be considered to be essentially random. Where there is zero variance in one variable, the correlation coefficient is undefined.

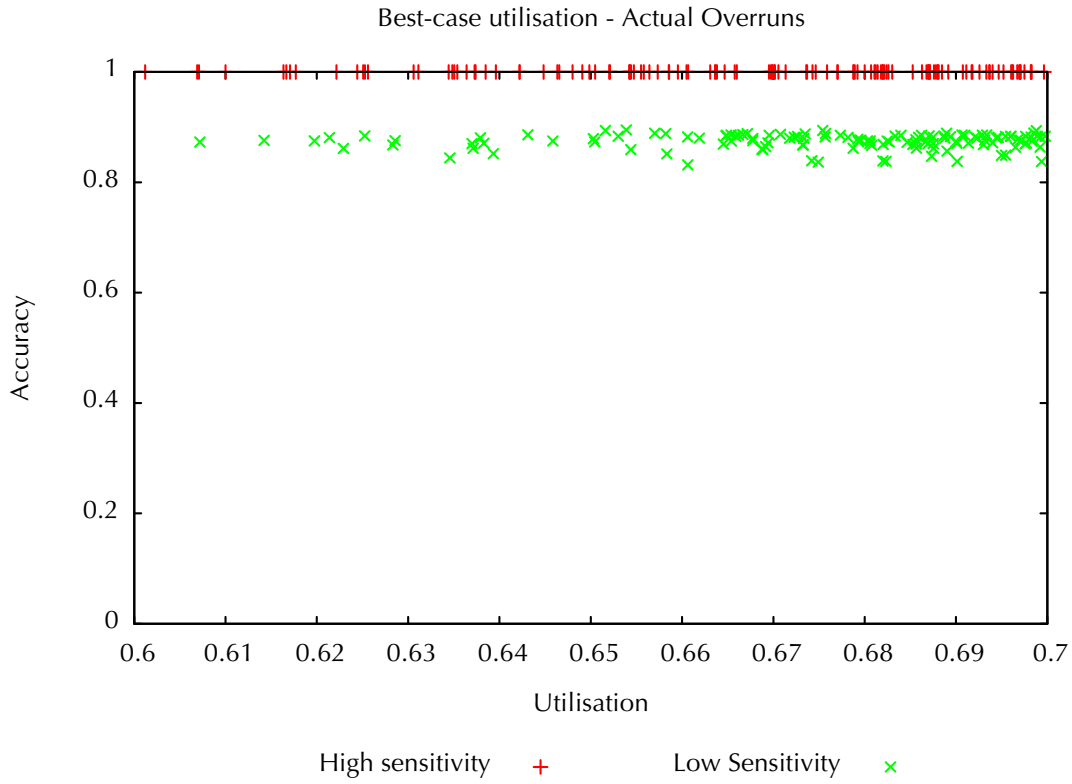


Figure 5.12: DCA accuracy with respect to BC system utilisation – actual overruns

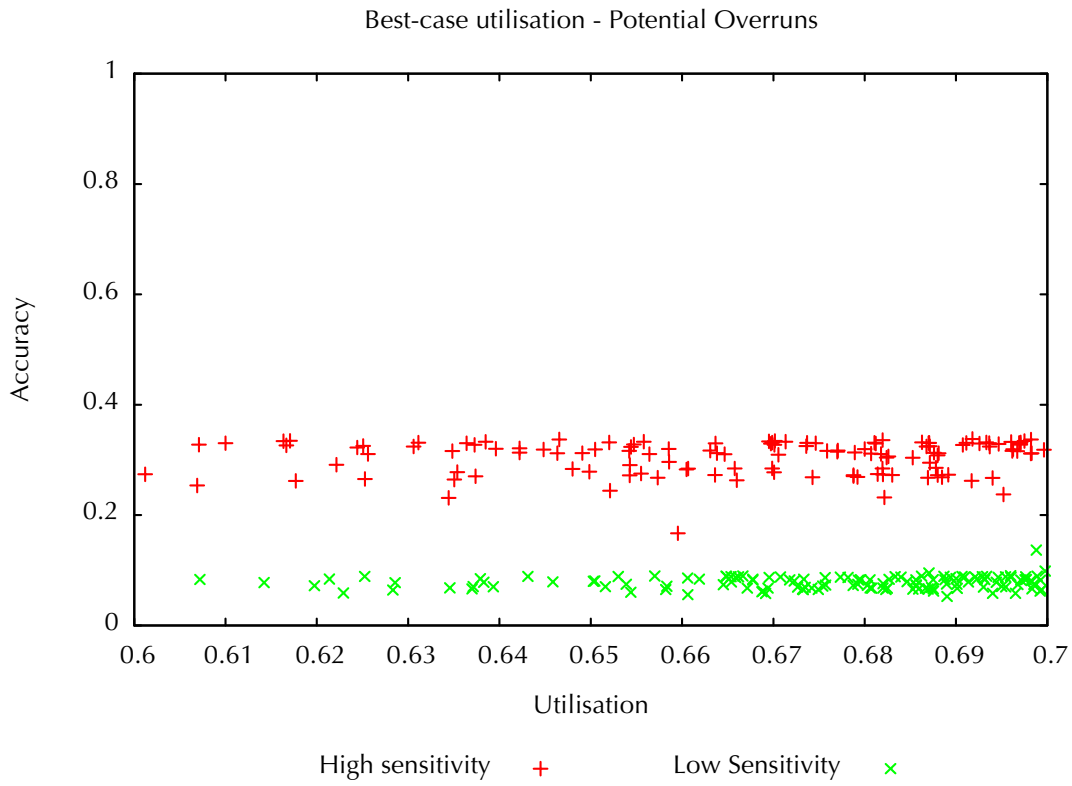


Figure 5.13: DCA accuracy with respect to BC system utilisation – potential overruns

Calculating the covariance (Table 5.9) and correlation coefficient (Table 5.10) of the data sets confirms that there is no linear relationship between utilisation and accuracy, in either the best-case or worst-case scenario, as in all cases the correlation coefficient is very close to zero.

Table 5.9: Covariance values for DCA accuracy with respect to system utilisation

	Actuals		Potentials	
	BC	**WC**	**BC**	**WC**
LS	6.7791×10^{-6}	-1.3247×10^{-5}	2.6442×10^{-5}	-8.8226×10^{-6}
HS	0	0	7.1085×10^{-5}	1.7513×10^{-5}

Table 5.10: Correlation coefficient values for DCA accuracy with respect to system utilisation

	Actuals		Potentials	
	BC	**WC**	**BC**	**WC**
LS	0.02358383	-0.03392832	0.11027491	-0.02708757
HS	-	-	0.09616984	0.02106284

5.6.4 DCA Responsiveness with respect to system utilisation

As detailed in section 5.5.3.2, as well as the accuracy of the DCA its responsiveness is also considered. Initially as with the accuracy this is evaluated with respect to the system's overall utilisation. The responsiveness is not calculated separately for potential and actual overruns. The responsiveness with respect to worst-case utilisation is shown in Figure 5.14, and with respect to best-case utilisation in Figure 5.15.

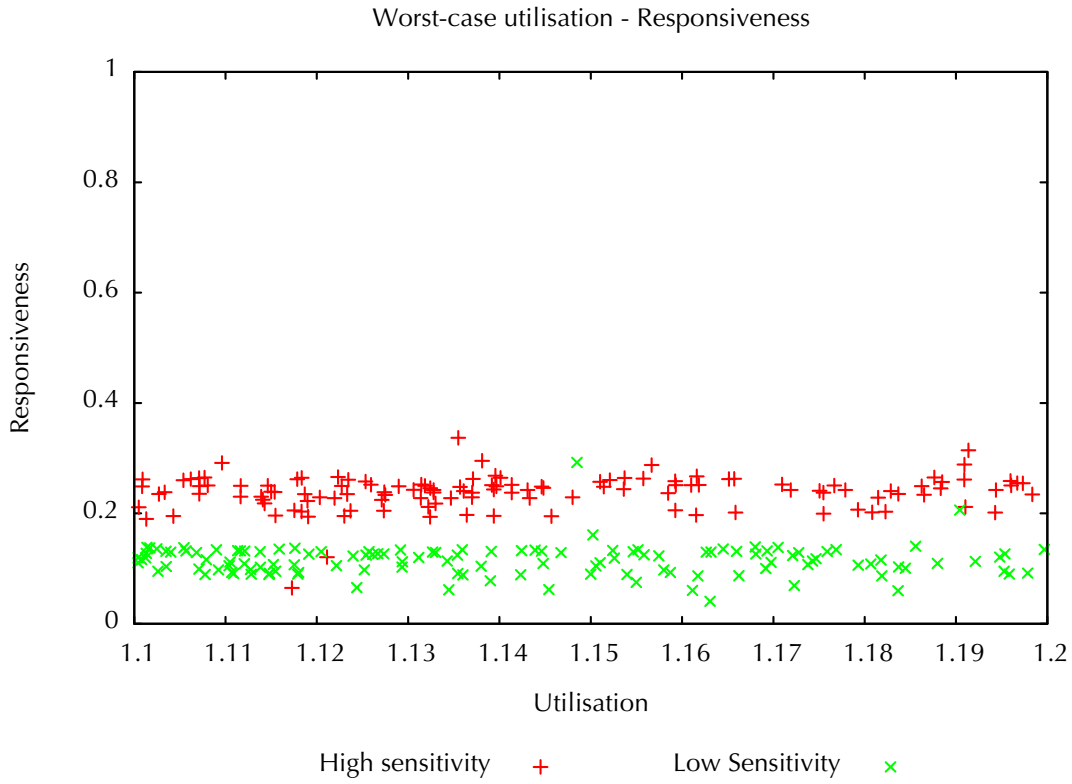


Figure 5.14: DCA responsiveness with respect to WC system utilisation

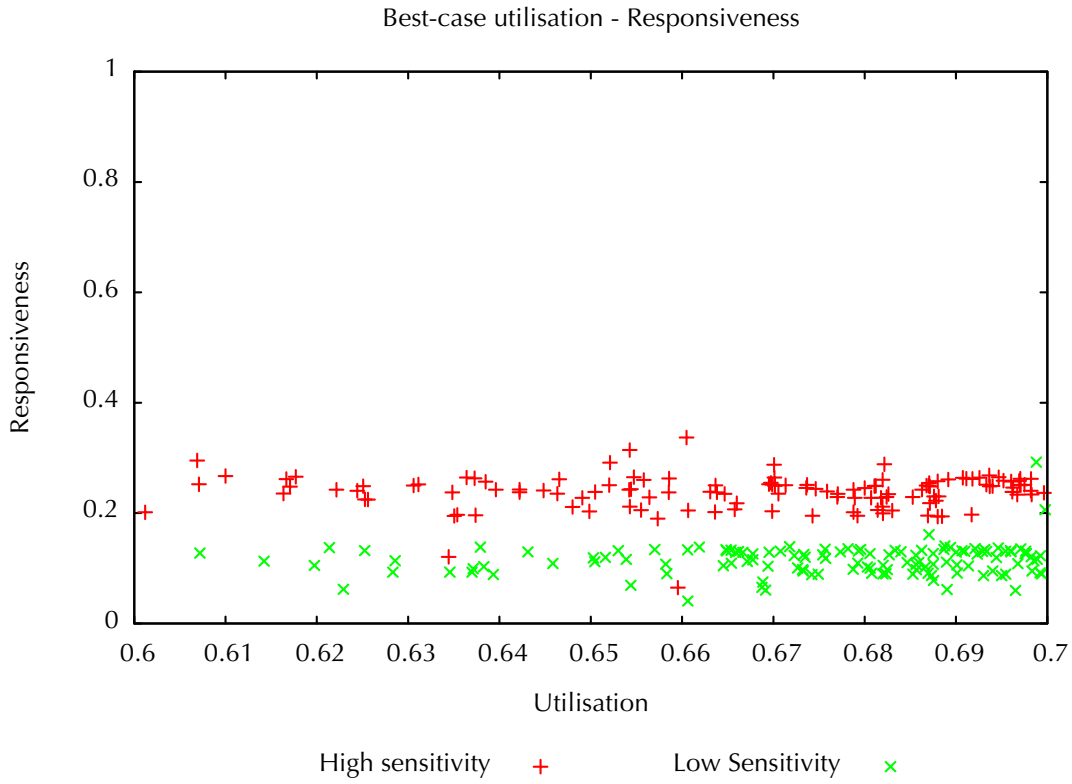


Figure 5.15: DCA responsiveness with respect to BC system utilisation

As seen with the accuracy, the high sensitivity version of the DCA is also more responsive than the low sensitivity version. The values for the overall average responsiveness and standard deviations are shown in Table 5.11.

Table 5.11: Averages and standard deviations for DCA responsiveness

	LS	HS
Average	0.114445	0.237935
St. Dev	0.027711	0.031399

Table 5.12 shows the correlation and covariance values for responsiveness across all task sets. As with the accuracy measure, there is not a significant level of correlation between the overall system utilisation and the responsiveness of the DCA.

Table 5.12: Correlation coefficient and covariance values for DCA responsiveness with respect to system utilisation

	Correlation		Covariance	
	BC	WC	BC	WC
LS	0.136973	-0.0156	8.08E-05	-1.2E-05
HS	0.031394	0.147528	2.47E-05	0.00013

5.6.5 Relationship between responsiveness and accuracy

Although the measures of accuracy and responsiveness are intended to be compared against other metrics, at this point the relationship between accuracy and responsiveness can be considered. Figure 5.16 and Figure 5.17 are graphs of responsiveness against accuracy, for actual overruns and potential overruns respectively, and Table 5.13 shows the correlation coefficients and covariance values between DCA responsiveness and accuracy.

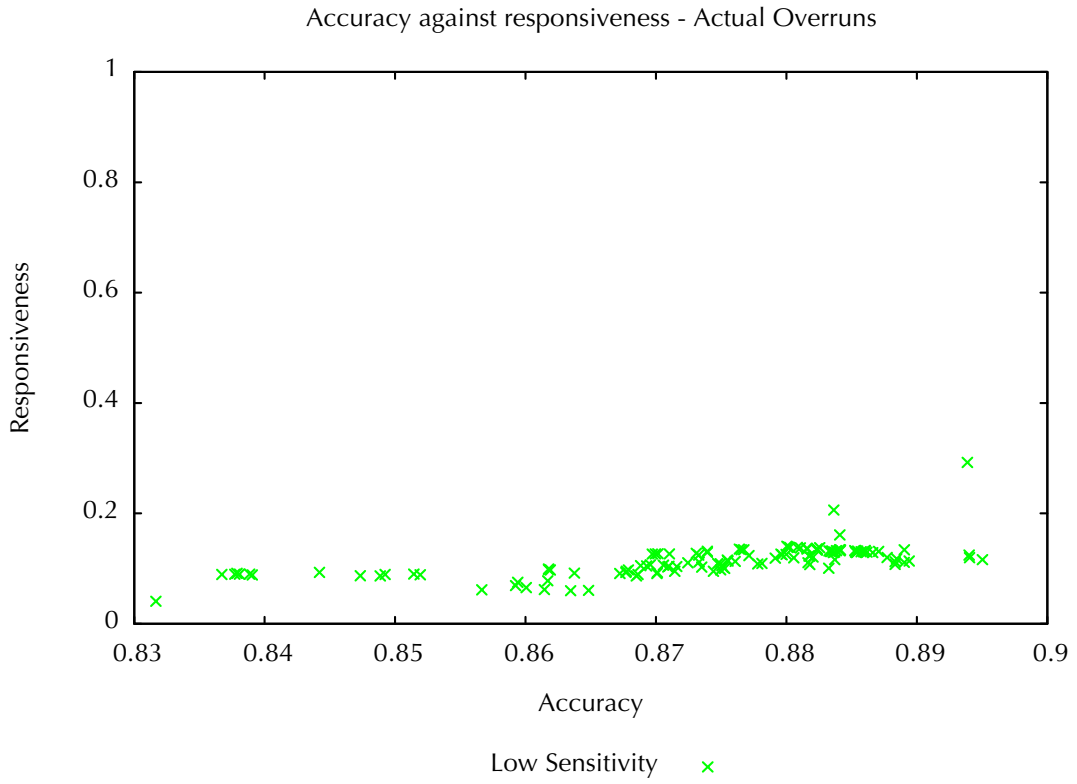


Figure 5.16: DCA accuracy with respect to responsiveness - actual overruns

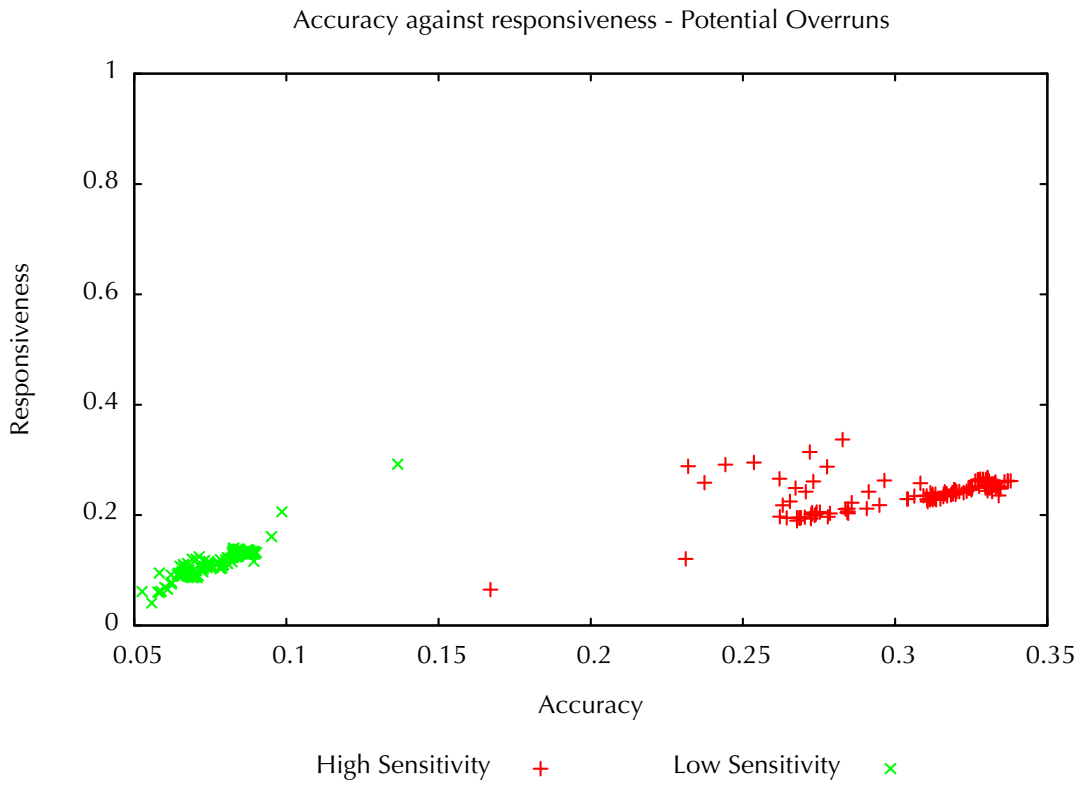


Figure 5.17: DCA accuracy with respect to responsiveness - potential overruns

Table 5.13: Correlation coefficients and covariance values for responsiveness with respect to accuracy

	Correlation		Covariance	
	Potentials	**Actuals**	**Potentials**	**Actuals**
LS	0.921925	0.64964	0.000288	0.000243
HS	0.528161	-	0.00049	0

Due to the high sensitivity DCA having an accuracy rate of 100% when detecting actual overruns, the covariance of the accuracy is 0 and the correlation coefficient is undefined. Consequently, the high sensitivity actuals are not shown in Figure 5.16: this allows the low-sensitivity values to be seen in greater detail as the x-axis can be scaled more appropriately for those values.

For the detection of potential overruns wth both variants, and for the detection of actual overruns with the low sensitivity variant, the correlation coefficients indicate that there is a significant relationship between the accuracy and responsiveness of the DCA, indicating that where the DCA is more accurate it is also more responsive. Given the use of the accuracy as a fundamental part of the responsiveness calculation this relationship is perhaps to be anticipated.

5.6.6 Alternative comparative measures

It is clear from examining the graphs that there is little correlation between the overall utilisation of a given task set and the accuracy or responsiveness of the DCA when applied to detect overruns in that task set: the values are roughly consistent across the entire range of utilisations, both for best-case values and also for worst-case values. It would be desirable to be able to correlate the accuracy and responsiveness of the DCA output with a metric which can be derived from the task set in question, allowing the suitability of the DCA to a particular problem task set to be assessed easily.

Clearly, system utilisation is not a suitable measure, as there is no correlation between either worst-case utilisation and accuracy/responsiveness or best-case utilisation. It is therefore necessary to investigate alternative measures of task set complexity and their effect on the operation of the DCA.

An alternative measure which can be used as a measure of system load is the average slack time for each task. If a task has a known deadline by which it must have completed (as is the case with the tasks in this scenario), the slack time is the difference between the time at which that task actually completes and the task deadline. Therefore, a task which completes exactly on its deadline has a slack time value of zero. A task which completes before its deadline has a positive slack time value, with larger values indicating earlier completion. A task which overruns, i.e. completes after its deadline, has a negative slack time value.

A slack time value can be obtained each time a task executes (each such time a task executes is referred to hereon as an *execution instance*). By measuring the slack time values for each execution instance of a given task, and combining them appropriately, an average slack time value for that particular task over the entire simulation can be obtained, and by combining the values across all tasks present in the task set, a value can be calculated which represents the overall slack time value for the entire task set. This value can then be used as a measure against which the accuracy and utilisation of the DCA can be compared across a number of different task sets.

Although a good potential measure of task set complexity, the concept of slack time is not particularly intuitive, as a larger value indicates a lower level of complexity. A more intuitive measure is perhaps to use the average overrun time, which is simply the opposite of slack time. A larger value of overrun time therefore indicates a higher level of complexity within the task set.

When calculating the overall average overrun time for a task set, only execution instances where the schedulability analysis detects a potential or actual overrun are considered. This has the effect of disregarding altogether any tasks which never overrun during the course of the simulation, and also of biasing the overall overrun time value towards those tasks which experience overrun conditions most frequently.

5.6.7 Accuracy and responsiveness with respect to average overrun time

Figure 5.18 and Figure 5.19 show respectively the actual overrun accuracy and potential overrun accuracy for the 10-task examples, with respect to the average overrun time for the entire task set. Figure 5.20 shows the average responsiveness with respect to the average overrun time.

With regard to the accuracy graphs, it can be seen that there is a much stronger correlation between the average overrun time and the overall accuracy of the DCA. This is particularly true for potential overruns, where there is an easily observable relationship between overrun time and accuracy for both the high and low sensitivity versions of the DCA. Calculating the correlation coefficients, shown in Table 5.14, shows that there is a significant level of correlation.

Table 5.14: Correlation coefficients and covariance values for DCA accuracy against average overrun time

	Correlation		Covariance	
	Potentials	**Actuals**	**Potentials**	**Actuals**
LS	0.772476253	0.482429	0.000288	0.064976
HS	0.86895438	-	0.00049	0

Table 5.15: Correlation coefficients and covariance values for DCA responsiveness against average overrun time

	Correlation	**Covariance**
LS	0.893813029	0.246918
HS	0.628051079	0.098317

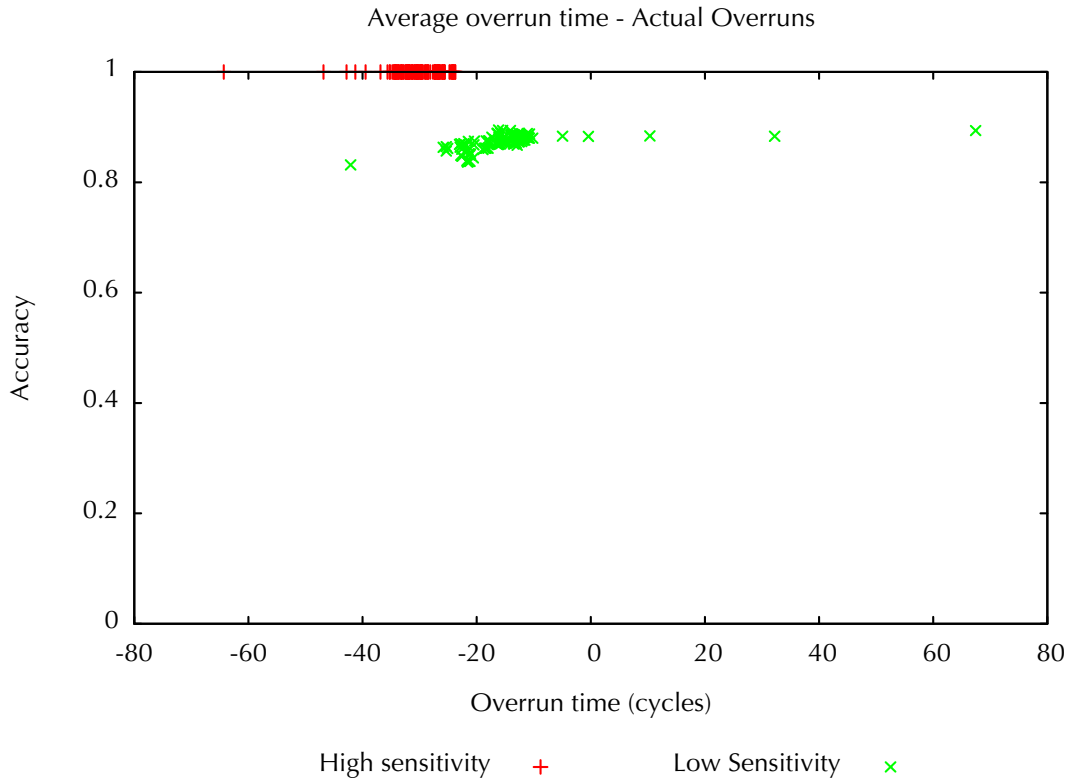


Figure 5.18: DCA accuracy with respect to overrun time - actual overruns

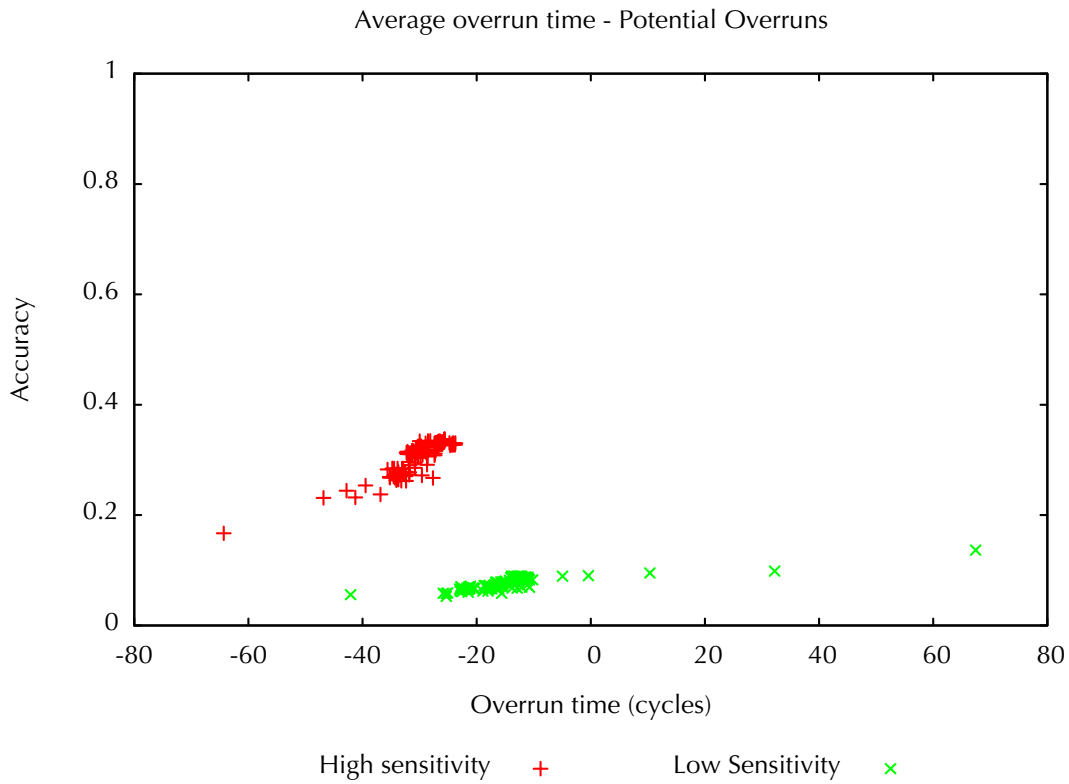


Figure 5.19: DCA accuracy with respect to overrun time - potential overruns

As well as a far greater level of correlation between the accuracy and the average overrun time, it can be observed from the responsiveness graph (Figure 5.20) that there is also a significant level of correlation between the responsiveness and the average overrun time. This greater level of correlation can be verified by the calculation of the relevant correlation coefficients, shown in Table 5.15.

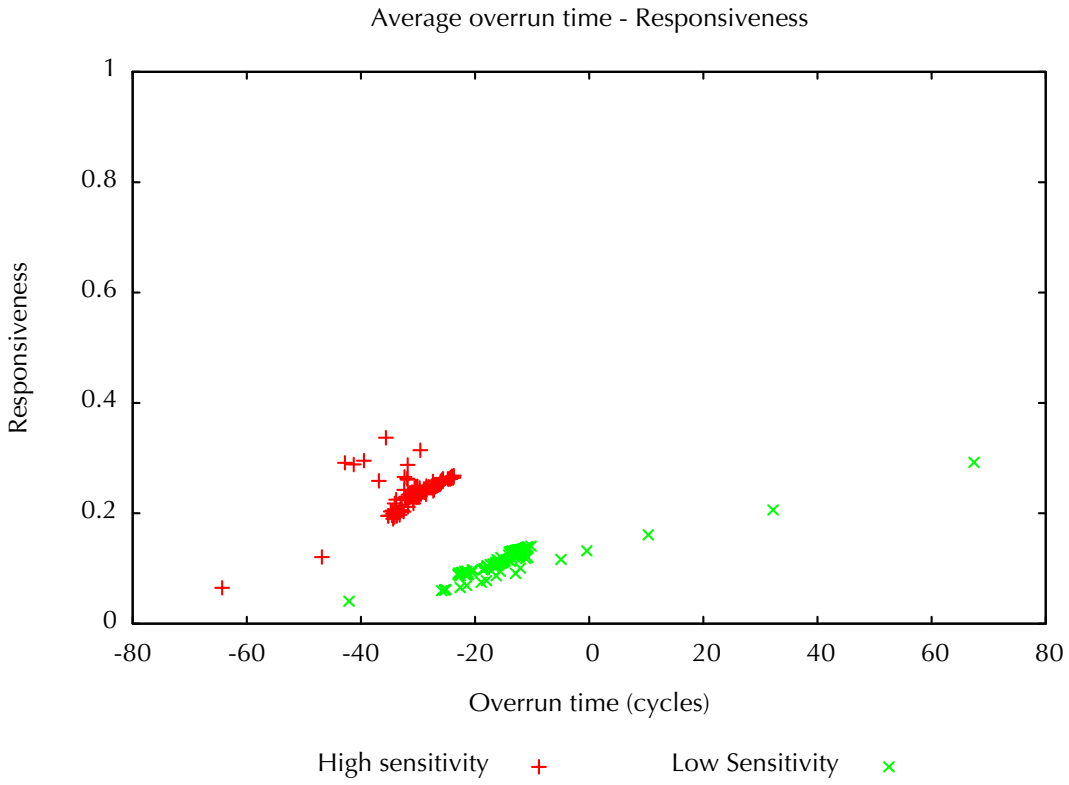


Figure 5.20: DCA responsiveness with respect to overrun time

5.6.8 Comparison of system utilisation and overrun time

The analysis of accuracy and responsiveness with respect to system utilisation in section 5.6.3, and overrun time in section 5.6.7, shows that while there is no discernable correlation between utilisation and the DCA accuracy or response, that there does appear to be a relationship with the average overrun time.

As can be seen from the graphs and highlighted by the positive correlation coefficient between overrun time and both accuracy and responsiveness, the accuracy and responsiveness of the DCA increase as the overrun time. From this it can be deduced

that, overall, the greater the severity of overruns experienced by a system, the more likely the DCA is to detect or predict those overruns accurately. This is not in itself an entirely surprising result however, as a task which overruns more severely should be detectable as such relatively earlier in its release cycle than a task which will only overrun by a few clock cycles. Consequently, a longer-overrunning task can be considered to be more easily detected than a shorter-overrunning task.

Despite the presence of an observed correlation with DCA accuracy and responsiveness, it is not readily apparent that the average overrun time is a better metric by which to categorise task sets than the overall system utilisation. Although on the surface, the average overrun time can be taken to give some measure of a task set's complexity, it is only one factor amongst many and basing an overall task set complexity on such an observation is likely to be over-simplistic.

Given the relatively limited variation in the observed results compared with the variation across, for example, DCA sensitivity, it may not be appropriate to consider classifying task sets by complexity in this context, but to instead examine the aspects of the DCA which can serve to enhance its operation irrespective of the complexity of the task set to which it is applied.

5.7 Overall observations

This chapter has examined the application of the DCA to deadline overrun detection in task scheduling, an example of a typical real-time problem. The DCA's signal and antigen based operation maps well onto the characteristics of the deadline overrun problem, providing that the signals and antigens are derived appropriately.

The DCA appears to perform favourably when its results are compared against a traditional schedulability analysis run in parallel with it. In particular, the DCA is able to successfully detect a high proportion of the total number of actual overruns which occur in the system. It is also able to successfully predict a proportion of the potential overruns present in the system which do not go on to actually overrun: this

is a more desirable ability as the detection of actual overruns is something which current real-time methods are able to detect; but there is not yet a straightforward method for the detection of overruns which may potentially occur without using time-consuming formal analysis methods.

From the results obtained so far, making use of a number of randomly-generated example task sets each containing ten tasks, there is a little overall variation in the performance of the DCA between task sets. However, this variation is not affected by task set complexity when measured using a traditional utilisation measure. It does appear to be dependent to some extent on the average overrun time for a given task set, such that the higher the overrun time on average, the greater the level of accuracy and responsiveness.

However, it can be seen that the factor with the most significant effect on the DCA's performance is the setting of its own parameters, here represented by the high and low sensitivity variants. The levels of accuracy and responsiveness are consistently higher for the high sensitivity variant than for the low sensitivity one, across all the evaluated task sets. The difference between these variants (see Table 5.3) amounts to a difference in a single value – the value of the maturation threshold.

It is clear that the performance of the DCA is more significantly affected by the choice of its own parameters than by any variation in the overall properties of the task sets themselves. In order to establish how the large number of DCA parameters, including the weights and thresholds considered so far, affect the overall operation of the algorithm, it is necessary to conduct further experiments. This will be the subject of the next chapter.

6

Understanding the effect of DCA parameters

Chapter 5 examined the potential for using immune-inspired techniques to enhance the reliability of such real-time embedded systems (RTES). In particular, the use of the DCA, a technique derived from the principles of innate immunity, has been investigated with respect to the detection of typical anomalies in RTES, and it has been shown that it has good potential to detect and predict deadline overruns. However, it is clear that further investigation of the algorithm is required, in the context of this problem area.

6.1 Areas requiring further investigation

The results of the initial investigations into the effectiveness of the DCA carried out in chapter 5 indicate that the DCA's own parameter settings have a greater effect on its output than the variation in problem characteristics. Therefore, this chapter

investigates the effect of a number of the DCA parameters in order to try and formulate an understanding of the role they play in the overall performance of the algorithm.

In addition to this, the investigations carried out so far have been limited to a small range of example problems. For the DCA to be of use in the RTES development process in the real world, it must be shown to be generally applicable either over a wide range of systems, or alternatively over a subset of systems which can be clearly defined. Further investigation in this chapter is conducted over a much wider range of example systems of varying sizes.

6.1.1 DCA parameter investigation

A key element of the DCA as outlined in section 3.7.2, is the large number of parameters which control its operation. The investigations carried out in chapter 5 have established that the variation of just one of these parameters, the migration threshold, has a significant effect on the overall performance of the algorithm. It is therefore highly likely that the alteration of other parameters will also affect the performance to some extent.

As presented by Greensmith et al in [74], the DCA employs eight numeric parameters as shown in Table 6.1. There are many different combinations of these parameters, resulting in a large number of possible parameter sets. In addition to these parameters, but equal in importance as far as the operation of the DCA is concerned, are the weighting values employed in the calculation of output signals, and the migration threshold used to determine DC maturity.

Table 6.1: DCA parameters (from [74])

Parameter	Description
I	Maximum number of input signals per category
J	Maximum number of categories (of input signal)
K	Maximum number of antigen in tissue antigen vector
L	Maximum number of DC cycles
M	Maximum number of DCs in population
N	Maximum number of antigen contained per DC
P	Maximum number of output signals per DC
Q	Number of antigens sampled per DC each DC cycle

Of the parameters in Table 6.1, three (I, J and P) relate to the various signals employed by the algorithm; the remainder relate to the overall operation of the algorithm itself.

A number of the parameters are likely to be highly significant in affecting the output of the DCA, for example the overall size of the DC population, and the number of antigens sampled per cycle. It is also reasonable to assume that some of these parameters will be co-dependent, such that the effect of altering one parameter may also depend on the value of one (or possibly more) other parameters.

In addition to this, these parameters are likely to have a significant effect on the computational requirements of the algorithm: for example, it can be deduced that an increase in the size of the DC population will require a corresponding increase in computational power in order to process the signal calculations within the additional cells, and also an increase in the amount of RAM required in which to store them. Resource issues will be considered in more detail in chapter 7, therefore it is important at this stage to consider which parameters may have an effect on the algorithm's computational requirements, in order to guide this future work. Particularly when considering the application of the DCA in resource-constrained environments, there is a need to investigate the relationship between DCA performance and, for example, the size of the DC population. Also, it must be

established whether there is an optimum DC population size for a system of a given complexity, or alternatively whether a trade-off can be reached between DCA performance and computational requirements.

6.1.2 General applicability of the DCA

In order to determine the general applicability of the algorithm, it is necessary to carry out further investigations into its performance over a wider range of problems.

In chapter 5 the DCA was tested using a number of purely periodic task sets, consisting of first four tasks and then ten tasks. The purpose of these tests was to establish that the DCA can be applied to this class of problem, and that it demonstrates reasonable performance potential. For this reason, the example problems used in chapter 5 are considerably simpler than those which may be encountered in a real-world system.

The most significant difference between these example task sets and a real-world system is the size of the task set: a system applied in a complex device will include significantly more than ten tasks. As the complexity of many classes of embedded system is increasing rapidly (with a corresponding increase in the number of tasks present in these systems), any technique which is intended to be applied as part of their development must be able to cope with anticipated increases in complexity in the foreseeable future. Therefore, it is necessary to demonstrate that the DCA operates successfully when applied to a system with a level of complexity more akin to that which may be found in a typical application.

Another significant difference between the example systems and typical real-world systems is that real-world systems will typically be comprised of a mixture of tasks, some of which are periodic as in the example systems, and others which only occur occasionally – i.e. they are either aperiodic or sporadic [24]. In a real-world system, these tasks may be triggered in response to external events, for example the user pressing a control button, or the receipt of a data packet through a network interface.

The presence of these tasks and the effect they may have on the effectiveness of the DCA has not been considered in the work carried out so far.

Although the release of these tasks does not occur according to a regular periodic cycle, it is generally the case that there will be a minimum period between any two releases of the same task: this is referred to as the task's minimum inter-arrival time. As it is difficult to analyse systems where unpredictable behaviour is present, it is relatively common, particularly when performing response time and other similar analyses, to model aperiodic or sporadic tasks as periodic ones, using the minimum inter-arrival time as the task's period: this is the worst-case scenario for the occurrence of an aperiodic task. A method making use of this property, known as the Sporadic Server algorithm, is well-known in the RTS domain [127]. Although for the purposes of the analysis the task is modelled as occurring more frequently than may actually be the case, a successful analysis in using worst-case values guarantees that the system will always be schedulable.

Although the scheduling component could be modified to accommodate aperiodic or sporadic tasks natively, the fact that it is possible to model aperiodic tasks as periodics means that the lack of support for non-periodic tasks in the simulation software is not considered to be a major omission – the use of purely periodic task sets should suffice for the purposes of the simulation. However, it is necessary to evaluate the effectiveness of the DCA with task sets significantly larger than those used in chapter 5.

6.2 Derivation of DCA parameters in the literature

Considering the large number of parameters which are involved in the DCA's operation, there appears to have been little investigation as to their effects in the published literature. Instead, the emphasis in the majority of the work has been on the selection and weighting of input signals, such as in [74], where the results are presented across a variety of combinations of input signals, but no investigation is conducted into the variation of any other parameters.

The signal weights are another area where there has been little investigation. The weighting values have, in work published to date (e.g. [73], [70]), been based solely on *in-vitro* observations of living DCs. Although the use of such values in *in silico* systems aligns the algorithm well with the underlying biology, the relevance of these weighting values to the types of problems to which the DCA is applied is at best questionable: ideally, weighting values should be devised which are specific to the problem in question.

The only significant investigation into any of the DCA parameters is conducted in Greensmith's PhD thesis [69], as a part of the initial development of the algorithm. The parameters derived from this study have subsequently been employed in a number of applications of the DCA, over a range of problem areas, with no further investigation into their effect on the operation of the algorithm. Although there is clearly a desire to create an algorithm which is effectively "plug-and-play", this must be balanced against the need for the algorithm to be as efficient and effective as possible.

In the literature, the majority of the applications of the DCA have been to intrusion detection problems. The method by which these problems have been approached involves offline analysis of test data. However, the application of the DCA to the problem of anomaly detection in RTES uses an entirely different application model: in order to be of use during the development process, the algorithm must run alongside a running system and produce relevant results in real time. Although it may be fair to assume that, for similar problems, the DCA can legitimately make use of the same set of parameters, this cannot be assumed for application in a completely different problem domain with a different method of operation, as the specific problem characteristics which may affect the algorithm's operation are not taken into account. When considering the application of the DCA to the deadline overrun problem, there is therefore a need to conduct a separate analysis in the context of that problem.

In addition, when considering the application of the DCA to embedded systems, there is a particular need to consider effect of resource utilisation on the operation of the algorithm. Any investigation into the algorithm's parameters must therefore be conducted with this in mind.

As an example, the analysis of DC population size in [69], which focuses solely on the effectiveness of the algorithm, only investigates the effect of population sizes of 10, 100, 200 and 500 cells, and concludes that a population size of 10 performs poorly, and that any population size of 100 or more performs well, with no discernable benefit to the use of additional cells. Consequently, a population size of 100 cells is used for all subsequent applications [69].

However, when looking to apply the algorithm in the context of an embedded system where resources are likely to be constrained, there is a need to trade off resource utilisation against algorithmic performance. It is therefore particularly important to investigate the relationship between parameters and algorithmic performance, particularly around points where a small change in the value of a parameter has a large effect on the overall performance. In the case of the previous population size study, it can be deduced that the algorithmic performance increases significantly between a population of 10 cells and 100 cells. However, the difference between these population sizes may be highly significant in a situation where resources are constrained. In this situation, it is necessary to conduct further investigations to establish the exact relationship between population size and performance between these two values, such that an appropriate trade-off can be made between them.

6.3 Experimental methodology

The investigation of the DCA's parameters is complex, due to the number of parameters which are used to control the algorithm's behaviour. A comprehensive investigation of the algorithm's performance needs to examine the effect of each of these parameters on the overall behaviour, in order to establish which parameters are

significant, and the range of values which have the greatest bearing on the results. In addition, consideration needs to be given to the effect that each parameter has on the overall resource requirements of the algorithm, in particular processing time and memory usage.

There is a distinct possibility that a number of the parameters used by the algorithm are co-dependent: that is, the effect of altering one parameter may differ depending on the values of a number of other parameters. A comprehensive parameter investigation needs to identify the presence of any co-dependencies between parameters, and the nature of any such relationships such that the effects of varying the relevant parameters are understood.

The need to investigate the effect of a large number of parameters, particularly where there are likely to be incidences of co-dependence, requires the use of a suitable experimental method to ensure that the range of possible parameter combinations is suitably covered.

Experimental methods have been subject to a significant amount of research, and as a result are well understood. In particular, techniques have been developed allowing the investigation of problems with multiple parameters.

When conducting investigations involving multiple parameters, perhaps the most obvious approach is to set all the parameters to a fixed or baseline state, and then to investigate the effects caused by varying one parameter at a time, while leaving all the others set to the baseline [112]. An example of this approach in a system with two parameters, each of which has three possible values, is shown in Figure 6.1.

As can be seen from Figure 6.1, a large amount of the potential parameter space is not covered, and consequently this method of investigation does not allow the investigation of co-dependencies between parameters [54]. With a problem such as

the DCA, it is probable that some degree of co-dependency will exist between parameters, and any investigation must be able to accommodate this.

In situations where parameter co-dependence is likely to be present, the ideal method is a factorial search, whereby each possible combination of parameters is investigated [112]. This allows the full analysis of parameter co-dependence, giving an accurate overall picture of how variations in the parameter set affect the outcome of the algorithm. The difference in coverage between a full factorial search as opposed to variation of single parameters can be seen in Figure 6.1.

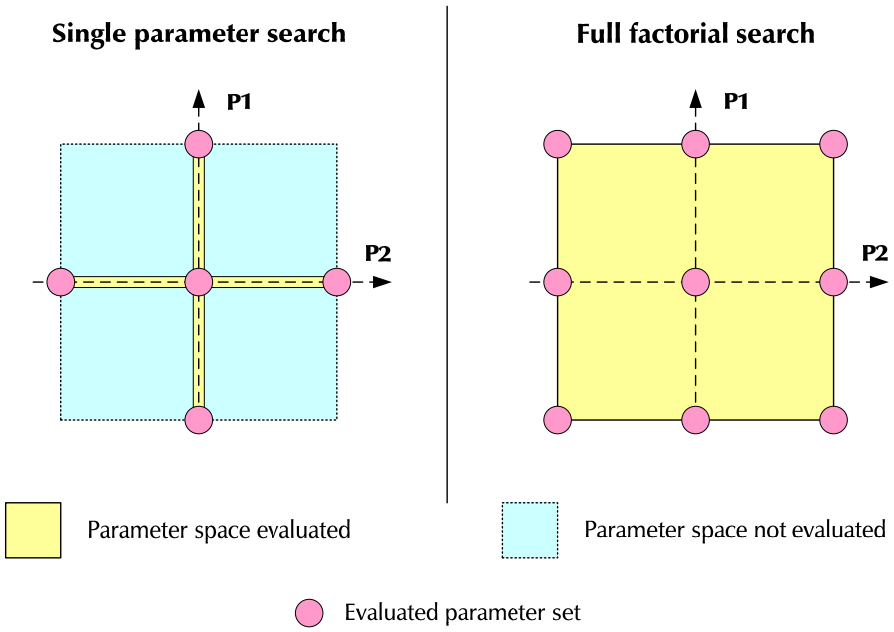


Figure 6.1: Parameter space coverage comparison: single parameter search and full factorial search

However, problems arise with factorial design when a large number of parameters are involved. The number of possible combinations of parameters in a system is given by the equation:

$$C = n^x \tag{8}$$

where: C = *total number of parameter combinations*
n = *number of values for each parameter*
x = *number of parameters*

Therefore, a system with two parameters, each of which can take three possible values, has a total of nine potential combinations as seen in Figure 6.1, a number for which it is computationally feasible to evaluate each one. However, additional parameters, and in particular increased numbers of potential values for those parameters, cause the number of possible combinations to grow exponentially: for example, a problem with four parameters, each with four values, gives 256 possible combinations.

For a problem such as the DCA, involving eight parameters most of which are integer-valued, the total number of different combinations is very large: assuming only ten values for each parameter gives over one billion different combinations, well beyond the scope of practical analysis, both due to the large amount of computation required to simulate each, and the difficulty of processing the large amounts of data produced.

It is therefore necessary to devise an experimental method which allows sufficient coverage of those parameter sets which have a significant impact on the overall outcome, whilst at the same time not using computational resources covering parts of the space which do not have a significant effect on the outcome.

6.3.1 Selecting simulation parameters

In order to investigate the effects of differing parameters on the operation of the DCA, it is first necessary to derive a set of suitable simulation parameters. Primarily this requires the overall simulation time to be determined. In addition, it is necessary to decide on the number of different task sets which will be evaluated, and also the number of times each of those task sets will be evaluated in order to ensure that the generated results are correct and consistent. It is also necessary to determine which parameter combinations will be evaluated.

In order to avoid ambiguity, the following terms will be used to describe the various possible different types of parameter variation which will be encountered in the remainder of this chapter:

- Iterations refer to different task sets which are generated and simulated in order to establish the effectiveness of the DCA. This is the top level of the simulation hierarchy. It is intended that the results obtained for different iterations can be compared in order to determine the general applicability of the DCA.

- Configurations refer to the parameter combinations employed to test the DCA. A single iteration will be simulated with a number of different configuration modes, each representing, for example, a particular combination of population size, cell lifetime, and any other variable parameters.

- Runs refer to individual simulation within a particular configuration. Each configuration within an iteration is simulated for a number of times in order to give consistency in the obtained results. In order to prevent two simulation runs being duplicates of each other, the simulation environment may take as a parameter an integer value which is used to give the initial seed for its random number generators. The use of differing seed values for each run allows different execution times and sequences to be produced, and where necessary the seed value can be logged to ensure future repeatability.

When determining the simulation parameters, it is necessary to trade-off between the desire for consistency in the obtained results, and the requirement for the simulation to complete in a reasonable amount of time: as has already been discussed in section 6.3, the complexity associated with the full factorial evaluation of a large number of parameters is exponential and quickly becomes computationally infeasible.

Of course, it is possible that there may be further trade-offs between the simulation parameters themselves. For example, a similar level of consistency may be achieved by using a relatively large number of short runs, as is achieved by using a smaller number of longer runs. It will also be the case that both will have an effect on the overall runtime of the simulation, which, given task sets of equivalent complexity, is directly proportional to both the number of cycles simulated and the number of simulation runs: therefore it follows that the same execution time will be taken for m runs of n cycles, as would be taken for $m/2$ runs of $2n$ cycles. It is therefore necessary to consider the relative consistency of these combinations so as to make best use of the available computational resources and minimise the required execution time for the simulation process.

In order to select suitable simulation parameters, a number of task set iterations of varying sizes are evaluated over a small subset of the total configuration space. In order to determine the desired simulation parameters for further experimentation, it is necessary to perform these initial evaluations over a large number of runs, with each run consisting of a large number of simulation cycles.

The output of these experiments is then analysed with respect to the simulation cycle count at regular intervals, to establish at which point the output converges to give stable values. Based on the results of this analysis, suitable simulation parameters can be selected to ensure that, in the majority of cases, the results converge to a stable value within the allotted simulation time.

The initial convergence experiments are performed using task sets of sizes at opposing ends of the likely task set spectrum. Therefore, the first set of experiments makes use of task sets containing ten tasks, as these are likely to be the smallest task sets used, and of 100 tasks, which are an order of magnitude more complex.

6.3.2 Selecting simulation parameters: analysis

For each task set size, ten iterations, each consisting of ten runs of one million simulation cycles, were simulated. These output values obtained for algorithmic accuracy can be seen graphically, plotted against simulation runtime. For the ten-task simulations, the output is shown over one run in Figure 6.2, five runs in Figure 6.3 and all ten runs in Figure 6.5. For the 100-task simulations, Figure 6.6 shows the output over one run, Figure 6.6 over five runs, and Figure 6.7 over all ten.

In each of these cases, the first n runs out of the ten simulated were used to produce the results. This allows the effect of running a smaller number of runs to be observed, and also allows direct comparisons to be made between different numbers of runs, as the results for $n+1$ runs use the same data as the results for n runs, with the addition of an extra run at the end. Summary graphs showing the complete set of results across all combinations of simulation runtime and number of simulation runs, for potential and actual overruns, are shown in Figure 6.8 and Figure 6.9 for the 10-task sets, and Figure 6.10 and Figure 6.11 for the 100-task sets. The values used to generate these graphs are shown tabulated in Appendix 2.

From the graphs, it can be observed that there is some initial variation at low cycle counts. This variation is a combination of the simulation not necessarily having run through one critical instant by this point, and also of the learning mode having had insufficient time to fully learn the characteristics of the task set being evaluated. However for all task set sizes and across all configurations, the results give the impression that they will, given a large enough simulation time, converge towards a single accuracy value. Given that a compromise between total simulation time and consistency of results needs to be reached, it is therefore necessary to establish an ideal simulation length and number of simulation runs to give consistent results.

In order to derive a suitable set of values, the outputs across the whole evaluated range are compared against that obtained from the maximum simulation time (i.e. ten runs and one million cycles). It is assumed that this particular accuracy value,

owing to having been derived from the greatest overall simulation time, is the most accurate value obtained, the difference from this value across the entire results set can be considered to give an idea of the accuracy of using a shorter overall simulation run, compared with allowing the simulation to run for the full amount of time.

These differences are summarised in the graphs shown in Figure 6.12 and Figure 6.13 for the 10-task sets, and Figure 6.14 and Figure 6.15 for the 100-task sets. The graphs show the percentage variation in accuracy values across the entire results set, and as with the absolute values, tables containing the percentage variation data can be found in Appendix 2. It can be seen from the graphs that the accuracy values converge rapidly as the number of cycles increases. From the tables in Appendix 2 it can be observed that in all except the case of actual overruns for the 100-task sets, by the time the cycle count reaches the order of 250,000 cycles the accuracy value obtained has stabilised to within 1% of the final value obtained. Even in this case, the value is only around 1.5% removed from the final value. As task sets containing 100 tasks are at the maximum extent of what is anticipated will be examined, this level of accuracy can still be considered to be sufficient, and therefore a simulation length of 250,000 cycles is adequate.

These values are averaged across all configurations and there may be cases where, for some configurations, the output is not perfectly stable after this point, but overall it appears that a stable output value – i.e. within 1% of the final value – is reached in the majority of cases within the first 250,000 simulation cycles.

It is also apparent that it is not necessary to perform 10 simulation runs in order to obtain a consistent result – although it is necessary to perform more than one. Particularly from the graphs of the 10-task results, it can be seen that there is a considerably greater variation in the results when only one run is used, than when the output of five or ten runs are combined, however there is less noticeable variation between the five and ten run results. By examining graphs and the raw data, it can be

see that at least three runs are required to give consistent results. In order to increase the consistency, it may be better to conduct five simulation runs.

Therefore, a set of simulation parameters for future investigation can be derived from these initial experiments, the results of which show that five runs, each consisting of 250,000 simulation cycles, are required to give consistently accurate results. Therefore, all further experiments will be conducted according to these parameters.

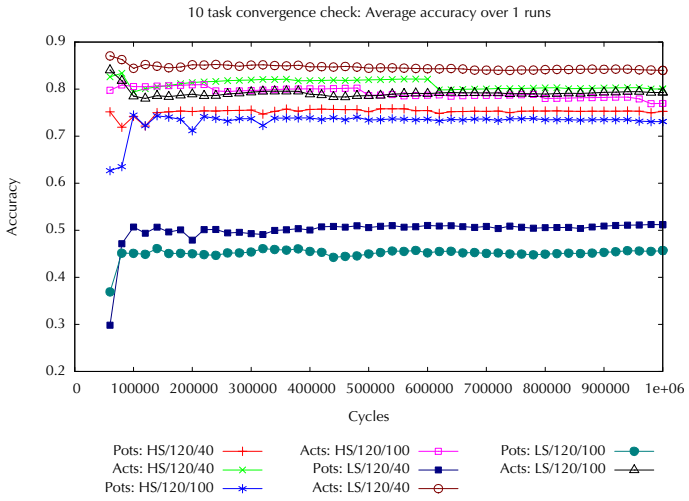


Figure 6.2: 10-task convergence over 1 run

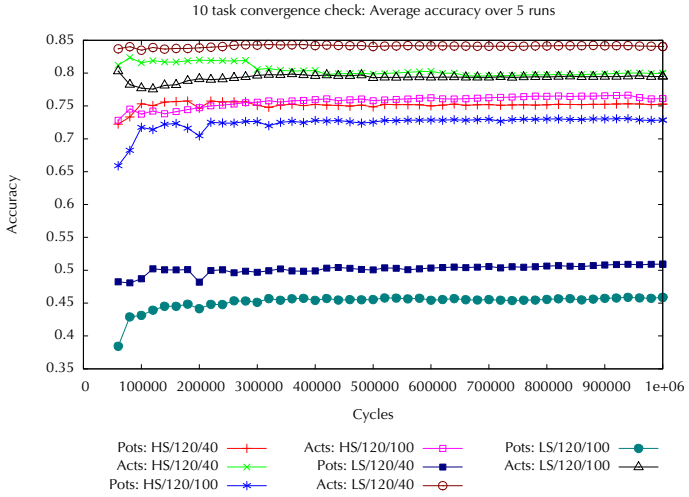


Figure 6.3: 10-task convergence over 5 runs

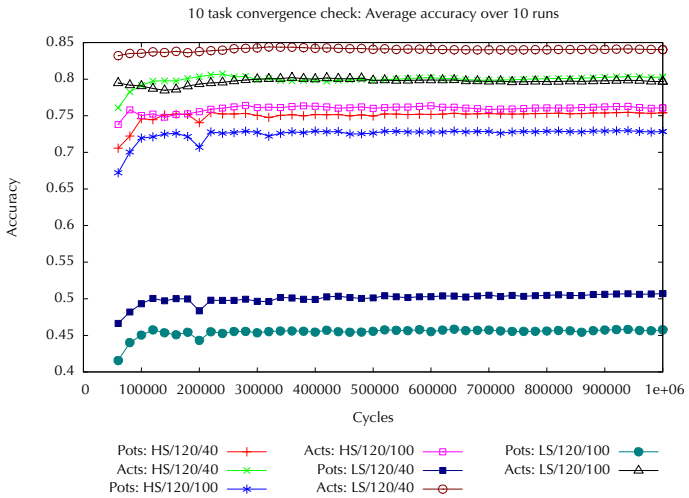


Figure 6.4: 10-task convergence over 10 runs

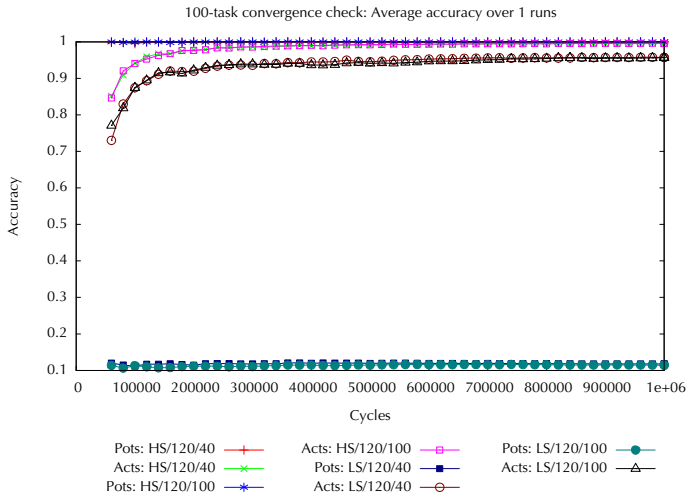


Figure 6.5: 100-task convergence over 1 run

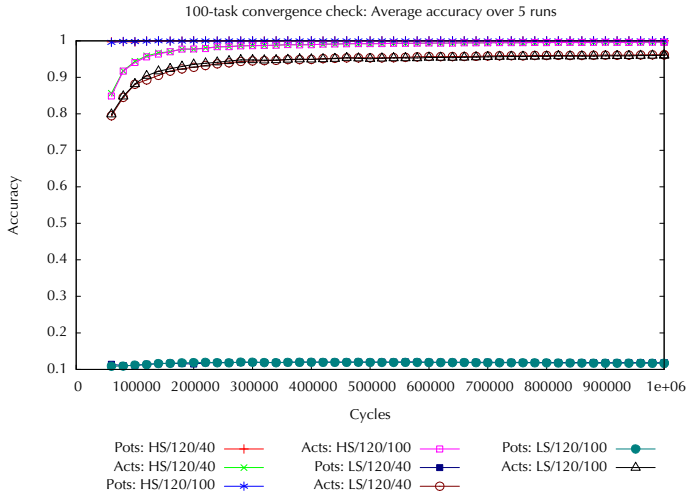


Figure 6.6: 100-task convergence over 5 runs

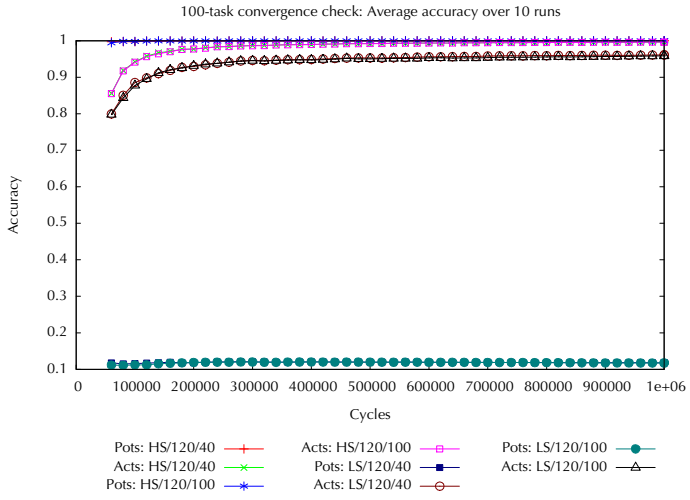


Figure 6.7: 100-task convergence over 10 runs

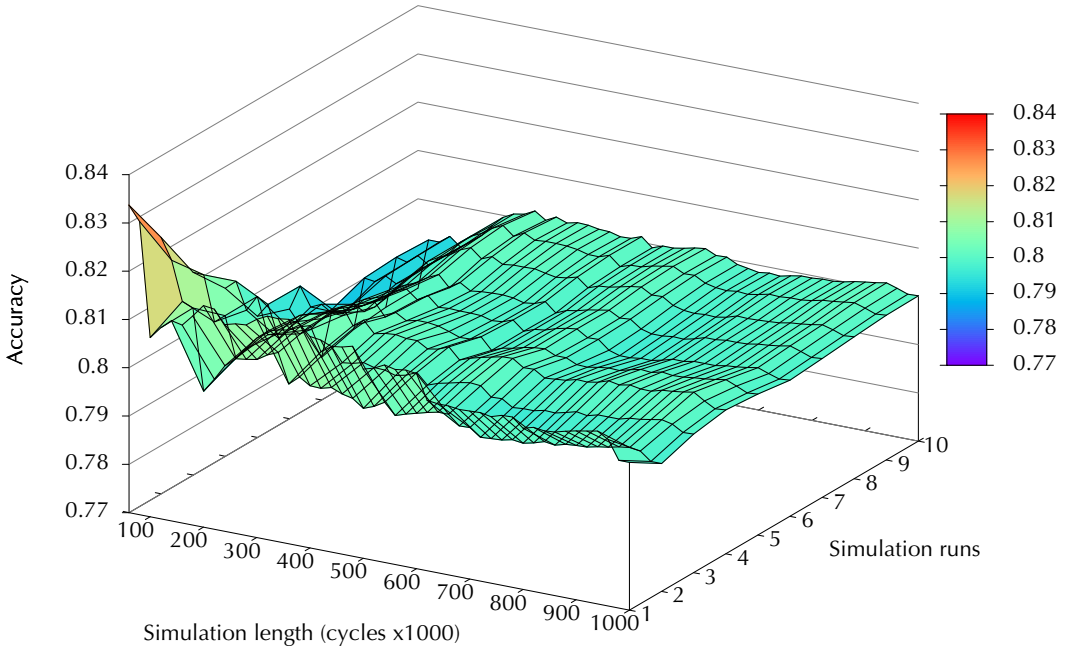


Figure 6.8: Actual overrun accuracy over all test configurations: 10-task sets, absolute values

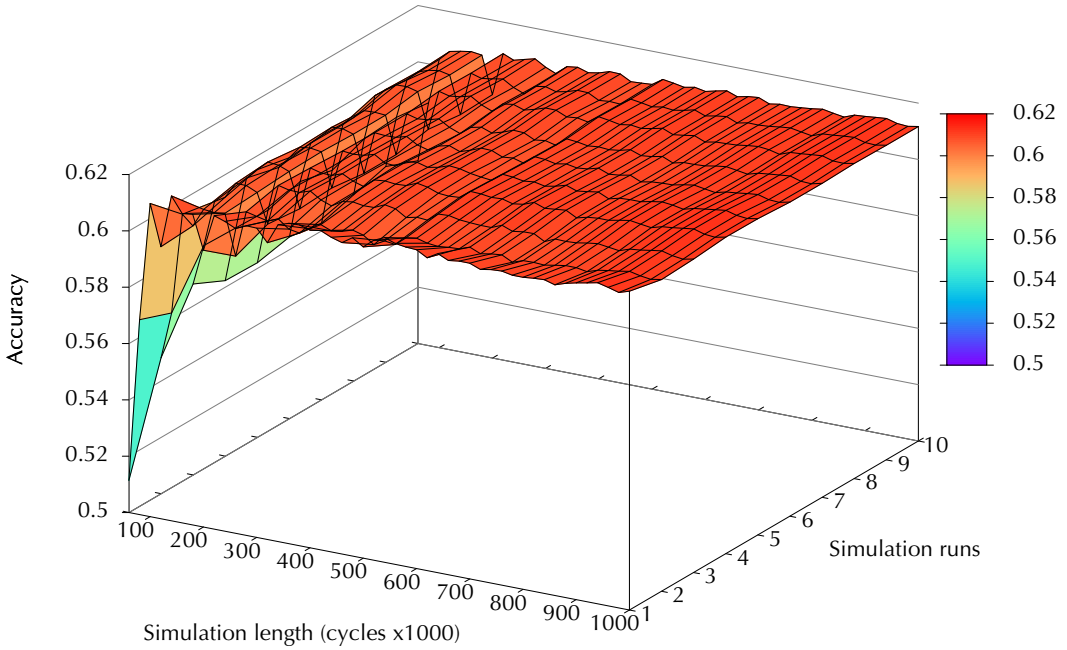


Figure 6.9: Potential overrun accuracy over all test configurations: 10-task sets, absolute values

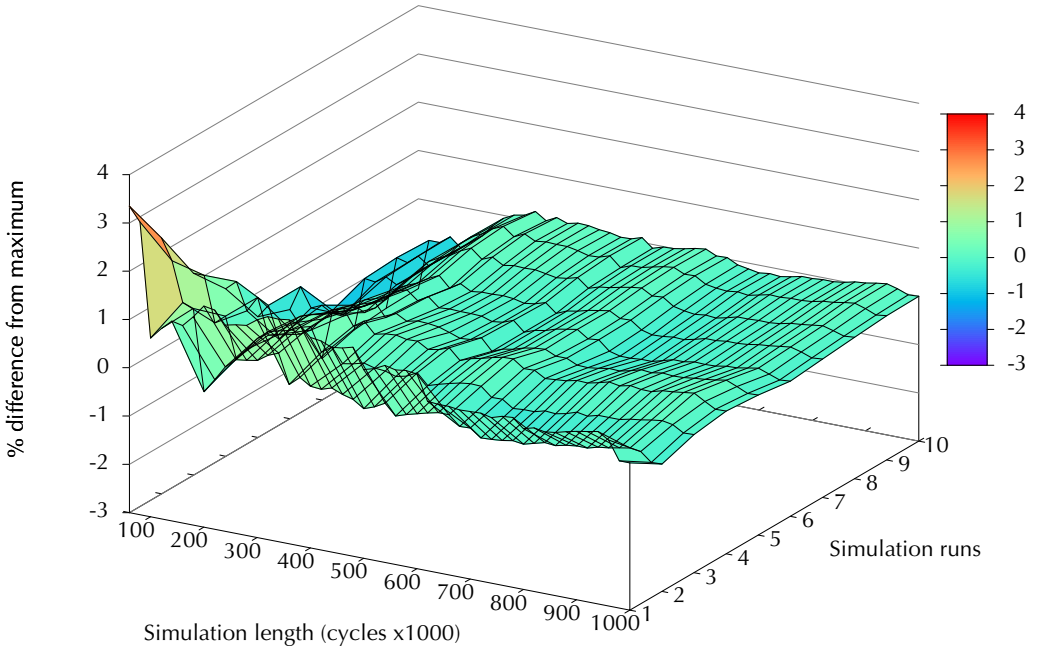


Figure 6.10: Actual overrun accuracy over all test configurations: 10-task sets, percentage variation from 10 runs/1,000,000 cycles

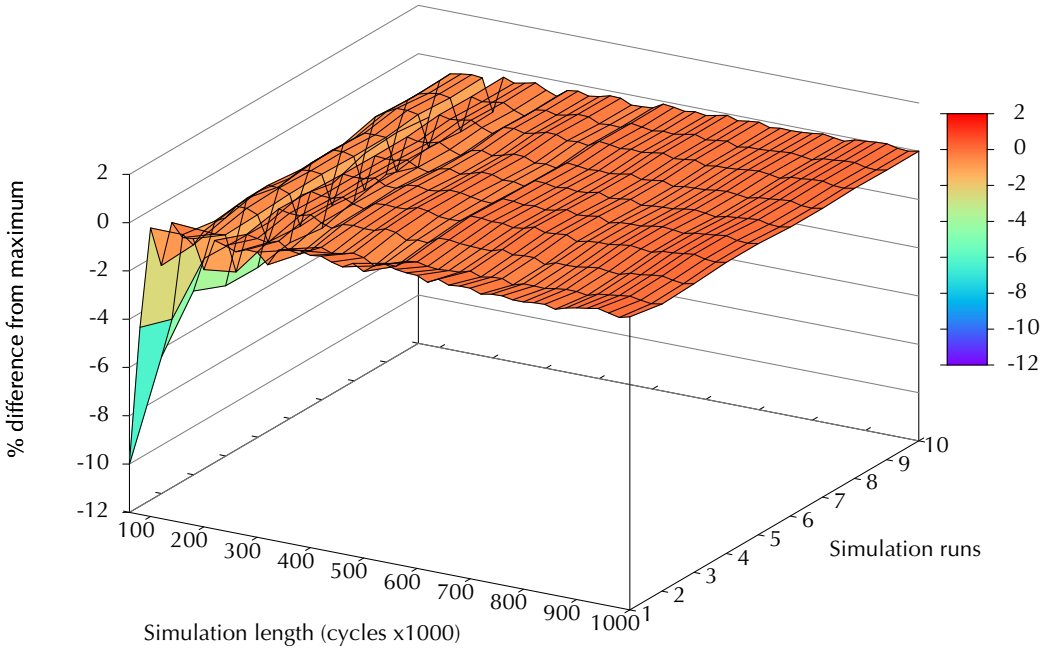


Figure 6.11: Potential overrun accuracy over all test configurations: 10-task sets, percentage variation from 10 runs/1,000,000 cycles

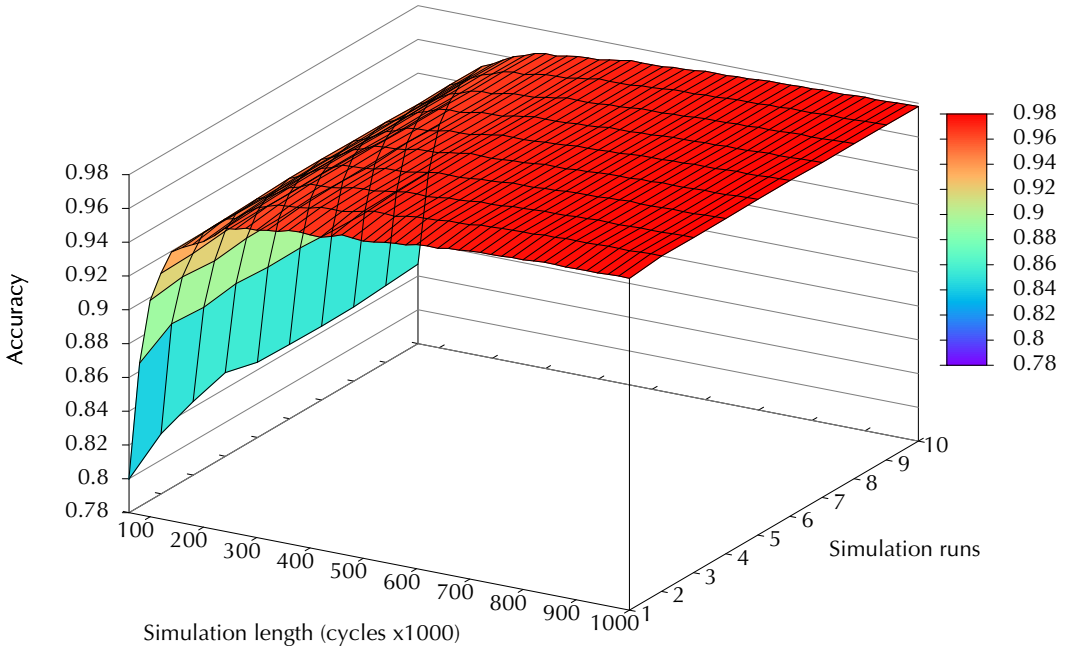


Figure 6.12: Actual overrun accuracy over all test configurations: 100-task sets, absolute values

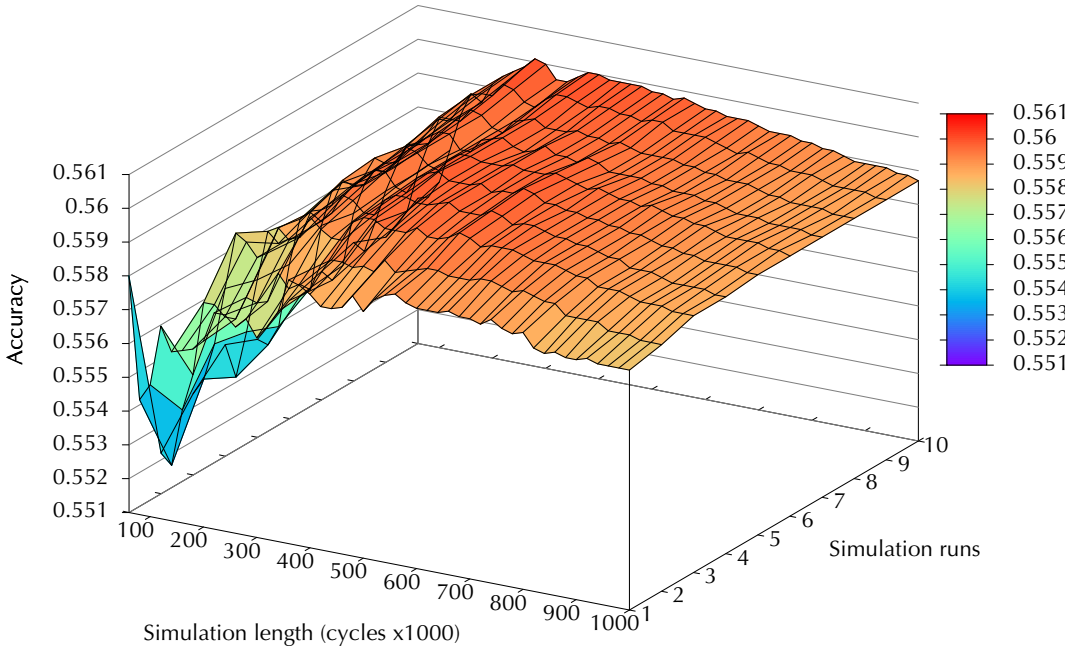


Figure 6.13: Potential overrun accuracy over all test configurations: 100-task sets, absolute values

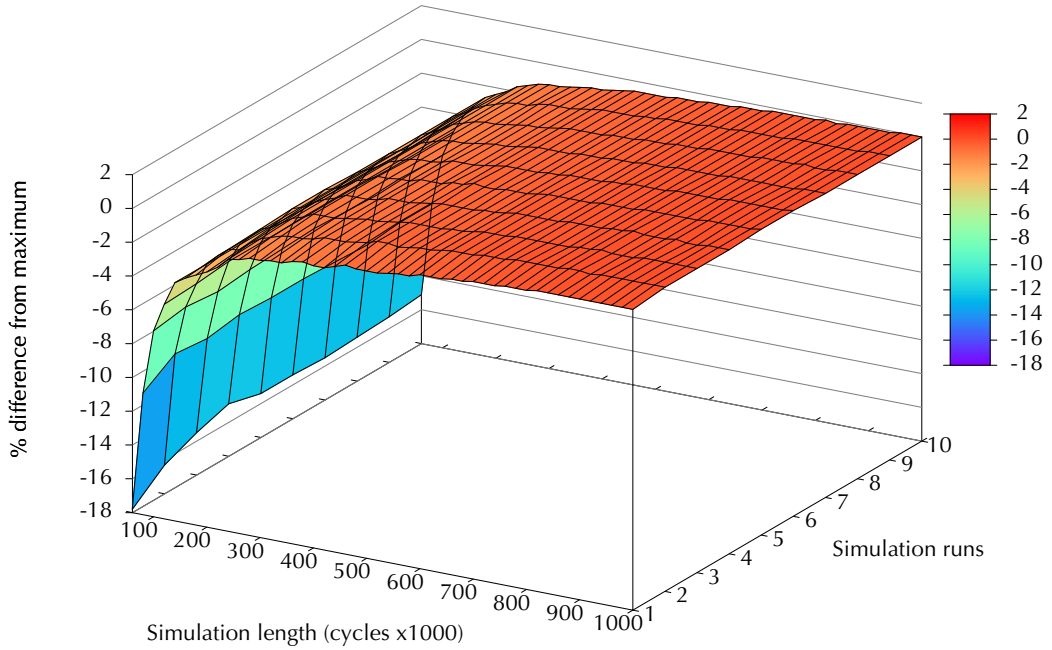


Figure 6.14: Actual overrun accuracy over all test configurations: 100-task sets, percentage variation from 10 runs/1,000,000 cycles

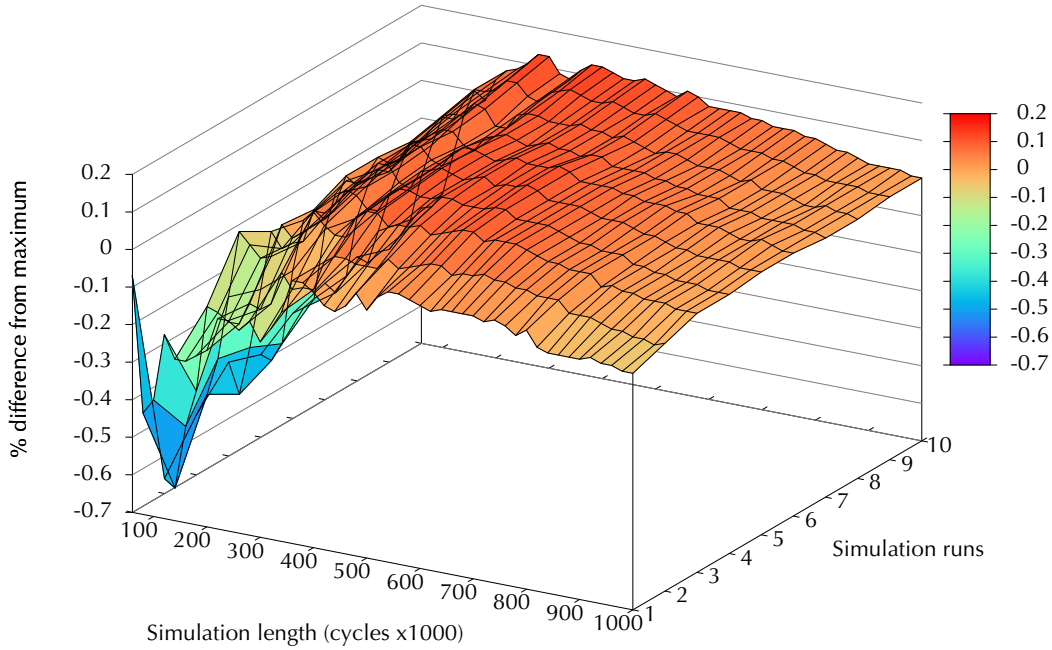


Figure 6.15: Potential overrun accuracy over all test configurations: 100-task sets, percentage variation from 10 runs/1,000,000 cycles

6.4 Further parameter investigations: DC population size

As outlined above, it is evident that more investigation into the application of the DCA to the deadline overrun problem is required. Of the parameters listed in Table 6.1, the most significant in terms of computational complexity and effectiveness of the overall algorithm is likely to be the size of the DC population. In the context of RTES, this is likely to be the parameter which has the greatest bearing on whether the algorithm can be viably implemented when subject to typically constrained resources.

In addition to varying the DC population size, investigation can also examine the effect of varying the proportion of the DC population which is evaluated each cycle. Altering the overall DC population size will affect both the computational complexity and the memory requirements of the algorithm, both of which are important considerations in embedded systems. The use of a DC "pool" selected from the main population reduces the effective DC population which must be evaluated each cycle, and thus also reduces the computational overhead. However, the retention of a large overall DC population may afford greater accuracy in the algorithm's output. In systems where the constraining factor is CPU capacity rather than memory availability, the use of such a DC pool mechanism may be advantageous.

Given the variation of these parameters, it is also important to consider the presence of a potential relationship between them, such that appropriate values for both can be selected which afford the best trade-off between accuracy, memory usage and computational complexity.

6.4.1 DC population size: effect on DCA accuracy

In order to investigate the effect of population size on the accuracy of the DCA, each sample system is evaluated multiple times, with DC population sizes ranging from 10 to 160 DCs. For each combination of task set and population size, multiple

evaluations are carried out with different percentages of the total DC population being evaluated each cycle, from 5% to 40%. This percentage of DCs evaluated each cycle is referred to as the "pool ratio".

In order to verify the general applicability of the DCA across a wide range of problems, these experiments will be conducted across a range of task sets, with sizes ranging from 10 tasks to 120 tasks. This will evaluate the effectiveness of the DCA across a range of task sets which are more representative of those which would actually be employed in typical RTES, and allow comparisons to be made across systems of differing complexities. For each task set size, the experiments will be conducted across ten randomly-generated task sets.

As with previous evaluations, the DCA is applied in both high-sensitivity and low-sensitivity variants, in order to investigate the effect of the weight/threshold combination over a range of task set complexities. For each set of parameters (i.e. DC population size, pool ratio, sensitivity and task set size), the DCA accuracy is calculated over five simulation runs, each 250,000 cycles long, as derived from the initial experiments in section 6.3.2.

The accuracy for each combination of parameters, using the calculation described in section 5.5.3.1 can then be compared across the range of task sets to gauge the effect of those parameters.

The following pages show summary graphs for each of the evaluated task set sizes: these are produced by averaging the accuracy and responsiveness figures across all ten generated task sets. In each graph, the plotted surface represents the mean accuracy or responsiveness as appropriate; the grids plotted above and below the surface represent the mean value plus or minus the standard deviation of the data used to produce the plots. Where the grids and the surface converge indicates a standard deviation close to zero.

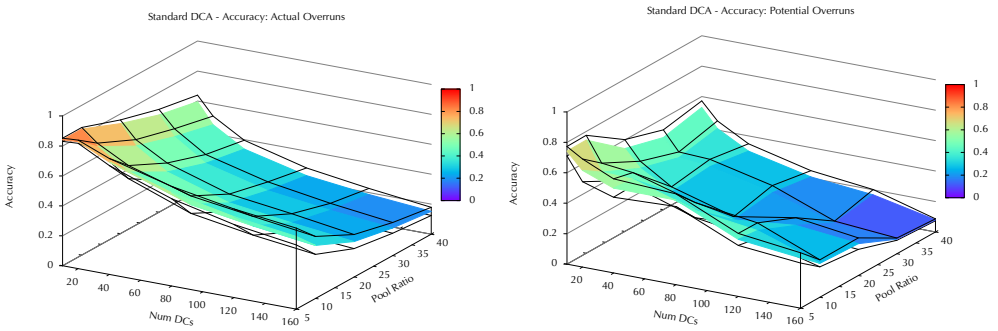


Figure 6.16: Accuracy compared with DC population and pool sizes (10 tasks, high sensitivity) – actual overruns (left) and potential overruns (right)

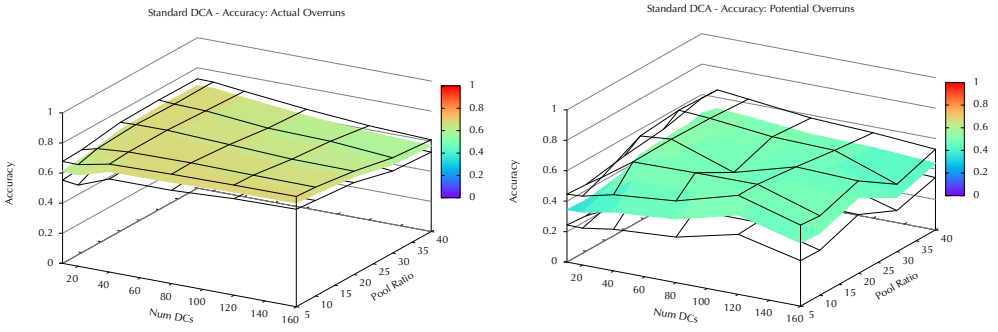


Figure 6.17: Accuracy compared with DC population and pool sizes (10 tasks, low sensitivity) – actual overruns (left) and potential overruns (right)

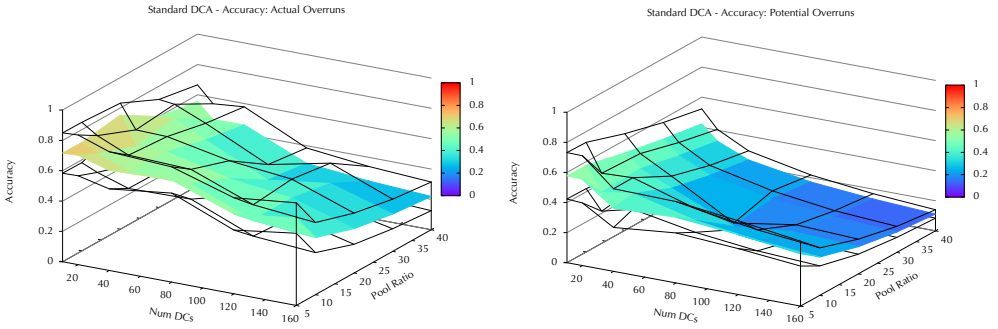


Figure 6.18: Accuracy compared with DC population and pool sizes (20 tasks, high sensitivity) – actual overruns (left) and potential overruns (right)

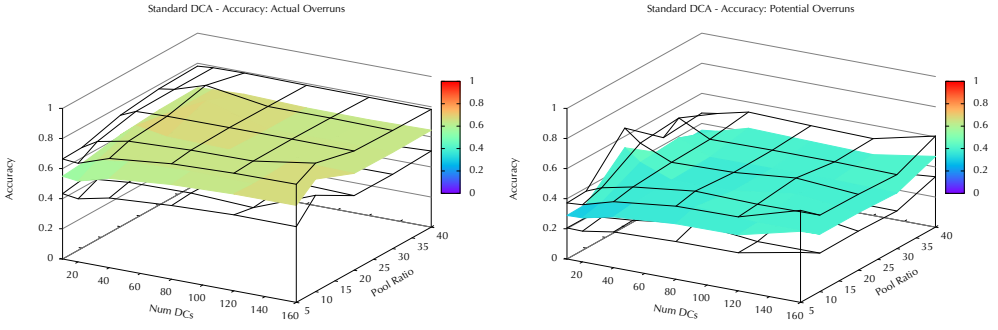


Figure 6.19: Accuracy compared with DC population and pool sizes (20 tasks, low sensitivity) – actual overruns (left) and potential overruns (right)

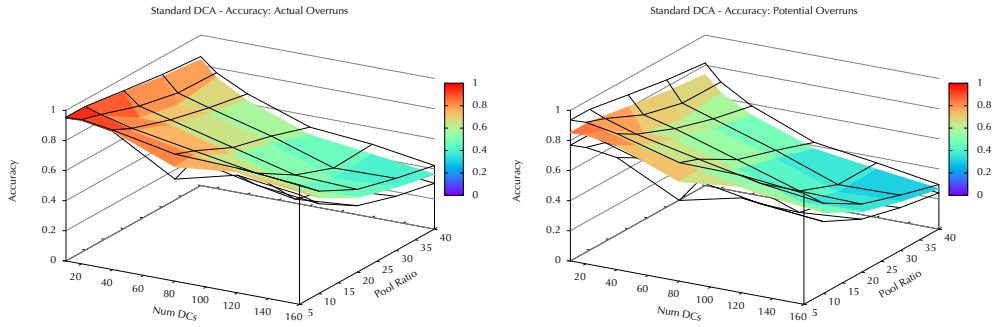


Figure 6.20: Accuracy compared with DC population and pool sizes (40 tasks, high sensitivity) – actual overruns (left) and potential overruns (right)

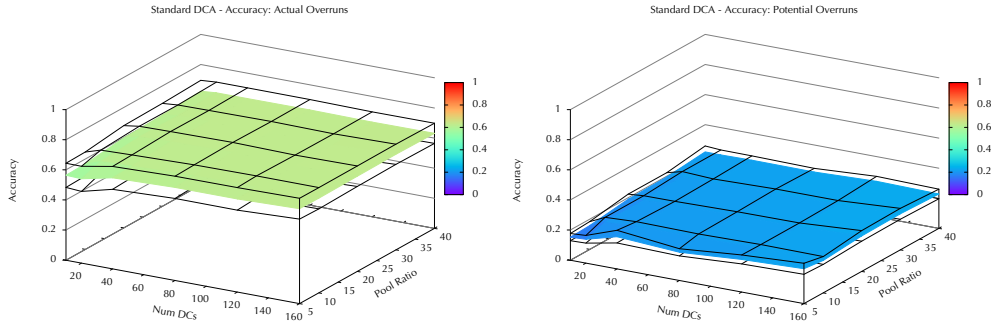


Figure 6.21: Accuracy compared with DC population and pool sizes (40 tasks, low sensitivity) – actual overruns (left) and potential overruns (right)

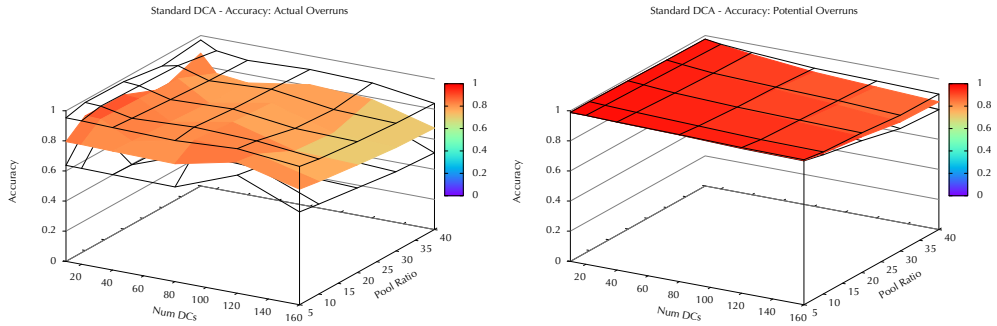


Figure 6.22: Accuracy compared with DC population and pool sizes (120 tasks, high sensitivity) – actual overruns (left) and potential overruns (right)

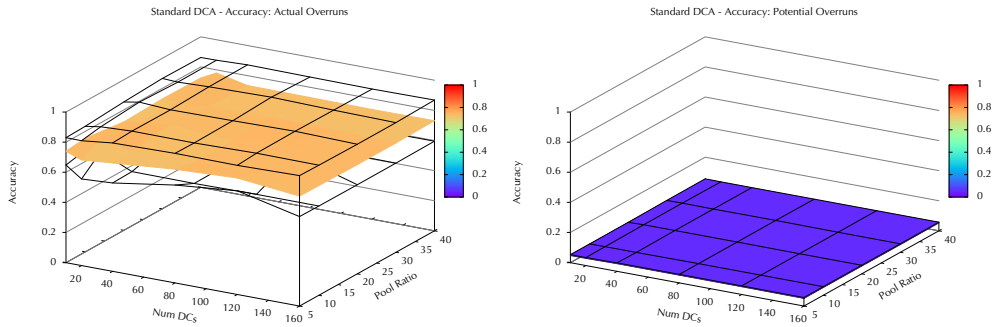


Figure 6.23: Accuracy compared with DC population and pool sizes (120 tasks, low sensitivity) – actual overruns (left) and potential overruns (right)

For reasons of space, graphs and tables for the individual results sets are omitted here, but can be found in the on-line results archive, details of which are included in the appendix at the end of this document.

Considering the overall results across task sets of all sizes, the results obtained here show similar initial patterns to those obtained in the initial testing in chapter 5: the accuracy of the detection of actual overruns is consistently high, while the prediction of potential overruns is not so successful. Across all the evaluated task sets, there is at least one combination of DCA parameters where the average overall accuracy for potential overruns is at least 50%, and at least one combination of DCA parameters where the average overall accuracy for actual overruns is at least 50%.

As the number of tasks increases, it appears that the performance of the DCA across the sampled task sets is dependent on its sensitivity. In particular, the high sensitivity variant appears to perform progressively better as the number of tasks increases, with respect to both potential and actual overruns, while the performance of the low-sensitivity variant decreases, particularly with respect to the prediction of potential overruns.

This effect can be attributed to the increase in the overall task set size, which brings with it a corresponding increase in the number of tasks in the run queue. The effect of a larger run queue will be to suppress detection of a potential overrun which only affects small proportion of the total in the run queue at that time. Conversely, the occurrence of an actual overrun often causes a number of other tasks to also experience potential overruns. The combined effect of multiple tasks experiencing overrun conditions will result in a higher level of danger signals being present in the system, and a resultant higher rate of detection.

In the case of the low-sensitivity variant of the DCA, the effect of this suppressive behaviour will be greater than with the high-sensitivity variant, as multiple consecutive detections of danger are required in order to cause maturation of a DC.

Where only a small proportion of the tasks present in a system experience overrun conditions, the presence of a greater number of tasks in the run queue will result in a smaller ratio of danger signals to safe signals being present when the DCs are evaluated, leading to a lower overall level of detection.

This will be further compounded by the use of the DC pool – for small pool ratios, it is less likely that any one DC will be selected and updated for two consecutive cycles. By the time a DC is selected from the pool and evaluated a second time, it is reasonably likely that transient danger will have passed without being reported. The high sensitivity variant, where a single instance of danger is sufficient to trigger DC maturation and thus a report of danger, does not experience the same effect.

From the graphs, it is evident that the DC population size has an effect on the overall accuracy of the DCA. The trends shown are broadly similar across the range of task sets, regardless of the number of tasks present.

The overall effect appears to be dependant on the sensitivity of the DCA. For the low-sensitivity version, the accuracy increases as the DC population size increases. However, the accuracy of the high-sensitivity variant decreases as the population size grows. Although there is clearly a relationship, the variation in accuracy does not appear to be linearly proportional to the number of DCs in either case.

6.4.2 Accuracy results – further analysis

In order to further investigate the behaviour of the algorithm in terms of accuracy, it is necessary to examine the output in more detail. The accuracy as shown in the graphs is calculated as outlined in section 5.5.3.1, and is an overall value incorporating true negatives, false negatives and false positives, as well as the true positive rate. By examining these components of the accuracy value across the range of evaluated task sets and DCA configurations, we can gain a better understanding of the behaviour of the algorithm across these scenarios.

For each task set size and sensitivity combination, three components of the overall accuracy can be analysed: a true positive rate for potential overruns, a true positive rate for actual overruns, and a false positive rate across the entire run (a false positive cannot be categorised as being a potential or an actual overrun, as no overrun has occurred). These results are shown in Figure 6.24 to Figure 6.31: as well as the true positive and false positive rates outlined above, each set also includes an overall accuracy graph for reference.

Examining the true positive rate across all task sets, it can be seen from the graphs that the high sensitivity variant of the DCA performs better than the low-sensitivity variant in terms of actually detecting the presence of actual overruns or predicting the occurrence of potential overruns.

Across all task set sizes and configurations, the detection of actual overruns by the high-sensitivity DCA is consistently over 80% accurate, and the prediction of potential overruns is consistently over 60% accurate. This rate of detection does not appear to be significantly affected by the alteration of the DCA population size, or pool ratio.

Considering the low-sensitivity DCA variant, it appears that, at low population sizes and pool ratios, increasing the DC population size does have a small positive effect on the rate of true positives. However, this effect is considerably less significant on the overall results than the size of the task set itself: it is evident from examining the results obtained from the 120-task systems that the low-sensitivity DCA performs poorly with larger task set sizes, particularly with respect to the prediction of potential overruns.

However, when considering the occurrence of false positives, the performance of the two variants is reversed: here, the low sensitivity version consistently records a low incidence of false positives, while the high sensitivity version suffers from an increase in false positives as the population size, both actual and effective, increases.

It is this increased number of false positives which causes the overall accuracy for the high sensitivity DCA to fall as the DC population size increases.

The increasing rate of false positives corresponding to an increased DC population size appears to counteract the suggestion that the overall accuracy of the DCA is being affected by interference from the remainder of the run queue. If it was the case that the presence of additional "safe" tasks was suppressing the maturation of DCs, a decrease in true positives would be anticipated, but this would not necessarily result in an increase in false positives.

The use of a DC pool mechanism provides a logical explanation for the lack of increase in false positives in the low sensitivity variant. As discussed above, the use of the pool ratio decreases the likelihood of a DC being selected over two consecutive cycles, and so reduces the overall probability that a DC will be stimulated sufficiently for its output signal value to reach the maturation threshold.

The effect of this is to suppress the reporting of transient danger signals, i.e. those which are not present for a significant amount of time. As established previously, a consequence of this is a decrease in the true positive ratio of correctly detected potential overruns, due to the failure of the DCs to detect transient potentials. However, this suppressive effect also appears to reduce the incidence of false positives, which can be presumed to be the result of occasional spurious danger signals. Although the reduction in the true positives is a definite disadvantage, this is in some ways counterbalanced by the reduction in false positives.

This suppressive effect will diminish as the absolute size of the DC pool increases: this can be seen in Figure 6.25 (low-sensitivity 10-task) where the false positive ratio can be seen to increase steadily, although not as rapidly as the high-sensitivity variant in Figure 6.24. If a pool ratio of 100% was to be evaluated it is likely that the recorded incidence of false positives in the low-sensitivity variant would have a value approaching that of the high-sensitivity variant's false positive rate.

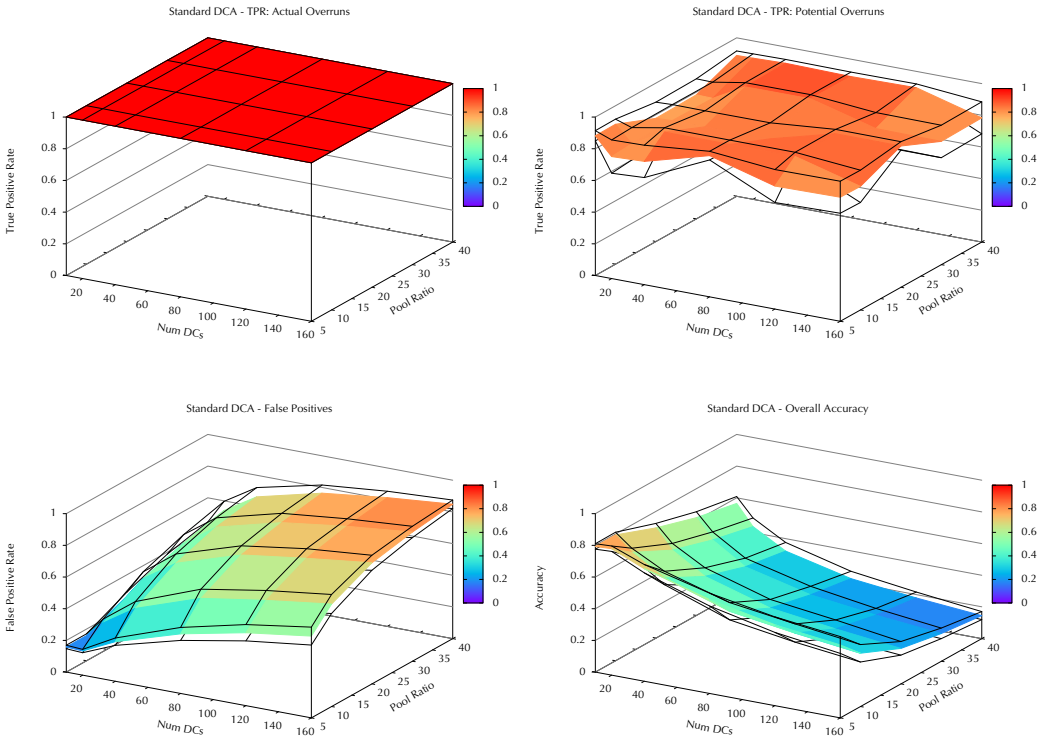


Figure 6.24: Accuracy component results: 10-tasks, high sensitivity: actual TPR (top left); potential TPR (top right); FP (bottom left); overall (bottom right)

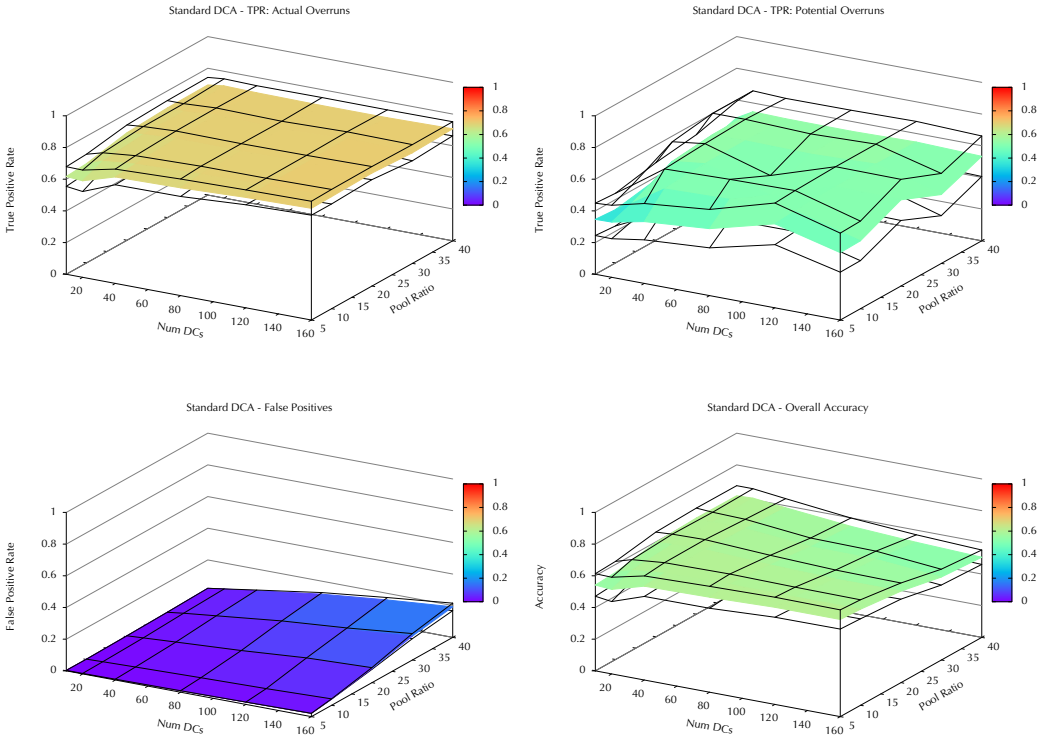


Figure 6.25: Accuracy component results: 10-tasks, low sensitivity: actual TPR (top left); potential TPR (top right); FP (bottom left); overall (bottom right)

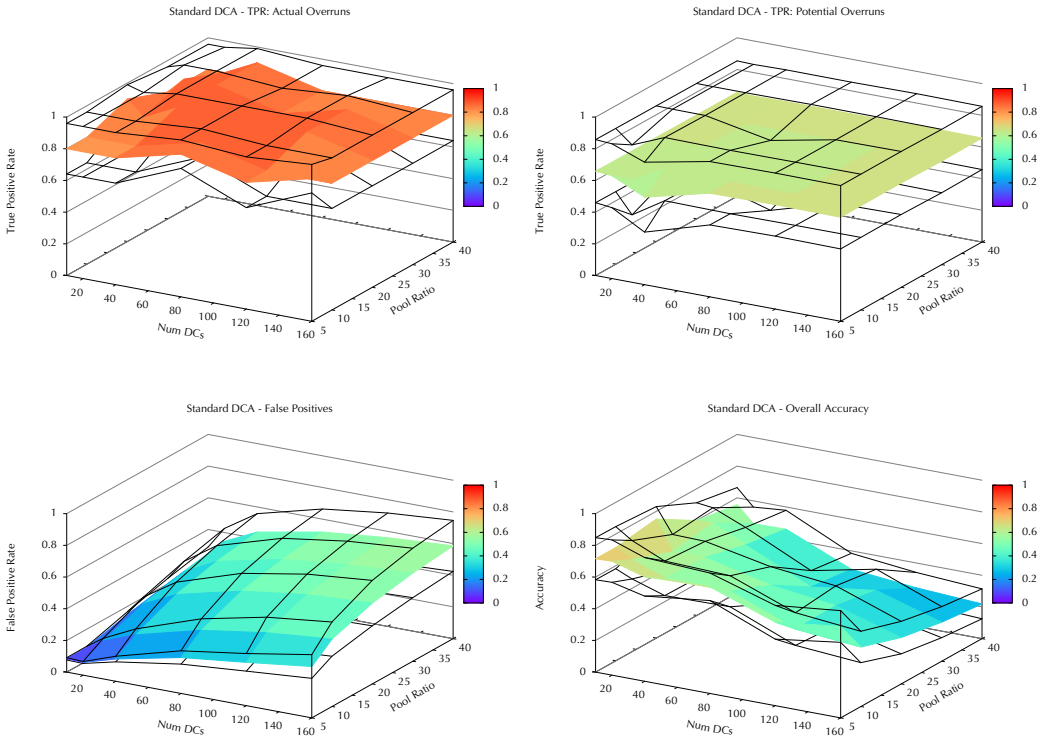


Figure 6.26: Accuracy component results: 20-tasks, high sensitivity: actual TPR (top left); potential TPR (top right); FP (bottom left); overall (bottom right)

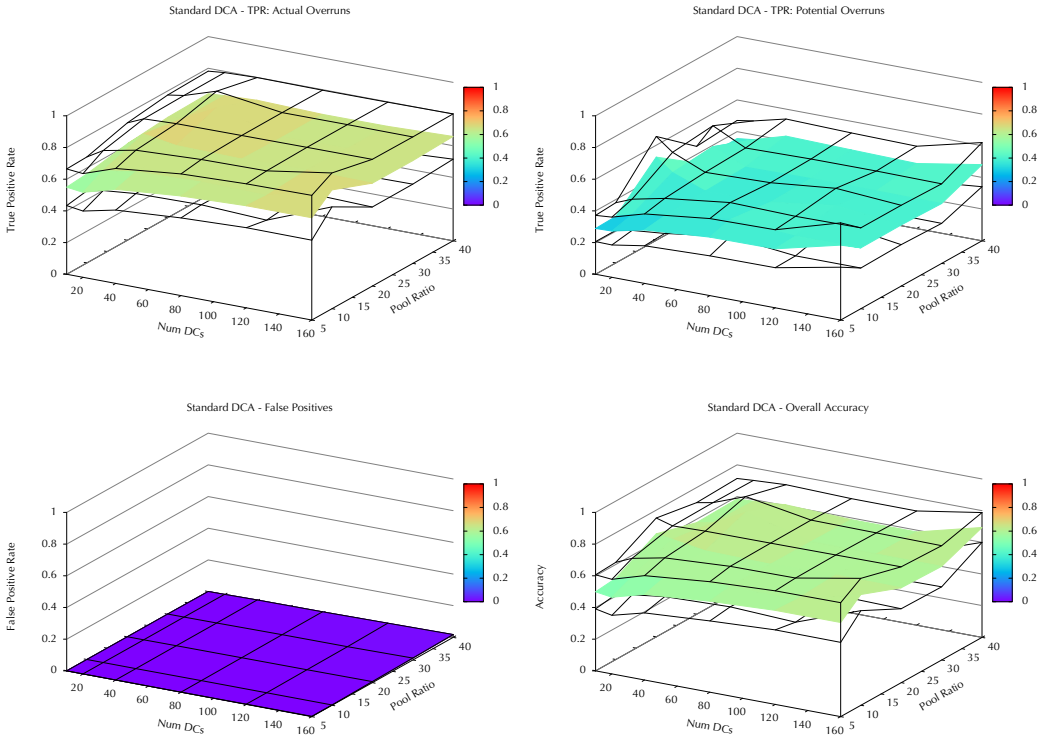


Figure 6.27: Accuracy component results: 20-tasks, low sensitivity: actual TPR (top left); potential TPR (top right); FP (bottom left); overall (bottom right)

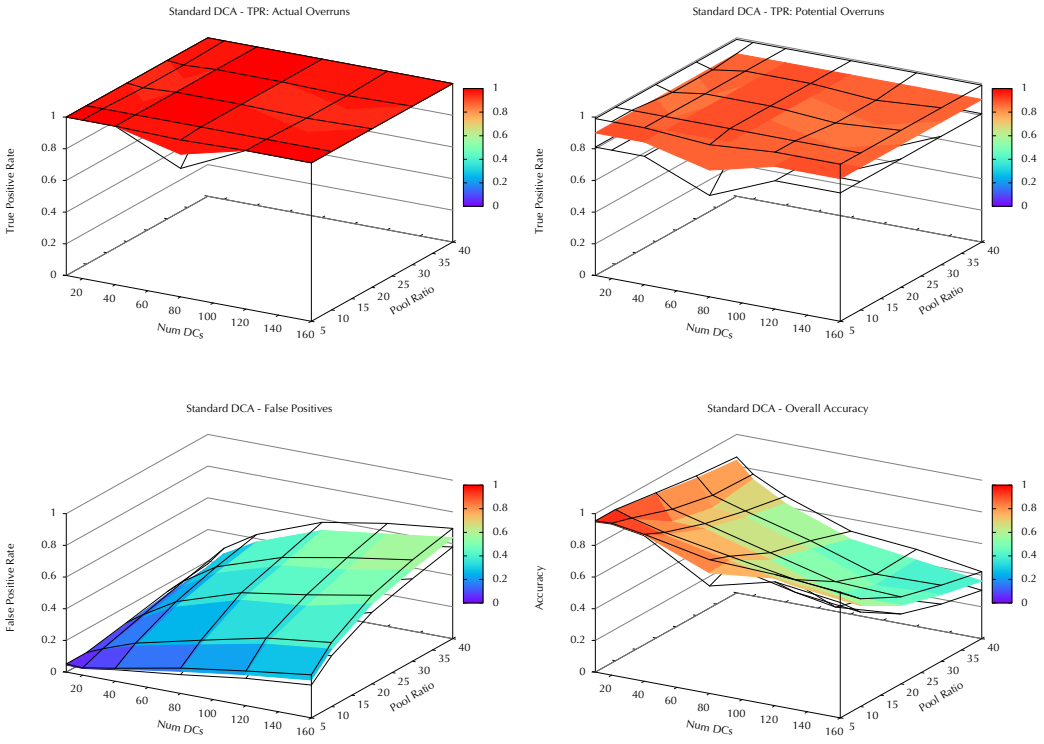


Figure 6.28: Accuracy component results: 40-tasks, high sensitivity: actual TPR (top left); potential TPR (top right); FP (bottom left); overall (bottom right)

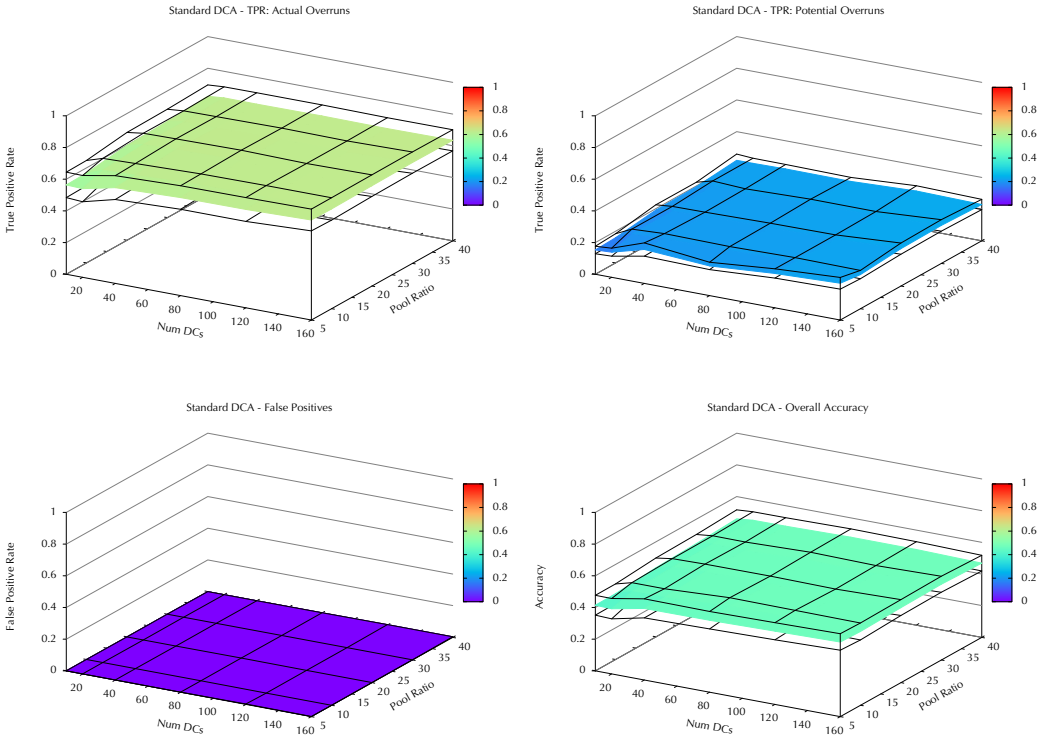


Figure 6.29: Accuracy component results: 40-tasks, low sensitivity: actual TPR (top left); potential TPR (top right); FP (bottom left); overall (bottom right)

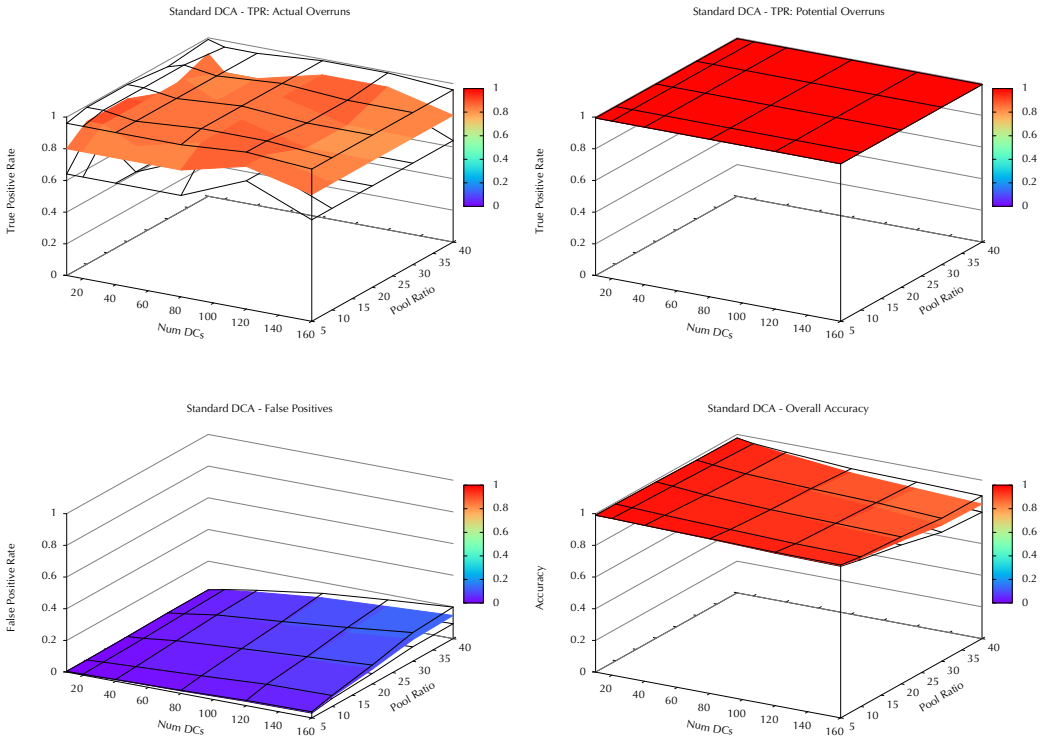


Figure 6.30: Accuracy component results: 120-tasks, high sensitivity: actual TPR (top left); potential TPR (top right); FP (bottom left); overall (bottom right)

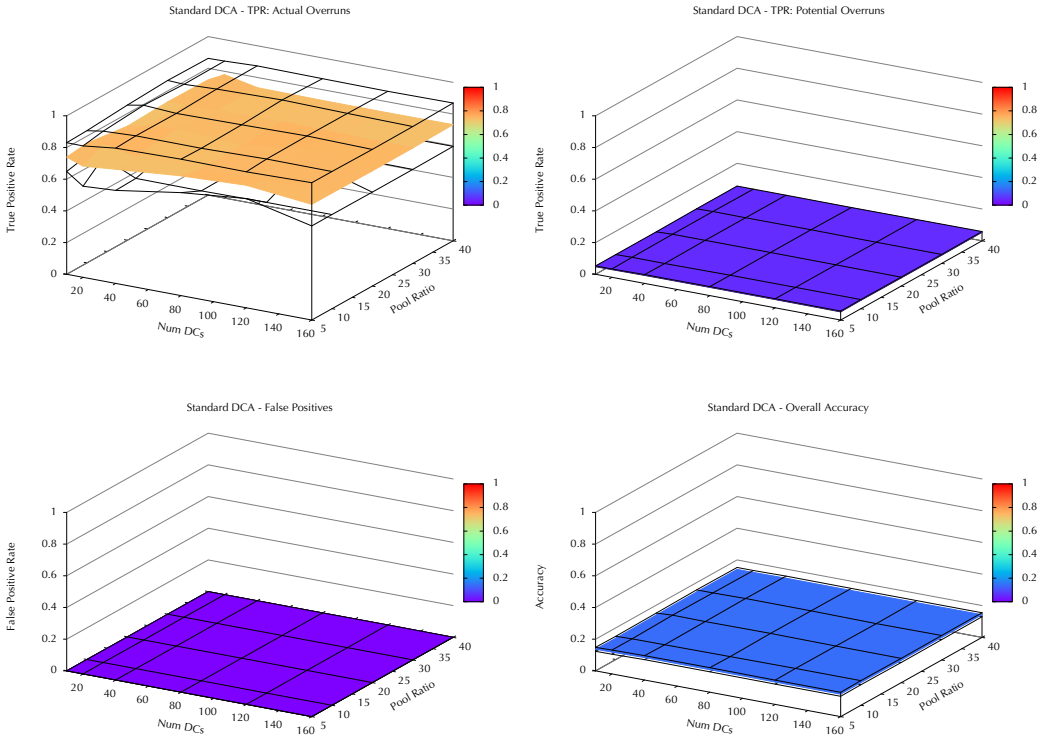


Figure 6.31: Accuracy component results: 120-tasks, low sensitivity: actual TPR (top left); potential TPR (top right); FP (bottom left); overall (bottom right)

6.4.3 DC population size – effect on DCA responsiveness

In addition to the DCA accuracy, the initial investigation of the DCA in chapter 5 also investigated the responsiveness of the algorithm. It therefore follows that the algorithm's responsiveness should be investigated across the range of evaluated population sizes, in order to determine whether there is a correlation between the two, or whether there is a particular combination of population and pool ratio which gives the highest level of overall responsiveness.

In particular, any variation in responsiveness across the parameter range will need to be taken in context with the overall accuracy, as it may be the case that any variation observed in the responsiveness of the DCA may not be mirrored in its accuracy. In such a case, it may be necessary to trade-off responsiveness for the sake of accuracy as the primary aim of applying the DCA to the deadline overrun is to detect and predict overruns accurately. Therefore, a parameter set which is highly responsive but has a low overall accuracy is less preferable to one which is less responsive but attains a higher level of accuracy.

For the purposes of this section, the individual responsiveness values for each overrun are calculated as outlined in section 5.5.3.2. These individual values are combined as before to produce an overall responsiveness value across all the simulation runs made for a particular configuration. These values are then plotted against the DC population size and pool ratio, in order to gauge the effect that these parameters have on the responsiveness across the parameter space.

Graphs for each task-set size and DCA sensitivity variant are shown in Figure 6.32 - Figure 6.39. The overall DCA accuracy across the same parameter space is also shown alongside each responsiveness graph, in order that the responsiveness across the parameter range may be judged with consideration for the variation in accuracy.

It can be seen from the graphs that, across the range of evaluated task-set sizes, the responsiveness of the DCA is not particularly influenced by the DC population size

than the accuracy: overall, the responsiveness remains at a reasonably level across all evaluated population sizes and pool ratios, with only small differences being observable at the lowest values of both parameters.

The lack of discernable variations mirrors the patterns seen previously with the true positive results: the decrease in overall accuracy levels generally being caused by an increase in false positive results, rather than a decrease in true positives. Given that false positives do not play a role in the calculation of the responsiveness (i.e. responsiveness is only calculated for true positives), they would not be expected to affect the responsiveness output.

There is greater variation in responsiveness across the different task-set sizes. Overall, the level of responsiveness exhibits similar behaviour to the overall level of accuracy with respect to task set complexity: as the task-set size increases, the responsiveness of the high-sensitivity variant increases. Conversely, the responsiveness of the low-sensitivity algorithm decreases as the task set size increases.

As with the variation in accuracy across task set sizes, the variation in responsiveness is likely to be due to the increase in the average number of tasks in the run queue which follows an increase in the task set size, and the combined effects of interference from other tasks in the run queue when only a small proportion of the queue experiences overrun conditions.

The lower levels of responsiveness from the low-sensitivity variant can be partially accounted for by the suppressive nature which is inherent in the algorithm's low-sensitivity configuration. Where more than one incidence of a danger signal is required to initiate DC maturation compared with a single incidence in the high-sensitivity variant, it is apparent that the responsiveness of the low-sensitivity variant will be lower than the high-sensitivity algorithm.

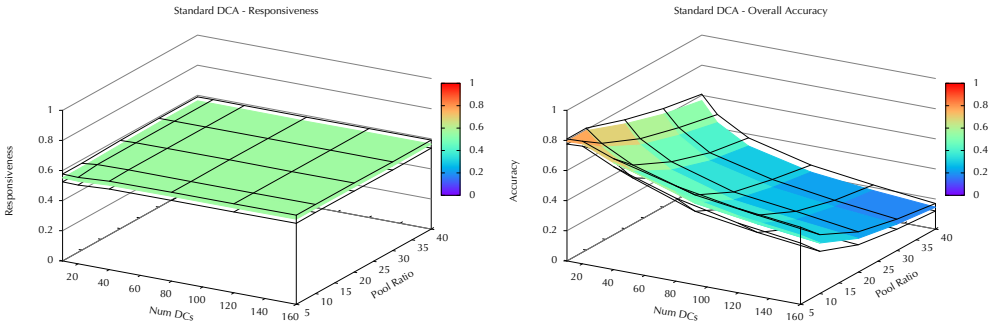


Figure 6.32: Overall responsiveness (left) and accuracy (right) against DC population size and pool ratio – 10 tasks, high sensitivity

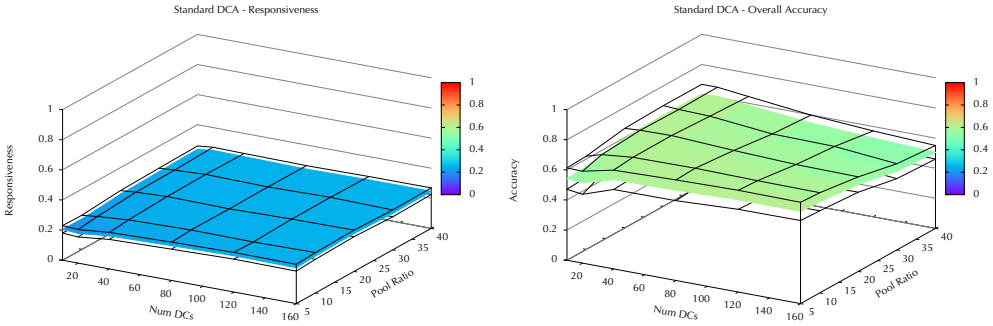


Figure 6.33: Overall responsiveness (left) and accuracy (right) against DC population size and pool ratio – 10 tasks, low sensitivity

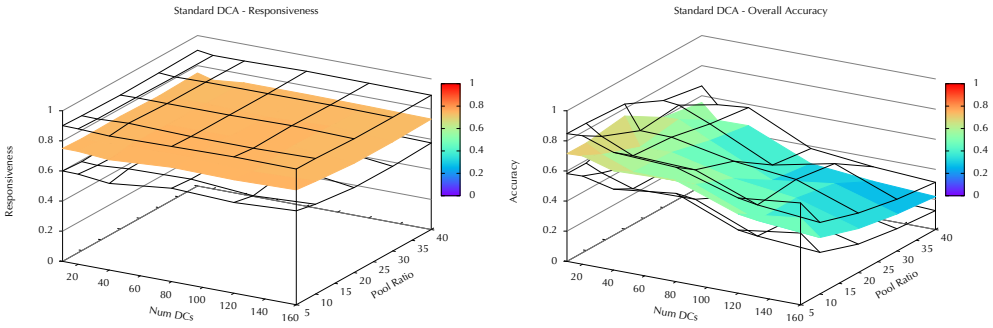


Figure 6.34: Overall responsiveness (left) and accuracy (right) against DC population size and pool ratio – 20 tasks, high sensitivity

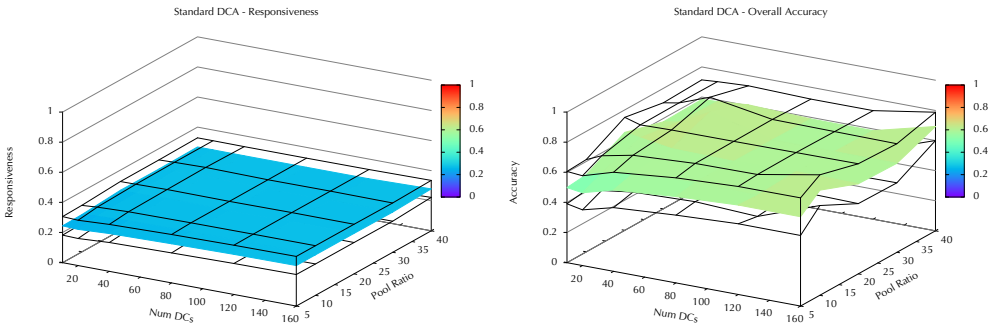


Figure 6.35: Overall responsiveness (left) and accuracy (right) against DC population size and pool ratio – 20 tasks, low sensitivity

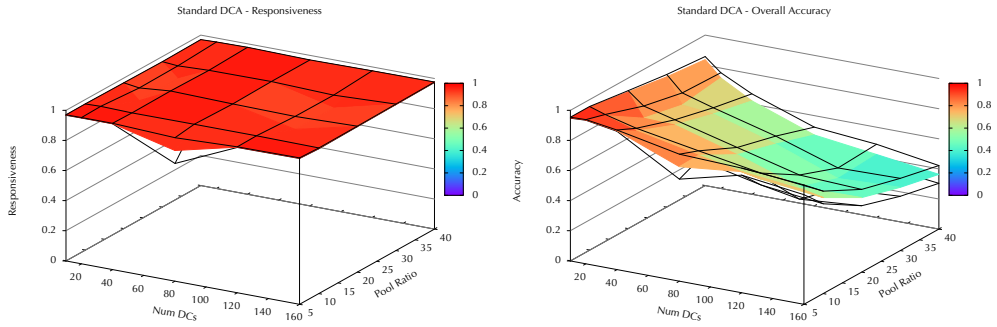


Figure 6.36: Overall responsiveness (left) and accuracy (right) against DC population size and pool ratio – 40 tasks, high sensitivity

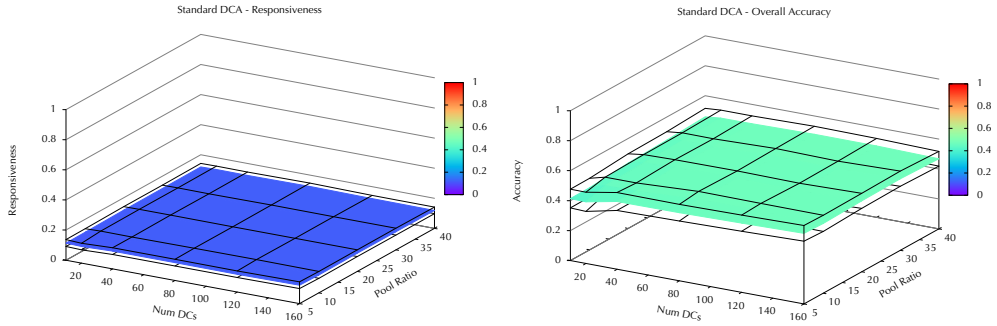


Figure 6.37: Overall responsiveness (left) and accuracy (right) against DC population size and pool ratio – 40 tasks, low sensitivity

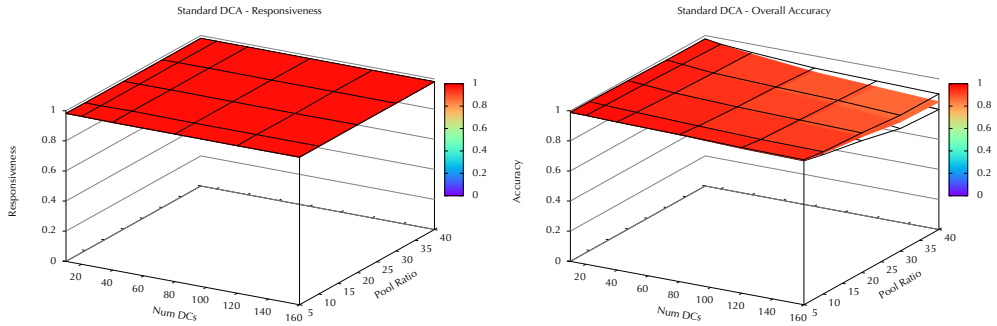


Figure 6.38: Overall responsiveness (left) and accuracy (right) against DC population size and pool ratio – 120 tasks, high sensitivity

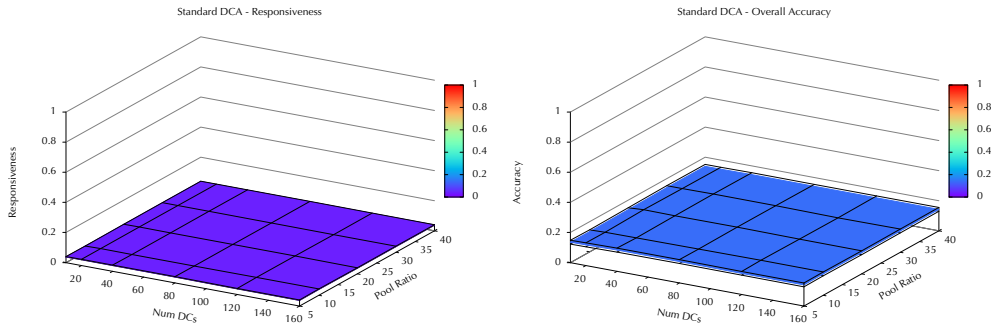


Figure 6.39: Overall responsiveness (left) and accuracy (right) against DC population size and pool ratio – 120 tasks, low sensitivity

It would not be unreasonable to expect the use of the DC pool to also have a suppressive effect on the responsiveness. If this was the case, an increase in the responsiveness as both the DC population size and pool ratio would be expected, for both sensitivities, similar to the increase seen in false positive rates. However, the graphs for either sensitivity variant of the algorithm show no discernable variation in the level of responsiveness across the evaluated DC population size and pool ratio parameter space.

Given the apparent lack of any variation in the responsiveness across the range of evaluated DC population and pool sizes, it must be concluded that the responsiveness of the DCA is not dependant on these parameters, although it does appear to be influenced by the weightings and thresholds applied to each DC.

6.5 Effect of DCA parameters – summary

From the graphs shown in the preceding sections, it can be seen that the DC population size does, to some degree, have an effect on the operation of the DCA. Across all the variants, the following trends can be seen:

As the DC population increases, there is a corresponding increase in the rate of true positive danger reports. This can be seen particularly in the results for the low-sensitivity variant of the algorithm.

As the DC population increases, there is also a marked increase in the number of false positive danger reports. This is particularly evident with the high-sensitivity variant of the algorithm, although at small task-set sizes a similar trend can also be seen for the low-sensitivity variant.

For the high-sensitivity variant of the algorithm, the overall effect of this increase in false positives is proportionally bigger than the increase in true positives over the same DC population size range. Consequently, the increase in false positives has a

significant adverse effect on the overall accuracy of the algorithm at higher DC population sizes.

The use of the DC pool concept shows interesting behaviour, particularly when coupled with the suppressive properties of the low-sensitivity variant of the algorithm. The use of small pool ratios significantly reduces the likelihood of any one DC being selected across two consecutive update cycles. When this is combined with the need for a single DC to experience more than one incidence of danger/PAMP signals, this leads to a level of suppressive behaviour which appears to be highly effective at reducing the rate of false positives, however at the same time it has a significant detrimental effect on the rate of true positive detections. The use of a pool ratio in this way has not been investigated before, and could provide a potentially useful way of refining the effectiveness of the DCA, when combined with appropriate weighting and threshold values, in order to give a high level of true positive detection combined with a low level of false positives.

The responsiveness of the DCA shows little variation across the range of evaluated DC population and pool sizes, although it is affected by the DCA sensitivity level and the size of the task set being evaluated. It is apparent that the most significant effect on the DCA responsiveness is caused by differences in the weights and thresholds used by the DCA and the complexity of the problem task set.

Although the rate of DCA accuracy is obviously affected by the DC population size and the DC pool ratio, the differences in the results obtained from the two variants of the DCA evaluated here, and the varying patterns of behaviour of those variants over the evaluated configuration space, suggest that, like the responsiveness rate, the accuracy rate is more significantly affected by the internal weights and thresholds allocated to each DC, rather than by the size of the DC population employed.

Further work is clearly necessary to establish the full effect of the remaining parameters, including weights and thresholds, on the overall effectiveness of the

DCA to this problem, but it appears likely that there will exist a combination of parameters which will allow a high rate of overrun detection, combined with a low rate of false positives. The results obtained so far show that the DCA is able to detect a large proportion of actual overruns, and in most cases predict a large proportion of potential overruns: given further parameter optimisation to reduce the rate of false positives, it should be possible to better these results.

However, a key aspect of incorporating the DCA to the deadline overrun is the ability of the algorithm to operate in the constrained resource environments often encountered in RTES. The next chapter will address this issue.

7

The DCA in RTES: resource considerations

In chapter 6 the effects of different task set sizes and different algorithm parameters were investigated. It was established that the DCA is reasonably effective at detecting actual overruns and predicting potential overruns across a wide range of task set sizes, and how the overall and effective size of the DC population affects the accuracy and responsiveness of the algorithm.

Given the purpose of this work is to provide a method for real-time anomaly detection in RTES, there is a need to conduct further investigations in order to determine the suitability of the algorithm when employed in a constrained resource environment. This chapter examines the complexity of the algorithm and the factors which affect that complexity. In addition to the overall complexity, the predictability of the algorithm is also examined. In order to reduce the overall complexity and improve the predictability of the problem, a variant of the DCA is proposed with a

number of adaptations made to allow it to be better suited for use in RTES development.

7.1 Implications of constrained resources

The development of many RTES, as discussed in chapter 1, is highly constrained by a number of factors. Particularly in the CE domain, the major constraints are those of cost, including both the one-off engineering costs associated with the design of the system, and the unit cost incurred in its manufacture [144].

These constraints have an effect on the implementation of fault-detection systems which are intended to be incorporated into a system in addition to the components which provide the system's actual functionality. Although the resources used by these systems are less of an issue where they are only employed during the development phase (where dedicated monitoring equipment can be used), it may be desirable for such a system to be included in every device that is manufactured, such that it can either run at all times, or can be activated as a diagnostic tool in the event of problems. The feasibility of this will be governed by the behaviour of the algorithm, and the resources required for it to execute. Given the need to reduce unit cost in the CE domain it is unlikely that there will be sufficient resources available to allow the inclusion of a resource-intensive fault prediction system.

7.1.1 Algorithmic complexity

The first important consideration when employing any algorithm in a computer system is the computational complexity of that algorithm [119]. This is particularly important when considering the addition of functionality into a constrained resource environment, such as those typically encountered during RTES development.

With most algorithms, there are two aspects of complexity which need to be considered: the amount of execution time it takes to complete on a dataset of a given size, and the amount of storage required by the algorithm during that time. It is

likely to be the case that in an embedded system, there will be relatively severe constraints on both of these resources.

It is not necessarily the case that there is a relationship between the execution time of an algorithm and its storage requirements: depending on the overall structure and behaviour of the algorithm. Some types of algorithm may require increased storage requirements in line with increased computation time, whereas others may be the opposite, trading off increased computation time for decreased storage requirements.

As an example, there are a variety of methods of sorting an array of numbers. Typical sorting algorithms, such as insertion sort, require little additional storage beyond that required to store the array being sorted, however the computation time required increases significantly as the size of the array increases. With certain types of data it is possible instead to use a bucket-sort algorithm: this reduces the computation time, at the expense of requiring more memory (and not being generally applicable to all data) [150].

This variation allows differing algorithms to be selected depending on the nature of the resource constraints. For example, if there is plenty of spare processing capacity but limited memory, insertion-sort can be used, but where the situation is reversed it may be more desirable to use a bucket-sort type algorithm instead.

When considering the complexity of algorithms, it is usual to evaluate them in terms of the size of the input data; in this way, the performance of that algorithm over a variety of input sets can be determined. In general, it will be that one aspect of an algorithm's complexity will dominate over the remainder: the algorithm's complexity is generally quoted as being "of the order of" this dominating factor. In general, therefore, algorithm complexity falls into one of the following categories [152]:

Table 7.1: Algorithmic complexity categories

Complexity category	Example execution time over n elements
Constant	k
Logarithmic	$k \log n$
Linear	$k n$
Linearithmic	$k n \log n$
Polynomial	$k n^2$
Exponential	k^n

It is of course possible for algorithms to have differing levels of complexity within each category, but in general the complexity of an algorithm will be considered as belonging to one of the categories listed in Table 7.1 [152]. For example, an algorithm with complexity $2n^2$ is less complex than one with complexity $4n^2$, but when compared with an algorithm of linear complexity – even with a high k-value – it is the polynomial factor in the complexity which has the most bearing on the overall performance of the algorithm. It is most desirable when attempting to reduce the complexity of an algorithm, to be able to reduce its complexity sufficiently that it moves down a by a whole complexity level, for example from polynomial to logarithmic. Where this is not possible, it may be feasible to reduce the complexity within the same overall complexity level, however particularly with large input sets the overall reduction will be less significant.

Much the same as when considering the execution times of tasks in RTS, the complexity of an algorithm is normally considered in the context of the worst-case value. This worst-case complexity is denoted using the "Big-Oh" (O) notation, where "Big Oh" denotes "worst-case complexity of the order of". Thus an algorithm of linear complexity over n elements will be denoted as having complexity O(n). This notation can be applied to both computational time and storage complexity.

7.1.2 Real-time predictability implications

In addition to considering its complexity, an important factor when considering the implications of incorporating an algorithm into a real-time system is its predictability. This is particularly important in the context of real-time analysis, which requires values for worst-case execution times and other similar properties. While making use of worst-case values in such analysis guarantees that there will be sufficient execution time for all tasks to complete, this is a cause of significant pessimism in situations where the worst-case execution time is significantly greater than the average, or where there is a particularly high variation in execution time over a number of instances [128]. In these cases, although the worst-case time is well-known, it becomes difficult to predict the behaviour of an algorithm for any one instance.

In some instances, a significant difference between average and worst-case values can affect the behaviour of a system. For example, in section 5.6 an example system was illustrated, in which an overrun could be produced in cases where a particular combination of tasks executed for less than their worst-case values. Therefore it can be supposed that similar issues will occur with any algorithm which has a wide variation in its operation. Such variation leads to difficulties in predicting the overall behaviour of that algorithm.

Consequently, it may be desirable that a fault prediction system exhibits predictable behaviour, or at least is predictable within a reasonably well-defined set of bounds.

7.2 Implications of constrained resources on the DCA

A particular reason for initially choosing the DCA to provide fault prediction in RTES was due to its inspiration from innate immunity [72] (see section 3.7.2). As the innate immune system does not possess the same evolutionary behaviour as the adaptive immune system, any algorithms taking direct inspiration from it should intuitively be less resource-intensive than those drawing their inspiration from the adaptive

immune system. Such techniques therefore offer significant benefits in situations where resources are constrained [114].

In order to gain some insight into the suitability of the DCA for incorporation in constrained-resource environments, it is necessary to consider its complexity, in terms of the computation time and the amount of storage required for it to operate. This will be primarily affected by the size of the task set being evaluated, and by the size of the DC population employed to monitor that task set.

The computational complexity of the DCA has only recently been considered in the literature [75], and then only in the context of one particular application. This is evaluated in the context of the application of the DCA to the deadline overrun problem. In addition, the predictability of the DCA is considered in the context of its application to the deadline overrun problem.

7.2.1 Computational requirements of the DCA

As discussed in section 5.5.2, the algorithm employed in chapters 5 and 6 operates according to the pseudocode shown again in Figure 5.5.

```
initialise cell population;
for each clock cycle loop
  select x DCs from DC population

  for each task in run queue loop
    compute input signals;
    for each selected cell loop
      update signal matrix;
      update antigen matrix;
    end loop;
  end loop;

  for each selected cell loop
    increment cycle count;
    compute output signal;
    if output signal > migration threshold then
      register danger output;
      reinitialise cell;
    else if cycle count >= lifecycle threshold then
      register safe output;
      reinitialise cell;
    end if;
  end loop;

end loop;
```

Figure 7.1: DCA pseudocode, amended for real-time operation

In order for the DCA to detect overruns in the system, there are a number of operations which must be carried out each simulation cycle. In order to establish the complexity of the algorithm, the necessary number of these operations for each simulation cycle must be established.

For the purposes of analysing the complexity of the algorithm, it is assumed that simple operations such as updating of antigen or signal matrices take a roughly equivalent constant time. This is because, in the software implementation, updating the antigen matrix is stored as a linked list, and adding a task into the antigen matrix is performed by adding an element to the head of that linked list. Each DC's signal matrix is stored as integers held in a data structure, and the updating of the signal values consists simply of incrementing the appropriate element within the data structure. Both of these individual operations are unaffected by the size of the task set or the size of the DC population, although obviously the number of times they are required to be executed will be affected.

The computation of the input signals for each task, however, depends on the relative priority of each task. In particular, the calculation of the danger and safe signals requires the calculation of the worst-case response time (WCRT) for the task in question. For any given task, this calculation also requires the calculation of the WCRT of all higher-priority tasks; therefore the complexity of this operation depends on the relative priority of the task in question relative to all the other tasks in the run queue at the time.

In the worst case, a system comprising n tasks will have n tasks in the run queue following a critical instant, therefore calculating the WCRT of the lowest-priority task requires the WCRT of all other tasks to be known. Although there are ways by which this procedure can be optimised (specifically by calculating the input signals in descending order of task priority, and storing the WCRT values between calculations) it cannot necessarily be assumed that these optimisations will be carried out.

Although in the worst case the WCRT calculation requires also calculating the WCRT for all other tasks in the system, the average case for any one task selected from the run queue at random will only require the calculation of the WCRT for half of the tasks present in the run queue at that present time. Given that the algorithm as illustrated in Figure 5.5 requires the calculation of input signals for all tasks currently present in the run queue at each simulation cycle, it can be assumed that over the whole range of calculations, each task's individual calculation period will take the average-case time.

The second stage of the computation, determining the DC's output signal and comparing its value against the relevant output thresholds, can be assumed to take constant time: in addition, these tasks involve simple increment or comparison operations and therefore take relatively little time compared with the more complex computation of the input signals in the first stage of the algorithm.

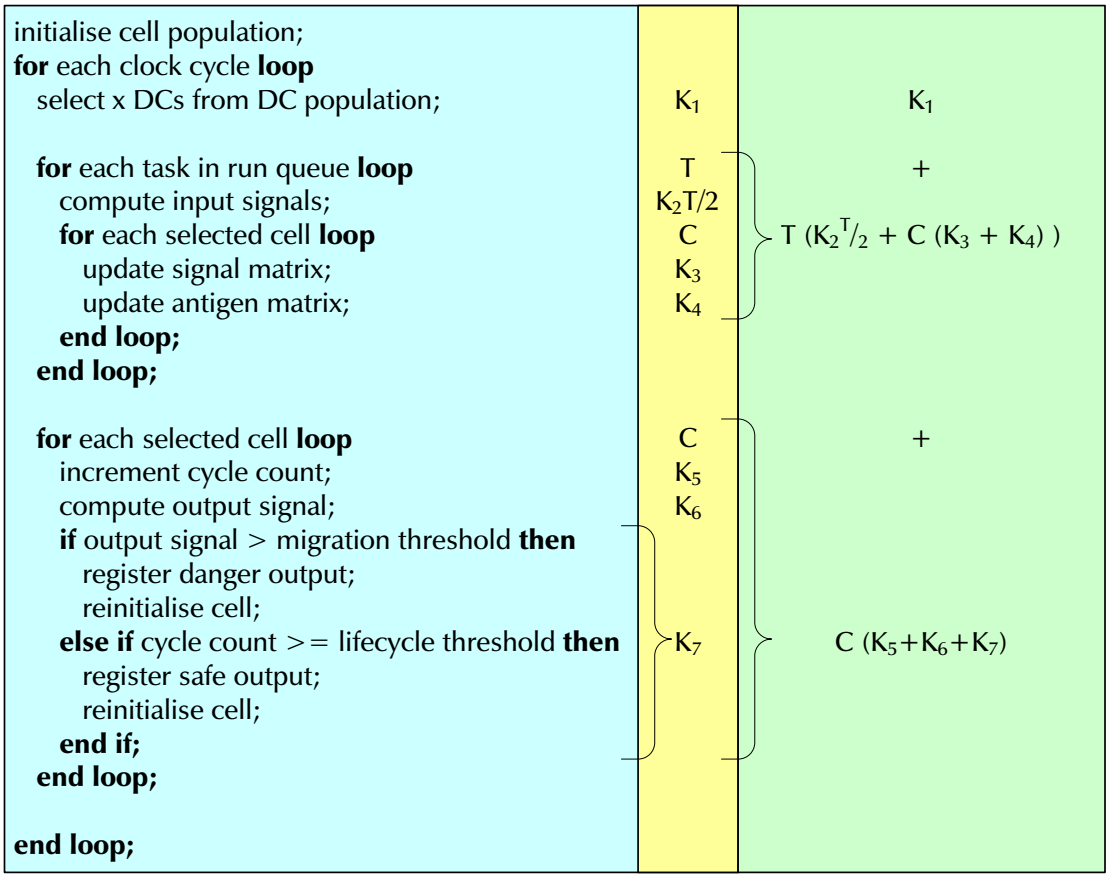


Figure 7.2: DCA pseudocode showing estimated complexity of statements

Therefore, the relative complexity of the algorithm can be estimated as shown in Figure 7.2. For each clock cycle, the complexity can be expressed as:

$$K_1 + T(K_2 \frac{T}{2} + C(K_3 + K_4)) + C(K_5 + K_6 + K_7) \quad (9)$$

where: T = *number of tasks in system*
C = *number of cells updated per cycle*

Assuming that all constant statements take approximately the same execution time, this can be simplified to give:

$$K + T(K\frac{T}{2} + 2CK) + 3CK \tag{10}$$

$$K + K\frac{T^2}{2} + 2TCK + 3CK \tag{11}$$

Discounting constant values gives:

$$1 + \frac{T^2}{2} + 2TC + 3C \tag{12}$$

Therefore, according to Equation 12 it can be seen that the complexity of the DCA in this application is quadratically proportional to the number of tasks in the system, and linearly proportional to the number of cells updated each cycle.

7.2.2 Storage requirements of the DCA

The storage requirements of the DCA are largely related to the space needed to accommodate its DC population.

The structure of each DC can be seen in Figure 7.3. The key elements within a DC requiring storage are the input signal matrix, and the antigen store – the output signals generally being generated from the input signal values each update cycle, and the migration threshold being a single integer value.

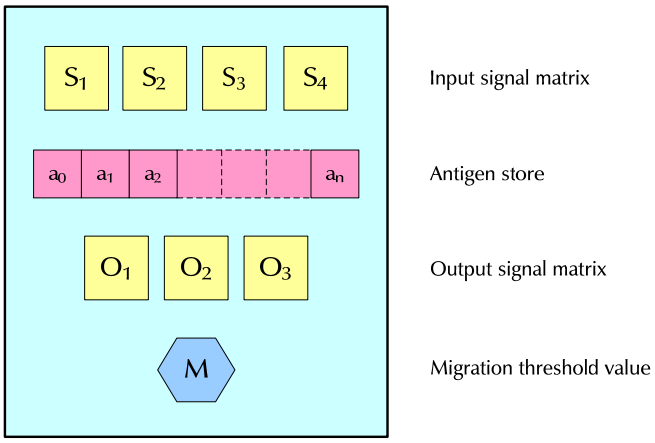


Figure 7.3: DC structure

As applied to the deadline overrun problem, the DC input signal matrix is stored as a single integer for each input signal, this integer being incremented whenever a signal is asserted during an update cycle. With a small number of signals, this amount of storage is trivial, amounting to only 4 bytes per signal where an integer is a 32-bit value.

The other element requiring storage, the antigen store, requires significantly more storage. In the simulation system, this is stored as a linked-list structure, with each element consisting of an integer value representing the task ID, and a pointer to the next element in the list. In order to reduce the computational complexity, and also to give information as to the relative concentrations of particular antigens, the task IDs of all tasks currently in the run queue are added to the antigen store during an update cycle, regardless of whether they are already present in the store or not. Therefore, in the worst case, the amount of storage required for the antigen store is:

$$K.T.L \tag{13}$$

where: K = *constant size of each list element*
T = *number of tasks present in system*
L = *DC lifecycle*

Therefore, across the entire DC population, the storage requirement of the DCA can be expressed as:

$$C(K_1.S + K_2.T.L) \quad (14)$$

where: C = *DC population size*
S = *number of input signals*
T = *number of tasks present in system*
L = *DC lifecycle*

Therefore, the storage requirement of the DCA is linearly proportional to the DC population size, the number of tasks present in the system, the DC lifecycle threshold value, and the number of input signals used. Of these factors, the most significant are likely to be the number of tasks present in the system, and the overall size of the DC population.

7.2.3 Non-determinism issues

The complexity of the DCA, as determined in the preceding sections, is calculated in terms of worst-case values. However, the use of these values introduces significant non-determinism issues, as frequently the actual values will be significantly less than the worst-case.

In particular, the computational complexity of the algorithm is quadratically dependent on the number of tasks currently in the run queue. In a system with, for example, 100 tasks, the number of tasks in the run queue at any particular time can be any number between 0 and 100.

Although the release of tasks is deterministic according to the task properties and the scheduling policy in use, the dispatch and execution of tasks is non-deterministic, being dependent on the exact execution times of both the task in question and any higher priority tasks. It is therefore impossible to determine *a priori* the number of tasks in the run queue at any particular time, and therefore the complexity of the DCA must always be considered in terms of the worst case.

Obviously the effect of this, particularly in systems with large numbers of tasks, is to make the DCA appear significantly more complex than it actually is, particularly given the quadratic relationship between computational complexity and the number of tasks present in the run queue.

Given the intention to apply the DCA in RTES, it would be particularly desirable to reduce the complexity of the algorithm. It would also be desirable to reduce the non-deterministic elements of the algorithm, so making it more predictable in its operation.

7.2.4 The deterministic DCA

In order to reduce the complexity of the DCA, a variant of the algorithm has been proposed, referred to as the "Deterministic DCA" (dDCA) [71]. Recognising the issues surrounding the antigen vector data structure and the need to store information regarding antigens encountered within the cell itself, two different solutions are suggested in [71]. The first of these is for the entire DC population to monitor the entire antigen population. A second solution is for each DC to be assigned an "antigen profile" whereby each DC is associated with a subset of the overall antigen population.

Considering the overall complexity of the DCA, it can be seen that reducing the number of antigens sampled by each DC would significantly reduce the computation time required by the algorithm, as the primary factor associated with the algorithmic complexity is the number of tasks present in the system being monitored.

Therefore, the concept of an antigen profile can be applied to the implementation of the DCA applied to the deadline overrun problem, in order to produce a reduced-complexity DCA which is better suited for use in the detection of anomalies in RTES. This variation is discussed in the next section.

7.3 A DCA for real-time anomaly detection

In order to reduce the complexity of the DCA and therefore to make it more suitable for application in RTES, alterations are made to the algorithm inspired by the antigen profile concept introduced along with the dDCA in [71].

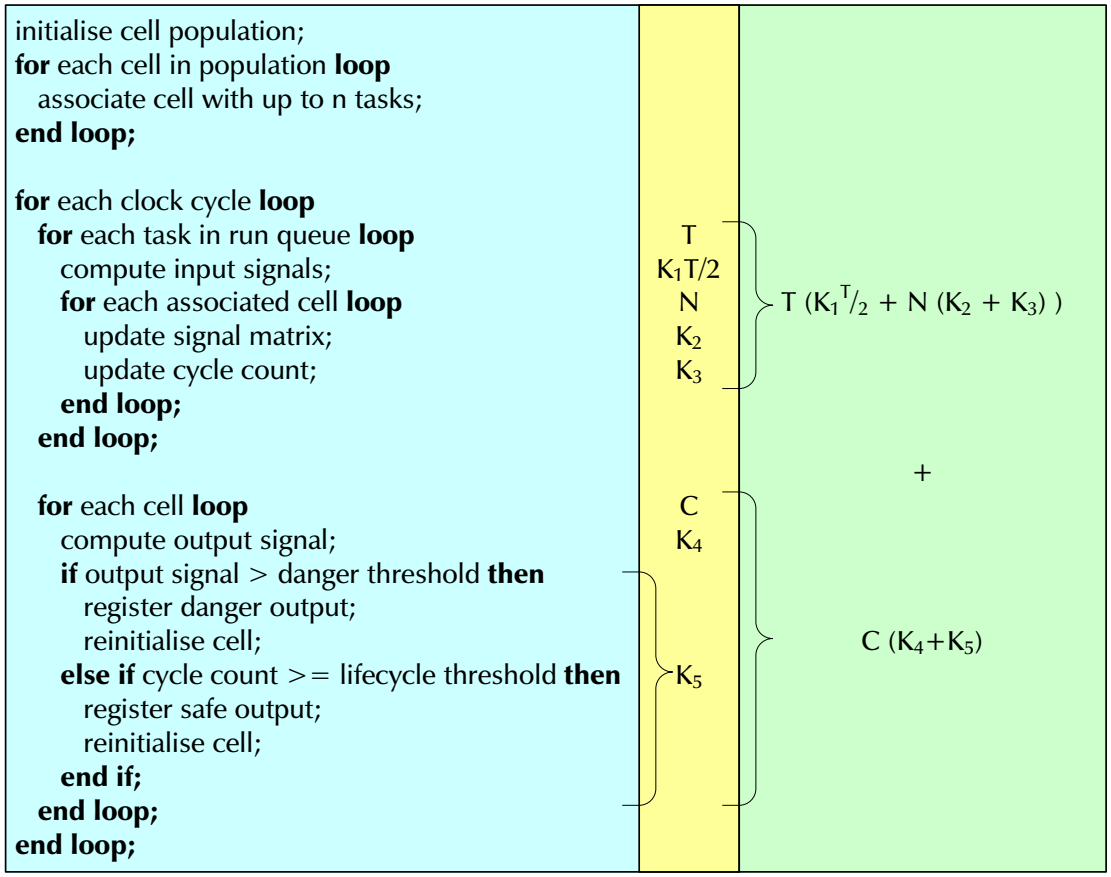


Figure 7.4: revised DCA pseudocode incorporating antigen profiling

The pseudocode for the revised algorithm, which will be referred to as the "Real Time DCA" (rtDCA) is shown in Figure 7.4, along with an estimation of the complexity or each of its constituent statements. For each evaluation cycle, the overall complexity is expressed as:

$$T(K_1\frac{T}{2}+N(K_2+K_3))+\mathrm{C}(\mathrm{K}_4+\mathrm{K}_5) \quad (15)$$

Where: *T = number of tasks in system*
N = number of tasks allocated to each DC
C = size of DC population

Which, assuming all constant values are equal, can be reduced to:

$$\frac{T^2}{2}+2TN+2C \quad (16)$$

Although this does not reduce the complexity significantly with respect to the number of tasks in the system, it does reduce the complexity with respect to the size of the DC population. In particular, the second term in equation 16 replaces C, the selected DCs, with the number of allocated tasks per DC.

This has some potential to reduce the complexity of the algorithm, as this number of allocated tasks per DC can potentially be quite small relative to the overall size of the DC population.

Where this makes the most significant impact, however, is on the storage requirements of the algorithm. By replacing the antigen store in each DC with a fixed antigen profile, there is no longer a need for the data structure associated with each DC to grow in size as antigens are encountered. Therefore, the amount of memory required to hold the DC population can now be easily established. This may have significant effects on the viability of the DCA in systems where there is a relative surplus of processing power but highly limited memory capacity.

7.4 Comparison of DCA variants

In order to gauge the effects caused by the modifications made to the DCA, it is necessary to compare the results obtained from the altered variant with those from the original.

In order to allow comparison with the results from chapter 6, the same task set sizes are used. Furthermore, the simulations are conducted using the same randomly-generated task sets as for the original version of the DCA, thereby enabling direct comparison of the results obtained from both variants of the DCA across individual task sets.

In order to investigate the effect of differing sizes of antigen profiles, the rtDCA will be evaluated with a range of DC population sizes and a range of differing antigen profile sizes. This should give a good range of task set sizes, task sets and DC population sizes over which the effects of antigen profiles can be examined. By incorporating the results of the complexity analysis carried out above, the overall accuracy of the two variants can then be considered with respect to the computational and storage requirements of the algorithm in each configuration. As before, the algorithm is evaluated in two different sensitivities, in order to gain some degree of insight into the effect of different combinations of weights and thresholds on the operation of the algorithm.

Graphs showing the overall accuracy and responsiveness for both the standard DCA and rtDCA, across the total evaluated parameter space over the range of task sets, are shown in the following sections. The overall trends will be discussed in section 7.4.5.

7.4.1 Overall accuracy/responsiveness comparison graphs: 10 tasks

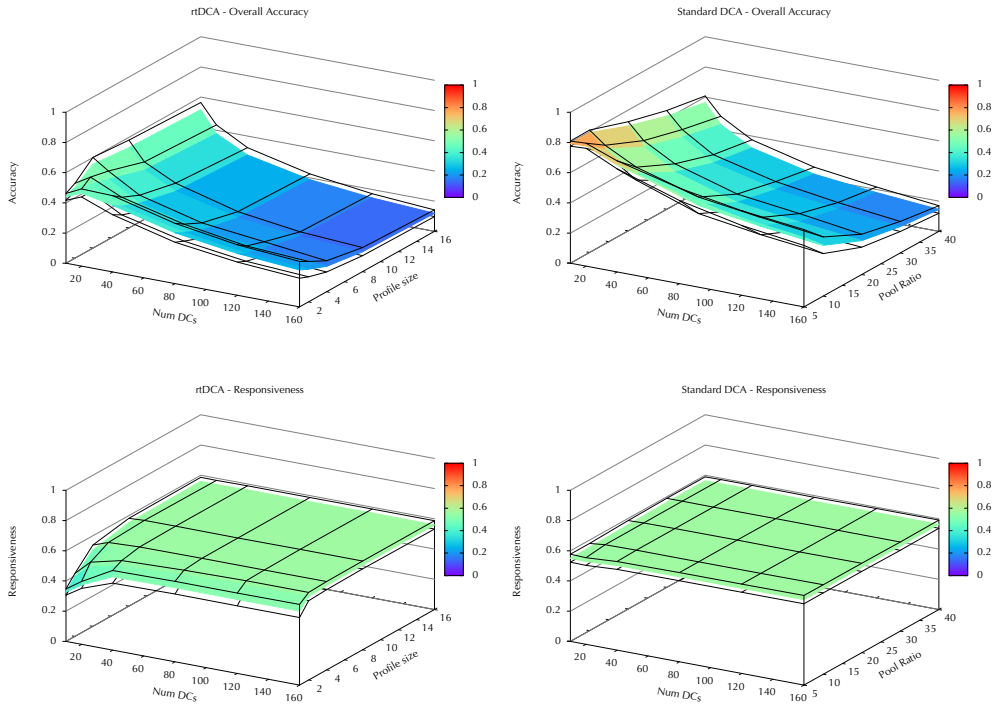


Figure 7.5: Accuracy (top) and responsiveness (bottom): rtDCA (left) and DCA (right) – 10 tasks, high sensitivity

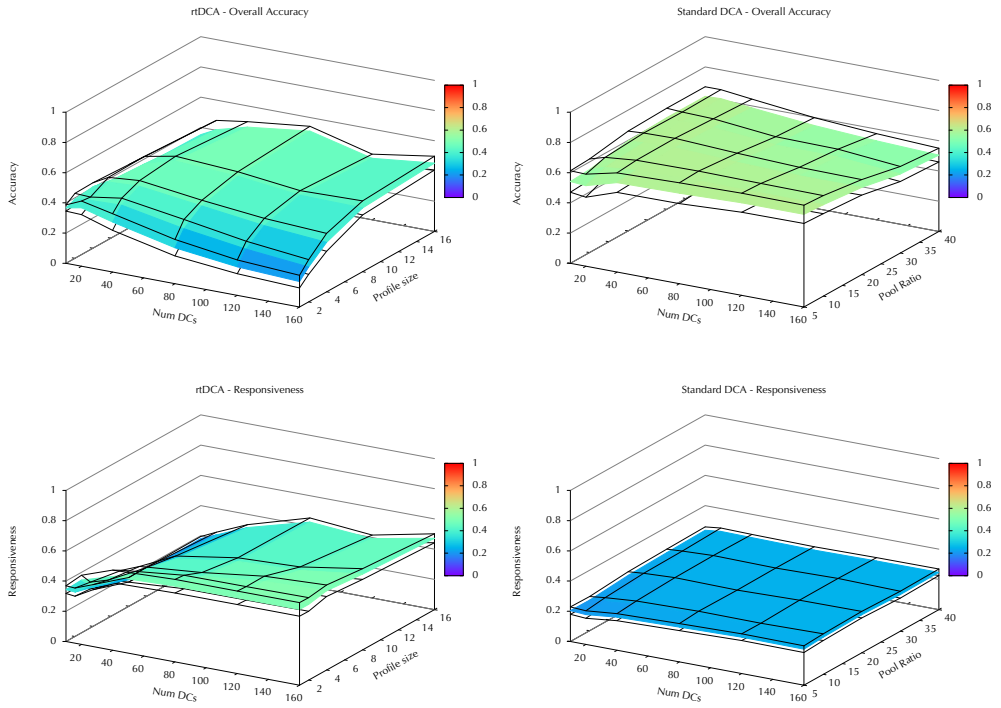


Figure 7.6: Accuracy (top) and responsiveness (bottom): rtDCA (left) and DCA (right) – 10 tasks, low sensitivity

7.4.2 Overall accuracy/responsiveness comparison graphs: 20 tasks

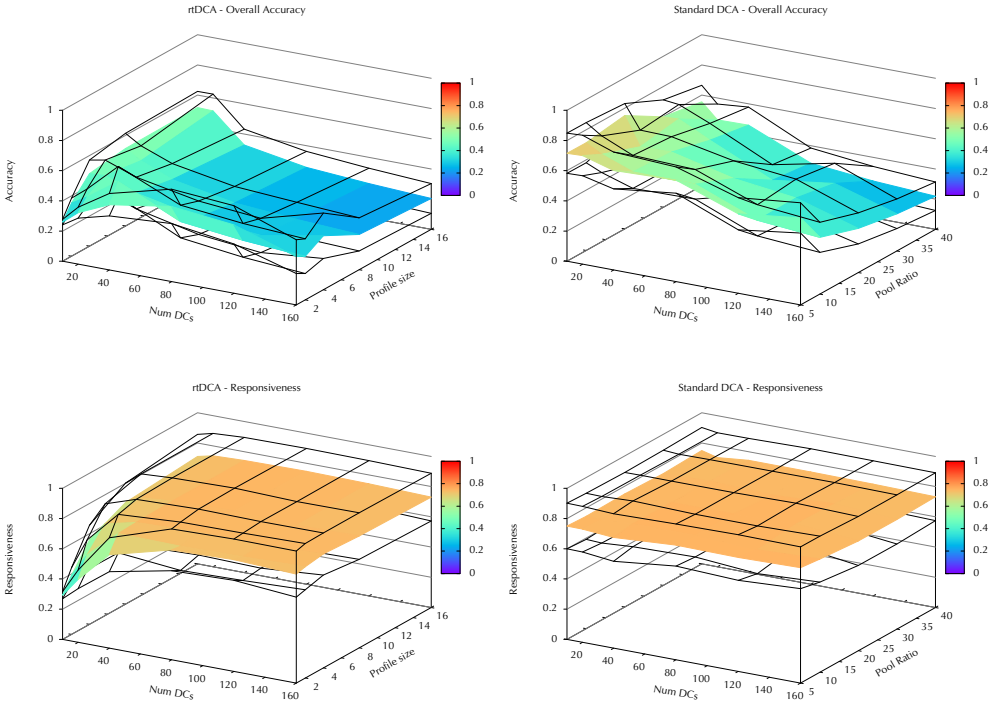


Figure 7.7: Accuracy (top) and responsiveness (bottom): rtDCA (left) and DCA (right) – 20 tasks, high sensitivity

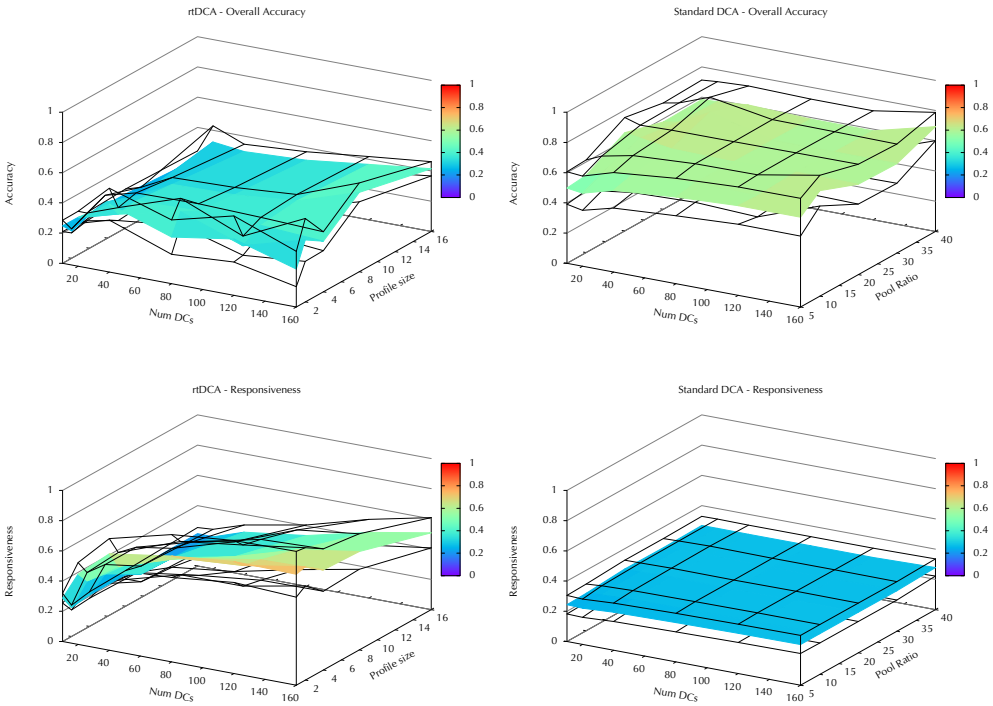


Figure 7.8: Accuracy (top) and responsiveness (bottom): rtDCA (left) and DCA (right) – 20 tasks, low sensitivity

7.4.3 Overall accuracy/responsiveness comparison graphs: 40 tasks

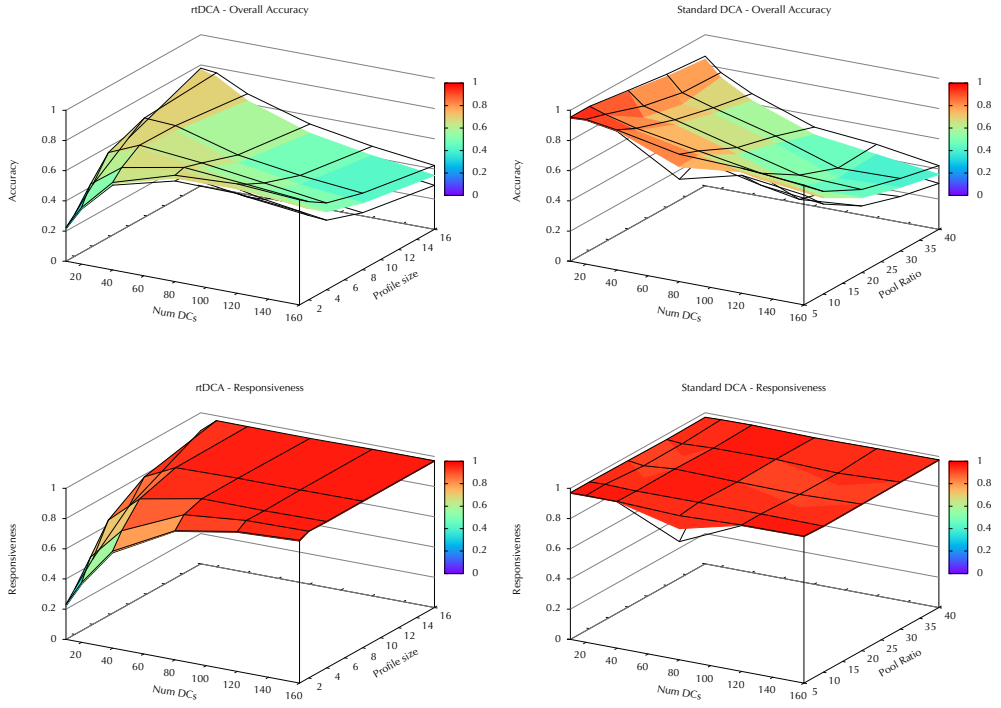


Figure 7.9: Accuracy (top) and responsiveness (bottom): rtDCA (left) and DCA (right) – 40 tasks, high sensitivity

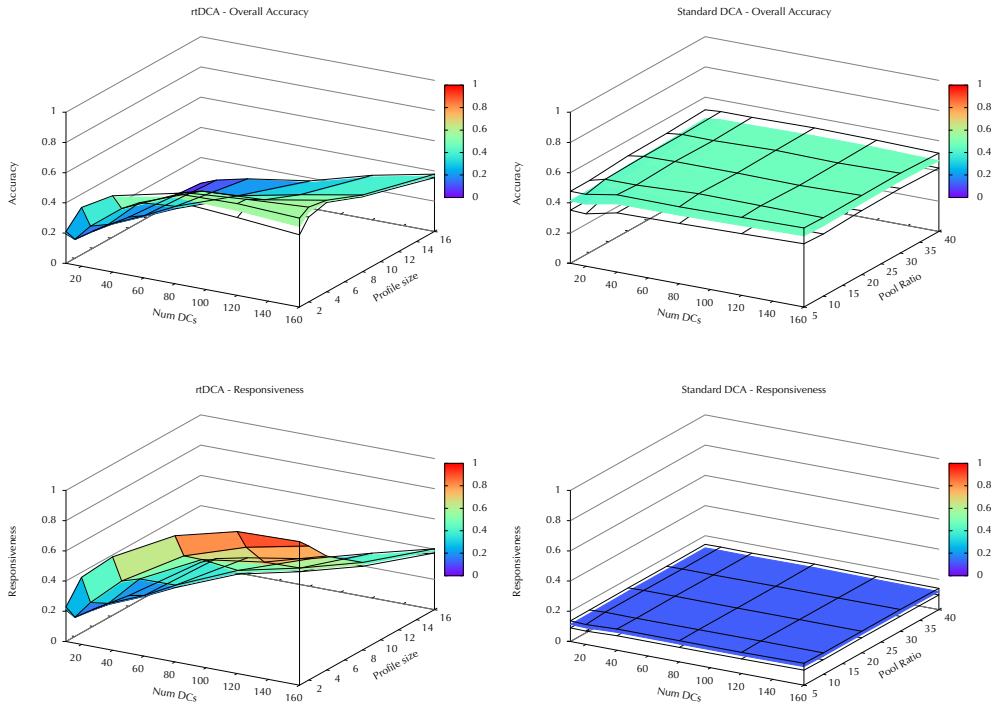


Figure 7.10: Accuracy (top) and responsiveness (bottom): rtDCA (left) and DCA (right) – 40 tasks, low sensitivity

7.4.4 Overall accuracy/responsiveness comparison graphs: 120 tasks

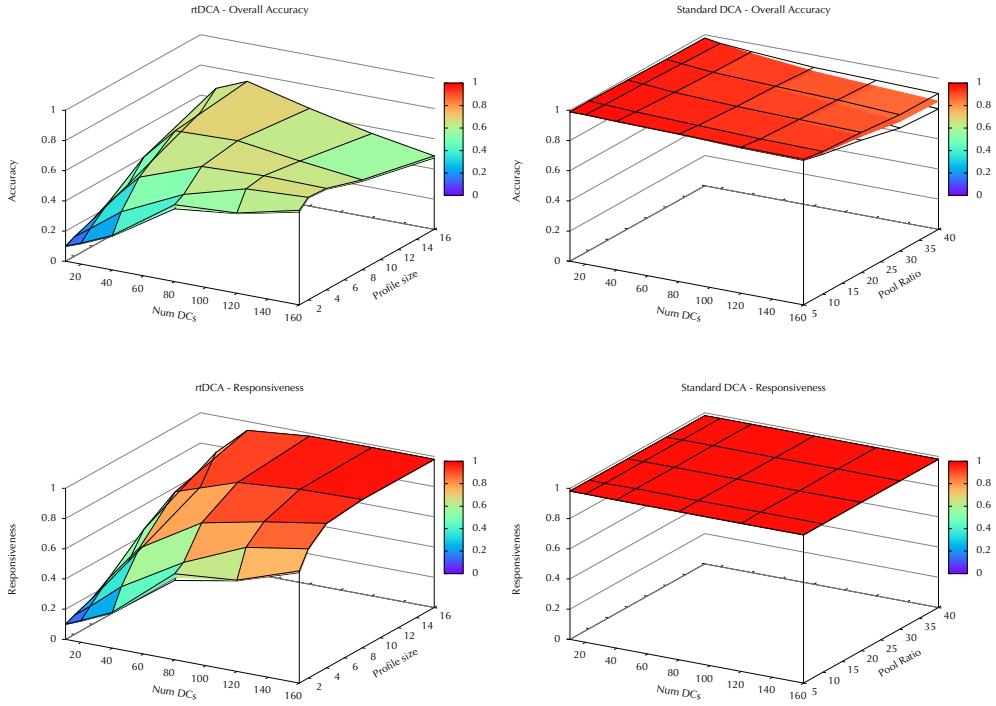


Figure 7.11: Accuracy (top) and responsiveness (bottom): rtDCA (left) and DCA (right) – 120 tasks, high sensitivity

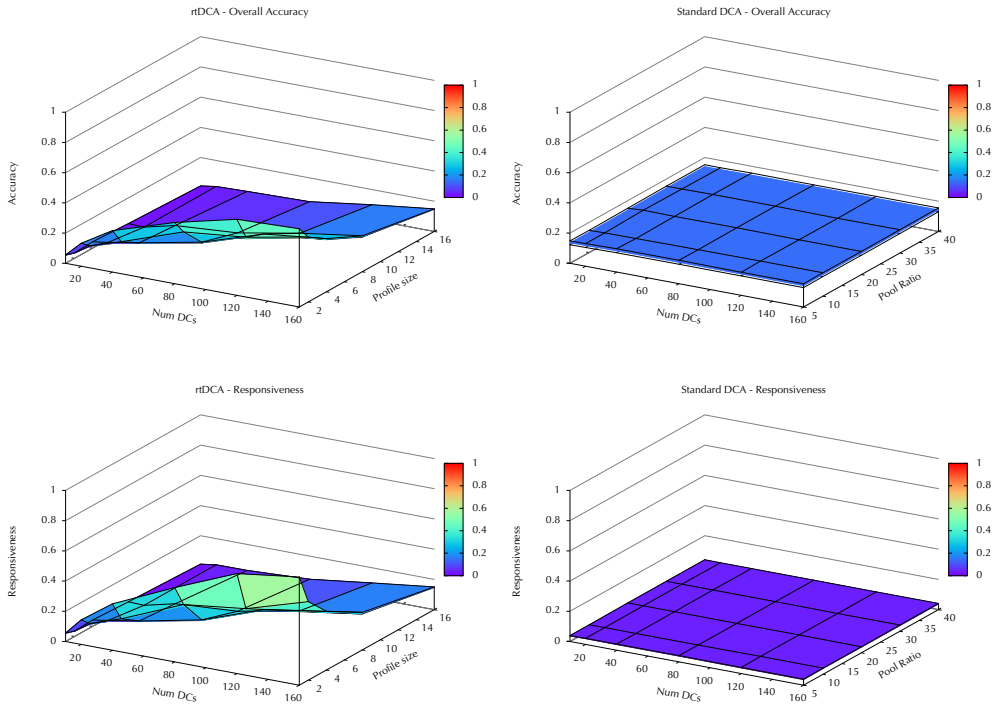


Figure 7.12: Accuracy (top) and responsiveness (bottom): rtDCA (left) and DCA (right) – 120 tasks, low sensitivity

7.4.5 Analysis of trends: rtDCA and standard DCA

When considering the performance of rtDCA, it is necessary to compare its performance with that of the standard DCA, in order to determine whether the changes made to the algorithm are beneficial compared to the standard algorithm.

7.4.5.1 10-task analysis

For small task sets, such as those with 10 tasks as shown in section 7.4.1, its overall performance is similar to that of the standard DCA. For the high-sensitivity variant, the behaviour is almost identical with respect to overall accuracy, and also for responsiveness in all but the smallest evaluated profile sizes – although the overall accuracy is slightly lower.

When considering the low-sensitivity variant, for rtDCA considerably greater variation in both the accuracy and responsiveness can be seen across the parameter range when compared with the standard algorithm. Considering accuracy, it appears that as the DC population increases, the accuracy level increases to a peak, followed by an apparent tailing off as the population size increases further. The position of this peak seems to be influenced by the profile size employed: for the largest evaluated profile size of 16 tasks per DC, the peak occurs at a DC population size of 80 cells.

As the profile size reduces, the size of DC population where the peak accuracy level is attained also reduces: at profile size 8, the peak accuracy is attained with a population size of 40 cells. At the smallest profile size evaluated, with only one allocated task per DC, the peak accuracy is attained with a population size of only 20 DCs, and the overall accuracy can be seen to decrease significantly as the DC population size increases.

Across the evaluated parameter space, the highest attained accuracy of 0.51 corresponds to a DC population size of 80 and a profile size of 16, closely followed by an accuracy of 0.50 for 40/8. Overall, however, when compared with the standard

algorithm, the accuracy of rtDCA is generally lower than that of the standard algorithm.

Although the accuracy of the algorithm is inferior, the overall responsiveness achieved by rtDCA is superior to that of the standard algorithm for the low-sensitivity variant. The responsiveness of the algorithm appears to increase as the DC population size increases then reach a plateau – the population size at which the plateau is reached being greater as the profile size increases. With a profile size of 1, the responsiveness reaches this point at a population size of only 40, but a profile size of 16 requires a population of 160 cells to reach a similar level of responsiveness. The overall highest responsiveness of 0.52 is reached with 160 cells and a profile size of 4; however 40/1 gives an accuracy of 0.49.

For the ten-task examples, it appears that with rtDCA the profile size has a greater influence on the overall accuracy and responsiveness of the algorithm than the population size. Indeed, in some cases it appears that lowering the profile size has the effect of also lowering the DC population size necessary to obtain the same level of overall accuracy or responsiveness. This is important as it appears that, by enabling the use of smaller DC population sizes, the required computational and storage time required by the algorithm can be significantly reduced by making use of a smaller profile size, with little effect on the overall accuracy of the algorithm. However, this advantage is somewhat tempered by the fact that, overall, the accuracy appears to be lower than that of the standard DCA.

7.4.5.2 20-task analysis

Examining the graphs for the 20-task examples shows that, for the high-sensitivity variant, rtDCA exhibits similar behaviour as observed for the 10-task examples, although where both the DC population and the profile size is particularly low there is a more noticeable decrease in the levels of accuracy and responsiveness attained. As with the 10-task examples, the overall level of accuracy is slightly lower for rtDCA

than the standard algorithm, but barring the low combinations of profile size and population size already observed, the responsiveness is the same between versions.

Also as seen before, the accuracy and responsiveness for the low-sensitivity rtDCA variant both show greater variation than with the standard algorithm. The overall accuracy attained by rtDCA is once again lower than that of the standard algorithm, and the responsiveness of rtDCA higher: again, as observed with the 10-task examples. The differences between the two algorithms are greater than with the ten-task examples – the accuracy of rtDCA being more noticeably lower overall, and the responsiveness being significantly higher.

Particularly with the responsiveness, the characteristic whereby reducing the profile size also results in fewer DCs being necessary to attain the same level of responsiveness is still present in the 20-task examples. At extremely low profile sizes this appears to also be the case for accuracy, but it is more difficult to determine whether this trend follows through the whole results set. This is due to the increased variation in the results with this task set size – the standard variant also shows much greater variation in its results for this size of task sets, suggesting that this variation is caused by the characteristics of the randomly-generated test task sets rather than any particular characteristics of either variant of the algorithm.

7.4.5.3 40-task and 120-task analysis

Considering first the high-sensitivity variant of the algorithm for the 40-task sets, it can be seen that rtDCA exhibits the same behaviour as it does for the 10 and 20-task sets. As the task set size increases, the algorithm's response becomes increasingly poor for smaller population size and profile size combinations, although the maximum accuracy is higher than for the smaller task sets. For the 120-task sets, rtDCA shows a similar pattern to that obtained from the other sizes of task sets: by comparison, it is actually the results obtained from the standard DCA which do not fit the pattern.

Across both 40 and 120-task sets, rtDCA also manages to attain a comparable level of responsiveness to the standard algorithm, given either a sufficiently large DC population or profile size, again mirroring the behaviour seen with smaller task sets.

For the larger task set sizes, it is again the low-sensitivity variant which exhibits the more interesting behaviour when compared with the standard DCA. With the 40-task sets, rtDCA produces a peak accuracy rating higher than that attained by the standard variant: this is also true of the 120-task sets, where the low-sensitivity standard variant performed particularly poorly. Also with the low-sensitivity variant across the larger sizes of task set, the maximum attained responsiveness is significantly higher.

As before, rtDCA seems to give higher values of both accuracy and responsiveness where the profile size is smallest. For the 40-task sets the accuracy result can be seen to reach its maximum possible value where the number of DCs is over 80, however the responsiveness still appears to be increasing as the number of DCs reaches 160 and may increase further if additional population sizes of greater than 160 were also evaluated. For the 120-task sets, this appears to be true for both accuracy and responsiveness, both give the impression that they would continue to increase if the DC population size was increased further.

7.5 Analysis of accuracy component results for rtDCA

Section 7.4 analysed the overall accuracy and responsiveness of rtDCA compared with the standard algorithm as evaluated in chapter 6. As well as examining the overall accuracy, section 6.4.2 investigated the individual components which are combined to produce the overall accuracy figure for the standard DCA. In order to gain further insight into the behaviour of rtDCA it is also necessary to examine its results in further detail.

7.5.1 High sensitivity rtDCA

As the high-sensitivity variant of rtDCA appears to behave in a similar manner to the standard variant of the algorithm overall, it is not unreasonable to assume that its true positive and false positive graphs would be similar to those obtained from the standard DCA.

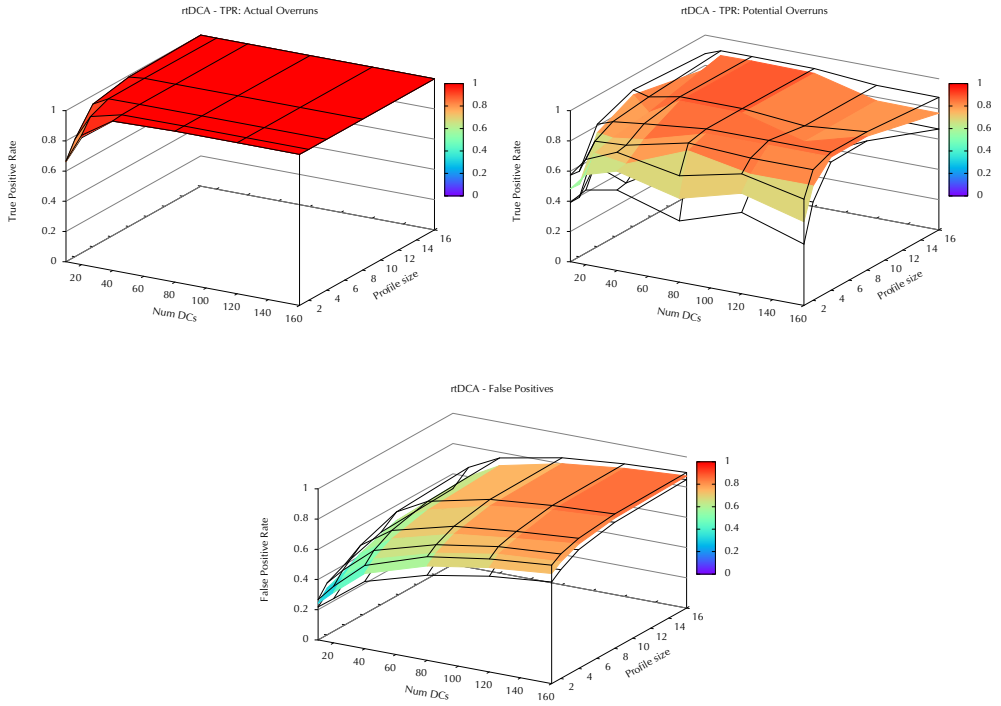


Figure 7.13: rtDCA accuracy component results: 10-tasks, high sensitivity: actual TPR (top left), potential TPR (top right), FP (bottom)

Graphs showing the accuracy component results for rtDCA are shown in Figure 7.13 (10 tasks) and Figure 7.14 (120 tasks), the graphs for 20 and 40 tasks being omitted for space reasons. By comparison with the equivalent graphs for the standard DCA (Figure 6.24 and Figure 6.30) it can be seen that, for the 10-task sets, the behaviour is largely similar, with the exception of very small DC population and profile sizes. As these increase, there is a steady increase in the number of false positives, which has a detrimental effect on the overall accuracy as shown in Figure 7.5.

For the 120-task sets, the component results are slightly different. It is apparent where both the DC population size and the profile size are small that the algorithm's

true positive rate is significantly affected. This behaviour is due to the way in which the DC profile is applied: where the profile size (and therefore the number of tasks monitored by each DC) is very small, the effect of a small DC population is that not all tasks in the system will be monitored: for example, given a population size of 20 and a profile size of 2, it is only possible to monitor a maximum of 40 tasks with that set of DCs (as the allocation is random it is quite likely that there will be some duplication, therefore the actual number monitored will be less than this). It follows intuitively, therefore, that where the profile size is very small, that there must be at least as many DCs in the system as there are tasks to ensure complete monitoring.

In addition, there is a notable increase in false positives as both the DC population size and the profile size increase. This effect is most pronounced when both sizes are at their largest, causing the tailing off in overall accuracy seen in Figure 7.11.

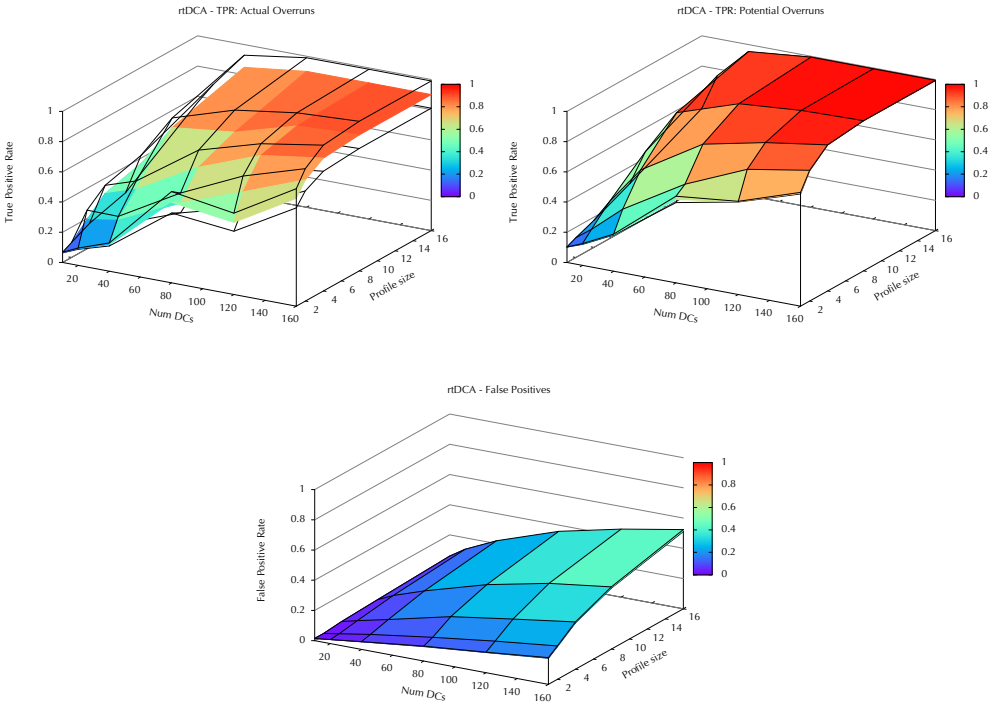


Figure 7.14: rtDCA accuracy component results: 120-tasks, high sensitivity: actual TPR (top left), potential TPR (top right), FP (bottom)

7.5.2 Low sensitivity rtDCA

Although in its high-sensitivity variant, rtDCA exhibits similar performance characteristics to the standard DCA, it is apparent from its overall accuracy and responsiveness graphs that the behaviour of the low-sensitivity variant is quite different.

Graphs for the accuracy components for the low-sensitivity variant of rtDCA are shown in Figure 7.15 - Figure 7.18. Comparing these with both the graphs for the high-sensitivity variant above, and also those for the low-sensitivity variant of the standard DCA (Figure 6.25 - Figure 6.31) shows that, in this configuration, rtDCA behaves quite differently from the other variants of the DCA.

Considering first the occurrence of false positive results, the rate at which false positives occur can be seen to increase as the DC population size increases, in common with other variants of the algorithm. However in this configuration there is a marked decrease in false positives as the profile size increases, in contrast to all other variants. At smaller task set sizes, these increased false positives at low profile sizes dominate over the generally high true positive rates, leading to the relatively low overall accuracy rate seen in Figure 7.6.

As the task set size increases, there is a reduction in the false positive rate overall, but this is also matched by a decrease in the true positive rate. However, the decrease in the true positive rate seen with rtDCA is less than that seen with the standard algorithm. Therefore, although the low-sensitivity standard algorithm outperforms rtDCA for smaller sizes of task sets, the reverse is true for the larger task sets. In addition, rtDCA is overall more responsive than the standard algorithm.

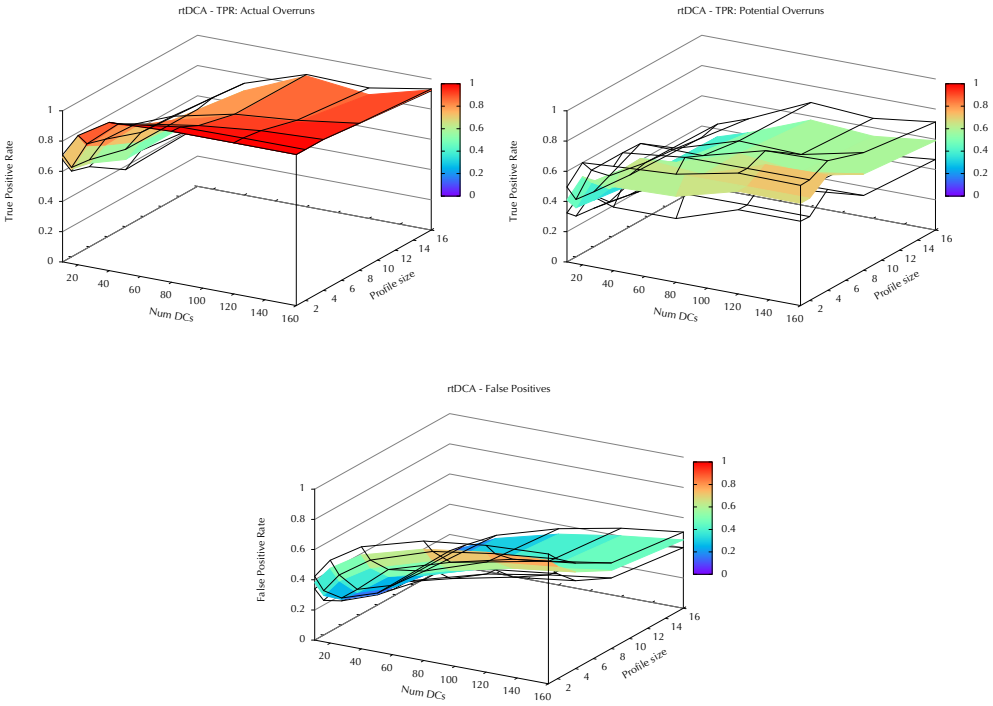


Figure 7.15: rtDCA accuracy component results: 10-tasks, low sensitivity: actual TPR (top left), potential TPR (top right), FP (bottom)

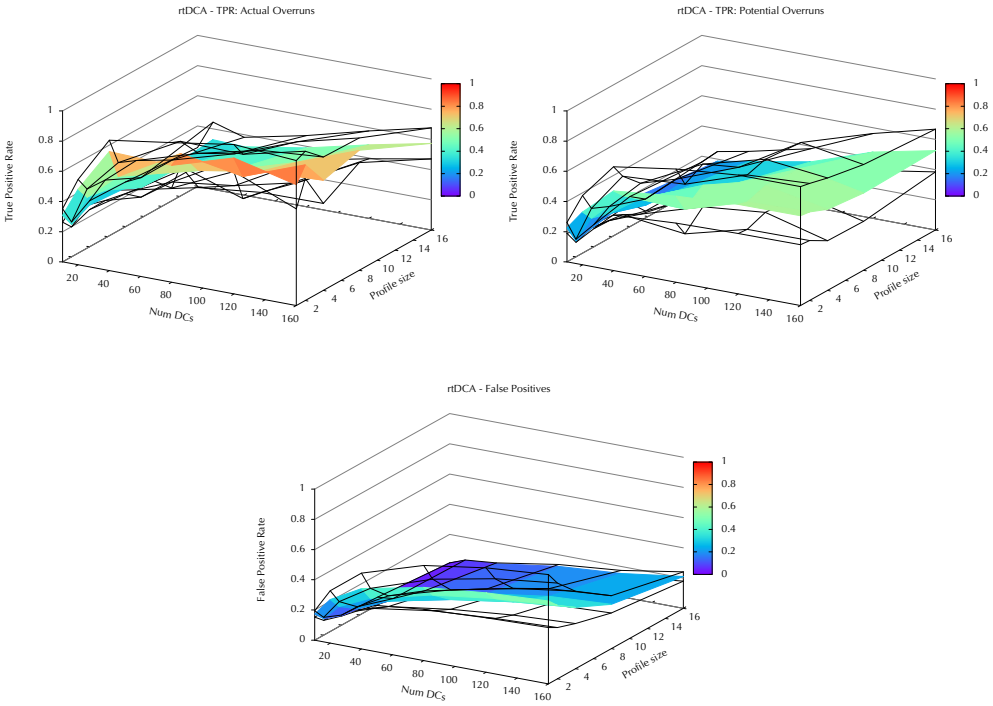


Figure 7.16: rtDCA accuracy component results: 20-tasks, low sensitivity: actual TPR (top left), potential TPR (top right), FP (bottom)

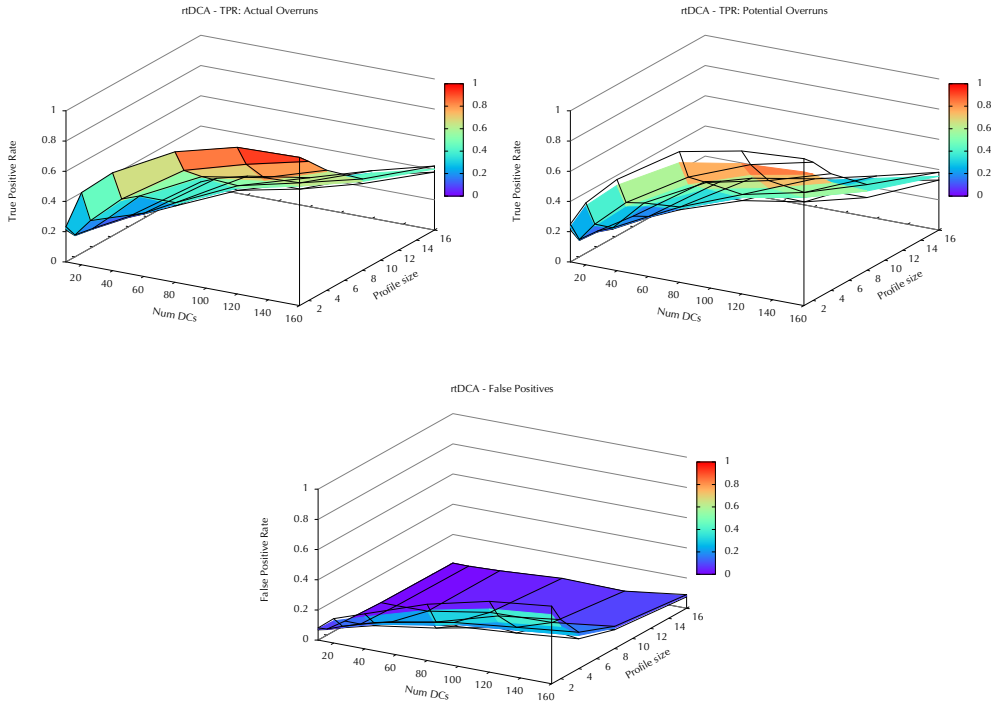


Figure 7.17: rtDCA accuracy component results: 40-tasks, low sensitivity: actual TPR (top left), potential TPR (top right), FP (bottom)

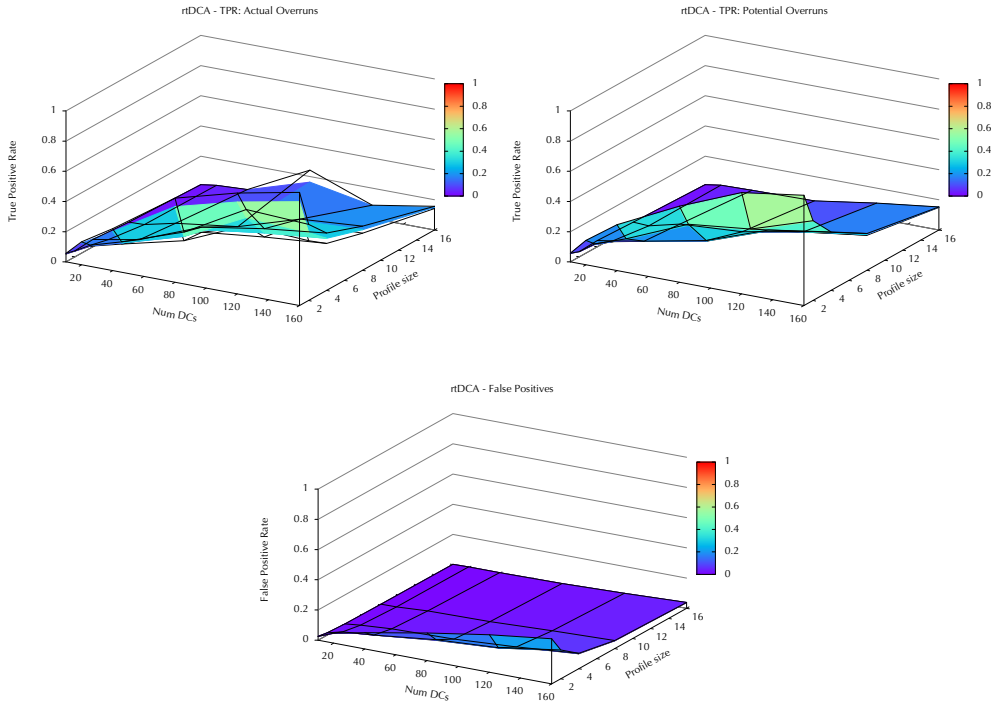


Figure 7.18: rtDCA accuracy component results: 120-tasks, low sensitivity: actual TPR (top left), potential TPR (top right), FP (bottom)

7.6 DCA in RTES: summary

This chapter has considered the implications of incorporating the DCA into a system with real-time and resource constraints such as those found in typical RTES. In particular, the computational complexity of the algorithm has been considered with respect to both the processor time and storage required for the algorithm to operate effectively. Also, the implications of non-deterministic behaviour on the predictability of a system have been examined.

In order to reduce the level of complexity and increase the predictability, a modification of the DCA has been evaluated and its performance compared with that of the standard algorithm. The revised algorithm, rtDCA, can be seen to outperform the standard algorithm in some cases, particularly in cases with larger task set sizes, although for smaller task set sizes the standard algorithm is superior.

As found with the standard algorithm, the biggest factor affecting the performance of rtDCA is the setting of the internal weights and threshold values: indeed, for rtDCA these seem to have a greater effect on the overall operation than for the standard algorithm.

A particular advantage of rtDCA over the standard algorithm is that there is no longer a need for each DC to maintain an antigen vector, the size of which is greatly variable and bounded primarily by the number of tasks present in the system being monitored. The fixed allocation system used by rtDCA gives each DC a fixed and known size in terms of its memory requirement. This makes the implementation of rtDCA in environments where there are significant memory constraints considerably more straightforward.

8

Conclusions and further work

Chapters 5, 6 and 7 have examined the use of the DCA, a technique inspired by the observed operations of the innate immune system, as a means of improving the reliability of RTES by providing a mechanism for the detection of deadline overruns.

8.1 Fulfilment of hypothesis

The thesis hypothesis as stated in section 4.1 was:

> *Artificial immune systems can provide an accurate, responsive method to detect anomalies across a range of real-time embedded systems*

To help judge whether a solution successfully meets the goals laid down by the hypothesis, chapter 4 laid down a number of criteria against which a candidate

solution incorporating AIS techniques should be evaluated in order to gauge its success. These were:

- Correctness
- Responsiveness
- Flexibility
- Scalability
- Resource efficiency

This chapter will evaluate the performance of the DCA as applied to the deadline overrun problem in this thesis, with respect to the above criteria.

8.1.1 Correctness of the DCA

With respect to correctness, section 4.1 required that:

> *any proposed solution must be able to successfully identify any anomalies present in a RTES – of all the metrics discussed here, the correctness of a solution is the most fundamental. When considering correctness, it is particularly important to evaluate the solution's overall accuracy, taking into account not just successful identifications, but also the presence of false positives, where the system incorrectly detects an anomaly where none is present, and false negatives where the system fails to detect an anomaly which is present. A solution which upholds the hypothesis will therefore be able to correctly identify anomalies present in a system.*

The correctness of both variants of the DCA has been evaluated in chapters 6 and 7, compared with the results obtained from an online response time analysis conducted at the same time. Across all evaluated problems and configurations, the ability of the DCA to both detect actual overruns and predict potential overruns has been examined. The accuracy figures by which the DCA has been analysed have taken the

incidence of false positives into consideration as well as the number of true positives, and these individual accuracy components have also been separately analysed.

Although there is a wide variation in accuracy between the different variations of the algorithm evaluated, it is evident that there are a number of parameters, in addition to those investigated, which have a significant effect on the performance of the DCA. It is likely, therefore, that a suitable configuration can be found which will give a good level of accuracy for any particular problem, although it appears likely that this will have to be individually established for each different problem, rather than there being a universal one-size-fits-all configuration which works equally well for every potential problem.

The ability of the DCA to act as a detection mechanism for overruns after they occur is useful, albeit that there exist mechanisms allowing systems to report this information. However, the ability of the DCA to predict overruns which may occur in a system – whether or not those tasks go on to overrun – is something which had previously required the use of static real-time analysis techniques to achieve. Although the DCA as currently implemented is perhaps not the drop-in monitoring system required, it does demonstrate that it has the potential to act as one given further refinement.

8.1.2 Responsiveness of the DCA

With respect to responsiveness, section 4.1 required that:

> *particularly with solutions applied to in-service systems, it is important that the identification of any anomalies present in the system are identified quickly whenever they occur. A system which is slow to detect anomalies is potentially less effective than one which is highly responsive, especially at detecting transient anomalies where the period for which the fault is present is shorter than the response time*

The effect of a number of DCA parameters on the responsiveness of the algorithm has been investigated in chapters 6 and 7. As with the accuracy, it has been established that across the different versions and configurations of the algorithm the responsiveness is significantly affected by the various algorithmic parameters.

The analysis conducted in chapter 7 has shown that in one of the two scenarios evaluated, the rtDCA variant exhibits greater responsiveness than the standard variant. As with the accuracy, it is probably fair to deduce that there will be an optimal parameter set for each individual problem to which the DCA is applied. Of course, it may not be the case that the optimal parameter set for accuracy also gives the optimum responsiveness: this has been observed across a number of scenarios. Consequently in this case it will be necessary to trade off these two results: probably, it will be the case that accuracy will be prioritised over responsiveness.

The ability of the DCA to respond particularly to potential overruns which have not yet occurred is particularly useful, as its output can then be used to prevent faults from occurring, by altering the execution of various tasks in the system to ensure that all tasks meet their deadlines. The DCA demonstrates good potential for incorporation into such a scheme.

8.1.3 Flexibility of the DCA

With respect to flexibility, section 4.1 required that:

> *any improved development technique is that it can be applied to a variety of problems with little or no effort on the part of the system designer. This will allow the technique to be used with families of similar products, or in updated versions of products, without the need to perform large amounts of time-consuming analysis for each one.*

In chapters 5 and 6 the DCA has been applied to a variety of randomly generated task sets of varying sizes, and has shown good potential to be able to detect overruns across this range of task sets. Overall, the rate of detection is good, but evidently there is the potential for this to be improved by the use of different parameter sets.

By generalising sporadic tasks into periodics, where the period is equal to the minimum inter-arrival time, the DCA can be demonstrated to cope with sporadic and aperiodic tasks in the same way that it currently deals with periodics.

The ability of the DCA to detect overruns across a wide variety of task sets goes some way to providing a drop-in solution which can be effective at detecting overruns over a wide range of systems. Although it would be particularly desirable to have such a system which would be able to function with zero configuration, it may be the case that this is not possible and that the DCA parameters will need to be tuned for each particular problem. In the context of the deadline overrun problem, only a subset of the possible parameters have currently been examined: further investigation will be required to determine whether it is possible to make use of the DCA in a "one-size-fits-all" style.

8.1.4 Scalability of the DCA

With respect to scalability, section 4.1 required that:

> *for a solution to be successfully applied in real-world systems, it must be capable of being scalable to work with systems of the size and complexity of those being produced today. In addition, it should be able to cope with projected increases in system complexity, making the solution of continued use in the longer term.*

The DCA has been evaluated over a range of task set sizes, ranging from a very small test example with four tasks, to much larger examples containing 120 randomly-

generated tasks. For each of these different task set sizes, the DCA is able to detect overruns successfully, although the rate of success varies depending on the exact properties of the task in question.

As with the flexibility requirement, the ability of the DCA to function across a range of different sizes of task set goes some way towards providing a solution which can be readily incorporated into any system. Although it is apparent that the effectiveness of the DCA as currently implemented is somewhat variable dependent on the size of the problems to which it applied, it is clear that the technique is viable across the evaluated range of problems.

8.1.5 Resource efficiency of the DCA

With respect to resource efficiency, section 4.1 required that:

> *the need for a solution to make efficient use of resource is of significance when considering systems where resources are limited, as is the case with RTES in the consumer electronics domain. Any solution intended to be integrated into an in-service RTES must therefore be able to function effectively within the available system resources for it to be useful.*

Chapter 7 investigated the implications of constrained resources on the operation of the DCA, particularly with respect to the algorithm's computational complexity and its memory usage.

A revised variant of the algorithm, rtDCA, has been proposed which reduces the computational complexity of the algorithm slightly, and reduces the complexity of its memory requirements considerably. This variant additionally appears to be superior in terms of accuracy and responsiveness at smaller DC population sizes than the standard variant: removing the need for large population sizes causes the complexity

to be reduced considerably, as the DC population size is a key factor in determining the complexity of the algorithm.

The ability to reduce the memory requirement complexity significantly is particularly useful, as it potentially allows the DCA to be implemented on a separate microcontroller or FPGA device with only limited available memory, while not affecting the performance of the algorithm significantly.

This opens up the potential to employ the DCA as part of an external monitoring device during the development of embedded devices, or to incorporate such a "fault-finding" chip into a completed system allowing it to detect and repair faults, or simply to monitor their presence and feed this information back to the system designers.

8.2 Conclusions and Further work

This work is, as far as can be ascertained, the first application of a "non-standard" biologically-inspired method to a very traditional real-time problem. Due to the safety-critical nature of many real-time systems, it is unlikely that such a technique will ever become mainstream in the RTS domain. However, the aim of this work was to find a technique which could be of use in the CE domain, where a good number of systems exhibit real-time properties, yet time and budgetary constraints preclude the use of traditional real-time engineering methods.

As a first application, this thesis has established that techniques inspired from biology can be applied with to a typical problem encountered during the development of CE devices and produce useful results. This has clear benefits to the CE development community, who would particularly welcome a method by which real-time problems in embedded systems can be detected easily, without the need for time-consuming analysis.

In addition, the real-time systems domain is an area where previously AIS techniques have not been applied. A key result of this work is therefore a new application area for AIS: at a time when AIS is becoming increasingly accepted as a branch of computer science in its own right, the opening up of new application areas should serve to increase its acceptance still further.

It is clear that the technique requires further work in order to gain additional understanding about its operation, and also to allow further refinement of the methods used in order to improve its operation. In particular, the following areas can be identified as requiring further work:

8.2.1 Further investigation of DCA parameters

In order to further understand how the DCA behaves when applied in RTES, further investigation of its parameters is required, in particular the weights and thresholds which in all other applications to date have been based on the observed behaviour of living DCs *in vitro*. These parameters appear, based on the simple high/low sensitivity variants investigated so far, to have a hugely significant effect on the operation of the algorithm and therefore on its accuracy and responsiveness.

A possibility for investigating these parameters further is the use of evolutionary techniques, such as genetic algorithms or even clonal selection to determine an optimal parameter set for a particular problem.

8.2.2 Implementation in a real system

The investigations of the DCA in RTES have so far been conducted in a simulation environment, largely because this allows for the output of the DCA to be compared with that of a perfect predictor in order to easily gauge its accuracy and responsiveness.

Any real-world application of the DCA in RTES will be as a part of a real system, rather than running as a simulation. There is potential for future work to investigate the physical implementation of the DCA as part of an embedded device, and also to evaluate the effectiveness of different types of implementation.

The low and easily predictable storage requirements of rtDCA make hardware implementations of this algorithm on a device such as a microcontroller or FPGA possible, leading to the possibility that a DCA-like algorithm could be implemented on one of these devices and used to monitor an existing system, requiring no modifications to be made to that system other than providing a physical monitoring interface.

8.2.3 Using DCA-provided information to generate feedback

As investigated in this work, the DCA is envisaged as a way by which information about the state of a system can be obtained, however as implemented there is no mechanism by which that information can be used to change the behaviour of that system such that the detected problem is avoided.

Further work could be concentrated on producing such a mechanism, providing a completely self-contained monitoring and feedback solution, thereby allowing RTES to be largely fault-tolerant by being able to detect and work around problems without the need for human intervention, for example in the case of the deadline overrun problem by shedding low-priority tasks or by pre-empting them for tasks which are about to overrun.

References

[1] "Microsoft Expands Xbox 360 Warranty Coverage," *http://www.microsoft.com/Presspass/press/2007/jul07/07-05WarrantyExtentionPR.mspx*, 2007, Microsoft Corporation.

[2] "homeostasis," in *Merriam-Webster Online Dictionary*: http://www.merriam-webster.com/dictionary/homeostasis, 2009.

[3] U. Aickelin, P. Bentley, S. Cayzer, J. Kim, and J. McLeod, "Danger Theory: The Link between AIS and IDS?" in proceedings of *International Conference on Artificial Immune Systems (ICARIS) 2003*, LNCS vol. 2787, pp 147-155, Springer-Verlag.

[4] U. Aickelin and S. Cayzer, "The Danger Theory and Its Application to Artificial Immune Systems," in proceedings of *International Conference on Artificial Immune Systems (ICARIS) 2002*, pp 141-148, University of Kent at Canterbury.

[5] R. K. Allen, K. Bluff, and A. B. Oppenheim, "Jumping into Java: object-oriented software development for the masses," in proceedings of *3rd Australasian Conference on Computer Science Education 1998*, pp 165-172, ACM.

[6] P. S. Andrews and J. Timmis, "Inspiration for the Next Generation of Artificial Immune Systems," in proceedings of *International Conference on Artificial Immune Systems (ICARIS) 2005*, LNCS vol. 3627, pp 126-138, Springer-Verlag.

[7] P. J. Angeline, "Genetic programming and emergent intelligence," in *Advances in genetic programming*, K. E. Kinnear, Ed: MIT Press, 1994, pp. 75-98.

[8] M. Ayara, J. Timmis, R. de Lemos, and S. Forrest, "Immunising Automated Teller Machines," in proceedings of *International Conference on Artificial Immune Systems (ICARIS) 2005*, LNCS vol. 3627, pp 404-417, Springer-Verlag.

[9] T. Bäck, U. Hammel, and H. P. Schwefel, "Evolutionary computation: Comments on the history and current state," *Transactions on Evolutionary Computation*, vol. 1, pp. 3-17, 1997, IEEE.

[10] S. Balsamo, A. Di Marco, P. Inverardi, and M. Simeoni, "Model-Based Performance Prediction in Software Development: A Survey," *Transactions on Software Engineering*, vol. 30, pp. 295-310, 2004, IEEE.

[11] A. Baresel, H. Sthamer, and M. Schmidt, "Fitness function design to improve evolutionary structural testing," in proceedings of *Genetic and Evolutionary Computation Conference (GECCO) 2002*, pp 1329-1336, Morgan Kaufmann.

[12] J. Barnes, *Programming in Ada95*: Addison Wesley Longman, 1995.

[13] I. Bate, "Dealing with Emergent Properties in Embedded Systems," in proceedings of *11th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications 2005*, pp 63-66, IEEE.

[14] D. Batory and S. O'Malley, "The design and implementation of hierarchical software systems with reusable components," *Transactions on Software Engineering Methodology*, vol. 1, pp. 355-398, 1992, ACM.

[15] R. Beckers, O. E. Holland, and J. L. Deneubourg, "From local actions to global tasks: Stigmergy and collective robotics," in proceedings of *Artificial life IV 1994*, pp 181-189, MIT Press.

[16] E. Bondarev, J. Muskens, P. de With, M. Chaudron, and J. Lukkien, "Predicting Real-Time Properties of Component Assemblies: a Scenario-Simulation Approach," in proceedings of *30th Euromicro Conference 2004*, pp 40-47, IEEE.

[17] B. Bouyssounouse and J. Sifakis, *Embedded Systems Design: The ARTIST Roadmap for Research and Development*, LNCS vol. 3436: Springer-Verlag, 2005.

[18] G. E. P. Box, "Evolutionary operation: A method for increasing industrial productivity," *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, vol. 6, pp. 81-101, 1957, Royal Statistical Society.

[19] H. J. Bremermann, "Optimization through evolution and recombination," in *Self-organizing systems*, M. C. Yovits, G. Jacobi, and G. Goldstein, Eds: Spartan Books, 1962, pp. 93-106.

[20] S. Bullock and D. Cliff, *Complexity and Emergent Behaviour in ICT Systems*, HP Technical Reports vol. HPL-2004-187: Hewlett-Packard Labs, 2004.

[21] F. M. Burnet, *The clonal selection theory of acquired immunity*: Vanderbilt University Press, 1959.

[22] F. M. Burnet, "Evolution of the Immune Process in Vertebrates," *Nature*, vol. 218, pp. 426-430, 1968.

[23] F. M. Burnet, "Clonal Selection and After," in *Theoretical Immunology*, G. I. Bell, A. S. Perelson, and G. H. Pimbley Jr, Eds: Marcel Dekker, 1978, pp. 63-85.

[24] A. Burns and A. Wellings, *Real-Time Systems and Programming Languages*, 3rd ed: Pearson Education, 2001.

[25] G. C. Buttazzo and F. Sensini, "Optimal deadline assignment for scheduling soft aperiodic tasks in hard real-time environments," *Transactions on Computers*, vol. 48, pp. 1035, 1999, IEEE.

[26] J. H. Carter, "The Immune System as a Model for Pattern Recognition and Classification," *Journal of the American Medical Informatics Association*, vol. 7, pp. 28-41, 2000.

[27] J. Catsoulis, *Designing embedded hardware*, 2nd ed: O'Reilly Media, Inc., 2005.

[28] G. P. Coelho and F. J. Von Zuben, "Omni-aiNet: An Immune-Inspired Approach for Omni Optimization," in proceedings of *International Conference on Artificial Immune Systems (ICARIS) 2006*, LNCS vol. 4163, pp 294-308, Springer-Verlag.

[29] C. Coello, D. Rivera, and N. Cortas, "Use of an Artificial Immune System for Job Shop Scheduling," in proceedings of *International Conference on Artificial Immune Systems (ICARIS) 2003*, LNCS vol. 2787, pp 1-10, Springer-Verlag.

[30] I. R. Cohen, "The cognitive principle challenges clonal selection," *Immunology Today*, vol. 13, pp. 441, 1992, Elsevier.

[31] I. R. Cohen, *Tending Adam's garden: evolving the cognitive immune self*: Academic Press, 2000.

[32] A. D. Daga, L. Mize, S. Sripada, C. Wolff, and W. Wu, "Automated Timing Model Generation," in proceedings of *Design Automation Conference (DAC) 2002*, pp 146-151, ACM.

[33] D. Dasgupta, *Artificial Immune Systems and Their Applications*: Springer-Verlag, 1998.

[34] D. Dasgupta, "Advances in artificial immune systems," *IEEE Computational Intelligence Magazine*, vol. 1, pp. 40-49, 2006, IEEE.

[35] D. Dasgupta, Y. Cao, and C. Yang, "An immunogenetic approach to spectra recognition," in proceedings of *Genetic and Evolutionary Computation Conference (GECCO) 1999*, pp 149-155, Morgan Kaufmann.

[36] L. N. de Castro and J. Timmis, "An artificial immune network for multimodal function optimization," in proceedings of *Congress on Evolutionary Computation (CEC) 2002*, vol. 1, pp 699-674, IEEE.

[37] L. N. de Castro and J. Timmis, *Artificial Immune Systems: A New Computational Intelligence Approach*: Springer-Verlag, 2002.

[38] L. N. de Castro and F. J. Von Zuben, "The clonal selection algorithm with engineering applications," in proceedings of *Genetic and Evolutionary Computation Conference (GECCO) 2000*, pp 36-39, Morgan Kaufmann.

[39] L. N. de Castro and F. J. Von Zuben, "An Evolutionary Immune Network for Data Clustering," in proceedings of *Brazilian Symposium on Neural Networks 2000*, pp 84-89, IEEE.

[40] L. N. de Castro and F. J. Von Zuben, "Learning and optimization using the clonal selection principle," *Transactions on Evolutionary Computation*, vol. 6, pp. 239-251, 2002, IEEE.

[41] L. N. de Castro and F. J. Von Zuben, *Recent developments in biologically inspired computing*: Idea Group Publishing, 2005.

[42] E. W. Dijkstra, "The structure of the “THE”-multiprogramming system," *Communications of the ACM*, vol. 11, pp. 341-346, 1968, ACM.

[43] E. W. Dijkstra, "Structured Programming," in proceedings of *NATO Working Conference on Software Engineering 1969*, NATO Science Committee.

[44] E. W. Dijkstra, *Structured programming*: EWD268 (unpublished), 1969.

[45] E. W. Dijkstra, *Notes on Structured Programming*: EWD249 (unpublished), 1970.

[46] S. Dutta, R. Jensen, and A. Rieckmann, "Viper: A Multiprocessor SOC for Advanced Set-Top Box and Digital TV Systems," *Design and Test of Computers*, vol. 18, pp. 21-31, 2001, IEEE.

[47] L. D. J. Eggermont, *Embedded Systems Roadmap - Vision on technology for the future of PROGRESS*: STW Technology Foundation, 2002.

[48] M. Eisenring, L. Thiele, and E. Zitzler, "Conflicting criteria in embedded system design," *Design & Test of Computers*, vol. 17, pp. 51-59, 2000, IEEE.

[49] S. Endoh, N. Toma, and K. Yamada, "Immune algorithm for n-TSP," in proceedings of *International Conference on Systems, Man, and Cybernetics 1998*, vol. 4, pp 3844-3849, IEEE.

[50] J. Engblom, "Analysis of the execution time unpredictability caused by dynamic branch prediction," in proceedings of *Real-Time and Embedded Technology and Applications Symposium (RTAS) 2003*, pp 152-159, IEEE.

[51] O. Engin and A. Döyen, "A new approach to solve hybrid flow shop scheduling problems by artificial immune system," *Future Generation Computer Systems*, vol. 20, pp. 1083, 2004, Elsevier.

[52] R. Ernst, "Codesign of embedded systems: status and trends," *Design & Test of Computers*, vol. 15, pp. 45-54, 1998, IEEE.

[53] J. D. Farmer, N. H. Packard, and A. S. Perelson, "The immune system, adaptation, and machine learning," *Physica D*, vol. 2, pp. 187-204, 1986, Elsevier.

[54] A. Field and G. Hole, *How to Design and Report Experiments*: SAGE Publications, 2003.

[55] A. V. Fioukov, E. M. Eskenazi, D. K. Hammer, and M. R. V. Chaudron, "Evaluation of Static Properties for Component-Based Architectures," in proceedings of *28th Euromicro Conference 2002*, pp 33-39, IEEE.

[56] S. Forrest, S. A. Hofmeyr, A. Somayaji, and T. A. Longstaff, "A Sense of Self for Unix Processes," in proceedings of *IEEE Symposium on Security and Privacy 1996*, pp 120-128, IEEE.

[57] S. Forrest, B. Javornik, R. E. Smith, and A. S. Perelson, "Using genetic algorithms to explore pattern recognition in the immune system," *Evolutionary computation*, vol. 1, pp. 191-211, 1993, MIT Press.

[58] S. Forrest, A. S. Perelson, L. Allen, and R. Cherukuri, "Self-Nonself Discrimination in a Computer," in proceedings of *IEEE Symposium on Security and Privacy 1994*, pp 202-212, IEEE.

[59] A. A. Freitas and J. Timmis, "Revisiting the Foundations of Artificial Immune Systems: A Problem-oriented Perspective," in proceedings of *International*

Conference on Artificial Immune Systems (ICARIS) 2003, LNCS vol. 2787, pp 229-241, Springer-Verlag.

[60] A. A. Freitas and J. Timmis, "Revisiting the foundations of artificial immune systems for data mining," *Transactions on Evolutionary Computation*, vol. 11, pp. 521-540, 2007, IEEE.

[61] R. M. Friedberg, "A learning machine: Part I," *IBM Journal of Research and Development*, vol. 2, pp. 2-13, 1958, IBM.

[62] J. Fromm, "Types and Forms of Emergence," *http://arxiv.org/abs/nlin.AO/0506028*, 2005.

[63] S. Gallucci and P. Matzinger, "Danger signals: SOS to the immune system," *Current opinion in immunology*, vol. 13, pp. 114-119, 2001, Elsevier.

[64] D. Garlan and M. Shaw, "An Introduction to Software Architecture," in *Advances in Software Engineering and Knowledge Engineering*, G. Tortora, Ed: World Scientific Publishing Company, 1993, pp. 1-39.

[65] W. W. Gibbs, "Trends in Computing: Software's chronic crisis," *Scientific American*, vol. 271, pp. 72-81, 1994, Holtzbrinck.

[66] B. Graaf, M. Lormans, and H. Toetenel, "Embedded software engineering: the state of the practice," *Software*, vol. 20, pp. 61-69, 2003, IEEE.

[67] R. L. Graham, "Bounds on Multiprocessing Timing Anomalies," *SIAM Journal on Applied Mathematics*, vol. 17, pp. 416-429, 1969, SIAM.

[68] D. G. Green, "Emergent behavior in biological systems," in *Complex Systems - From Biology to Computation*, D. G. Green and T. Bossomaier, Eds: IOS Press, 1993, pp. 24-35.

[69] J. Greensmith, "The Dendritic Cell Algorithm," PhD Thesis, University of Nottingham, 2007.

[70] J. Greensmith and U. Aickelin, "Dendritic cells for SYN scan detection," in proceedings of *Genetic and Evolutionary Computation Conference (GECCO) 2007*, pp 49-56, ACM.

[71] J. Greensmith and U. Aickelin, "The Deterministic Dendritic Cell Algorithm," in proceedings of *International Conference on Artificial Immune Systems (ICARIS) 2008*, LNCS vol. 5132, pp 291-302, Springer-Verlag.

[72] J. Greensmith, U. Aickelin, and S. Cayzer, "Introducing Dendritic Cells as a Novel Immune-Inspired Algorithm for Anomaly Detection," in proceedings of *International Conference on Artificial Immune Systems (ICARIS) 2005*, LNCS vol. 3627, pp 153-167, Springer-Verlag.

[73] J. Greensmith, U. Aickelin, and J. Twycross, "Detecting Danger: Applying a Novel Immunological Concept to Intrusion Detection Systems," in proceedings of *International Conference in Adaptive Computing in Design and Manufacture 2004*, Springer-Verlag.

[74] J. Greensmith, U. Aickelin, and J. Twycross, "Articulation and Clarification of the Dendritic Cell Algorithm," in proceedings of *International Conference on Artificial Immune Systems (ICARIS) 2006*, LNCS vol. 4163, pp 404-417, Springer-Verlag.

[75] F. Gu, J. Greensmith, and U. Aickelin, "Exploration of the Dendritic Cell Algorithm using the Duration Calculus," in proceedings of *International Conference on Artificial Immune Systems (ICARIS) 2009*, LNCS vol. 5666, pp 54-66, Springer-Verlag.

[76] J. Gustafsson and A. Ermedahl, "Experiences from Applying WCET Analysis in Industrial Settings," in proceedings of *International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC) 2007*, pp 382-392, IEEE.

[77] E. Hart, "Not All Balls Are Round: An Investigation of Alternative Recognition-Region Shapes," in proceedings of *International Conference on Artificial Immune Systems (ICARIS) 2005*, LNCS vol. 3627, pp 29-42, Springer-Verlag.

[78] E. Hart and P. Ross, "An Immune System Approach to Scheduling in Changing Environments," in proceedings of *Genetic and Evolutionary Computation Conference (GECCO) 1999*, pp 1559-1566, Morgan Kaufmann.

[79] E. Hart, P. Ross, and J. Nelson, "Producing robust schedules via an artificial immune system," in proceedings of *World Congress on Computational Intelligence (WCCI) 1998*, pp 464-469, IEEE.

[80] E. Hart and J. Timmis, "Application Areas of AIS: The Past, The Present and The Future," in proceedings of *International Conference on Artificial Immune Systems (ICARIS) 2005*, LNCS vol. 3627, pp 483-497, Springer-Verlag.

[81] E. Hart and J. Timmis, "Application areas of AIS: The past, the present and the future," *Applied Soft Computing Journal*, vol. 8, pp. 191-201, 2008, Elsevier.

[82] P. K. Harter, Jr, "Response Times in Level-Structured Systems," *Transactions on Computer Systems*, vol. 5, pp. 232-248, 1987, ACM.

[83] S. Hissam, G. Moreno, J. Stafford, and K. Wallnau, "Enabling predictable assembly," *The Journal of Systems and Software*, vol. 65, pp. 185-198, 2003, Elsevier.

[84] S. A. Hofmeyr and S. Forrest, "Immunity by design: an artificial immune system," in proceedings of *Genetic and Evolutionary Computation Conference (GECCO) 1999*, pp 1296-1303, Morgan Kaufmann.

[85] N. Holsti and S. Saarinen, "Status of the Bound-T WCET Tool," in proceedings of *2nd International Workshop on Worst-Case Execution Time Analysis 2002*, vol. YCS-2002-346, University of York.

[86] J. E. Hunt and D. E. Cooke, "Learning using an artificial immune system," *Journal of Network and Computer Applications*, vol. 19, pp. 189-212, 1996, Academic Press.

[87] C. A. Janeway, Jr., "Approaching the asymptote? Evolution and revolution in immunology," in proceedings of *Cold Spring Harbor Symposium on Quantitative Biology 1989*, vol. 54, pp 1-13, Cold Spring Harbor Laboratory Press.

[88] C. A. Janeway, Jr., "The immune system evolved to discriminate infectious nonself from noninfectious self," *Immunology Today*, vol. 13, pp. 11-16, 1992, Elsevier.

[89] C. A. Janeway, Jr., "How the immune system works to protect the host from infection: A personal view," *Proceedings of the National Academy of Sciences*, vol. 98, pp. 7461-7468, 2001, National Academy of Sciences.

[90] N. K. Jerne, "Towards a network theory of the immune system," *Annals of Immunology*, vol. 125C, pp. 373-389, 1974.

[91] Z. Ji and D. Dasgupta, "Applicability issues of the real-valued negative selection algorithms," in proceedings of *Genetic and Evolutionary Computation Conference (GECCO) 2006*, pp 111-118, ACM.

[92] Z. Ji and D. Dasgupta, "Revisiting negative selection algorithms," *Evolutionary Computation*, vol. 15, pp. 223-251, 2007, MIT Press.

[93] D. I. Katcher, H. Arakawa, and J. K. Strosnider, "Engineering and analysis of fixed priority schedulers," *Transactions on Software Engineering*, vol. 19, pp. 920-934, 1993, IEEE.

[94] J. Kelsey and J. Timmis, "Immune inspired somatic contiguous hypermutation for function optimisation," in proceedings of *Genetic and Evolutionary Computation Conference (GECCO) 2003*, LNCS vol. 2723, pp 207-218, Springer-Verlag.

[95] J. O. Kephart, "A Biologically Inspired Immune System for Computers," in proceedings of *Artificial Life IV: Fourth International Workshop on the Synthesis and Simulation of Living Systems 1994*, pp 130-139, MIT Press.

[96] J. Kim, P. Bentley, C. Wallenta, M. Ahmed, and S. Hailes, "Danger Is Ubiquitous: Detecting Malicious Activities in Sensor Networks Using the Dendritic Cell Algorithm," in proceedings of *International Conference on Artificial Immune Systems (ICARIS) 2006*, LNCS vol. 4163, pp 390-403, Springer-Verlag.

[97] J. Kim and P. J. Bentley, "An evaluation of negative selection in an artificial immune system for network intrusion detection," in proceedings of *Genetic and Evolutionary Computation Conference (GECCO) 2001*, pp 1330–1337, Morgan Kaufmann.

[98] C. W. Krueger, "Software Reuse," *ACM Computing Surveys*, vol. 24, pp. 131-183, 1992, ACM.

[99] A. Kubík, "Toward a formalization of emergence," *Artificial Life*, vol. 9, pp. 41-65, 2003, MIT Press.

[100] N. Lay and I. Bate, "Applying Artificial Immune Systems to Real-Time Embedded Systems," in proceedings of *Congress on Evolutionary Computation (CEC) 2007*, pp 3743-3750, IEEE.

[101] N. Lay and I. Bate, "Improving the reliability of real-time embedded systems using innate immune techniques," *Evolutionary Intelligence*, vol. 1, pp. 113, 2008, Springer-Verlag.

[102] E. A. Lee, "What's ahead for embedded software?" *Computer*, vol. 33, pp. 18-26, 2000, IEEE.

[103] B. Lisper, "Fully Automatic, Parametric Worst-Case Execution Time Analysis," in proceedings of *Workshop on Worst-Case Execution Time 2003*, pp 99-102, IEEE.

[104] C. L. Liu and J. W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment," *Journal of the ACM*, vol. 20, pp. 40-61, 1973, ACM.

[105] T. Lundqvist and P. Stenstrom, "Timing anomalies in dynamically scheduled microprocessors," in proceedings of *Real-Time Systems Symposium (RTSS) 1999*, pp 12-21, IEEE.

[106] C. R. Mackay, "Immunological memory," *Advances in Immunology*, vol. 53, pp. 217-265, 1993, Academic Press.

[107] P. Matzinger, "Tolerance, Danger, and the Extended Family," *Annual Review of Immunology*, vol. 12, pp. 991-1045, 1994, Annual Reviews.

[108] P. Matzinger, "The Danger Model: A Renewed Sense of Self," *Science*, vol. 296, pp. 301-305, 2002, AAAS.

[109] R. Medzhitov and C. A. Janeway, Jr., "Innate immunity: impact on the adaptive immune response," *Current Opinion in Immunology*, vol. 9, pp. 4-9, 1997, Elsevier.

[110] R. Medzhitov and C. A. Janeway, Jr., "Innate immune recognition and control of adaptive immune responses," *Seminars in Immunology*, vol. 10, pp. 351-353, 1998, Elsevier.

[111] R. T. Monroe and D. Garlan, "Style-Based Reuse for Software Architectures," in proceedings of *International Conference on Software Reuse 1996*, pp 84-93, IEEE.

[112] D. C. Montgomery, *Design and Analysis of Experiments*, 6th ed: Wiley, 2005.

[113] C. Müller-Schloer, "Organic Computing - On the Feasibility of Controlled Emergence," in proceedings of *International Conference on Hardware Software Codesign and System Synthesis (CODES) 2004*, pp 2-5, ACM.

[114] M. Neal, J. Feyereisl, R. Rascunà, and X. Wang, "Don't Touch Me, I'm Fine: Robot Autonomy Using an Artificial Innate Immune System," in proceedings of *International Conference on Artificial Immune Systems (ICARIS) 2006*, LNCS vol. 4163, pp 349-361, Springer-Verlag.

[115] R. Oates, J. Greensmith, U. Aickelin, J. Garibaldi, and G. Kendall, "The Application of a Dendritic Cell Algorithm to a Robotic Classifier," in proceedings of *International Conference on Artificial Immune Systems (ICARIS) 2007*, LNCS vol. 4628, pp 204-215, Springer-Verlag.

[116] N. D. Owens, J. Timmis, A. J. Greensted, and A. M. Tyrell, "On immune inspired homeostasis for electronic systems," in proceedings of *International*

Conference on Artificial Immune Systems (ICARIS) 2007, LNCS vol. 4628, pp 216-227, Springer-Verlag.

[117] D. L. Parnas, "On the criteria to be used in decomposing systems into modules," *Communications of the ACM*, vol. 15, pp. 1053-1058, 1972, ACM.

[118] P. Puschner and A. Burns, "A Review of Worst-Case Execution-Time Analysis," *Real-Time Systems*, vol. 18, pp. 115-128, 2000, Springer-Verlag.

[119] M. Ravasi and M. Mattavelli, "High-level algorithmic complexity evaluation for system design," *Journal of Systems Architecture*, vol. 48, pp. 403, 2003, Elsevier.

[120] E. Rechtin, "The synthesis of complex systems," *Spectrum*, vol. 34, pp. 51-55, 1997, IEEE.

[121] J. T. Russell and M. F. Jacome, "Architecture-Level Performance Evaluation of Component-Based Embedded Systems," in proceedings of *Design Automation Conference (DAC) 2003*, pp 396-401, ACM.

[122] T. Schöler and C. Müller-Schloer, "First Steps towards Organic Computing Systems - Monitoring an Adaptive Protocol Stack with a Fuzzy Classifier System," in proceedings of *2nd Conference On Computing Frontiers 2005*, pp 10-20, ACM.

[123] A. Secker, A. A. Freitas, and J. Timmis, "AISEC: an artificial immune system for e-mail classification," in proceedings of *Congress on Evolutionary Computation (CEC) 2003*, pp 131-138, IEEE.

[124] M. Shaw, "Architectural issues in software reuse: it's not just the functionality, it's the packaging," in proceedings of *Symposium on Software Reusability 1995*, pp 3-6, ACM.

[125] A. M. Silverstein, "The Clonal Selection Theory: what it really is and why modern challenges are misplaced," *Nature Immunology*, vol. 3, pp. 793-796, 2002, Nature Publishing Group.

[126] A. B. Somayaji, "Operating system stability and security through process homeostasis," PhD Thesis, The University of New Mexico, 2002.

[127] B. Sprunt, L. Sha, and J. Lehoczky, "Aperiodic task scheduling for Hard-Real-Time systems," *Real-Time Systems*, vol. 1, pp. 27-60, 1989, Springer-Verlag.

[128] F. Stappert and P. Altenbernd, "Complete worst-case execution time analysis of straight-line hard real-time programs," *Journal of Systems Architecture*, vol. 46, pp. 339-356, 2000, Elsevier.

[129] S. Stepney, S. L. Braunstein, J. A. Clark, A. Tyrrell, A. Adamatzky, R. E. Smith, T. Addis, C. Johnson, J. Timmis, and P. Welch, "Journeys in non-classical computation I: A grand challenge for computing research," *International Journal of Parallel, Emergent and Distributed Systems*, vol. 20, pp. 5-19, 2005, Taylor & Francis.

[130] T. Stibor, P. Mohr, J. Timmis, and C. Eckert, "Is negative selection appropriate for anomaly detection?" in proceedings of *Genetic and Evolutionary Computation Conference (GECCO) 2005*, pp 321-328, ACM.

[131] T. Stibor and J. Timmis, "Comments on real-valued negative selection vs. real-valued positive selection and one-class SVM," in proceedings of *Congress on Evolutionary Computation (CEC) 2007*, pp 3727-3734, IEEE.

[132] T. Stibor, J. Timmis, and C. Eckert, "On the appropriateness of negative selection defined over hamming shape-space as a network intrusion detection system," in proceedings of *Congress on Evolutionary Computation (CEC) 2005*, vol. 2, pp 995-1002, IEEE.

[133] T. Stibor, J. Timmis, and C. Eckert, "On the Use of Hyperspheres in Artificial Immune Systems as Antibody Recognition Regions," in proceedings of *International Conference on Artificial Immune Systems (ICARIS) 2006*, LNCS vol. 4163, pp 215-228, Springer-Verlag.

[134] H. Theiling, C. Ferdinand, and R. Wilhelm, "Fast and Precise WCET Prediction by Separated Cache and Path Analyses," *Real-Time Systems*, vol. 18, pp. 157-179, 2000, Springer-Verlag.

[135] J. Timmis, "An Introduction to Artificial Immune Systems (ICARIS tutorial presentation)," *http://www.artificial-immune-systems.org/ICARIS-2004/tutorial-timmis.ppt*, 2004.

[136] J. Timmis, "Artificial immune systems—today and tomorrow," *Natural Computing*, vol. 6, pp. 1-18, 2007, Springer-Verlag.

[137] J. Timmis, P. Andrews, N. Owens, and E. Clark, "An interdisciplinary perspective on artificial immune systems," *Evolutionary Intelligence*, vol. 1, pp. 5-26, 2008, Springer-Verlag.

[138] J. Timmis, C. Edmonds, and J. Kelsey, "Assessing the performance of two immune inspired algorithms and a hybrid genetic algorithm for function optimisation," in proceedings of *Congress on Evolutionary Computation (CEC) 2004*, vol. 1, pp 1044-1051, IEEE.

[139] J. Timmis and M. Neal, "A resource limited artificial immune system for data analysis," *Knowledge-Based Systems*, vol. 14, pp. 121-130, 2001, Elsevier Science.

[140] N. Toma, S. Endo, and K. Yamanda, "Immune algorithm with immune network and MHC for adaptive problem solving," in proceedings of *International Conference on Systems, Man, and Cybernetics 1999*, vol. 4, pp 271-276, IEEE.

[141] R. Torkar, "Dynamic Software Architectures," in *Building Reliable Component-based Systems*, I. Crnkovic and M. Larsson, Eds: Artech House Publishers, 2002, pp. 21–28.

[142] J. Twycross and U. Aickelin, "Towards a conceptual framework for innate immunity," in proceedings of *International Conference on Artificial Immune Systems (ICARIS) 2005*, LNCS vol. 3627, pp 112–125, Springer-Verlag.

[143] N. Urquhart, K. Chisholm, and B. Paechter, "Optimising an Evolutionary Algorithm for Scheduling," in proceedings of *Real-World Applications of Evolutionary Computing, EvoWorkshops 2000*, LNCS vol. 1803, pp 307-318, Springer-Verlag.

[144] F. Vahid and T. D. Givargis, *Embedded System Design: A Unified Hardware/Software Introduction*: Wiley, 2002.

[145] R. van Ommering, "Koala, a Component Model for Consumer Electronics Product Software," in proceedings of *2nd International ESPRIT ARES Workshop on Development and Evolution of Software Architectures for Product Families 1998*, LNCS vol. 1429, pp 76-86, Springer-Verlag.

[146] R. van Ommering, "Building product populations with software components," in proceedings of *International Conference on Software Engineering (ICSE) 2002*, pp 255-265, ACM.

[147] R. van Ommering, "Configuration Management in Component Based Product Populations," in proceedings of *International Workshop on Software Configuration Management (SCM) 2003*, LNCS vol. 2649, pp 16-23, Springer-Verlag.

[148] R. van Ommering, "Software reuse in product populations," *Transactions on Software Engineering*, vol. 31, pp. 537-550, 2005, IEEE.

[149] R. van Ommering, F. van der Linden, J. Kramer, and J. Magee, "The Koala Component Model for Consumer Electronics Software," *Computer*, vol. 33, pp. 78-85, 2000, IEEE.

[150] E.-C. Vladmir and W. Derick, "A survey of adaptive sorting algorithms," *ACM Computing Surveys*, vol. 24, pp. 441-476, 1992, ACM.

[151] X. Wang, X. Z. Gao, and S. J. Ovaska, "Artificial immune optimization methods and applications - a survey," in proceedings of *International Conference on Systems, Man and Cybernetics 2004*, vol. 4, pp 3415-3420, IEEE.

[152] M. A. Weiss, *Data Structures and Algorithm Analysis in Ada*: Benjamin/Cummings, 1993.

[153] R. L. Wilder, "Neuroendocrine-Immune System Interactions and Autoimmunity," *Annual Review of Immunology*, vol. 13, pp. 307-338, 1995, Annual Reviews.

[154] X. Zhang, G. Dragffy, A. G. Pipe, and Q. M. Zhu, "Artificial innate immune system: An instant defence layer of embryonics," in proceedings of *International Conference on Artificial Immune Systems (ICARIS) 2005*, LNCS vol. 3239, pp 302–315, Springer-Verlag.

[155] R. M. Zinkernagel, M. F. Bachmann, T. M. Kundig, S. Oehen, H. Pirchet, and H. Hengartner, "On immunological memory," *Annual Review of Immunology*, vol. 14, pp. 333-367, 1996, Annual Reviews.

A1

Appendix 1: Online results archive

The results presented in this thesis are combined results across a number of randomly generated task sets, allowing general trends to be established which are common across the range of evaluated results. For reasons of space, it is not possible to include results for each individual task set within this document. For the purposes of further reference, these are included in an online archive which is available at

http://www.nicklay.co.uk/thesis/

The online appendix also includes the source code for the simulation environment used to generate the results, along with details of the recommended environment for its compilation and execution.

A2

Appendix 2: Convergence tables

This appendix contains tables showing the full set of convergence data used to select the simulation parameters in section 6.3.2.

Table A1: Average actual overrun accuracy over all test configurations: 10-task sets, absolute values

Cycles	1	2	3	4	5	6	7	8	9	10
60000	0.8337	0.8231	0.8120	0.8048	0.7951	0.7927	0.7788	0.7807	0.7814	0.7815
80000	0.8307	0.8188	0.8119	0.8067	0.7980	0.7979	0.7892	0.7918	0.7922	0.7919
100000	0.8071	0.8107	0.8026	0.8011	0.7914	0.7940	0.7887	0.7915	0.7916	0.7919
120000	0.8096	0.8098	0.8043	0.8044	0.7940	0.7950	0.7911	0.7931	0.7934	0.7935
140000	0.8115	0.7930	0.7960	0.7969	0.7932	0.7936	0.7900	0.7919	0.7919	0.7917
160000	0.8110	0.7959	0.7984	0.7976	0.7945	0.7935	0.7914	0.7933	0.7935	0.7933
180000	0.8133	0.7991	0.8010	0.8003	0.7972	0.7954	0.7936	0.7954	0.7953	0.7950
200000	0.8159	0.8010	0.8028	0.8022	0.7992	0.7977	0.7963	0.7979	0.7979	0.7975
220000	0.8156	0.8013	0.8035	0.8029	0.7997	0.7985	0.7973	0.7991	0.7996	0.7996
240000	0.8128	0.8017	0.8039	0.8033	0.8004	0.7992	0.7982	0.8000	0.8006	0.8007
260000	0.8134	0.8031	0.8050	0.8047	0.8017	0.8008	0.7997	0.8015	0.8021	0.8011
280000	0.8136	0.8040	0.8057	0.8058	0.8029	0.8016	0.8007	0.8024	0.8031	0.8022
300000	0.8152	0.7979	0.8007	0.8022	0.8001	0.7998	0.7984	0.8005	0.8013	0.8007
320000	0.8163	0.7998	0.8022	0.8038	0.8015	0.8011	0.7999	0.8014	0.8024	0.8018
340000	0.8162	0.7984	0.7993	0.8014	0.8002	0.7991	0.7986	0.8002	0.8013	0.8010
360000	0.8165	0.7983	0.7996	0.8019	0.8007	0.7993	0.7989	0.8005	0.8015	0.8015
380000	0.8160	0.7989	0.8002	0.8021	0.8007	0.7992	0.7990	0.8005	0.8015	0.8014
400000	0.8138	0.7980	0.7996	0.8017	0.8004	0.7992	0.7991	0.8006	0.8014	0.8010
420000	0.8135	0.7975	0.7995	0.8017	0.7997	0.7991	0.7989	0.8001	0.8011	0.8009
440000	0.8126	0.7959	0.7988	0.8004	0.7989	0.7985	0.7983	0.7997	0.8005	0.8003
460000	0.8129	0.7967	0.7993	0.8008	0.7990	0.7987	0.7986	0.8000	0.8005	0.8004
480000	0.8135	0.7980	0.8003	0.8017	0.7999	0.7995	0.7994	0.8006	0.8012	0.8012
500000	0.8092	0.7957	0.7981	0.7994	0.7975	0.7976	0.7975	0.7988	0.7993	0.7993
520000	0.8098	0.7964	0.7989	0.8002	0.7984	0.7984	0.7984	0.7996	0.8000	0.8001
540000	0.8109	0.7970	0.7994	0.8008	0.7988	0.7987	0.7988	0.7998	0.8001	0.8001
560000	0.8104	0.7976	0.7999	0.8013	0.7994	0.7994	0.7994	0.8003	0.8007	0.8005
580000	0.8107	0.7981	0.8003	0.8018	0.7998	0.7999	0.7998	0.8008	0.8011	0.8010
600000	0.8104	0.7981	0.8001	0.8019	0.8000	0.8002	0.8002	0.8011	0.8015	0.8013
620000	0.8053	0.7961	0.7982	0.8004	0.7990	0.7988	0.7990	0.7999	0.8004	0.8003
640000	0.8050	0.7962	0.7983	0.8005	0.7990	0.7989	0.7992	0.8000	0.8005	0.8004
660000	0.8050	0.7946	0.7969	0.7991	0.7980	0.7971	0.7976	0.7986	0.7993	0.7993
680000	0.8047	0.7948	0.7971	0.7993	0.7979	0.7964	0.7970	0.7980	0.7988	0.7989
700000	0.8048	0.7949	0.7971	0.7995	0.7983	0.7964	0.7970	0.7978	0.7986	0.7985
720000	0.8049	0.7954	0.7976	0.8001	0.7986	0.7967	0.7973	0.7980	0.7989	0.7988
740000	0.8045	0.7951	0.7975	0.8000	0.7985	0.7967	0.7972	0.7979	0.7987	0.7986
760000	0.8052	0.7956	0.7980	0.8004	0.7991	0.7972	0.7977	0.7983	0.7991	0.7991
780000	0.8053	0.7963	0.7986	0.8011	0.7996	0.7978	0.7984	0.7990	0.7996	0.7995
800000	0.8034	0.7960	0.7982	0.8007	0.7994	0.7978	0.7984	0.7989	0.7995	0.7995
820000	0.8040	0.7966	0.7988	0.8011	0.7999	0.7982	0.7987	0.7990	0.7997	0.7997
840000	0.8037	0.7968	0.7988	0.8007	0.7996	0.7982	0.7987	0.7991	0.7997	0.7997
860000	0.8039	0.7969	0.7988	0.8007	0.7997	0.7982	0.7988	0.7993	0.8000	0.8000
880000	0.8045	0.7974	0.7994	0.8013	0.8000	0.7986	0.7992	0.7997	0.8005	0.8005
900000	0.8050	0.7977	0.7996	0.8015	0.8002	0.7988	0.7994	0.7999	0.8007	0.8008
920000	0.8054	0.7982	0.8000	0.8018	0.8005	0.7991	0.7997	0.8002	0.8010	0.8011
940000	0.8058	0.7985	0.8004	0.8022	0.8009	0.7995	0.8001	0.8006	0.8013	0.8015
960000	0.8045	0.7982	0.7999	0.8008	0.8000	0.7987	0.7994	0.8000	0.8006	0.8008
980000	0.8006	0.7965	0.7986	0.7996	0.7990	0.7980	0.7986	0.7992	0.7998	0.8000
1000000	0.8006	0.7965	0.7987	0.7998	0.7992	0.7982	0.7988	0.7994	0.8000	0.8001

Table A2: Average potential overrun accuracy over all test configurations: 10-task sets, absolute values

Cycles	1	2	3	4	5	6	7	8	9	10
60000	0.5114	0.5483	0.5678	0.5623	0.5620	0.5645	0.5628	0.5661	0.5695	0.5649
80000	0.5692	0.5649	0.5822	0.5818	0.5814	0.5849	0.5822	0.5868	0.5877	0.5862
100000	0.6112	0.6007	0.5985	0.6002	0.5973	0.5993	0.5992	0.6028	0.6019	0.6023
120000	0.5966	0.5973	0.6008	0.6039	0.6015	0.6027	0.6024	0.6057	0.6053	0.6059
140000	0.6153	0.6039	0.6065	0.6074	0.6060	0.6064	0.6056	0.6081	0.6066	0.6067
160000	0.6098	0.6035	0.6064	0.6070	0.6064	0.6066	0.6058	0.6082	0.6072	0.6073
180000	0.6103	0.6040	0.6047	0.6055	0.6058	0.6054	0.6055	0.6072	0.6064	0.6068
200000	0.5983	0.5895	0.5942	0.5930	0.5937	0.5922	0.5922	0.5945	0.5931	0.5935
220000	0.6113	0.6060	0.6087	0.6082	0.6076	0.6074	0.6074	0.6088	0.6085	0.6088
240000	0.6100	0.6051	0.6078	0.6072	0.6072	0.6071	0.6064	0.6078	0.6074	0.6072
260000	0.6082	0.6042	0.6077	0.6070	0.6075	0.6072	0.6068	0.6083	0.6080	0.6082
280000	0.6099	0.6057	0.6089	0.6084	0.6086	0.6083	0.6079	0.6093	0.6087	0.6092
300000	0.6098	0.6028	0.6065	0.6065	0.6062	0.6060	0.6056	0.6069	0.6064	0.6069
320000	0.6054	0.6021	0.6051	0.6053	0.6060	0.6050	0.6041	0.6054	0.6049	0.6053
340000	0.6125	0.6044	0.6074	0.6084	0.6082	0.6071	0.6070	0.6087	0.6083	0.6086
360000	0.6137	0.6066	0.6080	0.6088	0.6087	0.6080	0.6079	0.6093	0.6089	0.6091
380000	0.6139	0.6051	0.6074	0.6084	0.6077	0.6066	0.6065	0.6080	0.6078	0.6079
400000	0.6128	0.6057	0.6075	0.6086	0.6084	0.6078	0.6076	0.6090	0.6086	0.6086
420000	0.6134	0.6069	0.6092	0.6102	0.6097	0.6093	0.6090	0.6102	0.6098	0.6098
440000	0.6117	0.6058	0.6085	0.6095	0.6094	0.6092	0.6088	0.6102	0.6097	0.6097
460000	0.6106	0.6032	0.6061	0.6081	0.6085	0.6076	0.6070	0.6085	0.6077	0.6077
480000	0.6130	0.6052	0.6070	0.6078	0.6079	0.6080	0.6075	0.6089	0.6081	0.6079
500000	0.6105	0.6047	0.6067	0.6081	0.6075	0.6069	0.6066	0.6082	0.6079	0.6082
520000	0.6136	0.6078	0.6096	0.6110	0.6104	0.6100	0.6096	0.6108	0.6104	0.6107
540000	0.6153	0.6084	0.6098	0.6108	0.6101	0.6097	0.6093	0.6102	0.6098	0.6101
560000	0.6140	0.6074	0.6090	0.6103	0.6094	0.6089	0.6086	0.6095	0.6091	0.6094
580000	0.6134	0.6070	0.6087	0.6105	0.6099	0.6096	0.6090	0.6101	0.6098	0.6101
600000	0.6132	0.6074	0.6085	0.6095	0.6091	0.6089	0.6087	0.6096	0.6094	0.6094
620000	0.6111	0.6076	0.6085	0.6101	0.6098	0.6093	0.6092	0.6103	0.6101	0.6103
640000	0.6133	0.6089	0.6100	0.6115	0.6109	0.6103	0.6102	0.6111	0.6110	0.6111
660000	0.6116	0.6073	0.6086	0.6101	0.6096	0.6089	0.6088	0.6096	0.6096	0.6097
680000	0.6121	0.6080	0.6092	0.6105	0.6101	0.6093	0.6092	0.6101	0.6103	0.6105
700000	0.6121	0.6084	0.6093	0.6109	0.6107	0.6100	0.6100	0.6107	0.6108	0.6109
720000	0.6106	0.6065	0.6072	0.6089	0.6090	0.6084	0.6084	0.6092	0.6093	0.6094
740000	0.6113	0.6078	0.6088	0.6099	0.6100	0.6091	0.6093	0.6099	0.6101	0.6100
760000	0.6115	0.6079	0.6087	0.6103	0.6102	0.6094	0.6094	0.6101	0.6100	0.6099
780000	0.6108	0.6077	0.6089	0.6103	0.6102	0.6095	0.6096	0.6103	0.6103	0.6102
800000	0.6108	0.6084	0.6096	0.6110	0.6110	0.6103	0.6103	0.6109	0.6108	0.6106
820000	0.6112	0.6090	0.6101	0.6117	0.6116	0.6109	0.6108	0.6113	0.6112	0.6112
840000	0.6114	0.6090	0.6096	0.6111	0.6112	0.6103	0.6102	0.6109	0.6105	0.6105
860000	0.6103	0.6083	0.6091	0.6106	0.6105	0.6097	0.6096	0.6100	0.6099	0.6099
880000	0.6116	0.6096	0.6104	0.6117	0.6115	0.6107	0.6107	0.6114	0.6113	0.6113
900000	0.6125	0.6102	0.6108	0.6121	0.6120	0.6111	0.6111	0.6117	0.6116	0.6116
920000	0.6132	0.6107	0.6114	0.6126	0.6125	0.6116	0.6115	0.6121	0.6120	0.6120
940000	0.6140	0.6114	0.6120	0.6131	0.6130	0.6120	0.6119	0.6124	0.6123	0.6123
960000	0.6131	0.6105	0.6110	0.6120	0.6120	0.6109	0.6108	0.6113	0.6112	0.6113
980000	0.6120	0.6100	0.6105	0.6116	0.6116	0.6107	0.6106	0.6112	0.6110	0.6110
1000000	0.6132	0.6107	0.6113	0.6124	0.6123	0.6115	0.6115	0.6119	0.6117	0.6117

Table A3: Average actual overrun accuracy over all test configurations: 10-task sets, percentage variation from 10 runs/1,000,000 cycles

Cycles	1	2	3	4	5	6	7	8	9	10
60000	3.35	2.30	1.18	0.47	-0.50	-0.74	-2.14	-1.94	-1.88	-1.86
80000	3.06	1.87	1.18	0.66	-0.22	-0.22	-1.10	-0.83	-0.79	-0.82
100000	0.70	1.05	0.24	0.10	-0.88	-0.62	-1.15	-0.86	-0.85	-0.82
120000	0.94	0.97	0.42	0.42	-0.62	-0.52	-0.91	-0.71	-0.67	-0.66
140000	1.14	-0.71	-0.41	-0.32	-0.69	-0.65	-1.02	-0.83	-0.83	-0.85
160000	1.09	-0.42	-0.18	-0.26	-0.56	-0.66	-0.88	-0.68	-0.66	-0.68
180000	1.31	-0.11	0.09	0.01	-0.29	-0.47	-0.66	-0.47	-0.48	-0.51
200000	1.58	0.08	0.27	0.20	-0.10	-0.24	-0.39	-0.22	-0.23	-0.26
220000	1.54	0.11	0.33	0.28	-0.04	-0.17	-0.28	-0.10	-0.05	-0.06
240000	1.27	0.15	0.38	0.32	0.03	-0.09	-0.19	-0.01	0.05	0.05
260000	1.32	0.30	0.49	0.46	0.15	0.07	-0.05	0.13	0.20	0.09
280000	1.35	0.39	0.55	0.56	0.28	0.14	0.05	0.23	0.29	0.21
300000	1.51	-0.22	0.06	0.21	0.00	-0.04	-0.17	0.04	0.12	0.06
320000	1.62	-0.03	0.21	0.36	0.13	0.09	-0.03	0.13	0.22	0.17
340000	1.61	-0.17	-0.09	0.13	0.01	-0.10	-0.15	0.00	0.11	0.09
360000	1.64	-0.19	-0.05	0.18	0.06	-0.08	-0.12	0.04	0.14	0.14
380000	1.58	-0.12	0.01	0.20	0.05	-0.09	-0.12	0.03	0.13	0.13
400000	1.36	-0.21	-0.05	0.15	0.02	-0.09	-0.11	0.04	0.13	0.09
420000	1.33	-0.27	-0.07	0.16	-0.05	-0.10	-0.13	0.00	0.09	0.07
440000	1.24	-0.42	-0.14	0.03	-0.13	-0.17	-0.18	-0.04	0.03	0.02
460000	1.27	-0.34	-0.09	0.07	-0.11	-0.14	-0.15	-0.01	0.04	0.03
480000	1.33	-0.22	0.02	0.15	-0.03	-0.06	-0.08	0.05	0.11	0.10
500000	0.91	-0.44	-0.20	-0.07	-0.27	-0.26	-0.26	-0.13	-0.09	-0.08
520000	0.97	-0.37	-0.13	0.01	-0.18	-0.18	-0.17	-0.06	-0.01	-0.01
540000	1.08	-0.31	-0.07	0.06	-0.14	-0.14	-0.13	-0.03	0.00	0.00
560000	1.03	-0.25	-0.02	0.12	-0.07	-0.07	-0.07	0.02	0.05	0.04
580000	1.06	-0.20	0.01	0.17	-0.03	-0.03	-0.03	0.06	0.10	0.09
600000	1.03	-0.20	0.00	0.18	-0.01	0.01	0.01	0.09	0.13	0.12
620000	0.51	-0.40	-0.19	0.02	-0.11	-0.14	-0.11	-0.03	0.03	0.02
640000	0.49	-0.39	-0.18	0.03	-0.11	-0.12	-0.10	-0.01	0.04	0.03
660000	0.49	-0.55	-0.33	-0.10	-0.21	-0.31	-0.26	-0.16	-0.09	-0.08
680000	0.46	-0.54	-0.31	-0.08	-0.22	-0.37	-0.32	-0.21	-0.13	-0.12
700000	0.47	-0.52	-0.31	-0.06	-0.19	-0.37	-0.31	-0.24	-0.15	-0.16
720000	0.48	-0.48	-0.25	-0.01	-0.15	-0.35	-0.28	-0.21	-0.12	-0.13
740000	0.43	-0.51	-0.27	-0.02	-0.16	-0.35	-0.29	-0.23	-0.15	-0.15
760000	0.50	-0.45	-0.22	0.03	-0.11	-0.29	-0.24	-0.18	-0.10	-0.11
780000	0.52	-0.39	-0.15	0.09	-0.05	-0.23	-0.18	-0.12	-0.05	-0.06
800000	0.33	-0.41	-0.19	0.05	-0.07	-0.23	-0.18	-0.12	-0.06	-0.06
820000	0.39	-0.36	-0.13	0.10	-0.03	-0.19	-0.15	-0.11	-0.04	-0.04
840000	0.35	-0.33	-0.13	0.06	-0.05	-0.20	-0.15	-0.11	-0.04	-0.04
860000	0.38	-0.33	-0.13	0.06	-0.04	-0.19	-0.13	-0.09	-0.02	-0.01
880000	0.44	-0.27	-0.08	0.12	-0.01	-0.15	-0.09	-0.04	0.03	0.04
900000	0.49	-0.24	-0.05	0.14	0.01	-0.14	-0.07	-0.02	0.05	0.06
920000	0.53	-0.20	-0.01	0.17	0.04	-0.10	-0.04	0.01	0.09	0.10
940000	0.57	-0.16	0.02	0.20	0.08	-0.06	-0.01	0.04	0.12	0.13
960000	0.44	-0.19	-0.02	0.07	-0.01	-0.14	-0.07	-0.02	0.05	0.07
980000	0.05	-0.37	-0.15	-0.05	-0.11	-0.22	-0.15	-0.09	-0.04	-0.01
1000000	0.05	-0.36	-0.14	-0.04	-0.10	-0.20	-0.13	-0.07	-0.01	0.00

Table A4: Average potential overrun accuracy over all test configurations: 10-task sets, percentage variation from 10 runs/1,000,000 cycles

Cycles	1	2	3	4	5	6	7	8	9	10
60000	-10.03	-6.34	-4.39	-4.95	-4.97	-4.72	-4.90	-4.56	-4.23	-4.69
80000	-4.25	-4.69	-2.95	-3.00	-3.03	-2.68	-2.96	-2.49	-2.40	-2.55
100000	-0.05	-1.10	-1.32	-1.15	-1.45	-1.25	-1.25	-0.89	-0.98	-0.95
120000	-1.52	-1.44	-1.10	-0.79	-1.02	-0.91	-0.93	-0.60	-0.64	-0.58
140000	0.35	-0.78	-0.53	-0.44	-0.58	-0.54	-0.61	-0.37	-0.52	-0.50
160000	-0.19	-0.82	-0.53	-0.47	-0.53	-0.52	-0.60	-0.36	-0.46	-0.44
180000	-0.15	-0.77	-0.71	-0.63	-0.59	-0.63	-0.62	-0.45	-0.54	-0.49
200000	-1.34	-2.22	-1.76	-1.88	-1.81	-1.95	-1.95	-1.72	-1.86	-1.82
220000	-0.05	-0.58	-0.30	-0.35	-0.41	-0.43	-0.43	-0.29	-0.32	-0.29
240000	-0.18	-0.66	-0.40	-0.45	-0.45	-0.47	-0.53	-0.39	-0.44	-0.45
260000	-0.35	-0.75	-0.41	-0.47	-0.43	-0.45	-0.50	-0.34	-0.37	-0.36
280000	-0.18	-0.60	-0.28	-0.33	-0.31	-0.35	-0.39	-0.25	-0.30	-0.26
300000	-0.19	-0.89	-0.53	-0.53	-0.55	-0.57	-0.61	-0.48	-0.53	-0.48
320000	-0.63	-0.96	-0.67	-0.64	-0.57	-0.68	-0.77	-0.64	-0.69	-0.65
340000	0.07	-0.73	-0.43	-0.34	-0.35	-0.46	-0.48	-0.30	-0.35	-0.32
360000	0.20	-0.51	-0.38	-0.30	-0.30	-0.37	-0.38	-0.25	-0.28	-0.26
380000	0.22	-0.67	-0.43	-0.33	-0.40	-0.51	-0.52	-0.37	-0.39	-0.38
400000	0.11	-0.60	-0.42	-0.31	-0.34	-0.39	-0.41	-0.28	-0.31	-0.32
420000	0.16	-0.49	-0.26	-0.16	-0.21	-0.25	-0.28	-0.15	-0.19	-0.20
440000	0.00	-0.59	-0.33	-0.22	-0.23	-0.26	-0.30	-0.16	-0.20	-0.21
460000	-0.11	-0.85	-0.57	-0.36	-0.32	-0.41	-0.48	-0.33	-0.41	-0.41
480000	0.12	-0.66	-0.47	-0.39	-0.39	-0.38	-0.43	-0.29	-0.36	-0.39
500000	-0.13	-0.70	-0.50	-0.37	-0.42	-0.49	-0.51	-0.36	-0.39	-0.36
520000	0.19	-0.40	-0.22	-0.08	-0.14	-0.17	-0.21	-0.10	-0.13	-0.11
540000	0.35	-0.34	-0.20	-0.09	-0.16	-0.20	-0.25	-0.15	-0.20	-0.16
560000	0.22	-0.44	-0.27	-0.14	-0.23	-0.28	-0.32	-0.23	-0.27	-0.24
580000	0.16	-0.48	-0.30	-0.13	-0.18	-0.21	-0.27	-0.17	-0.20	-0.17
600000	0.14	-0.44	-0.32	-0.22	-0.26	-0.28	-0.31	-0.21	-0.24	-0.24
620000	-0.06	-0.42	-0.32	-0.16	-0.19	-0.24	-0.25	-0.14	-0.16	-0.15
640000	0.15	-0.29	-0.17	-0.02	-0.08	-0.14	-0.16	-0.07	-0.07	-0.06
660000	-0.01	-0.45	-0.31	-0.17	-0.21	-0.29	-0.30	-0.21	-0.21	-0.20
680000	0.04	-0.37	-0.26	-0.13	-0.17	-0.24	-0.25	-0.16	-0.15	-0.13
700000	0.04	-0.34	-0.25	-0.08	-0.10	-0.17	-0.18	-0.10	-0.09	-0.09
720000	-0.12	-0.52	-0.46	-0.28	-0.28	-0.33	-0.33	-0.25	-0.25	-0.23
740000	-0.04	-0.39	-0.29	-0.18	-0.18	-0.27	-0.25	-0.18	-0.17	-0.17
760000	-0.02	-0.39	-0.31	-0.14	-0.16	-0.24	-0.23	-0.17	-0.17	-0.18
780000	-0.10	-0.41	-0.29	-0.14	-0.16	-0.22	-0.21	-0.14	-0.14	-0.16
800000	-0.10	-0.33	-0.22	-0.07	-0.07	-0.14	-0.14	-0.08	-0.10	-0.12
820000	-0.05	-0.28	-0.16	-0.01	-0.01	-0.09	-0.10	-0.04	-0.06	-0.06
840000	-0.03	-0.28	-0.22	-0.06	-0.05	-0.14	-0.15	-0.09	-0.12	-0.13
860000	-0.15	-0.35	-0.26	-0.12	-0.12	-0.21	-0.21	-0.17	-0.18	-0.19
880000	-0.01	-0.21	-0.13	0.00	-0.02	-0.11	-0.10	-0.04	-0.05	-0.05
900000	0.08	-0.16	-0.10	0.03	0.02	-0.07	-0.07	-0.01	-0.02	-0.02
920000	0.15	-0.10	-0.03	0.08	0.07	-0.02	-0.02	0.04	0.03	0.03
940000	0.23	-0.04	0.02	0.14	0.13	0.02	0.02	0.06	0.05	0.05
960000	0.13	-0.12	-0.07	0.03	0.02	-0.09	-0.09	-0.04	-0.06	-0.05
980000	0.03	-0.18	-0.13	-0.01	-0.01	-0.11	-0.11	-0.06	-0.08	-0.08
1000000	0.14	-0.11	-0.04	0.06	0.06	-0.03	-0.03	0.02	0.00	0.00

Table A5: Average actual overrun accuracy over all test configurations: 100-task sets, absolute values

Cycles	1	2	3	4	5	6	7	8	9	10
60000	0.7998	0.8156	0.8235	0.8295	0.8246	0.8240	0.8236	0.8239	0.8255	0.8271
80000	0.8697	0.8818	0.8805	0.8834	0.8820	0.8823	0.8811	0.8805	0.8819	0.8825
100000	0.9080	0.9108	0.9092	0.9115	0.9118	0.9129	0.9121	0.9111	0.9125	0.9118
120000	0.9252	0.9259	0.9279	0.9284	0.9280	0.9283	0.9280	0.9267	0.9272	0.9271
140000	0.9392	0.9396	0.9394	0.9390	0.9383	0.9380	0.9379	0.9373	0.9383	0.9385
160000	0.9430	0.9448	0.9447	0.9448	0.9452	0.9450	0.9453	0.9444	0.9451	0.9451
180000	0.9460	0.9499	0.9509	0.9518	0.9519	0.9516	0.9516	0.9510	0.9515	0.9514
200000	0.9490	0.9515	0.9530	0.9543	0.9544	0.9540	0.9542	0.9540	0.9542	0.9540
220000	0.9536	0.9563	0.9570	0.9578	0.9576	0.9571	0.9576	0.9575	0.9580	0.9576
240000	0.9592	0.9602	0.9611	0.9614	0.9613	0.9609	0.9611	0.9610	0.9612	0.9608
260000	0.9600	0.9624	0.9627	0.9631	0.9632	0.9628	0.9630	0.9627	0.9630	0.9627
280000	0.9618	0.9644	0.9657	0.9658	0.9659	0.9655	0.9656	0.9652	0.9655	0.9652
300000	0.9619	0.9649	0.9658	0.9661	0.9666	0.9663	0.9664	0.9662	0.9665	0.9663
320000	0.9630	0.9653	0.9661	0.9664	0.9668	0.9664	0.9661	0.9660	0.9663	0.9661
340000	0.9636	0.9663	0.9671	0.9672	0.9675	0.9675	0.9673	0.9674	0.9677	0.9674
360000	0.9654	0.9680	0.9685	0.9688	0.9690	0.9689	0.9686	0.9686	0.9687	0.9683
380000	0.9657	0.9682	0.9690	0.9689	0.9692	0.9688	0.9687	0.9687	0.9690	0.9687
400000	0.9653	0.9680	0.9690	0.9691	0.9694	0.9690	0.9688	0.9688	0.9691	0.9688
420000	0.9651	0.9691	0.9701	0.9703	0.9707	0.9704	0.9702	0.9701	0.9703	0.9700
440000	0.9668	0.9705	0.9710	0.9713	0.9715	0.9713	0.9710	0.9709	0.9711	0.9709
460000	0.9689	0.9718	0.9721	0.9724	0.9727	0.9724	0.9722	0.9722	0.9724	0.9721
480000	0.9683	0.9717	0.9722	0.9723	0.9724	0.9723	0.9720	0.9720	0.9721	0.9719
500000	0.9677	0.9712	0.9718	0.9721	0.9722	0.9722	0.9720	0.9720	0.9722	0.9720
520000	0.9684	0.9725	0.9725	0.9726	0.9727	0.9725	0.9723	0.9723	0.9723	0.9721
540000	0.9694	0.9734	0.9734	0.9734	0.9733	0.9731	0.9729	0.9731	0.9732	0.9729
560000	0.9701	0.9735	0.9734	0.9736	0.9734	0.9733	0.9730	0.9731	0.9733	0.9730
580000	0.9704	0.9738	0.9740	0.9741	0.9740	0.9738	0.9735	0.9736	0.9738	0.9735
600000	0.9718	0.9747	0.9748	0.9749	0.9748	0.9746	0.9744	0.9744	0.9746	0.9743
620000	0.9722	0.9747	0.9747	0.9747	0.9747	0.9745	0.9743	0.9744	0.9746	0.9743
640000	0.9721	0.9749	0.9749	0.9747	0.9746	0.9746	0.9743	0.9743	0.9745	0.9742
660000	0.9732	0.9759	0.9758	0.9754	0.9753	0.9751	0.9749	0.9749	0.9751	0.9748
680000	0.9735	0.9760	0.9759	0.9755	0.9754	0.9751	0.9749	0.9749	0.9750	0.9748
700000	0.9741	0.9764	0.9763	0.9761	0.9760	0.9758	0.9755	0.9755	0.9756	0.9754
720000	0.9745	0.9766	0.9766	0.9764	0.9764	0.9762	0.9759	0.9759	0.9760	0.9758
740000	0.9745	0.9767	0.9765	0.9765	0.9765	0.9763	0.9759	0.9759	0.9760	0.9758
760000	0.9747	0.9767	0.9766	0.9765	0.9764	0.9763	0.9760	0.9760	0.9760	0.9759
780000	0.9752	0.9774	0.9772	0.9771	0.9770	0.9768	0.9765	0.9765	0.9765	0.9763
800000	0.9751	0.9772	0.9772	0.9771	0.9768	0.9767	0.9763	0.9763	0.9763	0.9762
820000	0.9756	0.9778	0.9777	0.9777	0.9775	0.9773	0.9769	0.9769	0.9770	0.9768
840000	0.9761	0.9777	0.9775	0.9772	0.9772	0.9772	0.9767	0.9767	0.9768	0.9766
860000	0.9760	0.9776	0.9773	0.9772	0.9772	0.9772	0.9768	0.9769	0.9768	0.9766
880000	0.9757	0.9779	0.9775	0.9775	0.9776	0.9776	0.9772	0.9774	0.9773	0.9771
900000	0.9759	0.9781	0.9778	0.9778	0.9779	0.9780	0.9776	0.9777	0.9776	0.9774
920000	0.9760	0.9780	0.9777	0.9776	0.9777	0.9777	0.9773	0.9773	0.9773	0.9771
940000	0.9759	0.9779	0.9777	0.9776	0.9778	0.9779	0.9775	0.9775	0.9775	0.9773
960000	0.9763	0.9783	0.9781	0.9781	0.9783	0.9784	0.9779	0.9779	0.9779	0.9777
980000	0.9765	0.9786	0.9783	0.9783	0.9785	0.9786	0.9781	0.9781	0.9781	0.9779
1000000	0.9764	0.9785	0.9783	0.9784	0.9787	0.9788	0.9783	0.9783	0.9783	0.9781

Table A6: Average potential overrun accuracy over all test configurations: 100-task sets, absolute values

Cycles	1	2	3	4	5	6	7	8	9	10
60000	0.5580	0.5522	0.5534	0.5543	0.5539	0.5545	0.5548	0.5551	0.5547	0.5553
80000	0.5544	0.5519	0.5539	0.5534	0.5538	0.5547	0.5546	0.5549	0.5551	0.5554
100000	0.5549	0.5536	0.5550	0.5548	0.5544	0.5550	0.5551	0.5558	0.5556	0.5559
120000	0.5567	0.5547	0.5567	0.5564	0.5563	0.5569	0.5567	0.5568	0.5568	0.5569
140000	0.5560	0.5569	0.5584	0.5579	0.5577	0.5580	0.5576	0.5576	0.5577	0.5580
160000	0.5561	0.5569	0.5582	0.5576	0.5580	0.5583	0.5581	0.5583	0.5585	0.5587
180000	0.5562	0.5566	0.5578	0.5578	0.5581	0.5583	0.5581	0.5584	0.5585	0.5585
200000	0.5565	0.5566	0.5580	0.5579	0.5583	0.5586	0.5585	0.5588	0.5590	0.5591
220000	0.5574	0.5573	0.5588	0.5587	0.5591	0.5592	0.5590	0.5593	0.5594	0.5594
240000	0.5574	0.5572	0.5585	0.5585	0.5590	0.5591	0.5591	0.5593	0.5595	0.5596
260000	0.5571	0.5573	0.5587	0.5588	0.5591	0.5592	0.5592	0.5593	0.5595	0.5598
280000	0.5574	0.5582	0.5593	0.5594	0.5596	0.5597	0.5596	0.5597	0.5599	0.5601
300000	0.5569	0.5578	0.5590	0.5593	0.5596	0.5596	0.5596	0.5596	0.5598	0.5600
320000	0.5573	0.5580	0.5588	0.5590	0.5592	0.5593	0.5592	0.5593	0.5595	0.5596
340000	0.5578	0.5579	0.5588	0.5590	0.5591	0.5592	0.5592	0.5592	0.5594	0.5596
360000	0.5588	0.5587	0.5594	0.5595	0.5596	0.5596	0.5596	0.5596	0.5597	0.5598
380000	0.5587	0.5589	0.5596	0.5597	0.5598	0.5598	0.5597	0.5597	0.5599	0.5600
400000	0.5585	0.5589	0.5596	0.5597	0.5599	0.5598	0.5598	0.5598	0.5599	0.5600
420000	0.5582	0.5588	0.5594	0.5596	0.5597	0.5597	0.5596	0.5596	0.5598	0.5599
440000	0.5582	0.5589	0.5594	0.5596	0.5597	0.5597	0.5595	0.5596	0.5597	0.5599
460000	0.5583	0.5588	0.5594	0.5597	0.5597	0.5597	0.5596	0.5596	0.5598	0.5599
480000	0.5587	0.5592	0.5596	0.5598	0.5598	0.5598	0.5598	0.5597	0.5598	0.5599
500000	0.5583	0.5589	0.5594	0.5596	0.5596	0.5597	0.5596	0.5596	0.5597	0.5598
520000	0.5587	0.5590	0.5594	0.5596	0.5597	0.5597	0.5596	0.5596	0.5597	0.5598
540000	0.5589	0.5591	0.5595	0.5596	0.5597	0.5597	0.5595	0.5595	0.5596	0.5597
560000	0.5590	0.5592	0.5597	0.5598	0.5599	0.5598	0.5597	0.5596	0.5597	0.5598
580000	0.5588	0.5592	0.5596	0.5597	0.5598	0.5597	0.5596	0.5595	0.5597	0.5597
600000	0.5587	0.5591	0.5594	0.5595	0.5596	0.5596	0.5594	0.5594	0.5595	0.5596
620000	0.5587	0.5591	0.5595	0.5596	0.5596	0.5595	0.5594	0.5594	0.5595	0.5596
640000	0.5587	0.5591	0.5595	0.5596	0.5595	0.5596	0.5594	0.5594	0.5595	0.5595
660000	0.5588	0.5592	0.5595	0.5595	0.5595	0.5595	0.5594	0.5594	0.5595	0.5595
680000	0.5587	0.5592	0.5594	0.5594	0.5594	0.5594	0.5593	0.5593	0.5594	0.5595
700000	0.5587	0.5591	0.5593	0.5594	0.5594	0.5594	0.5593	0.5593	0.5593	0.5594
720000	0.5586	0.5590	0.5593	0.5593	0.5593	0.5593	0.5592	0.5592	0.5592	0.5593
740000	0.5588	0.5591	0.5593	0.5593	0.5593	0.5594	0.5592	0.5592	0.5593	0.5593
760000	0.5586	0.5590	0.5592	0.5592	0.5592	0.5593	0.5592	0.5591	0.5592	0.5593
780000	0.5585	0.5589	0.5591	0.5591	0.5592	0.5592	0.5591	0.5591	0.5591	0.5592
800000	0.5586	0.5589	0.5591	0.5591	0.5591	0.5592	0.5591	0.5591	0.5591	0.5592
820000	0.5582	0.5586	0.5589	0.5589	0.5590	0.5591	0.5590	0.5590	0.5590	0.5591
840000	0.5581	0.5586	0.5589	0.5589	0.5590	0.5591	0.5589	0.5589	0.5590	0.5591
860000	0.5582	0.5586	0.5589	0.5589	0.5589	0.5590	0.5589	0.5589	0.5590	0.5590
880000	0.5581	0.5586	0.5588	0.5588	0.5589	0.5589	0.5589	0.5588	0.5589	0.5589
900000	0.5581	0.5584	0.5587	0.5587	0.5588	0.5589	0.5588	0.5588	0.5588	0.5589
920000	0.5582	0.5584	0.5587	0.5587	0.5588	0.5589	0.5588	0.5588	0.5588	0.5589
940000	0.5581	0.5584	0.5587	0.5587	0.5588	0.5589	0.5588	0.5588	0.5588	0.5589
960000	0.5581	0.5583	0.5586	0.5587	0.5588	0.5588	0.5588	0.5587	0.5588	0.5588
980000	0.5581	0.5583	0.5586	0.5586	0.5587	0.5588	0.5587	0.5587	0.5587	0.5588
1000000	0.5581	0.5583	0.5586	0.5586	0.5587	0.5587	0.5587	0.5587	0.5587	0.5587

Table A7: Average actual overrun accuracy over all test configurations: 100-task sets, percentage variation from 10 runs/1,000,000 cycles

Cycles	1	2	3	4	5	6	7	8	9	10
60000	-17.8	-16.3	-15.5	-14.9	-15.4	-15.4	-15.5	-15.4	-15.3	-15.1
80000	-10.8	-9.6	-9.8	-9.5	-9.6	-9.6	-9.7	-9.8	-9.6	-9.6
100000	-7.0	-6.7	-6.9	-6.7	-6.6	-6.5	-6.6	-6.7	-6.6	-6.6
120000	-5.3	-5.2	-5.0	-5.0	-5.0	-5.0	-5.0	-5.1	-5.1	-5.1
140000	-3.9	-3.9	-3.9	-3.9	-4.0	-4.0	-4.0	-4.1	-4.0	-4.0
160000	-3.5	-3.3	-3.3	-3.3	-3.3	-3.3	-3.3	-3.4	-3.3	-3.3
180000	-3.2	-2.8	-2.7	-2.6	-2.6	-2.7	-2.7	-2.7	-2.7	-2.7
200000	-2.9	-2.7	-2.5	-2.4	-2.4	-2.4	-2.4	-2.4	-2.4	-2.4
220000	-2.5	-2.2	-2.1	-2.0	-2.1	-2.1	-2.1	-2.1	-2.0	-2.1
240000	-1.9	-1.8	-1.7	-1.7	-1.7	-1.7	-1.7	-1.7	-1.7	-1.7
260000	-1.8	-1.6	-1.5	-1.5	-1.5	-1.5	-1.5	-1.5	-1.5	-1.5
280000	-1.6	-1.4	-1.2	-1.2	-1.2	-1.3	-1.3	-1.3	-1.3	-1.3
300000	-1.6	-1.3	-1.2	-1.2	-1.2	-1.2	-1.2	-1.2	-1.2	-1.2
320000	-1.5	-1.3	-1.2	-1.2	-1.1	-1.2	-1.2	-1.2	-1.2	-1.2
340000	-1.5	-1.2	-1.1	-1.1	-1.1	-1.1	-1.1	-1.1	-1.0	-1.1
360000	-1.3	-1.0	-1.0	-0.9	-0.9	-0.9	-1.0	-1.0	-0.9	-1.0
380000	-1.2	-1.0	-0.9	-0.9	-0.9	-0.9	-0.9	-0.9	-0.9	-0.9
400000	-1.3	-1.0	-0.9	-0.9	-0.9	-0.9	-0.9	-0.9	-0.9	-0.9
420000	-1.3	-0.9	-0.8	-0.8	-0.7	-0.8	-0.8	-0.8	-0.8	-0.8
440000	-1.1	-0.8	-0.7	-0.7	-0.7	-0.7	-0.7	-0.7	-0.7	-0.7
460000	-0.9	-0.6	-0.6	-0.6	-0.5	-0.6	-0.6	-0.6	-0.6	-0.6
480000	-1.0	-0.6	-0.6	-0.6	-0.6	-0.6	-0.6	-0.6	-0.6	-0.6
500000	-1.0	-0.7	-0.6	-0.6	-0.6	-0.6	-0.6	-0.6	-0.6	-0.6
520000	-1.0	-0.6	-0.6	-0.6	-0.5	-0.6	-0.6	-0.6	-0.6	-0.6
540000	-0.9	-0.5	-0.5	-0.5	-0.5	-0.5	-0.5	-0.5	-0.5	-0.5
560000	-0.8	-0.5	-0.5	-0.5	-0.5	-0.5	-0.5	-0.5	-0.5	-0.5
580000	-0.8	-0.4	-0.4	-0.4	-0.4	-0.4	-0.5	-0.5	-0.4	-0.5
600000	-0.6	-0.3	-0.3	-0.3	-0.3	-0.4	-0.4	-0.4	-0.4	-0.4
620000	-0.6	-0.3	-0.3	-0.3	-0.3	-0.4	-0.4	-0.4	-0.4	-0.4
640000	-0.6	-0.3	-0.3	-0.3	-0.3	-0.4	-0.4	-0.4	-0.4	-0.4
660000	-0.5	-0.2	-0.2	-0.3	-0.3	-0.3	-0.3	-0.3	-0.3	-0.3
680000	-0.5	-0.2	-0.2	-0.3	-0.3	-0.3	-0.3	-0.3	-0.3	-0.3
700000	-0.4	-0.2	-0.2	-0.2	-0.2	-0.2	-0.3	-0.3	-0.3	-0.3
720000	-0.4	-0.1	-0.2	-0.2	-0.2	-0.2	-0.2	-0.2	-0.2	-0.2
740000	-0.4	-0.1	-0.2	-0.2	-0.2	-0.2	-0.2	-0.2	-0.2	-0.2
760000	-0.3	-0.1	-0.2	-0.2	-0.2	-0.2	-0.2	-0.2	-0.2	-0.2
780000	-0.3	-0.1	-0.1	-0.1	-0.1	-0.1	-0.2	-0.2	-0.2	-0.2
800000	-0.3	-0.1	-0.1	-0.1	-0.1	-0.1	-0.2	-0.2	-0.2	-0.2
820000	-0.2	0.0	0.0	0.0	-0.1	-0.1	-0.1	-0.1	-0.1	-0.1
840000	-0.2	0.0	-0.1	-0.1	-0.1	-0.1	-0.1	-0.1	-0.1	-0.2
860000	-0.2	-0.1	-0.1	-0.1	-0.1	-0.1	-0.1	-0.1	-0.1	-0.1
880000	-0.2	0.0	-0.1	-0.1	-0.1	0.0	-0.1	-0.1	-0.1	-0.1
900000	-0.2	0.0	0.0	0.0	0.0	0.0	-0.1	0.0	-0.1	-0.1
920000	-0.2	0.0	0.0	-0.1	0.0	0.0	-0.1	-0.1	-0.1	-0.1
940000	-0.2	0.0	0.0	-0.1	0.0	0.0	-0.1	-0.1	-0.1	-0.1
960000	-0.2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
980000	-0.2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1000000	-0.2	0.0	0.0	0.0	0.1	0.1	0.0	0.0	0.0	0.0

Table A8: Average potential overrun accuracy over all test configurations: 100-task sets, percentage variation from 10 runs/1,000,000 cycles

Cycles	1	2	3	4	5	6	7	8	9	10
60000	-0.07	-0.66	-0.54	-0.45	-0.48	-0.42	-0.39	-0.37	-0.40	-0.34
80000	-0.43	-0.68	-0.48	-0.53	-0.49	-0.40	-0.41	-0.38	-0.36	-0.33
100000	-0.39	-0.51	-0.38	-0.40	-0.44	-0.37	-0.36	-0.29	-0.31	-0.29
120000	-0.21	-0.41	-0.21	-0.24	-0.24	-0.18	-0.20	-0.19	-0.19	-0.18
140000	-0.27	-0.18	-0.03	-0.08	-0.10	-0.08	-0.11	-0.11	-0.10	-0.07
160000	-0.27	-0.19	-0.06	-0.12	-0.07	-0.04	-0.06	-0.04	-0.02	0.00
180000	-0.25	-0.21	-0.09	-0.10	-0.07	-0.04	-0.06	-0.03	-0.02	-0.02
200000	-0.23	-0.21	-0.08	-0.09	-0.04	-0.01	-0.02	0.01	0.03	0.04
220000	-0.14	-0.15	0.01	0.00	0.04	0.04	0.03	0.05	0.07	0.07
240000	-0.14	-0.16	-0.02	-0.02	0.02	0.03	0.04	0.06	0.07	0.08
260000	-0.16	-0.14	0.00	0.01	0.04	0.04	0.05	0.06	0.08	0.10
280000	-0.13	-0.05	0.06	0.07	0.09	0.09	0.08	0.10	0.11	0.13
300000	-0.18	-0.09	0.03	0.06	0.09	0.09	0.08	0.09	0.11	0.13
320000	-0.14	-0.08	0.01	0.03	0.05	0.06	0.05	0.05	0.08	0.09
340000	-0.09	-0.08	0.01	0.03	0.04	0.05	0.04	0.05	0.07	0.08
360000	0.00	0.00	0.07	0.08	0.09	0.09	0.08	0.09	0.10	0.11
380000	0.00	0.02	0.09	0.10	0.10	0.11	0.09	0.10	0.11	0.13
400000	-0.02	0.02	0.09	0.10	0.11	0.11	0.10	0.10	0.12	0.13
420000	-0.05	0.01	0.06	0.09	0.10	0.10	0.09	0.09	0.11	0.12
440000	-0.06	0.01	0.07	0.09	0.09	0.09	0.08	0.09	0.10	0.11
460000	-0.04	0.01	0.07	0.10	0.10	0.10	0.09	0.09	0.10	0.11
480000	0.00	0.04	0.08	0.10	0.11	0.11	0.10	0.10	0.11	0.12
500000	-0.04	0.02	0.06	0.09	0.09	0.10	0.09	0.09	0.10	0.11
520000	0.00	0.03	0.07	0.09	0.10	0.10	0.09	0.09	0.09	0.10
540000	0.02	0.04	0.08	0.09	0.10	0.10	0.08	0.08	0.09	0.09
560000	0.02	0.05	0.09	0.10	0.11	0.11	0.09	0.09	0.10	0.11
580000	0.01	0.05	0.09	0.10	0.10	0.10	0.09	0.08	0.09	0.10
600000	0.00	0.03	0.07	0.08	0.08	0.08	0.07	0.07	0.08	0.08
620000	-0.01	0.04	0.08	0.08	0.08	0.08	0.07	0.06	0.07	0.08
640000	0.00	0.04	0.08	0.08	0.08	0.08	0.07	0.07	0.07	0.08
660000	0.00	0.05	0.08	0.08	0.08	0.08	0.07	0.07	0.07	0.08
680000	0.00	0.04	0.07	0.07	0.07	0.07	0.06	0.06	0.07	0.07
700000	0.00	0.04	0.06	0.06	0.06	0.07	0.06	0.05	0.06	0.07
720000	-0.01	0.03	0.05	0.05	0.06	0.06	0.05	0.05	0.05	0.06
740000	0.00	0.03	0.05	0.05	0.06	0.06	0.05	0.05	0.05	0.06
760000	-0.01	0.02	0.04	0.04	0.05	0.06	0.04	0.04	0.05	0.06
780000	-0.03	0.02	0.04	0.04	0.04	0.05	0.04	0.03	0.04	0.05
800000	-0.01	0.02	0.04	0.04	0.04	0.05	0.04	0.04	0.04	0.05
820000	-0.05	-0.01	0.01	0.02	0.03	0.03	0.02	0.02	0.03	0.04
840000	-0.06	-0.01	0.02	0.02	0.02	0.03	0.02	0.02	0.02	0.03
860000	-0.06	-0.01	0.01	0.01	0.02	0.03	0.02	0.02	0.02	0.03
880000	-0.06	-0.02	0.01	0.01	0.01	0.02	0.01	0.01	0.01	0.02
900000	-0.06	-0.03	0.00	0.00	0.01	0.01	0.01	0.00	0.01	0.01
920000	-0.05	-0.03	0.00	0.00	0.01	0.01	0.01	0.00	0.01	0.01
940000	-0.06	-0.04	0.00	0.00	0.01	0.02	0.01	0.01	0.01	0.01
960000	-0.06	-0.04	-0.01	-0.01	0.00	0.01	0.00	0.00	0.00	0.01
980000	-0.07	-0.05	-0.01	-0.01	0.00	0.00	0.00	0.00	0.00	0.00
1000000	-0.07	-0.04	-0.01	-0.01	0.00	0.00	-0.01	-0.01	0.00	0.00