An Algorithm for Evolving Protocol Constraints

Mark Collins

Doctor of Philosophy

Artificial Intelligence Applications Institute
School of Informatics
University of Edinburgh

2006

Abstract

We present an investigation into the design of an evolutionary mechanism for multi-agent protocol constraint optimisation. Starting with a review of common population based mechanisms we discuss the properties of the mechanisms used by these search methods. We derive a novel algorithm for optimisation of vectors of real numbers and empirically validate the efficacy of the design by comparing against well known results from the literature. We discuss the application of an optimiser to a novel problem and remark upon the relevance of the no free lunch theorem. We show the relative performance of the optimiser is strong and publish details of a new best result for the Keane optimisation problem. We apply the final algorithm to the multi-agent protocol optimisation problem and show the design process was successful.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Mark Collins*)

A testimony to the art of the moving target.

For:

All who loved me, and all I loved.

Disclaimer:

All contents of this work are my own,
rendered to the best of my ability.
Naturally any errors or omissions are also mine.

Mark Collins, 2005.

Table of Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation

This work is motivated by the requirement for a mechanism capable of effectively searching large complicated spaces with very small numbers of sample evaluations. We were obliged to create such a mechanism by the demands of optimising the performance of multi-agent systems. Multi-agent systems are increasingly common, but automatic parameterisation of the systems is technically challenging. This work addresses the causes of these challenges, and describes a development process which uses surrogate evaluation functions to create an algorithm that is capable of optimising multi-agent systems.

1.1.1 Automation of design

The design of complex systems is increasingly being automated. Of the multitude of motives for this, the cheap availability of massive brute-force computing power has been the primary development. Computers are now powerful enough to simulate large complex systems with reasonable accuracy. This ability to simulate complex systems, along with the parallel development of optimisation techniques which are capable of giving human competitive performance, set the stage for the automation of complex system design. However, the replacement of human system designers in optimisation tasks has merely moved the bottleneck in our understanding. Our previous ignorance of how to solve optimisation problems has now been replaced by our ignorance of how to create mechanisms to solve optimisation problems.

Racing to fill this void in our knowledge are a plethora of proposed techniques.

Many proposed optimisers are population based and are loose analogues of natural systems : Ant Colony Optimisation [19], Particle Swarm Optimisation [50] and Genetic Algorithms [40, 42] to name a few. Each of these has been extensively tested and offers good performance on some hard problems. Unfortunately the development of these systems is somewhat complicated by a scarcity of people who understand how they work well enough to optimise them. The paucity of explanatory knowledge in this field is best exemplified by the huge number of papers that offer new parameter tweaks and new operators for algorithms without justification for why the operators are employed or why they are effective or on which problems. Commonly a field fragments and subdivides in the pursuit of better and better optimisers on smaller and smaller ranges of problems. The subdivision of real number optimisation into linear and non-linear (constrained) optimisation is a classic example, where algorithms are not expected to perform strongly on both fields and so become segregated.

This work aims to contribute to the understanding in the field of real number optimisation. We wish to avoid producing yet another work on "The optimal parameterising of algorithm X" where we might publish one-off results that are not scientifically useful. Instead we wish to define our work in relation to the properties of a problem that it is designed to engage with, where, if we are successful, further development will build upon this design and increase our understanding of the consequences of our design decisions. Consequently the heart of this work is structured slightly differently from others: We begin by reviewing the properties of notable algorithms and their relative performances on common test functions. The main properties of the common designs are selected and used to guide design of an algorithm that has better properties overall than each of its components has in isolation. We then use this algorithm to optimise both common benchmark problems and more obscure hard design problems from the multi-agent literature. We show that the algorithm performs well over a range of functions without parameterisation. From these comparisons we conclude that the algorithm development has been successful and the design is likely to be a good choice for problems within this class. Finally we invite further analysis.

1.2 Hard problems

In general, hard optimisation problems can be classified as having at least one of two properties; those problems that are hard because the space is complicated and confusing to search, and those problems that for reasons of evaluation cost or other limits al-

low a very low number of samples to be taken from the space. The distinction between the two cases is founded on the amount of information that is available to work with. The information obtained through sampling may reveal a very complicated structure, in which case the difficulty of searching the space is located in correctly interpreting this complicated structure. Alternatively the number of samples available for use may be insufficient to reveal the true structure at all. The difficulty in this instance may be compounded two fold, in arranging the samples so as to obtain the best possible representation of the structure and then in interpreting the revealed structure, which may in addition be complicated.

In the first case, sampling can tell quite a lot about the space. It is possible, locally at least, to map the space and make informed decisions. Performing well on these spaces is a matter of balancing exploitation and exploration - attempting to not be deluded by the eccentricities of the space. The majority of common benchmarking problems are in this class.

In the second case, the complexity of the space is typically unknown. The algorithm must do the best it can with the few samples it has. The second case is the harder of the two since the problem may also be a member of the first class. In the few samples that are available, it is frequently impossible to characterise the space that is being searched. This presents a challenge to a search algorithm. Multi-agent systems, like the majority of large scale optimisations, are in this second class. We will return to this discussion in chapter 2.

Never unduly discouraged, in this work we attack both classes simultaneously. We aim to create an effective optimiser for specific multi-agent system optimisation problems. We use examples from the first class of problems as surrogate evaluation functions to create an algorithm that is effective when applied to problems from the second class. The motivation for this is simple: The use of surrogate functions dramatically reduces the development time of the algorithm. This tactic also carries associated risks requiring that precautions be taken to identify and avoid common mistakes in design. The use of surrogate evaluation functions in the design also entails a potential conflict with the No Free Lunch theorem, which we avoid falling foul of in chapter 5.

Successfully using surrogate evaluation functions requires careful thought. The chosen development process is not easy. Competitive performance in the optimisation of the surrogate functions, which are drawn from the benchmark functions used by the real number optimisation community, is a hard task in its own right. However, once we have succeeded in overcoming these obstacles we will then have developed a much

stronger optimiser, in a much shorter time, than we would have had by attempting to optimise the multi-agent problem directly.

1.2.1 Benchmarking functions

Due to the high profile and ubiquity of benchmarking functions, there are many papers published in the field of real number optimisation which report improvements against benchmarks. Obtaining good results in the field is a commendable achievement and is vigorously pursued. It should be mentioned, that not all improvements in the field are equally meritorious. Some algorithms, particularly if the chosen set of benchmarks is small, are over-fitted. Over-fitting occurs when the solver has become unacceptably specialised and performs well only on the test cases. Over-fitting makes the discovery of a better result somewhat easier but reduces its general utility. In this work we compare and contrast the performance of our algorithm against the best found results published by researchers in other domains. We use the benchmarking functions to illustrate various complexities faced when optimising in these complicated spaces and also use them to warn against the dangers of over-fitting by providing an example of a "best-ever" algorithm that does nothing more than use an exploit against the properties of the solution space.

1.2.2 Multi-Agent systems

A multi-agent system is any collection of autonomous agents capable of reasoning about their environment and acting accordingly. A designer of a multi-agent system attempts to exert control over the interactions of a potentially enormous heterogeneous system, one where autonomous agents are employed on behalf of different users to achieve a variety of private objectives. The design of multi-agent systems is an active research topic. Understanding such systems is complicated by the inherent dynamics of the participating agents - trust and deceit, agent overloading and failure, imperfect information etc. Despite the problems, engineering and applying effective multi-agent systems has significant promise and economic consequence. Multi-agent systems (and implicitly their designers) perform a role that is likely to increase in importance in the future.

Unfortunately the complexity of understanding and controlling the behaviours of multi-agent systems increases as the participating agents grow in number, size, scope, and responsibility. Engineering agents to participate in a large scale system involv-

ing heterogeneous autonomous agents engaged in achieving undisclosed objectives is in itself a hard problem. The search space is, for all but the most trivial of systems a very large space, and search for optimal parameters necessarily involves evaluating and omitting sections of the search space. The complexity of performing the generalisations required to omit sections of the search space is greatly increased by the possiblity of non-trivial interactions between parameters, requiring that samples be invested to test proposed configurations of multi-agent systems. Unfortunately detailed simulation of a multi-agent system is also extremely expensive in terms of computation time. Thus searching for good configurations of multi-agent systems is a hard problem that fits both of the criteria specified in section 1.2.

1.2.3 Protocols and multi-agent system design

In this work we use evolutionary search techniques to optimise an interaction protocol with regard to the emergent behaviour of simulated multi-agent systems. Lightweight Coordination Calculus (LCC) protocols [70] are a method which simplifies the agent design problem by ignoring the agents themselves and concentrating on defining and controlling the interactions between participating entities. There are several good reasons why the protocol based approach is of interest, and these will be expounded later.

The multi-agent systems community has traditionally examined the problem of developing complex systems through the eyes of an intimate designer, one who knows the specifications and capabilities of each of the agents in detail.

Frequently this system design decision leads to the bespoke development of special agents with characteristics and behaviours which are unique within the multi-agent system. System efficiency is attained by maximally exploiting the unique characteristics of the agents. Even without bespoke agents, exploiting individual agents is a popular design strategy e.g. [80]. Due to the significance of individual agents, large systems constructed in this manner are difficult to predict, particularly in the event of dynamic agent participation. Maintenance and design of such systems is knowledge intensive and complicated. The unpredictable interactions between the agent's unique behaviours make the design of such systems frustrating to automate.

An alternative design strategy [70] is to abstract the agent design to a specification of the communication behaviours which an agent must implement to be accepted as part of the system. The exact implementation of the agent is not critically important to the functionality of the system. By abstracting away agent specific details, the system

can be shown to have particular properties irrespective of the composition of the agent collective which composes the system. Agents may join and leave a system at any time for a multitude of reasons; completion of their objective, change in interests/objectives of user, introduction/referral by another agent, resource bounds and opportunistic exploitation by other agents, activity/availability cycles, communication noise/failure, agent/system failure, etc. The agents which remain actively participating in the system may change capabilities themselves over the duration of the interaction and the role of the system itself may also be modified.

Clearly to be able to predict the performance of multi-agent systems in the face of highly variable circumstances is extremely valuable. In this work we extend on work by [70] which shows how a basic protocol can be declared which guarantees a basic behaviour of the system if all participating agents in the system interact through the protocol. We then show how modification of the protocol to include constraints on the interactions between participants can modify the behaviour of the entire multi-agent system. Finally we show how evolutionary techniques can be used to search the space of possible constraints to optimise the performance of the system with regard to a set of performance measures and show how the quality of the solution resists degradation in spite of changes in the agent performance, number and capabilities.

1.3 Contribution of this work

This work contributes to the fields of both evolutionary algorithms and multi-agent systems. Succinctly it is characterised as “a development of an evolutionary algorithm that has been applied to the problem of multi-agent design”.

We discuss the applicability and robustness of evolutionary algorithms and their application to the multi-agent design problem, We contribute to the field of evolutionary algorithm design by the introduction of a new system which exploits developments in sampling methodology. We justify the design of the algorithm and give results against well known benchmarks. We contribute to the field of multi-agent systems by providing a further step in the automatic optimisation of protocol specifications. We provide empirical evidence for the consistent robustness of solutions thus derived.

We also make secondary contributions by publishing a new result for the Keane function, and providing an argument for why the no free lunch theorem need not prohibit improvement of optimisation methods.

1.4 Synopsis

Chapter 1 is this chapter, and is an overview of the work. Those chapters that remain are laid out as follows.

Chapter 2 introduces the topics of protocols and multi-agent systems. These topics form the backbone of the thesis and define the problems that will form the basis of the algorithm development.

We introduce the reader to the complexities of real number optimisation in chapter 3. We give examples of the difficulties faced by optimisers solving real number optimisation problems, and review some of the more common features of such problems with the aid of two examples taken from the literature

We then review some of the common forms and motivating principles behind modern evolutionary and population based algorithms in chapter 4. We highlight the aspects of their design which we believe contribute to their success, and we consider what lessons the relative performance of the algorithms might have for an algorithm designer.

The limits of algorithm design are considered when we review the no free lunch theorem in chapter 5. Understanding of the no free lunch theorem and its implications provides the evaluation context against which the successes of this thesis should be measured.

The detail of our algorithm design is found in chapter 6 which builds on the discussion from chapter 4 to elaborate the objectives of the design and the mechanisms which we employ to achieve them.

We discuss common benchmarking functions from the literature in chapter 7, and discuss the properties of the algorithms and best results by notable authors.

We then validate the performance of our algorithm in chapter 8 by comparison against the results detailed in chapter 7 and discuss the evidence for adequacy of the algorithm.

In chapter 9 we present the results of applying the developed algorithm to the example multi-agent problem we introduced in chapter 2. We discuss the performance of the algorithm on the multi-agent problem and the quality of the results found.

We conclude this work with a discussion of the achievements and possible directions for future work in chapter 10.

Chapter 2

Multi-agent systems

This work is focused on creating a search algorithm that can optimise protocol based multi-agent systems. The simulation of multi-agent systems is very time consuming, making the use of simulated multi-agent systems in the design process impractical. For instance the simulation of a single detailed multi-agent system of 256 agents over 1000 simulated interaction timesteps took minutes to perform. Obviously, to perform search over the space of possible optimisers requires performing sufficient tests to provide an adequate guarantee that the results are not subject to excessive noise. The maximum number of tests that can be performed is severely constrained by the disposable computing power.

Instead we turn to surrogate evaluation functions to permit rapid development of the search algorithm. In pursuing this development strategy we address the concerns raised by the No Free Lunch theorem. Once satisfied that the algorithm is functioning correctly, we then apply the prototype algorithm to the optimisation of the multi-agent systems.

2.1 Overview

This chapter describes the multi-agent systems which form the context for the later developments in this work. We describe the role of protocols in controlling multi-agent systems, the properties of multi-agent systems, the motivation for using protocols and the limitations and capabilities that the use of protocols implies. We provide a proof of concept system as evidence that constraints within protocols can be used to manipulate the behaviour of a multi-agent system, and we then prove the scalability of the approach by extending the system to a problem from the literature.

Fragments of this chapter, particularly the text on the proof of concept problem are based, in part, on work published in the AMEC VII workshop in Utrecht, part of AAMAS 2005.

2.2 Multi-agent systems

"Multi-agent systems" is a loose term, used to denote processes that are characterised by the participation of collections of interacting autonomous agents. In common usage it is implied, but not required, that the interacting agents are synthetic. Each agent is empowered to make autonomous decisions and acts and reacts in its environment in accordance with obtaining its objective. The purpose of design in multi-agent systems is to create a system which can facilitate a user in obtaining an objective through the use of a multi-agent system.

At the risk of understating the task, the multi-agent system design problem is complicated. The objective of the system is typically an emergent property of the interactions of all participating agents. To make matters worse the agents engaged in the system are not necessarily cooperative, nor do they necessarily receive the same information. Agents do not normally have the same capabilities, nor do they usually possess the same resources. In fact, multi-agent systems may be extremely dynamic, agents may join and leave the system, they may malfunction or even be malicious and the communication between agents is subject to delays and failures.

Initial approaches to multi-agent system design have been focused on the design of agents themselves. By equipping agents with certain behaviours, preferences and reasoning systems, the architecture of the whole system can be shaped. This system works well for small groups of agents where the system dynamics may be relatively easily predicted. Unfortunately the complexity of the interactions between agents rapidly grows as the system scales, and the interaction of agents developed by different design stables is, to say the least, unpredictable. The agent-centric design method places a heavy burden on the human designers who build, deploy and then maintain the multi-agent systems. The complexity of the agent orientated design process, and the difficulty of maintaining deployed systems is one reason for the scarcity and simplicity of systems currently deployed.

It is apparent that scaling the traditional design methodology to large open real world communities of heterogeneous agents is going to be a challenge. Reasoning about and predicting the consequences of interactions in heterogeneous multi-agent

systems is inherently complicated, but the designer of a multi-agent system faces further complications and restrictions.

Firstly, it is unreasonable to expect owners of currently active and deployed agents (which may use sensitive information or have been configured for multiple behaviours) to permit another designer access to each agent's configuration [7].

Secondly, even if full access to agents were granted, agents are likely to be multi-roled and currently participating in several endeavours. It is optimistic to expect system designers to have sufficient understanding of the roles and mechanisms used by the agents to be able to modify an agent's behaviour without error or compromising the agent's behaviour in other roles. This is particularly true if the agent's configuration is the product of an non-intuitive design process like evolution.

Certainly, for systems involving numerous agents, it is unrealistic to expect designers to be able to choose the best modifications to make to a subset of participating agents in order to achieve the dual objectives of optimisation of system performance and minimising interference and knock-on consequences.

Instead, if we wish to have these capabilities, we should consider other methods for defining, designing and modifying multi-agent systems. We must use design methods in which the validity of the design is not dependent on the componentry of the individual agents. This necessitates a change of design focus from the control of agents themselves to the control of the interaction between agents. One method of achieving this abstraction is by specifying system interactions through protocols.

2.2.1 The LCC

The Lightweight Coordination Calculus (LCC) [69] is a logic programming language for the process calculus. A set of behavioural clauses specified in the LCC syntax is sufficient to define the relevant message passing behaviour expected of the implementing agents. Agents action changes in their environment by direct action and request other agents to perform tasks by message passing. Constraints may be imposed on the messages that agents send and receive. The overall system behaviour is an emergent property of the individual agent actions and the inter-agent behaviours defined by the collection of message passing clauses.

The main advantage of pursuing a protocol based approach rather than an agent design based approach is that it changes the emphasis of the design from focusing on the capabilities of individual agents to the capabilities of the agent interactions. This

frees the designer from having to consider the peculiarities of the individual agents in the population, making the quality of the design independent of properties of the agents themselves. Consequently the design is considerably more robust to changes in the constitution of the agent population.

The agent-independent robustness of protocol based multi-agent system design allows the use of results obtained by simulation of protocol based multi-agent systems to be used in qualitative comparisons without concern that the parameterisation is exploiting unique properties of the agent's internal architecture. Qualitative comparison of simulated systems allows search to occur; we have the necessary facilities to perform automated optimisation of chosen properties.

Figure 2.1 shows the syntax of the LCC. The author is grateful to Dave Robertson for providing the definition. Figure 2.1 describes the message passing events that are possible in the LCC syntax, where *null* denotes an event which does not involve message passing, e.g. an agent changing role subject to some constraint *Cons*. *Term* is a structured term in Prolog syntax and *Id* is either a variable or a unique identifier for the agent. $M \Rightarrow Agent \leftarrow Cons$, should be read as the agent role being defined will send a message *M* to another agent *Agent* subject to constraints *Cons*, $Cons \leftarrow M \Leftarrow Agent$ means the agent being defined receives a message *M* from an agent *Agent* also subject to constraints. Further, the symbols *then* and *or* mean: the following action follows as a consequence of the first or there is a disjunction of actions. Figure 2.1 shows only the basic syntax; LCC is a more powerful system than we really use in this work. LCC also defines additional symbols that are not relevant to our discussion, e.g. instructions for parallelisation of execution.

Although LCC is the basis of the multi-agent system descriptions, the choice of LCC is not significant to this work. Using the constraint mechanism provided by LCC has benefits, but any mechanism capable of abstracting agents sufficiently to allow qualitative comparison of multi-agent system configurations would have been sufficient. As a consequence of the choice of LCC as the control medium, the focus of this work is on protocol based interaction. This work investigates the potential of automatically designing robust multi-agent systems to exhibit desirable emergent behaviour through the control mechanism of constraints applied to an agreed protocol.

$$
\begin{aligned}
Framework &:= \{Clause, \ldots\} \\
Clause &:= Agent :: Dn \\
Agent &:= a(Type, Id) \\
Dn &:= Agent \mid Message \mid Dn\ then\ Dn \mid Dn\ or\ Dn \\
&\quad \mid null \leftarrow Cons \\
Message &:= M \Rightarrow Agent \mid M \Rightarrow Agent \leftarrow Cons \\
&\quad \mid M \Leftarrow Agent \mid Cons \leftarrow M \Leftarrow Agent \\
Cons &:= Term \mid Cons \wedge Cons \mid Cons \vee Cons \\
Type &:= Term \\
M &:= Term
\end{aligned}
$$

Figure 2.1: Syntax of LCC dialogue framework, see section 2.5 for an example agent system defined in this syntax.

2.3 Scope of control

In this work we will use the LCC protocol specification [69, 54, 55] to provide the tools for design at the interaction level. The LCC protocol specification permits detailed control of the roles and obligations, but, unlike agent centric design methods, does not normally extend to specifying precise roles for particular agents.

This choice raises the question of whether or not the mechanisms available through the use of protocols are powerful enough to produce satisfactory optimisations of interesting systems without exploiting unique agent properties. We answer this question in two parts, initially using a small system to prove the feasibility of message passing and constraints as a control mechanism before turning to the literature for an interesting large problem. The computational burden of simulating genuine message passing is great. In order to be able to simulate the large problem with the available facilities, we abstract the message passing mechanism once it has been shown to be effective in the proof of concept system.

We prove the concept of using constraints for optimisation of protocols by creating a simple fully fledged trading system which fully implements a simple negotiation and exchange protocol. Using this small problem we show how the standard features of LCC allow for optimisation of the behaviour of the multi-agent system using protocols. To do this a basic protocol will be elaborated by the addition of constraints on when certain messages are passed. These constraints effectively limit agent behaviour

without penetrating the layer of abstraction, which makes predicting the consequences of such additions feasible. We then demonstrate optimisation of the system by manipulating the constraints to optimise the system for high trade volumes.

Having proven the basic concept, and demonstrated optimisation through manipulation of protocol constraints we extend our experiments and apply the same approach to a known problem from the literature. We have chosen to recreate the multi-agent system from the SADDE work published by Sierra et al [80]. Having reimplemented the problem from the literature, the second stage is to convert the original problem representation to one that is capable of being controlled by a suitable protocol based form. Genuine message generation, passing, and parsing is expensive to simulate. To increase the simulation speed of the SADDE system, we do not use a real string based message passing and parsing protocol in the actual simulation of the SADDE inspired system. Instead we simply achieve the necessary effects by intervention in the agents' state. This allows us to simulate a system that is larger and more complicated than that which we would have had resources for otherwise. To ensure we remain within the remit of protocol based manipulation of the agent behaviours, we modify only role based behaviours using constraints, as proven effective in the proof of concept protocol.

2.4 Protocol optimisation

A protocol specifies particular properties regarding an agent's obligations in a particular role during a transaction between agents. Constraints may be placed on the protocol to alter behaviour, usually to keep the system within consistent boundaries. An example of such a constraint is requiring successive bids in an English or Vickrey auction to be ascending in value. This can be simply encoded by the extension of a functional auction protocol with a constraint that any returning offer must be greater in value than the previous offer.

Constraints also offer a sensible place within the protocol framework to place encodings of relationships that are uncertain or likely to alter. Retraction and revision of a complete protocol is complicated. Agents may not have implemented the facilities to meet the requirements of a new protocol.

An agent using a protocol has to be capable of a certain calculus – of manipulating certain values as specified by the protocol. As long as the modifications to the protocol remain within the limits of this calculus modifications may be made freely without alienating participating agents. Such changes are "safe", at least in terms of contin-

uing participation. Agents are free to decline to participate under the new protocol specification. Agent autonomy is not compromised by such changes.

Safe modifications are easily identified. Within a single scenario, such as an auction, there is a minimum well defined vocabulary which must be understood by all participating agents. For instance, if it might be known that an agent manipulates free variables that represent the price and quantity of some goods. If so, then it is "safe" to define constraint based protocol modifications relative to these terms. The safety comes from the fact that it is known that all participating agents must have some representation of these quantities. To successfully participate in a protocol based exchange, agents which used these terms in negotiations must also have standardised their denominations. It would be "unsafe" to specify a protocol modification based on a concept that was not guaranteed to be well defined amongst all the participating agents. Using an unsafe modification would risk alienating any currently participating agents who were unable to parse the new protocol.

Constraints allow a system designer safe and easy access to variables that indirectly alter agent behaviours. A designer may use this access to the constrained aspects of a protocol definition to modify the multi-agent system to maximise user specified objectives. The designer is not capable of directly influencing agent behaviours through a protocol specification, for instance the participating agents may default and withdraw from the system, thus compelling a particular agent to perform a particular action is not within the system designer's remit. This has the consequence that the designer is not able to manipulate the agents themselves into satisfying the required system dynamics. Instead a designer alters the specification of agent roles, and by implication the scope of activities available to any agents which choose to participate in these roles.

In a protocol based multi-agent system the emergent behaviour of the system is not a consequence of the precise characteristics of the participating agents, but is more correctly a consequence of the roles in which the participating agents interact. This approach to multi-agent system design leads to greater robustness against changes in the agent population, since individual agents are not design critical. The increased robustness comes at a cost however, since systems designed in this manner are unable to exploit agent specific properties, and may under-perform relative to a more fragile system that does exploit such properties. See section 3.4.2 for an alternative interpretation of this type of fragility.

Properties desired by the user of the multi-agent system are typically non-trivial emergent properties like "maximise my rate of transactions", or "maximise my profit".

These properties are extremely difficult to determine analytically. Fortunately, though computationally expensive, it is possible to simulate the behaviours of the multi-agent system. The ability to simulate the multi-agent system allows the evaluation of different constraint values, and so we have the ability to search for potential constraints that best satisfy the users objectives. In this work we do not consider the mechanism for translating a user's objective to a set of constraints, instead we focus on the optimisation problem of locating the best set of constraints.

2.4.1 A dynamic optimisation process

The dynamic aspect of optimising multi-agent systems is more complicated than might first appear. Agents are deployed by users seeking to achieve one or more objectives. It is reasonable to expect the users to modify the agents they employ in response to perceived opportunities in the system. It is also reasonable to expect this modification of agent behaviour to provoke counter actions by other participants not wishing to be exploited. In addition the multi-agent systems themselves will be modified by the system architects in response to the need to maximise their objective yield, all of which in turn feeds back into the system and shapes the characteristics of participating agents. The continual evolution of the multi-agent system is something that the system designer should expect and prepare for. Protocols inherently provide for this eventuality by reducing system dependence on agent characteristics and providing tools for cleanly manipulating active systems.

Covering all eventualities by predictive design is difficult. Given the degree to which other users' objectives are unknown, the tactics used by participants are also unknown. Any successful parameterisation of a multi-agent system with such dynamic participants may be transient. Basic behaviours may be ensured by careful protocol design, but more complicated emergent behaviours are not predictable without simulation. Once deployed, a multi-agent system will, in all but the simplest of cases require frequent re-calibration and tuning to maintain the desired behaviour. Part of the motivation for modifying the system is reactive, attempting to keep the system on track in response to changes in the interacting agent community. Part of the motivation for re-calibrating the system is pro-active, as the analysis of the historic data indicates better choices for future interaction models. The system will, in any case, require re-calibration as the user's circumstances and objectives change. The optimisation process is dynamic and ongoing. The tools that are used should facilitate this

process as much as possible.

In this work we are considering the optimisation of a multi-agent system in which the properties of the participants remain consistent throughout the simulation period. Use of interaction protocols gives independence of any particular agent characteristics, however, a significant change in the behaviour or resources of the participating agents could invalidate properties which are important to the optimisation. Continued satisfactory system performance would then require re-calculation of the system parameters. The optimisation of a multi-agent system is not then a one off event with results that persist. Each optimisation of the system parameters is a single step in an on-going cycle. Maintaining satisfactory system performance is a continuous process, in which the participants are both the initiators and enactors of opportunities and are also simultaneously reactive to the actions of other participants or fluctuations in global variables.

2.5 A proof of concept

In the first of two multi-agent system applications we demonstrate the evolution of constraints on a small proof of concept protocol. The objective of the experiment is to demonstrate the use of constraints in a simple maximisation of trade scenario. The aim is to clearly show the mechanism by which the modification of protocol constraints can be used to alter the behaviour of a multi-agent system. The proof of the feasibility of this mechanism for controlling a multi-agent system is a necessary step to justify further development.

Because it is a proof of concept, the primary significance of this work is in demonstrating the constraint mechanism in action, thus in this demonstration there is no significance to the protocol used, nor the objective chosen, both of which are chosen on grounds of simplicity. In spite of the apparent simplicity of the problem, the reader should not be deluded into thinking that searching for emergent properties of a multi-agent system is an easy task. The simulation time severely constrains the number of simulations that can be performed, limiting the number of evaluations that can be made. Optimising an unknown system with very few samples is extremely hard.

2.5.1 Objective of the system

We implement a simple system that has many of the properties of a full multi-agent system, but which has been abstracted of any unnecessary detail. We consider a group of agents involved in a trading system, where some "goods" are bought and sold according to conventional economics. We simulate the producers and end consumers, and concern ourselves with optimising the trading between agents in the middle layer. The details of the monetary transactions are also abstracted, and the traders negotiate only the quantities offered, requested and then exchanged.

The optimisation objective of the system is chosen to be easy to understand: Increasing the volume of trade is beneficial to the user and is rewarded. The problem is complicated by the fact that agents have limited resources and can only store a certain quantity of goods. Agents which have too little or too much are considered to be underutilised or over-loaded respectively and incur penalties. The performance of the system is evaluated by summing the volume traded and any penalties incurred over the entire group.

2.5.2 Implementation of the system

Implementation of the proof of concept system requires a set of agents all of whom are capable of realising a simple trading protocol. The protocol message passing is constrained to achieve control of the system trading behaviour. Optimising the performance of the system with regard to an emergent property demonstrates the plausibility of using a constrained protocol to optimise the system behaviour.

The agents all implement the following simple protocol:

$$
\begin{aligned}
a(node, N) :: \quad & a(supplier, S) \text{ or} \\
& a(buyer, B) \\
a(supplier, S) :: \quad & \text{request}(T_R) \Leftarrow a(buyer, B) \text{ then} \\
& \text{offer}(T_O) \Rightarrow a(buyer, B) \\
& \leftarrow Cons_S(T_R, T_O) \\
a(supplier, T) :: \quad & a(buyer, T) \leftarrow Cons_{\vec{bs}} \\
a(buyer, B) :: \quad & \text{request}(T_R) \Rightarrow a(supplier, S) \\
& \leftarrow Cons_B(T_R) \\
& \text{offer}(T_O) \Leftarrow a(supplier, S) \\
a(buyer, T) :: \quad & a(supplier, T) \leftarrow Cons_{\vec{sb}}
\end{aligned}
$$

An agent which initially starts out in an untyped 'node' role, chooses a role at random. An agent in the buyer role seeks an agent currently in the supplier role using a randomised yellow pages/matchmaker agent. The buyer then requests T_R amount of goods from the supplier. An agent in the buyer role forms its purchase request subject to a buying constraint $Cons_B T_R$. An agent in the supplier role waits for a request for some quantity of goods T_R from an agent in the buyer role and then offers a quantity of goods T_O to the buyer as constrained by the supplying constraint $Cons_S(T_R, T_O)$. Agents may determine their role by deciding whether their current role is justified by the quantity of goods they possess; the protocol has constraints for changing roles from a buyer to a supplier $Cons_{\vec{bs}}$ and from a supplier to a buyer $Cons_{\vec{sb}}$.

Over or under-loading of an agent is calculated relative to an agent's starting load of goods Q_{start} plus or minus a certain margin Q_{margin} and breaches of this margin are penalised on a linear basis. When an agent changes roles from a buyer to a supplier (or vice versa) the services which it had been advertising are no longer available, and new services must be advertised, consequently changing an agent role (e.g. from buyer to supplier or vice versa) is considered undesirable and is also penalised on a linear basis. The setting of constraints on the agent protocol is then a straight forward multi-objective optimisation with the set of optimal choices likely to include near maximal throughput, near minimal overburdening and relatively low frequencies of role change.

2.6 A more advanced multi-agent system

The proof of concept system tests the feasibility of using protocol constraints to alter the behaviour of a multi-agent system. The simplifications made to the system which allow it to be used as a demonstration of protocol constraint based control also result in a system that is too simple to be of interest as a challenging application for evolutionary computation, for this we need a more sophisticated problem.

For a more interesting and complicated multi-agent system with challenging system dynamics, we turn to the SADDE work on evolving multi-agent systems published by Sierra, Robertson and Walton et al. in [80, 81, 79, 82, 88] as part of the SLIE project. We are extremely grateful to Jordi Sabater for providing us with the archived source code from the project. The work is a natural choice for a more sophisticated multi-agent system. It is recent work and apparently well documented with a series of papers describing the application of an evolutionary process to multi-agent system configuration problems derived from the main project. There is also evidence that the project

used vast amounts of processing power to achieve the results published [80]. The use of grid computation and parallelisation of the search process imply the project was only borderline feasible with the processing power available at the time. This makes the project interesting because the optimisation process that was used was too expensive to allow comparison of the quality of different approaches: in such circumstances using an optimiser that is expected to perform well is important.

The purpose of the SLIE/SADDE (Social Agents Design Driven by Equations) work was to describe the implementation of a complete system under the SADDE methodology. The SADDE framework is a design methodology for building a multi-agent system using equation based models. SADDE is a waterfall life cycle consisting of four main stages. The first phase of design in SADDE consists of building an equation based model that describes the desired global behaviour of the system. The next step is to examine what is required of the social interactions between agents to achieve the desired behaviour. The third phase of the SADDE design process is to create agents with the basic rationalities and the necessary awareness required to interact in the environment. In this last stage, agents are provided with internal variables for maintaining references to their current state and negotiation engines to allow them to interact with other agents. The final stage in the SADDE methodology is to create the behavioral configuration for the multi-agent system that extracts from the many potential system configurations the one that most closely matches the original equation based model of behaviour. In the original SADDE methodology a genetic algorithm was used to search for suitable agent centric parameterisations of the multi-agent system. In the SADDE framework the parameterisation was only performed to prove that the model proposed by the equation based model could be realised by an actual multi-agent system. The quality of the parameterisation was not considered a priority in the SADDE work. Any parameterisation that came close to matching the expected behaviour of the equation based model was acceptable.

2.6.1 The basic structure

The multi-agent system described in the SADDE work is more complicated than the proof of concept system in several regards. Firstly, there are now distinct types of agent roles, termed producers, manufacturers and consumers, each of which have differing capacities and capabilities. The agents are arranged in a tiered trading system: producing agents generate and sell a product to manufacturers, manufacturers sell the

product to consumers, and consumers dispose of the product and receive a salary from some external source. The trading system is also more advanced, and agents' negotiations involve both a product and money; agents involved in trading must be aware of both resources if they are to be successful and maximise utility in their trading. Lastly agents have non-linear negotiation behaviours, which may be altered to influence the trading behaviour of the system. Maximising trade by influencing these negotiation behaviours is the objective of the optimisation.

2.6.2 Recreating the SADDE system

We encountered some difficulties with recreating the experiments reported in the SADDE work. There is not enough information available in the reports to precisely recreate the same system that was used in the evaluation of the system. Consequently direct comparison against the results in the literature is unfortunately not possible. Here we describe some of the difficulties we encountered whilst recreating the SADDE system. Despite the difficulties the SADDE system is far from useless to us. We describe the changes we made to the SADDE system in section 2.6.3.

Recreating the multi-agent system described in the SLIE/SADDE literature is complicated by significant differences between the systems described in the various reports. There are significant differences in the flow equations and the parameterisation of the experiments between the reports and the source code. These differences seem to indicate the source code was in the process of being upgraded to support a more advanced system when the project was terminated. Since the flow equations used in the source code are unlike any that are used in published work they can not be used to patch information that is missing from the other reports.

2.6.2.1 Flow equations

The basic flow equations used in the SADDE system are described in [80, 88]. The flow equations should be considered a partial specification. They refer to variables that are not defined (*Cons*), and define equations for variables that are not used ($Delivery_2$). The flow equations also contain inequalities that are never satisfied under normal conditions ($MaxStockOut_i \leq StockOut_i + ProdRate_i$). Harder to guess are the variable ranges that are left undefined for some parameters. In particular the ranges of the parameters t_{max} and β which are used to optimise the system trading behaviour are not clear.

In section 6.1.1. of the annex to [80] a range for t_{max} is given, but the same description refers to variables that are not in the flow equations: ($maxPriceIn, minPriceIn, maxPriceOut$ and $minPriceOut$ which are maximum and minimum prices for buying and selling behaviours for each type of agent).

When interpreting the flow equations certain assumptions must be made to maintain integrity. Additionally certain limits on behaviours are necessary, for instance it is sensible to assume that agents are protected against performing division by zero. It is also sensible to attribute significance to the amount of cash an agent starts the simulation with; that the amount of cash an agent has access to is finite, and that agents may not go into debt. Agents are thus forced to cease buying when they have exhausted their cash reserves.

For clarity and ease of reference when reading this work, we define the meanings of our variables in figure 2.2.

Variable	Meaning
$role \in \{prod, man, cons\}$	Producer, manufacturer or consumer role
$minPriceIn_{role}$	The minimum buying price
$maxPriceIn_{role}$	The maximum buying price
$minPriceOut_{role}$	The minimum selling price
$maxPriceOut_{role}$	The maximum selling price
$maxStockIn_{role}$	The raw material storage capacity
$maxStockOut_{role}$	The refined goods storage capacity
$prodRate_{role}$	The rate at which raw material is made into goods
$maintenance_{role}$	The maintenance costs
$prodCosts_{role}$	The cost of refining one unit of goods
$delay_{role}$	The time taken to refine a unit
$initialCash_{role}$	The initial amount of money
$salary_{role}$	The payment per time unit

Figure 2.2: Extended system parameter meanings.

.

2.6.2.2 The number of negotiations

All the documents describing the SADDE multi-agent system describe a configuration consisting of $A = 60$ agents in total, subdivided into 30 consumers, 20 manufacturers and 10 producers. In [80, 81] the number of negotiations per generation is described as $N \times T \times (\frac{A}{2} + \frac{A}{3})$, where $N = 30$ is the number of individuals in the gene pool, each performing $T = 10$ iterations of the trading routine. The trading routine consists of negotiations between the layers. There are $\frac{A}{2}$ consumers, and every consumer attempts to trade with a manufacturer. There are $\frac{A}{3}$ manufacturers and similarly every manufacturer attempts to trade with a producer. Thus there are 30 genes in a generation, each being simulated for 10 cycles, where each cycle consists of 50 negotiations per cycle, a total of 15,000 negotiation processes per generation.

However, in the same paragraph that describes this evaluation of a generation (section 3.2.3 of [80], section 5.2 of [81]) both the documents describe the system as requiring 24,900 negotiations per cycle. Assuming that the values for N and T are correct, to achieve this number of negotiations in a generation the number of agents in the simulation *A* would have been 100, and not 60 as reported. This would partially explain the considerable effort that was reported to be expended in the evaluation, and the fact that the first values for the plots of total cash in the system on page 23 of [88] are multiples of 100 but not of 60. In this work one of our objectives is to ensure the system works well on different sizes of multi-agent system. Thus we alter the number of agents in our experiments, usually in the range 6 to 192. We retain the ratios between the agent roles.

2.6.2.3 The fitness function

In the documents available, the fitness function used to evaluate the system varies significantly. No description is complete enough to permit its use in a reimplementation. As we shall see, the source SADDE code is also inappropriate for our needs, not least in the fact it describes a system that is not in the published literature. In the SADDE work, the purpose of the genetic algorithm was to validate the design of the multi-agent system by comparing its best performance to that predicted by the equation based model. In all descriptions of the SADDE multi-agent system, the fitness function involve a comparison against expected values obtained from a partially described equation based model; [88] is an overview and does not describe the fitness function at all.

The fitness function described in [82] describes an exponential reward function based on a comparison between the regressions of the equation based model and the actual multi-agent based model. The equation based model is not given. This fitness specification is confirmed when examining the source code, which contains a fitness function that is described for a specific regression value of the cash at the production and manufacturing levels of the system. The fitness function in the source code rewards optimal fitness for cash accumulation regression values of 44.224 and 53.935 respectively. The source code also contains a parameterisation file which defines values for the initialisation and runtime dynamics of the multi-agent system that could have been used to bypass the requirement for a complete specification of the equation based model.

Sadly the parameterisation given in the source code does not create a system ca-

pable of creating the necessary regression values, which, when reimplemented and simulated are missed by some considerable margin. Without a complete description of the initialisation and the system properties that were used to create the expected regression values through the equation based model, the fitness function can not be used. Since we do not have access to the equation based model, we are thus unable to recalculate the equation based model values and are unable to use any of the fitness formulations that are dependent upon a precise regression of the equation based model value. This rules out the recreation of any of the fitness functions from the SADDE documentation or source code.

Instead of directly reimplementing the fitness functions used in the SADDE experiments, we create a new function based on the same principles. All the descriptions of the fitness functions have in common that they are based on the accumulation of wealth in the producers and manufacturers [81, 82, 79, 80] and fitness is attributed by some form of comparison between the simulated rate of accumulation of this wealth and the values predicted by an equation based model. Most descriptions of the fitness functions also award fitness based on the levels of stock, except [82] which considers levels of cash only. In calculating the fitness of the system, the levels of stock which are held by the agents at the various levels of the system are considered explicitly in [81] and implicitly in [80, 79].

A direct interpretation of [81] is that the objective of the multi-agent system is simply to obtain a “moderate linear increase” in the amount of money owned by the producers and manufacturers, and that “there is a positive flow of goods along the chain”. The rationality of the negotiation engine ensures that no agent will sell a good for less than it bought it for, which ensures that the flow is always positive along the chain. Agents pay upkeep costs every iteration of the simulation, additionally producer agents pay a cost to produce a unit of goods. These costs are non-negotiable; agents are obliged to pay these costs. Agents with no cash resources are forced to stop trading. The difficulty of parameterising such a system is not in getting the system to work, which should be an automatic consequence of basic good design, but in getting it to work well.

2.6.2.4 A re-implementation failure

The following discussion refers to the description of the SLIE/SADDE work in [80]; page numbers used in this discussion are relevant to that document. Following the description on pages 13, 14 and 21 [80] gives an implementation of the SADDE multi-

agent system, which when equipped with the negotiation model given on pages 16 and 17 results in a system that is unresponsive to the optimisation parameters. Part of the problem is that the description on page 21 specifies that the simulation of the multi-agent system occurs for only 10 time steps, which is a duration too short to permit significant trading to occur. Correcting this to 500 cycles, a figure supported by the other evidence in the text (page 39) permits enough trading to occur to allow testing of the effects of the parameterisation.

We searched for possible parameterisations using Monte Carlo sampling. We concentrated our search on the simplest prerequisites required for success in the SADDE system: we search for possible settings that produce an increase in wealth of non-consumer agents. We found no suitable parameterisations in 10,000 samples.

Concerned by our lack of success, we performed a more thorough search based on enumeration of possible behaviours. First we simplified the search space by converting the agent centric design to a role based protocol design. This involves partitioning the space into homogeneous behaviours for producers, manufacturers and consumers. Under this partitioning, the behaviours that are available to an individual in a particular role are the same as those available to all individuals in the same role. This approach reduces the number of free variables from 120 in the SADDE definition to just 6, two for each of the three roles (see section 2.7.1 for a description of the parameters used in the SADDE simulations). Partitioning the space is required to reduce the number of variables to a manageable number. With just 6 variables to search, we have sufficient resources to search the possible combinations of the variables at a resolution of 10 different values for each variable. Since producers and manufacturers simply have to make a profit from the sale of their goods, and are not in competition with each other to achieve the sale, homogenising the behaviours and partitioning the space in this way does not create any inherent conflicts which would obviously prevent successful parameterisation of the system.

We found no parameterisations that produced a positive growth in wealth of the producer or consumer agents, confirming that the combination of the fragments of descriptions in the SADDE related literature is insufficient to produce a feasible optimisation landscape.

There is one other source for a parameterisation that could be used to create a functional reimplementation of the SADDE system. In the annex A of [80] there is a description of an implementation which is supposed to relate to the flow equations in the same document. Unfortunately the values contained in the annex require different

flow equations since they describe maximum and minimum prices for buying and selling at each tier, properties that are not represented in the published flow equations. It is therefore necessary to expand rather than merely reimplement the SADDE system.

2.6.3 Expanding the SADDE system

Given the great difficulty involved in re-implementing the multi-agent system as described in the SADDE literature, we must recognise that re-implementation has failed and we are not going to be able to use comparisons against results published in the literature. Instead we use the opportunity to expand the multi-agent system described in the source code, creating a multi-agent system that is based on the description in the SADDE literature, but that is adapted to protocols and optimised with regard to a more portable fitness function.

We wish to remain as faithful as possible to the SADDE design for a multi-agent system. Using a peer reviewed system created by a different research group helps to prevent intellectual incest and the scale of the SADDE system and the difficulty that was reported in optimising the system guarantees protection against charges of triviality. The SADDE design has proven to represent a complicated problem which is costly to search, and the optimisation of the system has been previously tackled with evolutionary computing.

Using the SADDE design as a template for publishing a new problem specification allows us to retain the justifications of the scale of the simulations, whilst simultaneously allowing us to publish a new, unambiguous, and hopefully complete specification which can be used in future research.

2.7 Extended SADDE system

We attempted to minimise the changes made to the SADDE system, nonetheless it was necessary to make several changes. We expanded the abilities of agents in the multi-agent system to allow inclusion of the properties referred to in annex A of [80], we then referred to the source code for instruction on how to modify the flow equations. We also made modifications to the flow equations to allow floating point values for cash and included kinder handling of cash, allowing an agent to manufacture goods in quantities that it can afford rather than the bulk quantities used in the SADDE source code. Finally we changed the mechanism by which the system is parameterised to a

role based rather than an agent-specific mechanism, to better represent the limited type of control available when using protocols.

In the following sections we give a description of the major components of the extended SADDE system.

2.7.1 Negotiation engine

The basic design of the negotiation engine is unchanged from that specified in [80]. Agents negotiate by exchanging propositions for the value of the goods they are to exchange. Initially the seller approaches the buyer, a quirk of how the trade is tiered in the SADDE system. The buyer then makes the first proposition to trade. The seller then compares the utility of the buyer's offer of exchange against the utility of the counter offer they are prepared to make to the buyer. The deal is acceptable to both parties when the recipient of a proposal evaluates the utility of a received proposal as greater than the utility of the counter offer it is prepared to send next. If an offer is not acceptable then the prepared counter offer is sent, and the process continues until either an agreement is reached or an agent withdraws from the negotiations.

The possible interactions in the negotiation model are limited. Over a period of time agents lower their offers towards their reserve price, gradually decreasing their utility. The negotiation process performed by an agent is characterised by two values, β which alters the type of tactic an agent uses when it falls back to its reserve price and t_{max} which controls the rate at which an agent falls back to its reserve price, and so specifies the maximum period of time an agent will engage in negotiations. After t_{max} time steps of the negotiation, the agent will be at its reserve price and will cease to negotiate further. Negotiations may be concluded by agreement or by an agent withdrawing from the negotiation process after exceeding their t_{max} limit.

The possible states in the negotiation cycle are shown in figure 2.3. Initially an offer is made from the buyer to the seller. The seller considers the offer and chooses to either withdraw from the negotiation, accept the offer and trade under the conditions specified by the buyer or reject the buyer's offer and propose a counter offer. By proposing a counter offer the seller returns control of the trade to the buying agent. In figure 2.3 the states in which the buying agent has control of the negotiation process are coloured. As is normal in a state transition diagram the start state for the process (state 1) has a heavy border, and the final states are marked by double borders.

Once a proposition has been accepted the trading agents are committed to exchange

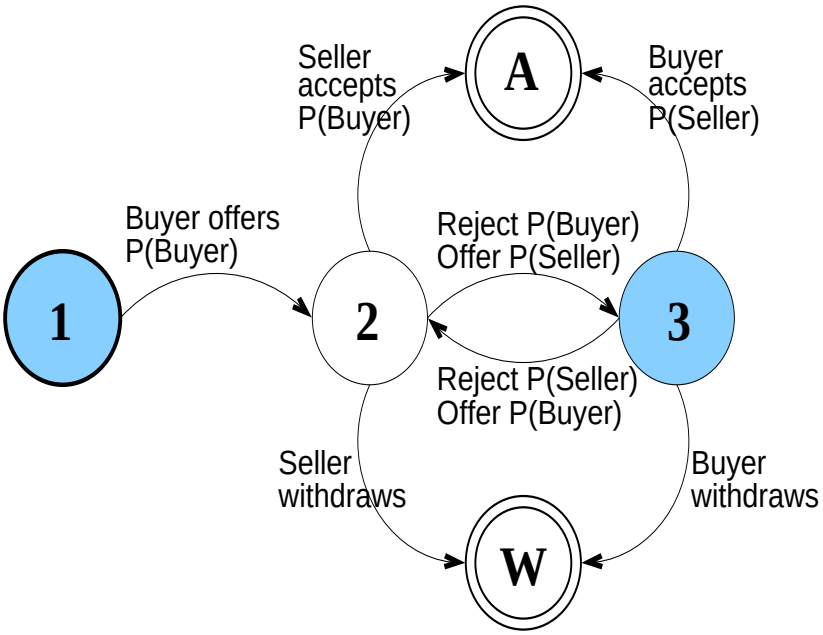

Figure 2.3: The negotiation cycle. Negotiations commence when a buyer in state 1 offers a proposed trade $P(Buyer)$ to a seller, the seller now has control of the negotiation process (state 2) and can accept the offer (A), withdraw from the negotiations (W) or change to state 3 by proposing a counter proposal $P(Seller)$ to the buyer. The buyer is now in control and may now either accept the seller's proposal, withdraw from the negotiations or continue by issuing another offer to the seller and returning the cycle to state 2

the goods at the price agreed. Goods are transferred in integer quantities. The actual quantity of goods transferred is determined by the raw material storage capacity, and finances of the buying agent and the selling agent's stock availability. The maximum quantity that the buyer can afford is given by $\lfloor \frac{cash}{price} \rfloor$. Once stock has been bought it is removed from the seller's available stock and placed into the buying agent's raw material. Consumer agents dispose of purchased raw material immediately and so a consumer is never limited by their storage capacity, nor are they subject to a processing delay.

An agent's negotiation tactics are described as either "boulware" or "conceder" depending upon whether they slowly or rapidly fall back to their reserve price as they approach their maximum negotiation time [80]. Agents have minimum and maximum price bounds and a function α that determines the difference between an agent's reserve price and their current bid. α is a time dependent function, characterised by t_{max} and β.

$$\alpha = \frac{t}{t_{max}}^{\frac{1}{\beta}}$$

$$proposal_{buyer} = \lfloor minPriceIn + \alpha(maxPriceIn - minPriceIn) \rfloor$$

$$proposal_{seller} = \lfloor minPriceOut + (1-\alpha)(maxPriceOut - minPriceOut) \rfloor$$

$$utility_{buyer}(\text{offer}) = \begin{cases} 0, & \text{if offer} > \text{maxPriceIn} \\ 1, & \text{if offer} < \text{minPriceIn} \\ \frac{offer - minPriceIn}{maxPriceIn - minPriceIn} & \text{otherwise} \end{cases}$$

$$utility_{seller}(\text{offer}) = \begin{cases} 0, & \text{if offer} < \text{minPriceOut} \\ 1, & \text{if offer} > \text{maxPriceOut} \\ \frac{maxPriceOut - offer}{maxPriceOut - minPriceOut} & \text{otherwise} \end{cases}$$

Figure 2.4: The utility and proposal generation formulas, the main constituents of the negotiation mechanism

Figure 2.4 shows the formulas used by negotiating agents to calculate their current bids. These formulae are taken directly from [80] and are in the SADDE source code. Note the role dependent generation of proposals and calculation of utility.

In all our experiments the variable t_{max}, which represents the amount of time an agent will spend in negotiation, is in the range $[1,100]$. The tactic parameter β is permitted in the range $[0,10]$. These values are essentially arbitrary, however the value of t_{max} defines a limit on the number of negotiation ticks that will be tried before an agent withdraws from the negotiation process, and therefore directly affects the simulation time.

Agents calculate the utility of a proposed exchange by comparison of the offered value against the maximum and minimum values that they would use in that role.

Figure 2.5 shows the possible range of α values available to agents. We follow the description in [80] precisely. It is interesting to note that agents are encoded with only one tactic variable β, forcing agents to use the same tactic in both buying and selling negotiation roles. An agent which has a high α value will negotiate prices close to its reserve price. An agents β value alters how the α value changes during the negotiation

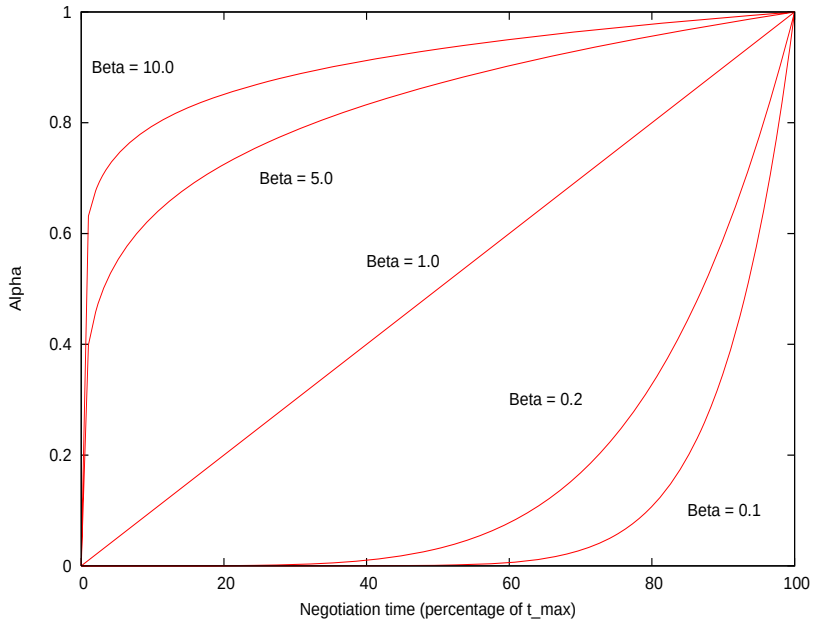

Figure 2.5: Each agent uses a tactic in negotiating a deal. In this example we are considering a selling agent. So called "conceder" agents rapidly fall back to their reserve price (a high α) long before exhausting their time limit. "Boulware" agents on the other hand persist in maintaining a low α value until finally rapidly conceding to their reservation price in the final stages. The choice of tactic is encoded in the values of β assigned to each agent, the effect on α of different values of β are shown here. The relation between α and buying price is given in figure 2.4.

period.

The time used by agents in negotiations is independent of the world clock, so an agent that takes many exchanges to complete its negotiation does not find that it is disadvantaged relative to agents that have negotiated deals quicker, despite this clearly being the case in the real world.

The simulation of the supply chain is performed for 500 ticks. Each tick of the clock replenishes the consumables in the supply chain and updates agent's production schedules and stock availability, as well as charging the agents maintenance costs. In one tick the following actions are performed in order (see also figure 2.6).

- Raw materials are introduced to the producer agents by the *entradaMat* function (figure 2.7).

- Producer and manufacturer agents perform one step of their production process according to the *production* function (figure 2.8).

- Consumer agents get paid a quantity of cash by the *salary* function (figure 2.10).
- All agents pay maintenance costs, simulated by the *maintenance* function (figure 2.9).
- One complete round of agent trading takes place.
- A record is made of the levels of cash in the production and manufacturing agents, for use in the fitness evaluation.

The population performs one complete cycle of negotiations per tick. The negotiation function is as shown in figure 2.11. Every production agent performs one negotiation as a seller with a buyer selected without replacement from the set of manufacturing agents. Next, every manufacturing agent performs one negotiation as a seller with a buyer selected without replacement from the set of consumers.

2.7.2 Parameters

In the flow equations in [80, 88], the agents have properties such as $minPrice_i$ which control the trading behaviour of level i agents. The experiment description in the appendix of [80] details the same variables but distinguishes the role of buyer and seller, thus $minPrice_i$ becomes two distinct values; $minPriceIn_i$ and $minPriceOut_i$. This formulation is also used throughout the source code.

Table 2.12 gives the parameters used in our experiments. The agent role types are denoted by the subscripts; *prod*, *man* and *cons* indicating a producer, a manufacturer and a consumer role respectively. The symbols in the table have the same meanings as in the SADDE work, *minPriceIn*, *maxPriceIn* and *minPriceOut*, *maxPriceOut* specify the minimum and maximum range of prices that agents are willing to purchase and sell goods. Stock held by an agent may be unprocessed raw material (*stockIn*) or processed goods ready for sale (*stockOut*), both of which are subject to storage limits; *maxStockIn* and *maxStockOut*.

Agents are charged a tax regularly by the function *maintenance*, which deducts cash from agents, simulating subsistence costs. Producers and manufacturers have a maximum production rate *prodRate*, which determines the maximum rate at which the unprocessed material may be converted to processed goods. Processing of goods is performed periodically, according to a processing delay *delay*. Processing goods costs money, and so agents pay *prodCosts* per unit of goods manufactured, consequently production and manufacturing agents require a quantity of cash to process their goods

for resale. To cover initial production costs and kick-start the flow of goods in the supply chain, production and manufacturing agents are initialised with a quantity of cash; *initialCash*. To maintain the flow of cash through the system consumer agents receive a regular payment *salary*, which combined with the injection of goods to production agents allows a well configured supply chain to maintain the flow of trade. Flow through the supply chain stalls when participating agents have insufficient money to cover the costs of purchasing or producing more goods. Stalled agents may be redeemed if they find a buyer of any products they have remaining, in the meantime they will be unable to pay maintenance costs.

2.7.3 A new fitness function

We create a fitness function that is based on the same properties valued in the fitness function of the SADDE agent system; an increase in net wealth in the production and manufacturing levels of the supply chain. We do not have an equation based model to compare the performance to, instead we simply view the fitness of the system from a capitalist perspective, and aim for maximisation of the profit accumulated. As in [80] our fitness function is based on a record of the cash levels of the production and manufacturing agents at each global tick of the simulation. The objective of optimising our extended version of the SADDE multi-agent system is to maximise the fitness of the system according to the fitness function given in figure 2.13.

2.7.4 Conversion to protocols and experiments

The multi-agent system described in the SADDE literature is implemented from an agent-centric perspective. In an agent centric design, individual agents are identifiable and the trading behaviour of each agent can be manipulated directly. In this work we are interested in the parameterisation of protocol based multi-agent system design methods. General protocols do not normally identify behaviours for particular agents, since this restricts their range of applicability. We need to express the behaviours available to each type of agent, rather than encoding behaviours for each agent.

There are three possible roles an agent may take in a protocol based interpretation of the SADDE supply chain. Each role has a trading behaviour specified by a (t_{max}, β) pair. Optimisation of the protocol in this instance is the task of searching the space of possible trading behaviours to maximise the fitness function given in figure 2.13 for a population of some number of agents. The SADDE work used a population of 60

agents in the proportions 10 producers, 20 manufacturers and 30 consumers. We keep these relative proportions for our experiments.

As shown in figure 2.14 the basic negotiation protocol implemented by the SADDE system is almost identical to that of the proof of concept system. Agents do not change role in the SADDE system so this aspect of control is fixed once the role has been assigned. As befits the granularity of a protocol based control mechanism, agent behaviours are controlled through the identification of their participating role; the protocol does not distinguish agents individually. Agents trade by offering exchanges. Exchanges are limited to occurring between producers and manufacturers or manufacturers and consumers. The trade conditions offered by a producer are denoted T_{PO}, similarly, offered trade conditions offered by manufacturers and consumers are indicated by T_{MO} and T_{CO} respectively. All offered trade conditions are calculated subject to the constraint imposed on the roles' trade negotiation behaviour $Cons_{P_{Neg}}(T_{PO}, T_{MO})$ indicating the constraint acting on a producer's negotiation behaviour when evaluating the utility of the currently tabled trade offers T_{PO} and T_{MO}. The mechanism of applying this constrained negotiation behaviour is through specifying values for β and t_{max} as discussed earlier. Similar constraints apply to the negotiation behaviour of the manufacturers and consumers. Note that the manufacturer is capable of fulfilling two roles, that of buyer when dealing with producer agents, and that of seller when dealing with consumer agents.

It is clear that the proof of concept system, which is used to prove the utility of message passing protocol based control is of a very similar form to the SADDE negotiation protocol. Both systems use role based characterisation of behaviours, the control of the emergent system dynamics is performed through the manipulation of constraints applied to the behaviours available to these roles.

Unlike the proof of concept system, the actual SADDE negotiation protocol is never implemented as a message passing system. The SADDE system is much larger than the proof of concept system. The additional burden of generating, exchanging and parsing exchanged messages is great and is unnecessary when the same effect may be simulated directly. This abstraction does not bear adversely upon the findings of the work. The SADDE system is fundamentally the same in operation and scope, but this alteration in the details of the implementation allows us to simulate larger systems than we would have otherwise been able.

The control of a multi-agent system can be viewed as the task of imposing correct constraints on behaviours. The constraints within a protocol definition are adjustable

without having to retract and then reissue the full protocol. The constraints may be modified almost arbitrarily as long as the modification remains "safe" with regard to the capabilities of the participating agents. Constraints within a protocol definition essentially allow indirect access to the concepts manipulated by agents. By modifying the relationship between variables in the constraints the designer may alter the behaviour of the system as a whole. The task of optimising a multi-agent system that is based on a constrained protocol may then be reformulated as a real number optimisation problem with no loss of expressive power. It is this connection that allows us to then use the resources of the real number optimisation fraternity for optimising multi-agent systems. The rest of this thesis is devoted to describing and developing these tools.

2.8 Summary

The control of multi-agent systems is complicated, and the task is genuinely interesting in itself. As multi-agent systems grow in complexity, so does the complexity of their development and maintenance. If such systems are to reach maturity and be deployed in a meaningful and useful sense the systems themselves (and not only the agents in them) must be adaptable. Suitably skilled human caretakers are rare and have a tendency to be too expensive to invest the time required to perform everyday maintenance. Reactive systems that are capable of adapting to changing user or system objectives are a desirable asset. Optimisation of reactive systems is also desirable. Mechanisms that are capable of searching the space of system parameterisations will have to be developed, and constitute a significant design challenge.

This work concerns the design of an algorithm for optimisation of multi-agent systems. It is a case study in building a mechanism to optimise a previously unknown system. The design process is difficult; we are building a mechanism for which there is no large body of relevant past works to guide efforts. To complicate matters the test function is computationally expensive (and arguably incompletely specified), and so can not be directly used in the algorithm development. Instead the optimisation algorithm must be designed and tested against test functions that are thought to reflect expected properties in the domain.

The multi-agent system in terms of an optimisation objective, has no particular significance to this work, and could just have easily been one of any number of similar real number optimisation problems drawn from the physical sciences or robotics. The

major challenge of the problem is to find optimal parameterisations despite the problem complexity inherently causing a a chronic lack of samples in the search.

The multi-agent system does impact upon the choice of functions that are used in the construction of the algorithm. The parameterisation of the agent system is a real number optimisation task so real number representation and manipulation are required. The problem is likely to have multiple optima and so the algorithm must be robust at locating global optima in spaces with many false attractors. The agent problem space may be subject to constraints (in the real number optimisation sense) and so be likely to have large discontinuities in the space of acceptable solutions. The algorithm must be capable of searching effectively in spaces that are constrained or otherwise discontinuous. The agent system is computationally expensive to simulate, limiting the number of evaluations that may be used and making effective search of the space difficult. The algorithm should therefore be efficient in its use of samples. The agent system may change over time, and the optimisation algorithm performing the adaptation must not be fragile in the face of this change. The algorithm should therefore not be sensitive to its parameters, and whichever parameterisation is used should have a strong likelihood of successfully operating on any related problem.

The results of our experiments on the multi-agent systems described in this chapter are in chapter 9, after we have described the design process of creating an algorithm capable of optimising on such problems.

```
// the simulation of a supply chain is performed for a period of ticks
function simulate(int ticks) {
    // reset the cash tracking values, and the maintenance default counter
    // these are used in the fitness calculation figure 2.13
    producerCash = 0
    manufacturerCash = 0
    maintenanceDefaults = 0
    for(t = 0; t < ticks; t++){
        tick()
    }
}

// each tick is a complete cycle of production, payment, maintenance, and negotiation
function tick() {
    for each agent{
        entradaMat()
        production()
        salary()
        maintenance()
    }
    negotiatePopulation()
    recordValues()
}

    // a utility function that records cash levels for use
    // in establishing the fitness of the supply chain
function recordValues() {
    // fitness is based on the total quantity of cash that
    // has been held by the producers and the manufacturers
    producerCash += ∑_{p=0}^{∀producers} cash held by producer p
    maufacturerCash += ∑_{m=0}^{∀maufacturers} cash held by manufacturer m
}
```

Figure 2.6: The tick function used in one cycle of the system simulation

```
function entradaMat() {
    // do nothing unless producer
    if(not producer){
        return
    }
    // calculate the price per item
    price = minPriceIn + ⌈ (maxPriceIn−minPriceIn) / (stockIn−stockOut) / (0.5∗(maxStockIn+maxStockOut)) ⌉
    // if can afford to produce an item
    if(cash > price){
        // examine the constraints on production:
        // the amount of free storage
        spareCapacity = maxStockIn - stockIn
        // the amount that can be payed for
        affordableQty = ⌊ cash / price ⌋
        // the (randomised) amount actually generatable
        roughMaterial = random(minRoughMaterial, maxRoughMaterial)
        // the actual amount produced by the agent
        materialFlow = min(spareCapacity, affordableQty, roughMaterial)
    } else {
        materialFlow = 0
    }
    // move the stock into stockIn
    stockIn += materialFlow
    // reduce cash by appropriate amount
    cash -= materialFlow*price
}
```

Figure 2.7: The function used to introduce material to the producing agents

```
function production() {
    // do nothing unless producer or manufacturer
    if(not producer or manufacturer){
        return
    }
    // goods are produced only after a delay has expired:
    if(productionDelayTime == 0){
        // the amount producable is limited by: production rate,
        // raw material and the capacity for storing the product:
        availableQty = min(prodRate, stockIn, maxStockOut - stockOut)
        // the production quantity that can be afforded :
        affordableQty = cash / prodCosts
        // the amount that will be produced
        quantity = min(availableQty,affordableQty)
        // the cost to the producer
        cost = quantity * prodCosts
        // effect the transfer of stock
        stockIn -= quantity
        stockOut += quantity
        cash -= cost
        // reset the production delay counter
        productionDelayTime = delay
    }else{
        productionDelayTime = productionDelayTime - 1
    }
}
```

Figure 2.8: The function used to simulate production of goods from raw materials by the production manufacturing agents

```
function maintenance() {
    // each agent type pays a different maintenance cost
    if(producer){
        cash -= maintenance_prod
    } else if(manufacturer){
        cash -= maintenance_man
    // otherwise must be a consumer
    } else {
    // erode unspent cash          cash = cash / 2
    }
    if(cash < 0){
        // no agent can have negative cash
        cash = 0;
        // record the failure to pay maintenance
        // for the fitness evaluation
        maintenanceDefaults ++
    }
}
```

Figure 2.9: The function used to simulate maintenance costs experienced by trading agents. An agent that has bankrupted itself has its back account returned to zero and is permitted to continue trading, the failure to manage to pay the maintenance is recorded and will be used in the calculation of the fitness of the multi-agent system

```
function salary() {
    // do nothing unless consumer
    if(not consumer){
        return
    }
    // increase cash reserve by the consumer salary
    cash += salary
}
```

Figure 2.10: The function used replenish the cash of consumers, permitting continued trading and stimulating the flow of goods

```
function negotiatePopulation() {

    // trade between producers and manufacturers
    // a list of potential buyers
    L = the list of manufacturers

    for(each producer, P){
        locate a manufacturer M
        negotiate: P sells to M
        // remove M from the list of buyers
        remove M from L
    }

    // trade between manufacturers and consumers
    // refresh the list of potential buyers
    L = the list of consumers
    for(each manufacturer, M){
        locate a consumer C
        negotiate: M sells to C
        // remove C from the list of buyers
        remove C from L
    }
}
```

Figure 2.11: The negotiation function. Every tick of the simulation involves one attempted negotiation by the producer and manufacturer agents.

Variable	Value	Variable	Value
$minPriceIn_{prod}$	10	$maxPriceIn_{prod}$	50
$minPriceOut_{prod}$	50	$maxPriceOut_{prod}$	100
$minPriceIn_{man}$	10	$maxPriceIn_{man}$	100
$minPriceOut_{man}$	80	$maxPriceOut_{man}$	135
$minPriceIn_{cons}$	10	$maxPriceIn_{cons}$	150
$maxStockIn_{prod}$	5000	$maxStockOut_{prod}$	5000
$maxStockIn_{man}$	5000	$maxStockOut_{man}$	5000
$maxStockIn_{cons}$	5000	$prodRate_{prod}$	2000
$prodRate_{man}$	1000	$maintenance_{prod}$	20
$maintenance_{man}$	40	$prodCosts_{prod}$	$minPriceIn_{prod}/2$
$prodCosts_{man}$	$minPriceIn_{man}/2$	$delay_{prod}$	5
$delay_{man}$	5	$initialCash_{prod}$	50000
$initialCash_{man}$	25000	$salary_{cons}$	4000;
number of producers	10	number of manufacturers	20
number of consumers	30		

Figure 2.12: Extended system parameters

```
function fitness() {

    // initialise the fitness
    fitness = 0
    // we reward the enrichment of producers and manufacturers
    fitness += producerCash
    fitness += manufacturerCash
    // we reward processing of raw material to product
    fitness += 4 * total processed by producers
    fitness += 3 * total processed by manufacturers
    // we reward the maintenance of a cash reserve
    fitness += 1000 * sum of final cash held by producers
    fitness += 100 * sum of final cash held by manufacturers
    // we penalise every instance an agent has failed to pay
    // their maintenance costs
    fitness -= 1000 * maintenanceDefaults
    return fitness
}
```

Figure 2.13: The fitness function rewards the following properties : The sum of the cash at each level throughout the simulation. The quantity of goods processed by producers and manufacturers. The total flow of goods, represented by the quantity of goods consumed. The final quantity of cash held by producers and manufactures. The fitness function penalises any failure of an agent to pay their maintenance costs.

$$
\begin{aligned}
a(node, N) :: \quad & a(producer, P) \\
& a(manufacturer, M) \\
& a(consumer, C) \\
a(producer, P) :: \quad & \text{offer}(T_{MO}) \Leftarrow a(manufacturer, M) \text{ then} \\
& \text{offer}(T_{PO}) \Rightarrow a(manufacturer, M) \\
& \leftarrow Cons_{P_{Neg}}(T_{MO}, T_{PO}) \\
a(manufacturer, P) :: \quad & \text{offer}(T_{PO}) \Leftarrow a(producer, P) \text{ then} \\
& \text{offer}(T_{MO}) \Rightarrow a(producer, P) \\
& \leftarrow Cons_{M_{Neg}}(T_{PO}, T_{MO}) \\
a(manufacturer, P) :: \quad & \text{offer}(T_{CO}) \Leftarrow a(consumer, C) \text{ then} \\
& \text{offer}(T_{MO}) \Rightarrow a(consumer, C) \\
& \leftarrow Cons_{M_{Neg}}(T_{CO}, T_{MO}) \\
a(consumer, C) :: \quad & \text{offer}(T_{MO}) \Leftarrow a(manufacturer, M) \text{ then} \\
& \text{offer}(T_{CO}) \Rightarrow a(manufacturer, M) \\
& \leftarrow Cons_{P_{Neg}}(T_{MO}, T_{CO})
\end{aligned}
$$

Figure 2.14: The basic SADDE negotiation protocol

Chapter 3

Real number optimisation

3.1 Chapter overview

In this chapter we discuss the problem of optimisation of problems parameterised by real numbers. Some example problems are presented and explained and the twin curses of the search process - dimensionality and precision - are introduced in the context of an example problem. We also discuss common sleight of hand in the field of real number optimisation and how to detect and avoid it.

3.2 Optimisation of real number parameters

Consider the task of designing a turbine blade. The most simple design process is to iteratively create and test blade designs, a costly and time consuming enterprise. Fortunately, the critical characteristics of a blade design can be expressed mathematically as parameters to equations, and more importantly the behaviour of the blade can be more rapidly predicted by a fluid dynamic model based on these parameters. The blade never has to actually exist in order for the design to be evaluated. The design process considers not a physical turbine blade but a series of values.

Indeed, most design problems can be reduced to the consideration of a number of parameters, which interact in some manner to produce a system with certain observable properties. The designer's objective is to find the best set of parameters given the limited time and costs available for their search. Most systems may be approximated by a computer model, which, when used as a surrogate with which to search for suitable parameterisations, greatly speeds the evaluation process.

Having automated the evaluation of parameters, it is logical to also automate the

search for new parameterisations, entirely removing the role of the human designer from the design process. The idealised manifestation of the process would be the fabled "black box general optimiser" which efficiently and effectively optimises any given problem. Unfortunately such a general device does not exist, at least not one that performs better than random search.

Instead we obtain improved specific performance by sacrificing the generality of the optimiser, which searches using structures that are expected to exist in the search space. The closeness of the pairing of the search algorithm to the search space is directly responsible for the quality of the search results over those of a randomised search. Many different search algorithms have been proposed, and even within one field, such as theoretical real number optimisation, there are a vast number of search techniques.

Which method is best employed to solve a particular parameterisation problem is dependent upon the precise properties of the problem. Some properties are non-negotiable, such as the number of system performance evaluations which can be afforded by the user (this may be very low in a physics or complex system simulation), or the requirement of an "anytime" answer provision service which requires that a valid solution (though not necessarily the best solution) be available for use at all stages of the search. Other properties of the problem are not immediately apparent and must be discovered by exploring the relationships between the various parts of the parameterisation space. This second set of properties are the most interesting, and it is the more complex of these, which manifest themselves to the observer as interactions between the variables, which give a problem its specific characteristics.

3.3 Properties of real number optimisation

Many design and decision processes can be expressed as a vector of real values. Each dimensional component of the vector relates to a parameter setting in the problem. The space of vectors is then the same as the space of all possible parameter settings. By searching the space of possible vectors it is possible to locate the best assignment of values to the vector, and subsequently solve the problem. In practice the problem is doubly complicated. Most problems have high dimensionality, making conducting even a low resolution mapping of the space infeasibly expensive. The real number system also allows for solutions to be expressed in infinite precision, forcing consideration of the limits of the numeric representation system. In some systems optimality

may never be reached. Genuine but infinitesimally small progress is possible for all solutions involving irrational number representations simply by increasing the precision of the representation.

Thus a search algorithm is doubly damned, once by dimensionality and the insignificance of the volume of sampled points relative to the size of the space, and once again by precision and the absence of atomic granularity in the search - there may simply never be a precise answer, the space may never be exhausted of possibilities. This problem with precision becomes antagonistic when deciding what granularity to use in the search. Unless enumerating the space it is never safe to exclude the possibility that the search is just too coarse grained, and should be modified to examine the space as closely as the representation permits. In most cases this is not possible; using very fine grained search takes a prohibitive number of samples to locate an optimum. As a way of explanation we will review the well known Rastrigin function.

3.3.1 An example: The Rastrigin problem

Rastrigin contributed [85, 63] a minimisation test problem which has become a standard benchmark problem in the field of real number optimisation. The function has the following definition.

$$f_{rastrigin}(\vec{x}) = a.n + \sum_{i=1}^{n}((x_i)^2 - a.cos(\omega.x_i))$$

$\vec{x} = (x_1, .., x_n)$, $-5.12 \leq x_i \leq 5.12$
where $a = 10$ and $\omega = 2\pi$ are constants and n is the number of dimensions.

$f_{rastrigin}(optimum) = 0$ (at (0,0,...0))

The Rastrigin function is simple to visualise. The first important component of the function is the x_i^2 term which dominates the structure creating a bowl shaped function in each of its dimensions. The second component of interest is the cosine of x_i which creates a (co)sinusoidal wave function which is phased to cycle once per each integer increment of x_i (due to the arbitrary choice of 2π as a constant multiplier). The sum simply collects together the results over all dimensions. Please see fig. 3.1. Notice that this function is clearly solvable by optimisation of one dimension at a time. Realisation of this function in all dimensions greater than 1 produces valleys in the landscape where the search may proceed towards a solution without being forced to cross every

peak. The function has one global optimum of height zero and for the default parameter range has $11^n - 1$ other optima.

3.3.2 Dimensionality in the Rastrigin problem

This landscape is not very complicated and yet has some challenging features which make it hard for a naive optimisation strategy. The function is well defined over all dimensions but is generally used in 25 or 30 dimensions. The apparent rationale for using this level of dimensionality is that it appears to create problems which are just beyond the solvable threshold for most approaches given the the 'standard' amount of evaluations. They are thus informative in comparing the relative performance of different solvers. We will look at the different evaluation contexts when we review the reported performances of other techniques in the 7^{th} chapter of this work.

If we consider a close up of the two dimensional case (figure 3.2), we can see the difficulty faced by any strategy searching for an optima. Figure 3.2 has five points of interest marked on it. The global optimum is at $(0,0)$ and is marked by the label *optimum*. Four other points are marked on the graph, labeled *A* through to *D*.

Position A is the point at $(-0.5,1.5)$ and is representative of the situation of optimising towards a global optimum that is not in the immediate neighbourhood of the search. The prospect of successful search from Position A is impeded by the presence in the immediate vicinity of Position A of 4 local minima. With very high probability a greedy optimiser that starts in Position A will fail to reach the optimum. Though not likely, it is however possible to descend monotonically from Position A to the global optimum by following the route that descends to the saddle point between Position A and Position B and then skirting Position B at the same height until descending again to the global optimum.

Position B is at $(-0.5,0.5)$ and so is in the neighbourhood of the global optimum but without further information simple search has a one in four chance of descending into the correct basin. Once misled a simple algorithm may find it hard to distinguish the following two cases: 1. that the algorithm has been deceived and is in the wrong basin, and 2. that the algorithm simply needs to further refine the point a little to locate the optimum. Successfully responding to situation 1. requires long range movement to escape the basin, whilst situation 2. would require very small movements.

Position C at $(-0.25,0)$ is in the basin of the global optimum. From this position, where one dimension of the position is optimally chosen most optimisers would be

expected to locate the optimum. However, as the number of dimensions in the problem increases the difficulty of finalising the remaining values increases. This problem, of identifying which dimensions require further adjustment, is the first of the two curses of high dimensional optimisation. Quite aside from the sheer size of the space, locating and then fixing a minority of miss-set values is hard, even in linear space. This problem is the cause of the exponential time complexity in search based on probabilistic mutation, see the discussion on mutation operators in section 6.7.2.

Optimising from Position C moves you towards the optimum and reduces the proportion of the space that remains into which successful movements may be made. This is represented by Position D which is at $(-0.1, 0)$. From this position, a mere 2.996×10^{-4} of the space is closer to or equally distant from the optimum. Precision, in terms of the accuracy required of an acceptable answer, has a clear influence on the difficulty of the search. The choice of precision implicitly determines the size of movements made in the search. The uncertainty in knowing if these steps were the right size is the source of the second curse of high dimensional optimisation.

The probability of making random moves towards the optimum is clearly proportional to the relative volume of the space nearer the optimum, which is also clearly adversely affected by the dimensionality of the space. Consequently getting within ε of the optimum in n dimensional space is considerably easier than getting within the same distance in $n+1$ dimensional space.

3.3.3 Dimensional precision in the Rastrigin problem

Understanding why high dimensionality should prove antagonistic to precision comes from realising that not all dimensions will be optimised at an equal rate. As shown earlier, once a dimension is approximately correct any change will with high probability worsen the performance. As the search progresses and more and more of the problem is solved the proportions of the dimensions which are set relative to those still to be set increases. The net effect is a decrease in the probability of a random change being attempted only against those dimensions which remain to be improved. The actual space of movement which can offer an improvement in any dimension decreases, as does the number of dimensions against which effective improvement can be made. Thus as the solver gets closer to the solution and solves or approaches optimal choices in different dimensions, it suffers from a reduction of effective power. The first problem is the problem of precision in movement and the relative volumes involved in the search

space, the second is the "coupon collector's paradox" in the guise of the problem of randomly selecting dimensions. The coupon collector's paradox occurs when attempting to accumulate a collection of "coupons" that are provided at random (in a cereal packet or similar). As the proportion of the collection owned by the collector grows they find that the majority of coupons now collected are already in the collection, and progress made in collecting the remaining coupons slows dramatically.

It is possible to remedy some of these effects. Consider Position C in figure 3.2. The probability of any local movement improving the score is limited to one half (for infinitely small movements) and this probability decreases as the distance moved increases. Making small movements is then more likely to be successful in a bowl shaped landscape than making large movements.

The area which qualifies as a local improvement from Position C is shaded differently for the reader's convenience. The situation faced when very close to the optimum is harder still. Position D is at $(-0.1, 0)$ and suffers from the same problems that searching from position C did, however now the movements have to be smaller still if there is to be any hope of improving further. Fixed length movements are unsuitable for this type of finessing of solutions, since any length small enough to optimise from Position D is also going to take a very long time to get from Position A to Position D. Most real number optimisers have a strategy for adjusting the length of movement to overcome this precision deficit. To the best of our knowledge there are no techniques for adequately handling the selection of dimensions.

3.3.4 Constraints : More complicated problems

The Rastrigin function is simple to visualise and properties learnt from studying low dimensional variants remain faithful in higher dimensions. Part of the reason why the Rastrigin function is easy to understand is that the problem is consistent over the entire range of values and between dimensions. Unfortunately this is not true for the vast majority of problems. Frequently in real number optimisation there are non-linear interactions between the dimensions of the problem that make certain combinations of values illegal as solutions. These are frequently found in applications where physical constraints are in operation. An example would be where several machines have to be used to process some goods, but the machines may not be operational all at the same time due to their power usage. There is then a hard constraint on which combinations of machines are permitted to operate at the same time.

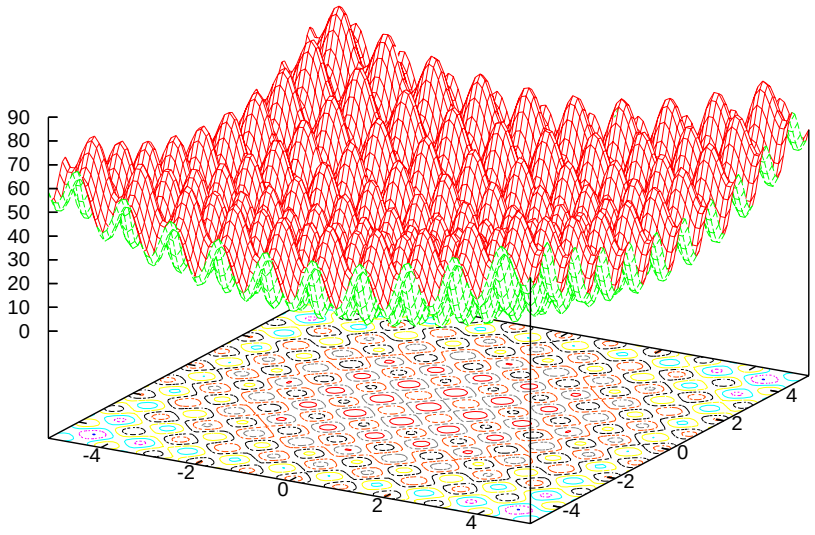

Figure 3.1: The 2D Rastrigin function.

3.3.5 Keane's function

In [47, 48] Keane gave details of a function he was using to test the ability of an algorithm to follow a ridge of constrained values. The function defines a maximisation problem and has the following form:

$$f_{keane}(\vec{x}) = \frac{|\sum_{i=1}^{n}(cos^4(x_i) - 2\prod_{i=1}^{n}(cos^2(x_i)))|}{\sqrt{(\sum_{i=1}^{n}(ix_i^2))}}$$

$$\vec{x} = (x_1, ..., x_n),\ 0 \leq x_i \leq 10$$
$$\text{where } \prod_{i=1}^{n}(x_i) > 0.75, \sum_{i=1}^{n}(x_i) < 7.5n$$
$$\text{n = 20 or 50 is the number of dimensions.}$$

$$f_{keane}(optimum) = \text{unknown}$$

Michalewicz [59] applied GENOCOP III (a highly specialised variant of a genetic algorithm) to several constrained non-linear optimisation problems including the Keane function with good results. GENOCOP III has bespoke mechanisms precisely designed to overcome the discontinuities in the space created by the constraints. In figure 3.3 a plot of the surface of the 2 dimensional Keane function is given. Only points within the constrained region are plotted, illegitimate parameter combinations are not shown. The Keane function is similar to the Rastrigin function. The denominator

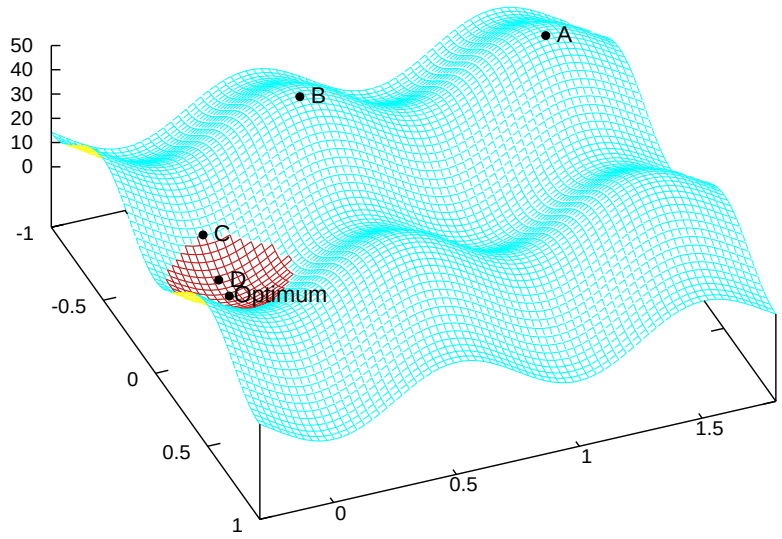

Figure 3.2: Detail of the 2D Rastrigin function near the optimum.

shapes the landscape such that better values are located near the origin, the constraint $\prod_{i=1}^{n}(x_i) > 0.75$ cuts across the optimal values, creating a landscape that rewards extremely fine adjustment against a hard boundary.

3.3.6 Penalty functions

Functions that have constraints in their definition can be converted to linear non-constrained functions by the adoption of a penalty function into the function definition. The role of the penalty function is to differentiate between infeasible points on the basis of the number of constraints they breach. This allows the solver to approach the problem in two stages: the first is minimisation of the number of constraints violated, the second is the finessing of a valid solution. These techniques have particular value if the proportion of valid to invalid space is small since without this information the search is likely to remain trapped in an infeasible part of space.

3.4 Dangers in optimisation

The purpose of research in this field should be to improve understanding of what occurs during a population based real number optimisation. For reasons unknown, it

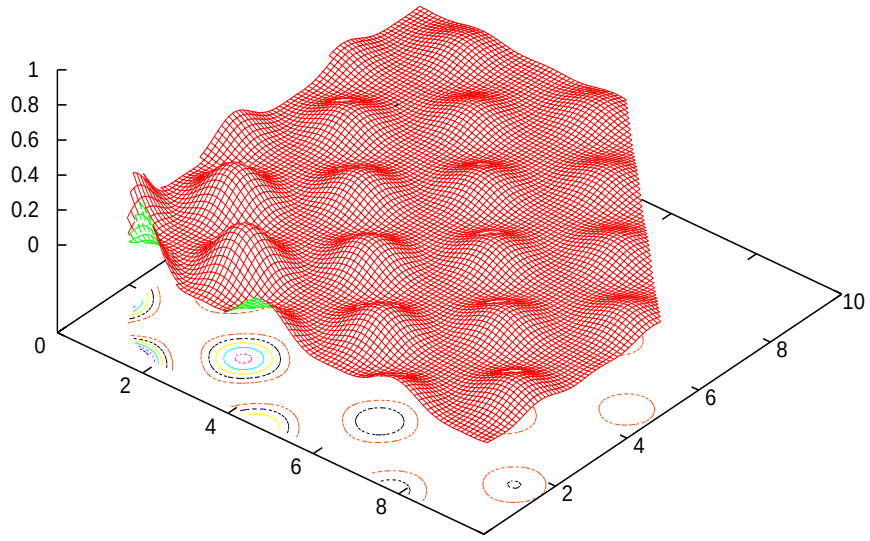

Figure 3.3: The 2D Keane function.

is common to see reviewed results published which suffer from one or more failings which weaken their contribution to this understanding. Here we give a quick introduction to some of the more common flaws which occur through deliberate misdirection or genuine mistake. We hope that by identifying these dangers now, we can prevent ourselves and others from wasting their time.

3.4.1 Dangers in optimisation : Over-fitting

Unless very well supported, finding better results on a single function is useless in terms of contributing to the understanding of algorithm design. Given enough patience and funding, anyone persistent enough can obtain an algorithm which performs "better" on a particular test case or problem. The mechanism is simple: either through developing understanding of the space or through blind search of the parameters and techniques available it is possible to develop an algorithm that performs better on a problem by being more closely matched to the problem structure than the competing algorithms. Under these circumstances, one of two things may have happened. The first possibility is that the algorithm has been improved in a true sense: it is just less

wasteful than previous versions and is still applicable to the same range of problems. The alternative is that the algorithm is actually over-fitted, and is consequently dependent upon a non-generalisable property of the problem for its success. Improving the performance of an optimiser is simply the task of removing only wasted evaluations. To validate that an improvement is genuine we need to show its performance against multiple and diverse benchmarks. To a certain extent we also need to be able to explain why the algorithm is performing better, at least to the level of detail to be sure that it is not simply exploiting an obvious common artifact of the benchmark functions.

3.4.2 Dangers in optimisation : Exploits

On some problems it is possible, with a little thought, to create operators that rely on the author's knowledge of the problem for improved probability of success. For instance in [60, 74] Michalewicz and co-authors create a system to tackle Keane's function (see 3.3.5). The method they employ uses a system to generate sample points that are only on the boundary of the feasible and infeasible space. This exploits the author's knowledge that, on this problem, the location of the optimum is precisely on this boundary. The increased utility of these operators is dependent on a property of the problem that is not explicit in the problem definition and not deduced during the search. As such, despite the good results this operator combination gives, they are none the less results of exploits that dramatically reduce the difficulty of the problem.

In general the ability to calculate operators capable of using such exploits declines as the complexity of the boundary surface increases. The Keane function has only one constraint crossing the global optimum which is the requirement that $\prod_{i=1}^{n}(x_i) > 0.75$. It is thus trivial to create a mechanism that creates initialisation values that observe the constraint $\prod_{i=1}^{n}(x_i) = 0.75$. The mechanism used in [60, 74] works by generating pairs of numbers that always balance to give the correct product, and thus only works for dimensions of even cardinality. It is more difficult to create a mechanism that successfully incorporates multiple non-linear constraints, and it is even harder to calculate from sampling the space during the run when/if such a mechanism should be employed.

Using the same knowledge you can create a different exploit that will improve performance on the Keane function. All you do is create an operator that ignores the result of the Keane function evaluation and attempts to move samples as close as possible to breaching the product constraint. Since the highest values are always

located on the boundary this will improve any given sample that is not already on the constraint boundary. Application of this method as a “finishing” tactic for another generalised search method should give good results.

The code given in figure3.4 is an example of how a very simple method can be used to “finish” results off such that they are closer to the product constraint boundary than before. Unlike the exploit used in [60, 74], this does not force the positioning of all points on the boundary. It thus a less gratuitous exploit. The point is only modified if the new point is better. Search using this type of constraint boundary exploit will search both sides of the constraint boundary, and will only move to the boundary if the boundary offers an improvement in score. Please see section 4.2 for an explanation of the pseudo-code used in this work.

The most common exploit is accidental and is best described as “the amazing average finding algorithm”. This exploit is commonly induced when the problem suite that is being tested against contains a preponderance of functions with their optimum values at or very near the centre of the space. The exploit occurs by reducing an initially well distributed population towards a predetermined objective. For example, consider an algorithm that starts from a random cluster and calculates the average of the cluster. If it then replaces a random candidate in the cluster with the new found average candidate, it will progressively tend to the centre of the space. Which also happens to be the location of the optimum. For precisely the same reason points nearer the origin are likely to be better scoring than those further away. Which means the adoption of a “replace worst in cluster if new average point is better” policy instead of the randomised replacement policy only accelerates the convergence. Sadly these results do not persist if the optimum is randomly relocated off the origin, or indeed is unfortunate enough to be found outside the initial cloud of points.

3.4.3 Dangers in optimisation : Not the No Free Lunch

The “No Free Lunch” theorem [93, 94] is a result that holds for certain classes of computation. It can be shown that if search is an ordering of sampling points from the space without replacement, and if the solution could be any of the points in the space, then all functions that search by ordering the points behave the same when evaluated over all possible searches (orderings) in the space. What is not so frequently understood about the no free lunch theorem is that we are not always interested in improving performance in all possible searches.

```
function exploit (vector point){
    // copy the current point and save the old score
    temp = copy(point)
    oldScore = fitness(temp)
    // select a random dimension of the point
    dimension = random integer modulo number of dimensions
    counter = 0
    while(counter < number of dimensions){
        // back up the value from that dimension
        oldValue = temp[dimension]
        // calculate the product of the dimensions without
        // the current dimension
        temp[dimension] = 1
        product = product of all dimensions of temp
        // calculate the "repair" needed to bring the
        // product to 0.75
        repair = 0.75 / product
        temp[dimension] = repair
        // check the answer for validity
        if(0 ≤ repair ≤ 10 ∩ fitness(temp) > oldScore){
            return temp
        }else{
            temp[dimension] = oldValue
            index = (index + 1) mod number of dimensions
            counter = counter + 1
        }
    }
    return point
}
```

Figure 3.4: An exploit that tests towards the $\prod_{i=1}^{n}(x_i) > 0.75$ boundary of Keane's function

The no free lunch theorem is often erroneously interpreted to mean "all algorithms perform the same" or "nothing can do better than this". For this reason the no free lunch theorem is sometimes seen as an impediment to optimising search. It has also become somewhat infamous as a "get out of jail free" card for parameter sweeps that fail to make anything other than sporadic improvements. It is common to see the theorem invoked as an incantation against failure when it is seen that none of the settings chosen by an author completely out-performs the other (e.g. [72]) but without first establishing that the theorem holds. In such cases it is simply used as a one-size-fits-all excuse for why an optimisation failed. The failure to locate an operator or operator combination that completely outperforms others on a suite of test functions is not necessarily as a consequence of the no free lunch theorem. It is quite possible that the choices of combinations used were just unfortunate.

Igel and Toussaint [44] believe the conditions required for the no free lunch theorem to hold are quite fragile, and actually not that common outside of artificial combinatorial optimisation problems. It is also common to see the no free lunch theorem name-checked to indicate the awareness of the authors who are going to try to optimise against a subset of functions anyway (e.g. [66, 62]). Justification for why it is believed that this may be plausible is rarely given, however the rate of progress made in optimising against broad ranges of functions seems to indicate that these authors are correct, indeed, we are not optimal yet.

The no free lunch theorem is worth understanding in the context of optimisation of search. We address it several times in this work. First in this section, starting with section 3.4.3.1, we give an informal argument to familiarise the reader with the topic and set the scene for our main argument. Later in chapter 5 we provide a more formal explanation.

3.4.3.1 An optimisation game

The no free lunch theorem can be confusing and for our purposes it is best considered from the following point of view, which is based on the game playing view of the no free lunch theorem proposed by Culberson in [13]. Consider an optimisation task, played against a malicious but honest opponent, which requires the selection of an integer value from a range. Your adversary is obliged to reveal nothing about the manner in which the evaluations are performed and simply responds "Yes" or "No" to your proposals. This is clearly a laborous pursuit. The order in which the values are proposed is actually irrelevant. In the worst case the adversary may actually be playing

the strategy of “Say no until there is only one number left, then say yes”. No strategy can be better than another against such an opponent. Under these circumstances and the no free lunch can clearly be seen to hold - all searches are the same.

The no free lunch result is not dependent on a malicious opponent. In the non-malicious case, the adversary initially chooses a fixed goal at random and is consistent and honest. The search will on average span half the space. Since the goal is chosen at random all search methods are equivalent, the no free lunch theorem holds again.

3.4.3.2 Gradients in optimisation

If the game from section 3.4.3.1 is modified such that the opposition has to reveal slightly more than the “Yes”/“No” minimum of information about the values the problem changes character. If instead of confirming or denying success the opponent says “Closer” or “Further” depending on the relative distance to a fixed goal then the amount of information each answer reveals about the problem changes significantly.

The first difference is that the range of antagonistic strategies available to the opponent is greatly reduced. The malicious opponent no longer has the option of freely relocating the goal anywhere in the space, but instead must remain consistent with previous evaluations given. Eventually the requirement to remain consistent with previous evaluations constrains the options available to the opponent until there is but one option remaining. This is analogous to playing the game of hangman against a malicious opponent. Hangman is a word guessing game, played by two players. The objective is to guess the component letters of a word known to the opponent in fewer than 10 guesses. If the opponent is malicious, every time a letter is guessed by the player, the opponent checks if there is a word that is consistent with all the letters tried so far in which the guessed letter is absent, if so then this word becomes the goal, and the letter is added to those searched and failed. If there is no such word then the opponent is forced to concede and the letter is added to those searched and found to be successful. The finite lexicon and the requirement of consistency between the past and the current guesses creates locality in the hangman search space. All methods of search that utilise this locality are better at this kind of search task than those that simply select from the space.

First let us define some concepts that will recur in our discussion:

Gradient locality: A property of one point is partially or wholly present in its neighbours, thus gradients exist over sub-sections of the space.

Continuity: Gradient locality is to some extent continuous throughout the space.

GC Landscape: An idealised landscape where both gradient locality and continuity hold.

Optimisable: A property of an interaction between a landscape and an optimiser, where without use of an exploit, the optimiser is likely to approximate an optimum sample to within a satisfactory margin.

If a landscape does have gradient locality then "Closer" or "Further" is all that is required to detect the direction of the local optimum. If the landscape has global continuity at some scale then it is possible to detect the direction of the global optimum from a distance; sample based movements are all that are required to move towards the optimum. Continuity in a landscape subsumes the property of causality – that movement by some distance in the domain has a resulting change in evaluation bounded proportional to the magnitude of the move. A completely continuous landscape is also fully differentiable, a property that is inherent in the definition of locality.

Note that by assuming the properties of gradient locality and continuity we define a class of structured landscapes. In doing so we then also accept the possibility of optimising search on such landscapes by creating mechanisms that better exploit the structures which are common between all such landscapes. Sampling and inferring the local gradient could be improved to better infer the direction of the optimum, and mechanisms that detect (dis)continuities in the space could be used to move between the basins of attraction of local optima. An algorithm that ignores or mishandles the information provided by the samples will always be worse on average over GC landscapes than one that guides the search using the information in the structure of the landscape.

Note that we are not claiming that the space need be a totally continuous GC landscape to be optimised. When sampling using a finite number of samples the relative strengths of the locality and continuity properties compose a signal to noise ratio for the samples. Loss of locality creates a misleading sample – one where extrapolation and interpretation under the assumption of locality will not necessarily lead to the optimum. Loss of continuity creates sections of the space which are potentially conflicting when interpreted under the assumption of locality.

The perfect landscape has no discontinuities and strong locality and no samples in the space are misleading. This makes it easier to search, but does not make it trivial, since the range of influence of the global optimum may be so small proportional to the

size of the space that saturating the space sufficiently to detect it costs a lot of samples and the problem remains hard.

To be optimisable, in the sense that proximity to the global optimum is obtainable, there has to be a significant probability of a sample landing in the basin of attraction of the global optimum during the search. The competence of the search method should ensure that when within the basin of attraction the search is directed towards the optimum. For this placement of samples to occur the space must be reasonably continuous relative to the number of samples available. Without this property the search is more likely than not to sample only from sub-optima. The most extreme form of this type of problem is a "needle in a haystack" problem, where there is no information indicating the location of the optimum.

There are also problems which mislead the search, such as Goldberg's deceptive trap functions, where a large proportion of the space is deceptive and all local optima are maximally distant from the global optimum. The deceptive trap functions are well explained in [34]. If there is a very low probability of sampling from the basin of attraction of the global optimum during the search, the problem is not generally optimisable, irrespective of how competent the search mechanism is at following gradients toward local optima. We will demonstrate the performance of random sampling and a basic hill climber on a simple deceptive problem in section 5.3.

In summary, the most important part of understanding no free lunch in function optimisation is the realisation that to be optimisable a landscape has to possess to some degree the properties of locality and continuity: There must be some signal against which the optimisation is expected to proceed. In all other cases the optimisation is degenerate; it is simply blind search. Thus all non-exhaustive searches must assume some degree of gradient locality and continuity.

In justifying the design of the algorithm used in this work (section 6), we will return to the concepts of locality and continuity and will more formally define the reason optimisation is likely to succeed even with the no free lunch theorem. We will show that if a landscape is optimisable in a true sense then these properties hold (at least in part) both macro and microscopically. We also consider the possibility of there being insufficient information in the samples taken to permit detection of the true signal.

3.4.4 Avoiding dangers

Thus to be of use as a basis for future work, we must avoid the three most tempting sirens in the field of real number optimisation: Over-fitting and her close relative Exploitation, and the mistaken use of appeals to the No Free Lunch theorem.

The most common occurrence of over-fitting is in the blind search for synergistic interaction between operators/settings. This is likely to lead to over-fitting, particularly if the suite against which the parameter sweep is performed is the same as that for which the results are announced. A warning sign that over-fitting may be present is if beneficial interactions occur only infrequently, e.g. a parameter setting is better but only for a subset of the problem suite.

All non-enumerative optimisation is based on the detection of features in the problem space. An improvement to an optimisation algorithm – such as removing redundant evaluations – does not affect the generality of the algorithm and is simply an improvement. Optimisation algorithms may also be improved by better implementing search mechanisms, so they are better matched to the features that are known to exist in all optimisable problems, such as gradients and basins of attraction. These results are of interest to all researchers who encounter similar structures in their research.

The author of an algorithm also has the opportunity to include problem specific knowledge which is not part of the problem specification and which could not have been known without prior experimentation. This tends to take the form of limiting the search to a manifold or subsection of the space, or employing operators which probabilistically do the same. Searches performed using an exploit are not found under the normal search processes, and the inclusion of the exploit generally greatly changes the character of the problem. Consequently results located using an exploit tend to tell little about how to improve the fundamental design of the algorithm.

Exploits have a more damaging effect than simply inconveniencing future learning. Because an exploit is problem specific, often to a precise formulation, it only conveys performance improvement over the set of precisely similar problems. On all other problems the exploit is highly detrimental to the search. The point of this objection is that the improvement in performance from employing an exploit comes from the wrong place; instead of locating weakness in the algorithm design and making the search more efficient the problem was made easier. The contribution such an effort makes to improving the quality of the algorithm design is doubtful.

It is useful to clarify that the critical distinction of when exploitation has been used

depends on the objective of the optimisation process. Optimisation of a single problem instance can, by definition not be exploited, since generality of the optimisation result is never claimed. To retain general applicability to a problem class the techniques and knowledge used in the optimisation process must not restrict the applicability of the algorithm to a smaller subset of the problem class. If such a restriction is imposed it must be announced that the optimisation is now targeting a smaller, simpler set of problems. Despite the frequency of authors using such restricting assumptions, the consequential loss of generality is rarely remarked upon [74, 60, 59].

As a rule of thumb, exploitation is present if knowledge from prior experimentation is added to either the problem definition or the search algorithm in such a way as to generate a simplified version of the problem for the algorithm to optimise. To be of interest good performances have to be repeatable over many different functions with the same or very similar algorithm parameter settings: there has to be a quality of generality in the findings. Otherwise, we will fail in our primary task; which is to inform the reader so our findings may be built upon. To be informative the algorithm design should be rationally justified.

The final concern is that the no free lunch theorem can not be used to explain a lack of success unless it is also shown that the preconditions of the theorem hold, it is far more likely that the given algorithm is wasteful and has been poorly optimised than it is that the algorithm is truly squeezed against the performance boundary of no free lunch. Since there is no clear method of casting optimisation of the GC class of landscapes into a no free lunch context, until informed otherwise, we assume immunity and continue to expect significant progress to be obtainable against a range of such landscapes.

In this work we will attempt to avoid over-fitting by using a large variety of problems from the real number optimisation literature. Where necessary we will modify the problem to remove known exploits. We will also discuss why our algorithm design has the properties it does and what features of the space these properties are expected to operate effectively on. We provide evidence for how the design handles different aspects of real number optimisation and we show how significant performance benefits may be seen across the test cases. We deliberately avoid extensive parameter sweeps that “optimise” our design. One of the contributions of this work is the demonstration that these kinds of results are still possible, with a little bit of thought and without finessing the algorithm. The results published in this work are thus unlikely to be the best that they could be; they are however, significantly better than the vast major-

ity of population based real number optimisation methods over significantly greater ranges of problems, which is a strong result and unlikely to be an artifact of the testing methodology.

Chapter 4

Population based real number optimisation.

4.1 Chapter overview

In this chapter we review the population based technologies currently deployed for real number optimisations. We give an overview of their salient features and where possible discuss the motivation behind the different tactics employed.

These techniques all have similar properties, they all use a population of candidates, which are evaluated relative to an objective function. Guided by the result of this evaluation, new candidates are created as modifications or replacements of current candidates. It is hoped that by proceeding in this way the algorithm will progress towards better solutions.

Because the representation of a proposed solution is frequently not convenient to manipulation or representation, a genotype-phenotype representation method is frequently used. The genotype/phenotype terminology is borrowed from biology where it is used to distinguish between a creature's inherited genetic material (its "genotype") and its actual physical expression (its "phenotype"). This is simply a recognition of the fact that an encoding genotype is always expressed relative to an environment, where upon it becomes a particular phenotype. If expressed in different environments the same genotype could render as different phenotypes.

In real number optimisation the use of genotype/phenotype terminology is used for the case where candidates are represented in one form (the genotype) to the optimiser and are translated into another form (the phenotype) for expression and for the purposes of evaluation. For instance if each of the values encoded by the genotype

("genes") are expressed as 12-bit strings, the genotype has the characteristic of being a sequence of some multiple of 12 ones-and-zeros. The optimiser then has the option of working at the bitwise level in the genotype – if the designer believes this may help – which is not available to other genotype encodings. To be evaluated the genotype must be interpreted relative to an encoding (usually Gray [14] or binary encoding) to translate it into a number. The numeric form of the genotype is then expressed relative to the objective function which in turn calculates a result. This result is used to identify solutions to the problem and by comparison with other results indicates the relative merit of the expression encoded in the genotype in question.

The choice of the genotype representation and genotype to phenotype mapping can have significant impact on the performance of the algorithm. Clearly the choice of search operators and encoding create a remapping of the search space by altering the notion of locality under the search operators. Less obviously, the choice of encoding also alters the relative size of the space. A genotype composed of genes with each allele encoded using 16 bits represents an encoding space of 65536 distinct values per allele. Reducing the accuracy of the representation to 12 bits per gene cuts the search space to $\frac{1}{16}^{th}$ of that at 16 bits, a mere 4096 distinct values are representable per allele. For historical reasons many functions in the literature are defined over dimensional domains of range [-5.11,5.12]. The precision a 12 bit encoding offers is increments of approximately 0.0025, whilst most floating point encodings are accurate to increments of 10^{-14} or smaller. The implicit sub-sampling of the landscape by a low precision representation may create a landscape that is radically different. Properties such as the number of optima in the landscape and the relative number of steps required to traverse the space are all reduced.

Some of the results reported in this work used binary encodings of limited precision. Where this has been noted by the original publishing author it is also noted here. This does not mean the results are incomparable, but that the representation space the authors chose to work in was decidedly different. For reasons justified fully in section 6.3 all work by this author avoids working directly with binary encodings and uses genes of double precision floating point numerical values (transparently encoded in 64-bit IEEE 754 floating-point number representation). Thus neither exhausting the allele space nor performing binary operations is possible. The mapping of the encoding of genes to the interpreted gene value is transparent and consistent throughout.

4.2 Pseudo-code

It is occasionally necessary to unambiguously describe algorithmic processes in a codified manner. Rather than use a particular language and risk alienating part of the readership, we use an easy to read pseudo-code. To clarify and give examples of its use, the pseudo-code used in this work has the following structures:

All comments start with the symbols "//" and continue to the end of the line. Functions are defined by the "function" keyword. If the function takes arguments then these are listed in brackets immediately following the function name. The scope of the function is emphasised by indentation and bracketing. Here we declare a function called "example" that takes one integer called "limit" as its argument.

```
function example (int limit){
    ...
}
```

Variable initialisation is written using the equality sign. If the initialisation is more clearly defined in English than pseudo-code then this is used instead. Variables are not typed, and comments are frequently used to disambiguate the possible meanings.

```
// initialise a variable x to the value 10
x = 10

// re-initialise x to the set of even integers
x = the set of even integers
```

A control loop, typically a "while" or a "for" loop, has the following representation. The loop controls are declared in the brackets immediately following the loop type.

```
// a for loop
for (i = 0, i < limit, i++) {
    ...
}

// a while loop
while (true) {
    ...
}
```

It is sometimes preferable to avoid giving unnecessary detail to code sections that are not relevant to the discussion. Where this is the case expressions such as “Create a random individual **x**” will be understood to mean that using whatever creation mechanism is appropriate, the variable **x** now contains a randomly created individual; the meaning of any unspecified terms, e.g. “random” and “individual” being drawn from the default use in the literature or the context of the use.

4.3 The genetic algorithm

The Genetic Algorithm (GA) attributed to Holland [41] (though Goldberg [33, 34] is mainly responsible for popularising the technique) is the most common of all evolutionary algorithms. Based directly on simplified evolutionary dynamics the genetic algorithm has been successfully applied to many real number optimisations. This is in spite of the general warning issued by De Jong [17] that simple genetic algorithms are not good at optimising by searching for ever smaller improvements. De Jong was correct. The canonical genetic algorithm as originally proposed did not include automatic preservation of the best found candidates (as it is not found in biological systems) and consequently could lose good candidates that were not frequently rediscovered. Once the required preservation feature, known in the genetic algorithm community as ‘elitism’ is included, the genetic algorithm does become a function optimiser, iteratively attempting to locate better replacements whilst preserving the best known candidate. This does not mean that the algorithm is guaranteed to reach the optimum. Since the current best candidate may not be within a traversable distance of the global best solution, it is sometimes beneficial to the algorithm’s overall progress to abandon the current best candidate if doing so releases the population from being trapped in a local optimum.

4.3.1 The canonical genetic algorithm

The general form of the genetic algorithm is as given in the following pseudo-code.

```
//Pseudo-code of the simple genetic algorithm
function GASearch (){
    Generate a population of N individuals at random.

    While (Still have time left){
```

```
            for (i = 0; i < population size; i++){
                Select a candidate
                Put candidate into breeding population
            }

            for (i = 0; i < population size - 1; i+=2){
                Breed members i, i+1 of breeding population
                Mutate the offspring
                Put offspring into next population
            }

            Replace current population with next population
            Check for termination conditions
        }
    }
```

The canonical genetic algorithm is characterised by the use of a fitness proportional representation when selecting candidates for breeding and by the use of simple one-point crossover and mutation rates proportional to the length of the chromosome. Fitness proportional representation awards breeding opportunities proportionally to the ratio of the candidate's fitness against the average population fitness. Single point crossover simply selects a location of the genotype representation and exchanges all material after that point with the other parent.

4.3.1.1 Problems with the canonical genetic algorithm

The canonical genetic algorithm is somewhat of an antique, and suffers from significant design flaws. The fitness proportional representation can grant high likelihood of breeding to only one individual, particularly if that individual is the first to exploit a property of the problem and consequently outperforms the rest by a large margin. The problem with this type of response to improvement is that it is largely myopic. Populations quickly become overwhelmed by close relatives of the first candidate to show significant improvement. The population based search deteriorates to a cluster of hill climbers as the population diversity collapses and crossover is no longer able to exploit significant differences between candidates.

Ranking and tournament based selection are the most commonly proposed replacements for the flawed fitness proportional selection, and instead of using the scale of performance differences between candidates simply use a ranking system, relieving the problems associated with large fitness differences in selection. Ranking selection awards selection opportunities proportional to the position of the candidate in the ordered ranked population. Tournament selection selects the best candidate from a small group (usually 2) selected at random from the population. It is known that the tournament selection and ranking selection are essentially the same [35], but of the two tournament selection is much more frequently used. Both techniques rely on obscuring the magnitude of relative fitness differences between candidates to protect the population from being swamped by large successes.

Single point crossover, whilst superficially similar to the biological inspiration of the genetic algorithm is hopelessly flawed for the vast majority of problems. The biggest problem with single point crossover is that it does not allow equal mixing of the alleles of the genotype. Since the probability of being exchanged increases as you approach the middle of the chromosome the crossover driven exploration of the genotype space is not fairly distributed. This is particularly excruciating if the genotype contains linkage between non-consecutive genes since it is impossible for a single application of single point crossover to successfully hybridise two candidates if they have overlapping gene sequences. Uniform crossover is now widely regarded as the standard default crossover operator; in uniform crossover alleles are selected from either parent entirely at random. This ensures a thorough mixing of alleles during the crossover and promotes more even handed sampling. Incestuous crossover preserves sub-populations and crossover between different sub-populations encourages exploration and allows for the possibility of successful hybridisation of alleles with complicated linkage patterns.

4.3.1.2 Real number optimising genetic algorithms

Genetic algorithms specifically designed to operate on real number optimisation problems have a number of modifications to allow them to search more effectively. All such algorithms use real number encodings. Binary encodings are obsolete in this field due to the discontinuities such an encoding creates in the space. Because the lion's share of the performance of genetic algorithms is attributed to the effects of the crossover operator (in the genetic algorithm community mutation alone is considered only a hill climber) great research effort has been expended to create new and better real parameter crossover methods.

Crossover operators used in real number optimisation have three distinct types. The first type are direct analogues of the uniform crossover and exchange whole allele values between genotypes. This type of crossover is a monotonically decreasing hypercube operator, either maintaining or decreasing the allele space spanned by the genotypes each time it is applied. Over long periods this convergence of alleles leads to loss of diversity and effective cessation of the search. Algorithms using this type of search have the unfortunate property of having no method of exploring outside the hypercube of alleles currently contained by the genotypes and are consequently dependent upon mutation to provide a mechanism for exploratory progress. Since the mutation operator in independence is essentially a hill-climber, this combination of operators has a very low probability of successfully exploring complicated space beyond the boundary of the initial population's hypercube.

The second type of operator uses an arithmetic [59] or geometric [60] based blending function to combine the values of the parents. In arithmetic crossover, two values for an allele c_1, c_2 are merged according to the rule $c'_1 = a.c_1 + (1-a).c_2, c'_2 = (1-a).c_1 + a.c_2$ where $a \in [0,1]$ is either a constant or related to the stage of the search. Geometric operators work in exactly the same manner but instead have a merging rule $c'_1 = c_1^{\omega} + c_2^{1-\omega}, c'_2 = c_1^{1-\omega} + c_2^{\omega}$. Notice that both these variants remain monotonically decreasing hypercube operators.

The expansion of merging crossover operators to merge values selected from neighbourhoods following a uniform, exponential or fuzzy (bi-modal triangular) probability distribution gives the final class of homogeneous crossover operators. The offspring are created by random selection from the relevant distribution. This class of crossover operators has the common potential (also in geometrical crossover) to explore beyond the hyperspace of the samples, and has been found to be beneficial to the reliability of the algorithm.

With the realisation that one crossover operator may not have all the properties that are desired, heterogeneous crossover operators, which are composite operators in which one offspring is created by one operator and the other offspring is created by another operator, have been investigated. Heterogeneous crossover operators have some of the best results against common benchmarking functions of any genetic algorithm based real number optimiser so far published. It is reasoned that the strengths and deficiencies of one operator are being traded against the strengths and deficiencies of the other, resulting in a more balanced and capable search.

4.4 Differential evolution

Differential evolution is a search strategy that is strongly reminiscent of the genetic algorithm structure. It differs from the real number optimising genetic algorithm in the way in which it performs recombination. Differential evolution uses a monotonic improvement based replacement strategy, where candidates are only replaced if a fitter alternative is found. The recombination operator uses information from multiple candidates and is typically defined using four parents. The recombination mechanism is very similar to geometric crossover. A new candidate is created by mixing the original candidate with a vector created from the weighted difference vector of two randomly selected vectors and one other vector selected from the population. That is $\vec{x}' = \vec{x} \otimes (\vec{w} + F(\vec{y} - \vec{z}))$, where F represents the weighted rescale of the difference vector and $\otimes$ represents the uniform crossover operator. Zelinka in [99] has a good graphical overview of the differential evolution crossover mechanism and an interesting discussion on the causes of search failure in small populations. As reported by Hansen [38] the differential evolution mechanism fails to handle rotations of the problem space, however, Vesterstrøm [86] reports several extremely strong results using the differential evolution mechanism. The algorithm has interesting properties in spite of its allegedly weak sampling.

4.5 Evolutionary strategies

An evolutionary strategy is a generational mutation based evolutionary algorithm. Two different forms are distinguished based on how the intermediate population is formed. A $(\mu+\lambda)$ strategy has a population of size μ which from which it generates λ offspring and then selects the next generation of μ members by selecting the μ best of the $\mu+\lambda$ candidates. A (μ,λ) strategy on the other hand generates λ offspring and then selects the next generation of μ members by choosing the μ best from the λ offspring only. A third notation occasionally seen; $(\mu/\rho+\lambda)$ or $(\mu/\rho,\lambda)$ denotes that the evolutionary strategy is using recombination over ρ parents in creation of at least part of the λ offspring.

On landscapes with many local optima it is considered that the (μ,λ) strategy is generally better performing, the ability to lose well performing candidates allowing a temporary worsening of the algorithm sufficient to escape a local optimum. In 1993 Bäck in [4] considered the (μ,λ) to be the state of the art in evolutionary strategies, an

opinion that is still current today.

Evolutionary strategies use two basic types of recombination, discrete and intermediate, each with global and two-parent variants. Discrete recombination selects the allele value from one parent or the other. Intermediate recombination alters the current allele value by some factor of the difference between the parent alleles. Global recombination allows the reselection of new parents from the population for each allele, two-parent recombination uses the same parents for all allele operations [24].

Each individual in an evolution strategy is usually a pair of vectors; the first vector $\vec{\alpha}$ represents the domain values and is termed the object vector, the second vector $\vec{\beta}$ represents the deviations of the Gaussian mutation operator applied to $\vec{\alpha}$. The individual is capable of not only adapting its position within the search space, but also the mutation rates to which it will be exposed. Mutation modifies $\vec{\beta}$ according to $\beta_i' = \beta_i \cdot exp(\tau' N(0,1) + \tau N_i(0,1))$ and $\vec{\alpha}$ according to $\alpha_i' = \alpha_i + \beta_i' N_i(0,1)$, where $N(0,1)$ is function returning a normally distributed random variable with expectation 0 and standard deviation 1. $N_i(0,1)$ indicates the random value is re-sampled for each i. The values $\tau' = (2n)^{-1/2}$, and $tau = (2\sqrt{n})^{-1/2}$ are the "learning rates" of the algorithm. The detail of this explanation is due to Eiben [24] who cites Bäck [3] as the original author.

It has become common to calculate the covariance matrix, which can be decomposed by its eigenvalues to give directions for the elliptical mutation sampling. This becomes clear if it is understood that an evolution strategy with the same adaptive mutation rate for all values of the object vector explores a space consisting of a hypersphere. Using distinct mutation rates for each value in the object vector creates a hyperellipse, and calculating the covariance matrix allows the hyperellipse to be rotated. This is a powerful mechanism, and the evolutionary strategies that use it (e.g. CMA-ES [38]) record some of the best results on many real number optimisation functions. Rechenberg [68] developed the initial theory of the evolutionary strategies and proposed an optimal mutation rate for the $(1+1)$ strategy; the "$\frac{1}{5}$ success rule" where mutation intensity should be increased if the historic success probability over recent generations was greater than $\frac{1}{5}$ and decreased if it was lower.

4.6 Evolutionary programming

Evolutionary programming is a mutation only based strategy that uses, to borrow from the evolutionary strategy terminology, a $(\mu+\mu)$ replacement strategy. Like evolution-

ary strategies the domain values are maintained as a vector with a separate variance vector, permitting self adaption of the mutation rates. The object vector $\vec{\alpha}$ is modified by the variance vector $\vec{\beta}$ as follows; $\alpha_i' = \alpha_i + \sqrt{\beta_i} \cdot N_i(0,1)$ and $\beta_i' = \beta_i + \sqrt{\gamma\beta_i} \cdot N_i(0,1)$. Here γ is a parameter that ensures the variance remains positive; if a mutation should render a variance of zero or smaller, then γ is set to a small positive value.

As pointed out by Bäck in [4], both evolutionary strategies and evolutionary programming share the same purpose in the modification to the variance vector, however due to the implementation differences they have distinct sampling properties. Selection in evolutionary programming is performed by 2μ tournaments between n candidates randomly selected from the μ parents and μ mutated offspring, the best μ forming the next population.

4.7 Particle Swarm Optimisation

The Particle Swarm Optimisation algorithm is due to Eberhart and Kennedy [21, 50]. Particle swarm optimisation represents the state of the search by a population of points in the search space. The points are considered particles, and at any time t have a position vector p_t and velocity vector v_t. The search progresses by moving particles to a new position by addition of the old position and the velocity vector. $p_{t+1} = p_t + v_t$. To guide the search, the particle retains the best point from its search history p_{best} and also has access to g_{best}, the best point from the global search history. This information is used when the velocity vector of each particle is updated every time step. The next velocity v_{t+1} is a combination of the previous velocity v_t, the vector to g_{best} and the vector to p_{best}, $v_{t+1} = \omega.v_t + random(0,\phi_1)(p_{best} - p_t) + random(0,\phi_2)(g_{best} - p_t)$, where ω is an 'inertia' weighting altering the relative significance of the previous velocity, ϕ_1, ϕ_2 control the relative contributions of the p_{best} and g_{best} respectively and $random(x,y)$ is a function that uniformly randomly returns a value in the range $[x,y]$. Velocity vectors are normally limited in magnitude. Initially each particle is assigned a random position and velocity. As the algorithm progresses the population converges towards the best found values. To assist in this process it is common to change the inertia values during the run [22].

Particle swarm optimisation algorithms have generated a lot of literature regarding the correct parameterisation on different landscapes. Multi-modal landscapes cause particular concern because the influence of only one best found optimum can lead to extreme convergence and loss of search ability. Consequently the simple particle

swarm optimisation algorithm has been elaborated by several patches. Clerc [9] suggests the additional evaluation of one or several centre of mass "queen" particles, and in [11] describes "tribes" – cliques of information sharing particles – between which information is shared. The design of tribes also borrows from the field of evolutionary computation in using a particle generation method which is capable of performing removal and replacement of particles in a tribe.

Significantly contributing to its popularity, the population size required by particle swarm optimisation is consistently smaller than that of most other algorithms for optimisation of the same problems. Shi in [78] shows the particle swarm optimisation mechanism is not overly sensitive to the population sizing.

In [22, 23] Eberhart and Shi show how the use of a constriction factor can improve performance by reducing the velocity each generation. This constriction factor helps combat the poor convergence properties of the particle swarm optimisation algorithm and improves results by focusing search effort on promising areas of the space. Convergence towards the end of the search is a desirable property for any algorithm with purely probabilistic local search, since increasing the sampling density raises the probability of landing close to the local optimum. Controlling convergence to balance the collapse in population diversity against the rate of approach to the optimum is very difficult.

Each particle in the population of a particle swarm optimiser has a primitive memory. The best position reached by the particle during the search is stored, and the short term movement history of the particle is implicitly recorded in the current velocity. The swarm also has access to the position of the best particle in the swarm, and, through it, is influenced by the global best position discovered in the search.

Particle swarm algorithms are amenable to modification of the communication topology. By creating small world local networks [49, 11] of particles which share influence, the algorithm designer hopes to create conditions in which local knowledge is correctly balanced against global exploitation. Kennedy reported that overall, a simple Von Neumann neighbourhood where connections are based on a two dimensional square lattice served the best and the star topology where all individuals are connected to the same individual performed badly. Kennedy attributes the performance differences between the topologies to the rapidity of convergence on early best solutions. The various topologies, with differing rates of communication, are in some way balancing the convergence rates of the algorithm, in much the same way that selection intensity is considered to control the behaviour of the genetic algorithms [6].

4.8 Ant algorithms

The family of ant algorithms, originating with the work of Dorigo [19], is a hybrid of reinforcement learning and random search. Using the analogy of foraging ants laying down pheromone trails and automatically discovering the shortest path to objectives, the algorithm probabilistically searches variants of the best solution(s).

Ant algorithms are frequently explained in terms of swarm intelligence, however, as Clerc showed in [10] the ant algorithm is actually only a series of local searches. Ant algorithms have primarily been applied to discrete combinatorial optimisation problems, where "the shortest path" analogy is easily maintained, the path being analogous to an allocation decision, and the length being the total assignment cost. The application of ant algorithms to real number optimisation problems has received little attention, partly due to difficulty of creating pheromone update rules that are successful at searching real number spaces.

Despite the difficulties, Socha in [83] made progress in producing an ant algorithm with reasonable performance on low dimensional problems. Extending the ant algorithm to higher dimensions is complicated, and we know of no ant based algorithm which is competitive over larger real number optimisation problems. One of the difficulties with extending the ant based search is it has a strong commitment to incremental solution development, whereas some problems are considerably easier if more non-linear operations are possible.

4.9 Hybrid mechanisms

Recently the population based search community has started to examine in more detail the concepts of hybridising techniques from one or more previously distinct fields. For instance when building the "swarm algorithm framework for numerical optimization" Xie [95] hybridises search methods from differential evolution and particle swarm and even tests but does not deploy a neural network controller. Xie reports some of the better results on constrained optimisation using this strategy. Yong in [97] uses simulated annealing to control the replacement policy in an evolutionary strategy, and Parsopoulos [65] uses differential evolution to tune particle swarm optimisation. The combination of mechanisms as complementary search techniques is promising. We will also use mechanisms inspired from several search mechanisms. For our purposes we are uncertain as to the utility of using one technique to parameterise another, since

this requires repeated evaluations and is in conflict with our general desire to minimise evaluations.

4.10 Our design process

The literature though diverse and difficult to homogenise on a conceptual level gives several indicators as to the properties that should be considered in the design of real number optimisers. We briefly review the range of properties available to the designer and relate them to the task of designing a population based optimiser for a general multi-agent parameterisation problem. Wherever possible we should avoid creating fragile mechanisms that require significant parameterisation. Full details of the design are given in chapter 6.

4.10.1 Representation

The use of a binary encoding (binary genetic algorithms) or a real number representation (almost all other algorithms) alters the range of operations that are possible, and to a certain extent may alter the complexity of the landscape detected through the sampling. We desire accuracy from our representation, and speed from our optimiser. The known mechanisms for manipulating high accuracy binary encodings are comparatively slow, requiring more iterations than lower accuracy encodings to traverse the same distances. It is notable that there are no binary encodings of comparable accuracy of the real number representations currently in competitive use. The desire to reduce the number of evaluations used in the search whilst maintaining accuracy indicates we are likely to be using a real number representation. The same constraint requires that the population size used in the work be as small as possible whilst still providing reasonable performance.

4.10.2 Modeling precision

Evaluation functions which are slow or expensive to calculate can be approximated by the use of a lower precision model [45] or by inference from previous related evaluations [73]. Typically instead of using the full system for every evaluation, the low precision model is used instead. Sufficiently good results found using the low precision model may then be re-evaluated precisely using the full model. Ideally the model should be cheap and fast to compute, but these factors alone are insufficient. The model

must be as accurate as possible, and typically must be built from only a few genuine samples of the space. The model should be optimistic, it should not under value results and thereby accidently exclude true optima. Conversely the function should not be overly generous either. To be of use in evaluating the space the low precision function has to direct the search towards interesting areas and the creation of false optima in the low precision function that are not in the real function will waste evaluation effort. Increasing the fidelity of the approximation is a challenge in its own right.

We are unable to simplify the specification of the multi-agent system to obtain an adequate approximate model because long term behaviour exhibited by the multi-agent system may be a product of any of the interactions of any of the agents in the system. Instead we keep simulation of the system feasible through control of the number of agents in the model, manipulating both the numbers of agents and the number of interactions simulated to speed the evaluation process. Once we have proven the basic concepts of the design process we then extend our experimental scale and evaluate over larger and more complicated multi-agent systems for the full evaluation of the optimisation process.

4.10.3 Initialisation and clustering

Generally the entire search space is available for initialisation. Some algorithms (such as adaptive cluster covering) use clustering to iteratively narrow the region that is searched. The initialisation range, and the mechanism of narrowing (if any) significantly alter the search. Narrowing mechanisms must balance the rate at which the algorithm is capable of detecting interesting points with the rate the space is narrowed. Clearly this gets very difficult in landscapes for which sampling reveals separate areas of promising candidates – an arbitrary decision must be made to concentrate the resources on only one. In general we expect the landscapes over which we are optimising to be multi-modal; we must then consider mechanisms that can prevent the loss of interesting points. Since the concept of an interesting point is one which is future directed, (a point is only interesting if it leads somewhere) such a decision can not be made on the basis of the current point's value, but by an evaluation of the point's potential for movement. We need a method of evaluating when a point has ceased to be likely to progress.

4.10.4 Mechanics of movement

The search may be defined over sections of hyperplanes (stochastic tunneling), vertex sampling of hypercubes (genetic algorithms), velocity vectors of particles with momentum (particle swarm) or implicit gradient estimation (evolutionary strategies and similar). Each brings a separate definition of locality to the search: many are defined over narrow ranges of the space, most are complementary. As far as possible, we wish to take representative techniques from each major movement class. For ease of understanding the interactions we are interested in the simplest mechanisms capable of performing, though this almost certainly reduces the algorithm's overall performance. More complicated mechanisms may be introduced in future research.

Cooling strategies and other mechanisms for encouraging convergence are effective if correctly calibrated to the search landscape. Auto-adaptive mechanisms relieve the necessity of using a cooling strategy, and the additional burden of calculating the correct parameterisation of the cooling strategy is unattractive.

4.10.5 Replacement and longevity

The selection of which individuals persist in the population, either by deterministic methods as in the evolutionary strategies and differential evolution, or by probabilistic methods as preferred in genetic algorithms, alters the algorithm's sensitivity to population diversity and alters the time span in which search mechanisms must be applied. If a good candidate is expected to be lost from the population after a certain number of iterations, it is beneficial to ensure the sample has been subjected to sufficient trials within this period to have had a chance of improving and or disseminating its information.

Using a deterministic replacement policy ensures that at all times the candidates in the population are in some way relevant and trustworthy, having "earned" their position through competition. The population is then a model of the points from the sampled landscape that are considered relevant by the replacement policy. If the replacement policy is well chosen and the sampling has been sufficient, the points are representative of significant points in their immediate locale. Points refined through local search sample the local region around single points; merit is then best assessed over this range. Crossover on the other hand generally samples between points in the population, and the merit of a crossover operation is best judged relative to the population. It is then desirable to have a replacement policy that evaluates the replacement of points relative

to the local sampled topology, the locality of the point being defined by the range of the operator that creates it.

4.10.6 Auto-adaptivity

Most mechanisms are auto-adaptive in at least some of their parameters (constriction in particle swarm, mutation rates in evolutionary strategies and evolutionary programming). Which mechanisms are capable of being adapted in a meaningful way depends strongly on the operator. Simple auto-adaptive methods such as constriction or mutation of variance rates in evolutionary strategies are effective. If there is sufficient understanding of the mechanism to permit their use, more complicated adaptive behaviours such as calculation of the covariance matrix (evolutionary strategies) bring performance benefits.

4.11 In summary

At this point in this work it is beneficial to review what has been discussed, where this fits into the strategic development of the algorithm and what has yet to be delivered. We have given details of both the multi-agent systems we will be using in this work and shown how their parameterisation is translated into a real number optimisation task. We have also discussed optimisation of real number functions, and we have highlighted and discussed significant aspects of various established and successfully employed algorithms.

We will use the conceptual foundations lain in these chapters when we create our algorithm for optimisation of the multi-agent systems. To decrease the algorithm development time to a feasible time horizon we will be using surrogate functions. First though we must show that this choice of development path and in particular the proposed use of surrogate functions does not have any inherent flaws. Superficially the use of surrogate functions is forbidden by the no free lunch theorem. Thus before proceeding with the development, we must show that we have addressed the no free lunch theorem.

In the forthcoming chapters, we first examine the no free lunch theorem and show that our development strategy will not contradict the theorem. We then create and test our algorithm using surrogate functions, and lastly test the algorithm on the multi-agent system.

Chapter 5

Formally dodging the no free lunch

5.1 Preliminaries

The no free lunch theorem [93, 94] states that when viewed over all possible objectives, any search which is an ordering of samples taken without replacement from a finite domain will produce the same mean performance. What the no free lunch theorem had shown was that, in general, optimisation is a zero sum game. The corollary that all improvement on one subset of functions is necessarily paid for by reduced performance on another is a significant finding in computer science. The no free lunch theorem has been extended and improved since its conception, but the basic result itself is sound.

One critical feature of the original no free lunch theorem is the assumption that one is interested in optimising over all possible objectives. Clearly, if the objective "select domain value A first" is in the set of all possible objectives, then so are all the related objectives "select domain value A second", "select domain value A third", etc, and "select domain value A last". The same is true for all the possible domain values. Clearly no fixed ordering can simultaneously produce the desired domain value first, last and all places inbetween.

This observation leads to a refinement of the no free lunch theorem. It has been proven in [75] that the smallest subset of functions on which the no free lunch theorem holds is the permutation closure of a single function. Igel [44] proved the no free lunch theorem holds for a uniform probability distribution on a set of functions if and only if the set of functions is closed under permutation. A set of functions $F = \{f : X \to Y\}$ is declared closed under permutation if for any function $f \in F$ and any permutation $\pi : X \to X$ the function $f o \pi$ is also in F. This ensures that for every function f in F, F contains, amongst others, its inverse. This is necessary and sufficient for the no

free lunch theorem to hold. Igel [44] showed that the fraction of uniformly distributed functions that are actually closed under permutation is vanishingly small.

As remarked by Droste [20], consideration of the space of all functions is useless in a practical sense, the fragment of the space of all functions that can be represented or evaluated is tiny, the space of all functions is simply not realisable.

The no free lunch theorem is of direct relevance to this work on two accounts. The main thrust of the thesis applies a technology developed on one set of problems to another set of problems. The no free lunch theorem prohibits general improvement over the set of all problems thus requiring proof that the surrogate problems used in the development are related to the multi-agent problems. For the same reason, the no free lunch theorem only permits improvement of algorithm performance against a set of benchmark problems if none of the current results are optimal. A naive interpretation of the no free lunch theorem might be used to infer that no general improvement can be made against the set of benchmark problems, but this is (as we shall see) equivalent to claiming pareto optimality of the current performances.

5.2 Structure in the benchmarking problems

The following discussion is heavily dependent upon the techniques described by Christensen in [8] and considers the distribution of new samples taken from the domain. The reader may also wish to familiarise themselves with the original works by Wolpert [93, 94], and the notable works by Culberson [13], Droste [20], English[28, 29, 26, 27], Igel [44] and Schumacher [75].

We wish to demonstrate that the set of problems spanned by the frequently used benchmark problems is one that exhibits a definite structure and consequently is a true subset of the set of all problems. That the set of random functions is not represented in the benchmarking literature is obvious; each value in the co-domain of the function is derived only from manipulation of the value(s) in the domain. This excludes random functions; the total complexity of the outputs of the system is bounded by the complexity of the system and the inputs.

Examination of the problem definitions reveals that the traditional benchmark functions are basically different perturbations of low order functions. Consider the Rastrigin function which in every dimension is a composition of a sinusoidal perturbation and a bowl shaped second degree function. The dominant feature of the Rastrigin function is the hyper-dimensional bowl shaped structure; the sinusoidal wave is essentially

an embellishment, creating local maxima and minima. Similar properties are evident in the Keane function, which is a sinusoidal perturbation of a slope dominated by the product term in the denominator.

This general gradient property is trivially present in De Jong's sphere function, and examination of any of the benchmarks in the suite will reveal similar properties, though the reader may find it easier to consider the sphere function in the remainder of the discussion. These gradient structures enforce a general trend in the co-domain values of the space. Samples taken from the domains of these functions have higher than average probability of being of above average evaluation in the co-domain if they are close to other domain samples of above average co-domain value. Since the function is not random, the maximum deviation a sample $x + \varepsilon$ may have from the next nearest sample x is less than the diameter of the range. This skews the mean of the evaluation; the sampled neighbourhood in the proximity of any given point is structured. How far this structure maintains is not actually important for this discussion, but it certainly is a calculable metric, for instance the minimum distance that the structure extends is calculable from the minimum movement in the domain required for the accumulated maximum deviation to span the range.

5.2.1 Structure and prior information

Recall the optimisation game from section 3.4.3.1, where the objective is to guess an integer from a range of possible values. As mentioned in section 3.4.3.2 if instead of only confirming or denying success the opponent says "Closer" or "Further" depending on the relative distance to the fixed goal then the amount of information each answer reveals about the problem changes significantly. The utility of the information transferred from the opponent to the player is dependent upon a common understanding of the property of "distance".

The existence of a structure between the domain values and the evaluation values necessarily implies local continuity in the function, and the function permutations that maintain this type of local continuity define an optimisation class. This type of mapping is easily imagined by adding an constant offset mapping to each of the domain values, creating a new landscape which is identical but relocated. The rotation mappings are also in this class, as are any homomorphic set operations.

The set of possible homomorphous mappings of the benchmark functions is the smallest set that we can operate over whilst resisting the charge of overspecialisation.

The set of mappings defines a set of attendant properties of benchmark functions; properties which are observed in all homomorphic remappings of a benchmark function, and represent the structure common to the entire set of related benchmark instances.

In this discussion we are considering the challenge faced by an algorithm designer who seeks to improve performance on a set of benchmark problems. The entire set of such properties represents the collective types of structures that are present in the benchmarking problems. It is these properties that an optimiser would require as prior information to be able to perfectly optimise all possible variants of the set of benchmarks. We will consider them the set of benchmark properties $P_{benchmark}$. For the purposes of this discussion, the set of benchmark properties defines the universe of problems in which the algorithm will optimise. The astute reader may have noticed that optimisation over the set of homomorphisms constitutes the basis for another interpretation of the no free lunch theorem, which we will discuss in detail in section 5.4

The difference between the prior information encoded into the optimiser's design P_{prior} and the actual set of properties of the problem set $P_{benchmark}$ defines an information gap, leading to properties incorrectly used by the algorithm P_{error}:

$$P_{error} = \{p : p \in P_{prior} \notin P_{benchmark}\} \cap \{p : p \in P_{benchmark} \notin P_{prior}\}$$

Examples of such properties could include known limits on the gradients in the space, correlations between subsets of parameters or known limits on the extent of noise or the function distribution that generated the noise in the landscape (e.g. a Gaussian noise distribution is a common event). Other properties could refer to the maximum and minimum feature sizes represented in the space or to the nature and maximum degree of the generating function.

Attempts to exploit properties that are not present in the problems, or failure to exploit properties that are in the problems results in a reduced relative performance. The information gap, and the proportion of the search volume that it accounts for, gives the degree of differentiation in performance between an algorithm equipped with prior information P_{prior} and an optimal search over this set of problems.

Correctly identifying the presence of properties in the problem cases, and incorporating this knowledge in the algorithm's prior knowledge P_{prior} increases the proportion of possible search instances that are informed. As long as the information is relevant to the optimisation, increasing the proportion of the search that is informed reduces the proportion of the search which is blindly probabilistic, and so raises the average performance of any algorithm so informed. Whilst there remains an informa-

tion gap, optimisation of a search algorithm is then possible over a set of problems by reducing the gap. The unlikely situation of a zero response gap would imply the algorithm wasted no information, and was optimal.

The problem set of common benchmark problems and their homomorphisms defines a subset of the space of possible functions which have certain properties. By improving the use of these properties the creation of better optimisation methods is possible. A possible source for improvement in performance is better matching of the algorithm to properties of the optimisable problem structures. We have shown the role of the no free lunch theorem in the face of such improvements: we "pay" for the improved performance on problems that have optimisable characteristics by increasing the degree of appropriate structure in our search. We do not breach the no free lunch theorem since by making these commitments we may now perform worse on problems that do not have these properties. The properties we attempt to encode as prior information are properties that are capable of being interpreted as signaling the location of optima. Using these properties is the only method that can be used for optimisation faster than random search. All informed search methods attempt to use this source of information. Identifying and encoding these properties with less noise, so they are more effectively tested when the search is conducted, creates better quality search over all problems that have these properties. We have then formally discharged our obligations to the no free lunch theorem where improving performance relative to the set of benchmark problems are concerned.

5.3 Application to a novel problem

Unfortunately, the fact an optimiser is well suited to a particular sub-set of problems is no guarantee that the optimiser will be well suited to any other problem drawn at random from the set of all problems; the "design a good optimiser and apply it" strategy is potentially flawed. In the following section we consider the limits of the repercussions and discuss what options this leaves a designer who is faced with exactly this task.

An algorithm that is equipped with a particular form of prior information is only better than random search for those problems with matching structure; over all problems that are not in this class the algorithm will necessarily perform worse. We are motivated by the need to demonstrate as far as possible that the set of problems spanned by the benchmark problems used in the algorithm design are related to the set of problems spanned by the multi-agent problems.

One may be tempted to use arguments from history: that the benchmark problems are selected because they represent particular problems that are relevant in the real number optimisation domain. This reasoning is flawed precisely because the benchmarks are chosen from past problems that have been encountered, and say nothing about the future problems that may occur.

So what are the properties of the agent problem that can be known a-priori? Some properties are obvious, for instance it is not a random system. True random systems are excluded by the bounds on the complexity – a system creates outputs only as complex as the inputs to the system and the system itself. Even a non-random system may have high degrees of dynamical instability, a sensitivity to initial conditions that renders the long term prediction of the system futile. It is then reasonable to assume the existence of neighbourhoods within the space. What form these neighbourhoods take is however unknown – they may not be exploitable. Application of an optimiser to an unknown domain is then a gamble. There is the reassurance that if any of the type of structures that are encoded into the prior knowledge of the optimiser are also present in the problem then the optimiser will have at least some improved ability. Equally if these properties are not present the optimiser can be no better than random search.

To know whether the optimisation methodology is well founded we must consider what type of landscapes could be present, and show that these are similar to the properties we encoded into the prior information of the optimiser.

There are landscapes with a general gradient towards the optimum, general gradient away from the optimum, and with no general gradient. The first is the class of problems on which we intend to base our optimiser, the second is the class of deceptive problems, and the latter class consists of needle in a haystack problems, plateau problems and random landscapes. We must consider the possibility that the multi-agent optimisation problem is either deceptive or random. We know that the true random landscapes are not part of the multi-agent problem, but the pseudo-random landscapes may be. A pseudo-random landscape that holds none of the properties of the benchmark problems has no properties of local self-similarity.

If we consider optimising on the landscapes with general gradient towards the optimum, it is clear that the landscape retains some properties which will be similar to those present in the benchmark optimisation set; better points are on average located near similar points. We should thus expect our optimiser to perform better than random search in this case. The second set – of deceptive problems – contains structures identical to the first set with the difference that they lead away from the optimum. Fol-

lowing local improvements will in the long-term result in a lower overall achievement than ignoring the gradients and using random sampling would.

To empirically validate that the optimisation landscape is fully deceptive is hard, since it requires knowledge of the optimum and the topology of the space. The majority of deceptive spaces are only partially deceptive, such that some proportion of the space has a gradient that leads towards a sub-optimal point and the remainder is either gradient neutral or leads towards the true optima. This means that the gradient information may be useful some of the time, but not always, and distinguishing the two instances requires an exhaustive knowledge of the structures in the space. Only the fully deceptive problems require that the gradient information be ignored at all times; all partially deceptive problems are honest for the fragment that is non-deceptive.

Even in a deceptive landscape “better than random” optimisation, by which we mean a better average performance than that of a random sampling algorithm, is not ruled out. A basically competent hill climbing type algorithm may be expected to out perform random search if the probability of the hill climber landing in the basin of attraction of the global optimum is sufficiently high, and the probability of a random sample landing in the global optimum is sufficiently low.

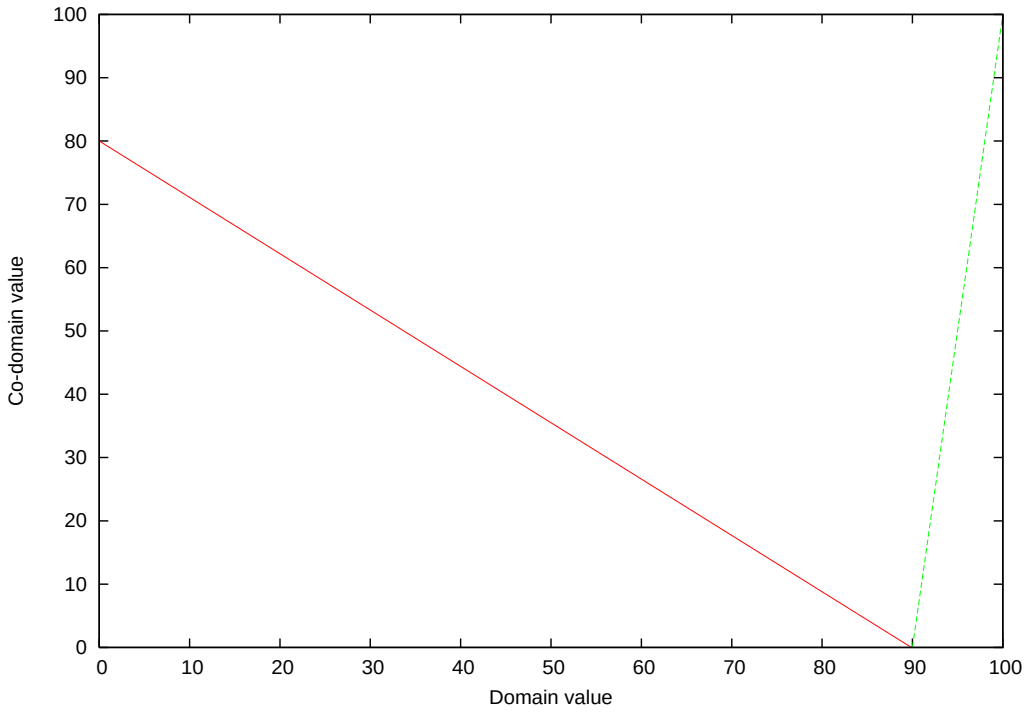

Figure 5.1: A simple one dimensional deceptive trap function. The objective is to maximise the co-domain value. Only 10% of the space is actually structured in a manner that leads to the optimum.

The relationship between structure, deception and random search success probability may seem counter-intuitive; figure 5.1 helps to clarify the situation. The figure shows a simple one dimensional real valued deceptive trap function. There are two optima in the space, a deceptive optimum at $x = 0, f(x) = 90$ and a true optimum at $x = 100, f(x) = 100$. 90% of the space is deceptive; simple hill climbing in the deceptive space leads towards the optimum at $x = 0$. The remaining 10% of the space is honest and hill climbing here leads towards the global optimum at $x = 100$. If the initial samples are placed at random, the samples have a 1 in 10 chance of starting in the honest domain range, and a 9 out of 10 chance of starting in the deceptive domain range. On average we would therefore expect to place 6.57 samples before one were likely to occur in the honest region of the domain. For the sake of simplifying this argument, we imagine the informed search is a very effective hill climbing mechanism that is capable of moving over 1% of the domain in the "uphill" direction (increasing co-domain values) each and every iteration. When the hill climber is incapable of further improvement it randomly restarts. This performance is rather better than one might normally expect, but it will serve the purpose for this demonstration.

The average initial co-domain value in the deceptive range is $x = 45, f(x) = 40$, and the average co-domain value in the honest range is $x = 95, f(x) = 50$. On average, a sample placed in the deceptive range will reach the deceptive optimum after 45 iterations, and a sample placed in the honest domain range will reach the true optimum of $f(x) = 100$ in 5 iterations. We are now in a position to calculate the expected number of iterations until success of the hill climber. We expect to perform 6.57 restarts, each with an average cost of 45 iterations, before using a final 5 iterations to locate the exact optimum. The hill climber is expected to reach the absolute optimum after 300.65 iterations.

If the goal of the search is to reach a termination value of $x \geq 99, f(x) \geq 90$, our hill climber is expected to take 299.65 iterations. Randomly sampling from the space is expected to need 68.97 samples before a sample greater than or equal to 99 is taken from the domain range. Clearly under these circumstances the random sampling will on average be successful before the hill climber. However if the termination accuracy is slightly tightened, to $f(x) \geq 99.9$ then the random sampling is expected to sample from the termination domain range after some 692.8 iterations. Under these circumstances the simple hill climber will out perform random sampling even though the hill climbing mechanism is being deceived the majority of the time.

To confirm that the optimisation is well based, we are therefore obliged to use ran-

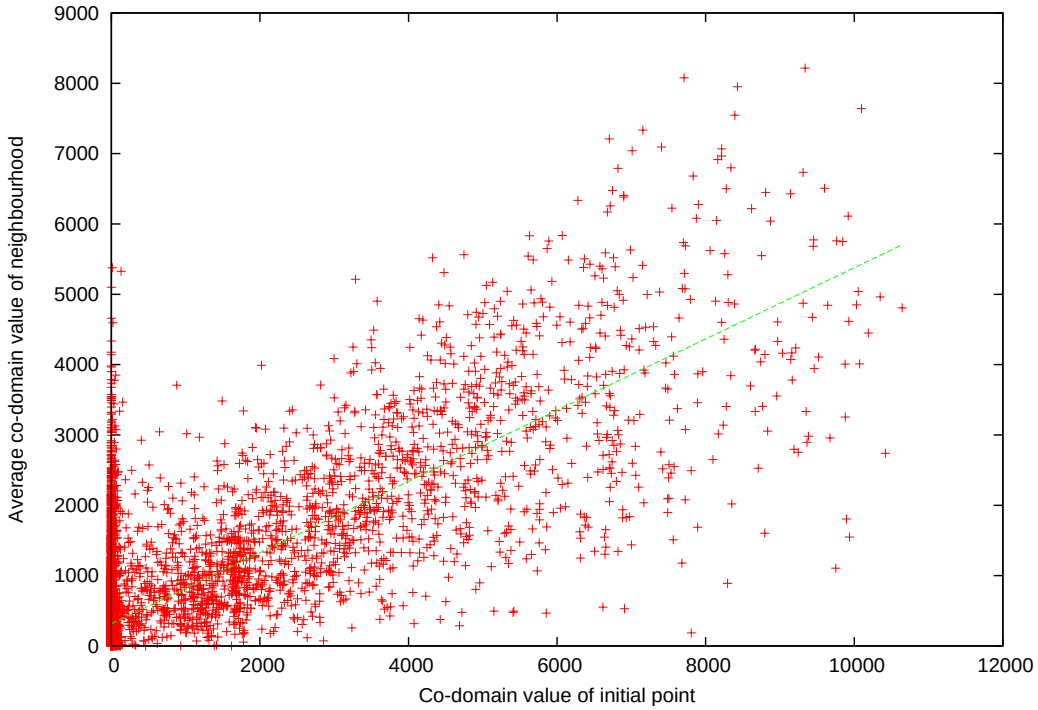

Figure 5.2: Self similarity in an instance of the proof of concept multi-agent problem, estimated using a 10% standard deviation normal Gaussian distribution. The y-axis values are the average of the co-domain values of the cluster of domain points sampled from the hyperspherical distribution centered on the initial point. Correlations in this graph indicate structure between the domain and co-domain values.

dom sampling of the multi-agent problem to demonstrate the existence of neighbourhoods within examples from the multi-agent domain. It is necessary to implement the multi-agent problem (which is defined fully in chapter 2) for this validation. We can not directly measure the structure in the space without an exhaustive enumeration, and so we use self-correlation over Gaussian distributed samples to provide an estimate: we select a point at random and then compare the co-domain value of the point with the average co-domain value of some of its neighbours in the domain. The average fitness of these sampled points is dependent on the topology of the space and the radius of the Gaussian distribution and we can use this to characterise the level of continuity in the space. For obvious reasons, we consider only local Gaussian distributions; a distribution over the entire space fails to be meaningful. Strong continuity in neighbourhood properties results in correlation for closely distributed samples, resulting in the mean fitness of the cluster being similar to that of the centre point. This measure reveals the presence of self similarity, and allows us to identify whether particular instances of the multi-agent optimisation landscape are pseudo-random or structured.

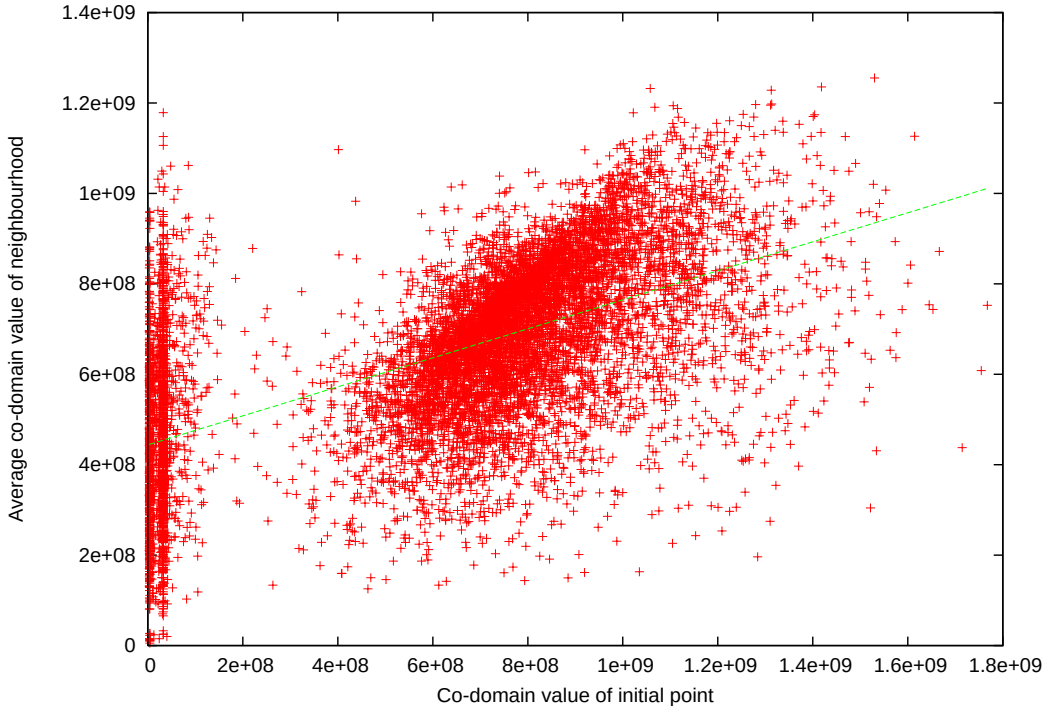

Figure 5.3: Self similarity in an instance of the SADDE multi-agent problem, estimated using a 10% standard deviation normal Gaussian distribution. The y-axis values are the average of the co-domain values of the cluster of domain points sampled from the hyperspherical distribution centered on the initial point. Correlations in this graph indicate structure between the domain and co-domain values.

A self-correlation plot from the proof of concept multi-agent domain is given in figure 5.2. This example is from a system configured with 4 agents trading for 1000 rounds. A self correlation plot from the SADDE multi-agent system is given in figure 5.3. The SADDE system, here shown configured with only 12 agents is lightly structured. The majority of configurations of the SADDE system have non-zero scoring trading behaviours. A hill-climber pursuing the average trend in the configurations will in general improve performance. Relative to the value of the initially sampled point the comparative low average value of the Gaussian clustered co-domain values indicates that either there is no general structure or that the 10% sampling resolution is too large, effectively stepping over local continuity structures.

Each point in the plot shows the relative fitness of a randomly selected point and the average fitness of its Gaussian sampled neighbourhoods. These samples were collected using a Gaussian sample of standard deviation spanning 10% of the parameter space, and expectation of zero. The line overlain on the plot indicates the actual regression value. A perfect correlation would indicate that all the points sampled had exactly the same performance as their neighbouring points.

To make comparison easier, figure 5.4 shows the same measure calculated for the highly continuous 2 dimensional De Jong sphere problem. We have also included the same measure from a pseudo-random function with no self similarity in figure 5.5. As you would expect, there is no correlation. The value of the initial sample has no effect on the the average value of the sampled neighbourhood.

High degrees of self similarity indicate structure within the problem. The SADDE system does not exhibit the same kind of continuity in its self similarity plot as is present in the other examples. The plot itself is divided into two distinct groupings, a low valued group composed of those that have initial co-domain sample values of 20,000,000 or less and a high valued group containing those that have higher initial co-domain sample values.

In the low valued group, the large number of points with a near zero valued initial sample and a much higher valued average co-domain sample indicates that there are large areas of the domain space with very low co-domain values. Within the sample radius of these low valued initial domain samples there are instances which are of relatively high value, thus raising the average of the sampled cluster.

Of those initial samples with high co-domain values, there is a correlation between the increasing co-domain value in the initial sample and the average co-domain value of the sampled neighbourhood. The evidence presented informs us that parts of the

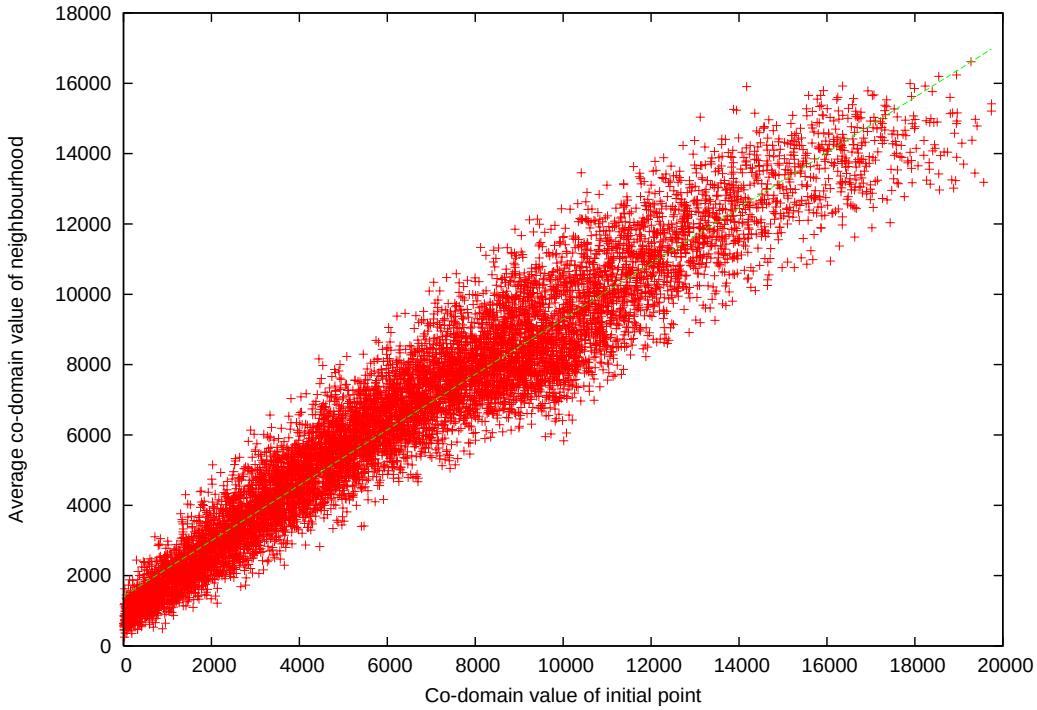

Figure 5.4: Self similarity in the 2 dimensional De Jong's sphere problem, estimated using a 10% standard deviation normal Gaussian distribution. The y-axis values are the average of the co-domain values of the cluster of domain points sampled from the hyperspherical distribution centered on the initial point. Correlations in this graph indicate structure between the domain and co-domain values.

SADDE optimisation domain are structured, but it is impossible to foretell at this stage whether this structure is one which we are capable of utilising.

5.3.1 Use of restarts

Without an ability to identify the global optimum we are never in a position to differentiate between the basin of attraction of the global optimum and that of a deceptive optimum. All partially deceptive landscapes have optimisable segments but their location is unknown.

In order to increase the probability of an algorithm sampling from the optimisable segments, random seeding of the initial search points is required. We must balance this requirement for restarting the search with the optimising behaviour of the algorithm, and so should only invoke random restart once an optimum has been reached. Once an optimum has been reached, the search should restart in a random location, mimicking random search and maximising the probability of the restarted search landing within

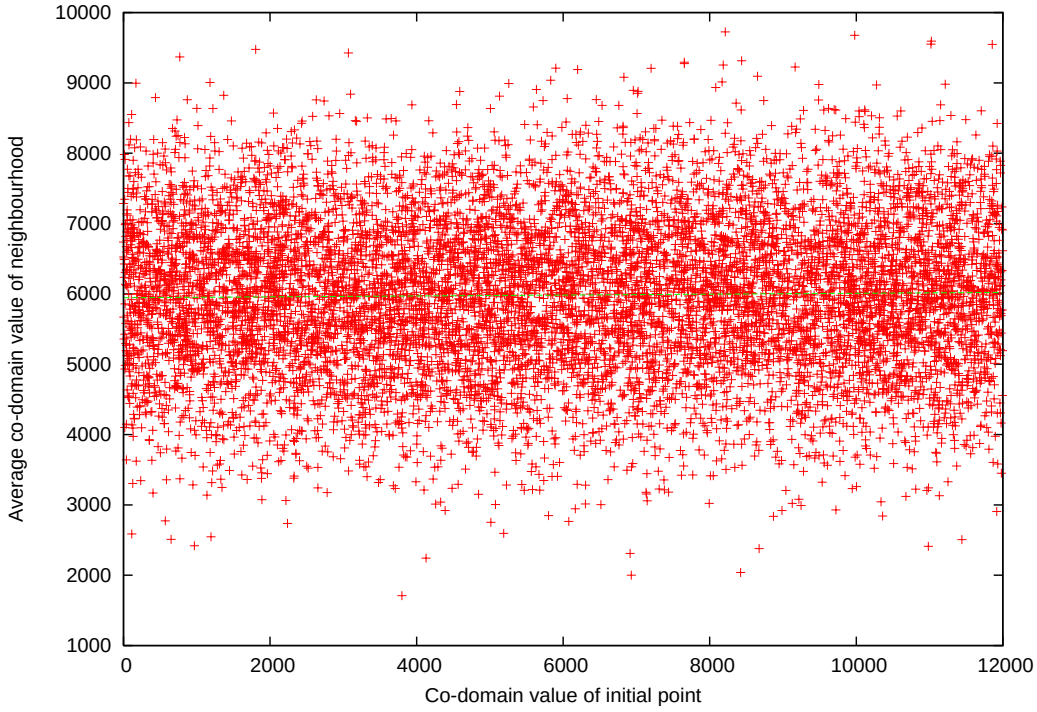

Figure 5.5: Self similarity in a random landscape, estimated using a 10% standard deviation normal Gaussian distribution. The y-axis values are the average of the co-domain values of the cluster of domain points sampled from the hyperspherical distribution centered on the initial point. Correlations in this graph indicate structure between the domain and co-domain values.

an optimisable neighbourhood of the global optimum. Identifying an optimum is not trivial, and in this work we make the decision to restart on an approximation. A compromise algorithm is desired, that locates optima through structure based search when optimisation is possible, and reverts to random search for re-seeding search points when optimisation not possible. This type of algorithm that restarts stalled searches has increased chances of searching within the basin of attraction of the global optimum than an algorithm that does not restart searches.

5.4 No Free Lunch : reprise

We are now in a position to clarify our statement made in section 3.4.2, regarding the use of the no free lunch theorem in regard to justifying aspects of an algorithm's performance.

Consider a hypothetical algorithm that is designed to perform on several distinct landscape types. If this algorithm's performance is claimed as optimal, it is by definition pareto optimal over the expected set of landscapes. Pareto optimality means any further "improvement" in performance on one landscape necessarily implies a loss of performance on at least one other. An optimal algorithm is inherently subject to a no free lunch with regard to future modification, so a pareto optimality claim implies a no free lunch. Since any non-trivial algorithm that has an optimal behaviour on a set of landscape types actually has pareto optimal behaviour, it may not be improved other than altering this behaviour along the pareto front. If an algorithm is subject to no free lunch with regard to a set of problems, then there exist no modifications that could improve the algorithm over the entire set of problems. Such an algorithm is therefore pareto optimal over the set of problems, so a no free lunch implies a pareto optimal behaviour. We can then see that no free lunch theorem is nothing more than a restatement of pareto optimality in terms of problem sets. This means appeals to the no free lunch theorem to explain an algorithm's performance over a subset of problems are also invoking a claim of pareto optimality over the set of problems. This consequence is rarely undersood.

5.5 Summary

The multi-agent problems are novel, the properties of the optimisation space are currently unknown. The landscape properties must be estimated, since the extensive

search required to correctly classify the landscape of the problem is too expensive to consider. For the same reason the algorithm development can not be performed on the actual multi-agent problems and instead development is performed on a set of surrogate functions. The no free lunch theorem threatens to strike twice in this work. The first occasion is when we wish to justify the expected retention of search ability when we change focus from the surrogate set of problems to the multi-agent problems. The second time is when we suggest that general improvement over the set of benchmark problems should be possible. In both cases we avoid contradicting the no free lunch theorem by appealing to the notion of structure in search.

In general, if we expect the optimisation to be successful relative to a random search, it is necessary to show that there are topological features that are true of the general set of problems and may be encoded in the prior information of the algorithm.

These properties of self-similarity have been shown for the set of benchmark problems, which being fundamentally geometric functions exhibit strong self-similarity and structure. We hope to use encodings of this structure to guide the search. Most evolutionary algorithm design is based on interpretations of naturally occurring systems and uses one or two search mechanisms to explore the possible structures. We believe that creating mechanisms to better explore structures known to exist in optimisable landscapes will result in better encodings of prior information. Implicitly we assume that current algorithms are not optimal encoders of this information. It follows that we expect performance on the benchmark problems to continue to improve and not be bound by the no free lunch theorem until optimality of the information used in the decision process can be proven.

We have shown that the multi-agent problem is not random, nor pseudo-random. It is computationally impractical to determine whether or not the space is deceptive, requiring an exhaustive enumeration of the parameter space. The evidence does indicate some degree of self-similarity, which is sufficient to provide structure to at least a subsection of the space.

The change from the surrogate problems to the multi-agent problems risked being an unsupported extrapolation. The arguments in this chapter illustrate that there is a form of self-similarity structure in the multi-agent domains. The creation of mechanisms designed to use this form of structure and then testing the algorithm design against similarly structured surrogate problems is not unreasonable.

Whilst other development courses may have been preferable, the nature of sampling the multi-agent problem left us with little choice. Under the circumstances, the

choices made here are the best that can be expected to be achieved. The algorithm is unlikely to be a bad starting point for optimising the multi-agent problem, and performance on the problem is expected to improve as more samples of the landscapes become available and initial operator choices can be refined.

Chapter 6

Algorithm design

In this chapter we combine different concepts, each with their own terminologies and natural thought model. We also describe and visualise concepts regarding search mechanisms and objective functions using analogies and terminology that designed to convey the properties that are considered relevant to our objective. Many terms that are used here and in the literature are interchangeable: a sample from the search space when viewed in terms of a population is an individual, and in a competition is a candidate, all of which are terms representing the same thing – a point in the domain of the objective function. The objective function may be referred to as a fitness, implying it has been applied to a point, or as a landscape, defined over some implicit ordering of samples. Naturally we normally define the ordering by Euclidean distance of points in the domain. Under certain circumstances, when the operators involved are well defined, we may discuss the "local" points. These are the sets of points that are within a low number of operations of the currently considered point or points, and may not be geometrically local at all. We may also use properties of these visualisations in descriptions; slopes, edges and ridges, peaks and troughs, etc. are terms relating to landscapes, and thus are really shorthand for the appropriate properties in the differential calculus of the hyper-dimensional surface.

The design of optimisers is a dark art. Research is conducted under hunches and prejudices about what properties an optimiser will have to sample in order to be able to detect relevant information from the landscapes on which it is expected to operate. These mechanisms describe and attempt to manipulate different properties of the space. It is hoped that by selecting mechanisms that operate over mutually complementary resolutions and that cover the major search methods reported in the literature we will produce an algorithm that is in some sense more complete. The sampling methods

employed are only a facet of the design. How they are combined, how sampling effort is divided and how points are maintained are significant aspects of the design. Following the establishment of the necessary definitions we approach the description of the design in two major sections, section 6.7 where we describe the devices of exploration, and section 6.8 where we describe the logic behind the relative distribution of effort and the population management mechanisms.

6.1 Optimisable landscapes

Truly random landscapes offer no hope for optimisation. The optimum may be located anywhere in the space, and there is zero correlation between landscape features at all scales. On such landscapes all algorithms that do not repeat samples perform equally on average. There is no pattern to learn, so no pattern exploiting optimisation may occur. Introducing a level of correlation between the landscape features creates a signal in the space. The strength of this signal must be higher than the minimum signal strength detectable by the optimiser for the landscape to be searched and the optimum found in a consistently successful manner. This is more complicated if the signal is only local since the probability of detecting the signal whilst sampling is reduced.

Optimisable landscapes by their definition must contain a pattern that leads to the optimum. A perfectly optimisable landscape would consist only of the pattern to be optimised, and the pattern would be clearly discernible irrespective of how the initial samples were taken. Such perfect landscapes do not exist. Even the simplest optimisable landscape, the n-dimensional slope is not disambiguated clearly by all sample placements. It appears to be flat if all the samples are placed in a line perpendicular to the optimum. There is thus the (very slight) possibility of failing to sample adequately even in the simplest of landscapes.

6.1.1 The requirement of detectable gradients

Locality is always defined “in the eyes of the algorithm”. A local region in a space is all those points which may be reached in a low number of applications of the search algorithm’s operators. Conversely a distant region is one that is likely to be reached only by many applications of the search algorithms operators. Locality is then representative of the order in which the algorithm is likely to search the points from the space. Different search operators generate different point sequences from the space

and consequently have different local regions.

In section 3.4.3.2 we introduced the concept of gradient locality and of continuity in the property of gradient locality. Without gradient locality there is no local landscape at all, without continuity at any resolution there is no global landscape. Thus there is a continuum of landscapes, each with greater or lesser degree of locality and continuity. An algorithm may overcome discontinuities and contradictions in the landscape by lucky sampling. Repeatedly overcoming such difficulties is in general unlikely since it amounts to blind search in a space, which is potentially populated with deceptive attractors.

Exploring different landscapes may be viewed as detecting a signal – the function dictating the location of the optimum – against a background of noise – misleading and contradictory sample points. Certain landscapes are more misleading than others, with corresponding increase in search difficulty. If the signal to noise ratio falls below that which the algorithm is capable of detecting with high probability, the algorithm begins to fail. Where one algorithm fails, it is possible that a different algorithm using better placement of samples might, without prejudicing the problem, be able to extract more of the signal. In which case, all other things being equal it should have proportionally higher success.

The efficiency of the search then depends upon the detection and efficient verification of indicative gradients in the space. Locality is a required property, there must be a sampling resolution at which such gradients are resolvable. Continuity is also a required property, for without it there is no large scale indicator as to where the global optimum may be.

6.1.2 Improving sampling of optimisers

A perfect optimiser would invest only the number of samples required to disambiguate the pattern that defines the location of the optimum and then jump directly to the solution. Even in perfect circumstances there are distinct differences in the complexities of different landscapes. Higher order functions take more points to disambiguate than lower order functions. In general the dimensionality of the pattern is not known in advance, and so placement of the samples is less than optimal even when examining a perfect landscape. A real optimiser does not know the rules governing the landscape, and instead infers the pattern in terms of general trends indicated by the samples collected. It is thus dependent upon two properties that are not universally true; that the

sampled trends are representative of the real trends in the space and that the real trends lead to the optimum.

Under the assumption that they will be applied to locally optimisable landscapes. optimisers may be improved by reducing the rate at which they "waste" samples. The definition of a "wasted sample" needs to be precise. There are samples that are placed that do not give any information with respect to the current model of the space. These are wasted. There are samples that are placed in a non-optimal manner but still reveal properties of the space. These are not wasted samples, but perhaps could be better placed to reveal more information.

All optimisers will place samples that are not directly leading to the optimum and which when further information becomes available are seen to have been badly placed. In general when establishing the dimensionality and direction of gradients in the space, it is impossible to consistently place perfect samples without prior knowledge. There are however ways in which the samples may be placed which change the amount of information gained by placing that sample. The improvement of real number optimisation algorithms occurs by improving the placement of samples relative to those already taken to maximise the information yielded per sample. By relying on the assumption of generalised locality the algorithm may then focus on the best samples and iteratively map the space until reaching the optimum. However, the locality assumption may not hold over the entire landscape, or alternatively the initial sample may have been unlucky and missed the basin of attraction of the optimum. In either case, even an algorithm placing locally perfect samples will proceed directly to optimise to a non-global optimum, where-upon the algorithm will stall. This is a result of being deceived by early samples. Real number optimisation may be improved by improving the way in which the effort of optimisation is divided between all promising localities instead of concentrating on just one. Some techniques exist which attempt to do just that (niching and crowding mechanisms in evolutionary search, the multiple queens technique from particle swarm optimisation). The difficulty is in obtaining the correct balance between attempting to improve the best found so far and seeking or improving other localities.

6.2 Rationale behind the design

The objective of the design is to create an optimiser that is effective and efficient for problem domains that share a common set of properties; gradients and approximate

global continuity, which we believe are useful in optimisation.

We desire a mechanism with strong local search capabilities, that can sample both probabilistically and topologically over as many scales of the representation space as is practical. Population convergence concentrates search effort in a small section of the space. If we successfully obtain a strong local search that can reliably detect gradients, we can relax the requirement for population convergence by depending on the quality of the local search to finalise local movements towards local optima. Relaxing the population convergence requirement allows the continual recycling of exhausted samples from the population, which in turn allows a smaller population to sample more points than would otherwise be the case. If there is a limit to the number of evaluations of the objective function that are available, the distribution of those evaluations amongst the members of the population can be critical to the success of the search.

A smaller population of samples gives us a better ratio of evaluations of the objective function per sample, and in the case of a long drawn out pursuit through a thread-like fraction of the space (i.e. the Rosenbrock function) will, all other things being equal, get closer to the optimum than a larger population. We require a design that can dispose of points irrespective of their quality to release the algorithm from local optima. We desire that the mechanism for selecting points for removal be based on the evidence that the point has ceased to progress, and it should not be random or fitness based. By evaluating the evidence that a point is unlikely to be further improved before permitting its deletion from the population we reduce the incidence of culling low quality but promising points.

We intend to use different mechanisms to explore different substructures in the search space. Clearly the operators should be selected to work in a complementary fashion, but further they should be implemented to be independent modules that share only the necessary resources. Modularity in the implementation of the mechanisms, such that interactions are minimised between the separate operators, allows the algorithm to be modified by "plugging in" new and better mechanisms as they become identified. Modularity also has a potential cost. Some algorithms are dependent upon complex interactions between their components to function, and, in creating a Frankenstinian algorithm created by cutting up other techniques, we risk losing the "magic ingredient". If the modular design is successful however, the algorithm is successful solely because of the composite of the abilities of the modules. There is no magic ingredient. This facilitates future analysis of the algorithm performance, and offers the promise of the identification and replacement of under-performing modules.

6.2.1 Non-optimality of our design

In reading the following it will become apparent that there are many aspects of the algorithm design that are un-optimised. All major design decisions are based on some form of reasoning, the validity of which is judged en masse by the performance of the algorithm. Thus there is no individual proof of validity to show each component of the algorithm is correctly configured from the huge space of possible configurations : Consideration of the huge parameter space and the possible interactions between operators foretold of a potential lifetime spent parameterising, deterrent enough to delay the task until proven necessary. In this regard, the adequacy of the components is considered as testified by the performance of the algorithm as an entity, which, if successful neatly sidesteps the problems of experimental control of operator interactions and avoids the need for further experiment. Optimal settings for the design decisions would only have been investigated if the initial experiments proved inconclusive or unsatisfactory. At the risk ruining the suspense, and the punch-line of chapter 8 where we discuss the algorithm's performance on the test suite of surrogate evaluation functions, the algorithm actually performs as hoped, and further refinement of the search process was considered unnecessary to achieve the objective of the current work. Further development may of course be pursued and will almost certainly bring benefits.

6.3 Representation

The original genetic algorithm work used binary encodings, and many researchers still publish work based on their use (Whitley foremost). To represent a vector as a binary string a particular resolution and encoding has to be chosen. Whitley has shown that under certain circumstances a Gray encoding is more effective than a traditional binary encoding. Whitley also has results that show that changing between distant Gray encodings may help the search avoid getting stuck in local optima.

Another problem with binary representation is that it requires the user to have some knowledge of the precision to which a representation will be required to resolve the landscape. This property remains unaltered during the run. Higher precision representations require longer genotypes and consequently are slower to converge on the optimum.

Without knowledge of how fast the algorithm converges on the optimum, the user can not know what precision they may use to represent the values and still approach

the optimum in under a certain number of evaluations. The lack of accuracy permitted by low level binary representations can be alarming. The 10 dimensional Griewank function as defined over the conventional domain range is a large space : $[-600, 600]^{10}$. The precision of the representation is the size of the represented space divided by the number of unique representable values, (i.e. for a binary string of n bits the precision is given by $(value_{max} - value_{min})/(2^n - 1)$). Homaifir in [43] also uses this definition of precision. If the representation chosen for domain values of the Griewank function uses 10 bits per dimension (e.g. Whitley and Barbulescu [91, 5]), the genome is 100 bits and can resolve each dimension down to no better precision than increments of 1.173. Doubling the length of the genome to a 20-bit per value representation improves the resolution limit on the standard Griewank function to 0.00114, but is reported to significantly slow the progress of the search.

The alternative is to use a real number encoding where the algorithm manipulates the values, not the representation of the values. Clearly the values themselves are encoded in binary at some point, but in the age of arbitrary precision maths, this need not be relevant to the algorithm. Using a real valued encoding has several advantages over a binary encoding. A real number encoding avoids the problem of choosing and modifying representations and removes one layer of potential complexity from the algorithm design. The interpretation of the values represented is direct, modifications to the values have obvious Euclidean analogues making visualisation easier, and the representation preserves the property of strong causality at least between the representation and the genotype. Weak causality is the principle that the same initial circumstances lead to the same outcome, and is natural in these systems. Strong causality is the principle that small changes in initial circumstances cause small changes in the outcomes, and is not guaranteed between phenotype and phenotype evaluation. It should be mentioned that the use of an arbitrary precision representation necessitates the use of either an iteration limit or a termination precision for the search, where the search is terminated after a certain accuracy is reached. This guards against the situation of the true optimum being an irrational number, which otherwise would result in non-termination.

When searching a bounded space using a vector based mechanism there is the potential for simple operations like vector addition to exceed the permitted value boundaries. When this occurs, for the sample to continue to have relevance it must be restored to within the normal boundaries of legal space. The mechanism that enforces the boundary can have a very strong impact on the properties of the search. For ex-

ample, enforcing a hard boundary that limits large values by rounding to the breached limit will tend to accumulate samples on the very edge of the space. Ideally the renormalisation mechanism should not alter the distribution of the samples – samples placed uniformly at random in the bounded space should be indistinguishable from samples placed uniformly at random in a larger space that has been renormalised. Simultaneously it is desirable for vector based analysis of the space that relative directions and distances should persist : Two points should have the same difference vector following the renormalisation of one or both of them as they had before. To achieve as close to this ideal as is practicable we consider all spaces toroidal. Renormalisation is simply continuation. This representation is used throughout this work, and several of the operators rely on the properties of the toroid. For the vast majority of functions this creates at least one large discontinuity at the "seam" of the toroid. There are many other equally plausible mechanisms. One alternative mechanism is to "reflect" the value back into the space. We have not investigated the relative merits of the different renormalisation schemes.

6.4 Initialisation

The ideal initialisation on an unknown landscape is one which gives maximum information from every sample. The placement of the samples should therefore be chosen to give maximum coverage throughout the space. Unfortunately the placement of some number of equidistant points into a high dimensional space is a form of the hypersphere packing problem, and is unsolved. The most common strategy is to initialise each value from the appropriate range independently at random. This is the strategy that is used in this work. It is also possible to initialise at least some of the values at the extremes of the permissible range. Initialisation at the edges of the hyperspace favours search mechanisms that are better at interpolation than extrapolation.

Michalewicz's initialisation from [60] in which candidates are initialised on the boundary of feasible space is an exploit. The knowledge necessary to use this initialisation mechanism is simply not available in the general case. The majority of constrained optimisation problems have more constraints than are active at the global optimum. Using such an exploit under these circumstances necessitates selecting a subset of constraints which are believed to be active at the global optimum – which requires knowing the location of the global optimum – and then initialising feasible candidates : this is not an easy task.

Some authors follow Angeline [2] in using an initialisation routine that deliberately avoids the subspace containing the global optimum. Such an approach greatly favours algorithms that are good at extrapolation and is somewhat suspect in its applicability to real problems, where a lucky initialisation may indeed solve the problem. The majority of published results are initialised over the full domain. We also follow this tradition.

6.5 Population and individual operators

This work uses several distinct operators and methods to search the space of candidates. These operators are of two types, distinguished by their use of other members of the population. Population based operators use information from both the current sample and from samples in the current population of candidates to create new samples. Individual operators use only the point itself to create new samples in independence of other samples in the population. Transient auxiliary samples may be created as references to aid this process.

6.6 Population sizing

Traditionally large populations are considered "de rigueur" in Genetic algorithm research. A large population contributes to the search in several ways.

Firstly, a large population gives a large initial sample, and if large enough allows a probabilistically complete sample of combinations of a certain size. This is a hangover from the dominance of the building block hypothesis, where for recombination to be effective the population had to be appropriately sized to contain with a high probability any given required component. There are several problems with the building block hypothesis in real number optimisation. In real number representations what constitutes a building block is debatable, and further most non-trivial functions have a degree of precision required of each of the variables that the probability of possessing them in an initial population of any size is small.

Secondly, a large population is slower to succumb to dominance by one candidate, allowing for larger sub-populations to form during the run and consequently greater chance of recombinative operators to succeed. This is true even if mechanisms are used to protect sub-populations from loss, since the larger population allows greater diversity within a sub-population.

Thirdly, a large population has a greater palette of individuals in which to try mutations. As a candidate reaches population dominance (through replication or incestuous crossover) the gene sequence it represents is subjected to an increasing degree of search. Large populations also improve the capability of an algorithm to interpolate between samples by increasing the sample density in the hyperspace spanned.

Still, real number genetic algorithms have traditionally had large population sizes, whilst particle swarm methods have tended to use small populations. Shi [78] provides evidence that is interpreted as meaning the performance of particle swarm optimisation is insensitive to population size. This is more or less true, however, they do record better results for larger populations. Unfortunately they also permit a proportionally larger number of function evaluations to be made, so the question of how population size alters particle swarm search performance with a fixed number of function evaluations is unanswered.

There are also specific arguments for keeping a population small. When subject to a function evaluation limit, a large population has proportionally fewer evaluations per candidate. An iterative search making small incremental progressions is therefore likely to get further using a small population than one which must increment a large number of candidates. If a population is too small, the genetic diversity it may contain is very limited. Without protection it will be rapidly overwhelmed by the best candidate and the population based operators will begin to degenerate.

We attempt to obtain the best of both worlds in this work. We use a very small population (20 individuals) to keep the iterations per candidate ratio high allowing each sample to undergo a greater number of the modification/evaluation cycles before exhausting the evaluation resources (please see section 6.2 for an explanation of this concept). We prevent wastage of population space by using mechanisms to prevent straight cloning of candidates in the population. Additionally, when required, since a small population is typically insufficient to provide detailed gradient information, we obtain higher resolution sampling by generation of local points.

6.7 Scales of operation

This algorithm is designed to operate at several distinct levels of optimisation simultaneously. The rationale is simple. Population based techniques such as crossover are very fast at interpolating between candidates, but are ineffective at finessing parameter settings. Local search mechanisms on the other hand fail to resolve enough informa-

tion over the whole landscape to guide search globally. Each particular operator, local search, crossover, centroid location, etc. have a particular resolution at which they are effective. Thus we distinguish the componentry of the algorithm by the resolution at which it is expected to operate. The aim of the design is to explore all plausible types of locality at all relevant resolutions simultaneously. We do not quite manage this (it is quite a tall order) but we do come close. The known shortcomings of the algorithm design are discussed in section 6.9.

6.7.1 Local microscopic techniques

The smallest resolution that a search can be conducted at, under limits of the representation, will be referred to as microscopic optimisation. Techniques which deal with landscapes at this resolution are highly likely to witness continuous gradients. At this resolution, population members are typically much further apart than the desired sampling resolution of the search. Population based techniques are unlikely to detect the required level of detail. The best known mechanisms for optimising single points in the presence of continuous gradients are hill climbers.

Traditional hill climbing techniques operate by making random changes to the representation and then evaluating the result. If the result is equally good or better then the result is kept, if it is worse, it may be kept with a certain probability (simulated annealing) or may be discarded (random ascent hill climber) depending upon the design. The magnitude of the changes made to the representation are either constant throughout the run or are subject to a "cooling strategy" which reduces the size of the movement during the search. For the uninitiated, simulated annealing is a method of randomised search which always makes a considered move if it improves the condition of the search, and will perform a considered move that worsens the condition of the search with a probability proportional to a "temperature". The temperature is lowered throughout the search, and it is proven that with an infinitely small decrement in temperature (and consequently a rather long run time) the search will always converge on the optimum.

There are several aspects of a traditional hill climber that can be improved for operation on real valued spaces. Due to the "coupon collector's paradox" the traditional "random generate and test" mechanism for locating gradients degenerates as the ratio of optimised to optimisable dimensions increases. The magnitude of the changes made also has to be well chosen to efficiently optimise the landscape. A simple cooling

strategy only begins to fine tune towards the end of the search, irrespective of how long the point has been known. Some kind of self adaptive mechanism is then required, one that will allow points that are stationary to explore the fine tuning landscape, and one that allows movement over various resolutions when the landscape permits. See section 3.3.2 and 3.3.3 for more discussion on these topics.

6.7.1.1 Auto-adaptive hill climbing

A codified explanation of the hill climb process described herein follows this description. It is sometimes much easier to read algorithmic concepts in code than English.

The simplest form of hill climbing is to repeat the same vector addition that was known to be effective the last time. We will refer to the current position as x and the direction taken previously as the historic vector $\vec{h}$. We use an auto-adaptive parameter d which defines the distance of sampled points from the current point. Initialisation and the mechanism of auto-adaption of d is dependent upon other properties of the search and so is explained later.

The sample point at the position $h = x + d.\vec{h}$ is the anticipated continuation of the current particle direction in the space. This is very similar to the concept of momentum used in particle swarm optimisation. If better than the current point x we set h as the proposed next point x_{next}. This mechanism is simply exploitative, it is incapable of initiating or changing direction. If the historically predicted point fails to improve upon the current point we need to re-acquire the gradient direction. We use a local scattering of random points to detect the gradient in the local space.

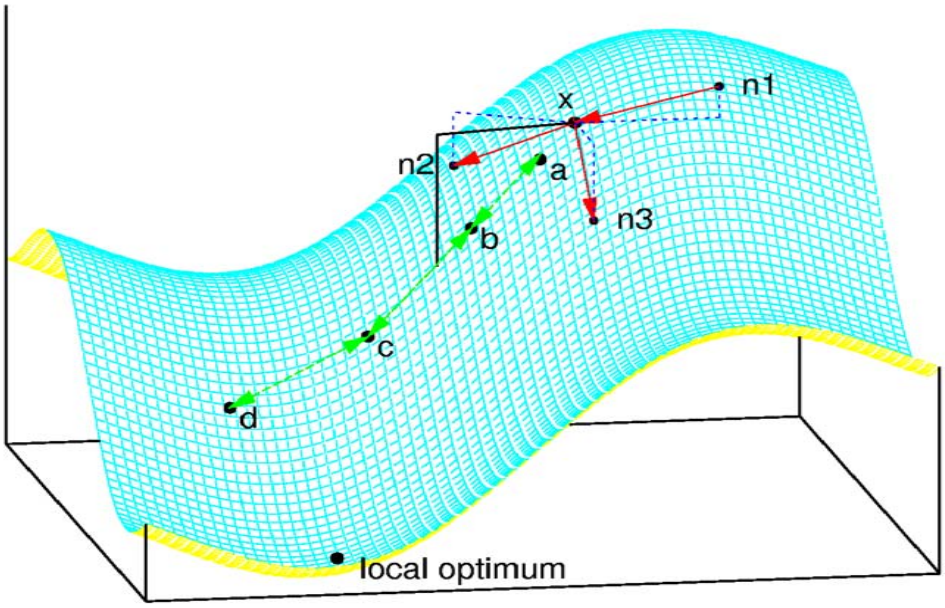

Figure 6.1: The adaptive gradient sampling mechanism of the auto-adaptive hill climber on a minimisation problem. If the historic direction of travel has failed to provide continued progress, re-sampling of the gradient is required. The samples, labeled n1,n2 and n3, are placed at random on the hypersphere represented by a circle. The samples n2 and n3 are better quality than the current sample x, and serve as attractors. Sample n1 is worse quality than the current sample and acts as a deterrent. The sample vectors are summed to create a normalised composite vector, which is then used to place samples in the scalar ranges. The scalar ranges shown in this example are sectioned as follows: a to b 0.5 to 0.75, b to c 0.75 to 1.5, c to d 1.5 to 2.5. One sample is placed in each of the scalar ranges, and if better a better sample is found the point is moved to its position.

A simplified example of the sampling method is illustrated in figure 6.1. Each of the n scattered points, labeled $n1, n2$ and $n3$ in the figure, is positioned randomly on a hypersphere of radius d centered on x, represented by a circle in the figure. To facilitate low dimensional search the hypersphere of $n/2$ of the samples is generated with reduced dimensionality. This is necessary because the default generation mechanism is extremely unlikely to sample low dimensional vectors. This type of movement is necessary for finalising the last dimensions of the search. The mechanism of achieving this is to simply mask some equiprobably selected proportion of the dimensions of the sampled point against the current point x, resulting in a sampled point that differs from

the current point only in the selected dimensions. Each of the n scattered points n_i is compared to the current point x and provides $\delta f(n_i) = f(x) - f(n_i)$ landscape gradient information. Though for simplicity only 3 sampling points are shown in the figure, we used 5 sample points for this work. Thus two points were allocated to searching low-dimensional vectors, and three are allocated to the search of the full dimensional space. Use of more sample points improves the probability of correctly detecting the true gradients in the space. However, this improvement in sampling accuracy comes at a great cost. The hill climbing mechanism is used frequently, and so increasing the number of samples used per hill climbing iteration significantly reduces the number of evaluations that the algorithm may allocate to other forms of search. This is one area of the algorithm that might benefit from auto-adaption, allowing the algorithm to allocate samples relative to the apparent difficulty of the sampling. The best ratio of low dimensional samples to full dimensional samples is also unexplored. For this work it was only necessary that the capability of this type of movement was maintained.

We collate the information gathered from the scattered points to create a composite vector $\vec{c}$. The composite vector $\vec{c}$ is initialised to the zero vector. If $\delta f(x)$ from x to the sampled point n_i is beneficial with regard to the search then the vector $(n_i \vec{-} x)$ is added to the composite vector. This reinforces the composite vector in the direction of the beneficial points. Similarly if $\delta f(x)$ indicates that moving in the direction of n_i may be detrimental to the search the vector is subtracted, thus reinforcing the composite vector in directions away from poor points. In figure 6.1 the point $n1$ is a repulsor, and points $n2$ and $n3$ are attractors. The composite vector $\vec{c}$ is then rescaled to absolute length d and added to the current point to create a sample reference based point p. The sample based point is then evaluated and compared to the current point. If the sample based point p is not better with respect to the search, but one of the random samples r_{best} was better then the construction of the composite vector failed. The point p is then replaced by the best of the random samples r_{best}. Failure to create a consistent composite vector can be caused by unlucky sampling, or the search being located on a saddle point or other region with a zero second order derivative. If none of r_{best}, p, h offer improvement over the current point, the local search has failed and the function will return. In response to this failure, which may be caused by either an incorrect estimation of the gradient or by taking an inappropriate length step, we reduce to one quarter of its value the range at which samples will be taken. The gradient information does not have to be modified because if necessary it will be re-estimated in the next round of hill climbing. We use a factor of one quarter for the reduction because the

scalar ranges are based on an expected reduction in step size of 0.5 times, and an expected increase in step size of 2.0 times. By scaling the radius to one quarter of its former value, we then expect the next round of samples to be placed just within the radius sampled by the failed iteration. Repeated failures then explore a series of concentric rings from the hyperspace, each smaller than the last. Re-establishment of successful sampling restarts the normal scalar based auto-adaptive control of the sample radius.

If one of r_{best}, p, h are better than the current point, then, relative to the current position, we have found a better point, call it x_{next}. The relative direction of x_{next} from the current point indicates the direction of a better sample, and forms the optimisation vector $\vec{z}$. We invest in sampling a further m samples to estimate what scaling of $\vec{z}$ is best. This information is used to auto-adapt the sample radius parameter d. Auto-adaption requires that at least two points be tested, one at a greater distance and one closer. We use three points, based on the three obviously interesting scalings of $\vec{z}$: 0.5, 1.0 and 2.0. Using only these three vector scalings would be clumsy, resulting in a fixed step size scalar adaption. Instead we use randomly distributed scalings in the ranges 0.25 to 0.75, 0.75 to 1.5, and 1.5 to 2.5. Having generated the m scalars, we then generate the corresponding scalar based sample points s_m which if better replace x_{next}. This concludes the active sampling of the hill climbing search. The current point is moved from x to x_{next} and the auto-calibrated sample distance is updated to be the distance between them.

The number of samples used to hill climb is then dependent upon the difficulty the hill climber is having with the gradients in the space. If previous directions are still adequate, then the hill climber uses only one sample to move up hill. Failure of the historic direction indicates the gradient information needs to be refreshed, at a cost of n samples to try to detect the new gradient at the current resolution and m samples to try to detect the changes in resolution.

6.7.1.2 Pseudo-code for hill climbing

```
// adaptively hill climb a point x
function hillclimb(vector x){

    create the next point, x_next, initialise to null

    // test for easy hill climbing
```

```
// exploit the historic movement direction:
if (have a movement history for the point x){
    create a point at extrapolation from historic position, h.
    // if better than current point remember it
    if((maximiser ∩ fitness(h) ≥ fitness(x)) ∪
        (minimiser ∩ fitness(h) ≤ fitness(x))){
        x_next = h
    }
}

// if historic information failed then
// sample the gradient information instead
if (x_next == null){
    create n samples at distance d, with best called r_best
    create composite vector c, initialise to zero vector
    for (each sample){
        // if it is pointing to improvement use as attractor
        if((maximiser ∩ fitness(sample) ≥ fitness(x)) ∪
            ( minimiser ∩ fitness(sample) ≤ fitness(x))){
            c = c + sample;
            // otherwise use as deterrent
        }else{
            c = c - sample
        }
    }
    scale c to absolute length d
    // sample the point suggested by the gradient
    create sample p at point + c

    // select the best point
    x_next = best of p, r_best
}

// test for scaling adjustment
create vector z = x - x_next
```

```
        create m scalars
        for (each scalar){
            create samples s = x + m z
            if((maximiser ∩ fitness(s) ≥ fitness(x_next)) ∪
                (minimiser ∩ fitness(s) ≤ fitness(x_next))){
                x_next = s
            }
        }

        // update the resolution information
        d = distance between x x_next
        // move the point
        x = x_next
        return
    }
```

6.7.1.3 Hill climber objectives

The basic auto-adaptive hill climber is designed to use as few points as possible whilst being able to exploit persistent gradients in the space, detect the need for adaptation of the direction of exploitation and detect the changes in the resolution at which the space may be exploited. The lean use of sampling results not in the strongest hill climber that achieves these properties, but one that is sufficient. With greater investment of samples the hill climber could be made considerably more robust.

The auto-adaptive hill climber is initialised with a sample radius of one twenty thousandth of the maximum distance representable in the space. This is a somewhat arbitrary initialisation value and places heavy reliance on the auto-adaptive mechanism to locate a more profitable resolution.

6.7.2 Local macroscopic techniques

Movement at the microscopic scale is too slow to be used for exploring the space, and since the mechanisms it uses are dependent upon the local continuity of the gradient information, it is vulnerable to misdirection by local discontinuities. Consequently the local search properties available to the algorithm for point-wise exploration of the space need an operator which performs over larger distances. Detection of the gra-

dients, only really feasible in the local space, is expected to be handled by the microscopic techniques. This leaves us with the requirement for point-wise exploration mechanisms capable of searching both the global range of possible points, and the local proximate space. Since the expected range of distances searched by these operators is a significant proportion of the space we refer to them as macroscopic. We consider it unlikely that accurate gradient information is obtainable over these resolutions without large amounts of sampling effort. We thus use techniques that do not rely on gradient sampling but instead are probabilistic operators which sample at random from the permitted range. Such operators have an iteration-wise probability of discovering an improvement defined by the relative volume of better samples from the range, irrespective of the gradients or local complexities of the space.

We use classical genetic mutation techniques to explore the dimensional movements of individual samples. Classical genetic mutation is simply the alteration of one or more values in the representation of the solution, typically by an amount selected from a Gaussian or Cauchy distribution. We use the same method, but we select the mutated value either randomly from the full range of available values or from a Gaussian distribution. Ranging mutation is a powerful local and global method for sampling different dimensional values, and can be used to rapidly approximate aspects of solutions even in the presence of discontinuities. Gaussian mutation tends to be local and is capable of jumping over small gradient discontinuities in the space. Gaussian mutation is an excellent complement to strong gradient based local search precisely because it is not a gradient based method, and an algorithm equipped with both is much harder to trap in a local optimum. Both distributions have advantages, and to choose one over the other prejudices the problems on which the mutation operator will be effective. Instead we use both. We select the mutation distribution by random binary choice.

We recognise that use of full range mutation is unorthodox. Full range mutation allows the algorithm to have a non-zero chance of reaching any point in the space in one step, irrespective of the current condition of the search. Gaussian mutation allows the locality principle to be tested at a moderate scale. The Gaussian distribution places samples with very high probability in the immediate vicinity of the current value, we use this to exploit the mid-ranged locality that may occur in the space. To improve the exploration of lower dimensional moves we select the number of dimensions to mutate from a uniform random distribution. This is necessary because when approaching an optimum, the approach is rarely uniform. Certain dimensions approach or reach their optimal value before others. Consequently the number of dimensions in which it is

probable for a mutation operator to make an improvement decreases.

It is important that the mutation range be sufficient to have a non-zero probability of reaching all the points in the space, a property referred to as Ergodicity. Ergodicity is necessary for ensuring that under mutation alone the algorithm has at all times a non-zero chance of reaching better positions, if they exist. Many algorithms (simulated annealing, genetic algorithms with non-uniform mutation) use a cooling schedule to reduce mutation activity during the search. The reduction in mutation activity or range leads to more localised and exploitative mutations in the later stages and reduces the distance that mutations are likely to move the sampling in the samples that are left. This loss of probable effective range means an algorithm must be close to the optimum in the later stages of the search. Creating an effective cooling strategy is difficult because it requires prediction of the conditions the optimiser will be encountering at all stages of the search. Cooling strategies are typically necessary to allow the algorithm to adjust towards a microscopic search strategy and finesse the best found points. We use a custom hill climbing mechanism to handle the microscopic search detail, and consequently have no requirement for a cooling strategy. The mutation operator thus maintains the desirable ergodic property throughout the search, and we are relieved of the burden of having to predict the search landscape to create the cooling strategy.

The genetic mutation operator is not without its flaws. Consider optimising the unitation value of a bit-string using a bit flipping mutation operator, choosing and setting a bit at random. This operator has a 50% chance of correctly setting a given bit. As more bits are set correctly, the probability of unsetting a correct bit increases, whilst the probability of correctly setting an incorrect bit decreases. The expected time to reach full unitation (known as the MAXONES problem) can be calculated from the inversion of the fundamental matrix of the Markov chain describing the bit transitions, and can be demonstrated to increase exponentially as the number of bits to optimise increases linearly. Figure 6.2 shows the expected number of operations that a bit-flip mutation operator is expected to take to solve the first few instances of the MAXONES problem. Note the log scale in use for the y axis.

In the domain of real number optimisation, a similar condition occurs, where values replace bits and the probability of correctly setting the bit is typically significantly less than 0.5. We recognise therefore that the mutation operator alone, though ergodic, is vulnerable to probabilistic stagnation. To ensure continued progression in the search a more elaborate replacement mechanism must be employed – we only replace candidates with those that are better scoring relative to the objective function. Because the

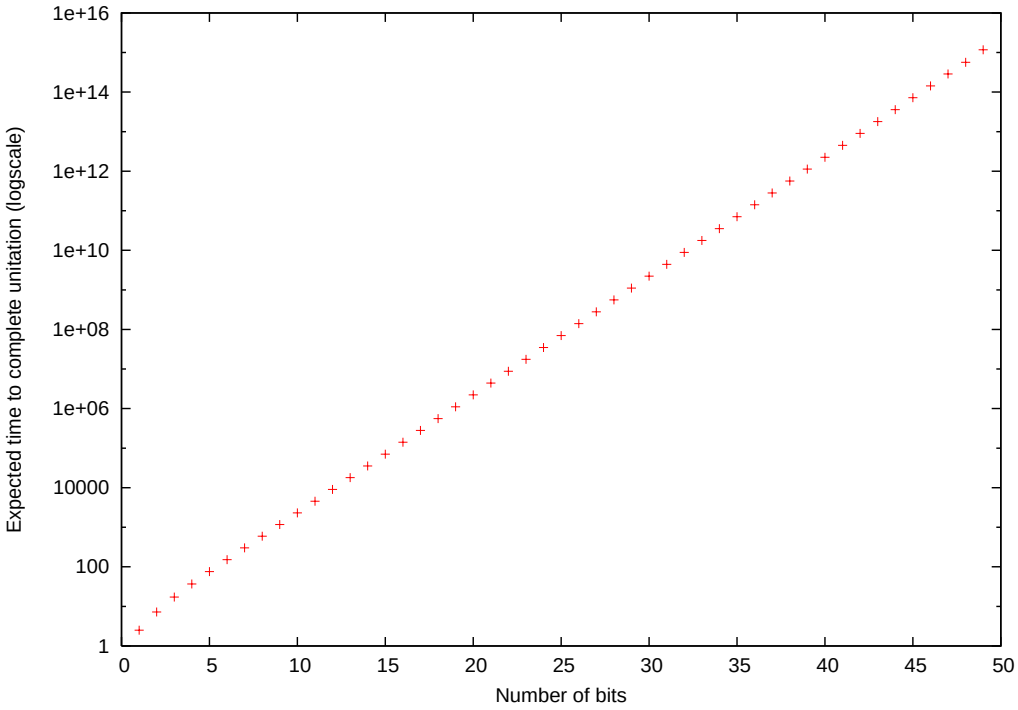

Figure 6.2: Mutation; an exponential time search operator.

choice of replacement policy has ramifications beyond the mutation operator alone, we discuss this decision separately in section 6.8.3.

6.7.2.1 Pseudo-code for mutation

```
// mutate a point X
function mutate(vector X){
    if (random boolean == true){
        use gaussian distribution
    }else{
        use uniform distribution
    }

    create Y a clone of X
    select dimensions to mutate at random
    for (each dimension to mutate d_m){
        if (use gaussian distribution){
            Y[d_m] += random gaussian distributed value
        }else{
```

Y[d_m] = random value from domain of dimension
}
}

if (Y is better than X){
X = Y
}
}

6.7.2.2 Objectives of mutation operations

The mutation operator is designed to perform moderate to long range movement within the search space. We deliberately avoid using gradient based or population distribution dependent mechanisms. The majority of the mechanisms used in this work do depend on such sampling. For completeness we require at least one mechanism that simply explores the probabalistic space. The mutation operator explores two types of space, either exploring globally with the use of the uniform random sampling mechanism or locally with the Gaussian distributed sampling.

6.7.3 Local population based techniques

The position of the individuals in the population is a model of the landscape formed from points that are considered significant. Geometric inference of properties suggested by this model may, if the model is accurate, be beneficial to the search. For instance, in a population based search, it is highly likely that at some point in the search, a proportion of the population will explore the same portion of space simultaneously. Once more than one member of the population is located in the same basin of attraction it is possible to use the relative positions and directions of the candidates to infer geometric properties of the space.

6.7.3.1 Centroid location

One such inference mechanism is already widely used in the field of particle swarm optimisation. “Queens” due to Clerc [9] is a mechanism for locating the centre of mass of the swarm. It is trivially extended to permit the location of the centre of mass of sub-sections of the swarm. The principle that is employed to good effect is that if the swarm is centering on a single attractor, then the sampling error made by individual

samples may be smoothed out by averaging over all samples, revealing the collective view of the location of the attractor.

We also use a centroid location mechanism which places samples at the average position of some sub-sample of the population. However, we must avoid creating a mechanism that simply replaces candidates with the mean of all samples. The loss of population diversity is not acceptable, and the assumption that the entire population is centered on the same attractor is almost certainly false. Under certain circumstances merciless use of such mechanisms can result in unwarranted exploitation of the space (see section 3.4.2), where the search collapses towards the centre of the sampled space irrespective of the nature of the landscape.

We use the same replacement policy throughout this work – samples are only replaced by those that are better. Regulating the replacement of samples by their relative merit helps to prevent increasingly intense sampling of the centre of the cluster. The cluster of samples does not change in distribution unless there is a net benefit in terms of fitness. In recognition of the locality of the landscape, we use clusters smaller than the whole population, allowing separate clusters to form and search the local landscape.

The mechanism of forming a cluster of individuals is to select an individual at random from the population and then locate the n individuals that are closest to it in toroidal space. This allows us the potential to search all the possible clusters of a certain size in the population. Cluster sizes have to balance the improved precision gained by using large clusters of samples against the desire to be able to maintain several independent clusters within the population. We use clusters of size $n = 3$, preferring to use planar sampling and sacrifice accuracy of the cluster estimation for increased flexibility offered by using the smallest possible meaningful cluster.

There are two reasons for choosing small clusters. Small clusters reduce the number of points that have to be within a basin of attraction to be capable of forming meaningful centroid samples. In our work we use very small populations, thus it is also preferable that clusters should be small to allow multiple clusters to form even in populations of this size. Though the individual iterations of the centroid location mechanism will be less accurate as a consequence, over a series of iterations the population within the basin of attraction of an optimum will nonetheless converge towards the optimum.

6.7.3.2 Pseudo-code for centroid location

```
// locate the centroid of those nearest a
// random sample chosen from the population P
// replace based on the best of N samples (N = round(P/20))
function centroid(population P,number of centroid samples N){

    // somewhere to keep a reference to the best sample
    // found when forming the clusters :
    create integer Best_sampled

    // somewhere to keep a reference to the best centroid
    // sample found :
    create point Best_centroid

    // for each centroid sample, we choose a starting point and
    // then find the neighbouring samples from which to form
    // the centroid.
    for each centroid sample {
        // first we select a random member of the population
        select integer X randomly from range [0,P-1]
        if(best_sampled is unset ∪
            P[X] is better than P[best_sampled]){
            Best_sampled = X
        }
        from P, choose the N points closest to P[X], called P_cluster

        create point X_centre initialised to zero
        // obtain the average score of the components
        calculate P̄_cluster, the average score of the points in P_cluster

        // create the weighted sum of the components
        for each point i in P_cluster {
            // weight points, fitness ∝ magnitude
            scale P^i_cluster relative to average fitness P̄_cluster
```

add scaled $P^{i}_{cluster}$ to X_{centre}

}

evaluate fitness of X_{centre}

if(X_{centre} is better than $Best_{centroid}$){

$Best_{centroid} = X_{centre}$

}

}

// only replace if the samples indicate

// a general improvement. If so, replace the

// appropriate member of the population

if($Best_{centroid}$ is better than P[$Best_{sampled}$]){

replace P[$Best_{sampled}$] by $Best_{centroid}$

}

return

}

6.7.3.3 Centroid location objectives

The centroid mechanism is primarily designed to re-centre good samples that are well placed but are within clusters which implicitly contain better placement information. Indirectly the mechanism is designed to increase sampling of interesting areas if a better point can be found. In many ways the centroid location mechanism is very similar to Clerc's queens method; we use a weighted average and sub-samples of the population in our calculations.

There are two significant oddities in the design of our centroid mechanism which alter its behaviour significantly. The first is that the point from which the cluster is defined is not part of the averaging process and so has no influence on the creation of the weighted average point, however, it is considered in the evaluation of the created centroid sample and replacement is considered only for these cluster defining points. This process attempts to replace a point in the centre of a cluster with a better point, but without significantly altering the density of the cluster, the point will just have been better "centered". The second aspect of note is that we have employed a mechanism for moderating the normally aggressive replacement rate. If more than one centroid sample is taken then only one replacement is made, and the replacement is evaluated

against all the central samples used in forming the clusters used in the search. Replacement is only performed if there is an improvement made against the fitness of the best point sampled from the population. This greatly slows the replacement rate, replacing the normal aggressive centering behaviour of the centroid based searches with one that only replaces already good points with better ones, whilst maintaining a high sampling rate of potential replacements.

We use auto-adaptive hill climbing to make small scale refinements to points which are successfully replaced by the centroid location mechanism. The centroid mechanism is an interpolative mechanism which explores the centre ground of clusters. When iterated in the basin of attraction of an optimum the centroid mechanism will tend to reduce the volume spanned by the cluster within the basin.

6.7.3.4 Point-wise extrapolation

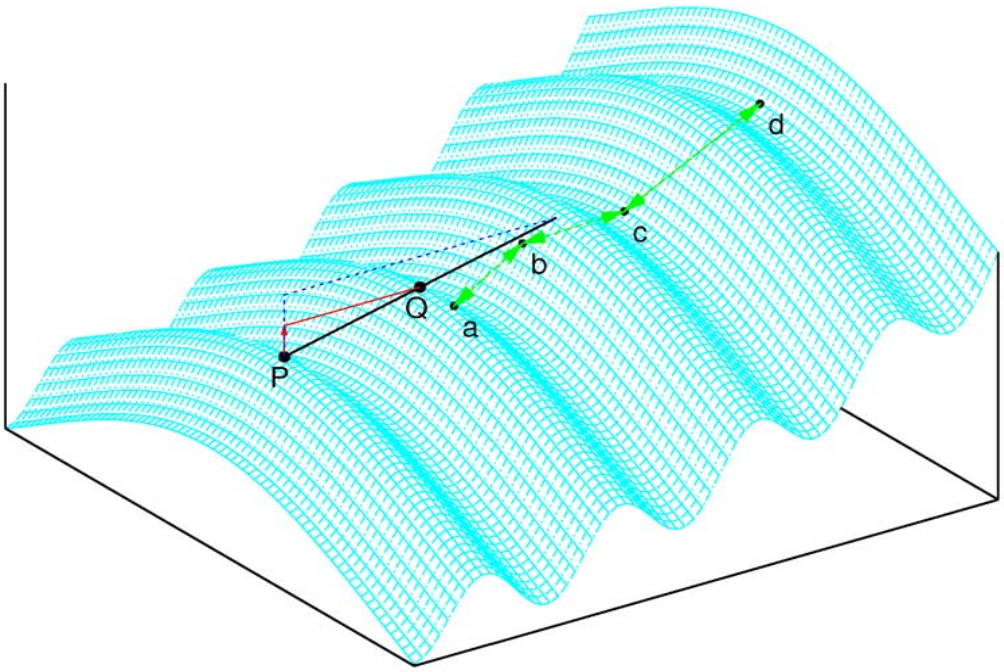

Figure 6.3: The sampling mechanism of the extrapolation mechanism on a maximisation problem. The extrapolation of the trend between two points P, Q gives an expected firection of continued improvement, which is then used to place samples in the scalar ranges. The scalar ranges shown in this example are sectioned as follows: a to b 0.5 to 0.75, b to c 0.75 to 1.5, c to d 1.5 to 2.5. One sample is placed in each scalar range, and if better the best sample replaces the worst of the pair of points, P.

It is common for population based techniques to have to climb slopes in the landscape. Mechanisms which explore the extrapolation space from samples of points are typically more successful on such landscapes. Most modern crossover operators designed for use in real number optimisation tasks are extended to incorporate both extrapolative search and the more conventional interpolation. This use of extrapolation is termed in the genetic algorithm literature “explorative” search – search beyond that of simple interpolation between points, which is deemed the “exploitative” region. The curious reader is referred to [39, 64, 60] and the references therein for a description of a large number of such operators.

We use a second local macroscopic mechanism which is designed to explore the boundaries of the population, using extrapolation from pairs of points. We choose the first point at random from the population, and use the closest neighbouring point from

the population for the second. Unlike the attempts to extend the reach of crossover operators, the mechanism we use for exploring possible extrapolations is a completely separate mechanism to that which we use to perform interpolation. By keeping the interpolative and extrapolative mechanisms distinct, we hope that they may be better engineered.

Interpolation and extrapolation are in general quite distinct. For interpolation we use a weighted centre mechanism to calculate the centroid of the hyperspace enclosed by a cluster of samples. This requires at least three points, extrapolation on the other hand, only requires two. Calculation of the weighted centre gives an exact location which is expected to offer an improved sample if the cluster is spanning a basin of attraction. Extrapolation on the other hand simply gives a vector which is expected to generally lead towards improved samples if the two points from which the extrapolation was made are positioned on the same slope. The conditions under which extrapolation may be expected to be successful and indicating a vector of improvement give no indication as to the range at which such an improvement may be resolved. This is an important consideration, since the basic extrapolation may be sound, but local anomalies may result in some choices of sampling range being worse than others.

We use an auto-adaptive ranging mechanism, identical to that used in the auto-adaptive hill climber to overcome the same difficulty with judging appropriate movement distances that we encountered with hill climbing. This is necessary because whilst the extrapolation vector may be correctly formed and indicative of a general trend of the landscape towards improvement, the landscape is unlikely to be smooth. The likely presence of local perturbations in the landscape requires that search be used to locate at which resolution the perturbations may be avoided. We thus use a further three samples generated using scalings selected from ranges of [0.25, 0.75], [0.75, 1.5] and [1.5, 2.5], allowing the adaption of sample spacing to closer, approximately the same, or larger distances. The extrapolation vector is scaled and added to the first point to create a sample. The best of the samples is compared to the worst of the two points, and if better replaces it in the population. Unlike the microscopic hill climbing case there is no requirement to store additional information about the last move made or the distances involved for the adaption to occur, since this information is encoded in the placement of the samples.

6.7.3.5 Pseudo-code for extrapolation

```
// extrapolate between points
```

```
function extrapolate(point P){

    Search population and locate Q the closest point to P

    // an indicator as to which of P or Q is to be replaced
    boolean replaceP = false;
    boolean replaceQ = false;

    // create the indication vector
    if (P is better than Q) {
        create vector R from Q to P
        // mark Q (the worst) for replacement
        replaceQ = true
    }else{
        create vector R from P to Q
        // mark P (the worst) for replacement
        replaceP = true
    }

    // we now need to calculate the preferred scaling
    create M, a set of scalars
    // somewhere to store the best sampled point
    create a point Best

    for (each scalar){
        create sample S = P + M R
        if(S is better than Best){
            Best = S
        }
    }

    // if an improvement has been made,
    // replace the appropriate member
    // of the population
    if (replaceP ∩ Best is better than P){
```

```
            P = Best
        }else if (replaceQ ∩ Best is better than Q){
            Q = Best
        }
        return
    }
```

6.7.3.6 Point-wise extrapolation objectives

The point-wise extrapolation operator is essentially a mechanism for sequentially hopping pairs of points through the space in the direction suggestive of general improvement. It is designed to require the least information from the population to be able to rapidly exploit the implicit information encoded in the relative positioning of the individuals in the population.

The extrapolation mechanism alone can fail, and used in isolation is very unlikely to successfully search the space, since it does not search off the line of the extrapolation vector. This is a potential weakness, but one that is hard to correct at this resolution. The off-vector search space is very large, and without placing further samples there is little guidance. We use the auto-adaptive hill climbing mechanism to search for refinements of successfully extrapolated samples. Hill climbing all samples is too expensive. The hill climbing mechanism is capable of moving in all directions in the space, and can move points away from the extrapolation vector.

One significant advantage of an extrapolation mechanism is in accelerating the process by which samples are taken from the near perimeter of the hyperspace spanned by the population. This allows rapid ascents of gradients that are on the edge of the population, and thus we have a medium range hill climber that is relatively robust against small irregularities in a generally regular local space and is much faster over long distances than the microscopic hill climber could be.

6.7.4 Pan-population techniques

All of the informed operators considered so far are geometrically based; they consist of moves calculated relative to gradients or vectors sampled within the space. None of the operators so far mentioned consider the population as a resource for storing and retrieving partially completed searches, only as a resource for directing the current search. All of the operators considered so far operate only on individuals or local

clusters of samples. If you consider the informed alteration of points as communication between points in the space, then all of the search mechanisms discussed so far gather and disseminate information locally. We need a mechanism which is capable of disseminating information globally and thereby communicating between individuals distant in the geometric space. This is the largest resolution which we will consider in this work. Larger resolutions are possible, parallel searches with multi-deme populations for instance have the potential to resolve the problem at the inter-population level, but we do not have a requirement for such mechanisms yet.

6.7.4.1 Crossover

The traditional genetic algorithm places great emphasis on the importance of the crossover mechanism to perform search. The precise nature of the contributions a crossover operator makes to the search is unknown. Crossover operators vary widely in design and capability, varying from hypercube restricted operators through full geometric interpolation and extrapolation mechanisms to bitwise disruptive techniques. In the centroid location mechanism and the point-wise extrapolation mechanism we already have methods designed to perform interpolation and extrapolation. We do not have a mechanism that is capable of exploring the vertices of the hypercube of possible population recombinations.

Crossover requires two or more candidates from which to copy the values. In the genetic algorithm literature the samples are typically referred to as parents, and the product or products of the recombination are termed the child, children or offspring. We will follow this naming convention.

Each unique dimensional value in the parent set defines a unique vertex in a hypercube of alternatives; when selecting a value for a dimension of the child, the legal values are only those present in the same dimension of the parents. Thus the possible samples that may be made by this type of recombination is restricted to the vertices of the hypercube.

This type of search is potentially useful because whilst independent optimisation of each dimension may occur, it does not occur synchronously between all points. Individually samples may be considered to be exploring the space in parallel. Each may have better values for certain dimensions than others. By swapping values between candidates and selecting for better performance, it is possible to recombine better values from distinct individuals which were fortunate enough to discover them into one composite individual, a process which is potentially quicker than waiting for them to

be optimised in isolation. This is in essence the thinking behind Holland's [42] much disputed "building block hypothesis". Part of the controversy over the building block hypothesis comes from it being interpreted as an attempt to explain all the phenomena of genetic search by a single mechanism. The failure of the hypothesis to be generally accepted as the only mechanism by which genetic search is performed does not invalidate its utility as a potential search mechanism.

We use the uniform crossover operator as the basis of our technique. The uniform crossover operator creates an offspring by selection of values from either of the parents at random. Conventionally the crossover operator is used with two parents. Where there are more parents contributing material to the offspring the term multi-parent is used. The use of multi-parent crossover operators is still rare in the evolutionary algorithm community, and several empirical analyses have proven inconclusive (see [24] for a comprehensive list). Multi-parent recombination allows the formation of new combinations of values that could not have been achieved with any significant probability by single step recombination. It also permits the generation of the same combinations that could have been achieved by conventional two-parent recombination. The set of possible recombinations obtainable using two parents is totally contained within the superset of recombinations obtainable using multi-parent recombination. Multi-parent recombination is quite common in the evolution strategies literature, and is regarded as potentially beneficial [25]. On balance we consider that the potential benefit of using multi-parent crossover operators is convincing. We allow the crossover operator to generate new individuals using dimensional values donated by any individual in the current population. Every time an individual is created by the crossover operator and accepted into the population it creates copies of the donated values and potentially reduces the population diversity. Consequently we only keep individuals created by the crossover operator if they are better than the best known candidate in the current population, which if beaten, is replaced.

This "replace the best" selection is extremely hard. Very few of those individuals generated by crossover will be better than the best member of the population. The purpose of this choice of replacement is to maintain the same context for replacement comparisons as is used for the generation of new crossover candidates. Any replacement that occurs during crossover represents a genuine improvement in the state of the search. Without a niching mechanism or some other similarity metric, other types of crossover replacement can not be permitted without risking significant population convergence. By replacing the best candidate with the offspring of successful crossover

operations we make the next replacement more difficult and prevent the crossover operation from simply converging the entire population on local optima. Conventional crossover mechanisms replace the parents involved in the recombination and may converge in the vicinity of an optimum by producing and accepting a series of permutations of the current best individual. Niching mechanisms are frequently used to attempt to prevent such premature convergence, but these require further parameterisation and are significantly complicated. By using a "replace the best" crossover replacement policy we avoid the thorny issue of selecting a niching mechanism appropriate to the problem structure. We also avoid premature convergence since the population can not be converged by the crossover operator unless there is a continuous fitness benefit in doing so.

6.7.4.2 Pseudo-code for crossover

```
// crossover candidates to create composites
// and replace the best individual if better.

function crossover (population Pop){

    // somewhere to store the best
    // of the samples

    create point Composite_best

    for (each crossover sample){
        create point Composite
        for (each dimension d){
            choose parent P from Pop at random
            // copy the dimensional value from the parent
            Composite[d] = P[d]
        }
        if (Composite is better than Replacement){
            Replacement = Composite
        }
    }
```

```
        locate Population_best, the current best

        if (Replacement is better than Population_best){
            Population_best = Replacement
        }
    }
```

6.7.4.3 Crossover objectives

The crossover operator is designed to allow the “dice and splice” type of recombinative search to proceed as effectively as possible whilst protecting the population from loss of diversity through genetic drift and the over investment in exploiting optima. We use the recombination operator to make copies of significant dimensional values only if the improvement offered by the new recombination is significant relative to the current population. In terms of search effort duplicated dimensional values are subject to greater search than those which are unique in the population. If the replacement point is derived from the current best point, then the point has been simply refined and should be replaced, the basic distribution of search effort has only slightly changed. If the replacement point is not derived from the current best point then the replacement of the current best point removes one set of duplicate dimensional values from the population and replaces it another. This effectively moves the focus of the search from one set of values to another, without directly causing the loss of the original donors of the values. If a crossover operation successfully replaces a candidate, the replacement is the target of an auto-adaptive hill climb attempt.

6.8 Controlling the search

The skeleton of our algorithm is as follows:

```
    function search{

        // generate the initial population
        create M individuals at random
        // storage for the best of the search
        create individual Best
```

```
    while (have more evaluations){

        // try to refine some points
        randomly select N_hc members of the population,
        for (each N_i of the N_hc selected){
            // microscopic level adjustment
            hillclimb(N_i)
        }

        // try to expand the span of the population
        randomly select N_ex members of the population,
        for (each N_i of the N_ex selected){
            // pointwise extrapolation
            extrapolate(N_ex)
        }
        // test crossover operations
        crossover(population)

        // explore the centroid
        centroid(population,N_ctr)

        // search for good mutations
        randomly select N_m members of the population,
        for (each N_i of the N_m selected){
            mutate(N_m)
        }

        // update the best found
        locate the best of the population Current_best,
        replace Best with Current_best if better
    }

    // return the best individual found
    return Best
}
```

For all experiments reported in this work we use a population size of 20, per generation the number of hill climbing attempts N_{hc} is 1, the number of point-wise extrapolation attempts N_{ex} is 10, the number of crossover attempts N_{cr} is 20, the number of centroid location attempts N_{ctr} is 1.

The large scale structure of the algorithm is dictated by a rough attempt to balance the relative performance strengths of the various mechanisms against the preferred types of optimisation. Operations such as crossover are, generally speaking, unlikely to create change in the population even when such changes are possible. This is in part because of the requirement of our crossover mechanism to improve the quality of the population as a whole, and partly because the probabilities of performing a successful crossover operation are quite slim, even under ideal circumstances. Dimensional values do not necessarily have to be independent to be successfully optimised in subsets. Thus we expect that separate development of dimensional values may occur, and if it does that recombination may save effort relative to the difficulty of rediscovering the same development. We thus assign crossover a high proportion of the evaluations.

The distribution of samples used in a cycle of evaluations is detailed in the following. It should be understood that each of the "evaluations" referred to in the following indicates a *novel* evaluation of the objective function. The word novel has been ommited from the phrase to avoid monotonous repetition.

If configured as described in this work, each of the N_{hc} members of the population that are hill climbed consumes either 4 or 9 evaluations, depending on the quality of the historic vector. The N_{ex} extrapolation attempts each consume 3 evaluations, and if successful, the auto adaptive hill climb mechanism is then employed involving a further 4 or 9 evaluations per success. There are as many crossover attempts made as there are members in the population, each using one evaluation. Every time the crossover is successful the best known candidate is replaced. The replacement candidate is then hill climbed, which uses another 4 or 9 evaluations. The N_{ctr} centroid operations each use 1 evaluation, and successful replacements are hill climbed at the cost of a further 4 or 9 evaluations. The mutation operator uses one evaluation for each of the N_m candidates that are mutated. Mutations are not hill climbed, since the mutation operator involves Gaussian sampling which frequently produces very small improvements. To hill climb every slight improvement made by the mutation operator would result in the use of vast numbers of evaluations.

6.8.1 Auto-adaptivity

Under certain circumstances, such as when the hill climbing mechanism must traverse a space which is very large relative to the hill climb step size, or when the extrapolation mechanism is approaching a discontinuity, auto-adaption is the only mechanism by which the search maintains a reasonable probability of success. The ranging mechanisms used in the microscopic hill climbing and the extrapolation mechanisms are essential to their operation. It is tempting to extend the auto-adaptivity notion to embrace the larger scale decisions, such as the distribution of evaluation effort between the various operators.

Auto-adaptation of the algorithm structure is complicated and we considered achieving it in a probabilistic manner. Probabilistic adaptation is achieved by the analysis of the relative success of the various operators. By identifying the operators that appear to make the most progress in the recent sampling period, the proportions of the operators which are applied in the next round of sampling may be modified. The task is significantly complicated by the variability in the operator success rates, and the distinct resolution differences over which they operate. Some operators like mutation are irregularly successful yet are capable of searching large spaces, some like hill climbing are regularly successful but only make small alterations to the points. Which is more relevant depends upon the current status of the search, and how close the candidates are to the global optimum and their relative probability of improvement, all of which are very difficult to ascertain from the current sampling conditions.

Auto-adaptivity of the distribution of effort between operators is not obvious. For instance, following a successful mutation, the resultant point is likely to be easy to improve through hill climbing. The inference that further mutation operations may be useful for the population as a whole ignores the unique properties of the point that has been mutated. The effect of the mutation is more likely to be faster shared amongst similarly placed points by a crossover operator, than waiting for each point to be individually mutated in a similar manner. Xie [95] reports the use of a simple neural network as a potential control medium, but does not provide convincing evidence of its effectiveness. See section 4.9 for a discussion of hybrid techniques in the literature.

The difficulties of designing an effective and justifiable auto-adaptive algorithm control mechanism are significant. We do not pursue auto-adaptive structural modifications in this work.

6.8.2 Inference via sampling

An optimisation algorithm is designed to distinguish and optimise a certain class of landscapes. Samples taken from the domain are interpreted relative to prior models of the structure and used to imply confirmation as to what properties the landscape apparently holds. The samples taken are the only source of confirmation the algorithm has. We believe that the evidence collected by the sampling process instructs the best future direction for the search. Consequently, we also consider that all decisions regarding which points are of interest should be evidence based, being qualified by an actual improvement in the objective value of the considered sample.

It is common in stochastic search to permit an algorithm to make moves that worsen the quality of the solution, in the hope that by doing so the search may be released should it have become trapped in a local optimum. By permitting the worsening move it is hoped that the search will be released from the local optimum and may now be able to improve. Detection of the local optimum is not typically performed, for instance in simulated annealing the worsening moves are controlled by a probabilistic method which decreases in likelihood as the search progresses.

We use only evidence inferred from the sampling to instruct the maintenance and replacement of samples. The samples maintained in the population are the current best model of the landscape. Ignoring the sampling and randomly worsening solutions without evidence that such a move is appropriate is potentially harmful to the search. At the same time a mechanism is required to release the algorithm from becoming trapped in local optima. We permit such down-climbing moves to be performed only on candidates that we believe have ceased to have a reasonable probability of further movement.

We detect the failure of a candidate through the failure of the most robust and highest resolution operator, the hillclimbing operator. We then attempt to mutate the cadidate, to try to explore the local area before finally restarting the candidate. The main characteristics of the downclimbing mechanism, which is invoked as part of the hillclimbing mechanism are given in the following code.

```
function downclimb{

   // under what conditions do we restart a point?
   int failedClimbs = hill climb failures;
```

```
   if(failedClimbs > hillClimbLimit &&
      failedClimbs < totalClimbLimit){
      try mutation
   }else if(failedClimbs > totalClimbLimit){
      reset the point
      failedClimbs = 0
   }

}
```

6.8.3 Replacement policy

The algorithm does not have an identifiably separate candidate selection process, instead selection is distributed throughout the algorithm. Candidates are selected for inclusion in operations randomly from the population, the new candidates produced by these operations are kept based on their relative fitness. This allows the selection and replacement method to be customised for each operation. Some operators such as the centroid location mechanism or the point-wise extrapolation mechanism operate over small groups of clustered individuals. Selection for inclusion in the operation is then an automatic consequence of the selection of the first point. In general, we use a policy of only replacing candidates if they are improved, or equally well performing, relative to the group over which the operator is operating. This type of choice allows the algorithm to alter the distribution of samples in the population only if it locates points that are better. The expected relative scale of the operation is taken into consideration when comparing relative performance. The short ranged microscopic hill climber only has to improve relative to the initial sample, in contrast the crossover operator manipulates information from the whole population, and relative improvement is judged against the whole population.

Our sampling method lends itself to very aggressive exploitation of local optima, and the “replace only if better” policy ensures new candidates must offer tangible improvement. Unfortunately selection based on monotonic improvement easily traps the algorithm in sub-optimal positions. It will be necessary to sacrifice some samples to continue the exploration of the space. Since all samples maintained in the population

are there on merit, removal of samples should also be performed on merit, however we do not perform a simple "replace the worst" cull. Replacing the worst samples in a population is short sighted, primarily because some samples may have longer development times; they may take longer to approach areas of interest than others. These relatively slow moving samples will be prematurely culled before they have explored their local space. The decision to terminate an exploration should not be made in terms of absolute quality of the candidate, but rather in terms of the likelihood of further progress. The diagnosis of exhausted points, and their re-initialisation is described in section 6.8.5.

6.8.4 Population diversity

Our population is small. We have a strong interest in maximising the utility of each sample. Occasionally the search operators such as crossover will produce candidates which are identical clones of other candidates already in the population. With such a small population it is important to prevent the population from becoming prematurely dominated by a single candidate. At the same time, significant effort is expended in other algorithms (such as simulated annealing) to ensure that the final stages of the search are spent refining the current candidate. We thus detect and reject clones created by likely mechanisms such as crossover and the centroid location mechanism, preferring to maintain population diversity and rely on the power of the local search mechanisms.

6.8.5 Elitism and abandoning points

Elitism, where the best candidate in the population is guaranteed to persist, is commonly used in population based searches. The principle is sound, that without protection an algorithm that loses samples may unfortunately dispose of the best candidate. It is generally considered that elitism is required for the general purpose genetic algorithm to effectively optimise. The traditional form of elitism maintains the best candidate in the breeding population. Maintaining the best candidate in the population without remission, combined with the normal disseminative operators, places a strong convergence pressure on the population. The search performed by the population is likely to be heavily influenced by any such perpetual members.

To avoid the domination of the population by any one candidate we do not use any form of elitism and do not distinguish between the best of the population and any other

member. Combined with the replacement policy of replacing candidates which have failed to progress, this means all candidates have a limited time-span they can remain stationary – irrespective of their quality. We are interested in the absolute best solution found at anytime during the search, not only at the termination of the search, and so we keep a copy of the best solution discovered so far. This copy is purely for book keeping and reporting purposes, and has no further influence on the progress of the search.

It should perhaps be underlined that this algorithm does not have a mechanism for removing the worst candidates from the population. Instead samples are allowed to persist until they are trapped by the landscape. The sampling history of a point gives some indication of when a point is trapped and has ceased to progress. The failure to progress after some number of iterations is indicative of either a failure in the sampling mechanism, or, if the sampling is working correctly, the high probability of an absence of anything of interest. We designed the sampling mechanisms as robustly as we could, and must assume them to be competent. A sustained failure to progress, then, implies the exhaustion of the point. The microscopic hill climbing mechanism generally contributes the smallest scale of improvement to the search and is the hardest mechanism to exhaust. When successive iterations of hill climbing have failed to locate improvement, we consider the point to be locally trapped. Trapped points are considered as unlikely to improve. By the time the microscopic search has stagnated for several iterations the point will have been in the population for some time. If the search operators are functioning correctly any influence the point can have on the search is likely to occur within this period, and so the contribution of this point to the search is likely to have already occurred; it may be removed with confidence.

When a point has been observed to halt, and hill climbing has failed to make progress for some period of time, we must consider the local search conducted by the point to be stalled, and will have to replace the point with one that may be more profitable. However significant investment in the sample has already been made, and total replacement of the point may not be necessary. If the point is trapped in a local optimum that hill climbing can not escape it is feasible that mutation may release the point. Instead of replacing stalled points immediately we attempt to use mutation to "jolt" the point, working from the premise that the local area may contain beneficial points. If the mutation operations also fail to locate improvement. We decide the point is stalled and curtail our investment in the area, we replace the sample with one randomly generated from the domain of the function.

Abandoning and re-initialising stalled points allows a small population to process

a large number of distinct points during a run. We use the repeated failure of microscopic search to diagnose stalled points. By definition the degeneration of gradient based search at this resolution implies the point is trapped in some form of local optimum. Whether the search has degenerated because it is trapped in a genuine optimum, or because the search operator is badly sampling is actually irrelevant. Either way, continued investment in the same process is unlikely to yield improvement. Because we probabilistically sample, it is obviously a guess as to at what stage the search has truly degenerated. The balance may be summarised as: the more persistent the search, the better the results of each exploration will be refined, but fewer explorations will be made. We allowed points to persist for ten iterations of the hill climb operator before considering them stalled. Once classified as stalled points, we attempted to jolt the point using mutation five times, after which the point was considered to be unpromising and was re-initialised.

6.8.6 Dealing with constraints

To allay suspicions of over fitting we wish to use the algorithm on a wide variety of problems. A large group of functions are well defined over the full range of the domain, and are referred to as unconstrained functions. Constrained functions on the other hand, such as Keane's function (introduced in section 3.3.5 and described in detail in section 7.5) are only valid for a subset of the domain space. In these functions the interactions between the variables defining the acceptable region of the domain is controlled by one or more constraint functions defined separately from the objective function.

Search mechanisms operating over the full domain are likely to create values that fall into the illegitimate region. A constraint handling mechanism must steer the search away from illegitimate regions of the space. Typically such mechanisms work by penalising candidates that use illegitimate domain values. The more sophisticated mechanisms treat the constraints as objective functions in their own right, and use a multi-objective approach to solve the whole problem. The interested reader is directed towards Michalewicz's book [59] for a comprehensive treatment of the topic. Clearly the better the penalty mechanism is at smoothing out disruption caused by the constraints the better the continuity in the space and the algorithm has a better chance of navigating the space successfully.

As noted by Hedar in [67], there is the possibility that the feasible space is fractured

and separated by infeasible regions, requiring that both the feasible and infeasible regions be searched. When handling the Keane function, Michalewicz and Schoenauer [74] use a constraint handling technique that explores only the boundary of the feasible and infeasible space. Even within the test suite proposed by Michalewicz in [59] the majority of functions have optima that are not fully constrained, and so searching only the constrained boundary is not a generally applicable technique.

Surry in [84] suggests that the majority of constraint handling techniques are sensitive to their free parameters. We avoid the issue of selecting the best constraint handling technique and use a trivial "parameterless" option instead: All of the possible algorithm specific free variables are held constant throughout all of our experiments. Our mechanism attempts to direct the search of the infeasible region towards searching the feasible region. Solutions from the feasible domain are evaluated by the objective function. Solutions from the infeasible region are evaluated according to their penalty. To distinguish feasible and infeasible solutions through the fitness function all infeasible solutions are penalised a large fixed value, chosen to be large enough to be significantly worse than the expected worst performance of feasible candidates. All the constrained functions we used had limited feasible values to within a few hundred thousand of zero or smaller: to be on the safe side we used an initial penalty of one million. This allows the algorithm to differentiate between feasible and infeasible solutions with a selection pressure on choosing the feasible solutions. To be able to distinguish between the various magnitudes of constraint infringement, we need a proportional second penalty. For each constraint that is breached the magnitude of the breach is scaled by one million and added to the penalty. This allows the algorithm to differentiate between solutions with various degrees of constraint infringement.

It should be noted that although we use a single number for representing the quality of solutions, which we do because the of the binary nature of the decision, there are other ways that could have been used that may have kept the number of constraints breached seperate from the result of the evaluation. Other than complicating the algorithm, maintaining this distinction makes little difference to the actual decision making process, since decisions of comparative quality still necessitate a prioritisation of the number of breached constraints relative to the result of the evaluation. In this case we have used a large scalar to prioritise meeting constraints more than improving the result of the evaluation.

6.8.6.1 Pseudo-code for penalty function

```
// evaluate a point X with regard to
// some constraints and an objective function.
function evaluate(X){

    // somewhere to store the penalty (if any)
    penalty = 0

    for (each constraint){
        if (constraint is breached){
            // penalty direction is dependent upon objective
            if(minimiser){
                penalty += 1,000,000 * magnitude of breach
            }else{
                penalty -= 1,000,000 * magnitude of breach
            }
        }
    }

    if (no constraints are broken){
        // if there are no broken constraints
        // the point is legitimate and may be evaluated
        // directly using the objective function
        return f_objective(X)
    }else{
        // ensure even minor constraint breaches are
        // clearly identified
        if(minimiser){
            penalty += 1,000,000
        }else{
            penalty -= 1,000,000
        }
        return penalty
    }
```

}

6.8.6.2 Objectives of penalty function

The main advantage of the mechanism described here is it is extremely simple. There are no parameters to consider. As long as the initial penalty is large enough to separate the majority of the feasible space from the infeasible space the penalty function will guide the search towards feasible space. Once within feasible space, the function transparently returns the objective function evaluation. We do not claim the mechanism to be better than those expounded in other works, rather we claim it is sufficient to allow our algorithm to operate on constrained functions, which being radically different to the conventional unconstrained test suites are an important addition to our test suite.

6.8.7 Sacrificing convergence

The convergence of a population on an optimum is considered a desirable property in a population based search. Concentrating search effort on a particular location has obvious performance benefits if the location is well chosen. Convergent behavior is weakened as a direct consequence of the decision to use the restart mechanism to remove stalled points. Population convergence is often used as an indicator of stabilisation in the search. The loss of the convergence property of the algorithm means there is no obvious marker by which the termination of the effective search period may be judged. We are also not able to guarantee the concentration of the search effort and instead rely on the quality of our local search operators to detect opportunities for improvement. We consider the loss of population convergence to be untroubling, having effectively exchanged the possibility of convergence for the possibility of improved sampling.

6.8.8 Parameterisation

Parameterisation is the hidden cost of modern search heuristics. Many algorithms are capable of producing extremely strong results when properly tuned, but tuning can be difficult. Clerc's TRIBES mechanism is auto-parameterising [11], and Parsopoulos [65] even uses a differential evolution method to on-the-fly parameterise a simple particle swarm optimisation algorithm. As Monson points out in [62] the Kalman swarm parameterisation space can be greater than the space of the function to be optimised. The amount of effort the researcher invests in the parameterisation of the algorithm is significant in evaluating the true quality of the reported performance.

Our algorithm has many complicated internal mechanisms, and the full parameterisation set is rather large. Clearly we are in danger of following the Kalman swarm algorithm into parameterisation hell. To avoid this, and to make the author's life easier, we created an auto-parameterisation mechanism which maintains the same ratios between the various critical parameters, scaling in proportion to the population size. Having found our initial settings to be promising, we were not particularly precious about the precise parameterisation of the algorithm and are certain that better choices for these ratios will be found by future research. All our experiments against the benchmark functions used the same population size, and consequently received exactly the same search heuristic parameterisation.

Obviously we depend on the user for those parameters that define the problem; the objective function, the dimensionality, iteration limit and the accuracy with which it is desired to locate the optimum (if known). A typical call to the optimiser as used to generate the results in this work is then only four arguments, with no scope for problem specific parameterisation or tweaking.

6.9 Comments on our design

There are several immediate aspects of the design of this algorithm which attract immediate attention. As further research unveils more detail of the interactions between the various aspects of the design, we expect to discover more aspects of the design which could have been better chosen.

One obvious compromise is that the ranges of values used in the auto-adaptive mechanisms is not perfect. The ranges 0.25 to 0.75, 0.75 to 1.5, and 1.5 to 2.5 give mean expected scalings of 0.5, 1.125 and 2.0. An ideal choice would have produced means of 0.5, 1.0 and 2.0. We considered the minor effect this is expected to have on the sampling of ranges to be insignificant.

Similarly the auto-adaptive ranges themselves are arbitrary, selected to form the basis of a binary search. The hill climbing auto-adaptive mechanism uses three samples per iteration to locate preferred scalings of the direction vector. Three samples are used because two samples are the minimum required to auto-adapt and using three samples allows the approximate maintainence of the current step size. It is possible that other choices of ranges could be more appropriate.

The auto-adaptive hill climber uses only 5 randomly distributed points to estimate gradients in very high dimensional spaces. Using more samples in the estimation of the

gradient is beneficial in high dimensional spaces. The effort invested in tracking the gradient should be proportional to the difficulty encountered, which we estimate by the time the hill climbing algorithm has spent stalled. Increasing the number of samples used as the search begins to stall could be achieved without inflating the search cost by proportionally reducing the number of auto-adaptive hill-climbing attempts that are made before the point is abandoned.

It is a point of oversight that the last position is not used in calculating the hill climbing gradients for the next move. If the last move was a hill climbing step then it is guaranteed that the last position is on the hypersphere of the next set of samples. It would have been trivial to check the utility of the last point and include this source of information, which would have contributed an additional sample to the information of one of the most sample-starved operators. This method is equivalent to including the historic direction vector in the construction of the composite vector. If the hill climber is collecting samples then the historic direction vector is known to have failed to improve the sample quality. Its inclusion as a deterrent is thus likely to bring an improvement in the sampling of the hill climbing.

There are aspects of the comparisons made when deciding between points that could be further refined. For instance before restarting points we attempt a series of mutations. As the algorithm stands, if the mutation improves upon the previous point then the point is retained. Otherwise, if the point is not improved then it will be restarted. It has been pointed out that the comparison is somewhat unfair, because of the comparison between a point that has been through the hill climbing process (the fatigued point) and one that has not (the proposed mutant). We can not afford to perform hill climbing on all points that are mutated. Instead we could store the points value before hill climbing was performed, and use this as an indicator of the potential differences between the points.

The hill climbing mechanism assumes a simple gradient structure, thus it does not consider the relative magnitude of improvements when constructing the composite vector. It is unknown under what conditions this assumption degenerates and causes poorer sampling than would have been performed if the composite vector was constructed using components scaled by their delta in the objective function.

The representation uses toroidal mapping to maintain the closure of the space for all vector operations. This is an arbitrary choice, and as remarked earlier is likely to create discontinuities in the evaluation function at the “seam” of the toroid. It is worth considering the alternatives, including using a geometric “reflection” to maintain the

integrity of the vectors.

The mutation mechanisms are strongly aligned along dimensional separations of the space. This is potentially a major weakness in the mid-range search mechanism. Though the algorithm performance has not been exposed as significantly weak, extensions to this work should consider the inclusion of mutation operators capable of more complete exploration of the space. Additionally the hill climbing mechanism is not employed following mutation to reduce the consumption of novel evaluations of the objective function. The hill climbing mechanism is employed after all other major disruptive search operations, and it could be argued that it should also be employed following range mutations. We do not employ the hill climbing mechanism to save evaluations of the objective function from being wasted on attempting to hill climb small embellishments which are better searched by the hill climbing mechanism directly.

The crossover mechanism uses comparison against the best candidate in the population as the context for replacement decisions, which is not precisely the same as the context of a random sample from the population which is the context used for generating potential replacements. This makes the crossover operator less likely to perform a successful replacement than otherwise might be the case, and simultaneously impedes the convergence of samples generated by the crossover operator. We believe, that given the small population, the reduced rate of population convergence merits this minor asymmetry.

We could also break the modules apart and try running sections of the algorithm in isolation, to try to determine what the manifest contribution of each of the modules actually was. This opens up a whole section of further experimentation, and offers the promise of significant performance rewards if the underperforming components could be identified.

Chapter 7

Surrogate evaluation functions

The following test functions are used as surrogate evaluation functions in this work. Several test functions have a variety of dimensionality or range settings in common use. A few (like the Rosenbrock) actually have distinct function definitions referred to by the same appellation. Care should be taken to ensure the discussed function is correctly identified.

Where reliable results have been discussed in the literature, the best known to the author are presented in the appropriate tables. These are not definitively the best results ever located, but are representative of the common state of the art. The literature review on which these tables are formed was performed in the August of 2005.

Where relevant properties are known, short notes are provided. For definitive algorithm descriptions and full details of implementation conditions the reader should refer to the referenced documents. Where an author has published several results, rather than repeat the annotation for each result published by the author, we remark only upon the first result.

7.1 General comments on reports

The functions used in this evaluation suite are chosen because they have two properties, they are diverse and they are widely reported by many authors using many optimisation techniques. The technical difficulty of some of these functions is no longer as great as once might have been assumed, however, they still serve as universal arbitrators of performance, which if chosen over a wide enough range of functions is hoped to reflect generality over the class of like functions.

Some functions are constrained (Michalewicz's test suite and the Keane function)

and so we include reported results that are known not to breach the constraints. Due to the constraint handling mechanisms involved, it is possible that the results published for the constrained functions are from the illegitimate region, but that the violation is by less than the acceptable measure. We follow the tradition in the field by considering such solutions acceptable.

Occasionally authors presented unusual aspects of the algorithm's performance, or used unclear definitions or terminology. Under such circumstances attempts have been made to normalise the presentation. Ambiguity was resolved by erring on the side of generosity. Generally if the result is questionable and no clarification could be obtained from the author we have had no choice but to omit the reported result. If the result was however the best reported performance in the literature we are obliged to include it – with suitable cautionary notes.

Most evaluation functions have a domain definition as as part of the function specification. Despite this the reported functions are evaluated over a large variety of domains. Where an author has not specified the domain they have used, and none of their other work gives indication of the domain chosen, they are assumed to be using the default domain.

7.2 Ackley's function

$$f_{Ackley}(\vec{x}) = -a\exp -b\sqrt{\frac{1}{n}\sum_{i=1}^{n}x_i^2} - \exp\frac{1}{n}\sum_{i=1}ncos(\omega x_i) + a + e$$

$\vec{x} = (x_1,...,x_n),\ -32.768 \leq x_i \leq 32.768$
where $a = 20\ b = 0.2\ \omega = 2\pi$
n is the number of dimensions.

$f_{Ackley}(optimum) = 0$ at $(0,0,...,0)$

7.2.1 Best results : Ackley's function

Herrera (25 dimensions, row 1) used an elitist GA applying the "non-uniform" mutation operator [59] which is a form of cooling mutation. The result reported by Herrera is the best of a sweep of 25 heterogeneous and homogeneous crossover operator choices. The best crossover operator used on the Ackley problem in the Herrera work is the "Dynamic Heuristic" crossover operator. This operator combines two distinct features, a heuristic exploitative method that produces offspring in close proximity to the

Row	Domain	Evaluation calls	Best	Mean (Std. dev)	Pop. size	Author	Notes
1	$[\pm 32]^{10}$	200,000	–	1.5e-3 (1.54e-2)	100	Das [16]	DE-TVSF Initialised to $[15, 32]^{10}$
2	$[\pm 32.768]^{10}$	125,000	–	2.34e-16 (1.07e-15)	125	Settles [77]	BS inertia (GA/PSO hybrid)
1	$[\pm 32.768]^{25}$	100,000	1.52e-7	3.81e-7 (1.67e-7)	61	Herrera [39]	GA
1	$[\pm 32.768]^{30}$	300,000	–	4.677e-9 (1.96e-9)	100	Ortiz-Boyer [64]	GA using CIXL2 crossover operator
2	$[\pm 32]^{30}$	40,000	–	0.104323	20	Clerc [12]	E&S PSO algorithm
3	$[\pm 32]^{30}$	500,000	–	-1.1901591e-15 (7.03e-16)	100	Vesterstrøm [86]	DE
4	$[\pm 32]^{30}$	1,500,000	–	1.1e-3 (8.8e-2)	300	Das [16]	DE-RANDSF Initialised to $[15, 32]^{30}$
5	$[\pm 32.768]^{30}$	250,000	–	5.74e-15 (3.5e-15)	125	Settles [77]	BS constriction (GA/PSO hybrid)
6	$[\pm 32.768]^{30}$	20,000	–	7.41370 e-7	20	Monson [62]	BareBones PSO
1	$[\pm 32]^{100}$	5,000,000	–	8.02321117 e-15 (1.74 e-5)	100	Vesterstrøm [86]	DE
2	$[\pm 30]^{100}$	2e5 without change	–	1.5e-2	–	Vesterstrøm [87]	ARPSO

Figure 7.1: Table of notable results for the Ackley function

best parent, and an explorative "dynamic" operator that protects against convergence. The algorithm reduces the amount of exploration in favour of heuristic exploitation during the run, resulting in the final stages of the run being highly dedicated to finessing the best results. This is an extremely strong cooling strategy, giving good results on a large number of problems.

In [87] Vesterstrøm publishes the best results for the ARPSO variant of particle swarm optimisation without the population size nor the number of evaluations used to reach the value reported. The experiment was terminated when 200,000 evaluation calls had passed without improvement. The value reported by Vesterstrøm in [86] is actually impossible. In a personal communication from Vesterstrøm we were informed that it is believed to be due to rounding error and an imprecise representation of the number *e*. If so, the rounding is problematic since the introduced error is large compared to the precision with which the system is solving the problem. All of the results from [86] are best considered as accurate rounded to 14 decimal places, since this is the precision of the constants used.

Monson [62] published his results in graphical format. The precise results used here are from a personal communication of the exact result set used. Though [62] does not include published results for the Ackley function, the Ackley function is present in the result set and those results are reproduced here by the kind permission of the original author.

7.3 De Jong's sphere function

Due to De Jong [18] and one of the more common test functions, approached by [39] amongst many others. The function is very simple, it is unimodal with the optimum at $(0,..0)$. This function is clearly solvable by a hill climber.

$$f_{Sphere}(\vec{x}) = \sum_{i=1}^{n}((x_i)^2)$$
$$\vec{x} = (x_1,...,x_n),\ -5.12 \leq x_i \leq 5.12$$
n is the number of dimensions.

$$f_{Sphere}(optimum) = 0 \text{ at } (0,..0)$$

7.3.1 Best results : Sphere function

The best results for the De Jong's Sphere function are listed in figure 7.2.

Row	Domain	Evaluation calls	Best	Mean (Std. dev)	Pop. size	Author	Notes
1	$[\pm 5.12]^{10}$	150,000	0	0	20	Liu [52]	MeSwarm (PSO)
2	$[\pm 15]^{10}$	62,500	–	0.3823 (0.1029)	125	Angeline [1]	Hybrid PSO
3	$[\pm 100]^{10}$	200,000	–	$0\ (\forall \leq 10^{-8})$	–	Vesterstrøm [87]	PSO Initialised to $[50, 100]^{10}$
4	$[\pm 100]^{10}$	16,740 (mean)	–	1e-4	100	Das [15, 16]	DE-RANDSF Initialised to $[50, 100]^{10}$
1	$[\pm 5.12]^{25}$	100,000	1.35e-15	1.37e-14 (9.63e-15)	61	Herrera [39]	GA (see sect. 7.2.1)
1	$[\pm 5.12]^{30}$	300,000	–	1.077e-16 (1.00e-17)	100	Ortiz-Boyer [64]	$EGNA_{BGe}$
2	$[\pm 20]^{30}$	40,000	–	$0\ (\forall \leq 10^{-6})$	20	Clerc [12]	E&S PSO algorithm
4	$[\pm 100]^{30}$	150,000	–	4.16e-5	100	Yao [96]	Fast EP (Gaussian and Cauchy mutation)
5	$[\pm 100]^{30}$	150,000 ?	7.9e-6	4.2e-4	100	Fogel [31]	Extinction EA (Cauchy mutation)
6	$[\pm 5.12]^{30}$	500,000	–	$0\ (\forall \leq 10^{-25})$	100	Vesterstrøm [86]	DE
7	$[\pm 5.12]^{30}$	15889.5 (mean)	–	$\forall \leq 10^{-2}$	30	Eberhart [22]	PSO with constriction
8	$[\pm 15]^{30}$	125,000	–	5.2181 (0.4935)	125	Angeline [1]	Hybrid PSO
9	$[\pm 50]^{30}$	20,000	–	4.723	20	Monson [61]	Kalman Swarm (PSO)
10	$[\pm 100]^{30}$	200,000	–	$0\ (\forall \leq 10^{-8})$	20	Vesterstrøm [87]	PSO Initialised to $[50, 100]^{30}$
11	$[\pm 5.12]^{30}$	152,070 (mean)	–	1e-4	300	Das [15, 16]	PSO-DV Initialised to $[50, 100]^{10}$
12	$[\pm 50]^{30}$	20,000	–	1.05443e-29	20	Monson [62]	PAKS (Kalman swarm)
1	$[\pm 5.12]^{100}$	5,000,000	–	$0\ (\forall \leq 10^{-25})$	100	Vesterstrøm [86]	DE

Figure 7.2: Table of notable results for the Sphere function

In [39] Herrera reports a result for a 25 dimensional function found with a GA using the homogeneous Dynamic Heuristic crossover operator. The $EGNA_{BGe}$ referred to by Ortiz-Boyer [64] is the "Estimation of Gaussian Network Algorithm", an estimation of distribution algorithm. Yao [96] used a modified Evolutionary Programming method called the Improved Fast Evolutionary Programming algorithm. The significant design feature of the algorithm is that 2 offspring are created for each mutation step, one by Gaussian mutation and one by Cauchy mutation. The best of the two is kept. Fogel [31] uses Yao's results as the basis for his experiments. The number of fitness function evaluations is not clearly indicated in the paper but it is implied to be derived from the settings used by Yao.

The performances published by Eberhart [22] though low precision are (according to Eberhart) the best results known up to the time of the publication of that work (2000). Das et al [15, 16] reports values of 0.0001 for the Sphere function, which are the stopping criteria for his search. How much better the values could have been if the search had continued is unknown. Since all methods tried by Das achieved the desired accuracy for 10, 20 and 30 dimensions of the sphere function there is nothing to distinguish the reported results other than the mean number of generations required to reach the desired accuracy. All other benchmark results published in [15, 16] have distinct differences in the mean best score found and are so distinguished by this metric.

7.4 Griewank's function

Created by Griewank [36] the function is used as a scalable multi-modal test function. As was noted by Whitley [90] and Locatelli [53] this function undergoes a collapse in complexity as it is scaled to higher dimensions. Despite the failure of higher dimensional variants to fulfill the promised complexity of the lower dimensional versions, it is still an interesting problem and one for which a large number of results are published by numerous different authors.

$$f_{Griewank}(\vec{x}) = \frac{1}{4000}\sum_{i=1}^{n}(x_i)^2 - \prod_{i=1}^{n} cos(\frac{x_i}{\sqrt{i}}) + 1$$

$\vec{x} = (x_1, ..., x_n),\ -600 \leq x_i \leq 600$
n is the number of dimensions.

$f_{Griewank}(optimum) = 0$ at $(0, 0, ..., 0)$

7.4.1 Best results : Griewank's function

in [39] Herrera publishes a result which has the best mean performance (for the dynamic heuristic crossover operator) here marked by †, but also has a result (found using a fuzzy recombination operator) that gives a better absolute best and a marginally worse mean ‡. Liu's results on the Griewank function are unusually high. The failure to approach the optimum for the 10 dimensional function is troubling, especially since all the comparison experiments in [52] also strongly approach 1 - this may be a typographic error and zero may be intended. The author was contacted in September 2005 to verify the situation, but no response has been received as of November 2005. The mean number of evaluation calls reported by Eberhart [22] is calculated from the 17 of the 20 repeats which are successful with less than 300,000 evaluation calls.

7.5 Keane's function

Due to [47] and tackled in [48, 59, 60, 74] amongst others. Keane's function is a challenge due to the positioning of the optimum exactly on the feasible boundary. The precision with which the boundary may be resolved is then of critical importance to the ability of the algorithm to optimise this function. Several of the results reported here are achieved with the aid of techniques expressly derived for solving the Keane function.

$$f_{Keane}(\vec{x}) = \frac{|(\sum_{i=1}^{n}(cos^4(x_i))) - (2\prod_{i=1}^{n}(cos^2(x_i)))|}{\sqrt{(\sum_{i=1}^{n}(ix_i^2))}}$$

$\vec{x} = (x_1, ..., x_n), 0 \leq x_i \leq 10$
where $\prod_{i=1}^{n}(x_i) > 0.75, \sum_{i=1}^{n}(x_i) < 7.5n$
n = 20 or 50 is the number of dimensions.

$f_{Keane}(optimum) =$ unknown

7.5.1 Best results : Keane function

Table 7.4 lists the best results known for the Keane function. No further details are available for the Keane function results provided by Michalewicz in [59]. The reported performances are the best results found. They do not describe the number of repetitions, the mean nor the standard deviation of the results. Schoenauer and

Row	Domain	Evaluation calls	Best	Mean (Std. dev)	Pop . size	Author	Notes
1	$[\pm 600]^{10}$	150,000	–	1 (0)	20	Liu [52]	MeSwarm (PSO)
2	$[\pm 600]^{10}$	62179 (mean)	–	0.001 (0.005)	–	Whitley [91]	(10-bit?) CHC [30]
3	$[\pm 600]^{10}$	13761.5 (mean)	–	0.108 (5.8e-2)	300	Whitley [91]	10-bit GENITOR
4	$[\pm 600]^{10}$	144,024 (mean)	–	0 (0)	300	Whitley [91]	10-bit GENITOR with Random Bit Climbing
5	$[\pm 600]^{10}$	24,354 (mean)	–	0	50	Barbulescu [5]	10 bit CHC (shifting)
6	$[\pm 15]^{10}$	62,500	–	11.985 (3.4305)	125	Angeline [1]	PSO
7	$[\pm 600]^{10}$	200,000	–	3.26 e-2	20	Vesterstrøm [87]	PSO initialised to $[300, 600]^{10}$
8	$[\pm 600]^{10}$	250,000	–	5.61e-3 (4.7e-2)	100	Das [15]	MPSO-TVAC initialised to $[300, 600]^{10}$
9	$[\pm 600]^{10}$	125,000	–	1.512e-2 (1.95e-2)	125	Settles [77]	BS constriction (GA/PSO hybrid)
1	$[\pm 600]^{25}$	100,000	3.21e-9	7.71e-3 (9.6e-3)	61	Herrera [39]	GA (see sect. 7.2.1) †
2	$[\pm 600]^{25}$	100,000	1.76e-12	9.67e-3 (1.32e-2)	61	Herrera [39]	‡
1	$[\pm 600]^{30}$	300,000	–	1.315e-3 (3.47e-3)	100	Ortiz-Boyer [64]	GA using Ext.F crossover operator
2	$[\pm 600]^{30}$	40,000	–	2.095e-3	20	Clerc [12]	E&S PSO algorithm
3	$[\pm 600]^{30}$	200,000	–	1.6e-2 (2.2e-2)	100	Yao [96]	Fast EP with Cauchy mutation
4	$[\pm 600]^{30}$	500,000	–	0 ($\forall \leq 10^{-25}$)	100	Vesterstrøm [86]	DE
5	$[\pm 600]^{30}$	9378 (mean)	–	$\forall \leq 0.1$	30	Eberhart [22]	PSO
6	$[\pm 15]^{30}$	125,000	–	2.5145 (11.3468)	125	Angeline [1]	PSO
7	$[\pm 600]^{30}$	20,000	–	0.996	20	Monson [61]	Kalman Swarm (PSO)
8	$[\pm 600]^{30}$	200,000	–	1.18e-2	20	Vesterstrøm [87]	SEPSO initialised to $[300, 600]^{30}$
9	$[\pm 600]^{30}$	1,500,000	–	1.6e-3 (2.2e-3)	300	Das [15]	PSO-DV initialised to $[300, 600]^{30}$
10	$[\pm 600]^{30}$	250,000	–	3.944e-3 (5.308e-3)	125	Settles [77]	GA
11	$[\pm 600]^{30}$	20,000	–	1.110223e-16	20	Monson [62]	PAKS (Kalman swarm)
1	$[\pm 600]^{100}$	5,000,000	–	5.4210109e-20 (0)	100	Vesterstrøm [86]	DE
2	$[\pm 600]^{100}$	200,000 without change	–	3.74e-2	–	Vesterstrøm [87]	ARPSO
3	$[\pm 600]^{100}$	2,000,000	–	1.25e-2	–	Vesterstrøm [87]	PSO

Figure 7.3: Table of notable results for the Griewank function

Row	Domain	Evaluation calls	Best	Mean (Std. dev)	Pop. size	Author	Notes
1	$[0, 10]^{20}$	700,000	0.80351067	–	70	Michalewicz [59]	GENOCOP III
2	$[0, 10]^{20}$	900,000	0.803553	$\forall \geq 0.802964$	30	Schoenauer & Michalewicz [74, 60]	GA
3	$[0, 10]^{20}$	350,000	0.803515	0.781975 (2e-2)	(20,300λ)	Runarsson & Yao [72]	ES
4	$[0, 10]^{20}$	1,400,000	0.79953	0.79671	70	Koziel & Michalewicz [51]	ALG_g 25-bit precision EA
5	$[0, 10]^{20}$	227,832 (mean)	0.7549125	0.3717081 (9.8e-2)	(50,2)	Hedar [67]	Filtered Simulated Annealing
6	$[0, 10]^{20}$	1,500,000	0.785	0.59	(100+300λ)	Hamida [37]	ASCHEA ES
7	$[0, 10]^{20}$	240,000	0.803601	0.785238 (1.6757e-2)	(100+300λ)	Mezura-Montes [56, 57]	Modified ES
8	$[0, 10]^{20}$	140,000	–	0.7828	70	Xie [95]	SWAF (CR = 0.1)
9	$[0, 10]^{20}$	350,000	0.792608	0.721749	50	Zavala [98]	PESO (modified PSO)
1	$[0, 10]^{50}$	150,000	0.785	0.779	250	Keane [48]	GA 12 bit
2	$[0, 10]^{50}$	700,000	0.83319378	–	70	Michalewicz [59]	GENOCOP III †
3	$[0, 10]^{50}$	–	0.8348	–	–	Bilchev via Michalewicz [59]	Unknown
4	$[0, 10]^{50}$	900,000	0.8331937	$\forall > 0.83$	30	Schoenauer & Michalewicz [74, 60]	GA ‡
5	$[0, 10]^{50}$	200,000 ?	0.828	$\forall \geq 0.814$	100 ?	Surry & Radcliffe [84]	COMOGA (GA)

Figure 7.4: Table of notable results for the Keane function

Michalewicz's 20 dimensional results (line 2) are from a GA that uses an exploit against the location of the optimum. It is thus a result that is illustrative of a strong result.

The 3rd of the 50 dimensional results, reportedly found by Bilchev, is unrepeatable, since it is referenced as a "personal communication". Interestingly, two different 50 dimensional results (here marked with † and ‡ on lines 2 and 4) both report exactly the same result for the 50 dimensional case, however, in [59] Michalewicz does not mention the exploit used in [74, 60] and used GENOCOP III with different population sizes and different maximum evaluation limits. For the purposes of this research we shall assume that each result was independently obtained via the two different methods, and it is simple coincidence that the results are exactly the same to 9 decimal places. Neither work refers to the results of the other, though both [74, 60] cite [59] for other details.

Surry's results in [84] do not have reliable data for the algorithm parameters. The phrase "evaluations" is used to refer to both the number of generations and the number of function calls when describing other authors' work. Their own results for the COMOGA algorithm are described as using 200,000 evaluations for an untuned COMOGA algorithm. The same paper describes an application of the COMOGA algorithm to a real world problem, in which the graphs show a population size of approximately 100 individuals. It is thus possible that the 200,000 evaluations reported here for the COMOGA algorithm were in fact 20,000,000 evaluations.

Hedar [67] uses a form of simulated annealing. The search retains a population of 50 ranked points from which annealing starts. The annealing process involves the generation of 2 individuals per iteration. Stalled searches are restarted from one of the ranked points.

7.5.1.1 Detailed results

In [48] Keane reports values less than 0.8 for all his experiments of up to 140,000 evaluations. Keane used a 16-bit representation and a bitwise genetic algorithm equipped with the popular Fiacco-McCormick constraint penalty function, niching and elitism. Because the optimum is unknown for this function, we use this opportunity to publish the best found domain values for reference. The best published results (found without exploits) on the 20 dimensional case were due to Michalewicz using GENOCOP III [59]. In [72] Runarsson reports a value of 0.803619 as being the best known. The actual domain values are not given, and the result does not occur in any of the tabled

results reported. The circumstances of its location are therefore unknown.

Michalewicz found a value of 0.80351067 for candidate :

3.16311359, 3.13150430, 3.09515858, 3.06016588, 3.03103566,
2.99158549, 2.95802593, 2.92285895, 0.48684388, 0.47732279,
0.48044473, 0.48790911, 0.48450437, 0.44807032, 0.46877760,
0.45648506, 0.44762608, 0.44913986, 0.44390863, 0.45149332

Michalewicz also found the best non-exploit based result for the 50 dimensional case, which is identical to the result reached using an exploit by Schoenauer.

Michalewicz/Schoenauer found a value of 0.83319378 for candidate :

6.28006029, 3.16155291, 3.15453815, 3.14085174, 3.12882447,
3.11211085, 3.10170507, 3.08703685, 3.07571769, 3.06122732,
3.05010581, 3.03667951, 3.02333045, 3.00721049, 2.99492717,
2.97988462, 2.96637058, 2.95589066, 2.94427204, 2.92796040,
0.40970641, 2.90670991, 0.46131119, 0.48193336, 0.46776962,
0.43887550, 0.45181099, 0.44652876, 0.43348753, 0.44577143,
0.42379948, 0.45858049, 0.42931050, 0.42928645, 0.42943302,
0.43294361, 0.42663351, 0.43437257, 0.42542559, 0.41594154,
0.43248957, 0.39134723, 0.42628688, 0.42774364, 0.41886297,
0.42107263, 0.41215360, 0.41809589, 0.41626775, 0.42316407

7.5.2 A new result

Out of curiosity, we sought and obtained a new result on the 20 dimensional Keane function. We wish to emphasise that this is not a normal result, having been obtained through deliberate search and the utilisation of a known exploit to help remedy our poor constraint handling. The best value we obtained was the following, which was found using the standard initialisation. We allowed 3,000,000 evaluations and allowed the use of the simple exploit from section 3.4 for the last 2,780,000 evaluations. The exploit is used to finesse the finish of the search, it is designed to encourage search towards the boundary of the feasible space. By 100,000 evaluations the best candidate has a fitness value of over 0.798. The exploit was engaged after 220,000 evaluations, at which point the value was 0.8027744572783648. The final result scores 0.803618805983517,

which is reached after 2,124,834 evaluations. No further progress is recorded and the rate of progress up to this point was slowing exponentially. This result is extremely strong, equaling that mentioned by Runarsson in [72], which Runarsson believes to be the best known for this problem. We provide our result below for reference.

Using an exploit we record a value of 0.803619 for the candidate:

$$3.162444814275533, 3.1285976379280918, 3.0942874132206226,$$
$$3.061314881522329, 3.0278010372528987, 2.9929127819363748,$$
$$2.959260697548787, 2.9215432094458897, 0.495324807306084,$$
$$0.48896307270866224, 0.4822586741532266, 0.4768060677713827,$$
$$0.4708439953127678, 0.46665737783951794, 0.46084960040891815,$$
$$0.45670617975408057, 0.4526007702090502, 0.44811737287940506,$$
$$0.444187015959455, 0.4402210840106526$$

This candidate avoids breaching the $\prod_{i=1}^{n}(x_i) > 0.75$ constraint by a mere 6.96e-13. While tabulating our results we have noticed that we have difficulty manipulating very small values, due to a break down in floating point precision. It is possible that the improvement of this result further would require a more accurate representation, see section 8.2 for more details.

7.6 Michalewicz's Constraints Suite

These functions are constrained. Parts of the space are unacceptable as solutions. All five of the functions in this suite are from Michalewicz [59]. The reader is referred there for authoritative discussion.

7.6.1 Constrained function #1

$$f_{mcf1}(\vec{x}) = 5x_1 + 5x_2 + 5x_3 + 5x_4 - 5\sum_{i=1}^{4} x_i^2 - \sum_{i=5}^{13} x_i$$

$$\vec{x} = (x_1, ..., x_{13}),\ 0 \leq x_{i \in [1,...,9]} \leq 1,\ 0 \leq x_{10,11,12} \leq 100$$

Subject to constraints :

$$2x_1 + 2x_2 + x_{10} + x_{11} \leq 10$$
$$2x_1 + 2x_3 + x_{10} + x_{12} \leq 10$$
$$2x_2 + 2x_3 + x_{11} + x_{12} \leq 10$$
$$-8x_1 + x_{10} \leq 0$$
$$-8x_2 + x_{11} \leq 0$$
$$-8x_3 + x_{12} \leq 0$$
$$-2x_4 - x_5 + x_{10} \leq 0$$
$$-2x_6 - x_7 + x_{11} \leq 0$$
$$-2x_8 - x_9 + x_{12} \leq 0$$

$$f_{mcf1}(optimum) = -15 \text{ at } (1,1,1,1,1,1,1,1,1,3,3,3,1)$$

This function (f_{mcf1}) has six of the nine constraints active at the optimum.

7.6.2 Constrained function #2

$f_{mcf2}(\vec{x}) = x_1 + x_2 + x_3$

$\vec{x} = (x_1, ..., x_8)$, $100 \leq x_1 \leq 10000$, $1000 \leq x_{2,3} \leq 10000$, $10 \leq x_{i \in [4,...,8]} \leq 1000$

Subject to constraints :
$1 - 0.0025(x_4 + x_6) \geq 0$
$1 - 0.0025(x_5 + x_7 - x_4) \geq 0$
$1 - 0.01(x_8 - x_5) \geq 0$
$x_1 x_6 - 833.33252 x_4 - 100 x_1 + 83333.333 \geq 0$
$x_2 x_7 - 1250 x_5 - x_2 x_4 + 1250 x_4 \geq 0$
$x_3 x_8 - 1250000 - x_3 x_5 + 2500 x_5 \geq 0$

$f_{mcf2}(optimum) = 7049.330923$ at
$(579.3167, 1359.943, 5110.071, 182.0174, 295.5985, 217.9799,$
$286.4162, 395.5979)$

All constraints are active at $f_{mcf2}(optimum)$.

7.6.3 Constrained function #3

$f_{mcf3}(\vec{x}) = (x_1 - 10)^2 + 5(x_2 - 12)^2 + x_3^4 + 3(x_4 - 11)^2 +$
$10 x_5^6 + 7 x_6^2 + x_7^4 - 4 x_6 x_7 - 10 x_6 - 8 x_7$

$\vec{x} = (x_1, ..., x_7)$, $-10 \leq x_i \leq 10$

Subject to constraints :
$127 - 2x_1^2 - 3x_2^4 - x_3 - 4x_4^2 - 5x_5 \geq 0$
$282 - 7x_1 - 3x_2 - 10x_3^2 - x_4 + x_5 \geq 0$
$196 - 23x_1 - x_2^2 - 6x_6^2 + 8x_7 \geq 0$
$-4x_1^2 - x_2^2 + 3x_1 x_2 - 2x_3^2 - 5x_6 + 11x_7 \geq 0$

$f_{mcf3}(optimum) = 680.6300573$ at $(2.330499, 1.951372, -0.4775414,$
$4.365726, -0.6244870, 1.038131, 1.594227)$

Two of the four constraints are active at $f_{mcf3}(optimum)$.

7.6.4 Constrained function #4

$$f_{mcf4}(\vec{x}) = e^{x_1 x_2 x_3 x_4 x_5}$$

$$\vec{x} = (x_1, ..., x_n),\ -2.3 \leq x_{1,2} \leq 2.3,\ -3.2 \leq x_{3,4,5} \leq 3.2$$

Subject to constraints :
$x_1^2 + x_2^2 + x + 3^2 + x_4^2 + x_5^2 = 10$
$x_2 x_3 - 5 x_4 x_5 = 0$
$x_1^3 + x_2^3 = -1$

$f_{mcf4}(optimum) = 0.0539498478$ at $(-1.717143, 1.595709,$
$1.827247, -0.7636413, -0.7636450)$

Due to the equality constraints and the imprecise nature of floating point arithmetic, this function is generally relaxed to an inequality based form where each equality $x = y$ is replaced by a broader $y \leq x \pm \varepsilon \leq y$ for some small violation value ε typically 0.001 or 0.0001 [58, 72].

7.6.5 Constrained function #5

$$f_{mcf5}(\vec{x}) = x_1^2 + x_2^2 + x_1x_2 - 14x_1 - 16x_2 +$$
$$(x_3 - 10)^2 + 4(x_4 - 5)^2 + (x_5 - 3)^2 + 2(x_6 - 1)^2 +$$
$$5x_7^2 + 7(x_8 - 11)^2 + 2(x_9 - 10)^2 + (x_{10} - 7)^2 + 45$$

$$\vec{x} = (x_1, ..., x_10), \; -10 \leq x_i \leq 10$$

Subject to constraints :

$$105 - 4x_1 - 5x_2 + 3x_7 - 9x_8 \geq 0$$
$$-10x_1 + 8x_2 + 17x_7 - 2x_8 \geq 0$$
$$8x_1 - 2x_2 - 5x_9 + 2x_{10} + 12 \geq 0$$
$$-3(x_1 - 2)^2 - 4(x_2 - 3)^2 - 2x_3^2 + 7x_4 + 120 \geq 0$$
$$-5x_1^2 - 8x_2 - (x_3 - 6)^2 - 3x_5^2 + x_6 + 30 \geq 0$$
$$-x_1^2 - 2(x_3 - 2)^2 + 2x_1x_2 - 14x_5 + 6x_6 \geq 0$$
$$-0.5(x_1 - 8)^2 - 2(x_2 - 4)^2 - 3x_5^2 + x_6 + 30 \geq 0$$
$$3x_1 - 6x_2 - 12(x_9 - 8)^2 + 7x_{10} \geq 0$$

$f_{mcf5}(optimum) = 24.3062091$ at $(2.171996, 2.363683, 8.773926,$
$5.095984, 0.9906548, 1.430574, 1.321644, 9.828726, 8.280092, 8.375927)$

Six of the eight constraints are active at $f_{mcf5}(optimum)$.

7.6.6 Best results : Michalewicz's constrained function Suite

Surry's results in [84] are not clear as to the parameters used. Michalewicz in [59] gives results for a real number GA using various constraint handling tactics. Here we list only those results that are not breaching constraints. The majority are from using the GENOCOP II constraint handling mechanism. The dynamic penalties used to obtain the 4^{th} result for f_{mcf5} are due to Jiones and Houck [46] and are described in [59] page 142. Hedar [67] uses a simulated annealing algorithm with a ranked re-annealing pool of 50 independent individuals. During the annealing process two points are generated each iteration. If the search stalls the search is restarted from one of the unused individuals in the ranked pool. Though the results in [67] specify an average of 404.501 fitness function calls when solving f_{mcf5}, this is almost certainly supposed to read 404,501 : None of the other average fitness calls in the work are reported as a decimal, and it is implausible to create the fractional part by division by

Row	Evaluation calls	Best	Mean (Std. dev)	Pop. size	Author	Notes
Function number 1 f_{mcf1}						
1	350,000	-15.0	-15.0	(20,300λ)	Runarsson & Yao [72]	ES
2	1,400,000	-14.7864	-14.7082	70	Koziel & Michalewicz [51]	ALG_g 25-bit precision EA
3	–	-14.997	$\forall \leq$-14.994	100 ?	Surry & Radcliffe [84]	COMOGA (GA)
4	350,000	-15.0	-15.0	70	Michalewicz [59]	GA with GENOCOP II penalties
5	205,748 (mean)	-14.99105	-14.993316 (4.8e-3)	(50,2)	Hedar [67]	Filtered Simulated Annealing
6	1,500,000	-15	-14.84	(100+300λ)	Hamida [37]	ASCHEA ES
7	240,000	-15	-15	(100+300λ)	Mezura-Montes [56, 57]	Modified ES
8	140,000	–	-15	70	Xie [95]	SWAF (CR = 0.1)
9	350,000	-15	-15	50	Zavala [98]	PESO (modified PSO)
Function number 2 f_{mcf2}						
1	350,000	7054.316	7559.192 (5.3e-2)	(20,300λ)	Runarsson & Yao [72]	ES
2	1,400,000	7147.9	8163.6	70	Koziel & Michalewicz [51]	ALG_g 25-bit precision EA
3	–	7081.43	$\forall \leq$8322.51	100 ?	Surry & Radcliffe [84]	COMOGA (GA)
4	350,000	7377.976	$\forall \leq$ 9652.9	70	Michalewicz [59]	GA with GENOCOP II penalties
5	243,520 (mean)	7059.86350	7509.3210442.34)	(50,2)	Hedar [67]	Filtered Simulated Annealing
6	1,500,000	7061.13	7497.434	(100+300λ)	Hamida [37]	ASCHEA ES
7	240,000	7051.902832	7253.047005 (136.023716)	(100+300λ)	Mezura-Montes [56, 57]	Modified ES
8	140,000	–	7214.176	70	Xie [95]	SWAF (CR = 0.9)
9	350,000	7049.495452	7099.101386	50	Zavala [98]	PESO (modified PSO)
Function number 3 f_{mcf3}						
1	350,000	680.630	680.656 (3.4e-2)	(20,300λ)	Runarsson & Yao [72]	ES
2	1,400,000	680.91	681.16	70	Koziel & Michalewicz [51]	ALG_g 25-bit precision EA
3	–	680.663	$\forall \leq$680.755	100 ?	Surry & Radcliffe [84]	COMOGA (GA)
4	350,000	680.642	$\forall \leq$680.955	70	Michalewicz [59]	GA with GENOCOP II penalties
5	324,569 (mean)	680.63008	680.63642 (1.45e-2)	(50,2)	Hedar [67]	Filtered Simulated Annealing
6	1,500,000	680.630	680.641	(100+300λ)	Hamida [37]	ASCHEA ES
7	240,000	680.631592	680.643410 (1.5529e-2)	(100+300λ)	Mezura-Montes [56, 57]	Modified ES
8	140,000	–	680.630	70	Xie [95]	SWAF (CR = 0.9)
9	350,000	680.630057	680.630057	50	Zavala [98]	PESO (modified PSO)

Figure 7.5: Table of notable results for the Michalewicz constraint test suite functions 1, 2 and 3

Row	Evaluation calls	Best	Mean (Std. dev)	Pop. size	Author	Notes
Function number 4 f_{mcf4}						
1	350,000	0.053957	0.067543 (3.1e-2)	(20,300λ)	Runarsson & Yao [72]	ES
2	–	0.058	$\forall \leq 0.57$	100 ?	Surry & Radcliffe [84]	COMOGA (GA)
3	350,000	0.054	$\forall \leq 0.557$	70	Michalewicz [59]	GA with GENOCOP II penalties
4	120,268 (mean)	0.0539498	0.2977204 (0.1887)	(50,2)	Hedar [67]	Filtered Simulated Annealing
5	240,000	0.053986	0.166385 (0.176855)	(100+300λ)	Mezura-Montes [56, 57]	Modified ES
6	350,000	0.081498	0.626881	50	Zavala [98]	PESO (modified PSO)
Function number 5 f_{mcf5}						
1	350,000	24.307	24.374 (6.6e-2)	(20,300λ)	Runarsson & Yao [72]	ES
2	1,400,000	24.620	24.826	70	Koziel & Michalewicz [51]	ALG_g 25-bit precision EA
3	–	24.340	$\forall \leq 24.71$	100 ?	Surry & Radcliffe [84]	COMOGA (GA)
4	350,000	25.486	$\forall \leq 42.358$	70	Michalewicz [59]	GA with dynamic penalties
5	404,501 (mean)	24.311	24.3795271 (7.1625e-2)	(50,2)	Hedar [67]	Filtered Simulated Annealing
6	1,500,000	24.3323	24.6636	(100+300λ)	Hamida [37]	ASCHEA ES
7	240,000	24.326715	24.474926 (0.132385)	(100+300λ)	Mezura-Montes [56, 57]	Modified ES
8	140,000	–	24.306	70	Xie [95]	SWAF (CR = 0.9)
9	350,000	24.306921	24.371253	50	Zavala [98]	PESO (modified PSO)

Figure 7.6: Table of notable results for the Michalewicz constraint test suite functions 4 and 5

30 (the number of repeats performed). Runarsson's Stochastic ranking [72] is at the time of writing (2005) widely regarded as the state of the art algorithm for constrained optimisation [98].

7.7 Powell's 4-Dimensional function

$$f_{Powell}(\vec{x}) = (x_1 + 10x_2)^2 + (x_2 - 2x_3)^4 + (\sqrt{5}x_3 - x_4)^2 + (\sqrt{10}(x_1 - x_4)^2)^2$$

$\vec{x} = (x_1, \ldots, x_4)$, $-5.12 \leq x_i \leq 5.12$
The function is defined for four dimensions only

$f_{Powell}(optimum) = 0$ at $(0,0,0,0)$

Powell's four dimensional function is a non-separable function that uses a four dimensional version of what Whitley terms a "weighted wrap" [89] expansion function. The net effect of such a combination is that there is no "start" dimension from where the problem unravels, but all the dimensions have strong interactions and have to be approached collectively. It is informative to contrast this characteristic with the Whitley Rosenbrock function $Rosenbrock_I$ which also shares this property and the common Rosenbrock function $Rosenbrock_{II}$ which does not.

7.7.1 Best results : Powell's function

It is notable that all the reported results for optimisation of the Powell four dimensional problem are using binary encodings. The precision of a 10-bit representation as used by Whitley (line 1) is much lower than that of the 20-bit representations. The 20-bit representations are also significantly slower to converge than the 10-bit alternative, so the results on lines 2 and 5 are particularly impressive. The method employed by Barbulescu to achieve the result included a shifting representation method. See the discussion of the same algorithm in section 7.4.1 for more details. All representations used in the algorithms reported here are of lower precision than that of a typical floating point representation. A 20 bit representation over this range has an increment size of 9.77e-6.

Row	Domain	Evaluation calls	Best	Mean (Std. dev)	Pop . size	Author	Notes
1	$[\pm 5.12]^4$	4,000,000	–	$\geq$1e-13	500	Whitley [89]	10 bit Island model GENITOR
2	$[\pm 5.12]^4$	200,998.1 (mean)	–	0.0 (0)	–	Whitley [91]	(20-bit?) CHC [30]
3	$[\pm 5.12]^4$	500,000	–	2.92e-4 (7.24e-5)	300	Whitley [91]	20-bit GENITOR
4	$[\pm 5.12]^4$	262,931.2 (mean)	–	3.46e-9 (9.07e-9)	300	Whitley [91]	20-bit GENITOR with Quad Search
5	$[\pm 5.12]^4$	96,497	–	0	50	Barbulescu [5]	20 bit CHC (shifting)

Figure 7.7: Table of notable results for the Powell 4D function

7.8 Rastrigin's function

$$f_{Rastrigin}(\vec{x}) = a.n + \sum_{i=1}^{n}((x_i)^2 - a.cos(\omega.x_i))$$

$\vec{x} = (x_1, .., x_n)$, $-5.12 \leq x_i \leq 5.12$
where $a = 10$ and $\omega = 2\pi$ are constants and n is the number of dimensions.

$f_{Rastrigin}(optimum) = 0$ at $(0, 0, ...0)$

7.8.1 Best results : Rastrigin's function

In [89] Whitley solves the 10 dimensional Rastrigin function using a 10 bit representation and the GENITOR algorithm. It is difficult to tell if the results are indicative of a precise solution, however this is likely. The low resolution of the representation encourages the finalisation of near solutions. It is thus reported here that the system solved the 10 dimensional Rastrigin function in 50,000 evaluations, though the exact figure may be slightly more or less.

Yong et al [97] also have results for the 20, 200, 400 and 500 dimensional Rastrigin function, requiring 3700, 4600, 5375, 6400 fitness evaluations by their Annealing Evolution Algorithm respectively. We have not included these results, since they have the property of requiring a linear increase in function evaluation calls for an exponential increase in problem complexity. Such imperviousness to the problem complexity casts doubt upon their generality. Using 6400 function evaluations Yong reports achieving the same 4 digit accuracy for the 10 dimensional Griewank function as the 500 dimensional Rastrigin function. There are approximately $4.9e^{535}$ local optima in the space. The differentiation of the local and global optimum with 6400 samples has strong indications of an exploit. Of the other results in the Yong paper the results on the 2 and 4 dimensional common Rosenbrock function are far more conventional and are included in this work. No explanation has been provided for how the optimiser described in [97] can search the space so fast, nor has the technique gained popularity since its publication in 1995.

Barbulescu [5] uses a binary representation and Whitley's shifting Gray codes to restart the alogrithm with a different Gray encoded mapping of the space every time the algorithm ceases to progress. This is an interesting ability. Essentially it allows the algorithm to search for represention that make solving the problem temporarily

Row	Domain	Evaluation calls	Best	Mean (Std. dev)	Pop. size	Author	Notes
1	$[\pm 5.12]^{10}$	50,000	0?	–	500	Whitley [89]	10 bit Island model GENITOR
2	$[\pm 5.12]^{10}$	150,000	–	1.95629 (0.76097)	20	Liu [52]	MeSwarm (PSO)
3	$[\pm 5.12]^{10}$	22,297 (mean)	–	0	50	Barbulescu [5]	10 bit CHC (shifting)
4	$[\pm 15]^{10}$	62,500	–	10.965 (0.2383)	125	Angeline [1]	Hybrid PSO
5	$[\pm 10]^{10}$	200,000	–	1.8e-3	20	Vesterstrøm [87]	GA Initialised to $[2.56, 5.12]^{10}$
6	$[\pm 5.12]^{10}$	300,000	–	1.4e-3 (3.9e-3)	100	Das [15]	PSO-DV Initialised to $[2.56, 5.12]^{10}$
7	$[\pm 5.12]^{10}$	125,000	–	0	125	Settles [77]	BS constriction (GA/PSO hybrid)
1	$[\pm 5.12]^{25}$	100,000	8.52e-13	1.13e-11 (1.09e-11)	61	Herrera [39]	GA (§ 7.2.1)
1	$[\pm 5.12]^{30}$	300,000	–	2.189 (1.417)	100	Ortiz-Boyer [64]	GA using BLX(0.3) crossover operator
2	$[\pm 5.12]^{30}$	40,000	–	57.194136	20	Clerc [12]	E&S PSO algorithm
3	$[\pm 5.12]^{30}$	500,000	–	4.6e-2 (1.2e-2)	100	Yao [96]	Fast EP with Cauchy mutation
4	$[\pm 5.12]^{30}$	500,000	–	0 ($\forall \leq 10^{-25}$)	100	Vesterstrøm [86]	DE
5	$[\pm 5.12]^{30}$	6403.50 (mean)	–	$\forall \leq 100$	30	Eberhart [22]	PSO constriction
6	$[\pm 15]^{30}$	125,000	–	42.844 (1.7718)	125	Angeline [1]	Hybrid PSO
7	$[\pm 5.12]^{30}$	20,000	–	53.293	20	Monson [61]	Kalman Swarm (PSO)
8	$[\pm 10]^{30}$	200,000	–	0.4541	20	Vesterstrøm [87]	SEPSO Initialised to $[2.56, 5.12]^{30}$
9	$[\pm 5.12]^{30}$	1,500,000	–	1.6e-3 (0.277)	300	Das [15]	PSO-DV Initialised to $[2.56, 5.12]^{10}$
10	$[\pm 5.12]^{30}$	250,000	–	1.78e-16 (6.41e-16)	125	Settles [77]	BS constriction (GA/PSO hybrid)
11	$[\pm 5.12]^{30}$	20,000	–	29.848747	20	Monson [62]	PAKS (Kalman swarm)
1	$[\pm 5.12]^{100}$	5,000,000	–	0 ($\forall \leq 10^{-25}$)	100	Vesterstrøm [86]	DE
2	$[\pm 5.12]^{100}$	200,000 without change	–	0 ($\forall \leq 5 \cdot 10^{-11}$)	–	Vesterstrøm [87]	ARPSO

Figure 7.8: Table of notable results for Rastrigin's function

easier.

Herrera's 25 dimensional result is the best result from a sweep of different crossover operator combinations. See the discussion in section 7.2.1 for more details.

7.9 Rosenbrock's function

The Rosenbrock function [71] forms a very shallow gradient banana shaped bowl with the optimum towards one end of the "banana". The difficulty of this function is twofold. The gradient is extremely weak approaching the optimum and continues to diminish up to the optimum. The diminishing gradient makes the pursuit of the optimum very challenging, since the signal strength indicating the location of the optimum is decreasing as the optimum is approached, causing the surface to resemble a large plateau to the optimiser. The second challenge of optimising the Rosenbrock function is that the approach to the optimum is non-linear - banana shaped in fact - and thus a simple mechanism to get you across the plateau like remembering the previous gradient information (e.g. inertia in particle swarm optimisation) will fail with high probability. See the example in section 7.9.1 for more details.

Rosenbrock's original function was two dimensional. There are two distinct manners of expanding the function to multiple dimensions. One method of expansion is the Whitley n-dimensional weighted wrap [89]. This is less common than the more obvious linear chaining expansion. The difference in the expansion methods create different networks of interactions, resulting in significant differences in the landscapes and alteration of the order in which the correct setting of variables may be correctly diagnosed.

7.9.1 Original Rosenbrock

$$f_{Rosenbrock}(\vec{x}) = (1-x)^2 + (105(y-x^2)^2)$$

$\vec{x} = (x,y),\ -2.048 \leq x_i \leq 2.048$
The function is only defined for two dimensions.

$$f_{Rosenbrock}(optimum) = 0 \text{ at } (1,1)$$

An example of the surface defined by the two dimensional Rosenbrock function is given in figure 7.9.1. The areas where $f_{Rosenbrock}(x,y) \leq 10$ and $f_{Rosenbrock}(x,y) \leq 1$ are

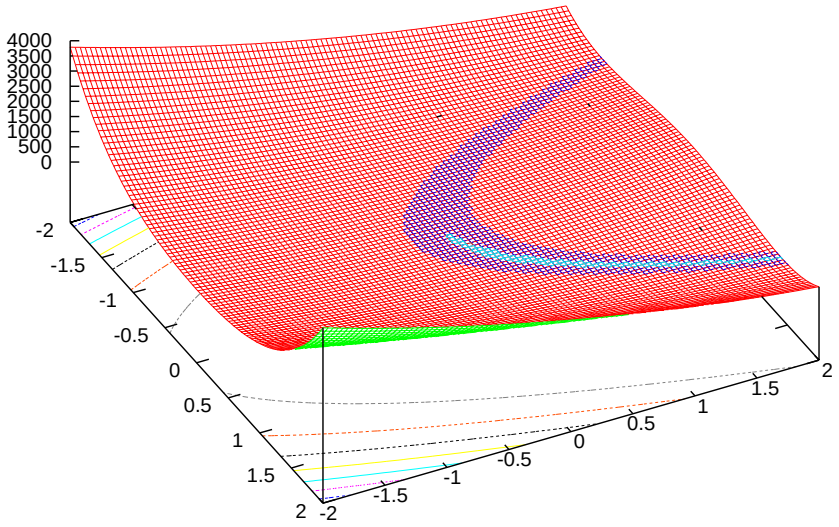

Figure 7.9: The 2D Rosenbrock function, areas where $f_{Rosenbrock}(x,y) \leq 10$ and $f_{Rosenbrock}(x,y) \leq 1$ are highlighted in different colour

highlighted in different colour. Compared to the magnitude of the function elsewhere in the space, the "banana" section of the space is extremely flat. The curvature of the region near the optimum forces a search navigating within the space to make constant corrections; extrapolation of previous samples has a high error probability. These corrections require sampling, but the gradient diminishes as the optimum is approached.

7.9.2 Whitley's N-Dimensional Rosenbrock

$$f_{Rosenbrock_I}(\vec{x}) = \sum_{i=1}^{\lfloor \frac{n}{2} \rfloor} f_{Rosenbrock}(x_{2i-1}, x_{2i}) + \sum_{i=1}^{\lfloor \frac{n-1}{2} \rfloor} f_{Rosenbrock}(x_{2i+1}, x_{2i})$$

$\vec{x} = (x_1, \ldots, x_n)$, $-2 \leq x_i \leq 2$
n is the number of dimensions.

$f_{Rosenbrock_I}(optimum) = 0$ at $(1, 1, \ldots, 1)$

7.9.3 The common N-Dimensional Rosenbrock

$$f_{Rosenbrock_{II}}(\vec{x}) = \sum_{i=1}^{n-1}(100(x_{i+1} - (x_i)^2)^2 + (x_i - 1)^2$$

$\vec{x} = (x_1, ..., x_n),\ -5.12 \leq x_i \leq 5.12$
n is the number of dimensions.

$$f_{Rosenbrock_{II}}(optimum) = 0 \text{ at } (1, 1, ..., 1)$$

7.9.4 Best results : Rosenbrock function

Herrera [39] has two results of interest, a best mean performing algorithm † (using heterogeneous crossover operator: Dynamic Heuristic and Simulated Binary Crossover with μ of two) and a best peak performance algorithm ‡ (using heterogeneous crossover operator : Uniform and Simulated Binary Crossover with μ of 5). We include both here since both the properties, the mean and the best of best, are of interest. The strength of Vesterstrøm's results on the Rosenbrock function is impressive. To the best of our knowledge, these are the best results on the common Rosenbrock function. Whitley [92] is the only author known to publish results on the Whitley Rosenbrock function. The results shown here are found using a Steepest Ascent Bit Climber (SABC) (a bitwise hill climber) with a coordinate rotation mechanism using Principal Component Analysis (PCA).

7.10 Schwefel's sum

$$f_{SchwefelSum}(\vec{x}) = \sum_{j=1}^{j=n}(\sum_{i=1}^{i=j} x_i)^2$$

$\vec{x} = (x_1, ..., x_n),\ -65.536 \leq x_i \leq 65.536$
n is the number of dimensions.

$$f_{SchwefelSum}(optimum) = 0 \text{ at } (0, 0, ..., 0)$$

Originally proposed by Schwefel [76] the doubled sum is a harder version of the sphere problem. Its primary difficulty comes from the fact that gradient based methods are apparently misled by the interactions between the variables [64]. Interaction between variables is sometimes termed epistasis. In this case, according to [64] this causes gradient only based search to take considerable time to reach the vicinity of the

Row	Domain	Evaluation calls	Best	Mean (Std. dev)	Pop. size	Author	Notes
The Whitley Rosenbrock function ($Rosenbrock_I$)							
1	$[\pm 5.12]^5$	148,042 (mean)	5.3e-7	2.4e-6 (1.3e-6)	–	Whitley [92]	PCA SABC
2	$[\pm 5.12]^{10}$	2,496,201 (mean)	3.8 e-6	5.9e-6 (2.2e-6)	–	Whitley [92]	PCA SABC
The common Rosenbrock function ($Rosenbrock_{II}$)							
1	$[\pm 2000]^2$	9525	2.1e-6	–	50	Yong [97]	Annealing Evolution Algorithm
2	$[\pm 5.12]^2$	484,001	4.2e-8	–	50	Corana reported by Yong [97]	Simulated Annealing
3	$[\pm 50]^2$	40,000	–	$0\ (\forall \leq 10^{-6})$	20	Clerc [12]	E&S PSO algorithm
1	$[\pm 200]^4$	53,350	2e-5	–	50	Yong [97]	Annealing Evolution Algorithm
2	$[\pm 5.12]^4$	1,264,001	5.9e-7	–	50	Corana reported by Yong [97]	Simulated Annealing
1	$[\pm 5.12]^{10}$	150,000	–	1.05867e-4 (3.54298e-4)	20	Liu [52]	MeSwarm (PSO)
2	$[\pm 15]^{10}$	62,500	–	38.049 (7.8662)	125	Angeline [1]	Hybrid PSO
3	$[\pm 5.12]^{10}$	200,000	–	1.4197	20	Vesterstrøm [87]	PSO
4	$[\pm 100]^{10}$	300,000	–	4.9e-3 (1.21e-2)	100	Das [16]	DE-RANDSF Initialised to $[15, 30]^{10}$
5	$[\pm 30]^{10}$	125,000	–	3.71e-6 (2.92e-6)	125	Settles [77]	BS constriction (GA/PSO hybrid)
1	$[\pm 5.12]^{25}$	100,000	16.5	21.2 (1.26)	61	Herrera [39]	GA (see sect. 7.2.1)†
2	$[\pm 5.12]^{25}$	100,000	4.2e-2	29.7 (24.7)	61	Herrera [39]	‡
1	$[\pm 2.048]^{30}$	300,000	–	24.63 (1.33)	100	Ortiz-Boyer [64]	GA with CIXL crossover
2	$[\pm 10]^{30}$	40,000	–	39.118488	20	Clerc [12]	"Type 1" PSO algorithm
3	$[\pm 30]^{30}$	2,000,000	–	5.06 (5.87)	100	Yao [96]	Fast EP with Cauchy mutation
4	$[\pm 30]^{30}$	500,000	–	$0\ (\forall \leq 10^{-25})$	100	Vesterstrøm [86]	DE
5	$[\pm 5.12]^{30}$	20062.5 (mean)	–	$\forall \leq 100$	30	Eberhart [22]	PSO constriction
6	$[\pm 15]^{30}$	125,000	–	711.04 (125.4468))	125	Angeline [1]	Hybrid PSO
7	$[\pm 100]^{30}$	20,000	–	3.28e3	20	Monson [61]	Kalman Swarm (PSO)
8	$[\pm 100]^{30}$	200,000	–	15.4375	20	Vesterstrøm [87]	SEPSO Initialised to $[15, 30]^{30}$
9	$[\pm 50]^{30}$	1,500,000	–	2.27e-2 (0.182)	300	Das [15]	PSO-DV Initialised to $[15, 30]^{30}$
10	$[\pm 30]^{30}$	250,000	–	6.248 (4.211)	125	Settles [77]	BS constriction (GA/PSO hybrid)
12	$[\pm 100]^{30}$	20,000	–	7.362558	20	Monson [62]	TRIBES (PSO)
1	$[\pm 30]^{100}$	5,000,000	–	$0\ (\forall \leq 10^{-25})$	100	Vesterstrøm [86]	DE
2	$[\pm 100]^{100}$	200,000 without change	–	88.71	–	Vesterstrøm [87]	ARPSO

Figure 7.10: Table of notable results for the Rosenbrock function

optimum. However, to the casual observer it appears to be identical to the sphere function with a heavy and accumulating emphasis on optimising the early values. This, like the *Rosenbrock*$_{II}$ function has a distinct sequence in which it is easier to solve. The lower dimensions are repeatedly summed by the squared sum emphasis. An improvement of an earlier dimension consequently alters a larger number of summations than an improvement of a later dimension, including the sets of sums that could be altered by any change to a later dimension. These summations are then squared and summed leading to a large difference in value. As an example consider a five dimensional vector $(5,5,5,5,5)$ to which we can make one improvement; we may change any five to a four. If we alter the last dimension we get the vector $(5,5,5,5,4)$ which produces the score $1326 = \sum_{i=1}^{5}(\sum_{j=1}^{j=i} x_j)^2 = 5^2 + 10^2 + 15^2 + 20^2 + 24^2$. If instead we had chosen to alter the first dimension we would have had the vector $(4,5,5,5,5)$, which gives the score $1230 = \sum_{i=1}^{5}(\sum_{j=1}^{j=i} x_j)^2 = 4^2 + 9^2 + 14^2 + 19^2 + 24^2$. Clearly the influence of optimising the initial dimensions is significantly greater than the influence of optimising the later dimensions. The entire problem thus comes unravelled from the first dimensions onwards. The epistatisis between variables appear to be nothing more than strong reinforcement of the location of the optimum and the overall challenge of the problem is questionable.

7.10.1 Best results : Schwefel's sum

Herrera's result on line 1 marked † is questionable, and there may be a typographic error in the printing of [39]. In Herrera's work the shown version of the Schwefel sum function is $\sum_{i=1}^{n}\sum_{j=1}^{j=i} x_j^2$ which sums the sums of the squares of the values. If true this would make the problem easier and into a direct analogue of the Sphere function, however, the published results are not anomalous, so this may be a bracketing error. The *EGNA*$_{BGe}$ used by Ortiz-Boyer and referred to on line 1 of the 30 dimensional results is an estimation of distribution algorithm, specifically the Estimation of Gaussian Network Algorithm.

Row	Domain	Evaluation calls	Best	Mean (Std. dev)	Pop. size	Author	Notes
1	$[\pm 65.536]^{25}$	100,000	6.06e−2	4.55 (4.45)	61	Herrera [39]	GA (see sect. 7.2.1)†
1	$[\pm 65.536]^{30}$	300,000	–	1.995e-3 (2.28e-3)	2000	Ortiz-Boyer [64]	$EGNA_{BGe}$
2	$[\pm 65.536]^{30}$	500,000	–	1.6e-2 (1.4e-2)	100	Yao [96]	Fast EP with Cauchy mutation
3	$[\pm 100]^{30}$	500,000	–	$0\ (\forall \leq 10^{-25})$	100	Vesterstrøm [86]	DE
6	$[\pm 100]^{100}$	5,000,000	–	$0\ (\forall \leq 10^{-25})$	100	Vesterstrøm [86]	DE

Figure 7.11: Table of notable results for the Schwefel sum function

Chapter 8

Validation of design

We validate the design of our algorithm by comparing its performance against significant results found in the literature. Good comparative performance provides evidence that the algorithm is performing in an acceptable manner on these problems. Bad performance would indicate a deficiency in the algorithm design, some property of the search that had not been considered and would indicate the need for further work. We extrapolate from this result and expect the algorithm to perform comparably on similar problems.

An objective of this work has been to keep to a minimum the parameterisation needed to deploy an effective algorithm against known difficult problems. To this end we use only the minimum of parameters that are required to clearly define the problem and the maximum number of objective function evaluations. All other required parameterisation, such as the allocation of fitness calls to the various solving strategies is performed automatically and consistently - there is no difference between initialisations on different problems. As a consequence, properties which may have been used for better problem specific performance have not been explored. The parameterisation is not customised, but rather is the first that successfully combined the properties that local testing implied were necessary. It is likely that searching for a better set of parameters would produce better overall benchmark results than those reported here and it is known that problem specific parameterisation would produce better performance in each individual case.

Beating the benchmarks is not the point of the research, however satisfying when it occurs. The benchmarks are intended to reflect properties of real number optimisation problems that are regarded as hard and appropriate by the community as a whole. The hope is that generally good performance on a diverse set of benchmark problems

will extrapolate to unseen but similar domains. The performances reported against benchmarks tend to be fleeting, with algorithms performing well only for a fragment of the set. In order to obtain the benchmark results published, many authors went to great lengths to customise and improve their algorithms. The results that are published are the best that are reached by any means. Performing comparably on all the benchmarks is then to be considered "good enough"; beating some of them would be excellent, but is not necessary to vindicate the algorithm design. In summary, we consider general ability over the set of benchmarks to be more important than outstanding success in a few isolated examples.

To ease the comparison across the diverse sets of benchmark functions and initialisation and termination criteria used by other authors we choose a simple tactic: we select the most impressive mean and the most impressive best performances from the tables of results in chapter 7 and try to obtain the same performances in the same number of evaluations. We believe this is the fairest way of choosing comparison points, since it gives results without ambiguity. To prevent confusion we clearly identify the results we believe are outstanding and the parameter settings we used. Where authors use different sized function domains, we use the largest to maintain comparability. This means for some functions (e.g. Schwefel's double sum) we have used domains larger than that of some the techniques we are comparing against. It is generally accepted that there is no benefit to be gained by searching a larger domain.

8.1 Parameters used

In all our work we used a population size of 20 candidates. We used a static auto-calibration method that calculated the proportions of fitness calls to allocate to the various search mechanisms from the population size. The population size and the proportionate allocations were static through-out the run. All experiments are results of repeats over 30 independent runs using distinct random seeds. We wish to emphasise that the same algorithm parameter settings were used for all the experiments in this work.

8.2 Our results

Whilst tabulating these results an interesting phenomenon was observed. For certain functions the larger dimensional problems are easier to solve to termination than their

lower dimensional brethren. Clearly the space is not easier to explore, so how can this be? Investigating the cause shows our results to also be (as were Vesterstrøm's) a victim of rounding and mathematical precision. For these experiments the termination criterion is zero error in function evaluation from the known optimum.

Due to implementation details each real coded value in the vector has a minimum value below which it fails to accurately represent the floating point values. When the represented values are sufficiently close to zero, the mathematical operators which are used to evaluate the objective function breach the representable threshold and round to zero. The level at which this occurs is dependent upon the number and type of operations applied. Candidate solutions to n dimensional problems are represented by n element vectors. Most scalable functions are characterised by repeated application of operator combinations over the vector length. The number of applications of the operators is thus greater in higher dimensions. For certain mathematical operations error is incurred, and lengthening the vector may raise the value at which the result of evaluation of the operators (which is for instance a product of the values) rounds to zero. It is in general unlikely that a zero error could ever be achieved by a randomised iterative approximation method. Where zero error is recorded in our results, the mechanism by which it was achieved is almost certainly exhaustion of the accuracy of the representation.

Two sets of results are thus presented. The first set, which is presented in table 8.1 show the results from our attempts to solve each of the problems to zero error. The second set of results is presented in table 8.2 and shows the same experiments evaluated to within an "acceptable" error margin of the optimum. In this case, Keane's function, not having a known optimum, is simply repeated. We chose 10^{-14} as the desired level of accuracy, since this allows us to maintain direct comparison with the significant results published by Vesterstrøm and is, relative to the magnitude of the domain ranges, a considerable degree of accuracy. Several of the accompanying statistics gathered during runs indicate that during division mathematical precision has been lost at the 10^{-11} level and it has been recorded that due to the phenomenon of "catastrophic cancellation" [32] the subtraction operator generates errors at the level of 10^{-14}.

To keep the tables of manageable size, all results have been rounded to 6 decimal places. This can cause incongruities; none of our searches solved the Michalewicz constrained function f_{mcf1} to the desired accuracy (error of $< 10^{-14}$). The mean score was -14.999999999998037, which when rounded for presentation in the table becomes -15. We do not know why the search degenerated at this level of precision.

We have included the proportion of the runs that reached the desired level of accuracy. In the same column the mean number of evaluations used by the search is also included. It should be noted that this is the mean over all the experiments - not only those which were successful, and so reflects the expected cost of running the algorithm to obtain the reported result. Due to the generational nature of our algorithm, it checks for termination conditions and will cease operation only after the termination of the current batch of processes. It may therefore exceed the maximum permitted fitness calls by up to $P-1$ calls for a population size of P candidates. This is of little consequence to our results. The magnitude of this error is tiny compared to the number of evaluations used during the run. In the experiments reported in this section the shortest run is 22,000 evaluations; we used a population size of 20 for all our experiments. Our maximum over-spend of evaluations is then less than 19 iterations, equivalent to 0.1 percent in the worst case and typically much lower.

8.2.1 How to read the tables

The simplest way to read our results tables is as a straight comparison between the notable authors from the literature and ourselves. The experiments are arranged such that the problem definitions are directly comparable, or (as in the case of unusual domain ranges) at least not in our favour.

The iteration limits are chosen to be equal to or lower than those reported by the author of the work against which we compare. Where the other author has published a mean number of iterations, we have had no choice but to use this mean performance as the iteration limit.

We tabulate our results on two separate tables, corresponding to the two levels of acceptable error from the known optimum that were used in our work. In both tables the success rate is given as a percentage and shows the actual rate at which the algorithm reaches the desired level of accuracy. A 40% success rate would indicate that the algorithm reached within the acceptable error margin of the objective in 40% of runs. The other 60% of the runs reached the iteration limit. In such a case, it could be concluded that the conditions of the test were in this instance found to be too demanding; the algorithm failed to consistently reach the desired performance. Conversely a 100% success rate indicates the objective accuracy was reached within the iteration limit by all of the experimental runs. The percentage of times that the objective accuracy is reached within the iteration limit is an approximate indication of how easy the optimi-

sation was found to be. We have indicated in bold all the experiments that obtain 100% success.

Failing to obtain 100% success does not imply a failure. On some benchmark functions no other author has achieved the desired level of accuracy either. It is possible for an experiment to fail to reach success and still be the best performance reported. For instance consider the 25 dimensional Rosenbrock experiments from table 8.1. We do not reach our desired level of accuracy and have a 0% success rate, but we do obtain a mean performance that compares favourably against the performance reported by Herrera in [39]. There are several such results where we miss the target accuracy but obtain mean results at least as strong as those reported by the other authors. These are marked in italic.

8.3 Discussion of our results

Our results show an acceptable performance from the algorithm and do not suggest the requirement for further design modifications of the algorithm itself. This level of performance across such a diverse set of benchmark problems is encouraging, and suggests that further development may produce an even stronger optimiser. The significantly poorer performance on the constrained problems, particularly those with challenging constraint interactions at the optimum (f_{mcf4} and Keane's function) seems to indicate that our constraint handling mechanism, whilst acceptable, could be improved. To aid the reader, discussion of each result is accorded a new paragraph headed by the function name in bold font.

8.3.1 Unconstrained problems

The suite of unconstrained test functions consist of the Ackley, De Jong sphere, Griewank, Powell, Rastrigin, common Rosenbrock, Whitley Rosenbrock and Schwefel's double sum problems. Our performance on the unconstrained problem set is strong, and only Vesterstrøm on the 30 and 100 dimensional common Rosenbrock function records reliably better performance.

Ackley function. When solving the Ackley function to zero error, we are potentially outperformed on the 10 dimensional version by results reported by Settles. We record 80% of runs reaching zero error. Our mean performance is 7.105e-16 with a standard error of 1.445e-15. Settles on the other hand records a mean error of only

Dimension	Evaluation calls	Best	Mean (Std. dev)	Success rate (mean evaluations)	Notable authors
Ackley's function $[\pm 32.768]^n$ Unlimited precision					
10	125,000	0	7.105427e-16 (1.445379e-15)	80% (72,660.167)	Settles [77]
25	100,000	0	1.657933e-15 (1.802705e-15)	*53.333% (79,303.467)*	Herrera [39]
30	250,000	0	7.105427e-16 (1.445379e-15)	80% (125,088.233)	Settles [77]
100	5,000,000	0	0	**100% (106,295.6)**	Vesterstrøm [86]
De Jong's Sphere function $[\pm 100]^n$ Unlimited precision					
10	150,000	3.500244e-259	1.850179e-82 (1.013385e-81)	0% (150,003.567)	Liu [52]
25	100,000	1.805402e-100	2.328288e-64 (1.274222e-63)	*0% (100,004.7)*	Herrera [39]
30	500,000	0	2.156987e-156 (1.181426e-155)	*10% (497,128.267)*	Vesterstrøm [86]
100	5,000,000	0	1.131350e-316 (0)	*60% (2,699,306.5)*	Vesterstrøm [86]
Griewank's function $[\pm 600]^n$ Unlimited precision					
10	24,000	0	0.005001 (0.006620)	*40% (21,804.833)*	Barbulescu [5]
25	100,000	0	0	**100% (31,745.433)**	Herrera [39]
30	500,000	0	0	**100% (37,463.733)**	Vesterstrøm [86]
100	5,000,000	0	0	**100% (81,598.1)**	Vesterstrøm [86]
Keane's function					
20	140,000	0.791540	0.771918 (0.015743)	0% (140,001.233)	Xie [95]
20	240,000	0.792571	0.784278 (0.010214)	0% (240,001.933)	Montes [56, 57]
20	900,000	0.795240	0.788151 (0.007143)	0% (900,002.633)	Schoenauer [74, 60]
20	1,400,000	0.797905	0.790505 (0.004230)	0% (1,400,002.133)	Koziel [51]
50	150,000	0.775609	0.713626 (0.030212)	0% (150,003.4)	Keane [48]
50	900,000	0.824547	0.806703 (0.010705)	0% (900,004.1)	Schoenauer [74, 60]
Michalewicz's constrained functions Unlimited precision					
f_{mcf1}	140,000	-15	-15 (2.282932e-12)	0% (140,002.5)	Xie [95]
f_{mcf2}	350,000	7049.807989	7132.580275 (97.619079)	0% (350,002.967)	Zavala [98]
f_{mcf3}	350,000	680.629031	680.630288 (3.517030e-4)	**100% (134,892.7)**	Zavala [98]
f_{mcf4}	350,000	0.059920	0.229140 (0.186962)	0% (350,003.1)	Runarsson [72]
f_{mcf5}	350,000	24.305719	24.373420 (0.058825)	3.333% (349,694)	Zavala [98]
Powell's function $[\pm 5.12]^n$ Unlimited precision					
4	96,000	2.731008e-125	1.444205e-72 (7.906350e-72)	*0% (96,002.567)*	Barbulescu [5]
Rastrigin's function $[\pm 5.12]^n$ Unlimited precision					
10	22,000	0	0.004549 (0.024709)	83.333% (19,078.733)	Barbulescu [5]
25	100,000	0	0	**100% (55,271.033)**	Herrera [39]
30	250,000	0	0	**100% (69,683.433)**	Settles [77]
100	5,000,000	0	0	**100% (289,130.6)**	Vesterstrøm [86]
Whitley's Rosenbrock function $Rosenbrock_I$ $[\pm 30]^n$ Unlimited precision					
5	148,000	0	7.687302e-27 (3.795481e-26)	*43.333% (122,913.633)*	Whitley [92]
10	2,496,000	0	1.801766e-29 (4.723938e-29)	*83.333% (889,575.267)*	Whitley [92]
Common Rosenbrock function $Rosenbrock_{II}$ $[\pm 30]^n$ Unlimited precision					
2	40,000	0	1.897260e-24 (5.002308e-24)	*10% (39,746.033)*	Clerc [12]
4	53,000	0	3.632764e-22 (1.944882e-19)	*6.667% (52,622.2)*	Yong [97]
10	125,000	8.393973e-30	1.480022e-22 (8.055590e-22)	*0% (125,003.5)*	Settles [77]
25	100,000	2.821536e-4	7.410229 (3.898546)	*0% (100,003.6)*	Herrera [39]
30	500,000	1.345640e-25	1.669340e-4 (6.389568e-4)	0% (500,003.5)	Vesterstrøm [86]
100	5,000,000	7.915798e-19	16.420428 (7.016641)	0% (5,000,003.667)	Vesterstrøm [86]
Schwefel's function $[\pm 100]^n$ Unlimited precision					
25	100,000	7.655976e-48	5.987809e-28 (2.672039e-27)	*0% (100,003.733)*	Herrera [39]
30	500,000	4.173794e-174	2.644485e-84 (1.267983e-83)	*0% (500,004.1)*	Vesterstrøm [86]
100	5,000,000	0	1.552339e-298 (0)	*10% (4,809,376.367)*	Vesterstrøm [86]

Figure 8.1: Table of our results for the benchmark functions, zero permitted error. Bold indicates 100% success at obtaining the target error, italic indicates failure to achieve the target error, but still achieving better mean performance than the compared author.

Dimension	Evaluation calls	Best	Mean (Std. dev)	Success rate (mean evaluations)	Notable authors
Ackley's function $[\pm 32.768]^n$ precision $< 10^{-14}$					
10	125,000	3.552714e-15	6.394885e-15 (1.445379e-15)	**100% (26,658.067)**	Settles [77]
25	100,000	3.552714e-15	6.631732e-15 (1.228336e-15)	**100% (53,501.767)**	Herrera [39]
30	250,000	3.552714e-15	6.394885e-15 (1.445379e-15)	**100% (53,930.767)**	Settles [77]
100	5,000,000	0	4.618528e-15 (2.313736e-15)	**100% (110,361.267)**	Vesterstrøm [86]
De Jong's Sphere function $[\pm 100]^n$ precision $< 10^{-14}$					
10	150,000	1.448823e-21	3.023500e-15 (2.903050e-15)	**100% (10,394.8)**	Liu [52]
25	100,000	8.234157e-17	4.637034e-15 (3.543359e-15)	**100% (17,850.867)**	Herrera [39]
30	500,000	2.236094e-16	4.408824e-15 (2.790458e-15)	**100% (20,346.667)**	Vesterstrøm [86]
100	5,000,000	7.115315e-16	6.124634e-15 (2.341497e-15)	**100% (48,771.233)**	Vesterstrøm [86]
Griewank's function $[\pm 600]^n$ precision $< 10^{-14}$					
10	24,000	2.220446e-16	0.002306 (0.004682)	*63.333% (20162.2)*	Barbulescu [5]
25	100,000	3.330669e-16	6.124730e-15 (2.572184e-15)	**100% (31,603.7)**	Herrera [39]
30	500,000	9.992007e-16	6.735353e-15 (2.548487e-15)	**100% (40,096.367)**	Vesterstrøm [86]
100	5,000,000	1.110223e-16	6.764959e-15 (3.014207e-15)	**100% (75,282.633)**	Vesterstrøm [86]
Keane's function					
20	140,000	0.792037	0.767180 (0.017007)	0% (140,003.533)	Xie [95]
20	240,000	0.792383	0.783684 (0.009208)	0% (240,023.7)	Montes [56, 57]
20	900,000	0.794588	0.790127 (0.003710)	0% (900,002.567)	Schoenauer [74, 60]
20	1,400,000	0.795396	0.788625 (0.007838)	0% (1,400,003.7)	Koziel [51]
50	150,000	0.770846	0.705050 (0.030483)	0% (150,002.7)	Keane [48]
50	900,000	0.821289	0.804965 (0.011491)	0% (900,024.1)	Schoenauer [74, 60]
Michalewicz's constrained functions precision $< 10^{-14}$					
f_{mcf1}	140,000	-15	-15 (1.400288e-12)	0% (140,064.63)	Xie [95]
f_{mcf2}	350,000	7045.675505	7157.928128 (86.934044)	0% (350,019)	Zavala [98]
f_{mcf3}	350,000	680.630117	680.630418 (1.334872e-4)	**100% (121,162.667)**	Zavala [98]
f_{mcf4}	350,000	0.062678	0.297280 (0.178877)	0% (350,019.3)	Runarsson [72]
f_{mcf5}	350,000	24.307399	24.372032 (0.052355)	0% (350,019.1)	Xie [95]
Powell's function $[\pm 5.12]^n$ precision $< 10^{-14}$					
4	96,000	1.658434e-18	4.741058e-15 (2.715405e-15)	**100% (13,226.3)**	Barbulescu [5]
Rastrigin's function $[\pm 5.12]^n$ precision $< 10^{-14}$					
10	22,000	0	2.646061e-12 (1.449307e-11)	96.667% (17,734.2)	Barbulescu [5]
25	100,000	0	0	**100% (54,740.767)**	Herrera [39]
30	250,000	0	0	**100% (64,712.833)**	Settles [77]
100	5,000,000	0	0	**100% (289,004.167)**	Vesterstrøm [86]
Whitley's Rosenbrock function $Rosenbrock_I$ $[\pm 30]^n$ precision $< 10^{-14}$					
5	148,000	1.472437e-16	4.436683e-15 (2.657605e-15)	**100% (28,205.333)**	Whitley [92]
10	2,496,000	1.756040e-16	6.407862e-15 (2.742831e-15)	**100% (82,180.8)**	Whitley [92]
Common Rosenbrock function $Rosenbrock_{II}$ $[\pm 30]^n$ precision $< 10^{-14}$					
2	40,000	1.70803e-16	4.574030e-15 (3.102918e-15)	**100% (11,508.333)**	Clerc [12]
4	53,000	6.419875e-16	4.908578e-15 (2.686512e-15)	**100% (23,019.667)**	Yong [97]
10	125,000	2.526701e-16	5.647693e-15 (3.006637e-15)	**100% (69,104.3)**	Settles [77]
25	100,000	4.879615e-6	7.689351 (4.104761)	*0% (100,121.933)*	Herrera [39]
30	500,000	3.305567e-15	0.003414 (0.011147)	53.333% (463,905.4)	Vesterstrøm [86]
100	5,000,000	6.037577e-15	18.292482 (6.422111)	3.333% (4,986,244.533)	Vesterstrøm [86]
Schwefel's function $[\pm 100]^n$ precision $< 10^{-14}$					
25	100,000	2.425078e-16	6.326215e-15 (3.006701e-15)	**100% (57,114.2)**	Herrera [39]
30	500,000	4.543354e-16	5.812169e-15 (2.368716e-15)	**100% (65,889.833)**	Vesterstrøm [86]
100	5,000,000	2.084813e-16	6.014375e-15 (2.857634e-15)	**100% (148,964.833)**	Vesterstrøm [86]

Figure 8.2: Table of our results for the benchmark functions, permitted error $\leq 10^{-14}$. Bold indicates 100% success at obtaining the target error, italic indicates failure to achieve the target error, but still achieving better mean performance than the compared author

2.34e-16, with a standard deviation of 1.07e-15. The difference is well within the standard deviation of both results. Solving the 10 dimensional Ackley function to an error of 10^{-14} or smaller 100% of our runs reach the desired accuracy, requiring an average of 26,658 evaluations to do so. For all the other scalings of the Ackley function (25, 30 and 100 dimensions) we record significantly better results in terms of the mean performance or the mean number of evaluations used or both.

De Jong's sphere. Our results on the 10, 25, 30 and 100 dimensional variants of De Jong's sphere problem either achieve a better accuracy than that recorded by other authors or achieve the desired accuracy in fewer function calls.

Griewank function. On the ten dimensional Griewank function the 10 bit algorithms used by Whitley [91] and Barbulescu [5] appear to offer better performance. We chose to compete against the results reported by Barbulescu who used a mean of 24,354 evaluations to reach a mean of zero error. Attempting to obtain zero error, we use a limit of 24,000 evaluations and reach a mean of 0.005 with a standard deviation of 0.0066, 40% of our runs reach the objective of zero evaluation error. Raising the level of the acceptable error to 10^{-14} we obtain a mean performance of 0.0023 with a standard deviation of 0.0047, 63% of runs reach within 10^{-14} of the objective. For all other forms of the Griewank function (25, 30 and 100 dimensional) we record 100% of runs obtaining zero error within the desired iteration limits. It is interesting to note that on the standard Griewank domain the 10 bit representation has a precision of 1.173 units. The smallest non-zero fitness value that can be achieved using this representation against the Griewank 10 dimensional function is then 0.068355, significantly larger than the mean performance we obtain.

Powell's 4D function. On the Powell function, we record better results in terms of absolute minimal mean error, or in terms of the number of runs reaching the desired error of less than 10^{-14}. We record a mean score smaller than the smallest non-zero value that can be achieved using a 20 bit representation, in 24,000 evaluations.

Rastrigin's function. On the 10 dimensional Rastrigin function, as with the Griewank function, we have good performance on all except the 10 dimensional variant. Once again it is the 10 bit encodings which beat us. We reach a mean error of 0.004549 with 83% of our runs reaching zero error in 22,000 evaluations. The large standard deviation indicates the series of experiments recorded at least one result of large error. When searching to within an error of 10^{-14}, we obtain a mean error of 2.65e-12 and 96.7% of runs (i.e all but one) reach the desired accuracy using an average of 17,734 evaluations.

Our performance is challenged by Barbulescu using a 10 bit encoding and reaching zero error with 100% of runs within a mean of 22,297 evaluations. Barbulescu reports results using the average number of fitness calls consumed. Because of this vagary and slightness of the performance differences, we do not claim superiority over Barbulescu's results despite the strong evidence of competitive results. For all other forms of the Rastrigin function (25, 30 and 100 dimensional) we record 100% of runs successfully reaching zero error within the specified iteration limits.

In both the 10 dimensional Griewank and 10 dimensional Rastrigin cases, the technologies that were competitive with or beat our results used a 10 bits per dimension representation. It is worth recalling the arguments presented in section 6.3, where we demonstrated that the precision of the 10 bit representation was poor.

Rosenbrock's function. Whitley's Rosenbrock function ($Rosenbrock_I$) appears to be easier to optimise than the common Rosenbrock function ($Rosenbrock_{II}$). This is almost certainly due to the much shorter chain of interactions present in the Whitley Rosenbrock function, allowing the variables to be optimised in a less order dependent manner or equivalently more variables to be optimised at any one time, depending upon your preferred perspective. On the common Rosenbrock function $Rosenbrock_{II}$ Vesterstrøm records better results than us on the 30 and 100 dimensional variants. Vesterstrøm's results are outstanding. On the other (2, 4, 10 and 25) dimensional variants of the common Rosenbrock function we have consistently stronger results, judged by the metric of mean performance, than those published elsewhere. Vesterstrøm's performance would seem to indicate that further improvement of the performance on the common Rosenbrock function is possible and should be an objective of further research.

Schwefel's double sum. Our results on Schwefel's double sum function are significantly stronger in terms of absolute smallest error than any reported by other authors. We attain the desired accuracy of an error of 10^{-14} or smaller in 100% of runs.

8.3.2 Constrained problems

The performance on the constrained problems (Michalewicz's test suite and the Keane function) is actually very good, despite appearing disappointing relative to our performance on the unconstrained problems. In no small part this is due to the crude constraint handling mechanism. Despite this apparent short coming, we have recorded results comparable with those published by other authors for all of the constrained test

cases using precisely the same algorithm and the same parameters throughout.

Michalewicz's constrained functions. We equal the best reported performance for the first and third Michalewicz constrained functions f_{mcf1} and f_{mcf3}. Only Zavala [98] has a better mean performance for the second constrained function f_{mcf2}. We are comprehensively beaten by Hedar [67], Montes [56, 57] and Runarsson [72] on the fourth of Michalewicz's constrained functions f_{mcf4}, which contains four equality constraints. Xie [95] has an outstanding result on the fifth of Michalewicz's constrained functions which neither the results obtained using our technique nor any other reported performance is close to matching. In comparison, on the fifth function (f_{mcf5}) we have comparable performance to that of the second strongest results, as reported by Zavala [98] and Runarsson [72], which have mean performances within one standard deviation of our mean performance.

The Keane function. The Keane function proves difficult for our optimiser. The lack of a known optimum means there is no difference between the conditions reported in table 8.2 and table 8.1. The differences in performance between the two tables are due to the use of different random seeds and are reassuringly slight. Because there is a general lack of consensus amongst the reporting authors, we have repeated our experiments on the Keane function with various evaluation limits to clarify relative standing.

The 20 dimensional variant of the Keane problem has far better representation in the community. Of the results reported, using 240,000 evaluations (due to Mezura-Montes [56, 57]) we compare favourably with four of them (Runarsson & Yao, Hedar, Hamida and Zavala) having advantage in either mean score, number of evaluations or both. We perform comparably against Mezura-Montes having a difference in mean performance within one standard deviation, however Mezura-Montes does record a higher best result than we do. We are definitively outperformed by Schoenauer & Michalewicz, who record a significantly better mean performance using a mechanism that searches only the boundary of the feasible space. Our mean results using the same number of evaluations as Xie (140,000) are within one standard deviation of the results published in [95]. The standard deviation for Xie's results is not published.

We have found only 3 results for the 50 dimensional Keane function against which meaningful comparisons may be made. Once again Schoenauer & Michalewicz obtain the best performing result in terms of mean score. Surry & Radcliffe also record a strong result, though the conditions under which this result was obtained are unclear. Keane's own result on the fifty dimensional problem, published in [48], is significantly

better than that which we record using the same number of evaluations. We do not have results indicating how much better the results may have been with a higher evaluation limit. We record a higher standard deviation on this problem configuration than on any of the other Keane function tests.

With the exception of the 50 dimensional experiment over 150,000 evaluations mentioned above, the standard deviation of the results on the Keane function are low relative to the magnitude of the mean value. We do not therefore expect our results to improve significantly with longer experimental runs. Further improvement would seem to require modification of the algorithm.

Chapter 9

Multi-agent experiments

9.1 Experiments

As described earlier in this work, the evidence that one particular optimiser should be better than another on the multi-agent protocol optimisation problem is based on argument that the problem retains properties of other optimisation problems. To judge whether the development was successful, we need other results for comparison.

We select three other optimisation techniques for use in generating that comparison; random search, random mutation hill climbing and the simple genetic algorithm. Comparison against these 3 techniques illustrates whether the problem has any exploitable structure at all, whether the structure is trivial, and whether the new algorithm design choices were constructive.

The multi-agent systems used in this work are expensive to compute, consequently it is impractical to obtain sufficient empirical results to mitigate the influence of randomised initialisation. To permit empirical comparison over a smaller set of samples than would otherwise be necessary, we use paired testing, and identically initialise the starting conditions of each of the compared runs. This is achieved through the synchronisation of the random seeds used to initialise the pseudo-random number generator. Two seeds are used in each experiment, one for the search algorithm, and one for the multi-agent system simulation. Consequently, irrespective of its consumption of random values, each search algorithm is faced with optimising the same multi-agent system, and the relative performances are due to differences in search and not differences in the initialisations of the simulated multi-agent system.

9.2 Comparative technologies

In the SADDE work, a genetic algorithm was used to search for suitable parameterisations. Equally other techniques could have been used. We present the results of optimising the multi-agent problems described in chapter 2 using four optimisation techniques. Comparison between the techniques gives insight into the relative differences in the quality of search performed. Here we describe the configuration of the other techniques used in the comparison.

9.2.1 Random search

The random search method is the simplest blind search mechanism. Random search proceeds by creating candidate solutions independently and at random. This is repeated for as many evaluations as are permitted and at the end of the search the best candidate sampled is returned.

The performance of random search is expected to equal other methods if there is no structure in the space. Random search may also out perform informed search if a large proportion of the structure in the space is deceptive (see section 5.3 for a discussion of when this is expected to occur). Random search represents the baseline of performance, against which all performances may be compared.

9.2.2 Random mutation hill climbing

Random mutation hill climbing is quite simply the iterated application of Gaussian mutation to a random candidate, where any improvement that is made replaces the current candidate. The process is iterated until the iteration limit is reached. Random mutation hill climbing is dependent upon the efficacy of the mutation operator for the progress of the search. As shown in earlier discussion (see figure 6.2) pure mutation search mechanisms scale poorly. Despite this inability to finalise search, random mutation hill climbers are considered generally effective and robust to search conditions. With its single member search, and non-adaptive operator, random mutation hill climbing is representative of the most primitive form of informed search.

9.2.3 Simple genetic algorithm

The simple genetic algorithm is implemented using real number encoding, pairwise tournament based selection, simple reselection based mutation, and uniform crossover.

The population is 100 individuals, crossover rate is a standard 80%, and the mutation rate is also typical at 1%. The simple genetic algorithm is not a trivial search algorithm. In this context the simple genetic algorithm represents the application of a sophisticated search mechanism which has not had the benefit of problem specific optimisation.

9.2.4 Prototype algorithm

The algorithm introduced in this work is termed a “prototype”, to reflect its status and allow a curt moniker to be used in the presentation of the results. The prototype algorithm is initialised precisely as it has always been initialised throughout this work, the population size and relative proportions of operator usage are exactly as reported earlier in section 8.1 and the algorithm structure is as described in chapter 6.

9.3 The proof of concept system

The role of the proof of concept system is to verify that the protocol encoding of a constraint is an effective means to extract desired behaviours from a multi-agent system. It also provides an interesting optimisation problem in its own right. We include results of optimisations by each of the techniques on the proof of concept problem.

9.3.1 Experimental conditions

The proof of concept multi-agent system is a maximisation problem concerning the parameterisation of a protocol encoding the behaviours of some number of agents and a simulation of their interaction space. We create experiments with 5, 10, and 20 agents participating in the system. We also experiment with different iteration limits, allowing 3000, 4000 or 5000 iterations per experiment. Each experiment is repeated 10 times. More repeats would have been desirable but are not currently feasible with the computing power at the author’s disposal. The mean result, best result, and the standard deviation of the experiments are reported.

9.3.2 Results

Figures 9.1 9.2 and 9.3 graphically show the mean, best and standard deviation of optimising the system for 5, 10 and 20 agents respectivly. The precise results of the

experiments using the proof of concept multi-agent system are provided in table 9.4. Each experiment is repeated with an iteration limit of 3000, 4000 and 5000 evaluations. The graphs show the relative performances of the different search techniques. The problem is a maximisation problem. Higher results are better. For a large proportion of the experiments the best results of all other techniques tested are worse than one standard deviation below the mean result of the prototype algorthm. In all cases, the prototype algorithm has a mean performance better than that of any of the other techniques.

9.3.3 Discussion

Figures 9.1 9.2 and 9.3 show the results tabled in figure 9.4 in a graphical format. The prototype algorithm, the design of which has been the main body of this work, has been applied without modification to the proof of concept multi-agent optimisation problem. The results are strong and consistent, the prototype algorithm clearly outperforming the comparison algorithms in this instance. This is taken as indication that the design decisions made earlier were sound, and the prototype algorithm performs well on all tested experimental configurations.

9.4 The extended SADDE multi-agent system

The SADDE multi-agent system and the necessary extensions that have been made to it are described in chapter 2. The problem defines an optimisation of 6 real number values, which dictate a trading pattern in a simulated multi-agent supply chain. The aim of the problem is to maximise the objective function.

9.4.1 Experimental conditions

The SADDE multi-agent system was used as the basis for a series of experiments with different numbers of agents. We experimented with configuring systems with 6, 12, 48, 96 and 192 agents, using a genetic algorithm, random mutation hill climber, a random sampling algorithm and the prototype algorithm to perform the optimisation. The SADDE multi-agent system is slower to evaluate than the proof of concept system, so we do not experiment with different iteration limits. In each optimisation process 5000 evaluations of the objective function were permitted. Each experiment is repeated 10 times. This is the maximum number of experiments that could be feasibly performed.

9.4.2 Results

Only one set of iteration limits was used in these experiments. All searches were performed with a maximum iteration limit of 5000 samples of the objective function. We perform experiments with different numbers of agents, simulating the requirement for reparameterisations as agents join and leave the system. The stable performance of an optimiser when performing a system reparameterisation is an important property. Figure 9.5 shows the mean result, the best result, and the variance represented as one standard deviation below the mean.

9.4.3 Discussion

As table 9.5 shows, the results of the optimisations on the extended SADDE multi-agent system are similarly structured to the results on the proof of concept optimistation domain in table 9.4. In all the experiments on multi-agent domains, the prototype algorithm has a better mean performance than that of any of the other techniques.

In both the multi-agent system parameterisation problems the prototype algorithm performs acceptably without requiring modification. Using the same parameterisation that we first used for the successful optimisations against the benchmark problems we deliver a reliable optimisation against two multi-agent problems that are difficult to optimise. The fact that all of the optimisations reported in this work are the result of one parameterisation of the search algorithm is especially satisfying, since it implies that the design decisions were, collectively at least, valid. This raises confidence that the successes found here are potentially going to transfer equally strongly onto other similar domains.

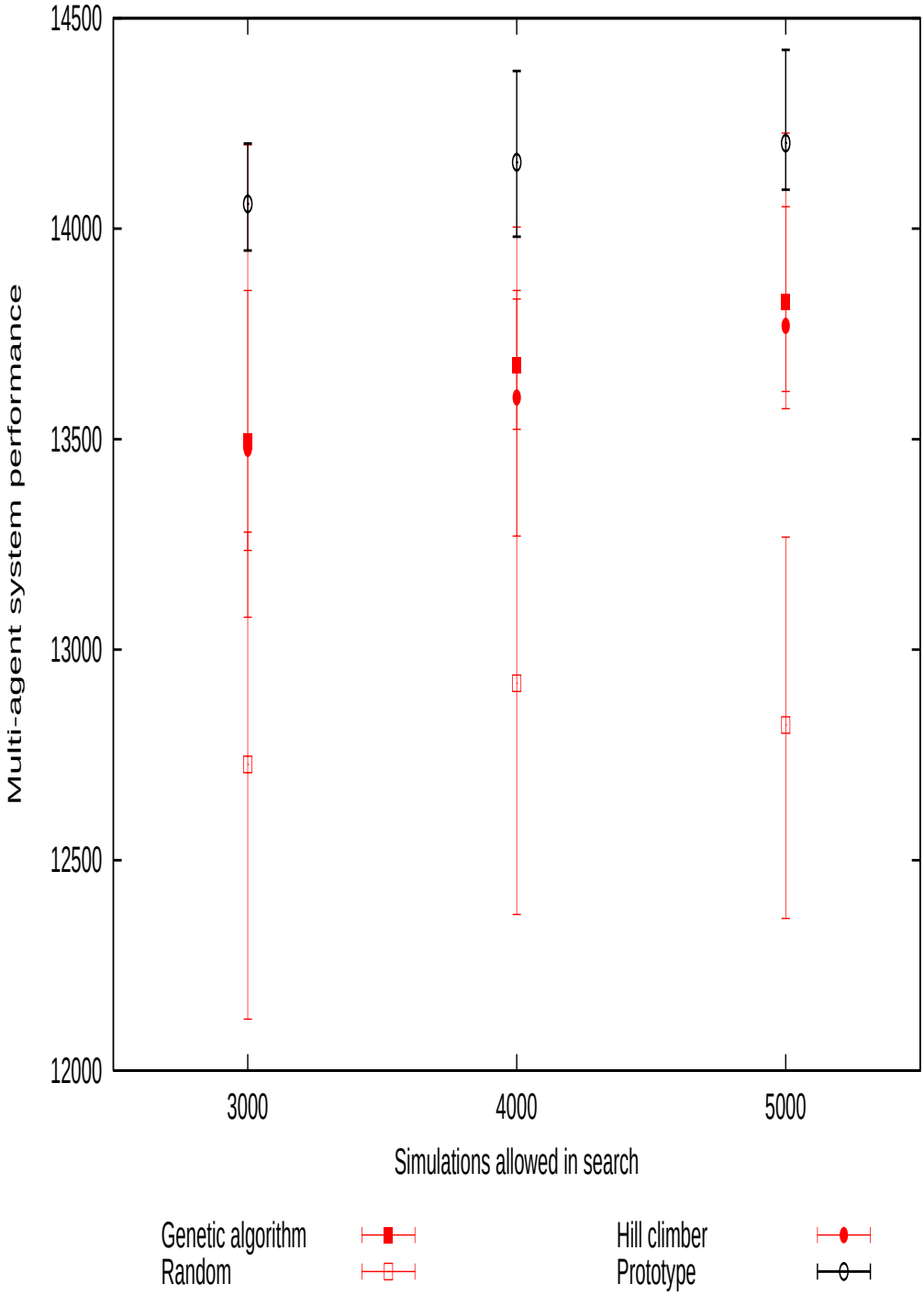

Figure 9.1: Results for the proof of concept multi-agent problem involving 5 agents with 3000, 4000 and 5000 function evaluations per search. Error bars are estimates, the top bar is the best result recorded, the lower is one standard deviation from the mean. “Random”, “Hill climber”, “Genetic algorithm” and “Prototype” indicate the results achieved by random search, random mutation hill climbing, the simple genetic algorithm and the algorithm described in this work respectively

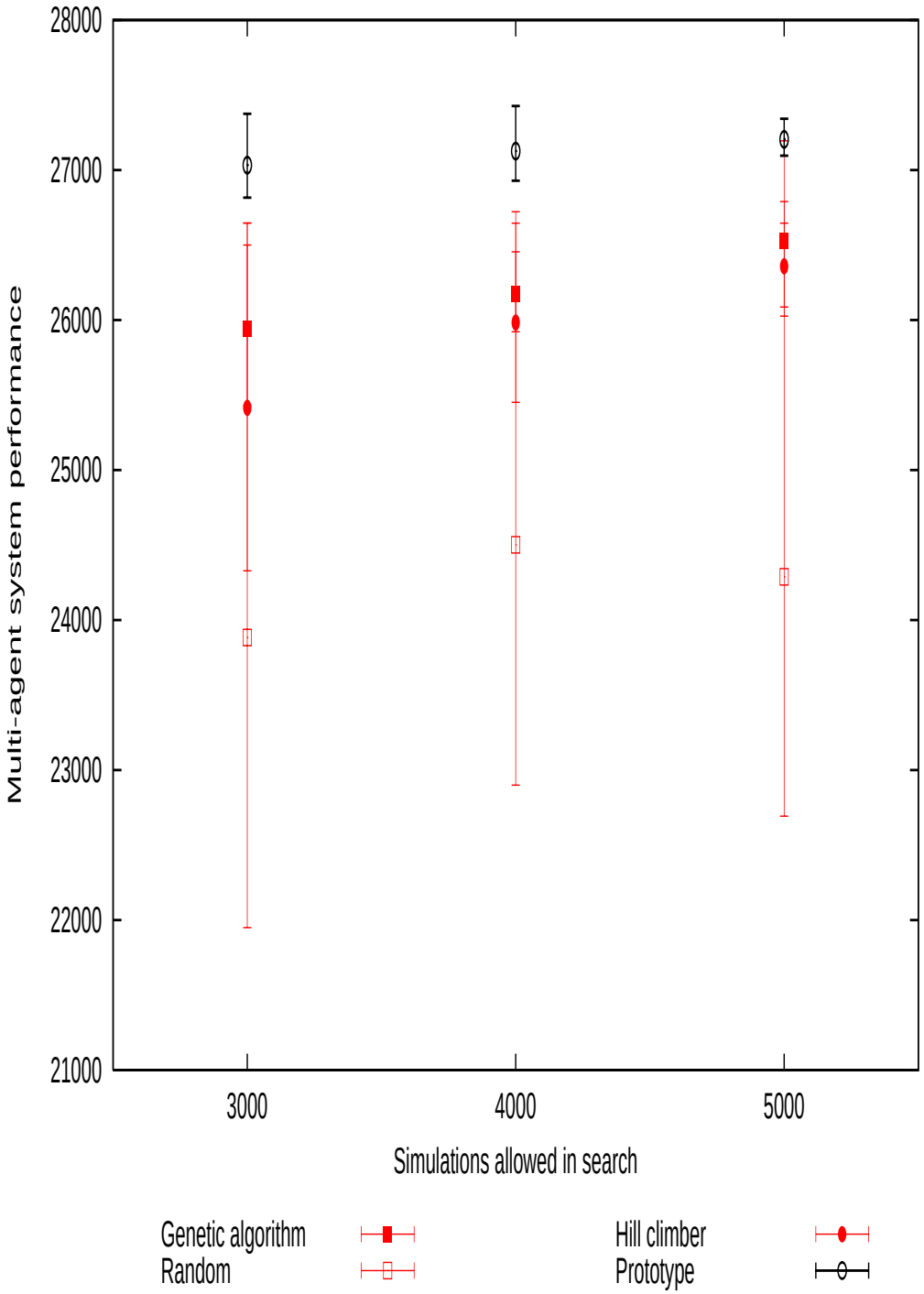

Figure 9.2: Results for the proof of concept multi-agent problem involving 10 agents with 3000, 4000 and 5000 function evaluations per search. Error bars are estimates, the top bar is the best result recorded, the lower is one standard deviation from the mean. “Random”, “Hill climber”, “Genetic algorithm” and “Prototype” indicate the results achieved by random search, random mutation hill climbing, the simple genetic algorithm and the algorithm described in this work respectively

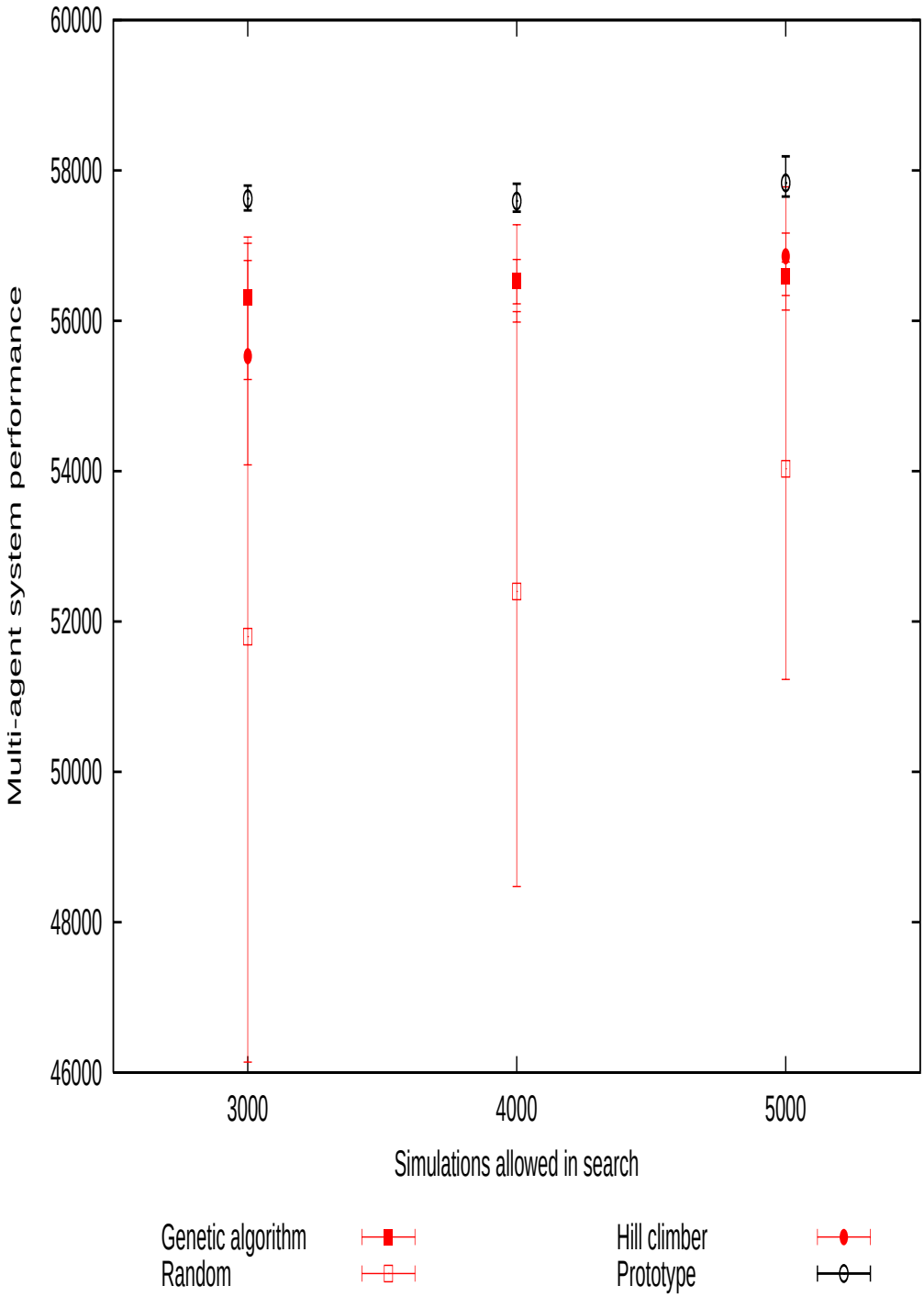

Figure 9.3: Results for the proof of concept multi-agent problem involving 20 agents with 3000, 4000 and 5000 function evaluations per search. Error bars are estimates, the top bar is the best result recorded, the lower is one standard deviation from the mean. “Random”, “Hill climber”, “Genetic algorithm” and “Prototype” indicate the results achieved by random search, random mutation hill climbing, the simple genetic algorithm and the algorithm described in this work respectively

Agents	3000 evaluations			4000 evaluations			5000 evaluations		
	mean	mean - σ	best	mean	mean - σ	best	mean	mean - σ	best
Genetic algorithm									
5	13494.240000	13235.958001	13853.350000	25941.285000	25430.068028	26647.450000	56311.160000	55217.457439	57116.050000
10	13675.375000	13523.728889	13853.350000	26175.425000	25921.400055	26455.300000	56526.770000	56227.174836	56817.400000
20	13825.290000	13613.342228	14227.400000	26528.530000	26086.183121	27195.250000	56596.895000	56144.060708	57168.350000
Hill climber									
5	13477.090000	13076.916105	14199.450000	25414.110000	24328.837359	26499.650000	55529.430000	54084.640018	57029.650000
10	13599.310000	13270.037346	14003.850000	25981.955000	25452.062098	26721.600000	56529.720000	55983.169142	57279.400000
20	13769.300000	13572.137566	14052.400000	26359.330000	26025.355907	26790.450000	56858.845000	56337.112175	57782.250000
Prototype									
5	14059.250000	13948.218433	14202.150000	27032.550000	26816.416534	27374.750000	57593.735000	57451.561208	57821.750000
10	14157.850000	13981.223099	14374.600000	27126.955000	26928.032921	27426.400000	57626.295000	57468.552026	57798.100000
20	14203.405000	14092.866939	14425.250000	27204.230000	27095.092362	27342.100000	57832.270000	57651.945778	58185.950000
Random Sampling									
5	12727.240000	12121.877758	13279.200000	24501.370000	22899.016719	26645.500000	51799.725000	46138.167764	56802.550000
10	12920.250000	12370.474365	13832.800000	24289.040000	22692.965420	26645.500000	52398.265000	48476.498547	56120.000000
20	12820.925000	12361.784442	13267.500000	23884.080000	21950.172859	26645.500000	54031.180000	51229.352581	56779.900000

Figure 9.4: Table of the results on the proof of concept multi-agent domain, these results are also graphed in figures 9.1, 9.2 and 9.3.

Number of Agents	Performance mean	mean - σ	best
	Genetic algorithm		
6	893,738,467	874,928,779	925,889,852
12	3,564,027,433	3,505,828,200	3,645,955,584
48	7,112,287,460	7,037,991,008	7,210,508,565
96	14,212,365,340	14,145,700,508	14,335,988,830
192	28,390,572,258	28,242,011,650	28,594,883,603
	Hill climber		
6	892,870,325	876,044,728	921,243,257
12	1,774,836,839	1,743,295,170	1,813,421,373
48	7,042,423,220	6,963,997,528	7,165,357,049
96	14,063,507,758	13,983,049,192	14,174,877,954
192	28,092,746,881	27,836,132,461	28,285,812,109
	Prototype algorithm		
6	906,863,446	889,780,708	930,471,094
12	3,584,205,701	3,533,140,996	3,644,685,365
48	7,161,112,326	7,104,865,812	7,248,106,298
96	14,298,566,393	14,219,798,397	14,443,658,665
192	28,477,630,500	28,392,315,556	28,591,351,980
	Random sampling		
6	850,709,167	834,176,571	884,642,150
12	1,722,032,719	1,685,488,578	1,807,678,851
48	6,796,911,956	6,664,205,042	6,997,197,634
96	13,600,422,886	13,357,038,915	13,970,903,443
192	27,289,176,329	26,809,852,264	28,136,447,061

Figure 9.5: Table of the results on the SADDE protocol domain. Because the differences between performances are relatively slight relative to the values involved, this set of results has not been graphed.

Chapter 10

Conclusions

10.1 Summary of the work

The purpose of this work has been to guide the reader through the decisions that were made in building a population based algorithm to evolve constraints for a multi-agent system. In the process, we have reviewed the major current real number optimisation technologies, and identified some of the properties that they are believed to use. For the success of this project we rely on the literature in the community and use sets of benchmarks from the literature to test the design of the algorithm, which is seen to perform adequately and not exhibit any obvious failings.

We apply the prototype algorithm to the problem of parameterising constraints in multi-agent systems. Two aspects of multi-agent systems are examined, the first demonstrates the use of protocols in controlling negotiations, the second system is a difficult optimisation problem which is known to have presented difficulties in the past. We provide evidence of strong performance on both problems.

Incidental to obtaining this result we also provide a new best result for the 20 dimensional Keane function and contribute an argument as to why the no free lunch theorem should not (yet) be expected to hinder improvement of optimisation methods.

10.2 Stages in this work

An ideal development process would have directly manipulated properties of the test domain and mapped the relevance of the various properties of the optimisation landscape. We were unable to use this approach in this instance because of the extraordinary cost of evaluating the multi-agent system. We are capable of demonstrating the

existence of structure in the search space. Proof of the existence of structure in the space requires a self similarity measurement, which we graphically illustrated in chapter 5. Without this proof of the existence of correlated structure within the domain, development of an algorithm to exploit structure in the search would have been on dubious ground.

Off-the-shelf optimisation methods all rely on properties in the landscape structure. In some cases what these properties are is unknown and the optimisation method is itself the subject of investigative research. Confident application of an optimisation technique requires both knowledge of the landscape that the optimiser is going to perform on, and a clear understanding of how this landscape is complemented by the properties utilised by the optimisation process. This intimacy of knowledge of the optimisation landscape is not obtainable within the limits on the number of samples available. Instead, using the knowledge that there is local self-correlation in the landscape we build a a composite algorithm designed to use obvious forms of local correlation to locate optima.

We verify the strength of this design by matching the performance of the new optimiser against the best results we could locate. The performance of the algorithm is shown to be exceptional, exhibiting high standards of performance across an extremely wide range of problems. We beat several of the best reported performances and use an exploit to publish a result for the Keane function. Contrary to the majority of search techniques which are biologically or sociologically inspired, the new algorithm is designed to use properties predicted to exist by basic optimisation theory. The contributions of each component in the design of the algorithm are clearly understood, and the performance of the whole algorithm can be explained as nothing more than the sum of its parts.

Having established the power of the new algorithm in external competition, we turn to the multi-agent system optimisation task. Without external standards to verify the performance, we use exemplar technologies for comparison, each chosen to illustrate the use of different degrees of structure within the search. By proving the new algorithm clearly offers an advantage on the tested multi-agent system parameterisation problems, we prove, to as great an extent as is practicable, that the design decisions were both correct and effectively implemented.

10.3 Contributions

This work examines the difficulty of designing a population based algorithm for optimisation of a multi-agent system parameterisation domain on which very little a-priori information is available. In part, this lack of information is caused by the unwieldy nature of sampling in the problem domains; domain samples are extremely time consuming to evaluate. For this reason it is inappropriate to directly use the domain in development of the the optimisation algorithm, a process which even for simple integrity testing is likely to consume millions of samples. The use of surrogate evaluation functions in the algorithm design stages is unorthodox, and parts of this work deal with the repercussions of this decision.

In tackling the difficulties of designing an optimiser for the multi-agent system parameterisation domain, we contribute both to the fields of evolutionary algorithm design and to the field of agent protocol design. We also make significant contributions to the understanding of the scope of the no free lunch theorem in optimisation, and provide several strong benchmark results. Out of curiosity and academic interest, we locate and publish a new result on the much studied Keane's function.

In turn the main contributions of this work are:

1. Evidence that a simple "first principles" modular design can compete with the strongest published results in the field.

2. New best results on several optimisation benchmarks.

3. Publish best result on the 20 dimensional Keane problem, show evidence that further improvement will require a higher precision representation.

4. Discussion of the "No Free Lunch" theorem in optimisation, where we show that the no free lunch theorem will not necessarily withhold further improvement from well designed optimisers, at least until optimality in search is reached.

5. A re-interpreted, repeatable implementation of the SADDE multi-agent system for future comparative work.

6. A self-similarity measure that allows the detection of structure in an optimisation domain.

7. Comparative results between the new optimisation algorithm and three common types of search algorithms show that, under these circumstances, the new algorithm is better suited to optimisation on these domains.

The greatest contribution of this work is demonstrating what can be achieved, given the right circumstances, from such apparently unpromising beginnings. Despite the potential difficulties of using surrogate evaluation functions the algorithm design process was successful in developing an algorithm that performs well in optimising the multi-agent parameterisation problem. The results reported in this thesis, both against the surrogate evaluation functions and the multi-agent problems are strong.

10.4 Conclusion

This work achieved the objective of designing an optimisation algorithm for searching the constraint space of a multi-agent system protocol. The majority of the design effort was aimed at extrapolating relevant optimisation characteristics from well known problems in the real number optimisation literature. In order for this effort to be legitimate, we have to relate our design method to the context of the no free lunch theorem. We show the existence of structure in at least one form of the multi-agent problem. We then place a gamble on the structure in the multi-agent problem being of an orthodox type, and apply the prototype optimisation algorithm. For comparative purposes we also implement and apply random search, random mutation hill climbing and a simple genetic algorithm. The prototype algorithm is shown to have consistently better results than the other methods.

10.5 Further work

The algorithm implemented here is actually relatively simple. Each component of the algorithm is as basic as the required functionality permits. The algorithm structure: multiple levels of search performed with intelligent sampling and replacement, shows strong potential. Using a modular design has the appealing property of allowing the selection of various mechanisms for the different tasks.

Several of the problems referred to in section 6.9 are relatively easy to fix, and should be investigated before further work proceeds. Future work should focus on the identification of the different aspects of the search. Once these are identified the focus should move onto the identification of the best mechanisms for achieving each aspect of the search. Ultimately the goal will be to incorporate these mechanisms, so that mechanisms formerly used in isolation are incorporated as functional components in a modular search superstructure.

Thanks

The author would like to thank John Levine for starting him on this path, and Jacques Fleuriot and Dave Robertson for keeping him on it.

Bibliography

[1] P. J. Angeline. Using selection to improve particle swarm optimization. In *ICEC-98*, pages 84–89, Anchorage, Alaska, 1998. IEEE World Congress on computational intelligence.

[2] Peter J. Angeline. Evolutionary optimization versus particle swarm optimization: Philosophy and performance differences. In *Evolutionary Programming*, pages 601–610, 1998.

[3] Thomas Bäck. *Evolutionary Algorithms in Theory and Practice*. Oxford University Press, New York, 1996.

[4] Thomas Bäck and Hans-Paul Schwefel. An overview of evolutionary algorithms for parameter optimization. *Evol. Comput.*, 1(1):1–23, 1993.

[5] Laura Barbulescu, Jean-Paul Watson, and L. Darrell Whitley. Dynamic representations and escaping local optima: Improving genetic algorithms and local search. In *AAAI/IAAI*, pages 879–884, 2000.

[6] Tobias Blickle and Lothar Thiele. A comparison of selection schemes used in genetic algorithms. Technical Report 11, Gloriastrasse 35, 8092 Zurich, Switzerland, 1995.

[7] W. Brauer, M. Nickles, M. Rovatsos, G. Weiss, and K. F. Lorentzen. Expectation-oriented analysis and design. *Lecture Notes in Computer Science*, 2222 / 2002, 2002. Springer-Verlag GmbH.

[8] Steffen Christensen and Franz Oppacher. What can we learn from no free lunch? a first attempt to characterize the concept of a searchable function. In Lee Spector, Erik D. Goodman, Annie Wu, W.B. Langdon, Hans-Michael Voigt, Mitsuo Gen, Sandip Sen, Marco Dorigo, Shahram Pezeshk, Max H. Garzon, and Edmund Burke, editors, *Proceedings of the Genetic and Evolutionary Computation*

Conference (GECCO-2001), pages 1219–1226, San Francisco, California, USA, 7-11 July 2001. Morgan Kaufmann.

[9] M. Clerc. The swarm and the queen. towards a deterministic and adaptive paricle swarm optimization. *The 1999 Congress on Evolutionary Computation CEC 99, 1999*, 1999.

[10] M. Clerc. When ant colony optimization does not need swarm intelligence, 2000. Unpublished manuscript. Available from *http* : *//clerc.maurice.free.fr/pso/aco/ACO_swarm_intelligence.zip*.

[11] M. Clerc. Tribes - un exemple d'optimisation par essaim particulaire sans paramtres de contrle. *Proceeding of the Optimisation par Essaim Particulaire 2003 (OEP 2003)*, 2003. English version available at :: *http* : *//clerc.maurice.free.fr/pso/Tribes/Tribes_doc.zip*.

[12] M. Clerc and J. Kennedy. The particle swarm - explosion, stability, and convergence in a multidimensional complex space. *IEEE-EC*, 6:58–73, February 2002. Draft at :: *http* : *//clerc.maurice.free.fr/pso/*.

[13] Joseph C. Culberson. On the futility of blind search: An algorithmic view of "no free lunch". *Evolutionary Computation*, 6(2):109–127, 1998.

[14] Bei C. D. and Gray R. M. An improvement of the minimum distortion encoding algorithm for vector quantization. *IEEE Transactions on Communications, COM33 (10):1132–1133*, 1985.

[15] Swagatam Das, Amit Konar, and Uday Kumar Chakraborty. Improving particle swarm optimization with differentially perturbed velocity. In *GECCO*, pages 177–184, 2005.

[16] Swagatam Das, Amit Konar, and Uday Kumar Chakraborty. Two improved differential evolution schemes for faster global search. In *GECCO*, pages 991–998, 2005.

[17] K. De Jong. Genetic algorithms are not function optimizers. 1993.

[18] Kenneth A. De Jong. *An analysis of the behavior of a class of genetic adaptive systems*. PhD thesis, University of Michigan, Ann Arbor, 1995. Dissertation Abstracts International 36(10), 5140B; UMI 76-9381.

[19] Marco Dorigo, Vittorio Maniezzo, and Alberto Colorni. Ant system: Optimization by a colony of cooperating agents. *IEEE Trans. on Systems, Man, and Cybernetics–Part B*, 26(1):29–41, 1996.

[20] Stefan Droste, Thomas Jansen, and Ingo Wegener. Optimization with randomized search heuristics – the (A)NFL theorem, realistic scenarios, and difficult functions. to appear: in a special issue of the Journal of Theoretical Computer Science, 2001.

[21] R. Eberhart and J. Kennedy. A new optimiser using particle swarm theory. *Sixth international symposium on Micro Machine and Human Science*, 1995. 0-7803-2676-8/95.

[22] R. C. Eberhart and Y. Shi. Comparing inertia weights and constriction factors in particle swarm optimization. *The 2000 Congress on Evolutionary Computation CEC 00, 2000*, 2000.

[23] R. C. Eberhart and Y. Shi. Particle swarm optimization: Developments, applications and resources. *The 2000 Congress on Evolutionary Computation CEC 00, 2000*, 2001.

[24] A.E. Eiben. *Multi-parent Recombination*, chapter 3.7. IOP Publishing Ltd and Oxford University Press, 1995.

[25] Agoston E. Eiben and Thomas Bäck. Empirical investigation of multiparent recombination operators in evolution strategies. *Evolutionary Computation*, 5(3):347–365, 1998.

[26] Thomas English. No more lunch: Analysis of sequential search. In *Proceedings of the 2004 IEEE Congress on Evolutionary Computation*, pages 227–234, Portland, Oregon, 20-23 June 2004. IEEE Press.

[27] Thomas English. On the structure of sequential search: Beyond “no free lunch”. In Jens Gottlieb and Günther R. Raidl, editors, *Evolutionary Computation in Combinatorial Optimization – EvoCOP 2004*, volume 3004 of *LNCS*, pages 95–103, Coimbra, Portugal, 5-7 April 2004. Springer Verlag.

[28] Thomas M. English. Evaluation of evolutionary and genetic optimizers: No free lunch. In Lawrence J. Fogel, Peter J. Angeline, and Thomas Bäck, editors, *Evo-*

lutionary Programming V: Proc. of the Fifth Annual Conf. on Evolutionary Programming, pages 163–169, Cambridge, MA, 1996. MIT Press.

[29] Thomas M. English. Practical implications of new results in conservation. In Marc Schoenauer, Kalyanmoy Deb, Günter Rudolph, Xin Yao, Evelyne Lutton, Juan Julian Merelo, and Hans-Paul Schwefel, editors, *Parallel Problem Solving from Nature – PPSN VI*, pages 69–78, Berlin, 2000. Springer.

[30] Larry J. Eshelman. The chc adaptive search algorithm: How to have safe search when engaging in nontraditional genetic recombination. In *FOGA*, pages 265–283, 1990.

[31] Gary B. Fogel, Garrison W. Greenwood, and Kumar Chellapilla. Evolutionary computation with extinction: Experiments and analysis. In *Proc. of the 2000 Congress on Evolutionary Computation*, pages 1415–1420, Piscataway, NJ, 2000. IEEE Service Center.

[32] David Goldberg. What every computer scientist should know about floating-point arithmetic. *ACM Computing Surveys*, 23(1):5–48, 1991.

[33] David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1989.

[34] David E. Goldberg. *The Design of Innovation: Lessons from and for Competent Genetic Algorithms*. Kluwer Academic Publishers, Norwell, MA, USA, 2002.

[35] D.E. Goldberg and K. Deb. A comparison of selection schemes used in genetic algorithms. pages 69–93. Addison-Wesley, 1991.

[36] A. O. Griewank. Generalized descent for global optimization. *Journal of Optimization Theory and Applicalions, 34:11–39, 1981.*, 1981.

[37] Sana Ben Hamida and Marc Schoenauer. ASCHEA: New results using adaptive segregational constraint handling. In David B. Fogel, Mohamed A. El-Sharkawi, Xin Yao, Garry Greenwood, Hitoshi Iba, Paul Marrow, and Mark Shackleton, editors, *Proceedings of the 2002 Congress on Evolutionary Computation CEC2002*, pages 884–889. IEEE Press, 2002.

[38] Nikolaus Hansen and Stefan Kern. Evaluating the CMA evolution strategy on multimodal test functions. In *PPSN*, pages 282–291, 2004.

[39] F. Herrera, M. Lozano, and A. M. Sanchez. Hybrid crossover operators for real-coded genetic algorithms: An experimental study. In press 2005.

[40] John H. Holland. Genetic algorithms and the optimal allocation of trials. *SIAM Journal of Computing*, 2(2):88–105, June 1973.

[41] John H. Holland. Genetic algorithms and the optimal allocation of trials. *SIAM J. Comput.*, 2(2):88–105, 1973.

[42] John H. Holland. *Adpatation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, MI, 1975.

[43] Abdollah Homaifar, H. Y. Lai, and Vance E. McCormick. System optimization of turbofan engines using genetic algorithms. *Appl. Math. Modelling*, 18(2):72–83, 1994.

[44] Christian Igel and Marc Toussaint. On classes of functions for which no free lunch results hold. *Information Processing Letters*, 86:317–321, 2003. See also Los Alamos Preprint cs.NE/0108011.

[45] Y. Jin. A comprehensive survey of fitness approximation in evolutionary computation. *Soft Computing Journal*, 2003.

[46] J. Joines and C. Houck. the use of non-stationary penalty functions to solve nonlinear constrained optimization problems with gas, 1994.

[47] A. Keane. Genetic algorithms digest thursday, may 19, 1994 volume 8 : Issue 16.

[48] A. J. Keane. A brief comparison of some evolutionary optimization methods, February 21 1996.

[49] J. Kennedy and R. Mendes. Population structure and particle swarm performance. In David B. Fogel, Mohamed A. El-Sharkawi, Xin Yao, Garry Greenwood, Hitoshi Iba, Paul Marrow, and Mark Shackleton, editors, *Proceedings of the 2002 Congress on Evolutionary Computation CEC2002*, pages 1671–1676. IEEE Press, 2002.

[50] James Kennedy and Russell C. Eberhart. Particle swarm optimization. In *Proceedings of the 1995 IEEE International Conference on Neural Networks*, volume 4, pages 1942–1948, Perth, Australia, IEEE Service Center, Piscataway, NJ, 1995.

[51] Slawomir Koziel and Zbigniew Michalewicz. Evolutionary algorithms, homomorphous mappings, and constrained parameter optimization. *Evolutionary Computation*, 7(1):19–44, 1999.

[52] Bo-Fu Liu, Hung-Ming Chen, Jian-Hung Chen, Shiow-Fen Hwang, and Shinn-Ying Ho. Meswarm: memetic particle swarm optimization. In *GECCO*, pages 267–268, 2005. full paper available on request.

[53] M. Locatelli. A note on the griewank test function. *Journal of Global Optimization*, 25:169 – 174, 2003. Issue 2.

[54] J. McGinnis and D. Robertson. Dynamic and distributed interaction protocols. In *Proceedings of the Fourth Symposium on Adaptive Agents and Multi-Agent Systems*, pages 45–54, 2004.

[55] J. McGinnis and D. Robertson. Realising agent dialogues with distributed protocols. In *Proceedings of the Autonomous Agents and Multiagent Systems Workshop on Agent Communication*, 2004.

[56] Efrén Mezura-Montes and Carlos A. Coello Coello. A simple evolution strategy to solve constrained optimization problems. In Bart Rylander, editor, *Genetic and Evolutionary Computation Conference Late Breaking Papers*, pages 227–234, Chicago, USA, 12–16 July 2003.

[57] Efrén Mezura-Montes and Carlos A. Coello Coello. An improved diversity mechanism for solving constrained optimization problems using a multimembered evolution strategy. In *GECCO (1)*, pages 700–712, 2004.

[58] Zbigniew Michalewicz. Genetic algorithms numerical optimization and constraints. In *ICGA*, pages 151–158, 1995.

[59] Zbigniew Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. 1999. 3rd, rev. and extended ed. 1996. Corr. 2nd printing, 1998 ISBN: 3-540-60676-9.

[60] Zbigniew Michalewicz, Girish Nazhiyath, and Maciej Michalewicz. A note on usefulness of geometrical crossover for numerical optimization problems. In *Evolutionary Programming*, pages 305–312, 1996.

[61] Christopher K. Monson and Kevin D. Seppi. The kalman swarm: A new approach to particle motion in swarm optimization. In *GECCO (1)*, pages 140–150, 2004.

[62] Christopher K. Monson and Kevin D. Seppi. Bayesian optimization models for particle swarms. In *GECCO '05: Proceedings of the 2005 conference on Genetic and evolutionary computation*, pages 193–200, New York, NY, USA, 2005. ACM Press.

[63] H. Mühlenbein, M. Schomisch, and J. Born. The parallel genetic algorithm as function optimizer. In *Proc. of the Fourth International Conference on Genetic Algorithms*, pages 271–278, San Diego, CA, 1991.

[64] D. Ortiz-Boyer, C. Hervs-Martnez, and N. Garca-Pedrajas. Cixl2: A crossover operator for evolutionary algorithms based on population features. *Journal of Artificial Intelligence Research, Volume 24*, pages 1–48., 2005.

[65] K. E. Parsopoulos and Michael N. Vrahatis. Recent approaches to global optimization problems through particle swarm optimization. *Natural Computing*, 1(2-3):235–306, 2002.

[66] Riccardo Poli, Cecilia Di Chio, and William B. Langdon. Exploring extended particle swarms: a genetic programming approach. In *GECCO '05: Proceedings of the 2005 conference on Genetic and evolutionary computation*, pages 169–176, New York, NY, USA, 2005. ACM Press.

[67] Abdel rahman Hedar and Masao Fukushima. Derivative-free filter simulated annealing method for constrainted continuous global optimization, 05 2005. This paper was originally published in April 2004, then revised in April 2005.

[68] Ingo Rechenberg. *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. frommann-holzbog, Stuttgart, 1973. German.

[69] D. Robertson. A lightweight method for coordination of agent oriented web services. In *Proceedings of AAAI Spring Symposium on Semantic Web Services*, California, USA, 2004.

[70] David Robertson. A lightweight coordination calculus for agent systems. In *DALT*, pages 183–197, 2004.

[71] H. H. Rosenbrock. An automatic method for finding the greatest or least value of a function. *Computer Journal, (3):175-184, 1960.*, 1960.

[72] Thomas P. Runarsson and Xin Yao. Stochastic ranking for constrained evolutionary optimization. *IEEE Transactions on Evolutionary Computation*, 4(3):284–294, 2000.

[73] Mehrdad Salami and Tim Hendtlass. The fast evaluation strategy for evolvable hardware. *Genetic Programming and Evolvable Machines*, 6(2):139 – 162, 2005.

[74] Marc Schoenauer and Zbigniew Michalewicz. Evolutionary computation at the edge of feasibility. In Hans-Michael Voigt, Werner Ebeling, Ingo Rechenberger, and Hans-Paul Schwefel, editors, *PPSN*, volume 1141 of *Lecture Notes in Computer Science*, pages 245–254. Springer, 1996.

[75] C. Schumacher, M. D. Vose, and L. D. Whitley. The no free lunch and problem description length. In Lee Spector, Erik D. Goodman, Annie Wu, W.B. Langdon, Hans-Michael Voigt, Mitsuo Gen, Sandip Sen, Marco Dorigo, Shahram Pezeshk, Max H. Garzon, and Edmund Burke, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, pages 565–570, San Francisco, California, USA, 7-11 July 2001. Morgan Kaufmann.

[76] H. P. Schwefel. Evolution and optimum seeking., 1995.

[77] Matthew Settles and Terence Soule. Breeding swarms: a GA/PSO hybrid. In *GECCO*, pages 161–168, 2005.

[78] Yuhui Shi and Russell C. Eberhart. Empirical study of particle swarm optimization. In Peter J. Angeline, Zbyszek Michalewicz, Marc Schoenauer, Xin Yao, and Ali Zalzala, editors, *Proceedings of the Congress of Evolutionary Computation*, volume 3, pages 1945–1950, Mayflower Hotel, Washington D.C., USA, 6-9 July 1999. IEEE Press.

[79] Carles Sierra, Jordi Sabater, Jaume Agustí, and Pere Garcia. Evolutionary programming in SADDE. In Maria Gini, Toru Ishida, Cristiano Castelfranchi, and

W. Lewis Johnson, editors, *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'02)*, pages 1270–1271. ACM Press, July 2002.

[80] Carles Sierra, Jordi Sabater, Jaume Agusti, Pere Garcia, Steve Phelps, Simon Parsons, Peter McBurney, Elizabeth Sklar, and David Robertson. Adaptive computation and ecological modeling, 2002.

[81] Carles Sierra, Jordi Sabater, Jaume Agustí-Cullell, and Pere Garcia. Evolutionary computation in MAS design. In *ECAI*, pages 188–192, 2002.

[82] Carles Sierra, Jordi Sabater, Jaume Agustí-Cullell, and Pere Garcia. Integrating evolutionary computing and the SADDE methodology. In *AAMAS*, pages 1116–1117, 2003.

[83] Krzysztof Socha. ACO for Continuous and Mixed-Variable Optimization. In Marco Dorigo, Mauro Birattari, and Christian Blum, editors, *Proceedings of ANTS 2004 – Fourth International Workshop on Ant Colony Optimization and Swarm Intelligence*, volume 3172 of *LNCS*, pages 25–36. Springer-Verlag, Berlin, Germany, 5-8 September 2004.

[84] Patrick D. Surry and Nicholas J. Radcliffe. The COMOGA Method: Constrained Optimisation by Multiobjective Genetic Algorithms. *Control and Cybernetics*, 26(3):391–412, 1997.

[85] Aimo A. Törn and Antanas Zilinskas. *Global Optimization*, volume 350 of *Lecture Notes in Computer Science*. Springer, 1989.

[86] Jakob Vesterstrøm and Rene Thomsen. A comparative study of differential evolution, particle swarm optimization, and evolutionary algorithms on numerical benchmark problems. In *Proceedings of the 2004 IEEE Congress on Evolutionary Computation*, pages 1980–1987, Portland, Oregon, 20-23 June 2004. IEEE Press.

[87] Jakob S. Vesterstrøm and Jacques Riget. Particle swarms: Extensions for improved local, multi-modal, and dynamic search in numerical optimization. Master's thesis, Department of Computer Science, University of Aarhus, 2002.

[88] Christopher Walton, Virginia Biris-Briehante, Stephen Phelps, and David Robertson. Review of slie framework and experiments, 2003.

[89] Darrell Whitley, Soraya Rana, and Robert B. Heckendorn. Exploiting separability in search: The island model genetic algorithm. *Journal of Computing and Information Technology, v. 7, n. 1, p33-47 1999. (Special Issue on Evolutionary Computing)*, 1998. This paper physically titled: The Island Model Genetic Algorithm: On Separability, Population Size and Convergence.

[90] Darrell L. Whitley, Keith E. Mathias, Soraya Rana, and J. Dzubera. Evaluating evolutionary algorithms. *Artificial Intelligence*, 85:245–271, 1996.

[91] L. Darrell Whitley, Deon Garrett, and Jean-Paul Watson. Quad search and hybrid genetic algorithms. In *GECCO*, pages 1469–1480, 2003.

[92] L. Darrell Whitley, Monte Lunacek, and James N. Knight. Ruffled by ridges: How evolutionary algorithms can fail. In *GECCO (2)*, pages 294–306, 2004.

[93] David H. Wolpert and William G. Macready. No free lunch theorems for search. Technical Report SFI-TR-95-02-010, Santa Fe, NM, 1995.

[94] David H. Wolpert and William G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, April 1997.

[95] Xiao-Feng Xie and Wen-Jun Zhang. SWAF: Swarm algorithm framework for numerical optimization. In *GECCO (1)*, pages 238–250, 2004.

[96] Xin Yao and Yong Liu. Fast evolutionary programming. In *Evolutionary Programming*, pages 451–460, 1996. Revised 1999.

[97] Liu Yong, Lishan Kang, and David J. Evans. The annealing evolution algorithm as function optimizer. *Parallel Computing*, 21(3):389–400, 1995.

[98] Angel E. Muñoz Zavala, Arturo Hernández Aguirre, and Enrique R. Villa Diharce. Constrained optimization via particle evolutionary swarm optimization algorithm (PESO). In *GECCO*, pages 209–216, 2005.

[99] Ivan Zelinka and Jouni Lampinen. On stagnation of the differential evolution algorithm, July 28 2000.