Project Reference Number	TP/3/PIT/6/I/15927
Project Title	Mobile Network Optimisation Through Advanced Simulation
Project Acronym	MONOTAS
Project Partners	Multiple Access Communications Limited
	Vodafone Group Services Limited
	Nortel Networks UK Limited

Report Title	Survey of Optimisation Techniques
Report Reference Number	020.PUB.2
Report Date	8 March 2006
Editor(s)	Peter Myerscough-Jackopson (MAC Ltd)
Author(s)	Peter Myerscough-Jackopson (MAC Ltd)
Reviewer(s)	Peter Gould (MAC Ltd), Damian Bevan (Nortel)

Abstract	This report reviews the areas of optimisation and control theory and examines the different techniques available and their potential for use in the optimisation of mobile radio networks.
Keyword(s)	Network Optimisation, Optimisation, Control, Adaptive Control.
Confidentiality	Public

Document History

Date	Version	Comment	Editor
15 February 2006	1.0	Issued for internal review.	Peter Myerscough, MAC Ltd
8 March 2006	1.1	Updated due to comments from Damian Bevan (Nortel) and Stefan Thiel (Vodafone).	Peter Myerscough-Jackopson, MAC Ltd

Executive Summary

Project Monotas is a collaborative research project, part funded by the UK Department of Trade and Industry, to examine the use of advanced simulation and other computer models in the control and optimisation of large-scale mobile radio networks, and in particular 3G networks. In this report we review the areas of optimisation and control theory as potential 'tools' to be used in Project Monotas.

Optimisation is concerned with the selection of optimal settings for a system, given a system function that can be used to evaluate the performance of a particular setting. Control theory deals with modelling the relationship between a system and its controller to produce the best system behaviour.

The optimisation techniques presented in this report cover the areas of continuous and discrete valued problems, consider the effects of stochastic environments and take their inspiration from biology, physics, neurology, statistics and other areas. The literature within optimisation considers the different approaches and methods and acknowledges that no single technique will be able to perform perfectly on all problems. Further to this it is argued that each technique will, on average, perform the same on the range of all problems, and so it is important to consider how the technique aligns itself to known information about the problem being considered.

In the control theory section the ideas covered include open- and closed-loop control as well as non-adaptive and adaptive control. Further to this, the adaptive control section considers the issues of exploring new settings verses using old settings that have proven capabilities for enhancing the system. Adaptive control also enables strategies that are developed within a simulation environment, once deployed, to develop further in the real system environment. This may be a useful feature of adaptive control that may show not only improvements in the final performance of the network, but if analysed after installation may reveal information about the network's true behaviour.

Our concluding remarks consider that the selection of an algorithm from those reviewed will involve evaluating each technique against each problem Project Monotas encounters. It is also noted that limits may need to be placed upon the operation of any of the techniques to prevent failure or system degradation that may be seen as unacceptable. Finally, measures for the evaluation of a technique should also be decided upon so that the integration of any developed technique into a system can be monitored.

This report should therefore help enable Project Monotas to explore algorithms that will provide enhanced network performance that cannot currently be achieved.

Table of Contents

List of Abbreviations

3G	Third Generation (Mobile Network)
ANN	Artificial Neural Network
MDP	Markov Decision Process
MONOTAS	Mobile Network Optimisation Through Advanced Simulation
NEAT	NeuroEvolution of Augmenting Topologies
PID	Proportional-Integral-Differential, a type of controller

1. Introduction

The aim of Project Monotas is to develop new schemes for optimising mobile networks, with a particular focus on using advanced simulation methods and computing techniques. The partners within the consortium have a current interest in the newly deployed and developing 3G networks and so, although techniques and ideas developed during the course of the project may be applicable to a range of different technologies, they will be developed and evaluated with particular reference to 3G networks.

The project is currently in the early stages of discovery and problem definition, and the literature review presented in this document represents a part of this discovery process. Other parallel activities include examining the current processes that are used in network optimisation, understanding the cost of modelling different aspects of the network accurately and considering which aspects of the 3G network may benefit most from optimisation. This means that although we have considered a range of different optimisation strategies and methods in this report, the exact problem to which they may be successfully applied is as yet undefined. Therefore, we are not able to perform a detailed evaluation of the different options with respect to our requirements at this stage, and so this process will be performed later in the project. Our aim has been to examine the potential 'tool kit' of algorithms and approaches that could be used within Project Monotas, without descending into too much detail relating to the manner in which the techniques may actually be used at this stage. In Section 2 we consider different ways in which the techniques examined in this report could be used in the optimisation of a mobile network. This is followed in Section 3 by a discussion of the various optimisation techniques available. We examine the potential advantages and limitations of each technique and consider how they may be used within Project Monotas. In Section 4 we consider different control theory ideas, focusing first on non-adaptive control then moving on to consider adaptive control. Control techniques have the potential to facilitate a more effective approach to real-time network optimisation than those ideas presented in Section 3. In Section 4 we also consider different approaches that can be used to 'train' or adapt controllers to match a particular problem. Finally, in Section 5 we present our conclusions.

2. Network Optimisation Strategies

In this section we examine five possible strategies for optimising a mobile network. Some of them utilise simulation in the final operation of the strategy, others may need advanced simulation to develop and set their parameters (but may not involve simulation within the ultimate solution) and yet others are techniques that operate as approximations to the real network.

To apply network optimisation, various network parameters must be made available for adjustment. Conversely, to detect changes in the network in response to these alterations, network statistics or real-time metrics must also be exposed. This view of the network is presented in Figure 1.

Figure 1 also suggests that a simulator could replace the real network. This will have to be the case during the course of Project Monotas so that different techniques and ideas can be applied to the simulated network without costly results should they fail. The concept of “Optimisation Algorithms” can be expanded upon, although it could simply refer to any of the techniques presented later in the document. Alternative structures for approaching network optimisation are also going to be considered and some current candidates are shown in Figures 2 to 5.

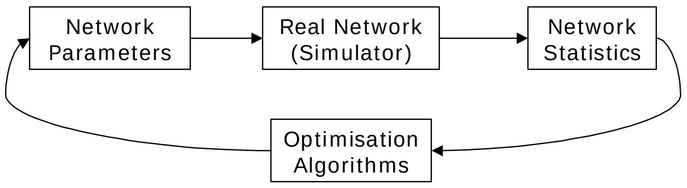


Figure 1 A simple diagram describing the general approach to optimising a real or simulated network.

Figure 2 shows that a network simulator could be used in a final solution to enable network parameters to be tested before application to the real network, ie, to try out ‘what if’ scenarios. Currently this is a common approach in automated network planning, but the links that enable the change in network parameters are human and the parameters are usually physical quantities that it may only be possible to adjust manually, eg, site placement, antenna downtilt. This project is aiming to consider network parameters that can be altered rapidly (ie, without manual intervention), since this would facilitate closed-loop network optimisation. The problems with the approach of Figure 2 are associated with simulating the network at a sufficient speed and accuracy to predict network statistics in sufficient quantity and quality. This in turn limits the ability of an optimisation algorithm to find a better set of network parameters in a reasonable timescale. It may therefore be necessary to produce multiple network simulators with differing degrees of speed and accuracy to solve different problems and support different optimisation algorithms. Increasing the speed at which the network parameters can be tuned with respect to changes in the network statistics may have to be approached in a fundamentally different manner. Figure 3 is a further proposal that uses a real-time controller with an optimisation algorithm able to change the manner in which the controller operates as the network changes.

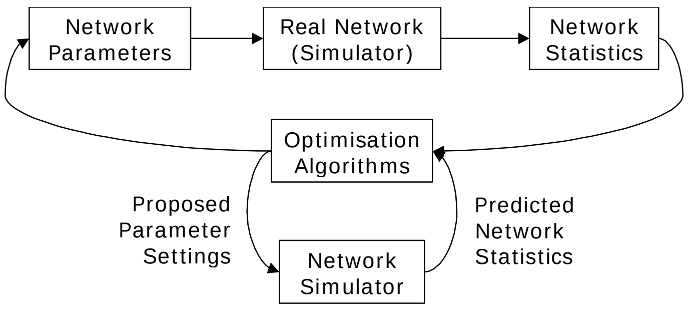


Figure 2 The use of a network simulator to enable proposed settings to be verified to test for improvements and validity.

In Figure 3, the use of a real-time controller will enable the network parameters to be altered in rapid response to changes in network statistics. An off-line optimisation algorithm can also be used to tune the controller to better match network behaviour.

The use of a real-time controller also brings in the field of control theory, which may allow estimates of the relationship between the network and the controller, particularly with regards to stability. This may mean that any technique developed using this paradigm can provide reassurances about the chances of failure, but they will be dependent upon the assumptions made about the network.

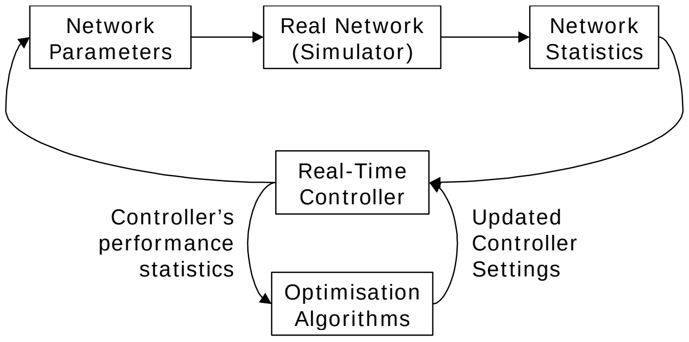


Figure 3 Network optimisation can be managed rapidly with a real-time controller, whilst the controller's performance can be monitored and controlled by an offline optimisation algorithm.

A further alternative to the arrangement shown in Figure 3 is the use of an adaptive controller, as presented in Figure 4. An adaptive controller would allow on-line optimisation to occur whilst the network is in operation. This type of controller can start with poor results as the internal parameters are often randomly initialised and it can be 'trained' to control the system via some feedback mechanism. This will mean that such an adaptive real-time controller will need to first be trained on a network simulator. The advantages of this technique could involve the ability of the controller to continue to adapt once it has been transferred to the real network and to do so continuously. This could provide a greater level of performance as the network controller and parameters are always being updated. The amount of network data that can directly influence this type of controller could also be orders of magnitude greater than any other technique. This will be because the controller can be adapted to network statistics immediately and so there is no requirement for storage or transmitting all the data across the network.

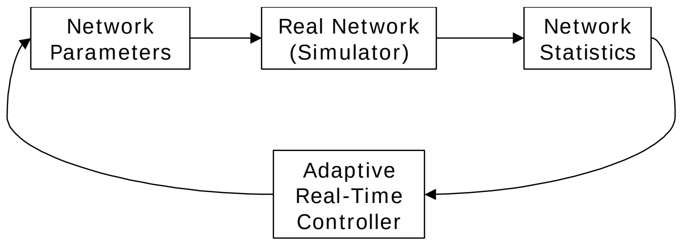


Figure 4 An adaptive real-time controller could be used to rapidly apply changes to the network, but also to rapidly 'learn' and adapt to the network whilst on-line.

The final method presented in this section is shown in Figure 5 where a model of the system is constructed that can be both adapted to the network's current state and can also be used to either predict future states of the network given certain input parameters or 'solved' to show the optimal parameter settings given the current status of the system. The latter approach could use mathematical techniques such as sets of simultaneous equations to solve the model's behaviour, if the model is constructed in an appropriate manner. Simplifications of the system will reduce its ability to predict far into the future, but may be reliable enough to be useful if the model is adapted as the system changes. This adaptation allows the model to be much simpler than the system and maintain a good estimate of the system's current behaviour.

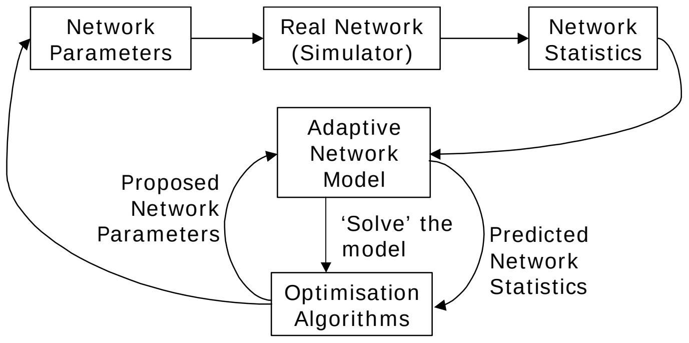


Figure 5 Adaptive Model-based optimisation algorithms rely on constructing a model of the network that can allow predictive behaviour to be evaluated or can be solved to give the best current action.

These five possible methods for optimising the network are a useful start to considering the most appropriate approaches to different network problems.

They do not consider the differences in scale that will exist between problems or between local or network wide optimisation strategies. Optimising the whole network in a single operation allows for the effects of each change in the network to be resolved and balanced before application. Alternatively, applying the optimisation process at a local level where only a subset of the network is examined simplifies the problem and allows the optimisation task to be effectively distributed throughout the network. This division and simplification may come at a cost to overall system stability as optimisations applied within neighbouring locales could cause destabilising behaviour of some system attributes, or degradation as race conditions or deadlock occurs due to controllers operating with insufficient information or authority.

Considerations also need to be given to the type of changes the network undergoes. This is particularly important when considering that a change in conditions could change the optimal settings for the network. Therefore any optimisation technique chosen must be able to cope with the different changes in the network, both gradual and stepwise in manner. Change may also be found in the transfer of any technique from a theoretical form, such as simulation, to the real network. The management of this change in the system that the technique is attempting to optimise is also an important consideration especially nearer the end of this project.

Final considerations must also be given to the rates at which the network may generate meaningful statistics and possible limitations on rates of change that network parameters may undergo. Both of these will affect how rapidly the network may be adapted. Slow generation of meaningful statistics may limit the changes applied to the network's parameters to prevent instability. It is also highly likely that statistics or metrics that the system outputs will be generated at different rates, possibly allowing some parameters to undergo more rapid adaptation than others, or for more minor variations to be applied rapidly and larger variations, which depend on more slowly generated statistics, being applied less rapidly.

3. Optimisation Techniques

Optimisation in the context of this report is the selection of input parameters for a system that, when applied to the system, gives the best response or output from the system over some period of time. In the case of this report the expected system is some as yet undefined aspect of the 3G mobile phone network, with the best response also an undefined quantity. To discuss the various approaches further we introduce some nomenclature, as follows.

*k*th set of parameters	X_k
the set of parameters that follow the *k*th set	X_{k+1}
the set of parameters that precede the *k*th set	X_{k-1}
the system function	$f(X_k)$
the gradient of *f()*	$\nabla f(X_k)$
an approximation to *f()*	$\hat{f}(X_k)$

Each optimisation method is expected to produce parameter settings, $X_{k,}$ by some process such that the system function, f(X_k), is set to an optimum, either a maximum or minimum dependent upon how the function is formulated. This system function may be a direct measurement of the part of the system that is undergoing optimisation or it may be a statistic or indirectly related measure. It may also be expected to come from the real network, if the optimisation process is applying parameter changes directly, or from a model of some description.

Commonly within the optimisation literature the concept of a problem's surface is used to relate parameter settings to the system function, f(X_k). This idea is shown in some of the figures used to illustrate the progression of certain algorithms later in this section. The surface in these plots describes the system function, but it is important to realise that any algorithm attempting to find the minimum or maximum of the function does not have the full knowledge of the system function's surface. An example is given of two surfaces in Figure 6. The five crosses signify where the function has been sampled, and are in the same positions for Figure 6(a) and Figure 6(b), but the underlying surfaces are different. An optimisation process that is just measuring the surface height at the five positions shown cannot detect this difference. It is only by exhaustively measuring the surface at each point in a function that the true surface structure can be found. However, this can be a time-consuming process, and in many situations it can be impractical due to the number of parameters (ie, the number of dimensions associated with the surface) and the variation possible in those parameters. We therefore consider optimisation algorithms that do not rely on exhaustively evaluating the system function.

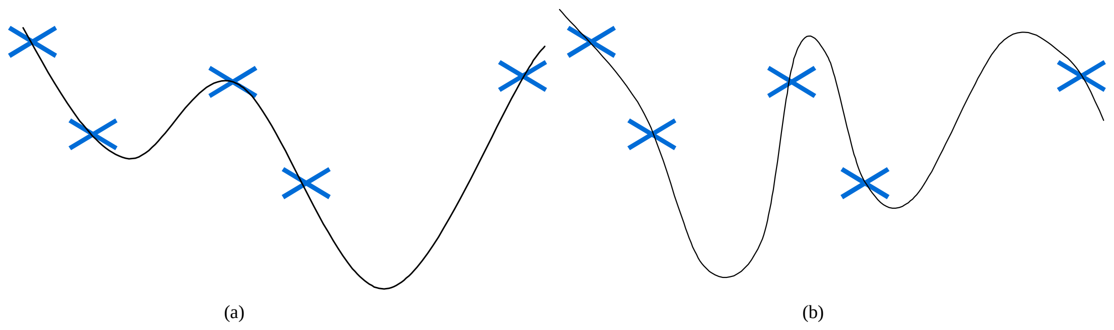


Figure 6 An example of two system function surfaces, five samples (crosses) from each function give matching results with different true surface shapes.

The selection of an optimisation algorithm is a difficult task, and in some cases it is simpler to state that a technique would not be useful because it cannot deal with known constraints. Alternatively some mathematical techniques may be designed to optimise problems that can be approximated by quadratic equations, but it may be that a particular system is more complex. There also exist optimisation techniques such as genetic programming whose full behaviour is not understood by the research community even now, and yet have been successfully applied to a wide range of problems. This can mean that selecting an optimisation technique suitable for solving a problem may be a ‘hit and miss’ process. Furthermore, a paper by Wolpert and Macready [1] suggests that even if an optimisation technique is shown to produce ‘good’ results up to a certain point, there is no guarantee that the same technique will be able to continue to optimise further based on its past performance. This paper also argues that over the set of all problems, each optimisation technique will perform on average the same as any other, but it is likely that there will be variation in performance on an individual problem basis. They also consider that problems where the performance for a particular optimisation technique is ‘good’ often reflect an alignment in structure between the problem and the optimisation technique. They therefore advise that any optimisation technique will benefit from including as much domain knowledge as possible to tailor the technique based on this knowledge.

Other considerations are the realistic constraints of time and resources when applying an optimisation technique to a problem. Techniques may or may not provide guarantees to reach the global optimum and even if a technique is guaranteed to find the global optimum, it may not be possible for it to achieve this within practical timescales. A more desirable property may be the arrival at a ‘good’ local optimum that satisfies some measure of ‘closeness’ to the global optimum within a reasonable timeframe.

In the remainder of this section we consider a wide range of optimisation techniques that have different limitations and may hopefully also align to different problems uncovered during Project Monotas.

3.1. Random Search

Randomly searching the problem domain for an optimal solution to a problem is one of the simplest methods used for optimising a set of parameters. It can be treated as a baseline algorithm. In circumstances where there is insufficient knowledge of the problem domain to infer any sort of structure it may perform comparably or better than any other technique that places constraints upon how new parameters are selected to be tested. The random element of the search can be altered if the distributions from which the random parameters are selected from are not uniform. These distributions can be altered to contain domain specific knowledge. Any further alterations lead to a random search becoming more similar to the optimisation techniques presented in the following sections.

3.2. Relaxation Methods or Optimisation Frameworks

Optimisation typically considers problems that are hard to solve; they may be formalised as Non-deterministic Polynomial time (NP)-hard or NP-complete problems, or they may be associated with a simulation or stochastic process. In considering these problems, methods for changing the problem to a simplified and more solvable one have been proposed. This process is called relaxation, and can involve removing some of the constraints to input variables or changing some of the assumptions a simulation makes. This should enable quicker computation and the following ideas are approaches published in the optimisation literature that consider how to relax a problem to solve it more effectively.

3.2.1. Sample Path

Sample path optimisation [2, 3, 4, 5] is designed to provide a framework for selecting optimal parameters when using a simulation that behaves stochastically. The technique relies on being able to re-run a stochastic simulation with the same random effects occurring at the same time. This then allows a set of parameter settings to be developed that are optimal for a single set of random events within the simulator. This process is then repeated, tuning the same parameter settings to a new set of random effects. The chosen settings should become more general as a greater number of sets of random effects are used to 'train' or select them. The combination of optimal settings for specific sets of random events and generalisation because of the number of sets used, should give good or optimal performance for new sets of random events.

3.2.2. Ordinal Optimisation

Ordinal optimisation [6, 7] can be viewed as an approach to optimisation as well as a separate technique. Ordinal optimisation reconsiders the question of searching for an optimal solution given a finite time and more realistically attempts to find a 'good' solution. This more realistic approach suggests a framework for optimising problems that are difficult to optimise given constraints that do not allow an algorithm to iterate through many different parameters. The ordinal optimisation [8, 9] approach relies on sampling theory that states that the probability of selecting a 'good' solution, ie, a solution in the top x%, is unaffected by the size of the problem domain the parameters are taken from, but is related to the number of different parameters, viewed as samples, taken from the problem domain. This will mean that given a target percentile, eg, top 5%, it is possible to estimate the likelihood of one of the selected parameter sets being a member of the top 5%. The added concept of optimising the parameters in an ordinal manner also means that instead of calculating the degree of improvement a new set of parameters makes (ie, how much better is X_{k+1} than X_k?), the question is simply: is X_{k+1} better than X_k? This change in approach should allow for calculations and simulations to take a less detailed or costly approach and also cause ordinal optimisation to converge on a 'good' solution more rapidly than cardinal optimisation techniques [10,11]. This allows for an initially large number of parameters options to be chosen and then all tested using a crude simulation to rank each parameter set. This ranking then defines which parameter sets to apply to a more detailed simulator. This process of applying an approximation to the system function to rank different parameter sets can be repeated, enabling a large number of different parameters to be investigated, and yet not wasting undue processing time on evaluating poorly performing parameter sets. Additional work on ordinal optimisation considers ordinal optimisation with constraints [12] and vector ordinal optimisation [13].

3.2.3. Perturbation Analysis

Gradient estimation within a multidimensional problem space requires many evaluations of the system function if the gradient estimate is generated from finite differences. This process is very dependent upon the cost of evaluating the system function, and also upon the number of dimensions in the problem space. Techniques have therefore been developed that attempt to calculate the gradient via other means, and perturbation analysis is one such method [14,15]. To apply perturbation analysis it must be possible to alter the system function, eg, if the system function is a simulator the source code must be available. The algorithm requires that the calculations at each stage calculate not only the values for the current settings, but also for settings that are slightly different or perturbed. This is a different process to applying multiple separate settings because it places a restriction on the perturbations to reduce the amount of additional calculation. This constraint requires any perturbation to be so small that it does not affect the order of any events within the system. This can limit the usefulness of the technique for systems that are excessively sensitive. It also complicates the system function calculation, but in a high dimensional problem should provide quicker estimates of the gradient than finite difference methods. Further developments and research into perturbation analysis in stochastic approximation can be found in Reference [16].

3.2.4. Branch and Bound

Branch and bound optimisation strategies typically rely on mathematical methods to estimate lower bounds and realisable bounds to a problem. Lower bounds describe the best results any settings can achieve on the current problem. This enables the process to assess if it has found the global optimum, or if it is sufficiently close. Realisable bounds or results are measured from applying settings to the system. Branch and bound then uses these two methods to take a problem and measure the theoretically best case given by the lower bound and the best solution that has been found given by the realisable bound. If these two bounds are the same then the optimal values have been found and the process may end. If not, then the problem is separated into sub-problems and both bounds are recalculated. Lower bounds in sub-problems that are greater than the current best realisable bound will mean that the selected sub-problem cannot contain any better results. This prevents additional calculation of the realisable bound.

3.2.5. Metamodels

A metamodel can be defined as a simpler model of a more complex model. In simulation optimisation it can be used to approximate the simulation process to filter the parameters applied to the actual simulation. This is particularly useful when the simulation process can take a large amount of processing. A metamodel is a coarse representation of the simulation process used to reduce the number of times the simulation must be run to find an optimum. It can be constructed from modelling the relationship between input and output parameters of the simulation, treating the simulation as a black box. It may also monitor the expected error between its results and the simulation’s results, giving a confidence measure alongside a setting's suitability. April *et al* [17] suggest the use of metamodels with particular reference to the type of techniques presented in Section 3.5.2.

Alternatively metamodels can be constructed using a model that is easily solved. For example it may be known that the system is non-linear and a linear metamodel could be adapted to match the current behaviour of the network. This could be extended to non-linear modelling techniques that may be solvable mathematically and can also approximate to a system’s behaviour. Both these use the metamodel to produce solutions that can be applied to a more advanced simulation or model or to the real system.

3.2.6. Clique Detection

Clique detection [18,19], although not actually an optimisation technique, is a useful strategy for aiding in the optimisation of some graph problems. In network optimisation, it can commonly be the relationships between many base stations or other network points that makes sharing some common resource more difficult. Clique detection allows areas within the network that have a high degree of “connectedness” to be extracted and these areas can be called clusters or cliques. This can enable difficult sub-sections of the network to be optimised before optimising the system as a whole.

3.3. Using ‘Optimal’ Settings in the Real World

In the course of optimising network parameters, some aspect of network modelling must take place. This may be simple equations or complex simulations, but each model cannot fully represent a real network. This therefore ties us to being able to make optimisations on models and these ties may mean that optimal parameters for the model are not as optimal on a real network. It is also important to consider that the error between the model of the network and the network itself may be significant enough that the error could grow, ie, there is an instability or cumulative effect that is not modelled and therefore the optimal parameters are unable to balance these effects. Applying noise to aspects of the model to include a degree of uncertainty in the model may enable the transfer of ‘optimal’ parameters between a model and a real system. This approach has been considered in Reference [20] and there was found to be a distinct difference in the ability of ‘optimal’ parameters to be transferred to the real system and this was dependent upon where the noise was applied in the control loop. Further discussions of the transfer of robotic controllers from simulation to the real world show a high degree of sensitivity to the degree of noise added to a simulation [21].

3.4. Continuous Value Optimisation Methods

The following techniques and methods are designed to be applicable to continuous valued parameters. This does not mean that they cannot be applied to discrete valued problems, but they will need to be altered to cope with this added constraint. It may also be impossible to define a gradient for some problems so the techniques in Section 3.4.1, which are gradient-based methods, may need to be applied on sub-problems that do not include these limitations. Section 3.4.2 describes methods that do not require gradient information to work.

3.4.1. Gradient Based Methods

3.4.1.1.Stochastic Approximation

Stochastic approximation is an optimisation technique that attempts to provide optimum parameter settings even given results that suffer from noise. Mathematically it has been shown to 'weakly' converge on the system optimum after an infinite number of iterations. The actual process of stochastic optimisation is governed by the following equation with its accompanying constraints.

$$\begin{aligned} X_{k+1} &= X_k + a_k \nabla f(X_k) \\ a_k &\in \Re, \mathrm{Lim}_{k\to\infty}(a_k) = 0 \end{aligned} \tag{1}$$

where the reduction of a_k, reduces the effect of the gradient of the system function as the number of iterations increases. Unlike some of the other techniques, stochastic approximation does not compare the new parameters, X_{k+1}, with the old, X_k, and select the best with respect to f(X). Instead it requires the process to run sufficiently long that the mean gradient directs the optimisation process to the system minimum. This also allows local minima to be escaped from as the estimated gradient is expected to vary if the simulator contains a stochastic element. Alternatively the gradient estimates may be perturbed by a random variable to simulate this behaviour and enable the technique to escape local minima. Although stochastic approximation does provide a simple process to search for or calculate the optimum, it does require the gradient to be calculated at each iteration. This can be an expensive measure to calculate since to calculate it in n-dimensional space requires $n+1$ different measurements to calculate a simple gradient measure. More accurate difference methods require even more measurements, but this process must be repeated after each alteration of the parameters, X_k. This problem has led to a number of techniques that attempt to calculate the gradient more efficiently, eg, perturbation analysis presented in Section 3.2.3.

3.4.1.2.Gradient Descent Method

The gradient descent method is a very simple method for optimising a problem. The technique randomly selects its initial set of parameters. It then compares the 'fitness' of the current parameter set with that of neighbouring parameter sets. The parameter set with best fitness is then chosen as the new parameter set. This process is then repeated with the current parameter set being compared to its neighbours repeatedly until either a certain number of iterations are performed, or the current parameter set is better than any of its neighbours, in which case an optimum is found. Gradient descent does not avoid local minima, but it can be used within more advanced techniques as a sub-technique. The distance between the current parameter set and its neighbours is controlled by the gradient of the system function for the current parameter set. This is reflected in Figure 7 where two start points are selected, one on the left and the other on the right. The right half of the system function has a steeper gradient so the selected neighbours are a greater distance from the current parameter setting. This allows the gradient descent process to reach the minima more rapidly in the right hand half of Figure 7.

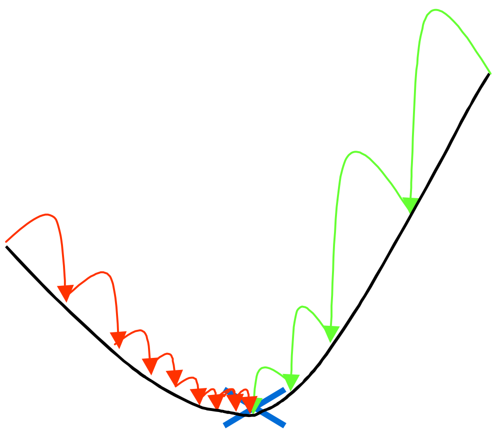


Figure 7 A comparison of the effects of the gradient on the convergence of the gradient descent method. The two starting points on the left and right are both able to find the minimum, but with a different number of iterations.

3.4.2. Non-Gradient Based Methods

3.4.2.1.Nelder-Mead Simplex

The Nelder-Mead Simplex algorithm [22] performs its optimisation process whilst maintaining d+1 solutions, where d is the number of dimensions. This would allow a simplex to be constructed from the d+1 solutions. The algorithm is initialised with parameters X_0, X_1, X_2....X_{d+1}, which are randomly distributed at the extremes of feasible solutions to a problem. Each X is then evaluated using the system function, f(X). The parameter set with the worst cost is then removed and replaced by a new set of parameters. This new parameter set is constrained to be on the line that lies perpendicular to a surface that can be formed by all the other parameter sets in the problem space and passes through the discarded point. After the point is added to the simplex, the next worst point is removed and the simplex is redefined again. This iterative replacement of parameter sets causes the simplex to reduce in size and converge on a minimum as a whole [23]. The stopping condition for the optimisation process could be that a number of iterations of the algorithm have been performed or the improvements or size of the simplex has reached a certain threshold.

In Figure 8 we provide a simple example of the Nelder-Mead Simplex algorithm in operation. A simplex is formed in a two-dimensional problem space using three vertices. The figure illustrates how the simplex shrinks as the vertices of the simplex are replaced. In this example the initial parameters are represented as the outer corners (the blue triangle). The first iteration then considers the top right hand vertex to have the worst performance. This parameter setting is then replaced with an improved setting found along the line running perpendicular to the ‘surface’ formed by the other two vertices and the rejected parameter set (the parameter sets now form the green triangle). The second iteration then considers the left most parameter setting to give the worst performance and so this is replaced with another new vertex, again with the new vertex being constrained to lie on the line perpendicular to the ‘surface’ formed by the other vertices and the rejected parameter setting (the parameter sets now form the red triangle). The simplex has thus reduced in size, and should be closing around the optimum of the system.

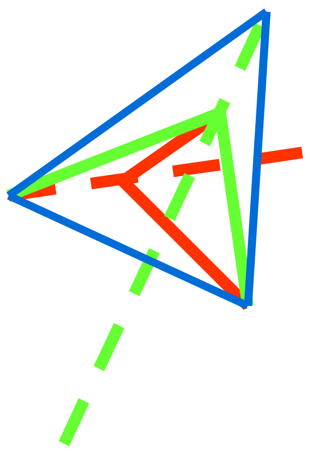


Figure 8 A simplex in 2D problem space is shrunk by replacing vertices to form a smaller simplex centring upon a minimum. This is repeated, shrinking the simplex further.

3.4.2.2.Hill Climber

The hill climber [24] algorithm is very similar to the gradient descent algorithm, with the most notable exception that it does not use gradient information to guide its optimisation approach. It is also formulated to find the maximum of the function, f(X), but this can be changed to searching for the minimum by returning any result multiplied by minus one, ie, -f(X). The hill climber algorithm is started with a random parameter set, X_0. The result of applying this parameter set is then compared with its neighbour(s). The neighbours are selected by altering parameter values in X_0 to produce new parameter values, X_n. The hill climber can then either select the best neighbour after evaluating all of its neighbours, or it can select the first neighbour that offers an improvement. In either case the process is then repeated with new neighbours being generated and better neighbours replacing the current best solution. The process stops when none of the neighbours are an improvement on the current parameter set. This can occur when the hill climber has found the global or a local maximum as there are no guarantees of finding the global maximum in hill climbing.

Figure 9 shows an example problem surface and three randomly selected starting parameters for X_0. In Position 1 the hill climber finds a local optimum by gradually comparing its current best solution to its neighbours and selecting those further up the slope. This behaviour is repeated for Points 2 and 3, but Point 3 finds the global maximum. Whether or not Point 2 results in the global maximum being found is dependent upon Point 2’s position within the ‘valley’. As a result of the hill climber being unable to escape local maxima (eg, such as the iteration starting from Point 1), the hill climber should be restarted with a new random X_0. This enables more of the parameter space to be searched for optimal solutions. Improvements on the basic hill climbing strategy have been proposed in Reference [25].

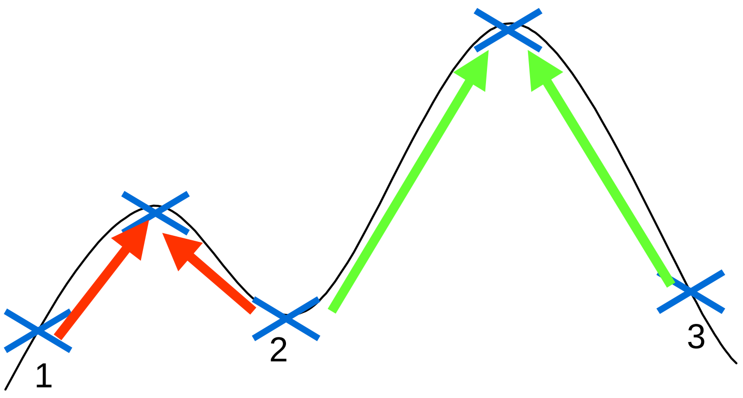


Figure 9 An example problem surface in black with three randomly selected starting parameters, X_0. Each parameter is shown to find the global or local optima.

3.5. Discrete Value Optimisation Methods

The following sections describe a range of optimisation methods that consider discrete valued problems. Section 3.5.1 considers two methods that are designed to optimally select parameter settings when there are a very limited number of possible variations. More general approaches to optimisation of discrete valued problems with many possible parameter settings are considered in Section 3.5.2.

3.5.1. Small number of input values

Many of the techniques presented in this report attempt to select the optimal parameter settings given a large number of different options. This section particularly focuses on a different aspect to optimisation that attempts to optimally select a set of parameter settings given the system that is needing to be optimised is stochastic in nature. This means that the optimal parameters may need to be resilient to different random effects and that any given simulation may give better results than the 'optimal' setting for a less than optimal setting.

3.5.1.1.Rank and Select

Rank and selection [26] takes a cardinal approach to optimally selecting some parameter set from a small number of options. The process of rank and selection evaluates all the parameter sets using the stochastic system function and collects results for each set. This is then repeated a number of times with the mean and variance of the results for each parameter set being collated. The results are then ranked according to some combination of the mean and variance before either the top setting is selected or the top *x* settings are selected and the process is repeated to obtain even more accurate results for each of the settings before a final selection is made.

3.5.1.2.Multiple Comparison Procedures

Comparison procedures [27] differ to rank and selection by comparing the different settings in an ordinal manner. Instead of calculating the mean and variance of each technique according to the system function, each of the parameter settings are applied to an approximation of the stochastic system function. This approximation should enable direct comparison between different settings on an instance of the system undergoing similar stochastic effects. The results for the different settings are then compared, with each setting winning or losing in comparison to all the other settings.

The results for each setting and each approximation of the system function are then combined to give the parameter settings that perform the best. This best can then be used as the proposed optimum, or the top *x* settings can be selected again and the process repeated.

3.5.2. Large number of input values

Problems that involve a large number of different input values will need some systematic method of searching for optimal values. This section considers many different algorithms and ideas for optimising a set of parameters with ranges of values that mean an exhaustive search is impractical in the available time.

3.5.2.1.Tabu search

Tabu search [28, 29, 30, 31] builds upon the idea of inverted hill climber[1] optimisation, but also includes the ability for the optimisation process to temporarily go ‘up hill’, whilst at the same time attempting to avoid returning to the same point in the problem space. The particular aspect that gives tabu search its name is that as the technique examines the parameter sets that neighbour the current best solution, some of the neighbours are considered “tabu”[2] and are prevented from being selected as the next ‘best’ point in the parameter space. This tabu status may be caused by the point having been previously encountered, or having been in a direction that has commonly or uncommonly been taken in the search for the optimum. This limits the number of possible options for a tabu search algorithm to take. The rules for generating the tabu points at each new position can contain problem specific knowledge, but will typically involve some aspect of the history of the search so far. Each iteration of tabu search considers the neighbours of X_k and selects the neighbour with the lowest f(X). This allows the search process to escape local minima, if the tabu list can contain information on the location of previous optima. Although points are given tabu status, they may still be evaluated with the system function. This enables an aspiration function to select tabu points that are significantly better than non-tabu points, and override the tabu list given that the point gives a significant improvement.

Additionally tabu search can also include the notion of a second fitness function. This second function can be used to escape local minima, assuming the alternative function relaxes system constraints. These behaviours, matched with the ‘memory’ aspects of tabu search, attempt to enable local minima to be escaped.

In Figure 10 we show a possible path that could be taken in a tabu search. In common with many methods the initial settings are selected at random and the tabu search proceeds to find the local minimum by considering its neighbours. On reaching the local minimum it continues to the right as neighbours to the left are present in the tabu list having been recently evaluated. This allows the tabu search method to escape the local minimum, before (at the third cross) it is able to descend into a better minimum.

[1] Here the term ‘inverted hill climber’ is used to signify that the algorithm is formulated to descend into the valley, rather than climb the hill.

[2] “tabu” is an alternative spelling of “taboo”, and is used here to match the literature that describes the “tabu-search” method.

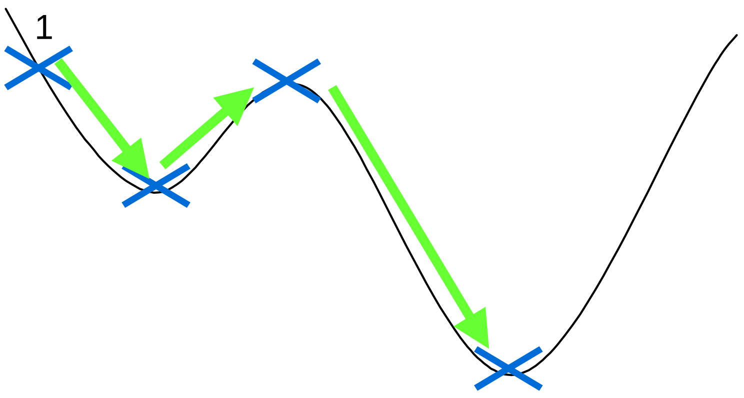


Figure 10 An example of a tabu search method finding a minimum for a simple problem surface.

3.5.2.2. Simulated Annealing

Simulated annealing [32] is an optimisation approach that takes an analogy from the physical process of annealing materials to achieve good internal structure by managing the cooling process. Annealing attempts to prevent structures forming in a material that are not at a minimal energy state and in simulated annealing this is equivalent to avoiding local minima. In annealing the temperature is lowered, and during simulated annealing the allowable variations in system 'energy' are reduced over time. This energy must be defined by the system being optimised and should reduce as a more optimal set of parameters is found. During simulated annealing the optimised parameters are selected dependent on the change in system energy. New parameters that lower the system energy (ie, provide a better solution) are kept. Parameters that produce a higher level of energy within the system (ie, provide a worse solution) are also kept with a probability, p, as defined by the following equation:

$$p = e^{\frac{-\delta E}{T}} \tag{2}$$

where ∂E is the positive change in energy, and T is the 'temperature' of the system at its current stage of annealing. This means there is a lower probability of accepting changes in parameters that cause greater increases in system energy (ie, provide worse solutions) as the temperature, T, is lowered. This behaviour of simulated annealing allows for the optimisation process to avoid local minima to a certain extent by accepting some changes that give worse results. The initial starting 'temperature' of the system, when to lower the temperature and by how much are all considered part of the annealing schedule. These aspects can be difficult to define, as can a useful measure of the energy in the system to use to indicate any improvements.

Figure 11 shows the progression of a simulated annealing technique towards a minimum. The initial parameter settings, Point 1, are randomly selected. Randomising some aspect of the current parameters generates a new parameter set. Thus from Point 1, the technique compares Point 2. This is an improvement and so is accepted, similarly Point 3 and 4 are both improvements. Point 5 is however a worse option to that of Point 4, and its selection relies on the temperature of the system. A high temperature gives a greater likelihood of acceptance and a low temperature will be more likely to reject this degradation. Assuming Point 5 is accepted the process would continue on to Points 6 and 7. It is then unlikely that the parameter setting of X_k will 'escape' the minimum as the temperature is lowered, because the probability of the system having sufficient energy to jump out of the valley will become very small.

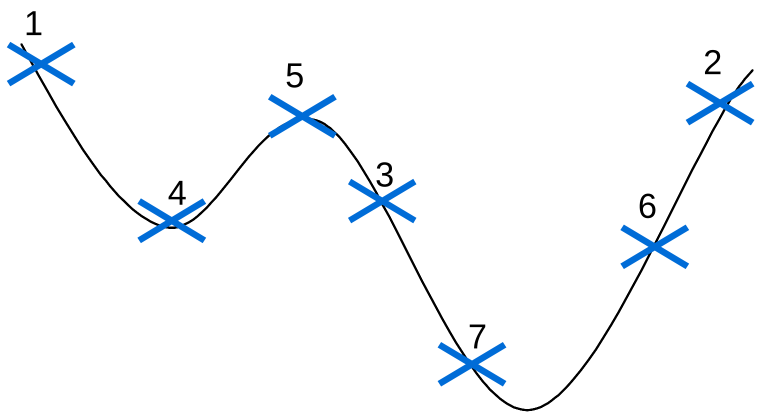


Figure 11 An example of the simulated annealing optimisation method finding a minimum for a simple problem surface.

3.5.2.3.Evolutionary Programming

Evolutionary Programming [33] is one of three algorithms presented in this report that are based upon the biological theory of evolution. Evolutionary programming is distinct from the following methods in its use of selection and mutation in defining new settings for the system. In evolutionary programming the parameters, X_k, are encoded in a 'genome'. This genome is then subjected to a process of mutation where each part of the genome may be altered causing a 'mutation' in the genome. The genome may be a simple vector, but may include transformations of various parameters to better fit the processes of mutation. One such transform is the conversion of integers from a standard encoding scheme to a gray code. This is a useful conversion when the effects of mutation are considered on the binary form of an integer. A single bit 'mutation' or toggling for a binary encoded number can give a very large mutation if the random process toggles one of the most significant bits. Gray coding integer values however can limit the numerical distance between the original and the mutated values. This can allow the parameters to be altered in a less volatile way. To add to the concept of mutation, evolutionary programming also maintains a 'population' or set of different parameter settings rather than a single current best. This population is added to by taking a current member and applying mutation. Typically this may be repeated for each member of the population. A process of selection is then used to reduce the population size, maintaining it at a predefined size. Selection is commonly a probabilistic process that favours genomes that have better performance for the system, but does not guarantee that unfit members of the population will be 'culled'. Through selection, a member can remain in the population through multiple generations. Evolutionary programming differs from genetic algorithms in its ability to operate upon parameters that may contain real numbers.

3.5.2.4.Evolutionary Strategies

Evolutionary strategies [34, 35, 36, 37] are very similar to evolutionary programming techniques. An evolutionary strategy can use the processes of selection, mutation and recombination (a process similar to the cross over mechanism found in genetic algorithms). It is designed to be able to optimise real valued problems. The genome of an evolutionary strategy contains not only the parameter settings, but also the settings that control the mutation. Variations applied to each parameter are zero mean Gaussian distributed, allowing larger variations less frequently, but with the aforementioned settings the mean and variation of the distribution can be controlled. The processes of mutation and selection are similar to those in evolutionary programming, but the process of recombination is different. Recombination uses multiple ‘parent’ genomes to produce a new genome for selection. This recombination can for example take the first part of one genome and the second part of another to make a new genome to be added to the population before selection. There are many different forms of recombination and the process can also be tailored to fit the specifics of the problem undergoing optimisation.

3.5.2.5.Genetic Algorithms

Genetic Algorithms [38, 39, 40] consider the parameters of a problem to be a vector representing the ‘genes’ of the solution. A number of possible solutions are then generated to form an initial population of solutions; each member of the population has their own genes forming a genome. This population then undergoes a series of transformations to form a new population. These transformations can involve selection, mutation, and cross over or some other form of combining. After generation each genome is applied to the system and its ‘fitness’ or response from the system function is measured. Each member of the population is typically used to create the new population in proportion to its ‘fitness’. The selection process selects members of the population that are to be involved in producing the next generation of solutions. This removes genomes that are considered unfit, typically in a probabilistic manner, thus favouring the ‘fittest’ members. However, it also allows poorly performing members a chance to contribute to the next generation. After selection, the process of cross over can be applied which takes genes from two or more members of the population and combines different parts of the parent genes to form a new genome or solution. Mutation can then also be applied altering the genes (or parameters). Both of these processes allow for the parameter space to be investigated further. Implementing a genetic algorithm does not require that selection, mutation and cross over are all performed. An example of a genetic algorithm being used to optimise a set of parameters is shown in Figure 12.

In Figure 12 the population is initialised with 3 random genomes, 1_a, 1_b, and 1_c. These genomes are then evaluated and 3 new members of the population are created through mutation and crossover to form members 2_{ab}, 2_{ac} and 2_{bc}. The total population is then put through a process of selection where it would be expected that the second generation members would be more likely to be selected as they have better fitness in this minimisation problem. Member 3 may then be a likely descendant of this new population, but it would depend on exactly how the mutation and cross over operations worked on the underlying genome.

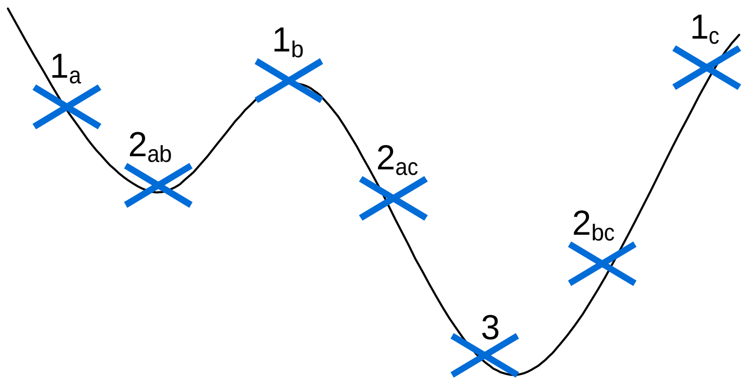
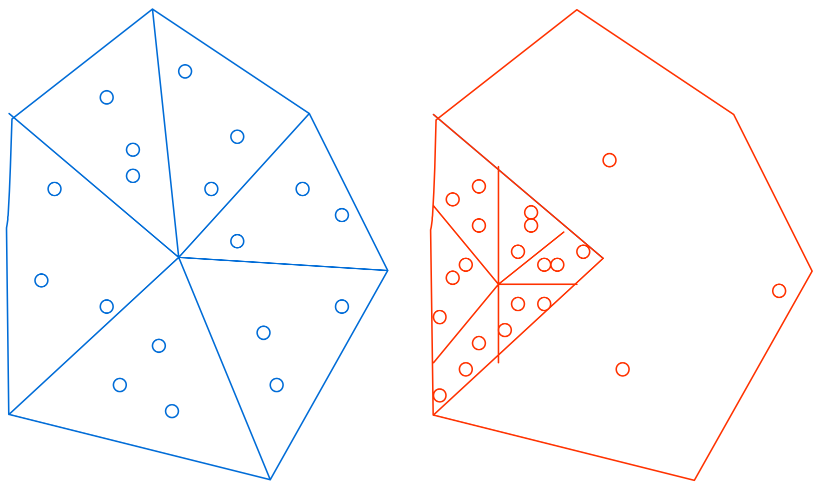


Figure 12 A possible history of a genetic algorithm searching the parameter space. The first generation, 1_a, 1_b, and 1_c are mutated and combined to form the second generation, 2_{ab}, 2_{ac} and 2_{bc}.

Genetic algorithms are still under research to understand exactly how each aspect of selection, mutation and cross over affects the optimisation process. In general, without mutation a population’s diversity may degenerate as variation in genes are likely to be lost without any method for regaining variation. With mutation, a perfect solution is always degraded, and an imperfect population can reach an equilibrium point where it cannot improve due to the level of mutation occurring in each generation. Crossover can increase the rate at which good solutions are found. One issue with genetic algorithms is the common requirement to be able to control the solution using a genome (set of genes), which is typically binary in nature.

3.5.2.6.Nested Partition

The nested partition algorithm [41] for optimisation attempts to solve optimisation problems by dividing the parameter space into partitions. The initial steps of the algorithm consider the whole of the parameter space and partition it into a number of partitions. Each of these partitions is then sampled from, and the results from each sample taken from the parameter partition are combined to give a representative value for that partition. An example of a search space having undergone this process is shown in Figure 13a. The partition with the best representative value is then divided into more partitions whilst all the other partitions are combined to form an outer partition. This is shown in Figure 13b. The process is then repeated with these new partitions. It is then expected that the algorithm will slowly select a smaller and smaller partition that has increasingly favourable results. If, however, at any stage the outer partition contains a better representative value then the algorithm selects the outer partition as the new partition and this is divided again into new partitions and the process continues. This option to select the outer partition enables nested partition optimisation to escape local minimum to some degree. To make sure the representative value is a useful measure, the number of samples taken from all the partitions is dependent upon the size of each individual partition. The algorithm can stop when the selected partition contains a single parameter setting or the partition is below some other size-based threshold. Alternative stopping conditions such as after a certain number of iterations or when a solution is close enough to a theoretical optimum may also be used.



(a) First Iteration (b) Second Iteration

Figure 13 Applying a nested partition technique to an irregular shaped search space.

4. Control Theory Techniques

Although the main task of this report is to evaluate current optimisation approaches, in practice, the use of these techniques with a simulator may not reach the ‘real-time’ operation criteria that may be needed to optimise certain aspects of the 3G network. Control theory could provide methods for controlling the network in ‘real-time’ and if the controllers were able to be adapted then this may provide a method for optimising the ‘real-time’ behaviour of the network. There are also sections of the control theory community that consider adapting the controllers whilst they are operational. This chapter therefore considers static or non-adaptive control theory and also adaptive control theory.

4.1. Non-adaptive Control

Control theory has been applied in many industrial settings where a process or plant must be controlled to give a certain output. Examples are found in managing chemical reactions in a chemical plant, the movement of a conveyor belt system in a factory or the power control between a mobile handset and a base station in the 3G mobile network. This control can either be open- or closed-loop in nature, where a closed-loop system takes the output of the process or plant and uses it to make future control decisions. In both cases the plant is modelled by a transfer function that takes the output of the controller and attempts to reflect the output of the plant. This modelling of the plant allows an open-loop system to predict the response of the plant to given input settings. It can also enable a feedback loop of the controller and the plant to be analysed to ensure ‘good’ behavioural properties of the overall system. These ‘good’ properties will include some measure of system stability. Stability in control can mean that for a given bounded input, a system produces a bounded output. This criterion relies on the state of the system being controllable and observable, ie, it must be possible to set the inputs of the plant such that a system state is reachable, and it must also be possible to observe the state of the system to avoid ‘bad’ system states. If either of these criteria is not satisfied, then stability may not be guaranteed.

4.1.1. Open-loop Control

Open-loop control is a type of control strategy that contains no feedback loop. It can be used when the inputs to a system can sufficiently control a system. Turning on a light would be an open-loop system, but if the light were to have blown no feedback is present to inform the controller that the behaviour has failed to meet its aims. We do not expect open-loop control to contribute significantly to Project Monotas.

4.1.2. Proportional-Integral-Derivative Control

The Proportional-Integral-Derivative controller or PID controller is a commonly used feedback controller. Its three separate parts provide different abilities to control a system. If we consider the system function f(X), and the target or reference value for our system then the difference is the current error. The proportional aspect of the controller will change the parameters, X_k, proportionate to this error. The integral controller changes with respect to past error between the reference level and the system output. The differential controller changes X_k with respect to the current rate of change of the system output. Each of these different controlling aspects adds the ability to select the most desirable control behaviour. Figure 14 shows this graphically, highlighting the aspects that the proportional, integral and derivative parts of the controller use to influence the degree of their affect on the new parameters.

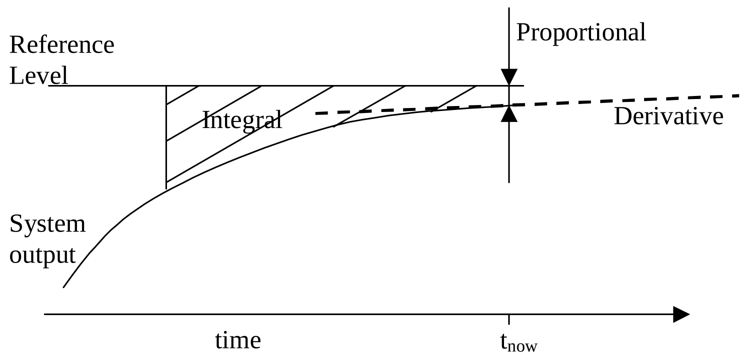


Figure 14 Given a reference signal, the proportional, integral and derivative aspects of a PID controller act on different aspects of the current output.

In designing a PID controller the system itself must also be modelled. This allows the overall system behaviour to be designed to match requirements of stability, speed of response and whether or not the system output may over shoot the target value.

If the modelling of the system is flawed it may cause instability and deviant performance. However, there are methods to make a controller more robust against certain inaccurate estimates of the system.

4.1.3. Model Predictive Control

Model predictive controllers (MPC) [42, 43] attempt to control the error that a system will produce given its past, present and predicted future responses. This can allow a model predictive controller to cause the current error to increase given that the expected error experienced over the predicted future will be lower. This type of controller has been used successfully in many industrial applications. It models a system in a similar way to a PID controller, but has the added advantage of predicting future responses. This can allow constraints on input and output parameters to be fulfilled by predicting the future behaviour of the system and so avoiding current behaviour that may cause these constraints to be violated in the future.

4.2. Expert Systems

Expert systems are rule-based programs that can take a set of predefined rules and apply them through inference to diagnose a problem or generate a response to a situation or system state. The rules in an expert system are a series of ‘if-then’ statements that are typically defined by an outside expert. There is a possibility that the rules could be adapted and altered by some external optimisation scheme, but the impact of alterations to rules can be hidden, as there are no predefined connections between rules that the method of inference would use. Expert systems can be directly improved by an expert, and new knowledge can be added to the system to enable it to respond to new stimuli and conditions as well as to give different responses. If an expert system was adapted by an optimisation process, with the choice of rules to include within an expert system controller being optimised, it may enable a controller to be developed that would be able to give an explanation for its behaviour. This may prove useful in developing an understanding of any system being controlled. However, because the outputs of the system are generated from the inference of different rules, this may produce a system that performs stably over all tested cases and may behave errantly for situations that may be different by a fraction.

4.3. Adaptive Control

Adaptive control in its forms presented in the next sections attempts to use a system's history to improve the current control strategies. This can be either through direct alteration of a controller's own parameters, or through those of a model. The study of adaptive control and learning systems has developed from the fields of psychology, statistics, computer science and neuroscience. Because of this, some of the ideas are biologically inspired, some have a strong mathematical basis and others have been developed iteratively as the power of computers has grown. Initially we consider the trade off between exploring new settings for a system and exploiting known settings in Section 4.3.1. The next two sections then discuss the need for any adaptation of the controller to be as fast as the underlying system, and the difference in approach between adaptively modelling the system being controlled or adaptively controlling the system directly.

4.3.1. Exploitation *vs* Exploration

Before employing any adaptive control strategy it is important to consider the trade-off between exploitation and exploration of any adaptive control strategy. If the primary consideration of a control strategy is exploitation, then all actions performed by the system will be targeted at placing the system into an optimal configuration given the current knowledge of the system. This optimal configuration may not be optimal in any global sense, but relies on knowledge captured from either the designer or from previous exploration. A controller that only considers exploitation may become fixed in its ability to make control decisions. Exploration however attempts to select parameters that may provide future benefits, but that contain an aspect of risk in the form of unknown effects. During any exploration of system settings the performance of the system is unlikely to be the best, given the current knowledge of the system. It should, however, allow new control behaviours to be discovered and possibly better controller behaviour to be discovered. In a commercial system such as a 3G network, system degradation due to exploration would also need to be quantified or limited. Examples are also given in the literature that suggest that even if a system seeks to only take the current 'optimal' action based on past history, ie, to act exploitatively, that the system may explore due to other factors, such as stochastic effects within the system or other complex effects. These examples are found in developing game playing software for backgammon [44, 45, 46] and checkers [47] where only exploitative methods are used, but the controllers 'learn' or adapt. It could be hypothesised that in the case of backgammon exploration is forced by the random elements of the dice used to govern the movement of pieces, and in checkers by the controller playing against itself.

4.3.2. Rates of Change: the System *vs* the Controller

Adaptive controllers must be able to change faster than the system they are controlling. If a controller adapts slower than the system it is controlling, it will not be able to control the system optimally and the system as a whole will suffer from lag. This constraint will limit the accuracy and processing time that will be available to any proposed method to maintain the controller's ability to adapt in line with the system.

4.3.3. Adaptive Model-based Control *vs* Adaptive Control

To provide control for a system, knowledge of the system must be contained in the controller that either matches or mirrors the system. One approach to controlling a system is for the adaptive controller to model the system in a form that the controller can extract information from or use more readily to draw conclusions. These conclusions could predict results for a series of actions the controller may perform with an expected reward or performance benefit. This type of behaviour allows a controller to test or hypothesise about its choice of actions and solve the optimisation problem with knowledge of the system. An alternative is to use a simpler, 'solvable' model of the system, ie, if the system is non-linear, the model may be linear. This simplification allows for control strategies to be produced for a linear view of the system. This approach is appropriate if the model can be updated often enough such that the current linear approximation matches the current mode of the non-linear system.

Adaptive control without a model directly adapts the controller. Previously in Section 3 it was stated that optimisation algorithms solved problems more effectively when their structures were aligned. The effective adaptation of the controller must therefore be the process of aligning the controller's structure to the system. This alignment is the same as encoding knowledge of the system into the controller, but in a 'solved' format. This second method must therefore combine a method for adapting and a method for solving that is separated in the model-based approaches.

4.3.4. Iterative Learning Control

Iterative learning control [48], unlike other control strategies presented in this report, is designed specifically for repetitive tasks. The process of adaptive control or controller optimisation is triggered after each iteration of the repetitive task.

This type of control scheme may only be useful if certain tasks can be identified as repetitive and can be triggered from the network. This could allow layers of controllers where an iterative learning controller would control sub-tasks that could be triggered by a more 'senior' controller.

4.3.5. Genetic Programming

Genetic programming [49] is a very similar paradigm to genetic algorithms. The difference lies in the building blocks that the genome controls. A genetic algorithm's genome is the parameters that are directly fed into a system; a genetic programming's genome contains the functions that control the system. For genetic programming this allows the designer to place problem specific knowledge into these initial functions. This should simplify the problem that the genetically inspired algorithm must solve, whilst at the same time providing assumptions and constraints as to how to solve a problem where innovative and more efficient methods may exist, but are hard to construct from the set of functions that was used to initialise the algorithm.

4.3.6. Artificial Neural Networks

The area of artificial neural networks [50, 51] (ANNs) has been developed from biologically inspired models of connected neurons. Their use within Project Monotas could be as controllers that are optimised either before deployment, and/or whilst deployed. An ANN consists of a network of simple processing units that take a set of inputs, combine them in a simple manner, condition the result and then output the result. The output can either connect to another neuron or be a controller output used for setting the parameters. Example combining functions are a weighted sum or a logical AND. The conditioning function can be a simple multiplication of the sum, a thresholding operation or a more complex function such as a sigmoid. Differences between ANNs with the same number of neurons can be due to different combining functions, different conditioning functions or different connections between neurons. Variations between ANNs may also be encoded in the parameters that control each of these aspects.

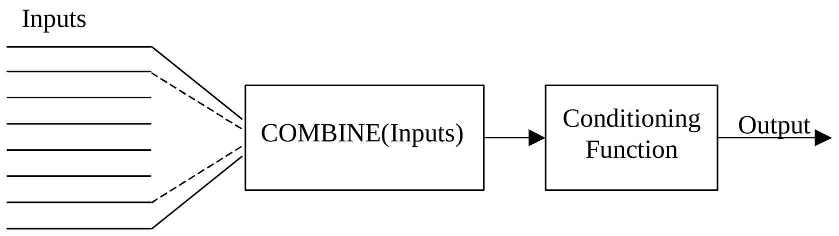


Figure 15 An example structure of a simple artificial neuron in an artificial neural network.

After defining the internal structure of the neurons used in the ANN, they are connected together to form a network. These connections will include connections from inputs, eg, network statistics, and connections to outputs, eg, network parameters, as well as inter-neuron connections joining neurons together. Further to this, ANNs are also divided into two classes dependent upon whether there are cycles in the connections of the network. Networks without cycles can be called feed-forward networks, and those with cycles can be called recurrent networks. Both are shown in Figure 16 with the feedforward network in (a) and the recurrent network in (b). There also exist a number of other names for the two different types of ANN as researchers from computer science, neurology, psychology and physics have all been involved in the field of ANNs. Recurrent networks give the network a form of memory, but are harder to train directly.

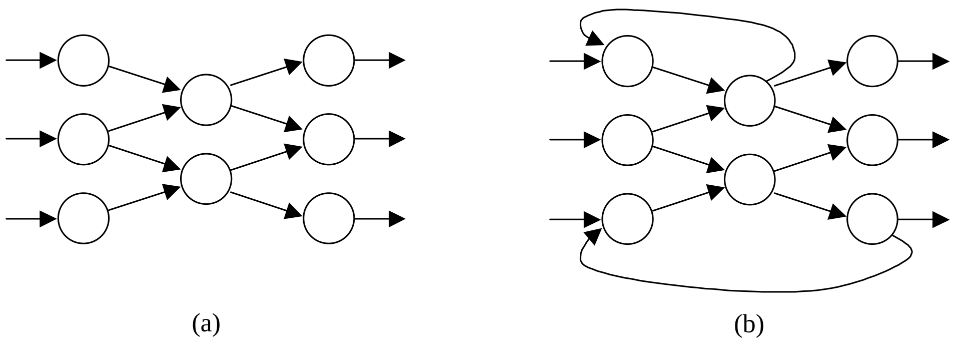


Figure 16 An example structure of a feedfoward and a recurrent artificial neural network in (a) and (b), respectively.

There are a number of different methods for optimising or training an ANN. Optimisation techniques such as genetic algorithms can be used to optimise the internal parameters of an ANN, but do not necessarily take advantage of the structure of the ANN being optimised. More direct approaches to optimising or training an ANN are techniques such as back-propagation. Larger, more highly connected ANNs can be more difficult to train because the parameters are coupled. The structure of an

ANN is also a consideration since it is often predefined before the 'learning' process begins. Trade offs exist when deciding the number and structure of the neurons in an ANN. With too many neurons, the network will be more difficult to train, with too few the network won't be able to meet the challenge. This trade off is considered in the technique presented in Section 4.3.10, which considers varying the structure of the ANN by adding new neurons and connections as well as managing the evolution of each neuron's internal parameters.

4.3.7. Markovian Decision Processing

Modelling a system as a Markov decision process [52] (MDP) allows predictions to be made as to the optimal action for the current state. MDPs are processes whose current best action is only influenced by the current state. Not all past state transitions influence the decisions of the current state. To optimise for a modelled MDP-like system, each set of available actions will carry a reward associated with choosing it. If the optimisation process is just considering the best current action to take, then the model can simply be used to look up the current best action. After each action, the reward can be updated with the results from repeating that action. It is also possible to use the MDP to select the best action if the rewards from the next *N* states are considered. To calculate the reward for taking action *A* and the best following action we can use the following equation,

$$R_{A_2} = R_{A_1} + \alpha \sum p_{n|A} R_{n\max} \tag{3}$$

where R_{A_1} is the reward for taking action A in the current state and R_{A_2} is the reward if the following state is considered. $p_{n|A}$ is the probability of arriving in the *n*th state given action A is taken in the current state. $R_{n\max}$ is the best reward that is available in state n. α is used to weight the future rewards. This equation can then be used to calculate the reward for each action in the current state with a 'horizon' of 1, ie, the process is looking into the future 1 step. This will allow an action that performs badly to be taken if it is likely to make the system enter states with highly rewarding actions. This process of looking ahead can be repeated to consider the best current action if two future actions can be considered. The reward from each future action can be summed evenly ($\alpha = 1$), or future rewards can be reduced depending on how far ahead they are ($\alpha < 1$). This allows earlier rewards to be favoured. The number of iterations of looking ahead, or the 'horizon', can be set to different lengths. That is, in the example the horizon was just one, but it could be extended further by iteration. This becomes exponentially more expensive as the horizon is extended. One of the key aspects to this approach is that to be able to control such a process it is important that the current state of the system is fully observable. MDPs are also closely associated with stochastic learning automata [53] that are able to control MDPs.

4.3.8. Training and 'Learning' methods

4.3.8.1.Reinforcement Learning

Reinforcement learning is a concept associated with adapting a controller, such as a neural network, to perform a task via the use of rewards and punishments. It does not specify how the controller that is being adapted must perform the task, but purely provides feedback on how well the controller is performing. The feedback in performance for a system can be instantaneous, but it can also be delayed. This can cause a problem when attempting to evaluate the actions a controller is taking. This builds upon the ideas of Markov decision processes, where each state of the system under control is observable and then a choice in actions is possible. These actions will then generate some form of reward or punishment. Transitions from state to state of the system typically are probabilistically defined, implying that an action in one state will not control the next state of the system. The rewards generated may also not necessarily be constant for a single action in the same state.

There are two general approaches to applying this concept to adapting systems. The simplest is to use an optimisation algorithm to search for an optimal set of parameters that can be applied to the controller. Alternative methods also exist specifically for ANNs, namely, back-propagation and Q-learning. Both enable a neural network to converge on an optimum in a timescale that is dependent on the complexity of the network.

This process of reinforcement will fail if the performance characteristics are not well represented in the reinforcement measure. For example, if during training an element such as coverage is not included, or not weighted highly enough in the calculation of fitness, then the system will reduce coverage to maintain other ‘more important’ elements. Care must therefore be taken in formulating the reinforcement function.

Reinforcement learning has been used in elevator control [54]. This application has interest because the elevators were each controlled by a separate ANN controller without direct communication. The controllers were then all subject to the same reinforcement-learning signal generated from the performance of all the ANNs as a group. This signal was ‘noisy’ due to the interactions between the different controllers as they all changed at the same time. This meant that elevator controllers with different properties could be developed to serve the different requirements or behaviours of the overall system. The training of the elevators used probabilistic models of passenger requests and destinations that may have parallels to the problem of call admission in mobile networks, especially if probabilistic models of user call behaviour are also available.

4.3.8.2.Supervised Learning

Supervised learning [55, 56] supplies the adaptive controller with a set of example inputs and outputs that should be representative of the task the controller is required to perform. The adaptive controller is then altered via some scheme to produce the desired outputs given the test inputs. It is then hoped that the controller will have ‘learnt’ how to control the system in a general manner. Problems arise in using this type of learning if the training data is not sufficiently broad to cover the functionality of the system requiring control. Other difficulties may be in defining the desired response to a complex system given a known output of the system. Thus the generation of data to train the controller can be a difficult task.

4.3.8.3.Unsupervised Learning

Unsupervised learning [55, 56] is used to detect patterns in data, and is not immediately suitable for training an adaptive controller. Unlike a supervised learning system, an unsupervised learning system is only given input data without any labels describing what the data represents, or if their immediate response is ‘right’. The unsupervised system’s aim is to label the data according to structures found within the data itself. This process could evaluate a set of data with each sample of an N-dimensional size to be generated from m different independent variables. This allows for the data to be simplified to m dimensions, which could allow for easier interpretation by a user and simpler computation by any following process. References for using unsupervised learning to present to users are found in [57, 58] or for input into automation schemes in [59]. Within Project Monotas, unsupervised learning could be useful to analyse user behaviour, or to separate out different conditions experienced by network elements. This could provide further insight into user and network behaviour that may not ordinarily be found. It may also highlight interdependencies within the network's statistics.

4.3.9. Neural Swarming

Neural swarming [60] is a combination of ideas present in non-adaptive control and neural networks. The starting point for applying neural swarming to a problem is a neural network trained to perform the task of a PID controller. This PID controller should be chosen such that it can effectively control the current state of the system. This ANN, trained as a PID controller, is then duplicated with each internal parameter of the neurons being varied by a small percentage. This produces ANNs that model behaviour close to the original PID controller. The different ANNs are then placed in control of the system for short amounts of time, after which their performance is 'rated'. In between each neural network taking control of the system its parameters are updated. An example scheme for updating the parameters is described in the following equation,

$$\begin{aligned} W_{n+1} &= W_n + V_n \\ V_n &= V_{n-1} + \alpha \cdot rand \cdot (W_{nb} - W_n) + \beta \cdot rand \cdot (W_{sb} - W_n) \end{aligned} \tag{4}$$

where W_n represents the current values for the ANN, V_n represents the current rate of change of those values, nominally called the velocity of the swarm member, α and β are constants that control the rate of change within the system and *rand* is a random number between 0 and 1. The variables W_{nb} and W_{sb} represent the neural network parameter values that produced the best rating for this swarm member and the current best ever values from all swarm members, respectively. Updating the swarm member's parameters using the overall swarm's best values should keep each member centred about the current optimal values. The use of the velocity should also allow the swarm members to investigate different settings around this optimum. This process of continually exploring controller settings around the current optimum allows the PID controller to change its approximation to the system as the system changes.

Using a population of ANN models of different PID controllers can allow a complex non-linear system function to be controlled with linear controllers. The example in Figure 17 shows a more complex system function, which at different points of its operation can be controlled by the three different controllers PID_1, PID_2, and PID_3. The neural swarm technique allows different controllers to be tested around the current optimal controller. For example, if the system were to be operating such that the controller represented by PID_2 was the optimal controller, when controllers PID_1 and PID_3 are tested on the system they would receive lower ratings from the system. If, however the mode of the system changed then one of the non-optimal controllers may become the optimal controller, and the internal settings of each of the swarm members would move towards this new optimum. Due to the method of updating the member's internal variables the members do not converge, but swarm around the optimum. This may allow for slightly less than optimal behaviour, but allows the controllers to adapt and centre themselves as a swarm around the system's current optimal value.

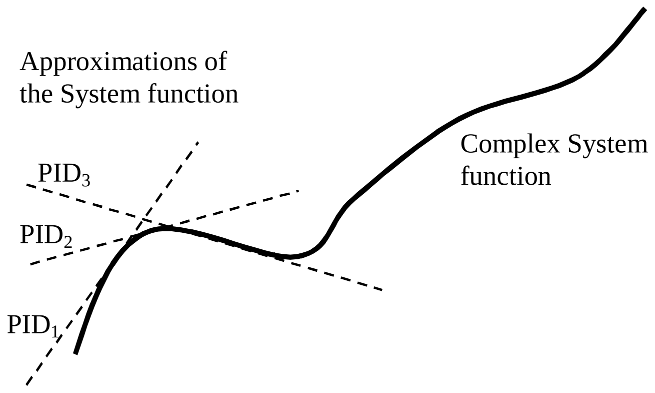


Figure 17 Using 3 different control strategies PID_1, PID_2, and PID_3 at different times will allow simpler systems to be used to approximate the more complex system function.

4.3.10.NeuroEvolution of Augmenting Topologies

NeuroEvolution of Augmenting Topologies [61] (NEAT) is similar to the developing neural networks in neural swarming. Both maintain a population of neural networks whose weights are changed to manage a scheme of iterative improvements and also a form of adaptation in line with changes in the system they are controlling. The difference between the two ideas lies in the ability for the NEAT method to add to the neural network’s topology, ie, to add complexity via new connections and new neurons. The initial size of a neural network deployed within the NEAT scheme is small, allowing for the network to be adapted to the system more rapidly. This adaptation is managed by the use of a genetic algorithm that represents the connections between the different neurons in its genome. The different members of the population are then placed in control of the system for a short time and then rated. Replacement members are then generated from fitter members of the population. Members are then discarded if their overall fitness is the lowest. This fitness rating is balanced with the number of neural networks NEAT has created with similar structure. ANNs with newly added links between neurons, or new neurons are temporarily prevented from being discarded. This allows the new structure to mature before it is allowed to be removed from the population.

5. Summary and Conclusions

This report has considered a wide range of methods that have been developed in order that computers can be used to solve difficult problems. The techniques have been grouped into the areas of network optimisation strategies, optimisation techniques and control theory. Network optimisation strategies presented in Section 2 suggested five approaches that could be useful for optimisation of networks within Project Monotas. Section 3 reviewed optimisation techniques that can be applied to difficult problems. This firstly considered frameworks to simplify the system being optimised, and then presented different techniques that can be used to find optimal settings for a system. In Section 4 the use of control theory techniques to directly influence the network were considered. These techniques should enable faster response times to changes in the system or mobile network, which would hopefully provide a greater level of performance than more static settings. This differs from optimisation by trying to define the rules or behaviours that should be used to respond to a system, rather than trying to define the system's inputs given the system's outputs. These rules form closed-loop control over a system and can be adapted or remain fixed.

The application of each of these techniques to problems encountered in Project Monotas is likely to be somewhat experimental. The beginning of Section 3 highlighted the difficulty in finding a good optimisation technique for any given problem, and considered it to be a 'hit and miss' process. There are also concerns about the number of iterations each algorithm may take when finding good settings for the network, especially if the optimisation process is required to update the network's parameters in a fixed time step. Selecting a good adaptive control scheme is also subject to the same 'hit and miss' process, where it is unknown whether a controller may adapt well or not given a particular system to control. Other difficulties with adaptive and non-adaptive types of control are associated with defining the models of the system the controller is meant to control. These models, used by some control theories, enable controllers to predict the stability of the overall system, which in turn constrains and defines their own outputs. Inaccuracies in the estimation of a system model, which to a certain extent may be countered by robust methods, could cause the resulting controllers to behave in an unstable manner.

Finally, whether an optimisation scheme or a real-time controller of some description is selected for solving problems within Project Monotas, some aspect of configuration of the network will be ceded to an automatic process. To prevent failure, these automations will need limits placed upon them, whose appropriate settings will also need to be decided. It will also be important that the performance criteria of the schemes be set so that they may be evaluated. It is unlikely that any scheme selected from this report will produce perfect performance. Therefore, understanding the failure modes, the variations in performance and the possible gains will enable the process of adopting a chosen scheme to be monitored. The potential benefits to computing 'optimal' or better settings for the network could be significant, and will hopefully allow an improvement to network conditions that could not normally be achieved. The use of real-time controllers within the network environment also allows optimisations that could not currently be performed by an engineer, and may help the network to tailor itself to the immediate demands imposed upon it.

References

1 Wolpert, D.H., and Macready, W.G. “No Free Lunch Theorems for Optimization”, IEEE Transactions on Evolutionary Computation, Vol. 1, pp. 67-82, 1997.

2 Plambeck, E. L., Fu, B. R., Robinson, S., and Suri, R. “Throughput optimization in tandem production lines via nonsmooth programming”, Proceedings of the 1993 Summer Computer Simulation Conference, pp.70–75, San Diego, CA, 1993.

3 Plambeck, E. L., Fu, B. R., Robinson, S., and Suri, R. “Sample-path optimization of convex stochastic performance functions”, Mathematical Programming, Vol. 75, pp.137–176, 1996.

4 Robinson, S.M. ”Analysis of sample-path optimization”, Mathematics of Operations Research, Vol. 21, pp.513–528, 1996.

5 Ferris, M.C., Munson, T.S., and Sinapiromsaran, K. “A Practical Approach to Sample-Path Simulation”, Proceedings of the 2000 Winter Simulation Conference, pp.795-804, 2000.

6 Ho, Y.C., Sreenivas, R., Vakili, P. "Ordinal Optimization of Discrete Event Dynamic Systems", Journal of Discrete Event Dynamic Systems 2(2), pp 61-88, 1992.

7 Ho, Y.C. "Heuristics, Rules of Thumb, and the 80/20 Proposition", IEEE Trans. on Automatic Control, Vol. 39, No.5, 1025-1027, May 1994.

8 Lau, T.W.E. and Ho, Y.C. "Universal Alignment Probabilities and Subset Selection for Ordinal Optimization", Journal of Optimisation Theory and Applications, Vol.39, No.3, pp 455-490, June 1997.

9 Lin, S.Y., and Ho, Y.C. "Universal Alignment Probability Revisited", Journal of Optimization Theory and Applications, Vol.113, No.2, pp.399-407, May 2002.

10 Ho, Y.C. "A New Paradigm for Stochastic Optimization and Parallel Simulation", Proceedings of 1993 DES Workshop in IMA/U.Minn Lecture Notes Series, Springer-Verlag, (1994)

11 Dai, L., and Chen, C.-H. "Rates of convergence of ordinal comparison for dependent discrete event dynamic systems", Journal of Optimization Theory and Applications, Vol. 94, No. 1, July, 1997.

12 Li, D., Lee, L.H., and Ho, Y.C., "Constraint ordinal optimization", Information Sciences, Vol.148, pp.201-220, 2002.

13 Zhao, Q.C., Ho, Y.C., and Jia, Q.S. "Vector ordinal optimization," Journal of Optimization Theory and Applications, Vol. 125, No. 2, pp. 259-274, May 2005.

14 Spall, J.C. “An Overview of the Simultaneous Perturbation Method for Efficient Optimization”, Johns Hopkins APL Technical Digest, vol. 19, pp. 482–492, 1998.

15 Spall, J.C. “Introduction to Stochastic Search and Optimization: Estimation, Simulation, and Control”, Wiley, Hoboken, NJ, 2003.

16 “http://www.jhuapl.edu/SPSA/ “ A Website dedicated to Simultaneous Perturbation Stochastic Approximation.

17 April, J., Glover, F., Kelly, J.P., and Laguna, M. “Practical Introduction to Simulation Optimization”, Proceedings of the Winter Simulation Conference, 2002.

18 Carraghan, R., and Pardalos, P. “An exact algorithm for the maximum clique problem.”, Operations Research Letters 9, pp.375-382, 1990.

19 Pardalos, P.M., Rappe, J., and Resende, M.G.C. “An Exact Parallel Algorithm for the Maximum Clique Problem”, in High Performance Algorithms and Software in Nonlinear Optimization, De Leone, R., Murl'i, A., Pardalos, P.M., and Toraldo, G. (eds.), Kluwer, pp.279-300, Dordrecht, 1998.

20 Gomez, F.J., and Miikkulainen, R., “Transfer of Neuroevolved Controllers in Unstable Domains”, Proceedings of Genetic and Evolutionary Computation Conference, 2004.

21 Mataric, M.J., and Cliff, D. "Challenges In Evolving Controllers for Physical Robots", Evolutional Robotics, special issue of Robotics and Autonomous Systems, Vol.19, No.1, pp.67-83, October 1996.

22 Nelder, J.A., and Mead, R. “A simplex method for function minimization”, Computer Journal 7, pp. 308–313, 1965.

23 Lagarias, J.C., Reeds, J.A., Wright, M.H., and Wright, P.E. “Convergence Properties of the Nelder–Mead Simplex Method in Low Dimensions”, Society for Industrial and Applied Mathematics Journal on Optimisation, Vol. 9, No. 1, pp. 112–147, 1998.

24 R. Hooke and T. A. Jeeves, "Direct Search Solution of Numerical and Statistical Problems", Journal of the ACM, Vol. 8, pp. 212-229, April 1961.

25 Kaupe, A.F. (Jr.), "Algorithm 178: Direct Search", Communications of the ACM, Vol. 6, pp.313-314, June 1963.

26 Olafsson, S. “Iterative Ranking-and-Selection for large-scale optimisation”, Proceedings of the 1999 Winter Simulation Conference, pp. 479-485, 1999.

27 Goldsman, D., and Nelson, B.L. “Statistical Screening, Selection, and Multiple Comparison Procedures in Computer Simulation”, Proceedings of the 1998 Winter Simulation Conference, pp 159-166, 1998.

28 Glover, F., and Laguna, M. “Tabu search”, in Modern Heuristic Techniques for Combinatorial Problems” C. Reeves (ed.), Blackwell, Oxford, UK; pp.70-141, 1993.

29 Glover, F. “Tabu search: Part 1”, ORSA Journal on Computing, Vol.1, Number 3, pp.190-206, 1989.

30 Glover, F. “Tabu search - Part II”, ORSA Journal on Computing, Vol. 2, pp.4-32, 1990.

31 Hertz, A., Taillard, E., and de Werra, D. “A Tutorial On Tabu Search”, Proceedings of Giornate di Lavoro AIRO'95 (Enterprise Systems: Management of Technological and Organizational Changes), 1995.

32 Kirkpatrick, S., Gelatt, C.D., and Vecchi, M.P. “Optimization by Simulated Annealing”, Science, No 220, pp.671-680, 1983.

33 Back, Fogel, Machalewicz. “Evolutionary Computation I and II.”, Institute of Physics Publishing, Breiman,1999.

34 Rechenberg, I. "Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution", Stuttgart: Fromman-Holzboog, 1973.

35 Schwefel, H.-P. "Numerische Optimierung von Computermodellen mittels der Evolutionsstrategie", Basel: Birkhaeuser, 1977.

36 Kursawe F. “Evolution strategies for vector optimization”, Proceedings of the Tenth International Conference on Multiple Criteria Decision Making, Taipei 1924. 07, Vol.1, pp.187-193, 1992.

37 Kursawe, F. "Evolution strategies: Simple models of natural processes?", Revue Internationale de Systemique, France, 1994.

38 Goldberg, D. E. “Genetic algorithms in search, optimization, and machine learning.” Addison-Wesley, 1989.

39 Beasley, D., Bull, D.R., and Martin, R.R. "An Overview of Genetic Algorithms: Part I, Fundamentals", University Computing, Vol.15, No.2, pp.58-69, 1993.

40 Beasley, D., Bull, D.R., and Martin, R.R. “An Overview of Genetic Algorithms: Part 2”, Research Topics, University Computing, 1993.

41 Shi, L., and Olafsson, S. “An Integrated Framework for Deterministic and Stochastic Optimisation”, Proceedings of the 1997 Winter Simulation Conference, pp.358-365, 1997.

42 Nikolaou, M., "Model Predictive Controllers: A Critical Synthesis of Theory and Industrial Needs", Advances in Chemical Engineering Series, Academic Press, 2001.

43 Kerrigan, E.C., and Maciejowski, J.M. "Robust Feasibility in Model Predictive Control: Necessary and Sufficient Conditions", 40th IEEE Conference on Decision and Control, Orlando FL, pp.728-733, December 2001.

44 Tesauro, G. “Practical issues in temporal difference learning.”, Machine Learning, Vol.8, pp.257-277, 1992.

45 Tesauro, G. “TD-Gammon, a self-teaching backgammon program, achieves master-level play.”, Neural Computation, Vol.6, No.2, pp.215-219, 1994.

46 Tesauro, G. “Temporal difference learning and TD-Gammon.”, Communications of the ACM, Vol.38, No.3, pp.58-67, March 1995.

47 Samuel, A.L. “Some studies in machine learning using the game of checkers.” IBM Journal of Research and Development, 3:211-229, 1959. Reprinted in E. A. Feigenbaum and J. Feldman, editors, Computers and Thought, McGraw-Hill, New York 1963.

48 Moore, K.L. “Iterative Learning Control for Deterministic Systems”, Springer-Verlag Series on Advances in Industrial Control, Springer-Verlag, London, January 1993.

49 Koza, J.R. “Genetic Programming: On the Programming of Computers by means of Natural Selection.”, MIT Press, ISBN 0-262-11170-5.

50 Gurney K. “An Introduction to Neural Networks”, UCL Press, ISBN 1 85728 503 4, 1997.

51 Haykin S. “Neural Networks”, 2nd Edition, Prentice Hall, ISBN 0 13 273350 1, 1999.

52 Puterman, M. L. “Markov Decision Processes.” Wiley, 1994.

53 Kachroo, P., Shukla, P.K., Erbes, T., and Patel, H. “Stochastic Learning Feedback Hybrid Automata for Power Management in Embedded Systems” IEEE International Workshop on Soft Computing in Industrial Applications, Binghamton University, Binghamton, New York, June 2003.

54 Crites, R.H., and Barto, A.G. “Improving Elevator Performance using Reinforcement Learning”, Advances in Neural Information Processing Systems, MIT Press, 1996.

55 Mitchell, T. “Machine Learning”, McGraw Hill. ISBN 0070428077, 1997.

56 Hung, C.-C., Coleman, T.L., and Long, O. “Supervised and Unsupervised neural models for multispectral image classification.”, Proceedings XXth International Society for Photogrammetry and Remote Sensing Congress, Istanbul, Turkey, 12-23 July 2004.

57 Höglund, A.J., and Hätönen, K. “Computer Network User Behavior Visualization using Self-Organizing Maps”, Proceedings International Conference on Artificial Neural Networks (ICANN), Vol. 2, pp. 899-904, 1998.

58 Laiho, J., Kylväjä, M. and Höglund, A. "Utilization of Advanced Analysis Methods in UMTS Networks", Proceedings 55th IEEE Vehicular Technology Conference (VTC) Spring, vol. 2, pp. 726 –730, 2002.

59 Höglund, A.J., Hätönen, K., and Sorvari, A.S. “A computer host-based user anomaly detection system using the self-organizing map”, Proceedings IEEE-INNS-ENNS International Joint Conference on Neural Networks (IJCNN), vol. 5, pp. 411-416, 2000.

60 Conradie, A.v.E., Miikkulainen, R., and Aldrich C. “Adaptive Control utilising Neural Swarming.”, Proceedings of the Genetic and Evolutionary Computation Conference, New York, USA. Adaptive Control utilising Neural Swarming , 2002.

61 Stanley, K.O., and Miikkulainen, R. “Evolving Neural Networks Through Augmenting Topologies”, Evolutionary Computation, Vol. 10, No. 2, pp.99-127, 2002..