Some modified stochastic global optimization algorithms with applications

Some Modified Stochastic Global Optimization Algorithms with Applications

by

M. Montaz Ali

A Doctoral Thesis
Submitted in Partial Fulfilment of the Requirements
For the Award of Doctor of Philosophy
of Loughborough University of Technology
September 1994.

Supervisor: Emeritus Professor C Storey, D.Sc., F.I.M.A.
Department of Mathematical Sciences.

To : My parents and my daughter.

Declaration

I declare that I am responsible for the work submitted in this thesis, that the original work is my own except as specified in acknowledgements, and that neither the thesis nor the original work contained therein has been submitted to this or any other institution for a higher degree.

M M Ali.

Acknowledgements

I would like to express my very sincere thanks to my supervisor Professor Colin Storey for his continual encouragement, guidance, comments and valuable supervision for the research in this thesis. I would like to acknowledge his enormous help in writing up this thesis. I am also obliged to him for introducing me to this important field of optimization.

It is with pleasure that I acknowledge the contributions of Professor Roger Smith to Chapters 5 and 6. In addition I am grateful for his valuable suggestions and the discussions we had regarding the importance and the demand for global optimization in many engineering fields, when I first arrived in Loughborough.

I would like to thank my Director of Research Dr. A. C. Pugh for his constant help during my candidature. I also sincerely acknowledge all the assistance I obtained from Dr. J. J. Forster, Professor G. A. Evans and Professor J. B. Griffiths.

I would like to thank my wife Panna for her patience and understanding. I am thankfull to my teachers Mohammed Shahidulla and Nurul Islam for their encouragements. I am also grateful to Dr. A. A. K. Mojumdar, who as my first teacher in operations research aroused and fostered my interest in research in general.

I would like to thank Louise and Helen for helping me on the use of TEX. Finally, I wish to thank the Commonwealth Scholarship Commission for sponsoring my study.

ABSTRACT

Stochastic methods for global optimization problems with continuous variables have been studied. Modifications of three different algorithms have been proposed. These are (1) Multilevel Single Linkage (MSL), (2) Simulated Annealing (SA) and (3) Controlled Random Search (CRS). We propose a new topographical Multilevel Single Linkage (TMSL) algorithm as an extension of MSL. TMSL performs much better than MSL, especially in terms of number of function evaluations. A new aspiration based simulated annealing algorithm (ASA) has been derived which enhances the performance of SA by incorporating an aspiration criterion. We have also proposed two new CRS algorithms, the CRS4 and CRS5 algorithms, which improve the CRS algorithm both in terms of cpu time and the number of function evaluations. The usefulness of the Halton and the Hammersley quasi-random sequences in global optimization has been investigated. These sequences are frequently used in numerical integration in the field of Bayesian statistics. A useful property of the quasi-random sequences is that they are evenly distributed and thus explore the search region more rapidly than pseudo-random numbers.

Comparison of the modified algorithms with their unmodified versions is carried out on standard test problems but in addition a substantial part of the thesis consists of numerical investigations of 5 different practical global optimization problems. These problems are as follows:

(1) A nonlinear continuous stirred tank reactor problem.

(2) A chemical reactor problem with a bifunctional catalyst.

(3) A pig-liver likelihood function.

(4) Application and derivation of semi-empirical many body interatomic potentials.

(5) A optimal control problem involving a car suspension system.

Critical comparisons of the modified and unmodified global optimization algorithms have been carried out on these problems. The methods applied to these problems are compared from the points of view of reliability in finding the global optimum, cpu time and number of function evaluations.

Table of Contents

CHAPTER 1
Introduction

1.1 Introduction

With the advancement of science and technology, problems often arise in contexts such as control theory, physical modelling, engineering design, data analysis, etc., in which an objective function has to be optimized subject to a set of constraints. Here, without loss of generality, we restrict ourselves to minimization. Let $f : \Omega \subset \mathbb{R}^n \rightarrow \mathbb{R}$ be a real-valued objective function. A (weak) local minimizer x^* of f is a point such that there exists a neighbourhood B of x^* with

$$f(x^*) \leq f(x), \quad \forall x \in B \; . \tag{1.1}$$

In general, however, for such problems, multiple minima may exist and they may also differ substantially. For problems with multiple minima one is interested in finding the very best minimum. We restrict our attention to this class of problems of a global nature. The global minimization problem for a function $f : \Omega \subset \mathbb{R}^n \rightarrow \mathbb{R}$ is to find x^* such that

$$f(x^*) \leq f(x), \quad \forall x \in \Omega \; . \tag{1.2}$$

We assume that f is a nonlinear function and for different algorithms it is assumed to have different smoothness properties. For instance, the interval arithmetic method (Hansen, 1979) and density clustering (Rinnooy Kan and Timmer, 1987) require f to be twice continuously differentiable whereas controlled random search (Price, 1983) does not require any derivatives at all. We also assume that the feasible region Ω is given by a set of lower and upper bounds on each variable, i.e.

$$\Omega = \{x \,|\, \underline{x}_i \leq x_i \leq \overline{x}_i, \; i = 1, \ldots, n\} \; .$$

In the general constrained global optimization problem the constraints usually consist of a set of equality or inequality constraints or a combination of both. Only a few methods have been developed to solve constrained problems. Levey and Gomez (1980) used a generalised tunneling method to deal with the constraints. Methods for constrained global optimization are also reported in Hoffman (1981) and Rosen (1981). Timmer (1984) adopted a penalty function approach.

However, our main concern in this thesis is to deal with problems which are 'essentially unconstrained', that is, the global minimum of f is attained in the interior of Ω. As

far as numerical calculation is concerned, the optimal solution is always regarded as an approximate solution. Thus a global optimization problem is considered to be solved if any one of the following sets is identified. For some $\epsilon > 0$,

$$A_x(\epsilon) = \{x \in \Omega \mid \| x - x^* \| \leq \epsilon\}, \tag{1.3}$$

$$A_f(\epsilon) = \{x \in \Omega \mid | f(x) - f(x^*) | \leq \epsilon\}, \tag{1.4}$$

$$A_\phi(\epsilon) = \{x \in \Omega \mid \phi(f(x)) \leq \epsilon\}, \tag{1.5}$$

where

$$\phi(f(x)) = \frac{m(\{z \in \Omega \mid f(z) \leq f(x)\})}{m(\Omega)} \tag{1.6}$$

and $m(.)$ is the Lebesgue measure.

Designing algorithms that can identify the best minimum is the subject of global optimization and forms the main objective of our research. Although, there is a large variety of problems which involve minimization with respect to continuous variables, there are also many problems of discrete optimization or even combinations of both. Discrete problems are widely known as combinatorial optimization problems because they involve arrangements of objects, for instance, chip placement in computer design, image processing, graph colouring, graph partitioning, travelling salesman problems etc. In the field of combinatorial optimization mathematicians use the performance of algorithms to distinguish between 'easy' and 'hard' problems. The travelling salesman problem is hard because no one has found an algorithm that computes the shortest tour of n cities in polynomial time. Despite the efforts of several generations of mathematicians and computer scientists, no one has found a complete solution. The problem of finding the global minimum for functions of continuous variables is also theoretically intractable. However, for the practical solution of problems, both discrete and continuous, approximate algorithms are often considered. Our research in this thesis involves only global optimization of continuous functions.

Local optimization problems can be solved with greater reliability than global ones and the nature of solutions can be characterized by the criteria of positive definiteness of the Hessian and zero gradient. Unfortunately, for the case of global optimization no such criteria exist in general. The aim of global optimization is to find the points in Ω for which the function attains its smallest value, the global minimum. The solution strategy often consists of a global stage and local stages. In the absence of a priori information all parts of the search region must be treated equally critically (the global stage). Of course, no significant parts of Ω must be neglected, unless one is willing to accept a considerable chance that the global minimum will be missed. When some information is accumulated some parts of the feasible region may be deemed more interesting than others and solutions in these parts are then required (the local stage). However, if the

function has been evaluated at a finite number of points then it is still possible that its global minimum may differ from the best minimum function value found so far by an arbitrary amount. In general therefore, it is almost impossible to find a computationally efficient algorithm which will always find the global minimum. In practical optimization applications, the evaluation of $f(x)$ is often very expensive computationally so that a large number of function evaluations would be the dominating expense. Therefore, there is always a trade-off between efficiency and reliability. So, in contrast to local optimization, most global optimization problems are practically impossible to solve. However, attempts have been made to construct algorithms that are more efficient than the most simple ones, such as pure random search or iterative use of local optimization from different starting points; but, so far, few algorithms for tackling global optimization have been developed, in comparison with the multitude of local optimization methods. Full details of existing global optimization algorithms can be found in Dixon and Szegö (1978), Ratschek and Rokne (1988), Törn and Žilinskas (1989), Horst and Tuy (1990) and Floudas and Pardalos (1992).

The known methods for global optimization can be divided into the two categories, deterministic and stochastic. Deterministic methods find global minima by an exhaustive search over the region of interest Ω. Therefore, most deterministic methods lose efficiency and reliability as the dimension of the problem increases. To guarantee success such methods unavoidably involve additional assumptions on f. For instance, many of them impose highly restrictive conditions on f such as satisfaction of a Lipschitz condition with a known constant. Good accounts of deterministic methods are given in Ratschek and Rokne (1988) and Horst and Tuy (1990). Deterministic methods do not involve any stochastic concepts. In the next section some deterministic methods are briefly discussed.

1.2 Deterministic Methods

Many approaches have been investigated for solving global optimization problems. For the general case where f can be any continuously differentiable nonlinear function, approaches that have been developed include deflation (Goldstein and Price, 1971) and piecewise approximation (Shubert, 1972) methods. Feasible space-covering methods with guaranteed convergence to the global minimum can be constructed if f satisfies some a priori conditions (Evtushenko, 1971). Examples of such conditions on f are bounds on the derivatives and the satisfaction of a Lipschitz condition.

Another kind of deterministic method is the trajectory method. This is a method of 'enumeration-of-local-minima' type which is based on the observation that each minimum, including the global one, is known to be a stationary point. Since f is assumed to have a finite number of stationary points, the global minimum could be found by locating

all stationary points, and comparing their function values. All trajectory methods are commonly known as generalized descent methods because they are based on modifications of the equation of local descent. In this context considerable attention has been paid to the trajectory methods due to Branin (1972) and Branin and Hoo (1972). Branin's idea is to introduce a dummy variable t and to consider points $x = x(t)$ that form a curve, parameterized by t, connecting a given stationary point to another stationary point. If it is possible to follow this curve then one could go from one stationary point to another and hopefully find them all by continuing the process.

How can such a curve $x(t)$ be created? This problem is solved by defining a differential equation, where differentiation is with respect to t, such that its solution is a function $x(t)$ with the desired property. To determine a differential equation which is suitable for this purpose, consider, for example,

$$\frac{dg(x(t))}{dt} + \mu g(x(t)) = 0 \ , \tag{1.7}$$

where $g(x(t))$ is the gradient of f at x and $\mu \in [+1, -1]$. The exact solution of (1.7) is

$$g(x(t)) = g(x(0))e^{\mu t} \ . \tag{1.8}$$

Clearly, if $\mu = -1$, the trajectory $x(t)$ satisfying (1.7) will tend to a stationary point with increasing t. If $\mu = +1$, then the trajectory will move away from that stationary point. Further details of the trajectory method can be found in Branin and Hoo (1972). Branin and Hoo noted that 'the trajectory method is not globally convergent in general, but may be in some instances. Moreover, not every trajectory passes through all solution points, although some may. Both of these limitations occur because of the existence of extraneous singularities of the basic differential equations'. Thus practical application of Branin's method raises many numerical and theoretical difficulties. In Yamashita (1979) a method based on Branin (1972) is also developed for solving a nonlinear programming problem with equality constraints.

A similar kind of trajectory method of 'enumeration-of-local-minima' type was proposed in Hassan (1982). This method locates a local minimum and computes the region of attraction of this minimum by a method described in White (1979). A point is then selected outside this region and the process continues until all minima are found. The system of equations involved in this method is given by

$$\dot{x} = -\nabla f(x) \ . \tag{1.10}$$

The solutions of (1.10) are the orthogonal trajectories of f. The equilibrium points of (1.10) are the solutions of $\nabla f = 0$ and hence are the stationary points of f. This method involves a special numerical integration of the partial differential equation of Zubov (Zubov, 1964)

for finding a Liapunov function and thereby the domain of attraction. Further details of the algorithm and some of its applications can be found in White (1979), Hassan and Storey (1981) and Hassan (1982). The major setback of this approach is that it appears to be only workable for the two variable case and more research is needed for possible extension to higher dimensions.

However, the trajectory methods apply only under specific imposed criteria such as, for example, f must be twice continuously differentiable. For these reasons trajectory methods do not qualify as general purpose global optimizers.

The tunneling method (Levy and Montalvo, 1985) is of 'improvement-of-local-minima' type. It is composed of a sequence of cycles, each cycle consisting of two phases: (a) a minimization phase having the purpose of lowering the current function value until a local minimizer is found; (b) a tunneling phase that has the purpose of finding a point $x \in \Omega$, other than the last minimizer, such that when x is employed as a starting point for the next minimization phase, the new stationary point will have a function value no greater than the previous minimum found. The major drawback of this algorithm is that it is difficult to be certain that the search for the global minimum has been sufficiently thorough. Therefore, the minimization problem of the tunneling phase virtually again becomes a problem of global nature.

The 'filled function' method for global optimization is due to Renpu (1990). This method is also of 'improvement-of-local-minima' type. As with many other deterministic methods, it is assumed here that f has only a finite number of minimizers in Ω and each of them is isolated. Whenever a minimizer x_1^* is known, a filled function can be constructed which determines a starting point for a local optimization of $f(x)$ which will produce a lower minimizer x_2^* than x_1^*, or which recognizes that x_1^* is already a global minimizer of $f(x)$. For a particular minimizer x^*, a typical filled function is given by

$$p(x) = \frac{1}{r + f(x)} \exp\Big(-\frac{\|x - x^*\|^2}{\rho^2}\Big) \tag{1.9}$$

where r and ρ are user provided parameters. After constructing $p(x)$ for a known minimizer x^* it is minimized by a local minimization method starting from the vicinity of x^* and then minimization of $f(x)$ starts from the point so obtained. This will hopefully produce a better minimizer for f. The procedure updates the present best minimizer of $f(x)$ and continues until no better minimizer can be achieved and ideally, the last local minimum in the sequence is the global minimizer. For an appropriate choice of parameters r and ρ, it can be shown that a method of local descent applied to $p(x)$ will arrive at a point x_{k+1} starting from which a descent procedure applied to f will arrive at better local minimum x_{k+1}^* (i.e., with $f(x_{k+1}^*) < f(x_k^*)$), if such an improved local minimum exists. Unfortunately, appropriate values for the parameters are based on information about f and this

is not readily available. Therefore, the parameters r and ρ are updated repeatedly in the procedure. This adjustment of the parameters makes the method very inefficient. Moreover, the adjustment does not guarantee that the appropriate values for the parameters will be obtained. Therefore, convergence to the global minimum can not be guaranteed.

The interval arithmetic method for global optimization was first introduced by Hansen (1979). Ichida and Fujii (1979) also derived a method of similar nature. The method is used to find the global minimizers of a twice continuously differentiable function f using interval arithmetic. Interval arithmetic (Moore, 1966) plays a key role in this kind of method. In this method it is also assumed that $f'(x)$ and $f''(x)$ have finitely many zeros in Ω. It is an iterative method where in each iteration each interval is subdivided into subintervals. Using tests such as monotonicity, convexity etc., subintervals where the global solution can not exist are discarded and the interval of largest length is chosen from the remaining list of intervals and the process continues. The stopping criterion is fulfilled when the combined length of remaining subintervals is sufficiently small. For recent work on the use of interval arithmetic algorithms in global optimization Ratschek and Rokne (1988) should be consulted.

1.3 Stochastic Methods

To overcome the inherent difficulties of deterministic algorithms, much research effort has been devoted to algorithms in which a stochastic element is introduced. Unlike deterministic methods, stochastic methods depend on probabilistic events and in most stochastic methods, two phases can be usefully distingushed, global and local. Stochastic techniques do not only play a role in the design and analysis of stochastic algorithms, but are also used to solve one of the basic problems in applying a stochastic method, which is when to stop. Most stochastic methods involve the evaluation of f in a random sample of points from Ω and subsequent manipulations of the sample. As a result, stochastic methods sacrifice the possibility of an absolute guarantee of success. However, the probability that an element of $A_x(\varepsilon)$, $A_f(\varepsilon)$ or $A_\phi(\varepsilon)$ is sampled can be shown to approach one as the sample size increases (Solis and Wets, 1981). Thus, such a global phase in which ultimately points are sampled in every subset of Ω with positive measure, gives rise to an asymptotic guarantee that is essential for the reliability of the method. Stochastic methods, thus, have a probabilistic convergence guarantee.

However, a method that contains only a global phase will be found lacking in efficiency. Although local improvement techniques cannot guarantee that the global minimum will be found, they are efficient tools that should be exploited to find points with relatively small function values. Thus a local phase is incorporated to improve the efficiency of the method. Therefore, stochastic methods involve random sampling or a combination of

random sampling and local search. They can be applied in much less restrictive situations than deterministic methods and have sound theoretical properties under which global minima can be found with a probabilistic guarantee of success. Therefore, stochastic methods have an intuitive appeal because of their inherent merits over the deterministic ones.

Some early stochastic methods were developed by Brooks (1958) and Bremermann (1970). These are simple random search algorithms. In general, stochastic methods are either 'two phase methods' or 'simulated annealing' type methods. Some well known two phase methods are:

Clustering with distribution function (De Biase and Frontini 1978).

Search Clustering (Törn 1978).

Controlled Random Search (Price 1983).

Pure Random Search (Rinnooy Kan and Timmer 1984, 1987).

Multistart (Rinnooy Kan and Timmer 1984, 1987).

Multilevel Single Linkage (Rinnooy Kan and Timmer 1984, 1987, 1987a).

Density, Single Linkage clustering (Rinnooy Kan and Timmer 1987).

Most two phase methods are iterative, and fit into the following framework:

- (Global phase): N points are drawn from a uniform distribution over Ω and the function is evaluated at these points. In this phase the search region Ω is explored.
- (Local phase): A subset of sample points is selected and a local search procedure (P) is applied to each element of this subset. This phase searches for a better solution than the previous best.
- A stopping rule decides whether to return to the global phase or to stop.

Later in this thesis many of the stochastic methods will be discussed in more detail but a brief general discussion of stochastic methods follows. Two phase methods such as pure random search (PRS) and multistart (MS) are very simple but inefficient. PRS is the simplest implementation of the Monte Carlo algorithm for global optimization. The main task of PRS is to find an improvement over the current function value by only random sampling. Its limited practical usefulness is mainly due to the fact that most of the information gathered during the execution of the algorithm is lost, as no use is made of function values and of function structure. MS is, in some sense, on the opposite side of PRS with respect to the use of local information. In this algorithm a local search routine is started from each sampled point. A stopping rule for MS is derived based on the number of points sampled and the number of different local minima found.

De Biase and Frontini (1978) describe a method based on random sampling of points in the region of interest Ω. Their first aim is to estimate a function $\psi(.)$, where

$$\psi(\xi) = \Pr\{\text{Random point } x \in \Omega \text{ has } f(x) \leq \xi, \quad \xi \in \mathbb{R}\}, \tag{1.11}$$

or, alternatively, $\psi(\xi)$ is the normalised Lebesgue measure of the subset $E(\xi)$ of Ω, where

$$E(\xi) = \{x \in \Omega, f(x) \leq \xi\}.$$

If $\psi(\xi)$ is known, then the minimum value of f in Ω may be obtained by setting $\psi(\xi) = 0$. De Biase and Frontini set out first to estimate $\psi(\xi)$ by a recursive spline technique, smoothing the data found by a sequential uniform random sampling both on Ω and on the expected range of function values. Sets of q random points and functions values at these points are generated iteratively and for each such set (say, i-th set of q points) ξ_i are chosen from a uniform distribution on $[f_l, f_u]$ where f_l and f_u are the lowest and highest function values. For every ξ_i the frequency $\psi(\xi)$ (i.e. $m(E(\xi))$) is evaluated by means of $\tilde{\psi}_i = \frac{p_i}{q}$, where p_i is the number of trial points in the i-th set of q points lying in the region $E(\xi_i)$. Therefore, a pair of values $(\xi_i, \tilde{\psi}_i)$ is obtained for each set. This is repeated and spline approximations are used to fit $\psi(\xi)$ to these results. This stage of the algorithm is terminated if a consistent fit is achieved and enough points are assumed to have been generated. The predicted minimum value of f^* can be obtained from these results. The second stage of the procedure is to group the points generated in the first stage into clusters and carry out a local search for a local minimum within each cluster.

Among the best performing methods for global optimization are those which mix local search procedures with the application of clustering techniques aimed at grouping together points in Ω belonging to the region of attraction of the same local minimum; the methods in this class try to identify the shape and location of the regions of attraction of local minima. A good review of clustering methods can be found in Törn and Žilinskas (1989). The most commonly used clustering technique in the context of global optimization consists of partitioning the available observations into groups, sequentially assigning sample points to clusters grown around 'seed points' which can be local minimizers or points with low function values.

The leaving out of unpromising points, which is known as 'reduction' or the 'concentration' of sample points are often used in clustering methods. In reduction, the sample is reduced by eliminating a fixed percentage, say $1 - \gamma$ $(0 < \gamma < 1)$, of points with higher function values; in concentration, a few steps of a descent algorithm are taken from each sample point. The clustering method due to Törn (Törn, 1978) uses the latter type of strategy. In Törn's method, initially a set of points (global points) are drawn from Ω and then the concentration strategy is applied to these points. The next step consists of

clustering the points obtained by this concentration. The clusters are formed sequentially, and each cluster is initiated by a seed point, s_o. This seed point is taken to be the point with the lowest function value from the unclustered sample points. Starting from s_o a cluster grows until the point density of the subregion it forms is greater than the average density of unclustered points in Ω. The process continues until all points have been considered. In the last phase of the algorithm a sample of points from each cluster is chosen and again concentration and clustering proceed. The algorithm stops when two successive clusterings result in the same number of clusters. The algorithm has no special features to escape from local minima and to concentrate only on the global one. This causes the algorithm to perform multiple local searches unnecessarily, especially when the function has many local minima. Moreover, there is no criterion which indicates that the search for the global minimum has been thorough enough and thus finding the global minimum can not be guaranteed.

Byrd, et. al. (1992) described a new stochastic global optimization algorithm that is oriented towards solving large scale problems. The algorithm incorporates some full-dimensional random sampling and local minimizations as in existing stochastic methods (Rinnooy Kan and Timmer, 1987a), but the keys to its success are two new phases that concentrate on selected small dimensional subproblems of the overall problem.

Recently, Törn and Viitanen (1992) developed a new topographical clustering method for global optimization. In this method a cluster is formed on the basis of topographical information on the function and only the cluster centre is determined rather than identifying the whole cluster. This type of clustering procedure does not use seed points but concentrates on identifying the cluster centre.

A different approach was taken by Schagen (1980) who introduced an algorithm in which a stochastic interpolating function is repeatedly optimized and reconstructed until an agreement is reached between the minimum of the interpolating function and the original objective function value at that point; then the minimizer of the interpolating function is taken as the global minimizer of the objective function. The stochastic interpolating function is a stationary stochastic model. However, finding the global minimum can not be guaranteed by optimizing such an interpolating function. Nonetheless a stochastic model may be appropriate if the original function is extremely expensive to evaluate.

The simulated annealing algorithm was proposed by Kirkpatrick, et. al. (1983). Methods based on simulated annealing use a stochastic mechanism which allows the algorithm to escape from a local minimum. Although the simulated annealing algorithm was initially designed for combinatorial optimization problems, several continuous versions are currently available (Vanderbilt and Louie, 1984, Aluffi-Pentini et al. 1985 and Bohachevsky et al. 1986). More recently, Dekkers and Aarts (1991) derived a continuous simulated annealing

algorithm which is theoretically similar to discrete simulated annealing. For an extensive annotated bibliography on both discrete and continuous simulated annealing see Collins et. al. (1988).

The variety of techniques that have been proposed is impressive, but their relative merits have neither been analysed in a systematic manner nor properly investigated by computational experiment. In this thesis we have studied and attempted to improve some recent stochastic global optimization algorithms and in view of the practical significance of the global optimization problem these algorithms have been critically assessed.

1.4 Outline of Thesis

Global optimization is creating considerable attention and more research is going on to try to deal with this intractable problem. Although a definite statement about the superiority of stochastic methods over deterministic ones is impossible to give, theoretical consideration as well as practical experience suggest that for problems of moderate to high dimension the use of stochastic techniques is perhaps the only feasible approach. We therefore consider those stochastic algorithms which perform well within this class of problem.

Three types of global optimization algorithms, namely (1) the Multilevel Single Linkage Method, (2) Simulated Annealing and (3) Controlled Random Search are discussed in detail in the subsequent Chapters. In Chapter 2 the connection between the clustering and single linkage methods is illustrated and a new topographical multilevel single linkage method is proposed. In Chapter 3 both discrete and continuous simulated annealing algorithms are reviewed. A brief theoretical background is also given. In the same Chapter a new aspiration based simulated annealing algorithm has been proposed together with a new adaptive polynomial-time cooling schedule. Chapter 4 deals with the controlled random search (CRS) algorithm (Price, 1983, 1987) and its modifications. We have proposed two new CRS algorithms and have demonstrated their superiority over the original algorithms.

Application of global optimization algorithms to real life problems and the critical comparison of these algorithms is an area where too little attention has been given. Therefore, our objective is to compare the effectiveness of some of the algorithms studied and described in previous sections in finding global minima for a number of real life problems. The problems we consider are from the fields of chemical engineering, material science, applied statistics and mathematical engineering. In Chapter 5 two problems from material science are investigated. A semi-empirical, short ranged many-body, interatomic potential for **bcc** metal is derived and global optimization algorithms are used to calculate the minimum energy of 'Tersoff' potentials (Tersoff, 1988, 1988a) for Silicon (Si) and 'Tersoff-like' potentials (Smith, 1992) for Arsenic (As). In Chapter 6 an

optimal feedback controller for a realistic car suspension system is proposed. In Chapter 7 global optimization algorithms together with an iterative dynamic programming method have been applied to two optimal control problems with a multiplicity of solutions. Also in Chapter 7, a global maximum has been sought for a pig-liver likelihood function. Comparisons of numerical results are given for each problem. The performance of the different stochastic methods has been assessed from a critical comparison of the numerical results obtained for each problem. Finally, conclusions and comments are given in Chapter 8.

Most of the numerical work has been carried out on the HP9000/870 computer except the numerical work for the problems in Chapters 5 and 6 and for the second problem in Chapter 7, for these problems we have used a faster HP9000/750 computer. A quadratic programming procedure E04UCF from the NAG Library has been used as the local search for the problems in Chapters 5, 6 and 7.

CHAPTER 2

Stochastic Methods: Clustering and Single Linkage

2.1 Multistart and the Bayesian Stopping Rule

The relative difficulty of global optimization in general is easy to understand and indeed, the global optimization problem as stated in (1.2) is inherently unsolvable because two crucial difficulties are encountered when an attempt is made to solve it. The first is that there exists no simple criterion according to which a point can be computed with a lower function value than the current best point. One feasible approach, however, is to seek such a point by applying a local search from a number of sample points. The second difficulty is that it will always remain uncertain whether or not the global minimum has been found. Hence, it is inevitable to make assumptions (e.g., Lipschitz continuity) about the objective function f or to widen the scope of global optimization algorithms.

A natural way out is a Bayesian approach in which the user is asked to specify a prior probability distribution on the unknown characteristics of f such as, for example, the exact number of local minima. Information gathered on f is then used to convert these probabilistic assumptions into a posterior distribution through Bayes Theorem (Boender, 1984). This posterior distribution reflects the way in which the initial beliefs are affected by the outcome of the experiments. The above mentioned difficulties in global optimization can now be posed as statistical decision problems, i.e. one can take a decision whether or not global optimality has been achieved with respect to posterior knowledge and a prespecified loss function (Boender and Rinnooy Kan 1983, 1985, 1987).

By far the most efficient methods for global optimization are based on starting a local optimization routine from points which are uniformly distributed over Ω (Timmer 1984 and Rinnooy Kan and Timmer, 1987). The widely known multistart (MS) algorithm is the prototype of these methods. In MS points are sampled iteratively from a uniform distribution over Ω, a local minimization is performed from each of these points and the local minimum with the smallest function value found in this way is a candidate value for f^*. A stepwise description of MS is given below.

The MS Algorithm

Step 1 Let $f^*=+\infty$

Step 2 If stopping condition is satisfied then stop; otherwise generate a uniform random x in Ω

Step 3 Perform a local optimization starting from x : let $\bar{x}$ be the local optimizer and $\bar{f}=f(\bar{x})$

Step 4 Let $f^*=\min(f^*,\bar{f})$

Step 5 Go to step 2

The MS method is reliable in the sense that the probability that the global minimum will be discovered increases as the sample size increases. Irrespective of whether a global optimization method is deterministic or stochastic, it always aims for an appropriate convergence guarantee. As with MS, all other stochastic methods aim for an asymptotic guarantee which will ensure convergence to the global minimum as the computational effort becomes infinite. The existence of such asymptotic guarantees raises the question of an appropriate stopping rule. In practice stopping criteria are used to stop the algorithm when there is sufficient evidence that the global optimum has been detected; or that the 'cost' connected with the search for a better estimate of the global minimum would be too high; or that some kind of 'resource' has been exhausted, such as, for example, computer time or number of function evaluations. Different algorithms however propose different stopping criteria and the stopping condition is part of the individual algorithm concerned. Of course, if the true number of local minima is unknown, methods based on local search can never provide an absolute guarantee in a finite time that the global optimum has been found: all that can be assured is that the probability of this event approaches 1 as the sample size tends to infinity. Thus, there exists a need for stopping rules to determine the sample size which corresponds to an optimal trade-off between reliability and computational effort.

Optimal Bayesian stopping rules for MS have been derived by Boender and Rinnooy Kan (1987). They have also described a rigorous Bayesian framework for the development of these optimal stopping rules. Their construction is based purely on a statistical analysis of the MS method. A crucial observation about MS is that its outcome, in the form of a sequential sample of local minima, can be viewed as a sample from a generalized multinomial distribution whose cells correspond to the local minima of f. Thus (Boender 1984; Boender and Rinnooy Kan 1987) it turns out to be possible to develop a Bayesian estimate of the probability that the next local search will locate a new local minimum. A decision whether or not to continue the search can be taken which is optimal with respect to a loss function which is based on a termination loss, if sampling is stopped before all local minima have been found and an execution loss, which expresses the cost of sampling and performing new local searches. Given the initial beliefs or prior distribution of unknown parameters, such a decision incorporates all information derived from the experiments, to weigh expected costs and benefits against each other in an optimal fashion. Therefore, the optimal Bayesian stopping rule is determined by specifying the costs and potential benefits

of further experiments and weighing these against each other probabilistically. Several loss structures and corresponding stopping rules are described in Boender (1984) and Boender and Rinnooy Kan (1987).

The region of attraction of a local minimizer x^* for a particular local search method is defined as the set of points x from which the local search will converge to x^*. If w different local minima have been found as the result of local searches started at each of N uniformly distributed points then Boender (1984) and Boender and Rinnooy Kan (1987) showed that a Bayesian estimate of the portion of Ω covered by the region of attraction of the local minimizers found so far is given by

$$E(C) = \frac{(N-w-1)(N+w)}{N(N-1)} \tag{2.1}$$

(the posterior expected value of the total volume of the observed regions of attraction) and a Bayesian estimate of the total number of local minimizers is given by

$$E(T) = \frac{w(N-1)}{N-w-2} \tag{2.2}$$

(the posterior expectation of the number of local minima). Here the total number of local searches N, must be greater than the number of distinct minima observed previously (i.e $N > w+2$). For most (N, w) pairs (2.2) will yield a non-integer estimate, although the true number of local minima is evidently an integer. However, it can be verified that the optimal integer Bayesian estimate under a quadratic loss function is a round-off of the non-integer estimate (Boender, 1984 and Boender and Rinnooy Kan, 1987). Therefore, after the k-th iteration the algorithm is terminated if the following criterion is satisfied;

$$E(T) \leq w + 0.5 \tag{2.3}$$

If the stopping criterion (2.3) is satisfied the estimated number of unobserved minima is equal to 0. This may cause an algorithm to run for a long period of time, especially when the objective function has many local minima with very small regions of attraction. Therefore another stopping criterion may be to terminate the algorithm if the total relative volume of the observed regions of attraction exceeds a prescribed value $\nu (0 < \nu < 1)$, i.e., stop if

$$E(C) \geq 0.995 \; . \tag{2.4}$$

However, condition (2.3) is widely recommended (Rinnooy Kan and Timmer, 1987a).

These stopping criteria, based on MS, are important as they can also be used by many other stochastic methods which use reduced samples, such as Clustering and Multilevel Single Linkage. Because the above Bayesian stopping rules for MS depend only on the

number of points sampled and the number of distinct minima found by performing local searches from these points, they are not only applicable to MS, but to every method which, given a sample, results in the same set of minima as MS. In particular, these stopping rules are applicable to the methods in which exactly one local search is started in every region of attraction in which points have been sampled (Rinnooy Kan and Timmer 1987). The only adjustment in the application of the stopping rules to the methods based on reduced samples is that the total number of points considered in a particular iteration, say the k-th iteration, is γkN instead of kN. In reduced sample methods a value between 0.1 and 0.2 is chosen typically for γ so that P is only applied to sample points with relatively small function values. More precisely, a prespecified fraction $1-\gamma$ of sample points, whose function values are relatively high is ignored. Thus, N in (2.1) and (2.2) is the total number of sample points but not the total number of local searches as in MS.

If $y_k^{(i)}$ is the i-th smallest function value in a sample of size kN obtained after k iterations, then all elements of the reduced sample are elements of

$$L\left(y_k^{(\gamma kN)}\right) = \left\{x \in \Omega \mid f(x) \leq y_k^{(\gamma kN)}\right\} \tag{2.5}$$

(Note that for ease of notation, integer round-up or round-down on γkN is ignored here.) However, it is not very efficient to apply P even to every reduced sample point, i.e. every point in $L\left(y_k^{(\gamma kN)}\right)$. Instead, methods in which P is started exactly once in every region of attraction which contains at least one reduced sample point are sought. But the probability that a region of attraction contains a reduced sample point depends on γ. To analyze this situation, for any γ with $0 < \gamma < 1$, let $y_\gamma \in \mathbb{R}$ be such that

$$\phi(y_\gamma) = \frac{m(\{x \in \Omega | f(x) \leq y_\gamma\})}{m(\Omega)} = \gamma \tag{2.6}$$

i.e., y_γ is the γ-quantile of f. Since ϕ is a monotonically increasing continuous function, there exists a unique value y_γ satisfying (2.6). If y_γ is known and the sample distribution is uniform then if P is applied to every sample point in $L(y_\gamma)$ the Bayesian analysis is still applicable. This is because the sample points whose function values exceed y_γ can simply be ignored and the Bayesian analysis obviously applies to the remaining points since they are still distributed according to the original uniform distribution over $L(y_\gamma)$.

However, since y_γ is not known in advance, P cannot be applied to the sample points in $L(y_\gamma)$. Instead one aims for methods in which P is applied to points in the level set $L\left(y_k^{(\gamma kN)}\right)$, such that all minima whose regions of attraction contain a reduced sample point are found. Since, the level above which the sample points are ignored depends on the sample, the cell probabilities of the multinomial distribution (Boender 1984) are no longer constant over time and the Bayesian analysis is no longer applicable. However this

effect can be ignored and the stopping rules applied to methods based on reduced samples (Rinnooy Kan and Timmer, 1987).

MS is very inefficient because the same local minimum might be found more than once. Given a sample of size kN that has been drawn from a uniform distribution over Ω and given a set of stationary points X^* (stationary points that are already known), a subset of the sample points must be determined to which P will be applied. To do so, one has to estimate the connected components (see next section) of the level set $L(y_k^{(\gamma kN)})$. A local search is then started once in each component that does not contain an element of X^*. The rationale of this approach is that if P is applied to an element of a component of $L(y_k^{(\gamma kN)})$, then P is known to converge to a local minimum in that connected component (Rinnooy Kan and Timmer 1987).

How can the components of $L(y_k^{(\gamma kN)})$ be identified? The natural way to identify these components is to make use of cluster analysis. In fact an adaptation of MS was provided first by clustering methods. Various superior clustering variants of MS were proposed by Rinnooy Kan and Timmer (1987, 1987a). The aim of these clustering algorithms is to apply local search more efficiently, that is, to apply local search only once in every region of attraction. Among them the Multilevel Single Linkage (MSL) method is considered to be the most successful adaptation (Timmer, 1984). MSL retains the theoretical properties of MS whilst attempting to eliminate its inefficiencies. Therefore, the number of local minima found by the variants of MS would be equal to the set of minima found by MS but at a much lower cost. In the following sections these clustering and single linkage variants of MS are discussed.

2.2 Clustering Methods

The idea used in clustering algorithms is to create groups of mutually close points that correspond to the relevant regions of attraction, and to apply P no more than once in each of these regions. The methods in this class try to identify the shape and location of the regions of attraction. As a result of the reduction of the sample, only points with relatively low function values are left. Intuitively speaking, these points will form clusters that correspond to the components of $L(y_k^{(\gamma kN)})$. A clustering technique is then introduced to attempt to identify each cluster. Clustering is a statistical method aiming at allocating individuals (sampled points) to one of several groups or clusters in such a way that each individual is more like individuals in its group than individuals outside its group. Grouping of sample points is done by means of some similarity measure (e.g, Euclidean distance) with respect to a threshold. Sample points are the data and the threshold is derived by applying statistical inference techniques in order to determine the accurate shape of clusters. Therefore, the most important issue in implementing any kind of clustering

technique is the choice of the threshold or critical distance, which is used to decide whether a point belongs to a certain cluster or not. The point around which a cluster grows is called its seed point. Local optimizers are often taken as seed points. In global optimization the clustering algorithms aim at forming clusters corresponding to each region of attraction and then P is started once in each cluster. To understand how far this objective can be achieved, we need the following additional definition. For $y \in \mathbb{R}$, let $L(y) = \{x \in \Omega, |f(x) < y\}$ (i.e the level above which the sample points are ignored). For any $x \in \Omega$ and $y \geq f(x)$, we define $L_x(y)$ to be the connected component (Dugundji 1966) of $L(y)$ containing x. If R_{x^*} is the region of attraction of the local minimizer x^* then $L_{x^*}(y)$ may contain a stationary point or stationary points other than x^*. Therefore, the groups created by sample reduction (for any particular y) correspond to the connected components of $L(y)$, and these do not necessarily correspond to the regions of attraction. We can clarify the possible impact of sample reduction by considering the one dimensional case in the following figure.

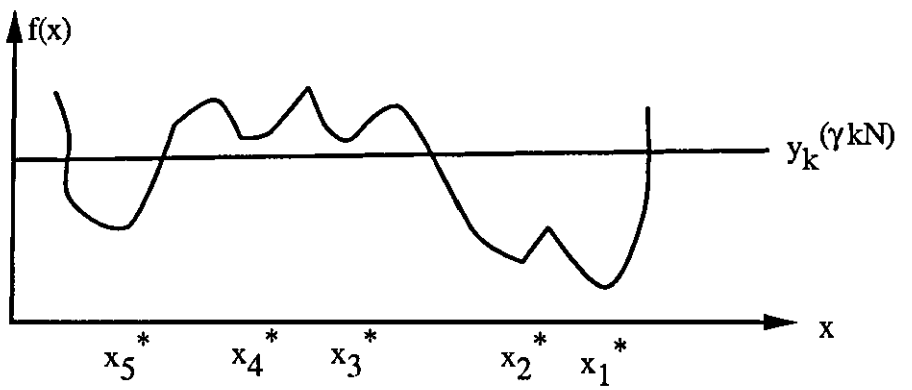


Figure 2.1

By the reduction of the sample we only consider those points whose corresponding function values are below the horizontal line indicated. Here $L(y_k^{(\gamma kN)})$ created only two connected components, namely the component containing x_5^* and that containing both x_1^* and x_2^*. These components do not correspond to the regions of attraction of x_1^* and x_5^*. Also notice that in the above figure local minima with a function value greater than $y_k^{(\gamma kN)}$, such as x_3^* and x_4^* will not be found, however this is not a serious drawback since we are interested in the global minimum. It is also possible that a component of $L(y_k^{(\gamma kN)})$ may contain several minima, e.g. x_1^* and x_2^*, in which case methods based on clustering will find only one of these. We will see later how the MSL method can overcome this difficulty.

After sample reduction, identification of the groups that correspond to the components of $L(y_k^{(\gamma kN)})$ is certainly a clustering problem, in which the objects are the reduced sample points and their characteristics are their locations and function values. Density and single linkage clustering identify the components of $L(y_k^{(\gamma kN)})$ and then P is started exactly once in each of these components. The purpose of an efficient clustering method is to invoke no more than one local search in each region of attraction. However, in global optimization,

there are several reasons for not using the ordinary clustering methods. The main reason is that there is more information about the problem than just the location and the function value of the reduced sample points. This extra information includes the fact that the reduced sample points are known to be a subset of a uniform sample and the fact that the groups searched for generally correspond to the components of a level set of a continuously differentiable function.

The following argument indicates that this extra information should not be ignored. A major difficulty in ordinary clustering problems is to determine the number of clusters. The desired number of clusters is a matter of subjective judgment and often has to be specified in advance. In the specific clustering problem involved in global optimization, the clusters should correspond to the components of $L(y_k^{(\gamma kN)})$. Hence, the correct number of clusters is one of the most important outputs of the method, and can certainly not be fixed a priori. Therefore, the extra information should be used in determining the number of clusters. This is justified in the sections below. In the next two sections we review the density clustering and the single linkage clustering methods. They differ only in the way points are added to the clusters and the termination criteria for the clusters. The description of the methods is given for a single iteration only. This is because any iteration does not use information from the previous iterations, apart from the set of local minima previously found.

2.2.1 Density Clustering (DC)

In this clustering method (Rinnooy Kan and Timmer, 1987) a modification is proposed in which ellipsoidal-shaped clusters are grown instead of the usual spherical ones; the idea being to try to approximate best the level sets of the objective function near local minima. In this approach a cluster is initiated by a seed point, which is a local minimizer. An iterative scheme then starts to form the cluster. In the scheme subsets T_i $i = 0, 1, 2, \ldots,$ of Ω of stepwise increasing volume are considered, where T_o is the seed point of a cluster and $T_{i+1} \supset T_i$. The formation of a cluster is 'terminated' if no points are added to it during a step. (Note that step here is in the clustering iteration but not the iteration of the algorithm.) In fact, in DC, Törn's clustering method (Törn, 1978) is adjusted in three ways; the choice of seed points, the shape of the sets T_i and the increase in size of these sets in each step. As mentioned above we will always describe only a single iteration (say the k-th) of the algorithm.

In the clustering methods which are being discussed here, it is clearly advantageous to choose a local minimum as the seed point. Therefore, the local minima in X^* are first used as seed points. Note that a local minimizer is obtained first followed by the clustering procedure that starts from that minimizer. If all local minima found so far have been used as seed points but there are still reduced sample points that have to be clustered, then a

local search is carried out from an unclustered sample point $\bar{x}$ with smallest function value. If the resulting local minimizer x^* found, is a member of X^* then $\bar{x}$ is added to the cluster initiated by x^* and again a local search is started from a reduced sample point with lowest function value. If the minimum found is not already known then a new cluster is started with this as the seed point.

Recall that, a cluster initiated with a local minimizer x^* should correspond to the component $L_{x^*}(y_k^{(\gamma kN)})$. This suggests letting T_i correspond to $L_{x^*}(y)$ for stepwise increasing values of y. The actual sets $L_{x^*}(y)$ are hard to construct but if f is twice continuously differentiable, they can be approximated by the level sets $\tilde{L}(y)$ around x^* that are defined by the second order approximation $\tilde{f}$ to f around x^*:

$$\tilde{f}(x) = f(x^*) + \frac{1}{2}(x - x^*)^T H(x^*)(x - x^*) \ , \tag{2.7}$$

where $H(x^*)$ is the Hessian of f at x^*. Hence at step i, let T_i be the set $\{x \in \Omega | (x - x^*)^T H(x^*)(x - x^*) \leq r_i^2\}$, for some r_i to be determined below, with $r_i < r_{i+1}$, $i = 1, 2, \ldots$

The distances r_i are derived by asymptotic considerations so that a cluster is terminated correctly. In other words the probability that the cluster terminates incorrectly should tend to 0 with increasing k. It remains to derive the rate at which r_i should increase with i so as to ensure proper termination for the growth of a cluster. Unlike Törn's clustering DC exploits the fact that the reduced sample is a subset of the original uniform distribution in finding the distances r_i. Moreover, a cluster initiated by a local minimizer x^* should not be terminated if there are still unclustered reduced sample points in $L_{x^*}(y_k^{(\gamma kN)})$. The probability that a cluster is terminated in step i, is equal to the probability that the set

$$A_i = \{x \in \Omega \mid x \in T_i, x \notin T_{i-1}\} \tag{2.8}$$

does not contain any reduced sample points. This termination would be incorrect if the component $L_{x^*}(y_k^{(\gamma kN)})$ still contained unclustered sample points. To determine the probability (probability of erroneous termination) that there are still unclustered sample points in $L_{x^*}(y_k^{(\gamma kN)})$, an assumption is made that the sets $L_{x^*}(y)$ with $f(x^*) \leq y \leq y_k^{(\gamma kN)}$ can be properly approximated by ellipsoids so that $T_i \subset L_{x^*}(y_k^{(\gamma kN)})$. Note that this is often the case when y is very close to $f(x^*)$. If α_k is the probability that a cluster is terminated incorrectly in step i, this implies that α_k is the probability that the none of the kN original sample points is located in A_i. A uniform sampling distribution gives

$$\alpha_k = \left(1 - \frac{m(A_i)}{m(\Omega)}\right)^{kN} \tag{2.9}$$

From (2.9), $m(A_i)$ and hence r_i can be found by fixing α_k. If the error probability α_k is taken to be α (typically a small number, say, 0.05) then $m(A_i) = m(\Omega)\left(1 - \alpha^{\frac{1}{kN}}\right)$ and in

step i, the volume of the ellipsoid is increased to $im(\Omega)\big(1-\alpha^{\frac{1}{kN}}\big)$. Since the volume of the ellipsoid $(x-x^*)^T H(x^*)(x-x^*) \leq r_i^2$ is equal to

$$\frac{\pi^{\frac{n}{2}} r_i^n}{\Gamma(1+\frac{n}{2})\big(detH(x^*)\big)^{\frac{1}{2}}}, \tag{2.10}$$

it follows that at the i-th step we have to check if there is at least one reduced sample point which has not yet been clustered and belongs to T_i with

$$r_i = \pi^{-1/2}\Big(i\Gamma\big(1+\frac{n}{2}\big)(detH(x^*))^{1/2} m(\Omega)(1-\alpha^{\frac{1}{kN}})\Big)^{1/n}. \tag{2.11}$$

If no points are found within this distance the clustering process terminates. Another way to construct r_i is by choosing $m(A_i)$ such that the probability α_k decreases with increasing k. Thus for some $\sigma > 0$, if the measure of A_i is chosen as $m(A_i) = m(\Omega)\frac{\sigma \log kN}{kN}$. Then

$$\alpha_k = \Big(1-\frac{\sigma \log kN}{kN}\Big)^{kN}. \tag{2.12}$$

The value of r_i with this α_k is given by

$$r_i = \pi^{-1/2}\Big(i\Gamma\big(1+\frac{n}{2}\big)(detH(x^*))^{1/2} m(\Omega)\frac{\sigma \log kN}{kN}\Big)^{1/n}. \tag{2.13}$$

This critical distance is recommended and frequently used. In this clustering method the cluster is not determined first and then followed by location of the corresponding local minimum but a local minimizer x^* is first located and then the reduced sample points in $L_x^*\big(y_k^{(\gamma kN)}\big)$ are identified. This change of order does not interfere with a wish to start P in each component of the level set. For iteration k with w different minima in hand, the DC algorithm can be defined as follows:

The DC Algorithm (k^{th} iteration)

Step 1 Determine the reduced sample by taking the γkN points with the smallest function values. Set $j := 1$.

Step 2 (Determine seed points). Set $i := 1$. If all reduced sample points have been assigned to a cluster, stop. If $j \leq w$, then choose the j-th local minimum in X^* as the next seed point. If $j > w$, then apply P to the unclustered reduced sample point $\bar{x}$ with the smallest function value. If $x^* \in X^*$ then assign $\bar{x}$ to the underlying cluster and repeat step 2. If x^* is new it is the next seed point and x^* is added to X^*.

Step 3 (Form cluster). Add all unclustered reduced sample points which are within distance r_i of the seed point x^*, to the cluster initiated by x^*. If no

point has been added to the cluster for a particular r_i, then $j := j + 1$ and go to step 2, else set $i = i + 1$ and repeat Step 3.

In this algorithm the shape of the cluster is assumed to be ellipsoidal, however, in practice, the set $L_x{}^*\left(y_k^{(\gamma kN)}\right)$ can differ substantially from an ellipsoid and may take any shape.

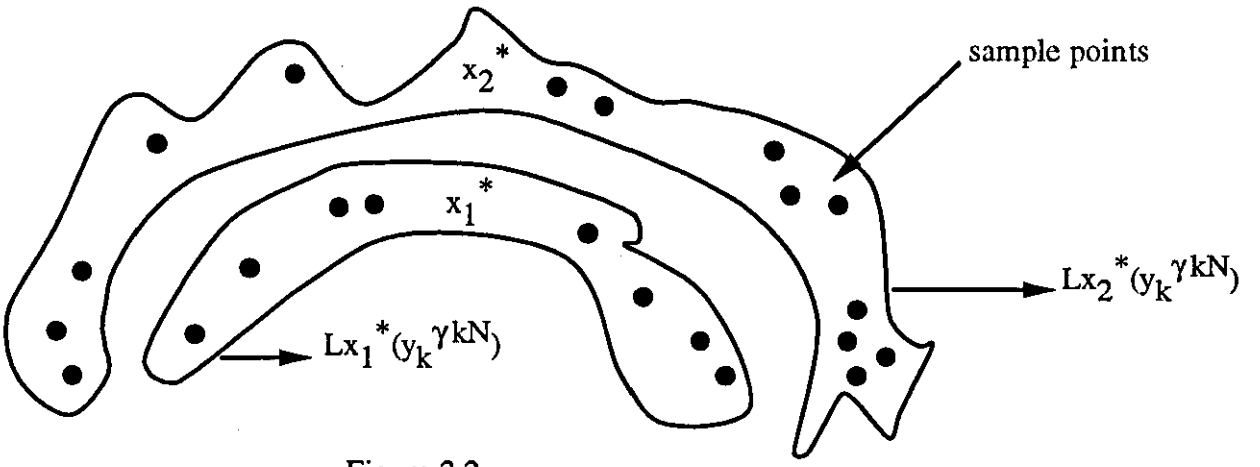


Figure 2.2

For a situation such as that shown in the two dimensional figure (2.2) it is clear that DC would not find two clusters that correspond to two different components of $L\left(y_k^{(\gamma kN)}\right)$. Therefore what is needed from a satisfactory clustering method is that the shapes of the resulting clusters are not fixed. In other words, the shapes of the clusters should converge to the shapes of the actual sets $L_x{}^*\left(y_k^{(\gamma kN)}\right)$. Single Linkage Clustering satisfies such a property.

2.2.2 Single Linkage Clustering (SLC)

In this method clusters are not forced to form any specific geometrical shape but can approximate the set $L_x{}^*\left(y_k^{(\gamma kN)}\right)$ directly. The original single linkage method can be viewed as a hierarchical procedure that starts with partition into single element subsets and in each subsequent step merges any two subsets E and E' whose minimal distance is less than a preset number. After every step of this procedure there exists a scalar $r > 0$ such that the partition has the following properties:

Every two elements belonging to different clusters are not within the critical distance r.

For every point x in a cluster there exists another point in the same cluster within distance r, provided that x is not the only element of the cluster.

However, the SLC method described here is a non-hierarchical procedure which, for some critical distance r_k, depending only on the number of sample points kN, produces clusters that satisfy the two above mentioned properties. In this SLC method, the clusters

are formed sequentially and each cluster is again initiated by a seed point. If C is a cluster and for any unclustered point x,

$$d(x, C) = \min_{x_1 \in C} \|x - x_1\| \ , \tag{2.14}$$

then x will be treated as member of C if $d(x, C)$ is less than the critical distance r_k. The procedure is repeated until $d(x, C)$ exceeds r_k. Obviously, the resulting clusters will indeed satisfy the above mentioned properties and also can have greatly varying geometrical shapes. The only condition for two points x_1 and x_2 to be in the same cluster, is that there is a sequence of points connecting them such that the distance between any two successive points in the sequence is less than r_k. This so-called 'chaining effect' is an obvious disadvantage of SLC in most applications. Because of it even if two points are arbitrarily far from each other they may be assigned to the same cluster. The situation can be visualized in the following two dimensional figure.

Figure 2.3

Obviously in the above figure there are two clusters but because of the chaining effect the SLC method may combine them both into a single cluster.

The critical distance r_k, is chosen to depend on kN only and to minimize the probabilities of two possible means of failure of the method:

The probability that a local search is started although the resulting minimum is known already.

The probability that no local search is started in a component of $L\left(y_k^{(\gamma kN)}\right)$ which contains reduced sample points (Rinnooy Kan and Timmer 1987).

For a suitable choice of r_k it can be shown that the probability that a local search is started incorrectly, tends to 0 with increasing k. However, the probability that a local

search is started at a reduced sample point a is certainly smaller than the probability that there is no sample point z in

$$B_{a,r_k} = \{x \in \Omega \mid \| x - a \| \leq r_k\} \tag{2.15}$$

with $f(z) < f(a)$, because a local search will not be started at a before z has been assigned to a cluster. Therefore, the probability that a local search is started incorrectly at a is bounded above by the probability that there is a sample point z in B_{a,r_k} with $f(z) < f(a)$. This probability can be calculated as follows. For an arbitrary sample point a the probability that none of the remaining $kN-1$ points is in $A_{a,r_k} = \{x \in \Omega | ||x-a|| < r_k$ and $f(x) < f(a)\}$ is given by

$$\Big(1 - \frac{m(A_{a,r_k})}{m(\Omega)}\Big)^{(kN-1)} . \tag{2.16}$$

But, if r_k tends to 0 with increasing k it can be proved that

$$\frac{m(A_{a,r_k})}{m(B_{a,r_k})} \geq \beta , \tag{2.17}$$

with $0 < \beta < \frac{1}{2}$ (Rinnooy Kan and Timmer 1987). Hence for any sample point a, the probability that there is no sample point z in B_{a,r_k} with $f(z) < f(a)$ is smaller than

$$\Big(1 - \frac{\beta m(B_{a,r_k})}{m(\Omega)}\Big)^{(kN-1)} \tag{2.18}$$

for sufficiently large k. Therefore, the probability that a local search is started incorrectly at a is bounded above by (2.18). It can also be shown that the above probability (2.18) decreases with increasing k if

$$r_k = \pi^{-1/2}\Big(\Gamma\big(1+\frac{n}{2}\big) m(\Omega) \frac{\sigma \log kN}{kN}\Big)^{1/n} \tag{2.19}$$

(Rinnooy Kan and Timmer, 1987). Now, if the critical distance r_k given by (2.19) tends to 0 with increasing k, it can also be proved that in every component of $L(y_k^{(\gamma kN)})$ in which a point has been sampled, a local minimum will be found by SLC within a finite number of iterations with probability 1 (Rinnooy Kan and Timmer, 1987). Hence, the possible failures of the method will vanish with increasing k.

The k-th iteration of SLC is as follows:

The SLC Algorithm

Step 1 Determine the reduced sample by taking the γkN sample points with smallest function values. Let w be the number of elements in X^*, set $j := 1$.

Step 2 (Determine seed points). If all reduced sample points have been assigned to a cluster; stop.

If $j \leq w$, then choose the j-th local minimum in X^* as the next seed point; go to step 3.

Otherwise, determine the point $\bar{x}$ from the remaining unclustered points which has the smallest function value. Apply P to $\bar{x}$ to find a minimizer x^*, adjust w and X^* accordingly, if x^* is new then take x^* as the next seed point; otherwise, assign $\bar{x}$ to the cluster initiated by x^* and repeat the process until a new local minimizer is found or no unclustered point is left.

Step 3 (Form cluster). Initiate a cluster using the seed point determined in step 2. Add reduced sample points which are within the distance r_k from the cluster to that cluster, until no more such points exist. Set $j := j + 1$, and go to step 2.

The ultimate goal of the clustering methods described so far is to start P exactly once in every region of attraction that contains a reduced sample point. The SLC method partitions the reduced sample points into clusters, such that each cluster corresponds to a component of $L(y_k^{(\gamma kN)})$. Although we know that a region of attraction cannot intersect two different components of a level set, it is possible that a component of $L(y_k^{(\gamma kN)})$ contains more than one region of attraction (see figure 2.1). Since only one local search is started in each cluster, it is therefore possible that a local minimum may not be found although its region of attraction contains a reduced sample point. In such a case both DC and SLC lack the important guarantee that MS can offer; if a point is located in a region of attraction then the local minimum of that region of attraction will be found. The function values of the sample points do not play any part in identifying the clusters in the DC and SLC methods. In the next section, we will see that these function values can be used explicitly in improving SLC.

2.3 Single Linkage Methods

2.3.1 Multilevel Single Linkage (MSL)

MSL is a modified version of the DC and SLC methods (Rinnooy Kan and Timmer 1987,1987a) which can overcome some of the drawbacks that some of the other clustering methods have. As with most stochastic methods it has two phases (1) a global phase and (2) a local phase. In the global phase, the function is evaluated at a number of random sample points. In the local phase, sample points are scrutinized to perform local searches in order to yield a candidate global minimum. As a stochastic method it offers a probabilistic guarantee that the global minimum will be found, assuming only that the function is continuously differentiable. The ultimate aim of this method is to start local searches exactly once in every region of attraction that contains a reduced sample point. One way to achieve this goal is to divide the reduced sample points into subsets, such that each subset coincides with the reduced sample points in a certain region of attraction. Former methods described in earlier sections subdivide the reduced sample points into clusters where each such cluster corresponds to a component of $L(y_k^{(\gamma kN)})$. They do not use function values to identify the clusters; function values are only used to calculate the reduced sample points. As a result the methods cannot distinguish between different regions of attraction which are located in the same component of $L(y_k^{(\gamma kN)})$. Function values can be of great importance for determination and separation of regions of attraction especially if one wishes to decide to which region of attraction a point x belongs. In the MSL method a r_k-descent sequence is considered. This is a sequence of sample points, such that any two successive points are within a distance r_k of each other and the function values in the sequence are monotonically decreasing. The clustering idea can be viewed as follows:

The MSL Clustering Algorithm

Step 1 Initiate w different clusters using the w different local minima at hand as seed points.

Step 2 Order the sample points, $f(x_i) \leq f(x_{i+1}), \quad 1 \leq i \leq kN-1$. Set $i := 1$.

Step 3 Assign the sample point x_i to any cluster which contains a sample point within a distance r_k. If x_i is not assigned to any cluster then P is applied to it to find another local minimum in order to start another cluster with this minimum.

Step 4 If $i := kN$ stop; else, set $i := i + 1$ and go to step 3.

The MSL method therefore makes use of function values in deciding whether a local search will start at a sample point or not. It is also clear that for any sample point x the decision whether P will be applied to it or not does not depend on the structure of the

cluster but on whether another point z, such that $f(x) < f(z)$, is within distance r_k of x. Hence the concept of clustering can be omitted altogether to give an algorithm of which the k-th iteration is as follows:

The MSL Algorithm

Step 1 Increase the existing set (initially empty) of sample points by N points randomly distributed over Ω. Compute function values at these points.

Step 2 Remove all sample points for which f is greater than some cut-off level (this reduction is optional). Order the sample points such that $f(x_i) \leq f(x_{i+1}), \quad 1 \leq i \leq kN - 1$. For every i, start local minimization from the sample point x_i if it has not been used as a starting point at the previous iteration or if there is another sample point, or a previously detected local minimum, within the critical distance r_k of x_i.

Step 3 Decide whether to stop. If the stopping condition is satisfied regard the lowest minimum found as the global minimizer, otherwise go to step 1.

A sample point x is only linked to a point with smaller function value that is within a distance r_k. The practical and theoretical success of this method is because of the selection of 'start' points for local minimization. A local search is only started at a point x_i if there is no sample point x_j, with $f(x_j) < f(x_i)$, within distance r_k of x_i. But the critical distance in (2.19) has been derived under the same circumstances as in SLC. Therefore, the same r_k is used here. With this critical distance the following strong theoretical properties of MSL can be established.

If the critical distance r_k of MSL is determined by (2.19) with $\sigma > 0$, and if x is an arbitrary sample point, then the probability that P is applied to x by MSL tends to 0 with increasing k. If $\sigma > 2$, then the probability that a local search is applied by MSL in iteration k tends to 0 with increasing k. If $\sigma > 4$, then, even if the sampling continues for ever, the total number of local searches ever started is finite with probability 1 (Timmer 1984).

To understand the superiority of MSL over SLC, especially because of the use of function values, we consider a level set of a one variable function in the following figure.

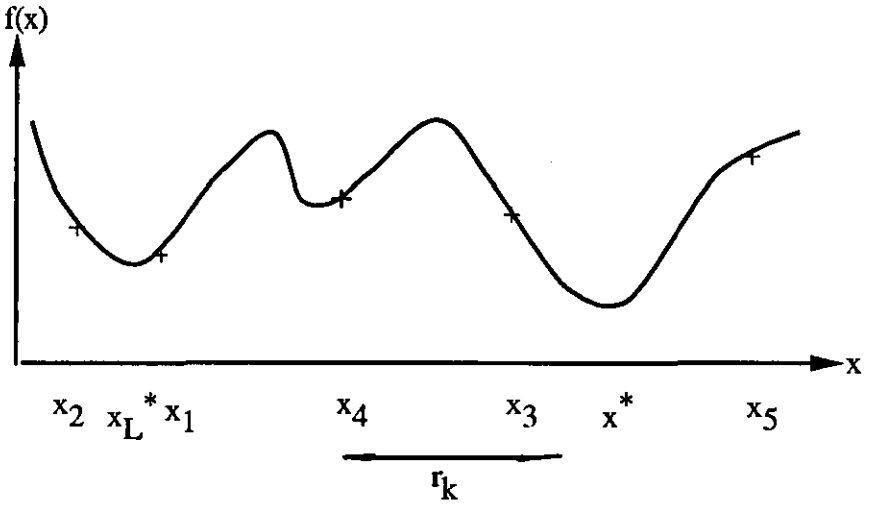


Figure 2.4

Suppose that $x_1 \ldots, x_5$ are reduced sample points and they are ordered according to ascending function values. Both SLC and MSL will start a local search from x_1 (the lowest function value) and when the clustering process begins MSL and SLC will assign x_1 to the cluster initiated by x_L^*, the local minimum found by starting local search at x_1. SLC assigns all other points $x_2, \ldots, x_5$ to the same cluster thus missing the global minimizer x^*. This occurs because of the so-called 'chaining' effect of SLC and this effect is because SLC does not use function values during the identification of a cluster. MSL however will assign x_2 to the cluster which is initiated by x_L^*, but when x_3 is considered it is not possible to link x_3 to x_L^*, x_3 to x_2 or to link x_3 to x_1, since $|x_3 - x_L^*| > r_k, |x_3 - x_2| > r_k$ and $|x_3 - x_1| > r_k$. Therefore, P will again be applied to x_3 and the global minimizer x^* will be located. Since any two local minima will always be separated by a region with higher function values, MSL will locate every local minimum in a neighbourhood of which a point has been sampled, if r_k is small enough.

Methods described so far in this Chapter use reduced sample points but a disadvantage of sample reduction is that there is a probability that a point is eliminated, which, currently, is the only sample point in the region of attraction of the global minimum. In the next section we derive a method which does not use a reduced sample as in MSL.

2.3.2 Topographical Multilevel Single Linkage (TMSL)

The aim of the clustering methods (DC and SLC) and MSL is to apply local search more efficiently, that is to apply local search only once in every region of attraction. But MSL is known to be superior to the clustering methods. However, the critical distance in MSL is derived using asymptotic considerations. As a result there is a likelihood of selecting false starting points within a region of attraction or of the use of the critical distance cancelling out true starting points for local search. Moreover, reduction of the sample may exclude

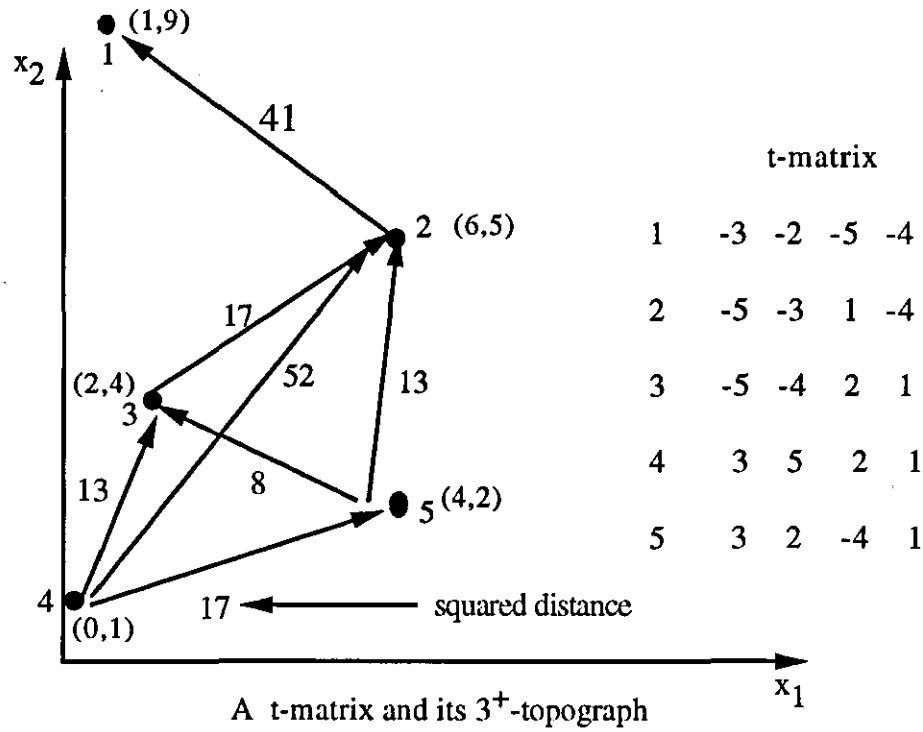
points in a potential region of attraction. The TMSL method is a new method, developed in an attempt to eliminate these drawbacks of MSL.

The Topographical Algorithm (TA) introduced by Törn and Viitanen (1992) uses topographical information on the objective function in identifying basins of local minima from the centre of each of which a local search is started. The TA algorithm is a non-iterative clustering method, based on exploration of the search space. The 'graph minima' are constructed by looking at the function values of some number g of nearest neighbour points for each point of a sample of size N. The aim of TA is to construct a topographical graph and then start minimization from just one point in each identified basin. These basins are identified by topographical information on the objective function using a directed graph. The graph connects neighbouring points to each other by directed arcs pointing towards points with higher function values. Törn and Viitanen (1992) gave a full description of how a graph minimum or centre of a basin can be constructed. For our purpose we restrict ourselves to the following brief review.

For each sample point from a sample of size N a reference list is constructed by ordering the points into nearest neighbour order. The list is further complemented by indicating if the reference is to a point with larger or smaller function value by giving the reference a plus or minus sign respectively. The $N-$reference lists constitute an $N \times (N-1)$ matrix, the t-matrix of the objective function. The $(N \times g)$ submatrix obtained by considering only the g nearest neighbours is called the g-t-matrix. The corresponding graph where arcs are drawn to the reference points with plus sign is called the g^+-topograph. The minima in the graph are all those nodes with no incoming arcs. In Figure 2.5 a t-matrix and the corresponding 3^+-topograph of the function

$$f(x_1, x_2) = x_1^2 + 2x_2^2 \tag{2.20}$$

is shown. There are $N = 5$ points numbered 1-5. Looking at the 3-t-matrix we see that point 4 has only positive references, i.e., its 3 nearest neighbours have larger function values, and that no positive reference to point 4 exists, i.e. no other point has point 4 as a point with larger function value among its 3 nearest neighbours. Choosing $g = 2$ there are two positive reference vectors, namely for points 4 and 5 but for point 4 there is a positive reference to point 5 in the 2-t-matrix, which means that the only graph minimum also in the 2^+-topograph is point 4. In the corresponding 1^+-topograph there are two graph minima, namely point 4 and point 5. Therefore, for higher values of g, the number of graph minima will be less. In our implementation we make no distinction between positive reference points and graph minima and call them all graph minima. Of course, the number of graph minima for a given f depends on the value of g and the size of N.



A t-matrix and its 3^+-topograph

Figure 2.5

TA requires the region to be explored as uniformly as possible. The number of points needed to be sampled in order to cover the whole region depends however, on the size of the region. Törn and Viitanen (1992) used a fixed size of sample points (N=100). To distribute the points evenly throughout the search region they adopted a technique of discarding points by prefixing a threshold. When a new point is generated from the uniform distribution a check is made in order to see whether this point is within the threshold distance of previously generated existing points. This expensive procedure, however, requires a large number of distances to be computed. For instance, for the Branin function (see appendix 2B) 1223 points were generated for singling out a sample size of 100. Törn and Viitanen (1992), however, gave a parallel version of TA to speed up the process.

The stopping condition for the sampling phase of the algorithm is based on the information that all parts of the search region have been explored. Once the sampling and construction of graph minima are completed, the algorithm terminates in an unrealistic manner leaving the users to start a number of local searches at their own discretion. For example, for the Branin function with the 3-nearest neighbour graph there are seven graph minima. Törn and Viitanen (1992) suggested starting local minimizations from the three best minima of this graph. But to make sure the search for global minima is thorough, one has to start seven local minimizations. For a smooth function it is likely that the graph will represent all local minima if the value of g has been chosen properly and sample points are distributed evenly enough. But a badly scaled function with peculiar regions of attraction would weaken this possibility. For a large value of g TA could loose a potential graph

minimum. However, this can be compensated by increasing the value of N. Having a large N on the other hand could give rise to too many graph minima in a region of attraction. Therefore, for a general purpose algorithm a fixed value of g would be too restrictive to represent the appropriate number of local minima for any function. Moreover, it would be increasingly difficult to construct the exact graph minima of functions, especially when the dimensions of the functions increases. As an appropriate value of g is difficult to predefine for an arbitrary function, TA can not guarantee that the global minimum will be found. In principle, therefore, both MSL and TA cause errors of the following nature.

- Type I Error, Local search will be repeated in some region of attraction.
- Type II Error, Local search will not start in some region of attraction even if a sample point has been located in that region of attraction.

Rinnooy Kan and Timmer (1987) argued that in MSL the above two types of error would not occur after a sufficiently large number of iterations. But clearly continuing the search for too large a number of iterations is wasteful. Furthermore, in MSL extended samples are considered and the resulting overheads could also rise to a prohibitive level. The question therefore arises of whether these errors can be avoided in every iteration if we use topographical information on the underlying function in a sensible way. We, therefore, propose a new algorithm, TMSL, that uses MSL together with topographical information on the objective function. This adaptation of MSL will guarantee that a local search will start at a point with a relatively low function value, thus making sample reduction no longer necessary. We, therefore, modify the MSL method in a way in which the strategy of reducing sample points is completely dropped. In contrast, a representative subset of sample points consisting of the graph minima is extracted first and then the critical distance criterion is applied to this subset. The TMSL method uses topographical information on the objective function, in particular the g-nearest-neighbour graph. The algorithm also uses evenly distributed points from a Halton sequence of uniform limiting density. We discuss the implementation of the algorithm and compare its performance with other well-known algorithms. The new algorithm performs much better (in some cases several times) than the MSL method in terms of number of function evaluations but is not quite so competitive with respect to cpu time.

Quasi-random Sequences

One of the major difficulties in global optimization problems is to identify the region of attraction of a global minimum. As the location of this region of attraction is unknown, it is an essential part of any global optimization algorithm to explore the search region thoroughly. Therefore, the sampling phase (global phase) of any algorithm aims at exploring the search region. In general, pseudo-random numbers are used for this purpose.

Pseudo-random numbers are independent realizations of a random variable. In contrast, quasi-random sequences (Hammersley and Handscomb, 1964) do not approximate a set of independent realizations from a uniform distribution but tend, in general, to be much more evenly distributed than the pseudo-random sequences. In the context of numerical integration in Bayesian statistics, where quasi-random sequences are known to give efficient numerical integration rules, Hammersley suggested using the k-dimensional quasi-random sequence

$$x_i = \left(\frac{i}{N+1}, \phi_{p_1}(i), \ldots, \phi_{p_{n-1}}(i) \right) \qquad (i = 1, \ldots, N). \tag{2.21}$$

The p_j are pairwise coprime (usually they are chosen to be the first $n-1$ primes), and $\phi_p(i)$ is the radical inverse function of i, obtained by writing i to base p and reflecting about the 'decimal' point. In Chapter 4 we will give further discussions on the Hammersley sequence where it is used in conjunction with the CRS algorithm. The first co-ordinate in this sequence depends on N. Thus Hammersley's sequence fails to qualify for an iterative algorithm. In the context of iterative global optimization algorithm, therefore, we argue that the Halton sequence (Shaw, 1988 and Halton, 1960) will explore the search region more evenly than pseudo-random sequences. The Halton sequence is given by

$$x_i = (\phi_{p_1}(i), \ldots, \phi_{p_n}(i)) \ , \qquad (i = 1, \ldots, N). \tag{2.22}$$

Figure (2.6) shows a comparison of the first 128 points of a two-dimensional Halton sequence† (with $p_1 = 2$ and $p_2 = 3$) with 128 pseudo-random points. The motivation for the Halton sequence also includes the fact that the sequence itself possesses the property of uniform limiting density, provided p_j are mutually prime. Furthermore, it is defined for arbitrary N, and is the initial segment of every Halton sequence with the same p_j $(j = 1, \ldots, n)$ but more than N points.

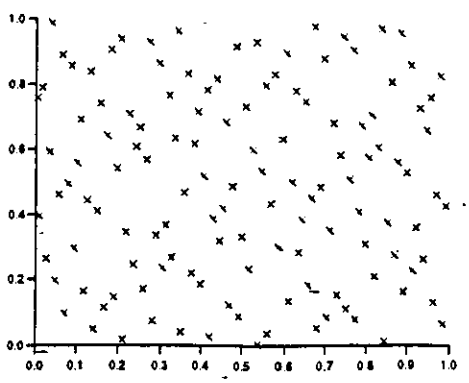


Figure 2.6(a) 128-point Halton sequence.

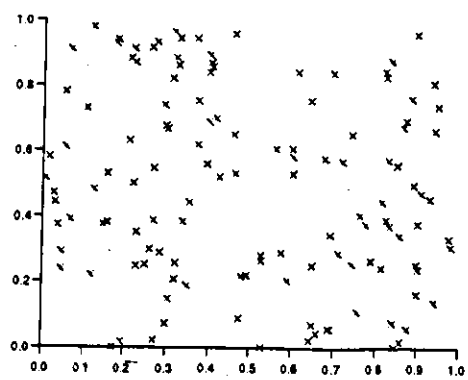


Figure 2.6(b) 128 pseudo-random points.

† A PASCAL subroutine for generating a two-dimensional Halton sequence is given in appendix 2A.

Therefore, it is clear that the Halton sequence will explore the search region more rapidly than the pseudo-random numbers. Moreover, pseudo-random numbers can be generated in many different ways and they may vary considerably so the performance of any given algorithm depends on the peculiarities of the random sample points at hand. The quasi-random sequences, however, do not have this drawback.

The New Algorithm

In MSL the decision to start a local search in the k-th iteration depends only on the threshold

$$r_k = \pi^{-1/2}\Big(\Gamma\big(1+\frac{n}{2}\big)m(\Omega)\sigma\frac{\log kN}{kN}\Big)^{1/n}.$$

This threshold is derived using asymptotic considerations. A point is taken as the starting point for local optimization if there is no other sample point, within the critical distance r_k, with lower function value. This check is carried out for all reduced sample points. We argue that error type I can be reduced by using graph minima with a suitable value for g instead of a reduced sample. Moreover, error II will tend to decrease as the number of iterations increases if the sample points are dependent on each other. Extended samples are not considered in TMSL because the sample points are dependent on each other and thus explore the search region more rapidly. The purpose of generating a comparatively large number of points and the sample reduction strategy in MSL is to make sure that

A. The search region has been explored thoroughly so that points are drawn in every region of attraction

B. A fraction of the points is discarded so that only points with relatively low function values are left for scrutiny.

We, however, propose using the selection of graph minima instead of sample reduction, together with evenly distributed sample points, to achieve conditions **A** and **B** above. Having found the graph minima, a local search is then carried out from a subset of them. No attempt is made to find the complete topograph of the function so the value of g is not so critical as it is in TA. Our experience shows however that a small value for g is normally to be recommended. We believe that this strategy is efficient because the number of graph minima depends on the choice of g and the particular function at hand as opposed to an empirical fraction γ as in MSL.

The basic principle of TA is to cover the search region with sample points as uniformly as possible. In our algorithm we therefore use the Halton sequence which is more evenly distributed than the pseudo-random numbers and has uniform limiting density. At the start of a new iteration we add all local minimizers found previously to the new set of sample points and then the construction of the graph minima takes place. In every successive iteration, therefore, local minima from previous iterations could become graph minima. In

principle, we use previous sample points in an implicit way by representing them by the local minima they produced.

Because the Halton sequence is uniform asymptotically, many of the theoretical results for MSL apply also to TMSL. We now discuss the theoretical justification of the method. In the MSL method the critical distance has been derived from asymptotic considerations. Consequently, it is justifiable for our case as the Halton sequence is uniform asymptotically.

For a particular graph minimum a, the probability that P is applied to a depends on whether there is a graph minimum z_g in B_{a,r_k} with $f(z_g) < f(a)$. Derivation of the critical distance is analogous to that given in the section 2.2.2 and so will not be given here. Briefly, however, if we adopt the idea of considering the extended sample points implicitly, then for an arbitrary sample point or a graph minimum a the probability that none of the remaining kN-1 points is in A_{a,r_k} is given by

$$\left(1 - \frac{m(A_{a,r_k})}{m(\Omega)}\right)^{(kN-1)} . \tag{2.23}$$

We now prove that the probability that P is applied to an arbitrary graph minimum a tends to zero with increasing k. We have

$$\begin{aligned} &\Pr\{P \text{ is applied to } a\} \leq \Pr\{\nexists \text{ a sample point } z \in A_{a,r_k}\} \\ &\leq \Pr\{\nexists \text{ a graph minimum } z_g \in A_{a,r_k}\} \end{aligned} \tag{2.24}$$

$$\text{Hence } \Pr\{P \text{ is applied to } a\} \leq \left(1 - \frac{m(A_{a,r_k})}{m(\Omega)}\right)^{(kN-1)}$$

$$\leq \Pr\{\nexists \text{ a graph minimum } z_g \in A_{a,r_k}\} .$$

Rinnooy Kan and Timmer (1987) have derived the critical distance from the consideration of the vanishing of the probability (2.23). It, follows from Rinnooy Kan and Timmer (1987) that if $m(A_{a,r_k}) \geq (\beta\sigma \log kN)/kN$ (where σ, β and N are constants) then the probability (2.23) is $O(k^{1-\beta\sigma})$. Of course, the probability that P is applied to a is bounded above by the probability defined by (2.23). Therefore, we can argue that if $\sigma > 2$ and $\frac{1}{\sigma} < \beta < \frac{1}{2}$ the probability that P is applied to a approaches zero with increasing k. Moreover, if n_k is the number of local searches started, then for $\sigma > 4$, it follows that

$$\sum_{k=1}^{\infty} \Pr\left[n_k > 0\right] < \infty \tag{2.25}$$

It follows immediately from the Borel-Cantelli lemma that even if the sampling continues forever the total number of local searches is finite with probability 1. We now give a stepwise description of the algorithm for a typical iteration k with w different minima found previously.

The TMSL Algorithm

Step 1. Generate N sample points from the Halton sequence over the search region Ω. Compute f at each point. Let M be the set of sample points plus the w minimizers found previously.

Step 2. Construct a topographical graph and find graph minima for these M points.

Step 3. A graph minimum is a 'start' point for local search if it is greater than the critical distance r_k from any point with smaller f value and if it is not a previously obtained local minimizer.

Step 4. Carry out a local search from each such point. If new local minima are found then update w accordingly.

Step 5. Is the stopping condition (Boender and Rinnooy Kan, 1987) $\frac{w(kN-1)}{kN-w-2} \leq w + \frac{1}{2}$ satisfied ? Yes, stop. No, go to Step 1.

Numerical Results

In this section we compare our new algorithm numerically with other recent algorithms using the test functions taken from Dixon and Szegö (1978), a set of commonly used functions in global optimization. The test functions are given in appendix 2B. We use the limited memory BFGS routine, from the NAG Library (version E04DGF) for local searches with tolerance $\frac{\|g(x)\|}{(1+|f(x)|)} < 10^{-10}$, where g is the gradient of f.

Table 2.1

Symbol	Test Function	Number of local minima (m)	Dimension (n)
BR	Branin	3 (all same f^*)	2
GP	Goldstein and Price	4 (all different f^*)	2
S5	Shekel5	5 "	4
S7	Shekel7	7 "	4
S10	Shekel10	10 "	4
H3	Hartman3	4 "	3
H6	Hartman6	4 "	6

Choice of Parameters in TMSL

The main user supplied parameters for TMSL are N, the sample size, σ in r_k and g the number of nearest neighbour points in the topographs. We carried out an extensive series of tests to see the effects of varying these parameter in the algorithm. We considered; $N = 10n, 10(n+1), 15n$ and $15(n+1)$ (where these were distinct), $\sigma = 2, 4$, $g = 2, 3, \ldots, N-1$.

These parameter values were used for each of the 7 test functions described in Table 2.1. The full results are given in Table 2.2. In this Table the largest values of g for which the global minima were obtained are also given. We use the following notation : LS is the number of local searches performed, LM is the number of local minima found, FE is the total number of function evaluations, N is the sample size and cpu is the cpu time.

After the construction of the set of graph minima, the critical distance, r_k, decides whether or not the local search procedure starts from a graph minimum. Since σ is a factor in the critical distance its value, therefore, plays a part in deciding whether or not a local search should start. A higher value of σ may prevent a local search starting at a particular graph minimum on the other hand a smaller value will attempt to invoke more local searches and in the limit $\sigma \to 0$ a local search will start at every graph minima. The effect of σ is particularly noticeable in Table 2.2. It is clear that $\sigma = 4$ is much better than $\sigma = 2$ for smaller values of g but this effect falls off as g increases. The global optimum was reached for almost all values of g for all functions, except S7. The algorithm also failed to obtained the global minimum for S5 when $N = 15(n+1)$ and $g \geq 7$. However, as g increased towards $N-1$ the number of function evaluations decreased to a smallest value at which $LS = LM = 1$. For some values of N, as g approaches $N-1$ the equality $LS = LM = 2$ held and remained static subsequently. Notice that this is not true for GP. However, as FE decreased the cpu time increased. Table 2.2 also shows that $N = 10n$ is the best sample size for all test functions. We have also run the algorithm with values of N less than $10n$, but for many values of g the algorithm failed to obtain the global minima.

Table 2.2

FUNCTION	N	σ	cpu	LS	LM	FE	g
BR	$10n$	2	0.14	4	3	94	2
		4	0.06	2	2	46	
		2	0.15	4	3	94	3
		4	0.06	2	2	46	
		2	0.13	4	3	94	4
		4	0.06	2	2	46	
		2	0.11	2	2	46	5
		4	0.08	2	2	46	
		2	0.14	2	2	46	6
		4	0.08	2	2	46	
		2	0.09	2	2	46	7
		4	0.09	2	2	46	
		2	0.10	2	2	46	8
		4	0.11	2	2	46	
		〃	〃	〃	〃	〃	〃
		〃	〃	〃	〃	〃	16
		2	0.12	1	1	34	17
		4	0.13	1	1	34	
		〃	〃	〃	〃	〃	〃
		〃	〃	〃	〃	〃	$N-1$
	$10(n+1)$	2	0.16	5	3	106	2
		4	0.12	3	3	68	
		2	0.14	5	3	101	3
		4	0.15	3	3	68	
		2	0.15	4	3	81	4
		4	0.17	3	3	68	
		2	0.18	4	3	81	5
		4	0.17	3	3	68	
		2	0.18	4	3	81	6
		4	0.17	3	3	68	
		2	0.19	3	2	69	7
		4	0.18	2	2	56	
		2	0.24	3	2	69	8
		4	0.24	2	2	56	
		2	0.24	2	2	56	9
		4	0.24	2	2	56	
		〃	〃	〃	〃	〃	〃
		〃	〃	〃	〃	〃	24
		2	0.41	1	1	44	25
		4	0.41	1	1	44	
		〃	〃	〃	〃	〃	〃
		〃	〃	〃	〃	〃	$N-1$

	$15(n+1)$	2	0.50	7	4	213	2
		4	0.30	3	3	82	
		2	0.39	3	3	82	3
		4	0.37	3	3	82	
		2	0.52	3	3	82	5
		4	0.49	3	3	82	
		2	0.62	3	3	82	7
		4	0.62	3	3	82	
		2	0.84	3	3	82	11
		4	0.82	3	3	82	
		2	0.88	2	2	70	12
		4	0.89	2	2	70	
		〃	〃	〃	〃	〃	〃
		〃	〃	〃	〃	〃	36
		2	1.76	1	1	59	37
		4	1.76	1	1	59	
		〃	〃	〃	〃	〃	〃
		〃	〃	〃	〃	〃	$N-1$
GP	$10n$	2	0.11	3	3	137	2
		4	0.06	1	1	53	
		2	0.11	2	2	90	3
		4	0.06	1	1	53	
		2	0.06	1	1	53	4
		4	0.06	1	1	53	
		2	0.09	1	1	53	8
		4	0.08	1	1	53	
		〃	〃	〃	〃	〃	〃
		〃	〃	〃	〃	〃	$N-1$
	$10(n+1)$	2	0.14	3	1	134	2
		4	0.12	2	1	94	
		2	0.18	2	2	103	3
		4	0.13	1	1	63	
		2	0.16	2	1	103	4
		4	0.17	1	1	63	
		2	0.20	2	1	103	6
		4	0.17	1	1	63	
		2	0.20	1	1	63	8
		4	0.21	1	1	63	
		〃	〃	〃	〃	〃	〃
		〃	〃	〃	〃	〃	$N-1$
	$15(n+1)$	2	0.35	4	3	161	2
		4	0.23	2	3	113	
		2	0.38	4	3	161	3

		4	0.37	3	3	143	
		2	0.42	3	2	131	4
		4	0.44	2	2	113	
		2	0.56	1	1	58	6
		4	0.55	1	1	58	
		〃	〃	〃	〃	〃	〃
		〃	〃	〃	〃	〃	$N-1$
S5	$10n$	2	0.37	6	3	238	2
		4	0.30	3	2	143	
		2	0.33	6	3	230	3
		4	0.29	3	2	136	
		2	0.37	4	3	175	4
		4	0.33	2	1	112	
		2	0.38	3	2	145	5
		4	0.36	2	1	116	
		2	0.41	3	2	145	6
		4	0.41	2	1	116	
		2	0.46	3	2	145	7
		4	0.45	2	1	116	
		2	0.48	1	1	77	8
		4	0.48	1	1	77	
		〃	〃	〃	〃	〃	〃
		〃	〃	〃	〃	〃	$N-1$
	$10(n+1)$	2	0.64	3	3	142	4
		4	0.62	2	2	113	
		2	0.66	2	2	118	5
		4	0.70	1	1	89	
		2	0.74	2	2	118	6
		4	0.74	1	1	89	
		2	0.84	2	2	118	7
		4	0.82	1	1	89	
		2	0.90	1	1	89	8
		4	0.90	1	1	89	
		〃	〃	〃	〃	〃	〃
		〃	〃	〃	〃	〃	$N-1$
	$15n$	2	0.96	6	3	253	2
		4	0.77	2	2	127	
		2	0.93	6	4	256	3
		4	0.79	2	2	123	
		2	1.01	3	3	152	4
		4	0.97	2	2	123	
		2	1.09	2	2	129	5
		4	1.08	1	1	100	
		〃	〃	〃	〃	〃	〃
		〃	〃	〃	〃	〃	10

		2	1.96	1	1	100	11
		4	1.98	1	1	100	
		〃	〃	〃	〃	〃	〃
		〃	〃	〃	〃	〃	$N-1$
	$15(n+1)$	2	2.20	5	2	244	4
		4	2.12	2	2	133	
		2	2.18	4	2	211	5
		4	2.12	2	2	133	
		2	2.38	4	2	205	6
		4	2.35	2	2	131	
		2	2.70	2	1	127*	7
		4	2.66	1	1	98*	
S7	$10n$	2	0.40	4	3	163	2
		4	0.36	2	2	98	
		2	0.40	4	3	163	3
		4	0.37	3	3	127	
		2	0.40	3	3	127	4
		4	0.35	3	3	127	
		2	0.42	2	2	98	5
		4	0.41	2	2	98	
		〃	〃	〃	〃	〃	〃
		2	0.48	2	2	98	8
		4	0.49	2	2	98	
		〃	〃	〃	〃	〃	〃
		〃	〃	〃	〃	〃	38
		2	1.20	1	1	60*	$N-1$
		4	1.21	1	1	60*	
	$10(n+1)$	2	0.60	2	2	108	4
		4	0.58	2	2	108	
		2	0.68	2	2	108	5
		4	0.69	2	2	108	
		2	0.80	2	2	108	6
		4	0.75	2	2	108	
		〃	〃	〃	〃	〃	〃
		〃	〃	〃	〃	〃	47
		2	2.80	1	1	70*	48
		4	3.25	1	1	70*	
	$15n$	2	0.85	5	4	222	2
		4	0.81	3	3	157	
		2	0.85	5	4	222	3
		4	0.85	4	4	186	
		2	1.04	2	2	118	4
		4	0.97	2	2	118	
		2	1.20	3	3	146	5

		4	1.12	2	2	118	
		2	1.28	3	3	146	6
		4	1.27	2	2	118	
		2	1.38	2	2	118	7
		4	1.27	2	2	118	
		〃	〃	〃	〃	〃	〃
		〃	〃	〃	〃	〃	57
		2	5.87	1	1	80*	58
		4	5.90	1	1	80*	
	$15(n+1)$	2	2.24	4	4	195	4
		4	1.54	2	2	133	
		2	2.23	4	4	195	5
		4	2.14	3	3	161	
		〃	〃	〃	〃	〃	〃
		〃	〃	〃	〃	〃	7
		2	2.90	3	3	161	8
		4	2.91	3	3	161	
		〃	〃	〃	〃	〃	9
		2	3.65	2	2	133	10
		4	3.66	2	2	133	
		〃	〃	〃	〃	〃	〃
		〃	〃	〃	〃	〃	72
		2	13.20	1	1	95*	73
		4	13.20	1	1	95*	
S10	$10n$	2	0.33	4	3	165	2
		4	0.19	2	2	100	
		2	0.33	4	3	165	3
		4	0.31	3	3	134	
		2	0.36	3	3	134	4
		4	0.34	3	3	134	
		2	0.40	2	2	100	5
		4	0.35	2	2	100	
		〃	〃	〃	〃	〃	〃
		〃	〃	〃	〃	〃	17
		2	0.94	1	1	79	18
		4	0.96	1	1	79	
		〃	〃	〃	〃	〃	〃
		〃	〃	〃	〃	〃	$N-1$
	$10(n+1)$	2	0.63	3	3	141	4
		4	0.62	2	2	110	
		2	0.72	2	2	110	5
		4	0.67	2	2	110	
		〃	〃	〃	〃	〃	〃
		〃	〃	〃	〃	〃	21
		2	2.02	1	1	89	22

		4	2.03	1	1	89	
		〃	〃	〃	〃	〃	〃
		〃	〃	〃	〃	〃	$N-1$
	$15n$	2	0.96	5	4	221	2
		4	0.87	4	4	156	
		2	0.89	4	3	185	3
		4	0.83	3	3	154	
		2	1.02	3	3	151	4
		4	0.96	2	2	120	
		2	1.09	2	2	120	5
		4	1.10	2	2	120	
		〃	〃	〃	〃	〃	〃
		〃	〃	〃	〃	〃	25
		2	4.01	1	1	99	26
		4	4.00	1	1	99	
		〃	〃	〃	〃	〃	〃
		〃	〃	〃	〃	〃	$N-1$
	$15(n+1)$	2	2.38	4	3	194	4
		4	2.17	3	2	169	
		2	2.25	3	2	169	5
		4	2.17	3	2	169	
		2	2.50	3	2	169	6
		4	2.50	3	2	169	
		2	2.80	2	2	135	7
		4	2.81	2	2	135	
		〃	〃	〃	〃	〃	〃
		〃	〃	〃	〃	〃	28
		2	8.71	1	1	114	29
		4	8.70	1	1	114	
		〃	〃	〃	〃	〃	〃
		〃	〃	〃	〃	〃	$N-1$
H3	$10n$	2	0.19	4	3	93	2
		4	0.15	2	1	60	
		2	0.19	4	3	93	3
		4	0.17	3	2	75	
		2	0.19	3	2	75	4
		4	0.18	3	2	75	
		〃	〃	〃	〃	〃	〃
		2	0.20	3	2	75	6
		4	0.18	3	2	75	
		2	0.21	2	2	60	7
		4	0.21	2	2	60	
		〃	〃	〃	〃	〃	〃
		〃	〃	〃	〃	〃	11

	2	0.29	1	1	45	12
	4	0.29	1	1	45	
	〃	〃	〃	〃	〃	〃
	〃	〃	〃	〃	〃	$N-1$
$10(n+1)$	2	0.35	5	3	121	3
	4	0.28	3	2	83	
	2	0.33	4	3	103	4
	4	0.34	3	2	83	
	2	0.36	3	2	83	5
	4	0.36	3	2	83	
	〃	〃	〃	〃	〃	〃
	〃	〃	〃	〃	〃	10
	2	0.60	1	1	53	11
	4	0.60	1	1	53	
	〃	〃	〃	〃	〃	〃
	〃	〃	〃	〃	〃	$N-1$
$15n$	2	0.42	7	3	166	2
	4	0.40	5	3	129	
	2	0.40	5	3	126	3
	4	0.41	4	3	106	
	2	0.47	5	3	126	4
	4	0.51	4	3	106	
	2	0.55	4	2	108	5
	4	0.50	3	2	88	
	2	0.55	3	2	88	6
	4	0.55	3	2	88	
	〃	〃	〃	〃	〃	〃
	〃	〃	〃	〃	〃	10
	2	0.79	2	2	73	11
	4	0.78	2	2	73	
	2	0.89	2	2	73	12
	4	0.90	2	2	73	
	2	0.95	1	1	58	13
	4	0.96	1	1	58	
	〃	〃	〃	〃	〃	〃
	〃	〃	〃	〃	〃	$N-1$
$15(n+1)$	2	1.17	6	3	161	3
	4	1.04	5	3	141	
	2	1.04	5	3	141	4
	4	1.04	5	3	141	
	2	1.10	5	3	141	5
	4	1.11	5	3	141	
	2	1.27	3	2	103	6
	4	1.26	3	2	103	

		〃	〃	〃	〃	〃	〃
		〃	〃	〃	〃	〃	15
		2	2.61	2	2	88	16
		4	2.60	2	2	88	
		2	2.73	1	1	73	17
		4	2.73	1	1	73	
		〃	〃	〃	〃	〃	〃
		〃	〃	〃	〃	〃	$N-1$
H6	$10n$	2	2.87	13	2	414	2
		4	1.91	9	2	294	
		2	0.87	6	2	213	3
		4	0.26	3	1	107	
		2	1.09	5	2	179	4
		4	1.0	5	2	179	
		2	1.39	3	2	127	6
		4	1.33	3	2	127	
		〃	〃	〃	〃	〃	〃
		〃	〃	〃	〃	〃	21
		2	3.33	2	2	106	22
		2	3.32	2	2	106	
		〃	〃	〃	〃	〃	〃
		〃	〃	〃	〃	〃	53
		2	5.50	1	1	84	54
		4	5.51	1	1	84	
		〃	〃	〃	〃	〃	〃
		〃	〃	〃	〃	〃	$N-1$
	$10(n+1)$	2	2.0	4	2	161	6
		4	2.19	4	2	161	
		2	2.43	4	2	161	7
		4	2.40	4	2	161	
		〃	〃	〃	〃	〃	〃
		〃	〃	〃	〃	〃	12
		2	3.60	3	2	137	13
		4	3.62	3	2	137	
		〃	〃	〃	〃	〃	〃
		〃	〃	〃	〃	〃	24
		2	5.93	2	2	116	25
		4	5.92	2	2	116	
		〃	〃	〃	〃	〃	〃
		〃	〃	〃	〃	〃	61
		2	9.94	1	1	94	62
		2	9.94	1	1	94	
		〃	〃	〃	〃	〃	〃
		〃	〃	〃	〃	〃	$N-1$

$15n$	2	8.61	15	2	502	2
	4	7.32	13	2	443	
	2	2.81	10	2	349	3
	4	2.70	10	2	349	
	2	3.40	7	2	270	4
	4	3.30	7	2	270	
	2	3.61	5	2	215	5
	4	3.66	5	2	215	
	2	4.79	5	2	215	6
	4	4.60	5	2	215	
	2	4.62	4	2	181	7
	4	4.64	4	2	181	
	〃	〃	〃	〃	〃	〃
	〃	〃	〃	〃	〃	14
	2	8.57	3	2	157	15
	2	8.60	3	2	157	
	〃	〃	〃	〃	〃	〃
	〃	〃	〃	〃	〃	30
	2	15.27	2	2	136	31
	2	15.27	2	2	136	
	〃	〃	〃	〃	〃	〃
	〃	〃	〃	〃	〃	78
	2	26.20	1	1	114	79
	4	26.21	1	1	114	
	〃	〃	〃	〃	〃	〃
	〃	〃	〃	〃	〃	$N-1$
$15(n+1)$	2	5.78	6	2	245	6
	4	5.10	4	2	196	
	2	5.5	5	2	221	7
	4	6.0	4	2	196	
	2	8.21	4	2	196	8
	4	8.23	4	2	196	
	〃	〃	〃	〃	〃	〃
	〃	〃	〃	〃	〃	17
	2	15.99	3	2	172	18
	4	15.97	3	2	172	
	〃	〃	〃	〃	〃	〃
	〃	〃	〃	〃	〃	34
	2	26.38	2	2	151	35
	4	26.42	2	2	151	
	〃	〃	〃	〃	〃	〃
	〃	〃	〃	〃	〃	70
	〃	〃	1	1	〃	〃
	〃	〃	1	1	〃	$N-1$

* Local minimum found

From Table 2.2 it is clear that the general nature of the results is quite similar. Therefore, in Table 2.3 we summarize the interaction between g and N for S5. The results given here are the values of g for which the smallest number of function evaluations to satisfy the stopping conditions was achieved.

Table 2.3 Summary of results showing effect of g for different N on S5

N	σ	cpu	LS	LM	FE	g
75	2	2.70	2	1	127*	7
	4	2.66	1	1	98*	7
60	2	1.96	1	1	100	11
	4	1.98	1	1	100	11
50	2	0.90	1	1	90	8
	4	0.90	1	1	90	8
40	2	0.48	1	1	77	8
	4	0.48	1	1	77	8
30	2	0.22	2	2	100	7
	4	0.22	2	2	100	7
20	2	0.11	2	2	90	7
	4	0.11	2	2	90	7

* indicates only a local minimum was obtained

It was interesting therefore to see how the best value for g varied amongst the 7 test functions. Table 2.4 summarizes this effect for $N = 10n$.

Table 2.4 Best g for $N = 10n$

	g	n	m
BR	17	2	3
GP	5	2	4
S5	8	4	5
S7	38*	4	7
S10	22	4	10
H3	12	3	4
H6	54	6	4

* This was the largest value for g to produce the global minimum.

Clearly there is no obvious connection between the best value of g and the dimension, n, or the number of local minima, m, but the nature of the function is important since for some functions to get the best value of g it has to be increased until it is close to $N-1$. When $g = N-1$, of course, the only graph minima are the point(s) with smallest function value(s).

In summary, the information on the use of user supplied parameters that was obtained from the numerical test function results was as follows. $N = 10n$ was a reasonable sample size. $\sigma = 4$ was the best value to use for small values of g but as g increased there was little difference between $\sigma = 2$ and $\sigma = 4$. All values of g eventually produced global optima (except for values of g close to $N-1$ for S7) and the number of function evaluations required decreased as g increased but at the expense of extra cpu time. Choice of g would therefore seem to be dependent on the cost of computing the function values. Extrapolation of these suggestions must clearly be treated with caution since the test functions used are all well behaved mathematical functions with a relatively small number of local minima.

Comparison of MSL and TMSL

The major differences between MSL and TMSL lie in the use of the Halton sequence instead of a pseudo-random sequence and the use of the g-topograph rather than sample reduction. To judge the relative importance of these two changes we compared MSL with TMSL and also with MSLH, which is MSL with Halton instead of random sampling and with MSLG, which is MSL with the g-topograph replacing sample reduction. The results are summarised in Table 2.5 where IT represents the number of iterations. In Table 2.5 we have used $N = 50$ and the values of g and σ for which the global minimum was obtained but with the smallest number of function evaluations and the best cpu time.

Table 2.5

	FE	*cpu*	*LS*	*LM*	σ	*IT*	
			MSL				
BR	225	0.10	5	3	2	2	
GP	78	0.04	1	1	4	1	
S5	197	0.08	2	2	4	2	
S7	1705	2.00	21	7	2	4	
S10	574	0.33	6	6	4	3	
H3	82	0.06	2	1	4	1	
H6	304	0.16	6	2	4	2	
			MSLH				
BR	246	0.08	4	3	4	2	
GP	63	0.05	1	1	4	1	
S5	90	0.04	1	1	4	1	
S7	844	0.45	8	6	2	2	
S10	960	0.83	8	8	4	4	
H3	78	0.05	2	1	4	1	
H6	254	0.12	5	2	4	2	
			TMSL				*q*
BR	87	0.35	3	3	4	1	2
GP	63	0.80	1	1	4	1	6
S5	90	0.85	1	1	4	1	6
S7	108	0.69	2	2	2	1	4
S10	110	0.64	2	2	4	1	4
H3	93	0.84	3	3	4	1	6
H6	123	1.20	3	2	4	1	10
			MSLG				
BR	110	0.65	4	3	4	1	4
GP	132	0.66	3	3	4	1	4
S5	112	0.65	2	2	4	1	4
S7	113	0.69	2	2	4	1	4
S10	113	0.68	3	3	4	1	4
H3	101	1.02	3	3	2	1	8
H6	98	0.66	2	2	4	1	4

To make the comparison more visual we have totalled the number of function evaluations and cpu time for all seven functions in Table 2.6.

Table 2.6 Comparison of MSL and TMSL

	FE	cpu
MSL	3165	2.77
MSLH	2535	1.62
MSLG	799	6.18
TMSL	674	5.38

Clearly the dominating factor is the introduction of the g-topograph. The reduction in function evaluations by a factor of about 5 for TMSL is offset by it requiring about twice the cpu time. In the comparisons we tried to be as 'fair' as we possibly could to all four methods but fully realize the difficulties involved, particularly with respect to the stopping condition used. From Table 2.5 it is clear than MSL performed more iterations than TMSL. In fact TMSL stops after the first iteration. This is because MSL uses the reduced sample size in the stopping condition $(\frac{w(\gamma kN-1)}{\gamma kN-w-2} \leq w+\frac{1}{2})$ but TMSL does not $(\frac{w(kN-1)}{kN-w-2} \leq w+\frac{1}{2})$. Therefore we felt that it would be interesting to run TMSL with the total number (cumulative total) of graph minima instead of total sample points since the graph minima correspond to the reduced sample in MSL. So we ran the TMSL algorithm with the stopping condition $\frac{w(G-1)}{G-w-2} \leq w+\frac{1}{2}$, where G is the total number of graph minima up to the k-th iteration. Again we took $N = 50$ and ran TMSL for a moderate value of g, for example we used $g = 6$. The results are shown in Table 2.7.

Table 2.7

	FE	*cpu*	*LS*	*LM*	σ	*IT*
BR	539	7.05	9	3	2	8
	539	7.05	9	3	4	8
GP	772	7.98	10	3	2	9
	772	7.96	10	3	4	9
S5	1049	15.63	7	4	2	17
	1049	15.56	7	4	4	17
S7	1064	17.04	5	4	2	18
	1073	17.10	5	4	4	18
S10	2005	38.89	9	8	2	35
	1910	36.73	9	8	4	33
H3	547	6.43	12	3	2	7
	530	6.35	11	3	4	7
H6	483	4.51	9	2	2	5
	483	4.53	8	2	4	5

From this Table it is clear that the number of iterations, the cpu time and the number of function evaluations are very high. However, if we compare the number of local minima found by TMSL with that of MSL in Table 2.5, we see that there is no significant difference between MSL and TMSL even though this time the number of function evaluations for TMSL is higher. However detailed analysis shows that in almost all cases all local minima were obtained within the first few iterations and the rest of the iterations were continued to accumulate the total number of graph minima G to satisfy the stopping condition.

We also compared TMSL with the currently available algorithms given in Table 2.8 using the number of function evaluations as a basis for comparison and the results are shown in Table 2.9 where AVE is the average of the number of function evaluations of all

test functions for which the global minima were obtained. The results other than that for MSL and TMSL have been taken from the references listed in Table 2.8, the data for TMSL is $N = 10n$, $\sigma = 4$ and $g = 7$ and the data for MSL is $N = 100$,$\gamma = 0.2$ and $\sigma = 2$.

Table 2.8 Listing of different methods

Method	Name	Reference
A	Multistart (MS)	Rinnooy Kan and Timmer (1984)
B	Controlled Random Search (CRS1)	Price (1978)
C	Density Clustering	Törn (1978)
D	Simulated Annealing (SA)	Dekkers and Aarts (1991)
E	Modified Controlled Random Search (CRS5)	Ali and Storey (1995)
F	Modified Controlled Random Search (CRS4)	Ali and Storey (1995)
G	Clustering with distribution function	De Biase and Frontini (1978)
H	Aspiration based Simulated Annealing (ASA)	Ali and Storey (1994a)
I	Multilevel Single Linkage (MSL)	Rinnooy Kan and Timmer (1987a)
J	Topographical Multilevel Single Linkage (TMSL)	Ali and Storey (1994)

Table 2.9 Comparison of TMSL with 9 currently available methods

Method	GP	BR	S5	S7	S10	H3	H6	AVE
A	4400	1600	6500	9300	11000	2500	6000	5900
B	2500	1800	3800	4900	4400	2400	7600	3914
C	2499	1558	3649	3606	3874	2584	3447	3031
D	563	505	365*	558	797	1459	4648	1421
E	402	346	1866	1719	1709	343	1321	1100
F	436	279	1423	1238	1213	545	1581	959
G	378	597	620	788	1160	732	807	726
H	834	408	524	524	524	451	558	532
I	307	206	576	334	1388	166	324	471
J	53	46	98	116	100	60	127	85

* Local minima found

Conclusion

A new global optimization algorithm based on MSL and topographical global optimization, which seems to be robust and competitive with MSL has been developed. Clearly, most of the cpu time needed here is for the construction of graph minima and this could possibly be reduced by a more efficient implementation. We can also anticipate an increase

in efficiency by using a more sophisticated implementation in general. There is clear evidence that great care must be taken over the stopping condition for TMSL and MSL so that the algorithms do not continue needlessly after the global minima have been found.

CHAPTER 3

Simulated Annealing (SA)

3.1 Introduction

Simulated Annealing is a stochastic optimization method, initially designed for discrete optimization especially in the field of combinatorial optimization. It is a Monte Carlo technique which corresponds to the simulation of the physical process of annealing, i.e. the process of driving a physical system to a minimum energy configuration by means of a slow reduction of its temperature. The strong connection between statistical mechanics (behavior of particles in thermal equilibrium at a finite temperature) and combinatorial optimization helped people design the algorithm and so the historical background of the algorithm is an interesting one. SA is one of many heuristic approaches designed to give good, though not necessarily optimal, solutions, within a reasonable computing time. SA has also been extended to optimization problems for continuous variables. In this Chapter a new aspiration based SA algorithm for continuous variables is proposed. However, we first review the SA method in the following subsections.

3.2 Historical Background

Statistical mechanics is the central discipline of condensed matter physics where annealing is known as a thermal process for obtaining low energy states of a solid in a heat bath. First the solid is heated until it melts and then it is cooled by slowly lowering the temperature of the heat bath. This is done by the following scheme.

- Increase the temperature to a value at which the solid melts.
- Decrease the temperature until the particles form a regular pattern.

In the liquid phase all particles of the solid arrange themselves randomly, in this phase the energy is at its highest. But the ground state of the solid, which corresponds to the minimum energy configuration, will have a particular structure, such as seen in a crystal. In practical contexts, low temperature is not a sufficient condition for finding ground states of solids. The ground state of a solid is obtained only if the initial temperature is sufficiently high and the cooling is done sufficiently slowly. Slow cooling is particularly important in such systems as spin glasses (Edwards, 1983) where the Hamiltonian[‡] has a large number of local minima. Starting off at a high temperature, the cooling phase of the annealing

[‡] The Hamiltonian is the internal energy function of a physical system.

process can be described as follows. At each temperature T, the solid is allowed to reach thermal equilibrium. At thermal equilibrium, the probability distribution of the states of the solid follows a Boltzmann distribution where the probability of the system being in a state i with energy E_i at temperature T is given by

$$\Pr\{X = i\} = \frac{1}{Z(T)} \exp(-E_i/k_B T) \ , \tag{3.1}$$

where X is the stochastic variable denoting the current state of the solid. The normalizing constant $Z(T)$ is the partition function, which is defined as

$$Z(T) = \sum_k \exp(-E_k/k_B T) \ , \tag{3.2}$$

where k_B is the Boltzmann constant and the sum is over all possible configurations or energy states. Clearly as the temperature decreases, the Boltzmann distribution concentrates on the states with the lowest energy and finally, when the temperature approaches zero, only the minimum energy states have a non-zero probability of occurance. Physical annealing, therefore, refers to the process of finding the low energy states of a solid by initially melting the substance, and then lowering the temperature slowly. An example would be producing a crystal from the molten substance. If the cooling is too rapid, the resulting solid will be frozen into a meta-stable state (locally optimal structures i.e. the resulting crystal will have many defects, or the substance may form a glass) rather than into the ground state (crystalline lattice structure). Therefore, cooling too quickly means that the disorder encountered at higher temperatures gets frozen in as the temperature is lowered, corresponding to the system sticking in a local minimum of the Hamiltonian.

Physical annealing has been successfully modelled as a Monte Carlo simulation. Metropolis et al. (1953) proposed a method for computing the Boltzmann or equilibrium distribution of a set of particles in a heat bath using a computer simulation. They did this by generating on their computer a collection of particles with a random configuration and calculating the energy of the ensemble. In this method, a given state with energy E_1 is compared to a state that is obtained by moving one of the particles of the state to another arbitrary location (Monte Carlo technique) by a small displacement. This new state, with energy E_2, is accepted if $E_2 - E_1 \leq 0$, i.e., if the move brings the system into a state of lower energy. If $E_2 - E_1 \geq 0$, the new state is not rejected, but accepted with probability $\exp(-(E_2 - E_1)/k_B T)$. So a move to a state of higher energy is accepted in a limited way. By repeating this process for a large enough number of iterations, Metropolis et al. (1953) showed that the Boltzmann distribution is approached at a given temperature. The acceptance rule defined above is known as the Metropolis criterion and the algorithm using it is known as Metropolis algorithm. Thus the Metropolis procedure from statistical

mechanics provides a generalization of iterative improvement in which uphill steps can also be incorporated in the search for a better solution or to escape from local minima.

When annealing begins (with an arbitrary initial state) if the system is kept for a long time at a particular temperature, equilibrium will be attained. Hence for each value of T there is an equilibrium state that will be used as the initial state for the next reduced value of T. Of course, during the solidification, T plays an important role, as it is normally decreased very smoothly and at each time the system is allowed to settle into a new equilibrium. Due to the Metropolis acceptance criterion, at high temperature large fluctuations of the internal energy will still be tolerated, whereas at the end of the cooling process, the value of the energy E will gradually stabilize (relative to T) because, as the temperature decreases, the probability that the system is in a lower energy state increases. Eventually, at the final temperature T_f, the system will freeze into one of the ground states with minimal energy.

3.2.1 Simulation of Annealing in Optimization

Kirkpatrick et al.(1983) introduced a useful analogy between the process of solidifying liquid up to equilibrium, and the properties exhibited by the convergence of combinatorial optimization methods (Papadimitriou and Streiglitz, 1982). The SA algorithm was introduced for combinatorial optimization problems, through simulation of the physical process, by establishing a correspondence between the system's energy (E) and the cost function (f), and between the physical states and solutions (i).

For the annealing temperature T, there is no specific analogue available in combinatorial optimization. In optimization, however, T is used as a control parameter and the perturbation mechanism of the particles in the physical system becomes the generation mechanism of solutions in the combinatorial optimization. In optimization, therefore, one usually attempts to simulate annealing by allowing equilibrium to be approached at a given value of the control parameter before lowering it by some small amount. In the context of optimization slow cooling means that for each value of T a large number of transitions or trials has to be generated. A global optimization algorithm, therefore, can be viewed as a sequence of Metropolis algorithms evaluated at a sequence of decreasing values of the control parameter (T). Assuming that function minimization is required, in most conventional optimization methods, newly generated states are only accepted if the corresponding objective function value decreases. But presumably, it can be advantageous also to allow 'up-hill climbing' under certain conditions. In the SA algorithm this provides a mechanism which enables the algorithm to avoid becoming trapped in a local minimum in its search for the global minimum. Hence, in the simulated annealing approach, a control parameter is defined which plays the same role as the physical temperature; and while lowering this

artificially introduced parameter T, the increments in the objective function which are still accepted become smaller and smaller.

The original annealing algorithm as proposed by Kirkpatrick et al.(1983) is presented below. The algorithm consists of two loops. In the outer loop the temperature is gradually decreased until convergence is detected; in the inner loop a number of random 'moves' in the configuration space are proposed, until a second convergence criterion is satisfied. The moves are accepted or not according to the Metropolis criterion.

The Basic SA Algorithm

Compute the initial control parameter T_o and the initial configuration

while stop criterion not satisfied **do**

begin

while no convergence **do**

begin

Generate move; calculate objective function

if accept **then** update state and objective function

end

Update T

end

Simulated annealing theory has recently lead to some general and powerful global optimization methods. As a result, the SA approach has been the subject of intensive study by mathematicians, statisticians, physicists and computer scientists, and it has also been applied to numerous areas. In nature it is a randomization algorithm and in any practical implementation it behaves as an approximation algorithm. To establish the convergence of the SA algorithm, or to implement it practically, certain quantities, like average energy, entropy etc. have to be addressed. These quantities for a combinatorial optimization problem are defined by analogy to certain microscopic averages in statistical mechanics. Therefore, some important features in the SA algorithm are described below.

The equilibrium distribution

After applying the Metropolis acceptance criterion for a sufficiently large number of transitions at a fixed value of T, the SA algorithm finds a configuration i of a combinatorial optimization problem with a probability equal to

$$q_i(T) = \frac{1}{N(T)} \exp(-f_i/k_B T) \ , \tag{3.3}$$

where

$$N(T) = \sum_{j \in \Omega} \exp(-f_j / k_B T) \tag{3.4}$$

denotes the normalization constant, f_i is the function value for state i and Ω is the state or configuration space i.e., the set of all states. The probability distribution (3.3) is called the stationary or equilibrium distribution and is the equivalent of the Boltzmann distribution (3.1). The normalization constant is the equivalent of the partition function (3.2). The existence of such a distribution is essential for the convergence of the SA algorithm.

The relation between statistical physics and optimization of combinatorial problems can now be made more explicit: given a physical system in thermal equilibrium whose internal states are distributed according to (3.1) and a combinatorial optimization problem whose configurations are distributed according to (3.3), the quantities, expected cost (average energy) at equilibrium, the variance in the cost at equilibrium and the entropy can be defined for optimization problems in a way similar to that for the physical system.

Expected Cost $\langle f \rangle$

Theoretically (Kirkpatrick et al., 1983), the expected mean value of the cost at equilibrium is given by

$$\langle f \rangle = -\frac{d(\ln N(T))}{d(1/k_B T)} , \tag{3.5}$$

where $N(T)$ is defined by (3.4). This equation is clearly equivalent to

$$\langle f \rangle = \sum_{j \in \Omega} f_j q_j(T) , \tag{3.6}$$

where q_j is given by (3.3). (It should be noted that in optimization problems, the Boltzmann constant k_B can be taken as unity.)

Since in (3.6) the summation ranges over the entire search-space Ω the correct average therefore cannot be determined without searching the entire configuration-space which is infeasible. Hence only an estimate can be supplied in any practical implementation of the SA algorithm. Therefore, an estimate such as;

$$\bar{f}(T) = \frac{1}{L} \sum_{i=1}^{L} f_i(T) \tag{3.7}$$

can be used, where L is the number of solutions generated at a particular temperature. Clearly the average cost in (3.7) is an approximation to $\langle f \rangle$.

The Variance in Cost at equilibrium σ^2

The variance of the cost function at equilibrium is given by

$$\sigma^2(T) = \sum_{i\in\Omega}(f_i(T) - \langle f(T)\rangle)^2 q_i(T) \tag{3.8}$$

and the numerical value of the standard deviation of the cost at equilibrium can be approximated by the quantity

$$\sigma(T) = \Big(\frac{1}{L}\sum_{i=1}^{L}(f_i(T) - \bar{f}(T))^2\Big)^{\frac{1}{2}} . \tag{3.9}$$

The entropy S

In thermodynamics, the entropy is related to the average energy in the following way (Kirkpatrick et al., 1983):

$$\frac{dS}{dT} = \frac{1}{T}\frac{d\langle f\rangle}{dT} \tag{3.10}$$

Alternatively, at equilibrium the entropy could be defined as (Otten and van Ginneken, 1984)

$$S = -\sum_{i\in\Omega} q_i \ln(q_i) \tag{3.11}$$

where q_i is given by (3.3). The summation ranges over all configuration space. At high temperatures all states or solutions are accepted and are likely, so S becomes proportional to the logarithm of the total number of feasible states. This follows from;

$$\begin{aligned} S &= -\sum_{i\in\Omega} q_i \ln(q_i) \ , \\ &= -\sum_{i\in\Omega} \frac{1}{|\Omega|} \ln\Big(\frac{1}{|\Omega|}\Big) \ , \\ &= \sum_{i\in\Omega} \frac{1}{|\Omega|} \ln(|\Omega|) \ , \\ &= \ln(|\Omega|) \ , \end{aligned} \tag{3.12}$$

where Ω is assumed to be finite and $|\Omega|$ is the number of states in Ω. Therefore, if the stationary distribution is given by (3.3) then

$$\begin{aligned} &\lim_{T\to\infty} S_\infty = \ln(|\Omega|) \\ &\text{and } \lim_{T\to 0} S_o = \ln(|\Omega_{opt}|) \end{aligned} \tag{3.13}$$

where Ω_{opt} is the set of global minima and S_∞ and S_o are the entropies corresponding to the highest and the lowest temperatures. We therefore remark that in physics, if there is only one ground state, the entropy becomes zero, since

$$\lim_{T\to 0} S_o = \ln(1) = 0 \ . \tag{3.14}$$

A physical meaning can be attached to the entropy as defined above, it is a natural measure of the amount of disorder or information in a system. In the context of optimization it gives some measure for the SA algorithm of the number of states with an energy equal to or less than the presently obtained optimum. Thus the final entropy S_o could be used to see whether or not the global optimum has been found. Unfortunately, this quantity only provides a posteriori check and cannot be determined accurately. So in practice the relevance of S is questionable.

Initial configuration

As the SA algorithm is supposed to be independent of the choice of the starting configuration in the search space (Kirkpatrick et al., 1983), it seems reasonable to select the starting point at random. No heuristic arguments should be taken into account to start from an appropriate guess, in the presumed neighbourhood of the global optimum, since no information whatsoever is available concerning the location of this optimum. In fact, one has to choose the initial temperature such that initially all proposed solutions are accepted. At the initial temperature, a number of random iterations (inner loop) will have to be made so that the initial configuration becomes irrelevant.

The acceptance criterion

When annealing is started, a new state has to be selected in an appropriate way. When the new objective function is computed, the optimization algorithm has to decide whether to accept or reject this solution. As mentioned earlier the SA algorithm is based on the analogy with the cooling of liquid in statistical mechanics. Therefore the majority of the contributions to SA theory adopt the same acceptance criterion as that originally suggested by Metropolis, et.al.,(1953) for the solidification process. Hence, a new state with an energy difference $\Delta f_{ij} = f_j - f_i$ is only retained if the Boltzmann probability

$$A_{ij}(T) = \min\{1, \exp(-\Delta f_{ij}/T)\} \tag{3.15}$$

is greater than a randomly chosen real number in the interval (0,1). This approach relaxes the intrinsic restriction to allow only 'down-hill' moves which is incorporated in methods based on iterative improvement. Therefore, function increases are now tolerated too, but with an exponentially decreasing probability which is a function of the difference between the new objective function value and the current value. Hence in contrast to iterative improvement or gradient methods, the SA technique does not suffer from a heavy dependence on a sufficiently accurate guess for the initial solution (configuration). In Greene (1984) and Greene and Supowit (1986), an alternative acceptance strategy has been proposed, which actually does not reject any attempted state at all. According to these authors, this approach is especially suited for temperatures in the vicinity of the final temperature,

T_f. Later in this Chapter we will, however, propose a new acceptance criterion for the SA algorithm for the continuous variable case.

Cooling Schedule

Typical annealing proceeds by starting the system at some high temperature. The system is then allowed to approach equilibrium, at which time the temperature is reduced, the system allowed to equilibrate again, and so on. The process is stopped at a temperature low enough that no more useful improvement can be expected. This protocol for cooling the system of particles (or solid) is known as the annealing or cooling schedule. In the context of optimization, the determination of the initial temperature, the rate at which the temperature is reduced, the number of iterations at each temperature and the criterion used for stopping is also known as the cooling schedule. In any implementation of the SA algorithm, a cooling schedule must be specified. The temperature parameter, T, is set to an initial value; this is generally relatively high, so that most trials are accepted and there is little chance of the algorithm being unable to move out of a local minimum in the early stages. A scheme is then required to reduce T through the course of the algorithm. Finally, a stopping criterion is required to terminate the algorithm. The choice of cooling schedule has an important bearing on the performance of the SA algorithm and will be further discussed later in this Chapter.

3.2.2 Mathematical Modelling of the Discrete SA Algorithm

SA can be viewed as an algorithm that continuously attempts to transform the current state into one of its neighbours. This mechanism is mathematically best described by means of a Markov chain. A Markov chain is a sequence of trials, where the probability of the outcome of a given trial depends only on the outcome of the previous trial. A Markov chain is therefore described by means of a set of conditional probabilities $p_{ij}(k)$. For each pair of outcomes (i, j), the probability $p_{ij}(k)$ is the probability that the outcome of the k-th trial is j, given that the outcome of the $(k-1)$-th trial is i, i.e.,

$$p_{ij}(k) = \Pr\{X(k) = j | X(k-1) = i\} \ , \tag{3.16}$$

where $X(k)$ is a stochastic variable denoting the outcome of the k-th trial. Moreover, if $a_i(k)$ denotes the probability of outcome i at the k-th trial, i.e., if

$$a_i(k) = \Pr\{X(k) = i\} \tag{3.17}$$

then $a_i(k)$ is given by the following recursion:

$$a_i(k) = \sum_{l \in \Omega} a_l(k-1) p_{li}(k) \ . \tag{3.18}$$

If the conditional probabilities in (3.16) do not depend on k, the corresponding Markov chain is called homogeneous, otherwise it is called inhomogeneous. In the SA algorithm, a trial corresponds to a transition, and the set of outcomes is given by the finite set of states in Ω. Moreover, in the SA algorithm, the outcome of a trial depends only on the outcome of the previous trial. Consequently, we may use the concept of finite homogeneous Markov chains. The $|\Omega| \times |\Omega|$ matrix $P(T)$ whose elements are given by (3.16) for a fixed value of T, is called the transition matrix.

The transition probabilities depend on the value of the control parameter T. Thus if T is kept constant, the corresponding Markov chain is homogeneous and its transition matrix P can be defined as:

$$p_{ij}(T) = \begin{cases} g_{ij}(T)A_{ij}(T), & \forall i \neq j, \\ 1 - \sum_{l=1, l\neq i} g_{il}(T)A_{il}(T), & i = j \end{cases} \tag{3.19}$$

i.e. each transition probability is defined as the product of the following two conditional probabilities: the generation probability $g_{ij}(T)$ of generating configuration j from configuration i, and the acceptance probability $A_{ij}(T)$ of accepting configuration j, once it has been generated from i. The acceptance probability is given by (3.15). The generation probabilities are independent of the control parameters T and are uniform over the neighbourhood of the state i (transitions are implemented by choosing at random a neighbouring configuration j from the current configuration i). The corresponding matrices $G(T)$ and $A(T)$ are called the generation and acceptance matrix, respectively. The definitions of the generation and acceptance probabilities correspond to the original definitions of the SA algorithm and closely follow the physical analogy. Notice that the transition matrix P and generation matrix G are stochastic (a matrix M is stochastic if $m_{ij} \geq 0$, for all i, j, and $\sum_j m_{ij} = 1$, for all i) but the acceptance matrix A is not.

Algorithms based on a more general class of acceptance and generation probabilities are possible as was shown by Lundy and Mees (1986), Anily and Federgruen (1987) and Faigle and Schrader (1988), etc. Hence the following two formulations of SA can be pursued:

- Inhomogeneous algorithms* described by a single inhomogeneous Markov chain. The value of T is decreased between consecutive transitions.
- Homogeneous algorithms described by a sequence of homogeneous Markov chains. Each Markov chain is generated at a fixed value of T and T is decreased between consecutive Markov chains.

* Not used in practice

SA algorithms find a global minimum if the following relation holds:

$$\lim_{k\to\infty} \Pr\left\{X(k) \in \Omega_{opt}\right\} = 1 \ . \tag{3.20}$$

Essential to the convergence proof for a homogeneous algorithm is the existence of a unique stationary distribution. Such a distribution exists only under certain conditions on the Markov chains (Aarts and Korst, 1989) that can be associated with the algorithm. The stationary distribution is defined as the vector $\mathbf{q}$ whose i-th component is given by

$$q_i = \lim_{k\to\infty} \Pr\left\{X(k) = i | X(0) = j\right\}, \ \forall j \ . \tag{3.21}$$

If such a stationary distribution $\mathbf{q}$ exists we have,

$$\begin{aligned} &\lim_{k\to\infty} a_i(k) = \lim_{k\to\infty} \Pr\left\{X(k) = i\right\} \ , \\ &= \lim_{k\to\infty} \sum_j \Pr\left\{X(k) = i | X(0) = j\right\} \Pr\left\{X(0) = j\right\} \ , \\ &= q_i \sum_j \Pr\left\{X(0) = j\right\} = q_i \ . \end{aligned} \tag{3.22}$$

Thus, the stationary distribution is the probability distribution of the outcomes after an infinite number of trials. For the homogeneous algorithm equation (3.20) holds asymptotically provided:

1. Each individual Markov chain is of infinite length.

2. The following conditions on the matrices $A(T)$ and $G(T)$ defining the homogeneous Markov chain hold ensuring that the stationary distribution $\mathbf{q}$ exists.

(i) The Markov chains are irreducible (van Laarhoven and Aarts, 1987), i.e, for all pairs of states (i, j) there is a positive probability of reaching j from i in a finite number of transitions or,

$$\forall i, j \ \exists n, 1 \leq n < \infty \text{ such that, } p_{ij}^n > 0 \ . \tag{3.23}$$

Thus, irreducibility of Markov chains establishes the fact that there is a path from every local minimum to the global minimum.

(ii) The Markov chains are aperiodic (van Laarhoven and Aarts, 1987), i.e, for each state $i \in \Omega$, the greatest common divisor of all integers $n \geq 1$, such that

$$p_{ii}^n > 0 \tag{3.24}$$

is equal to 1. This means that the probability of not leaving the state i in any finite number of steps is greater than zero.

As to establishing irreducibility, in the case of SA, the transition matrix $P(T)$ is defined by (3.19) and for all pairs of states (i, j), the acceptance probability $A_{ij}(T) > 0$ for non-zero T (see 3.15), it is therefore sufficient to assume that the Markov chain induced by the generation matrix $G(T)$ is irreducible itself (since $p_{ij}(T) = g_{ij}(T)A_{ij}(T)$), i.e,

$$\begin{aligned} &\forall i, j \in \Omega \; \exists l_o, l_1, \ldots, l_p \in \Omega (l_o = i, \; l_p = j), \\ &g_{l_k l_{k+1}}(T) > 0, k = 0, 1, \ldots, p-1. \end{aligned} \tag{3.25}$$

As regards aperiodicity, an irreducible Markov chain is aperiodic, if the following condition is satisfied (Romeo and Sangiovanni-Vincentelli, 1985),

$$\forall T > 0 \; \exists i_T \in \Omega : p_{i_T i_T}(T) > 0 \; . \tag{3.26}$$

Therefore under the above considerations the irreducibility and aperiodicity of a Markov chain can be shown to hold. More rigorous proofs and full discussions can be found in Aarts and Korst (1989). The convergence proof for SA remains unchanged for any symmetric generation probability (Lundy and Mees, 1986), i.e, when

$$g_{ij}(T) = g_{ji}(T) \; . \tag{3.27}$$

3.

$$\lim_{t \to \infty} T_t = 0 \tag{3.28}$$

where t is the temperature counter.

The asymptotic convergence proof for the homogeneous Markov chain requires that an infinite number of transitions be generated. Thus implementation of the algorithm requires generation of a sequence of infinitely long homogeneous Markov chains at descending values of the control parameter T, which is impracticable. However it is possible to regard the SA algorithm as a sequence of homogeneous Markov chains of finite length, generated at descending values of the control parameter, so that the homogeneous Markov chains are combined into a single inhomogeneous Markov chain. In this way the infinite sequence of finitely long homogeneous Markov chains becomes a single inhomogeneous Markov chain of infinite length. Therefore, from a theoretical point of view, a simulated annealing algorithm can be modelled as an inhomogeneous Markov process. The conditions for convergence to the global minima, for the inhomogeneous algorithm, not only depend on the matrices $G(T_t)$ and $A(T_t)$ but also impose restrictions on the way the current value of the control parameter, T_t, is changed into the next one, T_{t+1} (Van Laarhoven and Aarts, 1987). However, in this framework a rigorous analysis of the convergence behavior appears to be quite involved (Hajek, 1988; Chiang and Chow, 1988; Tsitsiklis, 1989). Therefore, we only

briefly mention that for the inhomogeneous algorithm, equation (3.20) holds asymptotically if:

1. Certain conditions on the matrices $A(T_t)$ and $G(T_t)$ are satisfied (see Aarts and Korst, 1989).
2. $$\lim_{t \to \infty} T_t = 0 \,. \tag{3.29}$$
3. Under certain additional conditions on the matrix $A(T_t)$, the rate of convergence of the sequence $\{T_t\}$ is not faster than $O\big([\log t]^{-1}\big)$.

The theoretical results for the asymptotic convergence of SA modelled as an inhomogeneous Markov chain do not require the stationary distribution to be achieved at any nonzero temperature (Aarts and Korst, 1989). In fact, there are quite general results about Markov chains (Anily and Federgruen, 1987) showing that if the inhomogeneous Markov chain defined by $P(T_t)$ does converge, then its limit distribution is identical to $\mathbf{q}$. Thus the annealing algorithms (homogeneous and inhomogeneous) will find the optimal solution with probability one, i.e.

$$\lim_{\substack{T \downarrow 0 \\ k \to \infty}} \Pr\Big\{X(k) \in \Omega_{opt}\Big\} = 1 \tag{3.30}$$

In any implementation of the homogeneous algorithm, however, asymptotic convergence can only be approximated. Evidently, this is at the cost of the guarantee of obtaining optimal solutions. In the homogeneous model of the SA algorithm one chooses a few (say, m) different temperature levels, $T_1 \geq T_2 \geq \ldots \geq T_m \approx 0$, and runs a large number, say M, of iterations at each temperature level T_t successively (the reasons will be explained in the next section). In the next section we describe in detail how the approximation is carried out.

3.2.3 Finite-time implementation of the SA algorithm

The description of the SA algorithm in terms of the generation of Markov chains makes it possible to analyze the asymptotic convergence of the algorithm. In practice, one is of course, mainly interested in the finite-time behavior of the algorithm, i.e. in the behavior of SA as an approximation algorithm. In this section, we discuss the behaviour of the homogeneous algorithm in finite time on the basis of the notion of the cooling schedule. The parameters of the cooling schedule are chosen so as to imitate the asymptotic behaviour of the homogeneous algorithm in polynomial time, thereby losing any guarantees with respect to the optimality of the solution retained by the algorithm. We do not describe any approximations to the asymptotic behaviour of the inhomogeneous algorithm. Such approximations are not reported in the literature.

In any finite time implementation of the algorithm the number of transitions for each value T_t, must be finite and $\lim_{t\to\infty} T_t = 0$ can only be approximated by using a finite number of values for T_t. Because of these approximations, the algorithm is no longer guaranteed to find a global minimum with probability 1. Practical implementation of an annealing algorithm, therefore, requires Markov chains of finite length at a finite sequence of descending values of the control parameter. To implement the algorithm, therefore, one must define a cooling schedule (Van Laarhoven and Aarts, 1987; Aarts and Korst, 1989) that governs the convergence of the algorithm. In this section we describe a cooling schedule proposed by Aarts and Van Laarhoven (1985, 1985a). Notice that the same cooling schedule is applicable for the continuous simulated annealing algorithm.

Central in the construction of many cooling schedules is the concept of quasi-equilibrium. If L_t is the length of the t-th Markov chain and $a(\ell, T_t)$ denotes the probability distribution of the solutions after ℓ transitions of the t-th Markov chain, then the homogeneous algorithm is said to be in quasi-equilibrium at T_t if $a(L_t, T_t)$ is close to $\mathbf{q(T_t)}$. The actual construction of a cooling schedule is usually based on the following arguments.

- For $T \to \infty$, the stationary distribution is given by the uniform distribution on Ω, which follows directly from (3.3) and (3.4). Initially, quasi-equilibrium can therefore be achieved by choosing the initial value of T so that virtually all transitions are accepted.
- The length L_t of the t-th Markov chain and the decrement rule for T_t are related through the concept of quasi-equilibrium. If the decrement in T_t is large then it is necessary to attempt more transitions at the new value of T_{t+1} to restore quasi-equilibrium at T_{t+1}. For given quasi-equilibrium at T_t, the larger the decrement in T_t, the larger the difference between $q(T_t)$ and $q(T_{t+1})$ and the longer it takes to establish quasi-equilibrium at T_{t+1}. Therefore, there is a trade-off between fast decrement in T_t and small values for L_t. Usually, one opts for small decrement in T_t to avoid extremely long chains.
- A stop criterion is usually based on the argument that execution of the algorithm can be terminated if the observed improvement in function value over a number of consecutive Markov chains is small.

In this section we discuss briefly the schedule proposed by Aarts and Van Laarhoven (1985). This schedule is based on empirical rules rather than on a choice based on theory. We emphasize once more that this cooling schedule leads to polynomial-time execution of the algorithm on the one hand, but on the other hand precludes any guarantee for the proximity of the final configuration to a globally minimum one. We now give a full description of a cooling schedule proposed by Aarts and Laarhoven (1985). The specifications of the set of parameters that constitutes the cooling schedule are as follows:

- An initial value T_o of the control parameter;
- A decrement function for T;
- A final value for T, i.e. a stopping criterion;
- A finite length L_t, of each Markov chain.

Initial value of the control parameter

Based on the assumption of a Gaussian distribution of the state density function, the initial temperature can be equated with the standard deviation of this normal distribution, as suggested in White (1984). However, in Catthoor et al. (1988) it is argued that this assumption is not generally valid. A less restrictive approach is to require that the maximum possible energy-change is accepted with a sufficiently high probability at T_o (Otten and Ginneken, 1984). In Aarts and Laarhoven (1985) a strategy is proposed in which the initial temperature is assigned based on an experiment for the acceptance ratio. A starting value for T_o is updated until this ratio becomes acceptable. If no upper bound is available for the energy range, this approach is superior to other existing methods. The initial control parameter usually takes a very high value so as to accept almost all trial points, i.e.

$$\exp(-\Delta f_{ij}/T_o) \simeq 1 \ , \tag{3.31}$$

for almost all cost-increasing transitions. This can be achieved by generating a number of trials and requiring that the initial acceptance ratio $\chi_o = \chi(T_o)$ is close to 1, where $\chi(T)$ is defined as the ratio between the number of accepted transitions and the number of proposed transitions. Suppose for a particular value T of the control parameter m_1 transitions have been generated for which $(\Delta f_{ij} < 0)$ and m_2 for which $(\Delta f_{ij} > 0)$ and $\overline{\Delta f^+}$ is the average increase in cost over the m_2 transitions. Then the expected acceptance ratio χ is approximately given by

$$\chi \simeq \frac{m_1 + m_2 \exp(-\overline{\Delta f^+}/T)}{m_1 + m_2} \ ,$$

which can be rewritten as

$$T = \overline{\Delta f^+} \left(\ln \frac{m_2}{m_2\chi - (1-\chi)m_1} \right)^{-1} \ . \tag{3.32}$$

We now assume that the value of T is T_o, which is determined as follows. Initially, T_o is set equal to zero. Next, the algorithm is executed for a fixed number of transitions, say m_o, and after each transition (3.32), with χ set to $\chi_o(= 0.9)$, is used to update the current value of T_o. Numerical experiments indicate that fast convergence to a final value of T_o is obtained in this way (Aarts and Korst, 1989). This final value is then taken as

the initial value of the control parameter. A PASCAL subroutine for generating the initial temperature is given in appendix 3A.

Decrement of the control parameter

The new value T_{t+1} is calculated from

$$T_{t+1} = T_t \left(1 + \frac{T_t \ln(1+\delta)}{3\sigma(T_t)}\right)^{-1} , \tag{3.33}$$

where $\sigma(T_t)$ denotes the standard deviation of the values of the cost function at the points in the Markov chain at T_t. The constant δ is known as the distance parameter and determines the rate of decrease of the control parameter. The numerical value of $\sigma(T)$ is calculated from (3.9).

Final value of the control parameter

The stopping criterion is based on the idea that the average function value $\bar{f}$ over a Markov chain decreases with T_t, so that $\bar{f}(T_t)$ converges to the optimal solution as $T_t \rightarrow 0$. The algorithm is terminated if

$$\left| \frac{d\bar{f}_s(T_t)}{dT_t} \frac{T_t}{\bar{f}(T_o)} \right| < \varepsilon_s , \tag{3.34}$$

where $\bar{f}(T_o)$ is the mean value of f at the points found in the initial Markov chain, $\bar{f}_s(T_t)$ (see Simulated Annealing Procedure) is the smoothed value of $\bar{f}$ over a number of chains in order to reduce the fluctuations of $\bar{f}(T_t)$, and ε_s is a small positive number. The numerical value of $\bar{f}(T_o)$ is found from (3.7).

Length of the Markov chain (L_t)

The length of the Markov chains is usually based on the intuitive argument that for each value T_t of the control parameter, a certain amount of computational effort should be spent to restore quasi-equilibrium. Therefore, a minimum number of transitions should be accepted i.e., L_t is determined so that the number of acceptance transitions is at least η_{min}, a fixed number. However, since transitions are accepted with decreasing probability, one would obtain $L_t \rightarrow \infty$ for $T_t \rightarrow 0$. Consequently, L_t is bounded above by some constant to avoid extremely long Markov chains for low values of T_t (Kirkpatrick, et. al., 1983; Leong and Liu, 1985; Leong, et.al., 1985). The length of Markov chain determines whether the algorithm has explored the neighbourhood of a given point in all directions. For the optimization of continuous variables, Dekkers and Aarts (1991) suggested

$$L_t = 10n , \tag{3.35}$$

where n is the dimension of f (for discrete optimization n may represent the problem size, e.g., the number of cities in the travelling salesman problem). Note that this choice leads to a chain length which is constant for a given problem.

Thus, a homogeneous SA algorithm can be viewed briefly as follows:

- At a fixed temperature level T_t one has to perform nothing but a certain random walk on the collection of all possible configurations.
- Certain parameters have to be chosen in such a way that it is more likely that a good configuration is obtained as t increases (i.e. T_t decreases). In the limit the global optimal solution state will be reached with probability 1.

The above cooling schedule leads to the following simulated annealing algorithm in pseudo-PASCAL.

Procedure Simulated Annealing

begin

initialize a state i by random selection

calculate initial temperature T_o by the procedure described earlier in this section.

$t := 0$; (initialize the temperature change counter t)

stopcriterion:=false;

while stopcriterion=false do (outer loop begins)

begin

$k := 0$; (initialize repetition counter for inner loop)

$L_g := 0$; (initialize solution generation counter for inner loop)

repeat until $k = L_t$; (inner loop begins)

begin

generate state j; (a neighbour of i)

calculate $\Delta f_{ij} := f_j - f_i$; (in moving from i to j)

accept $(\Delta f_{ij}, T_t)$;

if accept=true then

begin

$L_g := L_g + 1$; (increase the number of acceptances by one)

$i := j$; (make j the current state)

end;

$k := k + 1$; (increase number of trial by one)

end; (end of inner loop)

$\bar{f}(T_t) := \frac{1}{L_g} \sum_{i=1}^{L_g} f_i$;

$\sigma(T_t) := \left(\frac{1}{L_g}\sum_{i=1}^{L_g}(f_i - \bar{f}(T_t))^2\right)^{\frac{1}{2}}$;

$T_{t+1} := T_t\left(1 + \frac{T_t \ln(1+\delta)}{3\sigma(T_t)}\right)^{-1}$;

if $t = 0$ then

begin

$\bar{f}_s(T_t) := \bar{f}(T_t)$

stopcriterion:=false

end else

begin

$\bar{f}_s(T_t) := 0.75\bar{f}(T_t) + 0.25\bar{f}(T_{t-1})$;

$\frac{d\bar{f}_s(T_t)}{dT_t} := \frac{\bar{f}_s(T_{t-1}) - \bar{f}_s(T_t)}{T_{t-1} - T_t}$;

stopcriterion:=$\left|\frac{d\bar{f}_s(T_t)}{dT_t}\frac{T_t}{\bar{f}(T_o)}\right| \leq \varepsilon_s$;

end;

$t := t + 1$; (increase the temperature change counter by one)

end; (end of outer loop)

end.

function **accept** $(\Delta f_{ij}, T_t)$;

begin

if $\Delta f_{ij} \leq 0$ then

begin

accept:=true;

else

if $\exp(-\Delta f_{ij}/T_t) >$ random $(0, 1)$ then

accept:=true;

else

accept:=false;

end;

end;

3.3.1 Annealing Algorithm for Continuous Optimization

In combinatorial optimization the number of outcomes (states or solutions) can be very large but is finite. Unfortunately the number of outcomes for the continuous case is infinite.

If the annealing algorithm defined above is to work for the continuous case an appropriate transformation from discrete to continuous is needed. Application of SA to continuous functions has been addressed by a number of authors. The proposed approaches can be divided into the following two classes.

In the first class, implementations of the algorithm are described that follow closely the original physical approach introduced by Kirkpatrick et al.(1983). For example Vanderbilt and Louie (1984) introduced the idea of a covariance matrix for controlling the transition probability. In particular, random points are generated which try to take into account the local structure of the objective function. Khachaturyan (1986) presents a method that is closely related to the physical system as described by Metropolis et al.(1953). Bohachevsky *et al.* (1986) present a simple and easy to implement SA algorithm in which the approach followed is basically that of a random direction method, in which, at each step, a random point is generated on the surface of a sphere centered on the current point with a prefixed radius. Kushner (1987) describes an appropriate method for cost functions, for which the values can only be sampled via a Monte Carlo method.

In the second class of approaches, the annealing process is described by Langevin equations, and proven to converge to the set of global minima (Gidas, 1985; Geman and Hwang, 1986; Chiang, et al., 1987). The so-called Langevin equation in $\mathbb{R}^n$ takes the form

$$dx(t) = -\nabla f(x(t))dt + \sqrt{2T(t)}dw(t) \tag{3.36}$$

where ∇f is the gradient of the function f, $T(t)$ the temperature at time $t \in [0, \infty)$ and $w(t)$ is the standard Brownian motion in $\mathbb{R}^n$. The equation (3.36) can be seen, from the point of view of stochastic optimization algorithms, as the law of motion of a point in $\mathbb{R}^n$ whose movement is subject to two different components: one is the tendency to follow down-hill trajectories along the direction of $-\nabla f$; the other is a random fluctuation whose amplitude is governed by the temperature parameter $T(t)$. Aluffi-Pentini, et. al (1985) proposed the computation of global minima by following the paths of a system of stochastic differential equations. They use a time-dependent function for the acceptance criterion which tends to zero in a suitable way. The papers of Geman and Hwang (1986) and Chiang et. al. (1987) consider the same concept. A continuous path seeking a global minimum will, in general, be forced to 'climb hills', with a standard n-dimensional Brownian motion, as well as follow down-hill gradients. The Brownian motion is controlled by a time dependent factor, tending to zero as time goes to infinity. The convergence proof given by Geman and Hwang (1986) is based on the Langevin equations.

However, it is clear that these methods are somewhat different from the original SA approach to discrete optimization. Recently Dekkers and Aarts (1991) gave an algorithm for the continuous problem which is a direct transformation from the original discrete to

the continuous case. The convergence proof is based on the equilibrium distribution of Markov chains. This method has proved to be a reliable annealing method for continuous optimization. We, therefore, restrict ourself to this version of the simulated annealing algorithm. For the continuous case we will denote points in Ω by x, y etc.

As far as SA is concerned, a near optimum solution is often considered as the global solution (Dekkers and Aarts, 1991). A near minimal solution can be formalized in the following way.

For $\epsilon > 0$, we call a point $x \in \Omega$ near minimal if $x \in B(\epsilon)$ where $B(\epsilon) = A_x(\epsilon) \cup A_f(\epsilon)$ and $A_x(\epsilon)$ and $A_f(\epsilon)$ are defined by (1.3) and (1.4) respectively, in Chapter 1.

3.3.2 Mathematical Model of the Continuous Algorithm

We now present a mathematical model of the homogeneous, simulated annealing algorithm for continuous optimization based on the ergodic theory of Markov chains. The following definitions are introduced by Dekkers and Aarts (1991).

Definition 3.1 The transition probability of transforming $x \in \Omega$ into a point $y \in C \subset \Omega$ is the probability of generating and accepting a point in C if $x \notin C$. Thus, if x is the current point of the Markov chain, then the probability that an element in C is the next point of the Markov chain is

$$P(C|x;T) = \begin{cases} \int_{y \in C} p_{xy}(T)dy & \text{for } x \notin C, \\ \int_{y \in C} p_{xy}(T)dy + \left(1 - \int_{y \in \Omega} p_{xy}(T)dy\right) & \text{for } x \in C, \end{cases} \tag{3.37}$$

where

$$p_{xy}(T) = g_{xy}A_{xy}(T) \ , \tag{3.38}$$

and

$$P(C|x;T) = \Pr\{X(k) \in C | X(k-1) = x; T\} \ . \tag{3.39}$$

Note that $p_{xy}(T)$ is not a proper distribution function since

$$\int_{y \in \Omega} p_{xy}(T)dy \neq 1 \ . \tag{3.40}$$

Therefore, $p_{xy}(T)$ is called the quasi-probability distribution function. The acceptance probability $A_{xy}(T)$ is similar to that of discrete optimization (Aarts and Korst, 1989).

Definition 3.2 The probability that a point $x \in \Omega$ is transformed into a point $y \in C \subset \Omega$ in k trials is

$$P^{(k)}(C|x;T) = \begin{cases} \int_{y \in C} p_{xy}^{(k)}(T)dy & \text{for } x \notin C, \\ \int_{y \in C} p_{xy}^{(k)}(T)dy + \left(1 - \int_{y \in \Omega} p_{xy}(T)dy\right)^k & \text{for } x \in C, \end{cases} \tag{3.41}$$

where

$$p_{xy}^{(k)}(T) = \int_{z\in\Omega} p_{xz}^{(k-1)}(T)p_{zy}(T)dz + p_{xy}^{(k-1)}(T)\big(1 - \int_{z\in\Omega} p_{yz}(T)dz\big) + \big(1 - \int_{z\in\Omega} p_{xz}(T)dz\big)^{k-1} p_{xy}(T)\ . \tag{3.42}$$

Notice that $p_{xy}^{(k)}(T)$ is the quasi-probability distribution function of transforming x into y in k trials, and is equal to the sum of three terms:

The first term is the quasi probability distribution function of transforming x into z in $k-1$ trials, and from z to y in the next trial generated over all z.

The second term is the quasi probability distribution function of transforming x into y in $k-1$ trials and then rejecting the k-th trial.

The third term is the quasi probability distribution function of transforming x into y in one trial after $k-1$ rejected trials from x.

The aim of the continuous SA algorithm is achieved if it converges asymptotically to a point $x \in A_f(\epsilon)$ (here we assume that $A_f(\epsilon) = \Omega_{opt}$) that is if

$$\forall_{\epsilon>0}: \quad \lim_{\substack{T\downarrow 0 \\ k\to\infty}} \Pr\{X(k) \in A_f(\epsilon)\} \geq 1-\epsilon \tag{3.43}$$

for all starting points $X(0)$. The proof of asymptotic convergence is based on the convergence proof for the SA algorithm when applied to the discrete minimization problem (Dekkers and Aarts, 1991). Essential to the proof is the fact that under certain conditions there exists a unique stationary probability distribution function of a homogeneous Markov chain.

Definition 3.3 A probability distribution function $r(x,T)$ is stationary if

$$\forall_{x\in\Omega}: r(x,T) = \int_{y\in\Omega} r(y,T)p_{yx}(T)dy + r(x,T)\big(1 - \int_{y\in\Omega} p_{xy}(T)dy\big) \tag{3.44}$$

and

$$\int_{x\in\Omega} r(x,T)dx = 1 \tag{3.45}$$

Dekkers and Aarts (1991) have proved the following theorem.

Theorem: Let $p_{xy}(T)$ be given by Definition 3.1 and let Ω be the only ergodic set not having any cyclically moving subsets for the Markov chain induced by $P(C|x;T)$ (Definition 3.1). Furthermore, let the following conditions be satisfied:

(i)

$$\forall_{x,y\in\Omega}: \ g_{xy}(T) = g_{yx}(T)\ . \tag{3.46}$$

(ii) $g_{xy}(T)$ is independent of T (and therefore can be written as g_{xy}).

Then a unique stationary probability distribution function exists and is given by

$$q(x,T) = \exp(-(f_x - f_{min})/T)/N(T) \ , \tag{3.47}$$

where f_{min} is the minimum function value and

$$N(T) = \int_{y\in\Omega} \exp(-(f_y - f_{min})/T)dy \ . \tag{3.48}$$

Dekkers and Aarts (1991) also proved that the SA algorithm, for continuous minimization, modelled as a Markov chain with the transition probability (3.37) defined by (3.38) and (3.39), converges to the set of minimal points (i.e. to $A_f(\epsilon)$) of f if the following conditions are met:

f is uniformly continuous.

All minima are interior points of Ω.

The number of minima is finite.

$A_{xy}(T) = \min\{1, \exp(-(f_y - f_x)/T)\}$.

The generation probability distribution function $g_{xy}(T)$ is defined by

$$\forall_{x_o\in\Omega}\forall_{C\subset\Omega} : \ m(C) > 0 \Rightarrow \int_{y\in C} g_{x_o y}(T)dy > 0 \tag{3.49}$$

which means if the set C has positive measure then the probability that a point $y \in C$ will be generated from $x \in \Omega$ is greater than zero.

$g_{xy}(T) = g_{yx}(T)$.

$g_{xy}(T)$ does not depend on T.

However, these conditions are sufficient but not necessary.

To implement SA on a function of continuous variables, the crucial factor is the choice of an appropriate neighbourhood structure, i.e. the way in which a neighbour y of the current point x is defined. Thus the requirements are that the generation mechanism should satisfy (3.46) with g_{xy} being independent of T and

$$\forall_{x\in\Omega}\forall_{C\subset\Omega} : \ m(C) > 0 \Rightarrow \int_{y\in C} g_{xy}(T)dy > 0 \ ,$$

where $m(C)$ is the Lebesgue measure of the set C. The following two alternative generation mechanisms are discussed by Dekkers and Aarts (1991).

Alternative A. A uniform distribution on Ω, i.e.

$$g_{xy} = \frac{1}{m(\Omega)} \,. \tag{3.50}$$

Clearly, this alternative satisfies the above requirements. An obvious disadvantage of this choice is that no structural information about function values is used. This disadvantage can be circumvented by introducing an additional mechanism that uses descent directions.

Alternative B.

$$g_{xy} = \begin{cases} LS(x), & \text{if } \omega \geq t_o, \\ \frac{1}{m(\Omega)}, & \text{if } \omega < t_o, \end{cases} \tag{3.51}$$

where t_o is a fixed number in $(0,1)$, and ω a uniform random number on $(0,1)$. $LS(x)$ denotes a local search that generates a point y in a descent direction from x, thus $f_y < f_x$ (y is not necessarily a local minimum). In this implementation, however, $g_{xy} \neq g_{yx}$. Nonetheless, it can be shown that the method still converges to a neighbourhood of the optimal solution (Dekkers and Aarts 1991).

3.4.1 The Aspiration based SA Algorithm (ASA)

The SA algorithm sometimes accepts solutions which are worse than the current solution. It is therefore possible in any single SA run for the final solution to be worse than a solution found during the run. In fact, since the SA algorithm is a randomization device, which by means of an acceptance/rejection criterion allows some ascent steps during the optimization process, it is quite possible that at some fixed temperature level the procedure will visit the near global optimal solution but due to the acceptance/rejection mechanism it (the procedure) will leave the best solution and arrive at a worse solution. In addition, since the algorithm is heavily dependent on the cooling schedule and an appropriate cooling schedule is very difficult to construct, the algorithm may never come back to the best solution it left during the course of a run. The SA procedure is completely memoryless, i.e., new solutions are accepted disregarding previously obtained intermediate results. The SA algorithm therefore has the following shortcomings:

Simulated annealing does not use strategic decision rules which could be based on knowledge of the global problem structure.

No learning procedure is incorporated to make effective use of information gained in previous iterations.

An aspiration[†] based SA algorithm has been designed to take into account the above drawbacks by adapting the acceptance criterion in a suitable way. Let x_t be the starting

[†] The concept of aspiration level was first introduced by Glover (1989) in his TABU search technique for combinatorial optimization.

solution of the t-th Markov chain and x_a the point obtained by carrying out a local search from x_t. We then define $f_a(x_t)$ as $f(x_a)$. The local descent procedure is not a complete local search but only a few steps of some appropriate descent local search. During the execution of the t-th Markov chain with length L_t, if a solution is generated whose function value is less than or equal to the aspiration value, then no more attempt is made to generate the next solution. In other words the inner loop stops, the aspiration value is updated and a new Markov chain begins. If a solution cannot be found whose function value is less than $f_a(x_t)$ then the complete chain of length L_t is executed and the aspiration value is not updated at the begining of the next Markov chain.

As far as simulated annealing for discrete optimization is concerned many researchers have considered alternative acceptance probabilities (Romeo and Sangiovanni-Vincentelli, 1985; Anily and Federgruen, 1987 and Faigle and Schrader, 1988). In all cases theoretical results regarding asymptotic convergence have been established. Romeo and Sangiovanni-Vincentelli provide some experimental evidence that an alternative acceptance scheme does not significantly alter the quality of solutions found. In reality, a system of cooling particles and an optimization problem are not same and therefore the simulation has to be adapted. In our modification of the SA algorithm we use an acceptance criterion which is independent of the current function value f_x whenever Δf_{xy} is positive. This acceptance criterion is given by

$$A^a_{xy}(T) = \min(1, A^*_{xy}(T)) \ , \tag{3.52}$$

where

$$A^*_{xy}(T) = \begin{cases} \exp\big(-(f_y - f_a)/T\big) & \text{if } f_y > f_x \geq f_a \ , \\ 1 & \text{otherwise} \ , \end{cases} \tag{3.53}$$

where f_a is the current aspiration value. In our implementation of ASA we use the generation mechanism defined by **Alternative B**. We also use the same method as was used by Dekkers and Aarts (1991) to determine the initial temperature. A brief description of how the ASA algorithm works is introduced below.

The initial temperature T_o is found by applying the original Metropolis acceptance criterion and using **Alternative B** as a solution generation scheme. Therefore, the initial temperature calculating scheme of ASA is the same as that of SA and the same T_o will be produced if we use exactly the same local search in **Alternative B**. A few local descent steps are then taken from a random starting point x_o of the initial Markov chain, the resulting solution gives the aspiration value and then the initial chain begins. The regularly updated aspiration value, allows us to have effective information on the objective function as the search proceeds. At the begining of each Markov chain (say, the t-th Markov chain) the aspiration value is updated if required and the length, L_t, is then determined. The greater the difference between f_{x_t}, the starting solution of the t-th Markov chain, and f_a, the aspiration value, the longer will be the current Markov chain. Therefore, the aspiration

value f_a plays a part in determining the length of a particular chain in a way to be clarified later. An inner loop of ASA starts at x_t with the acceptance criterion defined by (3.52) and new trial solutions are then attempted. If the newly generated solution f_y is greater than the current solution f_x then it is clear from (3.52) and (3.53) that the acceptance probability, $A^a_{xy}(T)$ will be less than the Metropolis acceptance probability defined by (3.15) and consequently the transition probability, $p^a_{xy}(T)$, in ASA will satisfy

$$p^a_{xy}(T) = g_{xy} \cdot A^a_{xy}(T) \leq p_{xy}(T) \tag{3.54}$$

From (3.54) it is clear that the ASA procedure is more aggressive than SA but this can be justified by the following arguments:

At the start of an inner loop the higher the difference between the starting and aspiration solutions, the lower will be the transition probability for lower to higher solutions. This can be compensated by considering a proportionately lengthy Markov chain. Moreover we will see later that ASA may increase the temperature at some stage of the procedure. Therefore, the above considerations will balance the effect of aggressiveness of the search.

No doubt SA proceeds in the right direction with the decreasing temperature as far as the global minimum is concerned but it cannot memorize the best solution found during the course of its search. ASA can safeguard the best solution and its aspiration value is a useful tool which can be used as a guide to the procedure. Moreover, it will not be reasonable to accept very high solutions if a solution is known (aspiration solution) whose function value is much lower than the current one. Therefore, an aggressive search can be justified.

In ASA, the choice of L_t is not constant throughout the course of the algorithm, it can vary from short to long depending upon the present aspiration solution. If the inner loop starts with a particular T_t and a solution is reached whose function value is lower than the present aspiration value (f_a) then at this point the aspiration level is updated, a new inner loop starts with T_{t+1}, and the process continues. For some iterations the number of trial solutions could be very small but this is not a drawback. In fact, Glover and Greenberg (1989) argue that there is little need for the SA algorithm to rely on a strong stabilizing effect over time. In other words, there is no need to consider very long Markov chains. In their implementation they also store details of the best solution found so far and consider this to be the final solution. These ideas are supported by Connolly's (1988) modification of SA, where having found a suitable fixed temperature, all the remaining iterations are carried out that temperature. In the final phase, a descent algorithm can be carried out from the best solution found at the earlier phases.

Central to the construction of ASA there is a mechanism for keeping track of the best solution during the course of the procedure. This mechanism can also send signals to the procedure to increase the temperature if required. The question, of course, arises whether it is possible to perform such modifications and, at the same time, keep the convergence properties of SA. We investigate the impact on the convergence of the modification of the acceptance criterion. Indeed, we will see that the basic theoretical ASA algorithm will converge to the optimal solution no more slowly than the algorithm described by Dekkers and Aarts (1991).

3.4.2 Theoretical Investigation

In the implementation of the SA algorithm for continuous variables Dekkers and Aarts (1991) suggested adopting the generation mechanism, **Alternative B**. But this generation mechanism implies that $g_{xy} \neq g_{yx}$. However, if $X(k)$ and $Y(k)$ are defined as the outcomes of the trials in the SA algorithm using **Alternative A** and **Alternative B**, respectively, then Dekkers and Aarts proved that

$$\begin{aligned} \forall_{\epsilon>0}: \quad & \lim_{\substack{T \downarrow 0 \\ k \to \infty}} \Pr\{Y(k) \in A_f(\epsilon)\} \\ \geq \quad & \lim_{\substack{T \downarrow 0 \\ k \to \infty}} \Pr\{X(k) \in A_f(\epsilon)\} \geq 1 - \epsilon \,. \end{aligned} \tag{3.55}$$

Theorem 3.1 Let the random variables $R(k)$ and $Z(k)$ be defined as the outcomes of the trials using Alternative A and Alternative B respectively but with transition probability (3.54). Then

$$\forall_{\epsilon>0}: \quad \lim_{\substack{T \downarrow 0 \\ k \to \infty}} \Pr\left\{Z(k) \in A_f(\epsilon) | T\right\} \geq \lim_{\substack{T \downarrow 0 \\ k \to \infty}} \Pr\left\{Y(k) \in A_f(\epsilon) | T\right\} > 1 - \epsilon \tag{3.56}$$

Proof. Let

$$PB(T) = \Pr\{X(k) \in A_f(\epsilon) | X(k-1) \in A_f(\epsilon); T\} \tag{3.57}$$

$$PB'(T) = \Pr\{R(k) \in A_f(\epsilon) | R(k-1) \in A_f(\epsilon); T\} \tag{3.58}$$

$$PLS(T) = \Pr\{LS(Y(k-1)) \in A_f(\epsilon) | Y(k-1) \notin A_f(\epsilon); T\} \tag{3.59}$$

$$PLS'(T) = \Pr\{LS(Z(k-1)) \in A_f(\epsilon) | Z(k-1) \notin A_f(\epsilon); T\} \tag{3.60}$$

$$\begin{aligned} & \Pr\{Z(k) \in A_f(\epsilon) | Z(k-1) \in A_f(\epsilon); T\} \\ & = t_o \Pr\{R(k) \in A_f(\epsilon) | R(k-1) \in A_f(\epsilon); T\} \\ & + (1 - t_o) \Pr\{LS(Z(k-1)) \in A_f(\epsilon) | Z(k-1) \in A_f(\epsilon); T\} \\ & = t_o \Pr\{R(k) \in A_f(\epsilon) | R(k-1) \in A_f(\epsilon); T\} + (1 - t_o) \,. \end{aligned}$$

Similarly

$$\begin{aligned}&\Pr\{Y(k) \in A_f(\epsilon) | Y(k-1) \in A_f(\epsilon); T\} \\ &= t_o \Pr\{X(k) \in A_f(\epsilon) | X(k-1) \in A_f(\epsilon); T\} + (1 - t_o) \ .\end{aligned}$$

But, due to the effect of the acceptance criterion (3.52)

$$\begin{aligned}&\Pr\{R(k) \in A_f(\epsilon) | R(k-1) \in A_f(\epsilon); T\} \\ &\geq \Pr\{X(k) \in A_f(\epsilon) | X(k-1) \in A_f(\epsilon); T\} \ .\end{aligned} \tag{3.61}$$

Therefore,

$$t_o PB'(T) + (1 - t_o) \geq t_o PB(T) + (1 - t_o) \ . \tag{3.62}$$

Again

$$\begin{aligned}&\Pr\{Z(k) \in A_f(\epsilon) | Z(k-1) \notin A_f(\epsilon); T\} \\ &= t_o \Pr\{R(k) \in A_f(\epsilon) | R(k-1) \notin A_f(\epsilon); T\} \\ &+ (1 - t_o) \Pr\{LS(Z(k-1)) \in A_f(\epsilon) | Z(k-1) \notin A_f(\epsilon); T\} \\ &= t_o \frac{m(A_f(\epsilon))}{m(\Omega)} + (1 - t_o) \Pr\{LS(Z(k-1)) \in A_f(\epsilon) | Z(k-1) \notin A_f(\epsilon); T\} \ .\end{aligned}$$

Similarly

$$\begin{aligned}&\Pr\{Y(k) \in A_f(\epsilon) | Y(k-1) \notin A_f(\epsilon); T\} \\ &= t_o \frac{m(A_f(\epsilon))}{m(\Omega)} + (1 - t_o) \Pr\{LS(Y(k-1)) \in A_f(\epsilon) | Y(k-1) \notin A_f(\epsilon); T\} \ .\end{aligned}$$

Since $PLS'(T) = PLS(T)$ we have

$$t_o \frac{m(A_f(\epsilon))}{m(\Omega)} + (1 - t_o) PLS(T) = t_o \frac{m(A_f(\epsilon))}{m(\Omega)} + (1 - t_o) PLS'(T)) \tag{3.63}$$

$$\begin{aligned}&\Pr\{Z(k) \notin A_f(\epsilon) | Z(k-1) \in A_f(\epsilon); T\} \\ &= t_o \Pr\{R(k) \notin A_f(\epsilon) | R(k-1) \in A_f(\epsilon); T\} \\ &+ (1 - t_o) \Pr\{LS(Z(k-1)) \notin A_f(\epsilon) | Z(k-1) \in A_f(\epsilon); T\} \\ &= t_o (1 - \Pr\{R(k) \in A_f(\epsilon) | R(k-1) \in A_f(\epsilon); T\}) \ .\end{aligned}$$

Similarly

$$\begin{aligned}&\Pr\{Y(k) \notin A_f(\epsilon) | Y(k-1) \in A_f(\epsilon); T\} \\ &= t_o (1 - \Pr\{X(k) \in A_f(\epsilon) | X(k-1) \in A_f(\epsilon); T\}) \ .\end{aligned}$$

Using (3.61) we have

$$t_o (1 - PB'(T)) \leq t_o (1 - PB(T)) \tag{3.64}$$

$$\Pr\{Z(k) \notin A_f(\epsilon) | Z(k-1) \notin A_f(\epsilon); T\}$$
$$= t_o \Pr\{R(k) \notin A_f(\epsilon) | R(k-1) \notin A_f(\epsilon); T\}$$
$$+ (1-t_o) \Pr\{LS(Z(k-1)) \notin A_f(\epsilon) | Z(k-1) \notin A_f(\epsilon); T\}$$
$$= t_o\big(1 - \frac{m(A_f(\epsilon))}{m(\Omega)}\big) + (1-t_o)(1 - \Pr\{LS(Z(k-1)) \in A_f(\epsilon) | Z(k-1) \notin A_f(\epsilon); T\}) \ .$$

Therefore

$$t_o\big(1-\frac{m(A_f(\epsilon))}{m(\Omega)}\big)+(1-t_o)(1-PSL(T)) = t_o\big(1-\frac{m(A_f(\epsilon))}{m(\Omega)}\big)+(1-t_o)(1-PSL'(T)) \quad (3.65)$$

Consequently using (3.57)-(3.60)

$$E(\text{waiting time of } Z(k) \text{ in } A_f(\epsilon)|T)$$
$$= \sum_{k=1}^{\infty} k \times \Pr\{\forall_{0 \le i < k} : Z(i) \in A_f(\epsilon) \text{ and } Z(k) \notin A_f(\epsilon) | Z(0) \in A_f(\epsilon); T\}$$
$$= \sum_{k=1}^{\infty} k\big[t_o PB'(T) + (1-t_o)\big]^{(k-1)}\big[t_o(1-PB'(T)\big]$$
$$= t_o(1-PB'(T)) \sum_{k=1}^{\infty} k\big[t_o PB'(T) + (1-t_o)\big]^{(k-1)}$$
$$= t_o(1-PB'(T)) \frac{1}{[t_o(1-PB'(T)]^2} = [t_o(1-PB'(T)]^{-1}$$

$$E(\text{waiting time of } Z(k) \text{ in } \Omega \backslash A_f(\epsilon)|T)$$
$$= \sum_{k=1}^{\infty} k \times \Pr\{\forall_{0 \le i < k} : Z(i) \notin A_f(\epsilon) \text{ and } Z(k) \in A_f(\epsilon) | Z(0) \notin A_f(\epsilon); T\}$$
$$= \sum_{k=1}^{\infty} k\big[t_o\big(1 - \frac{m(A_f(\epsilon))}{m(\Omega)}\big) + (1-t_o)(1-PLS'(T))\big]^{k-1}\big[t_o \frac{m(A_f(\epsilon))}{m(\Omega)}$$
$$+ (1-t_o)(1-PLS'(T))\big]$$
$$= \big[t_o \frac{m(A_f(\epsilon))}{m(\Omega)} + (1-t_o)PLS'(T)\big] \frac{1}{\big[t_o \frac{m(A_f(\epsilon))}{m(\Omega)} + (1-t_o)PLS'(T)\big]^2}$$
$$= \big[t_o \frac{m(A_f(\epsilon))}{m(\Omega)} + (1-t_o)PLS'(T)\big]^{-1} \ .$$

Similarly

$$E(\text{waiting time of } R(k) \text{ in } A_f(\epsilon)|T)$$
$$= (1-PB'(T))^{-1} \ .$$

$$E(\text{waiting time of } R(k) \text{ in } \Omega \backslash A_f(\epsilon)|T)$$
$$= \frac{m(\Omega)}{m(A_f(\epsilon))}$$

Dekkers and Aarts (1991) proved that the SA algorithm for continuous variables converges to the optimal solution using the generation mechanism (3.51). In other words, they have proved that

$$\forall_{\epsilon>0}: \quad \lim_{\substack{T\downarrow 0 \\ k\to\infty}} \Pr\{Y(k)\in A_f(\epsilon)|Y(0)\in\Omega;T\}\geq 1-\epsilon$$

$$\Rightarrow \frac{[t_o(1-PB(T))]^{-1}}{[t_o(1-PB(T))]^{-1}+[t_o\frac{m(A_f(\epsilon))}{m(\Omega)}+(1-t_o)PLS(T)]^{-1}}\geq 1-\epsilon$$

If the ASA algorithm converges to the optimal solution then

$$\forall_{\epsilon>0}: \quad \lim_{\substack{T\downarrow 0 \\ k\to\infty}} \Pr\{Z(k)\in A_f(\epsilon)|Z(0)\in\Omega;T\}\geq 1-\epsilon \tag{3.66}$$

But (3.66) is defined by

$$\frac{E(\text{waiting time of } Z(k) \text{ in } A_f(\epsilon)|T)}{E(\text{waiting time of } Z(k) \text{ in } A_f(\epsilon)|T)+E(\text{waiting time of } Z(k) \text{ in } \Omega\backslash A_f(\epsilon)|T)}\geq 1-\epsilon$$

$$\Rightarrow \frac{[t_o(1-PB'(T))]^{-1}}{[t_o(1-PB'(T))]^{-1}+[t_o\frac{m(A_f(\epsilon))}{m(\Omega)}+(1-t_o)PLS'(T)]^{-1}}\geq 1-\epsilon$$

Now using (3.64)

$$[t_o(1-PB'(T))]^{-1}\geq[t_o(1-PB(T))]^{-1}$$

$$\Rightarrow \frac{1}{[t_o(1-PB'(T))]^{-1}}\leq\frac{1}{[t_o(1-PB(T))]^{-1}}$$

$$\Rightarrow \frac{[t_o\frac{m(A_f(\epsilon))}{m(\Omega)}+(1-t_o)PLS(T)]^{-1}}{[t_o(1-PB'(T))]^{-1}}\leq\frac{[t_o\frac{m(A_f(\epsilon))}{m(\Omega)}+(1-t_o)PLS(T)]^{-1}}{[t_o(1-PB(T))]^{-1}}$$

$$\Rightarrow 1+\frac{[t_o\frac{m(A_f(\epsilon))}{m(\Omega)}+(1-t_o)PLS(T)]^{-1}}{[t_o(1-PB'(T))]^{-1}}\leq 1+\frac{[t_o\frac{m(A_f(\epsilon))}{m(\Omega)}+(1-t_o)PLS(T)]^{-1}}{[t_o(1-PB(T))]^{-1}}$$

$$\Rightarrow \frac{[t_o(1-PB'(T))]^{-1}+[t_o\frac{m(A_f(\epsilon))}{m(\Omega)}+(1-t_o)PLS(T)]^{-1}}{[t_o(1-PB'(T))]^{-1}}$$

$$\leq\frac{[t_o(1-PB(T))]^{-1}+[t_o\frac{m(A_f(\epsilon))}{m(\Omega)}+(1-t_o)PLS(T)]^{-1}}{[t_o(1-PB(T))]^{-1}}$$

$$\Rightarrow \frac{[t_o(1-PB'(T))]^{-1}}{[t_o(1-PB'(T))]^{-1}+[t_o\frac{m(A_f(\epsilon))}{m(\Omega)}+(1-t_o)PLS(T)]^{-1}}$$

$$\geq\frac{[t_o(1-PB(T))]^{-1}}{[t_o(1-PB(T))]^{-1}+[t_o\frac{m(A_f(\epsilon))}{m(\Omega)}+(1-t_o)PLS(T)]^{-1}}$$

However by (3.63)

$$[t_o \frac{m(A_f(\epsilon))}{m(\Omega)} + (1-t_o)PLS(T)]^{-1}$$
$$= [t_o \frac{m(A_f(\epsilon))}{m(\Omega)} + (1-t_o)PLS'(T)]^{-1}$$

Therefore

$$\frac{[t_o(1-PB'(T))]^{-1}}{[t_o(1-PB'(T))]^{-1} + [t_o \frac{m(A_f(\epsilon))}{m(\Omega)} + (1-t_o)PLS'(T)]^{-1}} \geq$$

$$\frac{[t_o(1-PB(T))]^{-1}}{[t_o(1-PB(T))]^{-1} + [t_o \frac{m(A_f(\epsilon))}{m(\Omega)} + (1-t_o)PLS(T)]^{-1}} \geq 1-\epsilon$$

So

$$\frac{[t_o(1-PB'(T))]^{-1}}{[t_o(1-PB'(T))]^{-1} + [t_o \frac{m(A_f(\epsilon))}{m(\Omega)} + (1-t_o)PLS'(T)]^{-1}} \geq 1-\epsilon \qquad (3.67)$$

This completes the proof of Theorem 3.1.

3.4.3 An Adaptive Polynomial-time Cooling Schedule

In addition to the cooling schedule described in section 3.2.3, a number of cooling schedules have been reported in the literature (Otten and van Ginneken, 1984; Lam and Delosme, 1986 and Huang and et al., 1986). However, there has always been an open question as to how fast the simulation should be 'cooled', i.e, the question of the length of Markov chains and how much the temperature may be decreased to achieve convergence to the global minimum. Different arguments have been addressed in different cooling schedules. In our proposed adaptive cooling schedule some of the annealing parameters proposed by Aarts and Van Laarhoven (1985) are changed. Suppose, at the start of a particular temperature level T_t the aspiration value is given by f_a. Adjustments are made to the annealing parameters in the following way.

Length of the Markov chain (L_t^a)

As mentioned above, until now, no generally acceptable solution has been presented for the 'inner-loop criterion', which decides how many 'local move-iterations' are required at each temperature. The optimal value of this constant, which has to depend on the problem size, can not be determined in a rigorous way. Dekkers and Aarts (1991) have chosen the value to be

$$M = 10n\ , \qquad (3.68)$$

where n is the problem dimension. Rather than allowing M to depend on the problem dimension only it would be more sensible to link it with the topograph of f in some way to provide additional information to the procedure. Therefore, in our implementation, we determine the length of Markov chain in an adaptive way that depends on the starting

solution f_{x_t}, for the t-th Markov chain and the value for f_a of this Markov chain. In fact, we define

$$\begin{aligned} L_t^a &= M + \mathbf{Int}\big[MF\big] \ , \\ F &= 1 - \exp(-(f_{x_t} - f_a)) \ . \end{aligned} \tag{3.69}$$

Clearly L_t^a is equal to M when $f_{x_t} = f_a$ and tends to $2M$ as $f_{x_t} - f_a$ tends to ∞, i.e., $L_t^a \in [M, 2M]$.

Decrement of the control parameter

In many implementations of the SA algorithm, the temperature is reduced by a small factor at each iteration. This temperature scale-factor is the ratio between the old and the new temperature. Usually a constant is applied, or one that switches between at most two values (Kirkpatrick et al., 1983; Romeo et al., 1984 and White, 1984), such as e.g. 0.95 in the transition region (where the energy changes rapidly) and 0.8 or 0.85 elsewhere. In practice, however, it has to approximate unity only in the critical regions; or equivalently, when the specific heat, which is defined as the derivative of $\langle f \rangle$ with respect to T, becomes large, because this event signals a phase transition (Kirkpatrick et al., 1983) in physical annealing. In the context of optimization, therefore, when the function value drops by a significant amount at a particular temperature then the next temperature should not be reduced at all or it should only be reduced by a small amount. We assume that the function value drops by a large amount if it is less than f_a at the current Markov chain. Dekkers and Aarts (1991) find the new temperature T_{t+1} by (3.33) where the distance parameter δ determines the rate of decrease of the control parameter. Considering the different characteristics of physical annealing and global optimization we also decrease the temperature by (3.33) but our distance parameter δ satisfies

$$\delta = \delta_{\min} + (\delta_{\max} - \delta_{\min})\frac{V}{2M} \ , \tag{3.70}$$

where V is the number of iterations carried out so far in the current Markov chain and $\delta_{\min}$ and $\delta_{\max}$ are user supplied values (see later). Clearly δ varies between its maximum and minimum values, if V is relatively small then the value of δ is made smaller. Whether the aspiration criterion is satisfied or not the distance parameter δ is calculated from above rule, therefore, the greater the number of trials the greater the decrement in temperature and in the limit if $V = 2M$ then $\delta = \delta_{\max}$. We keep $\delta_{\min} = 0.05$ throughout the implementation of the ASA algorithm. However caution has to be taken when V is very small because, even if the temperature is higher, the number of acceptances will be smaller. Moreover, if the number of solutions generated at a high temperature is small and if they are close to each other then the standard deviation will be very small. Clearly if the standard deviation, $\sigma(T_t)$, in (3.33) is small then the next temperature will be reduced dramatically. Therefore, to safeguard a smooth temperature decrement we use $T_{t+1} = 0.95T_t$ when the

number of acceptances $\leq n_a$, where n_a is a small positive integer. For all implementations we use $n_a = 3$.

Final value of the control parameter (stopping criteria)

The last problem to be solved is that of finding a criterion to terminate the annealing. An easy (but perhaps not very satisfactory) way consists in stopping when the objective function has not significantly changed over a reasonable number of temperature steps. Therefore in our implementation, conditions C1 and C2 given below are used:

$$\begin{aligned} C1: &\quad \left| \frac{d\bar{f}_s(T_t)}{dT_t} \frac{T_t}{\bar{f}(T_o)} \right| < \varepsilon_s \ , \\ C2: &\quad |f_t^* - f_a| \leq \varepsilon_r \ , \end{aligned} \tag{3.71}$$

where ε_s and ε_r are small positive numbers and f_t^* is the final solution obtained in t-th Markov chain, at the end of which C1 is satisfied. Notice that the condition C1 is satisfied only if T_t is very small and no improved solution is found over a number of chains. For the termination of the ASA procedure at the end of t-th Markov chain we check condition C1 and then C2. If condition C1 is satisfied but C2 is not then the algorithm starts again with the aspiration point as the starting point of the $(t+1)$-th Markov chain and the aspiration value is found by updating the old level. The temperature T_{t+1} is now increased by setting

$$T_{t+1} = \mu T_o \ , \tag{3.72}$$

where T_o is the initial temperature and μ satisfies $0 < \mu < 1$. (This is known as re-annealing (Ingber, 1989)). If we increase the temperature by the above rule, this may, of course, introduce cycling in the algorithm, especially when the aspiration point gets stuck on a local minimizer. However, this can be overcome using

$$T_o^{i+1} = \mu T_o^i \ , \tag{3.73}$$

where T_o^i is the initial temperature of the i-th re-annealing. Obviously, in the first re-annealing, $T_o^1 = \mu T_o$. When the algorithm stops the aspiration solution is taken as the optimal solution.

The new aspiration based simulated annealing procedure is given below in pseudo-PASCAL.

Procedure Aspiration Based Simulated Annealing

begin

initialize a point x_o by random selection

calculate initial temperature T_o (see appendix 3A).

$t := 0$; (initialize the temperature change counter t)

stopcriterion:=false;

while stopcriterion=false do (outer loop begins)

begin

$k := 0$; (initialize repetition counter for inner loop)

$L_g := 0$; (initialize solution generation counter for inner loop)

calculate $f_a(x_t) := LS(x_t)$; (x_t is the initial point of the t-th Markov chain)

calculate $L_t^a := M + M \times \text{round}\,(1 - \exp(-(f_{x_t} - f_a)))$; ($M = 10n$)

aspiration-check:=false;

repeat until $k = L_t^a$ or **aspiration-check**; (inner loop begins)

begin

generate point y; (a neighbour of x; see appendix 3A)

if $f_y \leq f_a$ then

begin

aspiration-check:=true;

$L_g := L_g + 1$;

$x := y$; (make y the current point)

$f_a(x) := LS(x)$; (x is now the initial point of next Markov chain)

end

else

begin

calculate $\Delta f_{xy}^a := f_y - f_a$; (in moving from x to y)

accept $(\Delta f_{xy}^a, f_x, f_y, T_t)$;

if accept=true then

begin

$L_g := L_g + 1$; (increase the number of acceptances by one)

$x := y$; (make y the current point)

end;

end;

$k := k + 1$; (increase number of trials by one)

end; (end of inner loop)

$\bar{f}(T_t) := \frac{1}{L_g} \sum_{i=1}^{L_g} f_x$; (average of accepted solutions at T_t)

$\sigma(T_t) := \left(\frac{1}{L_g} \sum_{i=1}^{L_g} (f_x - \bar{f}(T_t))^2 \right)^{\frac{1}{2}}$; (standard deviation of accepted solutions)

$\delta := \delta_{\min} + (\delta_{\max} - \delta_{\min}) \frac{k}{2M}$;

If $L_g \leq n_a$ then $T_{t+1} = 0.95 T_t$ else

$T_{t+1} := T_t \left(1 + \frac{T_t \ln(1+\delta)}{3\sigma(T_t)} \right)^{-1}$;

if $t = 0$ then

begin

$\bar{f}_s(T_t) := \bar{f}(T_t)$

stopcriterion:=false

end else

begin

$\bar{f}_s(T_t) := 0.75 \bar{f}(T_t) + 0.25 \bar{f}(T_{t-1})$;

$\frac{d\bar{f}_s(T_t)}{dT_t} := \frac{\bar{f}_s(T_{t-1}) - \bar{f}_s(T_t)}{T_{t-1} - T_t}$;

C1:=$\left| \frac{d\bar{f}_s(T_t)}{dT_t} \frac{T_t}{f(T_o)} \right| \leq \varepsilon_s$;

C2:=$|f_t^* - f_a| < \varepsilon_r$

if C1 and not **C2** then

begin

$T_{t+1} := \mu T_o$; (see 3.73, in this section)

end

stopcriterion:=C1 and C2;

end;

$t := t + 1$; (increase the temperature change counter by one)

end; (end of outer loop)

end.

```
function accept (Δf^a_xy, f_x, f_y, T_t);
begin
    if f_y − f_x ≤ 0 then
     begin
      accept:=true;
      else
      if exp(−Δf^a_xy/T_t) > random (0,1) then
        accept:=true;
        else
        accept:=false;
     end;
end;
```

3.4.4 Numerical Results and Discussion

In our numerical comparison initial focus is made on the different parameters of ASA. Except for δ, which is allowed to vary between its maximum and minimum values, the values of the other parameters that are common to SA and ASA are kept the same as those suggested by Dekkers and Aarts (1991). Therefore for the cooling schedules of both SA and ASA we use the following common parameters: $\chi_o = 0.9$, $\varepsilon_s = 10^{-4}$ and $t_0 = 0.75$ (Alternative B) but $\delta = 0.1$ was chosen for the SA algorithm and $\varepsilon_r = 10^{-3}$ for ASA (see previous section). We found the initial temperature T_o for both ASA and SA by generating $m_o = 10n$ solutions (see Appendix 3A). For the generation mechanism, Alternative B, we use steepest descent in the early stages and limited memory BFGS (version E04DGF) from NAG (implemented for two iterations) in the later stages for the SA and ASA algorithms. If the current temperature level, T_t, falls below a certain fraction of the initial temperature, T_o, i.e. if $T_t \leq 0.05T_o$ or $T_t \leq 15$, BFGS is implemented.

Throughout a run of the ASA procedure the above mentioned BFGS routine is used for two iterations to find the successive aspiration solutions. We first examine the effect of the imposition of the condition C2. To do so we removed the condition C2 and ran the ASA procedure for $\delta_{\max} = 0.2$ until C1 was met. The results obtained are given in Table 3.1 under μ =null. The effect was that the global minimum was obtained for all functions but GP, however when the condition C2 was put back the global minimum for GP was also obtained.

The effect of μ was also examined by running the program with μ = 0.10, 0.15, 0.20, 0.25 and 0.30. For these values of μ the global minima of all test functions

were found. In Table 3.1, t represents the temperature counter, R the number of re-annealings and Q represents the accuracy of the final solution which is measured as follows: the global minimum f^* is found by conducting a local search from the vicinity of the global minimizer for each problem with the local search tolerance 10^{-10} and if the optimal solution found by SA is $\tilde{f}^*$ then $Q = \frac{f^*}{\tilde{f}^*}$ where both solutions have been taken up to 8 decimal places. Table 3.1 shows that the greater the number of re-annealings the greater the number of function evaluations. Notice that re-annealing only occur for for GP. In this table the averages (AVE) are taken over the data for which the global minima were obtained. The results for GP for different values of μ are also given since re-annealings only occur for this function.

Table 3.1

FE	cpu	t	T_o	T_f	R	Q		
697	0.03	23	335.34	7.51E-5	-	99.99%	BR	μ=(null)
1187	0.09	29	21815.39	5.58E-5	-	(*)	GP	
1746	0.20	24	10.87	5.57E-5	-	99.99%	S5	
1748	0.26	24	11.16	5.57E-5	-	100%	S7	
1748	0.26	24	11.25	5.57E-5	-	100%	S10	
920	0.19	25	5.42	1.56E-4	-	99.99%	H3	
1681	0.44	23	4.77	4.88E-5	-	100%	H6	
1423	0.23	24						AVE
2394	0.15	57	21815.39	1.74E-1	3	100%	GP	μ=0.10
2544	0.11	62	21815.39	1.67E-1	3	100%	GP	μ=0.15
2539	0.15	68	21815.39	4.05E0	2	100%	GP	μ=0.20
2820	0.10	81	21815.39	8.32E-5	4	100%	GP	μ=0.25
2832	0.33	73	21815.39	6.40E0	2	100%	GP	μ=0.30
2626	0.17	68						AVE

* Local minimum found.

In essence, the choice of μ could provide extra freedom to deal with more difficult and complicated problems, especially when the desired initial temperature for a particular problem is not known. Of course changing the value of μ does not affect the results if re-annealing does not occur. From Table 3.1 it is clear that the best result is obtained when $\mu = 0.1$ and therefore in the rest of our numerical studies we take value of μ to be 0.1.

We now investigate the effect of $\delta_{\max}$ and the results are given in Table 3.2. Since in all implementations the initial temperature remains the same therefore it has been excluded in Table 3.2.

Table 3.2

			$\delta_{\max} = 0.3$			
FE	cpu	t	R	T_f	Q	
581	0.07	18	0	7.41E0	99.99%	BR
2406	0.10	57	3	1.27E-1	100%	GP
1122	0.16	16	0	3.88E-5	99.99%	S5
1142	0.17	16	0	3.88E-5	100%	S7
1147	0.19	16	0	3.88E-5	100%	S10
764	0.15	21	0	7.23E-2	99.99%	H3
708	0.19	13	0	7.35E-3	100%	H6
8870	1.03	157				Total
			$\delta_{\max} = 0.4$			
528	0.03	18	0	1.28E-1	99.99%	BR
2086	0.13	48	4	2.76E0	100%	GP
1122	0.12	16	0	3.02E-5	100%	S5
1142	0.14	16	0	3.02E-5	100%	S7
1147	0.19	16	0	3.02E-5	99.99%	S10
750	0.13	19	0	1.17E-2	100%	H3
694	0.15	12	0	1.22E-3	99.98%	H6
7469	0.89	145				Total
			$\delta_{\max} = 0.5$			
479	0.13	16	0	5.86E-5	99.99%	BR
1982	0.08	44	4	2.69E-4	100%	GP
779	0.10	16	0	2.88E-4	99.99%	S5
783	0.12	16	0	2.88E-4	100%	S7
1147	0.18	16	0	2.51E-5	100%	S10
728	0.17	19	0	6.49E-3	99.99%	H3
694	0.17	12	0	1.15E-3	100%	H6
6592	0.95	139				Total
			$\delta_{\max} = 0.6$			
453	0.03	14	0	2.38E-5	99.99%	BR
1430	0.05	33	3	1.07E-4	100%	GP
779	0.10	16	0	2.72E-4	100%	S5
783	0.13	16	0	2.71E-4	100%	S7
783	0.13	16	0	2.71E-4	99.99%	S10
772	0.12	19	0	1.93E-2	100%	H3
602	0.13	14	0	8.92E-3	100%	H6
5602	0.69	128				Total

			$\delta_{max} = 0.7$			
453	0.08	16	0	6.30E0	99.99%	BR
1508	0.09	35	3	6.33E-2	100%	GP
789	0.08	17	0	2.10E-4	99.99%	S5
793	0.14	17	0	2.10E-4	100%	S7
783	0.13	16	0	2.57E-4	100%	S10
607	0.09	19	0	1.59E-2	99.99%	H3
602	0.14	11	0	8.50E-3	100%	H6
5535	0.75	131				Total
			$\delta_{max} = 0.8$			
383	0.10	16	0	2.39E-5	99.99%	BR
1602	0.06	35	3	5.73E-2	100%	GP
592	0.12	15	0	7.50E-1	100%	S5
596	0.09	15	0	7.50E-1	100%	S7
598	0.11	16	0	7.50E-1	100%	S10
607	0.09	19	0	1.48E-2	99.99%	H3
602	0.17	11	0	8.12E-3	100%	H6
4980	0.74	127				Total
			$\delta_{max} = 0.9$			
498	0.14	17	0	2.19E-5	99.99%	BR
1673	0.08	35	3	5.24E-2	100%	GP
592	0.08	15	0	7.25E-1	100%	S5
596	0.09	15	0	7.25E-1	100%	S7
598	0.12	16	0	7.25E-1	100%	S10
607	0.11	19	0	1.38E-2	99.99%	H3
602	0.17	11	0	7.78E-3	99.99%	H6
5166	0.77	128				Total
			$\delta_{max} = 1$			
447	0.07	12	0	5.61E0	99.99%	BR
1673	0.12	35	3	4.85E-2	100%	GP
592	0.09	15	0	7.01E-1	100%	S5
596	0.11	15	0	7.01E-1	100%	S7
598	0.11	16	0	7.01E-1	100%	S10
607	0.11	19	0	1.30E-2	99.99%	H3
754	0.18	12	0	8.26E-3	99.98%	H6
5267	0.70	124				Total

Analysis of Table 3.2 shows that in each case the global minimum is located without any difficulty. For all values of $\delta_{\max}$ re-annealing occurs for GP. The total figures indicate that the best results are obtained when $\delta_{\max} = 0.8$. Table 3.2 also shows that for all values of $\delta_{\max}$ the total cpu times are quite small and the global minima are obtained with high accuracy. So far, for the length of the Markov chain we have used, L_t^a, given by (3.69). However, it would be interesting to see how ASA performs using the length of Markov chain defined by (3.68). We have therefore run ASA with this length of Markov chain and studied the effect of $\delta_{\max}$ and the results are shown in Table 3.3. Clearly the performance is much better for every value of $\delta_{\max}$ considered (but see Chapter 5, Section 5.3). For all values of $\delta_{\max}$ re-annealing occurs for GP and for $\delta_{\max} = 0.7, 0.9, 1.0$ re-annealing also occurs for H3. Once again the best value for $\delta_{\max}$ is 0.8.

Table 3.3

			$\delta_{max} = 0.3$			
FE	cpu	t	R	T_f	Q	
435	0.07	19	0	6.65E-5	99.99%	BR
1242	0.09	51	2	2.90E0	100%	GP
719	0.08	19	0	3.57E-5	99.99%	S5
723	0.10	19	0	3.57E-5	100%	S7
723	0.12	19	0	3.57E-5	100%	S10
556	0.10	19	0	5.16E-2	99.99%	H3
694	0.17	15	0	8.59E-3	100%	H6
5092	0.73	161				Total
			$\delta_{max} = 0.4$			
347	0.04	15	0	5.84E-5	99.99%	BR
1282	0.09	51	2	1.83E0	100%	GP
719	0.07	19	0	3.03E-5	100%	S5
723	0.08	19	0	3.03E-5	100%	S7
723	0.13	19	0	3.03E-5	100%	S10
544	0.09	19	0	2.52E-2	99.99%	H3
694	0.17	15	0	8.08E-3	100%	H6
5032	0.67	157				Total
			$\delta_{max} = 0.5$			
462	0.11	16	0	5.22E-5	99.99%	BR
1016	0.13	41	3	4.89E-5	100%	GP
589	0.05	15	0	9.85E-4	100%	S5
591	0.08	15	0	9.94E-4	99.99%	S7
591	0.11	15	0	9.95E-4	100%	S10
482	0.08	17	0	2.28E-2	99.99%	H3
730	0.16	18	0	5.19E-3	100%	H6
4461	0.72	137				Total
			$\delta_{max} = 0.6$			
485	0.13	17	0	4.72E-5	99.99%	BR
947	0.08	36	3	4.78E0	100%	GP
524	0.06	14	0	4.49E-5	100%	S5
540	0.08	14	0	4.69E-4	100%	S7
560	0.09	14	0	4.64E-4	100%	S10
471	0.08	19	0	2.95E-2	99.99%	H3
808	0.26	18	0	4.93E-5	100%	H6
4335	0.78	132				Total

			$\delta_{\max} = 0.7$			
424	0.12	15	0	5.49E0	99.99%	BR
976	0.07	37	3	3.65E-2	100%	GP
524	0.08	14	0	4.14E-5	100%	S5
524	0.09	14	0	4.33E-4	100%	S7
524	0.08	14	0	4.29E-5	100%	S10
475	0.08	19	1	2.64E-2	99.99%	H3
840	0.25	23	0	4.39E-1	100%	H6
4287	0.79	136				Total
			$\delta_{\max} = 0.8$			
408	0.09	19	0	4.71E0	99.99%	BR
734	0.12	32	3	3.64E-5	100%	GP
524	0.06	14	0	3.85E-5	100%	S5
524	0.09	14	0	4.03E-5	100%	S7
524	0.08	14	0	3.99E-5	100%	S10
451	0.06	16	1	2.41E-2	99.99%	H3
558	0.14	13	0	4.16E-1	100%	H6
3723	0.64	122				Total
			$\delta_{\max} = 0.9$			
405	0.08	19	0	3.97E0	99.99%	BR
834	0.11	32	3	3.64E-5	100%	GP
630	0.06	26	0	3.85E-4	100%	S5
645	0.07	26	0	4.03E-4	100%	S7
641	0.09	26	0	3.99E-4	100%	S10
478	0.08	16	1	2.41E-2	99.99%	H3
568	0.15	18	0	4.16E-1	99.99%	H6
4201	0.64	163				Total
			$\delta_{\max} = 1$			
397	0.09	15	0	3.44E-5	99.99%	BR
834	0.03	32	3	3.09E-5	100%	GP
630	0.05	26	0	7.84E-6	100%	S5
658	0.06	27	0	7.09E-6	100%	S7
641	0.08	26	0	7.84E-6	100%	S10
458	0.08	16	1	1.99E-5	99.99%	H3
599	0.22	15	0	7.82E-5	99.99%	H6
4217	0.61	157				Total

In our final comparison of ASA and SA we therefore use the results of Table 3.3 for $\delta_{\max} = 0.8$. We note that both ASA and SA determine the initial temperatures by generating $10n$ solutions and using the Metropolis acceptance probability and generation mechanism, Alternative B. Since we use the same local search in the generation mechanism of Alternative B, obviously the initial temperatures for both algorithms are the same. In Table 3.4, the results of comparing SA and ASA are shown.

Table 3.4

		SA				
FE	cpu	t	T_o	T_f	Q	
1088	0.05	42	335.34	5.24E-5	100%	BR
1102	0.09	49	21815.39	5.24E-5	(*)	GP
1120	0.10	26	10.87	3.49E-5	100%	S5
1122	0.12	26	11.16	3.49E-5	100%	S7
1179	0.12	27	11.25	3.49E-5	100%	S10
1252	0.18	38	5.42	5.24E-5	99.99%	H3
1817	0.33	27	4.77	2.62E-5	100%	H6
1263	0.15	31				AVE
		ASA				
408	0.09	19	335.34	4.71E0	99.99%	BR
734	0.12	32	21815.39	3.64E-5	100%	GP
524	0.06	14	10.87	3.85E-5	100%	S5
524	0.09	14	11.16	4.03E-5	100%	S7
524	0.08	14	11.25	3.99E-5	100%	S10
451	0.06	16	5.42	2.41E-2	99.99%	H3
558	0.14	13	4.77	4.16E-1	100%	H6
532	0.09	17				AVE

* local minimum found

From Table 3.4 it is clear that the ASA algorithm performs much better than SA both in terms of cpu time and the number of function evaluations. Moreover, SA failed to locate the global minimum for GP. Finally, we compare our numerical results with other recent algorithms using the number of function evaluations as a basis for comparison and the results are shown in Table 3.5 where the results other than that for TMSL, MSL, ASA and SA have been taken from the references listed in Table 2.8 in the previous Chapter. The results in Table 3.5 show that the new ASA algorithm compares favourably with other algorithms except MSL and TMSL.

Table 3.5

Number of function evaluations

Method	GP	BR	S5	S7	S10	H3	H6	AVE
A	4400	1600	6500	9300	11000	2500	6000	5900
B	2500	1800	3800	4900	4400	2400	7600	3914
C	2499	1558	3649	3606	3874	2584	3447	3031
D (SA)	1102*	1088	1120	1122	1179	1252	1817	1263
E	402	346	1866	1719	1709	343	1321	1100
F	436	279	1423	1238	1213	545	1581	959
G	378	597	620	788	1160	732	807	726
H (ASA)	834	408	524	524	524	451	558	532
I	307	206	576	334	1388	166	324	471
J	53	46	98	116	100	60	127	85

* Local minima found

From the final comparison it is clear that ASA is much better than not only SA but many other methods. For higher dimensional problems and for the problems with many local minima SA-type algorithms may be necessary because the amount of data that has to be stored while running the ASA is negligible and no complete local searches are needed. Moreover, if the dimension or the number of local minima is increased, this has no effect on the amount of data stored. Therefore, in many situations the ASA algorithm will be preferable since this method performs better than the original SA algorithm. However, further research may yield yet more efficient SA algorithms.

CHAPTER 4

Controlled Random Search Algorithms (CRS)

4.1 Introduction

The stochastic methods described in earlier Chapters are often preferred to deterministic ones, because they are applicable to a wider class of functions, but they still require differentiability of the underlying function. Moreover, stochastic methods use local searches and therefore a local search procedure is needed. In practice, however, there are problems where analytical differentiation is not available and numerical differentiation may cause instability. Therefore, methods which do not use derivatives can be useful. Controlled random search (CRS) is a popular algorithm because it does not require any derivative evaluations, analytical or numerical. CRS (Price, 1977, 1983, 1987) is a 'direct search' technique and purely heuristic. A direct search method is a method which relies only on evaluating $f(x)$ at a sequence of points $x^{(i)} \in \Omega$ $(i = 1, 2, \ldots)$ and comparing values, in order to reach the optimal point x^*. Direct search methods are, in general, less efficient than methods based on the use of local searches. A wide variety of direct search methods can be found in Törn and Žilinskas (1989). Recently Palosaari et.al. (1992) have developed a direct search algorithm for global optimization which is based on alternating sequences of uniformly distributed and concentrated random searches in the variable space. The search space is reduced so that the best values of the variables will be approximately in the centre of the reduced search range. The method can also deal with constrained problems.

CRS is a kind of two phase method with few mathematical complexities and is applicable to a wide class of functions including nonsmooth and, to some extent, constrained functions. In the original version, CRS1, of CRS (Price, 1977) the search region Ω is sampled and then a simplex is formed from a subset of this sample. One of the points of the simplex is reflected in the centroid of the remaining points (as in Nelder and Mead, 1965) to obtain a new trial point and the process is then repeated until some stopping condition is met. Price enhanced the efficiency of CRS1 by a modification which he called the CRS2 algorithm (Price, 1983) and in (Price, 1987), a further modification CRS3 was given. In CRS2 a more sophisticated use is made of the simplexes in obtaining new trial points and in CRS3 a Nelder and Mead-type local search is incorporated.

In each of the CRS algorithms initially a set called the 'trial set', with a fixed number N of trial points and the corresponding function values are generated. The global phase always aims to select a new trial set. This new trial set is constructed by replacing the worst trial point in the original set by a promising one found in the local phase. The local phase

is a continuous iterative process in which a trial point, the 'new' point, is defined in terms of a configuration of $n+1$ points, n being the dimension of the problem. As the algorithm proceeds the points in the trial set tend to cluster around the global minimum. Of course, the probability that the points ultimately converge to the global minimum depends on the value of N, the complexity of the function and the way in which the trial points are chosen. However, practical numerical experience suggests that the CRS algorithms are slower than recent stochastic algorithms. To attempt to make the CRS algorithm more efficient we have devised two new algorithms, CRS4 and CRS5. The CRS4 algorithm is non-gradient but CRS5 includes a gradient-based local search procedure. Details of the two new versions are discussed in this Chapter and numerical results and comparisons are given. Some concluding observations based on the new algorithms are also given.

4.2 The CRS1, CRS2 and CRS3 Algorithms

CRS is an appropriate heuristic method for global optimization because it demonstrates a 'reasonably intelligent' pattern recognition capability. The principle features of CRS1, the first version of the algorithms, are given below.

In the search region Ω a fixed number of points N is generated from a uniform distribution. The N points and corresponding function values are stored in an array A and the highest and the lowest function values are found and denoted by f_h and f_l respectively. At each iteration a new trial point, p, is determined using a set of randomly chosen points from the N points currently held in A. The function value at p, f_p is then compared with the greatest function value f_h, if $f_p < f_h$ then h and f_h are replaced with p and f_p respectively, but if $f_p > f_h$ then the point p is discarded and a new trial point is chosen. At each iteration $n+1$ distinct points, $R_1, R_2, R_3, ..., R_{n+1}$, are chosen at random from the N ($N \gg n$) points in store and these constitute a simplex in n-space. The point R_{n+1} is arbitrarily taken as the pole of the simplex and the new trial point p is defined as the image point of the pole with respect to the centroid G, of the remaining n points so that $p = 2G - R_{n+1}$, where p, G and R_{n+1} represent position vectors. The points generated by this procedure are known as primary trial points. Without significantly reducing the effectiveness of the primary search, the efficiency of the procedure is increased by making use of secondary trial points defined by $q = \left(G + R_{n+1}\right)/2$. While the primary trial points are search oriented (p lies outside the chosen simplex), the secondary points are conducive to convergence (q lies within the simplex). At any stage in the optimization procedure if the percentage of successes, $f_p < f_h$, in the total number of trials so far is below 50% then whenever a primary trial fails, the corresponding secondary point is chosen for the next trial. In this way the cumulative success rate tends to converge on a value around 50%, maintaining a reasonable balance between search and convergence. A stepwise description of the CRS1 algorithm is as follows.

The CRS1 Algorithm

Step 1 Choose N points at random over Ω, evaluate the function values at these points and store the points and function values in an array A.

Step 2 Find in A the worst point h with function value f_h and the best point l with function value f_l; if the stopping condition is satisfied, stop. (The stopping condition is that the absolute difference $|f_h - f_l|$ should be less than a given tolerance.)

Step 3a Choose randomly $(n+1)$ distinct points $R_1, R_2, R_3, ..., R_{n+1}$ from A. Take R_{n+1} as pole and find the centroid G of the remaining n points from

$$G = \left(\sum_{i=1}^{n} R_i\right) \Big/ n$$

Find the new trial point $p = 2G - R_{n+1}$. If $p \in \Omega$ and satisfies the other constraints (if there are any) then evaluate f_p. If $f_p < f_h$, replace h in A by p and go to step 2. Else if the success rate $\geq 50\%$ then go to step 3a, otherwise determine $q = \left(G + R_{n+1}\right) \Big/ 2$. If $q \notin \Omega$ go to step 3a, else go to step 3b.

Step 3b If $f_q < f_h$ then replace h in A by q and go to step 2 else go to step 3a.

The number of different ways in which $(n+1)$ points can be chosen from N is ${}^{N}C_{n+1}$ and because the choice of pole R_{n+1} is arbitrary the total number of equiprobable next trial points associated with the configuration of N stored points is $(n+1)^{N}C_{n+1}$. For CRS1, N=$25n$ is recommended. It should be noted that the CRS1 algorithm will be much more efficient than pure random search if the probability of success at each iteration is sufficiently high. In fact this probability is expected to be much higher than pure random search because of its use of simplexes for new trial points. The disposition of the set of $(n+1)^{N}C_{n+1}$ points reflects a general trend in the current configuration and hence the random choice of any point from this set as the next trial point is likely to result in a more efficient search than a procedure based on pure random search. On the other hand the domain of the set is not restricted to the immediate neighbourhood of the configuration and this is conducive to exploration. The CRS1 procedure achieves a reasonable compromise between the conflicting requirements of thoroughness of search and convergence by defining the set of potential trial points in terms of the configuration of the N points of the current trial set. However, its efficiency can be enhanced by a modification of step 3 resulting in the CRS2 algorithm. The modification is given by

Step 3 Choose at random n distinct points $R_2, R_3, ..., R_{n+1}$ excluding l, the lowest point. Let $R_1 = l$. Determine the centroid $G = \left(\sum_{i=1}^{n} R_i\right) \Big/ n$ of the

n points $R_1, R_3, \ldots, R_n$ and compute the next trial point $p = 2G - R_{n+1}$. If $p \in \Omega$ and $f_p < f_h$ then replace h in A by p and go to step 2 else repeat step 3.

In CRS2, because R_1 is always the point l, n points are chosen randomly from $N-1$ points. Moreover, l can never be the pole of the simplex. Thus the number of trial points in CRS2 is ${}^{N-1}C_n$. A suggested value for N in this case is $10(n+1)$. In general, CRS2 is much more efficient than CRS1 in terms of both convergence and efficiency (Price 1983). Clearly the greater the value of N the more thorough the search and the greater the probability of getting a global minimum. By contrast, increasing the value of N slows down convergence, so the choice of N is a matter of experience. However larger values for N are generally advocated.

The stopping criterion for the CRS algorithms is defined in terms of the worst and the best points in the array A in such a way that when the N points are clustered around the global optimum, the algorithm stops. Typically $|f_l - f_h| < \epsilon$ is taken as the criterion. The value of ϵ depends upon the problem in hand but usually a small number is preferred. In our implementation we have taken $\epsilon = 10^{-4}$, that is, when the function values of all points in array A are identical to an accuracy of four decimal places.

The CRS3 algorithm is a modified version of CRS2 which comprises CRS2 together with a non-gradient local search procedure (LOC), selected to preserve the nongradient feature of the CRS2 algorithm. An adaptation of the Nelder and Mead simplex algorithm is used for LOC. The number of initial sample points used for CRS3 is the same as for CRS2. The $n+1$ best points in A constitute a simplex in n-space and if the function values in A are arranged in descending order, LOC operates only on the smallest one-tenth of the array A. Therefore the data required by LOC is explicitly available within the CRS2 database A. A stepwise description of LOC is given below.

The procedure LOC

Step$^{\mathrm{L}}$1 Let w be the worst point of the simplex of $(n+1)$ best points in A. Let G be the centroid of the other n points. Let s be the second worst point of the simplex. Compute three potential trial points,

$$p = 2G - w,$$
$$q = (G + w)/2,$$
$$r = 4G - 3w.$$

Step$^{\mathrm{L}}$2 If p fails to satisfy the constraints, then go to Step$^{\mathrm{L}}$4; else, evaluate the function at p. If $f_p < f_s$, then go to Step$^{\mathrm{L}}$3; else, go to Step$^{\mathrm{L}}$4.

Step^{L}3 If r fails to satisfy the constraints, then accept p as the replacement point and go Step^{L}5; else, evaluate the function at r. If $f_r < f_s$, then accept r as the replacement point and go to Step^{L}5; else, accept p as the replacement point and go to Step^{L}5.

Step^{L}4 If q fails to satisfy the constraints, then stop; else, evaluate the function value at q. If $f_q < f_s$, then accept q as the replacement point and go to Step^{L}5; else, stop.

Step^{L}5 Update the simplex by removing w and including the replacement point. Return to Step^{L}1.

The composite CRS3 algorithm begins with the CRS2 procedure and uses the ordered array A. The array A is rearranged according to descending order of function value whenever a new successful trial point is found. During the course of the CRS2 procedure if a trial point, p, is generated such that f_p, is less than $(n+1)$-th smallest function value in A then A is reordered and LOC is executed otherwise the CRS2 procedure continues. After the execution of LOC, the CRS2 procedure begins. The whole process continues until CRS2 stops. We now give a stepwise description of the CRS3 algorithm.

The CRS3 algorithm

Step 1 Run CRS2 until either it satisfies $|f_h - f_l| < \epsilon$, in which case stop, or it generates a new point, p, which falls within the bottom one-tenth of the ordered array A in which case go to step 2.

Step 2 Run LOC until it stops; then return to step 1.

Two features of CRS3 should be noted. Firstly, LOC operates only on one-tenth of A, and thus has only a slight effect on the global search performance of the CRS2 phase. Moreover, CRS2 involves the best point of A and if the best point is further improved by LOC each time it is executed then the CRS3 procedure becomes more conducive to convergence. Such effects tend to speed up the convergence of the algorithm and thus reduce, to some degree, the global search capability. If desired, it is easy to counter this effect by making the operation of LOC a probabilistic event or by not requiring that CRS2 should invariably include the best point.

Secondly, LOC can operate at any stage of the CRS3 procedure and so may be run several times producing multiple local minima and thus slowing down the procedure. On the other hand, the advantage of LOC is that it can provide the user with useful information concerning the progress of the search. In addition, CRS3 can be modified easily so as to permit the interactive user to switch LOC in or out as required allowing the use of LOC to be deferred until it is clear that the global search phase is nearing completion.

In Price (1987), it is shown that CRS3 performs better than CRS2 in terms of the number of function evaluations. To verify this, we implemented both algorithms and compared their performance on the same computer using the seven standard test problems for global optimization from Table 2.1 (see Chapter 2). For each test problem, the same series of four different random sequences was used. The performances of CRS2 and CRS3 are compared in terms of cpu time and the number of function evaluations required to achieve the stop criterion. The results are given in Table 4.1. This Table shows that the totals of the minimum and average number of function evaluations (over the series of four trials) for CRS3 are slightly less than that of CRS2. However, in terms of cpu time CRS3 is very much worse than CRS2. This is because the CRS3 procedure spends the bulk of its time in updating the ordered array A. The cpu time, however, could have been reduced significantly if we had picked out the highest point and only the $(n+1)$ best points from the set of N points in A instead of ordering the whole array whenever a successful trial point is found.

Table 4.1

	CRS2		CRS3		
	FE	cpu	FE	cpu	
Min	457	0.05	387	0.15	BR
Av	559	0.057	434	0.18	
Max	658	0.07	558	0.24	
Min	408	0.03	476	0.23	GP
Av	632	0.05	612	0.27	
Max	805	0.07	675	0.31	
Min	2817	0.47	2526	3.00	S5
Av	3152	0.52	3075	3.14	
Max	3385	0.56	3925	3.30	
Min	2721	0.50	2339	2.70	S7
Av	2891	0.52	3029	2.83	
Max	3019	0.53	3979	3.05	
Min	2690	0.54	2516	2.70	S10
Av	3186	0.62	3356	3.36	
Max	3866	0.75	4557	4.30	
Min	831	0.17	813	0.69	H3
Av	909	0.18	917	0.75	
Max	989	0.20	1000	0.80	
Min	3251	1.19	2636	5.39	H6
Av	4000	1.46	3839	6.48	
Max	4695	1.76	5457	8.70	
Min	13175	2.95	11693	14.86	Total
Av	15329	3.40	15262	17.01	
Max	17417	3.94	20151	20.70	

The main difficulty with the CRS algorithms appears to lie in the slowing down of convergence as the region of the global minimum is approached. It would therefore be sensible to incorporate additional features so as to make the convergence more rapid as soon as this region is reached. Therefore, designing an algorithm which explores the search region in the early stages and makes rapid convergence when confidence is attained would be desirable. In the next section we propose a new version (CRS4) of the CRS algorithm which incorporates a periodic feature into the CRS2 algorithm. This additional feature

helps to allow the search to be more exploratory, to some degree, in the early stages and to become more aggressive in the later stages. In fact, we have attempted to remedy the above mentioned defect of the CRS algorithm by incorporating two new ideas into the algorithm. Firstly we use a Hammersley sequence (Shaw, 1988) rather than a uniform distribution to select the initial sample points and secondly, instead of carrying out local searches, we explore the region around the present best point using a beta distribution. These modifications result in the following new CRS4 algorithm. A PASCAL subroutine for generating a two-dimensional Hammersley sequence is given in appendix 4A.

4.3 The CRS4 Algorithm

The CRS4 algorithm retains the fundamental features of CRS2 whilst attempting to eliminate some of its inefficiencies. CRS4 does not use any local search procedure as the CRS3 algorithm does, but uses CRS2 with two additional features so as to diversify the search in the early stages and to intensify it in the later stages.

One of the important features of CRS is the choice of N and the way the trial set is generated over the search region. When choosing points at which to calculate the initial function values, the most important concern is that, in the absence of any prior information about where the global minimum might be located, the whole search region is explored. One way of doing this is to choose the points to be randomly uniformly distributed throughout the region of interest and this is the preferred method of Price. In practice, the initial points are chosen from a pseudo-random sequence which closely approximates a set of independently distributed uniform random variables. One major drawback with this approach is that pseudo-random points are not evenly distributed throughout the search region. A similar problem occurs in the field of Monte Carlo numerical integration and has, to some extent, been overcome by the use of quasi-random sequences (Hammersley and Handscomb, 1964). These sequences do not approximate a set of independent realisations from a uniform distribution, tending, in general, to be much more evenly distributed. Figure 4 compares 50 points of a two-dimensional Hammersley sequence with 50 pseudo-random points. Shaw (1988) gives an overview of the quasi-random approach to numerical integration within the particular application of Bayesian statistics.

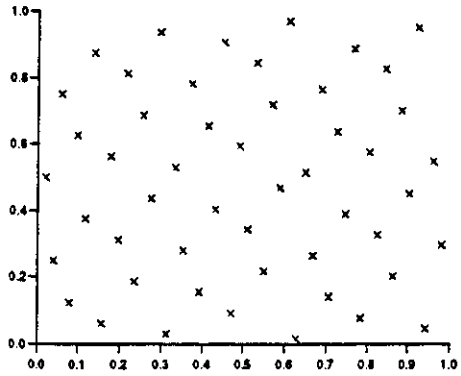


Figure 4(a) 50-point Hammersley sequence.

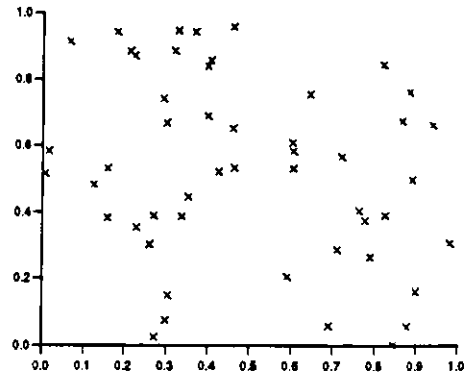


Figure 4(b) 50 pseudo-random points.

It would therefore seem sensible, when an initial set of points is required in order to investigate the objective function throughout the search region, to use a quasi-random sequence. For this purpose, we advocate using a Hammersley sequence (see section 2.3.2 in Chapter 2). These points are more evenly distributed than a set of points independently generated from a uniform distribution. We have made a comparison of using a Hammersley sequence to generate a set of points with using pseudo-random number for the same purpose in two dimensions in Figure 4 and the difference is clearly evident.

It would appear that the natural uneveness of pseudo-random points could cause rapid clustering around an arbitrary local minimum as soon as the search algorithm is initiated. Alternatively, the global minimum might lie in an area of the search region in which no initial points are generated. This is less likely to happen if we choose our initial points from a Hammersley sequence. A quasi-random sequence enables a much better initial exploration of the objective function throughout the whole search region.

In the optimization phase of the algorithm, unlike LOC in CRS3 which might force the system to a local minimum, whenever a new best point is found by the CRS2 procedure, we generate M new points close to it. The purpose of this modification is to explore the region around a new minimum by generating a small (relative to N) number of points in the area concerned. In order to achieve this we propose generating the coordinates of these M points independently from an appropriately scaled beta distribution. A helpful property of the beta distribution is that the points generated are restricted to the required search region. The beta distribution on $(0,1)$ has probability density function given by

$$d(x) = \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} x^{\alpha-1}(1-x)^{\beta-1}, \quad 0 < x < 1, \quad \alpha, \beta > 0\,, \tag{4.1}$$

with mean $\frac{\alpha}{\alpha+\beta}$ and variance $\frac{\alpha\beta}{(\alpha+\beta)^2(\alpha+\beta+1)}$. Let $\theta = \frac{\alpha}{\alpha+\beta}$ be the mean of the β-distribution then the variance $\frac{\alpha\beta}{(\alpha+\beta)^2(\alpha+\beta+1)} = \frac{\theta(1-\theta)}{A+1}$, where $A = \alpha + \beta$. Therefore $\alpha = A\theta$ and $\beta = A(1-\theta)$.

We use the algorithm of Cheng (1978) for generating the β-variates. For the i-th coordinate of the new point we choose the beta distribution with mean given by the i-th coordinate of the current best point l and standard deviation given by

$$SD = \gamma \text{DIST} \ , \tag{4.2}$$

where

$$\text{DIST} = |l_i - h_i| \ ,$$
$$h = (h_1, h_2, \ldots, h_n)$$

and γ is a user supplied parameter. The values of α and β are determined from the given mean and given standard deviation as follows: Let $K_i = \overline{x}_i - \underline{x}_i$ be the difference between the upper and lower limits on the i-th coordinate. The new coordinate point x_i is such that $x_i = \underline{x}_i + K_i r_\beta$ where r_β is generated from standard β-distribution on $(0,1)$ with the scaled mean $\theta_s = (l_i - \underline{x}_i)/K_i$. Now, $A = \left[K_i^2 \theta_s (1-\theta_s)/SD^2\right] - 1$ is found by setting $\left(\frac{SD}{K_i}\right)^2 = \frac{\theta_s(1-\theta_s)}{A+1}$. Therefore we can write $\alpha = A\theta_s$ and $\beta = A(1-\theta_s)$.

Clearly, the calculated parameters α and β for the beta distribution can take both positive and negative values but to get reasonable distributions we restrict them to values greater than or equal to one by 'clipping', i.e., if $\alpha < 1$ then we set $\alpha = 1$ and if $\beta < 1$ then $\beta = 1$. Notice that in the limiting case $\alpha = \beta = 1$, the β-distribution is a uniform distribution. Hence, if the computed standard deviation is high we will merely be generating a realisation from a uniform distribution.

Early on in the routine, when the array of points is dispersed, the standard deviations will be reasonably large and so the effect of generation from the beta distribution will be to explore a wide area around each new best point. The major gain is made, however, at later stages of the routine. Here, generating extra points from the beta distribution forces the array to form a dense cluster around the best point more quickly, once the standard deviation becomes very small. A PASCAL subroutine for generating the β-variates is given in appendix 4B.

The CRS4 algorithm

Step 1 Generate N quasi–random points from the Hammersley sequence over Ω, evaluate the corresponding function values and store the points with their function values in an array A. Find the best and worst points and function values, l, h, f_l, f_h, respectively in A. .

Step 2 Choose randomly n distinct points $R_2, R_3, ..., R_{n+1}$ excluding l and set $R_1 = l$. Determine the centroid G of the n points $R_i, i = 1, 2, \ldots, n$,

$$G = \left(\sum_{i=1}^{n} R_i\right) \Big/ n$$

and compute the next trial point $p = 2G - R_{n+1}$.

Step 3 if p is in Ω then evaluate f_p, if $f_p < f_h$ go to step 4; else, return to step 2.

Step 4 If $f_p > f_l$ replace h by p in A, find h, f_h in new A and go to step 6, otherwise replace h by p in A, find l, h, f_l, f_h in new A, go to step 5.

Step 5 Choose a new trial point $p = (x_1, x_2, \ldots, x_n)$ from an appropriately scaled β-distribution as follows. Each $x_i, i = 1, 2 \ldots, n$, is found from the β-distribution using Cheng's method (Cheng, 1978) with mean the i^{th} coordinate of the current best point $l = (l_1, l_2, \ldots, l_n)$ and standard deviation given by (4.2). Evaluate f_p. If $f_p < f_h$ replace h by p in A, find h, f_h (and l, f_l if $f_p < f_l$) in new A, go to step 6. Repeat this step M times.

Step 6 If the stopping criterion is satisfied then stop, otherwise if step 6 is reached from step 5 go to step 5 but, if not, go to step 2.

To investigate the effect of using the Hammersley sequence we have tested CRS4 against CRS4[1] which is just CRS4 with the initial N points determined pseudo-randomly. Preliminary numerical work suggested that γ=0.1 is a compromise between exploration at the initial stage and convergence in the later stages. Therefore, with this γ, we first investigate the effect of M on CRS4, by running the program several times with different values of M. The results are given in Table 4.2 and indicate that the effect of increasing M is rather random but the number of function evaluations to satisfy the stopping condition does decrease as M increases with a slight indication that a reasonable value for M is about $3n$. It is also clear from the final comparison[‡] on Table 4.2 that introducing the Hammersley sequence has the effect of making CRS4 only 6% better than CRS4[1] in terms of the number of function evaluations but about 15% better in terms of cpu time. On the other hand CRS4 is, on average, about 50% better than both CRS2 and CRS3 in terms of the number of function evaluations and about 89% and 46% better in terms of cpu time respectively and consequently the main effect of the new algorithm lies in introducing the β-distribution. CRS4 is also more robust than CRS4[1] in the sense that it never failed to find the global minima but CRS4[1] failed to do so in 8 runs.

[‡] In the final comparison the data for CRS4 and CRS4[1] is the average of averages over 7 test functions.

Table 4.2

	CRS4[1]		CRS4		CRS3		CRS2		
M	FE	cpu	FE	cpu	FE	cpu	FE	cpu	
n	394	0.05	380	0.04	434	0.18	559	0.05	BR
n+1	308	0.04	382	0.05					
n+2	430	0.05	279	0.04					
n+3	478	0.05	315	0.03					
3×n	537	0.06	316	0.05					
3×n+1	268	0.03	409	0.06					
Total	2415	0.28	2081	0.27					
AVE	403	0.046	347	0.045					
n	429	0.05	530	0.05	612	0.27	632	0.05	GP
n+1	351	0.04	532	0.06					
n+2	569	0.06	436	0.05					
n+3	357	0.04	398	0.04					
3×n	385	0.05	397	0.04					
3×n+1	339	0.04	358	0.05					
Total	2430	0.28	2651	0.29					
AVE	405	0.046	442	0.048					
3	2088	0.41	2133	0.39	3075	3.14	3152	0.52	S5
n	1696*	0.32	1864	0.35					
n+1	1527	0.30	1529	0.30					
n+2	1132*	0.24	1686	0.32					
n+3	1591	0.32	1183	0.25					
2×n	1482	0.29	1423	0.29					
3×n	1182*	0.25	897	0.24					
3×n+1	1054	0.25	1212	0.28					
Total	7742	1.57	11927	2.42					
AVE	1548	0.314	1491	0.302					
3	1835	0.36	1876	0.37	3029	2.83	2891	0.52	S7
n	1637	0.34	2034	0.41					
n+1	1066*	0.23	1683	0.35					
n+2	1486	0.31	1735	0.35					
n+3	1257	0.27	1498	0.30					
2×n	1320	0.28	1238	0.26					
3×n	911*	0.20	1295	0.31					
3×n+1	1080*	0.24	1031	0.24					
Total	7535	1.56	12390	2.59					
AVE	1507	0.312	1549	0.323					
3	1920	0.41	2191	0.44	3356	3.36	3186	0.62	S10
n	1993	0.42	1769	0.38					
n+1	1615	0.35	1350	0.31					
n+2	1493	0.33	1384	0.31					
n+3	1608	0.34	993	0.24					
2×n	1240	0.29	1213	0.29					
3×n	1555	0.37	950	0.26					
3×n+1	1052	0.27	1265	0.32					
Total	12476	2.78	11115	2.55					
AVE	1560	0.347	1389	0.318					

	CRS4[1]		CRS4		CRS3		CRS2		
M	FE	cpu	FE	cpu	FE	cpu	FE	cpu	
n	565	0.13	646	0.14	917	0.75	909	0.18	H3
n+1	556	0.13	634	0.14					
n+2	593	0.13	565	0.10					
2×n	517	0.13	545	0.12					
2n+1	524	0.12	441	0.11					
3×n	406	0.10	471	0.10					
3×n+1	439	0.10	370	0.09					
Total	3600	0.84	3672	0.80					
AVE	514	0.120	525	0.114					
3	3418	1.36	3295	0.32	3839	6.48	4000	1.46	H6
5	2378	0.97	2071	0.84					
n	2618	1.10	2116	0.83					
n+1	2917	1.17	2203	0.92					
n+2	1697	0.73	1637	0.70					
2×n	1399*	0.29	1581	0.67					
3×n	1100	0.50	1395	0.57					
3×n+1	1346	0.60	1103	0.51					
Total	15474	6.43	15401	5.36					
AVE	2211	0.918	1925	0.670					

FC

	FE	cpu
CRS4[1]	1164	0.30
CRS4	1095	0.26
CRS3	2180	2.43
CRS2	2190	0.48

* Local minimum found; FC : Final Comparison

We therefore continued our computational experiments with the CRS4 algorithm with especial attention given to γ and M. This further investigation is based on the following criteria, diversification of search in the early stages and progressive intensification of the search as the points move towards the global minimum. To fulfill these demands, we consider M as a variable whose value increases along with the improvement of the present best point. In other words, initially M is set to zero and if the CRS4 algorithm finds a trial point with function value better than the present best one stored in the current trial set, M is increased by one. M trial points are generated then from the beta distribution and so on until the algorithm stops. Numerical investigation is carried out with these variable values of M and the results are given in Table 4.3 in the column under CRS4. The robustness of the algorithm is also examined by choosing a set of values for γ. To see the effect on the results due to the changes in M, we also ran the CRS4 algorithm with only a fraction of M. For instance we ran the CRS4 algorithm generating $0.5M$ and $0.75M$ (rounded down) points from β-distribution and the results are given in columns under CRS4† and CRS4‡ respectively in Table 4.3. In each case the global minimum was located without difficulty except that CRS4 failed to locate the global minimum for S7 when $\gamma = 0.15$. Table 4.3

shows that varying γ has a considerable effect, with FE and cpu decreasing, in general, with γ.

Table 4.3

	CRS4		CRS4†		CRS4‡		CRS3		CRS2		
γ	FE	cpu	FE	cpu	FE	cpu	FE	cpu	FE	cpu	
0.15	342	0.05	466	0.05	458	0.06	434	0.18	559	0.05	BR
0.12	389	0.04	434	0.05	330	0.04					
0.10	352	0.04	386	0.04	327	0.04					
0.07	265	0.04	323	0.04	498	0.05					
0.05	255	0.03	410	0.05	381	0.04					
0.15	426	0.04	396	0.05	434	0.05	612	0.27	632	0.05	GP
0.12	510	0.05	445	0.05	384	0.03					
0.10	406	0.05	418	0.05	422	0.05					
0.07	461	0.05	405	0.05	494	0.05					
0.05	324	0.03	476	0.05	348	0.04					
0.15	1780	0.36	2023	0.38	2038	0.40	3075	3.14	3152	0.52	S5
0.12	1802	0.35	1974	0.36	1770	0.36					
0.10	1040	0.22	2015	0.38	1859	0.38					
0.07	1226	0.25	1886	0.35	1479	0.32					
0.05	1234	0.26	1671	0.32	1338	0.26					
0.15	2884*	0.52	1631	0.33	1468	0.29	329	2.83	2891	0.52	S7
0.12	1611	0.32	1881	0.37	1637	0.36					
0.10	1732	0.36	1593	0.33	1212	0.27					
0.07	1224	0.27	1588	0.33	1063	0.25					
0.05	1631	0.33	1368	0.29	1065	0.23					
0.15	1424	0.31	1230	0.30	1721	0.37	3356	3.36	3186	0.62	S10
0.12	2037	0.44	1965	0.41	1718	0.38					
0.10	1053	0.27	1542	0.37	1441	0.39					
0.07	1075	0.26	1582	0.35	1596	0.36					
0.05	1147	0.27	1834	0.41	1287	0.32					
0.15	601	0.13	695	0.15	613	0.15	917	0.75	909	0.18	H3
0.12	655	0.14	642	0.14	597	0.14					
0.10	614	0.13	768	0.16	767	0.16					
0.07	706	0.15	719	0.15	590	0.13					
0.05	592	0.15	449	0.11	498	0.12					
0.15	3457	1.40	2269	0.90	2077	0.92	3839	6.48	4000	1.46	H6
0.12	1770	0.78	1914	0.78	2042	0.87					
0.10	2376	0.98	2000	0.79	1627	0.72					
0.07	2357	1.01	2142	0.84	1729	0.76					
0.05	2260	0.97	1772	0.74	1664	0.73					
	1150	0.30	1237	0.30	1113	0.28					AVE

* Local minimum found

We summarise the results of Table 4.3 in Table 4.4 by taking the average number of function evaluations and cpu time on all test functions for different values of γ.

Effect of γ and variable M

Table 4.4

	CRS4		CRS4†		CRS4‡		CRS3	CRS2
			FE					
γ	Total	AVE	Total	AVE	Total	AVE	AVE	AVE
0.15	8030	1338*	8710	1224	8809	1258	2180	2190
0.12	8774	1253	9255	1322	8478	1211		
0.10	7573	1082	8722	1276	7655	1094		
0.07	7314	1045	8645	1235	7449	1064		
0.05	7443	1063	7980	1140	6581	940		
			cpu					
γ	Total	AVE	Total	AVE	Total	AVE	AVE	AVE
0.15	2.29	0.38*	2.16	0.30	2.24	0.32	2.43	0.48
0.12	2.12	0.30	2.16	0.30	2.18	0.31		
0.10	2.05	0.29	2.12	0.30	2.01	0.28		
0.07	2.03	0.29	2.11	0.30	1.92	0.27		
0.05	2.04	0.29	1.97	0.28	1.74	0.24		

* Average over 6 functions

In this Table the ratio of the averages of FE and cpu for the best value of γ to the worst are 0.78 and 0.76; 0.86 and 0.91; 0.74 and 0.75 for CRS4, CRS4† and CRS4‡ respectively. The best result is for CRS4‡ when $\gamma = 0.05$ and this is better (about 15%) than the average result for CRS4 with M fixed and $\gamma = 0.1$.

In summary we can readily conclude that the introduction of the β-distribution has a greater effect than the Hammersley sequence but the combination of the two changes greatly improves the original CRS algorithm.

The effect of the Hammersley sequence has also been investigated on CRS2 and CRS3 and the results are compared in Table 4.5. The results in the columns under CRS2^H and CRS3^H are the results of CRS2 and CRS3 respectively using the Hammersley sequence. From the average results it is clear that the Hammersley sequence on its own does have some slight effect in improving CRS3 but on the contrary it has worsened CRS2.

Table 4.5

CRS3^H		CRS3		CRS2^H		CRS2		
FE	cpu	FE	cpu	FE	cpu	FE	cpu	
487	0.20	434	0.18	400	0.06	559	0.05	BR
705	0.29	612	0.27	550*	0.07	632	0.05	GP
3506	3.01	3075	3.14	2985	0.51	3152	0.52	S5
4181	3.72	3029	2.83	3032	0.56	2891	0.52	S7
2559	3.17	3356	3.36	2876	0.57	3186	0.62	S10
902	0.67	917	0.75	1039	0.20	909	0.18	H3
2780	5.61	3839	6.48	3526	1.31	4000	1.46	H6
2160	2.38	2180	2.43	2309	0.53	2190	0.48	AVE

* Local minimum found

So far we have shown that the CRS4 algorithm gives a significant improvement over the other CRS algorithms. Such an improvement of non-gradient global optimization algorithms should be very useful because of the simplicity and ease of use of these methods. There seems also to be an important need to devise an algorithm which incorporate a gradient based local search instead of the Nelder and Mead simplex algorithm as in CRS3.

4.4 The CRS5 Algorithm

Although direct search-type algorithms for optimization have a role to play in certain situations where derivatives are not available it is quite clear that the use of gradient type techniques are preferable, in general, even when the gradients have to be computed numerically. Consequently we have also devised an algorithm, CRS5, that uses a Hammersley sequence to determine the initial sample but instead of a simplex-type local search as in CRS3 or the use of the β-distribution as in CRS4, may (with a pre-set probability) use a gradient based local search when a new best point is detected. However, as in CRS3, this local search procedure has been implemented within the framework of the CRS2 algorithm in a suitable manner. The CRS3 algorithm operates LOC whenever CRS2 finds a trial point within the best $(n+1)$ points of the array A. CRS5 has been designed so that a local search is only started with a certain probability, whenever CRS2 locates a new best point. Therefore, the frequency of starting a local search in CRS5 is much less than that of LOC in CRS3 and therefore the global search capability of CRS5 is higher than that of CRS3. Moreover, LOC needs more function evaluations to improve the best point in CRS2 than a gradient based local search usually does. We now give a stepwise description of the CRS5 algorithm.

Algorithm CRS5[‡]

Step 1 Generate N points from the Hammersley sequence. Evaluate the objective function f at each point $x^{(i)}$, $i = 1, \ldots, N$, and store points and corresponding function values in an array A.

Step 2 Find h, f_h,l, $f_l \in A$. If stopping condition $|f_l - f_h| < \epsilon$ is satisfied then go to step 6, otherwise go to step 3.

Step 3 Choose randomly n distinct points $R_2, R_3, \ldots, R_{n+1}$ from A but excluding l, the point with lowest function value and set $R_1 = l$. Compute

$$p = 2G - R_{n+1}$$

where G is the centroid of $R_1, \ldots, R_n$. If $p \in \Omega$ (and satisfies any other constraints present) evaluate f_p, if $f_p > f_h$ then repeat step 3, else go to step 4.

Step 4 If $f_p > f_l$ replace h by p and go to step 2, otherwise go to step 5.

Step 5 Start a local search from p if $w < t$, where w is a random number in $(0, 1)$ and t is a preset number in $(0, 1]$, replace h by the point resulting

[‡] Steps 1, 2 and 3 are similar to those of the CRS2 algorithm

from the local search and go to step 2. If $w \geq t$ replace h by p and go to step 2.

Step 6 Carry out a final local search from the best point and then stop. The minimum from the final local search is taken as the global solution.

The local search in step 5 is carried out for a small fixed number of iterations (normally 1 or 2) and the local search in step 6 continues until a user supplied accuracy is achieved. The tolerance for the latter local search is 10^{-10}. Clearly a higher number of iterations in step 5 is conducive to convergence and therefore reduces the exploration phase of the algorithm. In step 5 we use the limited memory BFGS algorithm from the NAG Library (E04DGF) but in step 6 we use a different local search from NAG (E04UCF) for convenience in programming.

The CRS5 algorithm will maintain a reasonable balance between the exploration of the search region and efficiency if t is chosen properly. If in step 2 of the CRS5 algorithm ϵ is small enough this is an indication that the N points have already formed a cluster near the best point. Different aspects of the CRS5 algorithm have been examined, especially the value of ϵ in step 2, the local search in step 5 and the value of t. Results are given in Table 4.6 (ITR in Tables 4.6 and 4.7 represents the number of iterations carried out by the local search in step 5). We also ran the CRS5 algorithm with the initial sample points generated from a pseudo-random sequence and the results are given in Table 4.7.

Table 4.6

		ITR=1	$\epsilon = 1.5 \times 10^{-2}$					
t =1.0		0.75		0.50		0.25		
FE	cpu	FE	cpu	FE	cpu	FE	cpu	
382	0.04	445	0.07	385	0.06	385	0.06	BR
480	0.08	555	0.10	442	0.08	411	0.06	GP
2009	0.43	2009	0.41	2145	0.39	2486	0.48	S5
2463	0.48	2670	0.51	2027	0.43	2007	0.40	S7
2173	0.60	2230	0.55	2070	0.60	1912	0.49	S10
526	0.14	488	0.14	488	0.13	549	0.17	H3
1781	0.76	1768	0.73	1748	0.77	1950	0.82	H6
1402	0.36	1452	0.36	1329	0.35	1386	0.35	AVE
		ITR=2	$\epsilon = 1.5 \times 10^{-2}$					
384	0.05	345	0.05	442	0.07	442	0.07	BR
390	0.06	391	0.06	363	0.06	422	0.06	GP
2132	0.41	2005	0.35	1885*	0.40	2702	0.47	S5
2554	0.58	2253	0.51	2237	0.56	2231	0.46	S7
1986	0.48	1986	0.45	1986	0.49	2028	0.48	S10
466	0.09	533	0.12	487	0.15	565	0.12	H3
2786	1.17	2786	1.12	2786	1.15	1947	0.85	H6
1528	0.40	1471	0.38	1384	0.41	1477	0.36	AVE
		ITR=1	$\epsilon = 10^{-1}$					
305	0.07	361	0.06	346	0.07	346	0.07	BR
448	0.06	467	0.05	402	0.05	396	0.07	GP
1694	0.33	1694	0.37	1866	0.35	2195	0.40	S5
2155	0.49	2321	0.50	1719	0.39	1652	0.34	S7
1858	0.44	1845	0.40	1709	0.40	1652	0.39	S10
401	0.11	343	0.12	343	0.12	352	0.13	H3
1281	0.61	1281	0.62	1321	0.57	1345	0.60	H6
1163	0.30	1187	0.30	1101	0.28	1134	0.29	AVE
		ITR=2	$\epsilon = 10^{-1}$					
320	0.05	312	0.05	399	0.06	399	0.06	BR
300	0.04	348	0.04	349	0.04	381	0.06	GP
1804	0.38	1744	0.34	1609*	0.34	2286	0.43	S5
2274	0.49	1931	0.39	1862	0.40	2019	0.44	S7
1674	0.39	1814	0.36	1683	0.38	1704	0.45	S10
342	0.10	387	0.12	374	0.13	428	0.12	H3
2143	0.97	2143	0.93	2143	0.91	1342	0.57	H6
1265	0.35	1241	0.32	1135	0.32	1223	0.30	AVE

* Local minimum found.

Table 4.7

		ITR=1	$\epsilon = 1.5 \times 10^{-2}$					
t =1.0		0.75		0.50		0.25		
FE	cpu	FE	cpu	FE	cpu	FE	cpu	
242	0.04	238	0.04	240	0.05	296	0.05	BR
521	0.06	521	0.0.08	503	0.08	414	0.07	GP
2273*	0.48	2273*	0.48	1953	0.40	2076	0.42	S5
1991	0.45	1991	0.39	1991	0.42	2092	0.44	S7
1984	0.56	1834	0.48	1870	0.47	2025	0.46	S10
552	0.19	449	0.13	449	0.13	486	0.15	H3
1562	0.64	1796	0.74	1743	0.78	1948	0.80	H6
1142	0.32	1138	0.31	1250	0.33	1334	0.34	AVE
		ITR=2	$\epsilon = 1.5 \times 10^{-2}$					
329	0.05	299	0.05	299	0.05	316	0.06	BR
463	0.05	464	0.07	489	0.07	426	0.06	GP
2536	0.51	2291	0.48	2339	0.50	2027	0.40	S5
2306	0.45	3420	0.75	2211	0.42	2086	0.45	S7
2184	0.52	2136	0.54	2136	0.52	2168	0.58	S10
452	0.15	484	0.17	485	0.16	424	0.12	H3
1699	0.73	1699	0.73	1834	0.74	1834	0.71	H6
1424	0.35	1542	0.40	1399	0.35	1326	0.34	AVE
		ITR=1	$\epsilon = 10^{-1}$					
215	0.04	211	0.05	211	0.04	244	0.05	BR
462	0.07	439	0.05	453	0.06	355	0.06	GP
2045*	0.41	2045*	0.42	1713	0.38	1838	0.36	S5
1728	0.40	1728	0.34	1730	0.37	1760	0.38	S7
1661	0.43	1580	0.45	1596	0.44	1730	0.43	S10
409	0.17	350	0.10	444	0.18	345	0.12	H3
1163	0.47	1233	0.53	1274	0.54	1398	0.59	H6
940	0.26	924	0.25	1060	0.29	1096	0.28	AVE
		ITR=2	$\epsilon = 10^{-1}$					
247	0.05	256	0.04	256	0.04	261	0.04	BR
433	0.06	430	0.05	380	0.06	360	0.05	GP
2119	0.44	1963	0.39	1975	0.35	1827	0.34	S5
1970	0.40	1970	0.41	1899	0.40	1722	0.39	S7
1848	0.40	1848	0.48	1848	0.50	1880	0.43	S10
368	0.10	358	0.11	365	0.09	312	0.12	H3
1199	0.55	1199	0.54	1329	0.59	1329	0.58	H6
1169	0.29	1146	0.29	1150	0.29	1099	0.28	AVE

* Local minimum found.

Averaging the results in Tables 4.6 and 4.7 shows that for CRS5 the introduction of the Hammersley sequence makes the algorithm about 8% and about 6% worse with respect to FE and cpu time respectively than when pseudo-random points are used. The Tables also indicate that the best results were obtained for $t = 0.50$ with a single iteration in step 5 when $\epsilon = 10^{-1}$. It is also clear that if we increase the number of iterations (ITR) of local search in step 5, the number of function evaluations increases accordingly. Therefore, in the early stages of the CRS5 algorithm a single step steepest descent local search will be preferable. The stopping condition in step 2 is an important factor where a small ϵ may cause unnecessary local searches to be carried out (leading to more function evaluations) in which case the final local search may not be necessary but a bigger ϵ may weaken the global search capability. Hence a tradeoff has to be made. The proper choice of ϵ therefore remains an open question. In any case if the best point is close enough to the global minimum then it is likely that the final local search in step 6 will find the global minimum with a prescribed accuracy.

We now compare CRS4 and CRS5 in Table 4.8. The data for CRS5 have been taken from Table 4.7 for $\epsilon = 10^{-1}, t = 0.50$ and ITR=1.

Table 4.8

	CRS4		CRS5	
Test Problems	Function Eval.	cpu Time	Function Eval.	cpu Time
BR	416	0.04	211	0.04
GP	397	0.05	453	0.06
S5	897	0.24	1713	0.38
S7	1295	0.23	1730	0.37
S10	950	0.26	1596	0.44
H3	471	0.10	444	0.18
H6	1395	0.57	1274	0.54
Total	5721	1.57	7421	2.01

The results in Table 4.8 clearly show that CRS4 is superior to CRS5 both in terms of number of function evaluations and cpu time.

Finally, we compared our new algorithms with other recent algorithms using the number of function evaluations as a basis and the results are shown in Table 4.9. In this Table the methods representing A, B, C and G are noted in Table 2.3 in Chapter 2 and the results for these methods have been taken from the references listed in the same Table. The results for the rest of the methods including CRS4 and CRS5 (Ali and Storey, 1995) have been found by implementing them on our own computer. From Table 4.9 it is clear

that the new algorithms are superior to many of the recent stochastic methods. They could also be preferable in many practical applications because they are simple and easily programmable.

Table 4.9

Number of function evaluations

Method	GP	BR	S5	S7	S10	H3	H6	AVE
A	4400	1600	6500	9300	11000	2500	6000	5900
B	2500	1800	3800	4900	4400	2400	7600	3914
C	2499	1558	3649	3606	3874	2584	3447	3031
CRS2	632	559	3152	2891	3186	909	4000	2190
CRS3	612	434	3075	3029	3356	917	3839	2180
D (SA)	1102*	1088	1120	1122	1179	1252	1817	1263
CRS5	211	453	1713	1730	1596	444	1274	1060
CRS4	316	397	897	1295	950	471	1395	817
G	378	597	620	788	1160	732	807	726
H (ASA)	834	408	524	524	524	451	558	532
I (MSL)	307	206	576	334	1388	166	324	471
J (TMSL)	53	46	98	116	100	60	127	86

* Local minima found

Since the global minima for the test functions, except BR and the Hartmann family, are known analytically numerical experiments were also carried out using the difference between the current best function value and the (exact) global minima in the stopping condition. Therefore, $|f^* - f_c| < \epsilon$ was used as the stopping condition where f_c is the current best function value. The global minima (f^*) for BR, H3 and H6 are found by performing a local search from the vicinity of the global minimizer with local search tolerance 10^{-10}. The test therefore indicates up to how many decimal places the solution can be obtained by the CRS algorithms. Since step 6 in the CRS5 algorithm is used as a refinement step, inevitably in our implementations solutions are obtained with a required accuracy and therefore we treat CRS5 slightly differently. For CRS5 we use the above mentioned stopping condition in step 2 as the only stopping condition. The results are summarized in Table 4.10.

Table 4.10

CRS2							
10^{-2}	10^{-3}	10^{-4}	10^{-5}	10^{-6}	10^{-7}	10^{-8}	$\leftarrow \epsilon$
√	√	√	√	√	√	×	BR
√	√	√	√	√	√	√	GP
√	×	×	×	×	×	×	S5
√	×	×	×	×	×	×	S7
√	×	×	×	×	×	×	S10
√	√	√	√	√	√	√	H3
√	√	√	√	√	√	×	H6
CRS3							
√	√	√	√	√	√	√	BR
√	√	√	√	√	√	√	GP
√	×	×	×	×	×	×	S5
√	×	×	×	×	×	×	S7
√	×	×	×	×	×	×	S10
√	√	√	√	√	√	√	H3
√	√	√	√	√	√	×	H6
CRS4							
√	√	√	√	√	√	√	BR
√	√	√	√	√	√	√	GP
√	×	×	×	×	×	×	S5
√	×	×	×	×	×	×	S7
√	×	×	×	×	×	×	S10
√	√	√	√	√	√	√	H3
√	√	√	√	√	√	√	H6
CRS5							
√	√	√	√	×	×	×	BR
√	√	√	×	×	×	×	GP
√	×	×	×	×	×	×	S5
√	×	×	×	×	×	×	S7
√	×	×	×	×	×	×	S10
√	√	√	√	√	√	√	H3
√	√	√	√	√	√	×	H6

× Result not obtained; √ Result obtained

From Table 4.10 it is clear that the CRS algorithms are not reliable as far as accuracy is concerned. The results are worst for the Shekel family for which all the CRS algorithms can find solutions only up to an accuracy of two decimal places but for the other test problem results are quite reasonable. No doubt, the non-gradient CRS algorithms are easy to implement but if these algorithms are used for problems for which a high accuracy is needed they may fail. The reason is that they stop when all points form a dense cluster rather than when some goal in terms of accuracy is attained. However, CRS5 may overcome this because its final solution depends on local search and therefore if a point is found in the region of attraction of the global minimum then the global minimum can be obtained up to a required accuracy.

From Table 4.10 it is evident that high accuracy can not be obtained if the final local search is removed from CRS5. Therefore the question can be raised as to whether CRS4 is better than CRS5 in terms of number of function evaluations and cpu time if a final local search is incorporated in CRS4 the same way as in CRS5. We therefore ran CRS4 for $\gamma = 0.1$ and $M = 3n$ with the stopping condition $|f_h - f_l| < 10^{-1}$ and with the final solution refined by the same local search with the same tolerance as was used for CRS5. The results are compared with the best results for CRS5 in Table 4.11.

Table 4.11

CRS4		CRS5		
FE	cpu	FE	cpu	
205	0.05	211	0.04	BR
263	0.06	453	0.06	GP
524	0.13	1713	0.38	S5
821	0.23	1730	0.37	S7
692	0.22	1596	0.44	S10
239	0.12	444	0.18	H3
923	0.50	1274	0.54	H6
3667	1.31	7421	2.01	Total

This Table shows that the CRS4 algorithm can be improved even further if a final local search is incorporated. The benefits are twofold. Firstly the number of function evaluations and cpu time are lessened and secondly the final solution is obtained with a prescribed accuracy.

The CRS5 algorithm is designed so that a few steps of local search (not the final local search) are carried out with a preset probability. Since CRS5 differs from CRS3 only by the nature of the local search therefore we ran CRS3 with LOC executed with a certain probability. As in CRS5 we preset $t \in (0, 1]$ and generate a random number w. If $w \leq t$ then LOC is executed, otherwise not. Notice that $t = 1$ gives the original CRS3 algorithm. The results are given in Table 4.12.

Table 4.12

t	1		0.75		0.50		0.25	
Test Problems	Function Eval.	cpu Time	Function Eval.	cpu Time	Function Eval.	cpu Time	Function Eval.	cpu Time
BR	558	0.24	528	0.23	410	0.19	492	0.20
GP	675	0.28	438	0.17	434	0.16	640	0.30
S5	3925	3.24	2864	3.32	3168	3.32	3149	3.23
S7	3979	3.06	2376	2.68	2326	2.57	2562	2.76
S10	4557	4.31	4388	3.87	3718	3.75	2475*	2.87
H3	927	0.80	850	0.65	850	0.64	921	0.65
H6	2636	5.64	5750	5.82	2913	5.20	5815	10.10
AVE	2465	2.51	2456	2.39	1974	2.26	2263	2.87

* local minimum found

The average results in above Table shows that the performance of CRS3 is improved by executing LOC with a preset probability. The best result is obtained if LOC is executed with probability $\frac{1}{2}$. However, in comparison with the results of CRS5 in Table 4.11 it is clear that CRS5 is much better than CRS3. Therefore incorporation of a gradient based local search procedure has improved the CRS algorithm.

4.5 Conclusion

Modifications have been suggested to the original controlled random search method of Price and the resulting algorithms have been shown to be superior to the original algorithm. Both of the new algorithms, CRS4 and CRS5, are improved if a final accurate local search is introduced. The effect of the modifications is to make the CRS approach much more competitive with the other algorithms tested, only MSL and TMSL having an overall superiority. This combined with the heuristic direct search nature of the new algorithms seem to suggest they have a useful part to play in global optimization.

CHAPTER 5

Application of Global Optimization to some Problems in Material Science

5.1 Introduction

In this Chapter we describe the application of global optimization to two problems in materials science. The first problem is to calculate the minimum energy of small clusters of particles which interact through well-defined many-body interaction potentials. The second problem is the fitting of such interaction potentials to bulk crystal data.

The first problem arose as a result of some work involving molecular dynamics (MD) to simulate the ejection of particles after a solid surface had been bombarded with energetic ions. Such a bombardment can be used together with mass spectrometry of the ejected particles to determine the composition of the surface; the secondary ion mass spectrometry (SIMS) technique. The mass spectra consist of single particles and ejected clusters. In any simulation of this process it is necessary that the interaction potentials give roughly the correct energetics and structure of these clusters.

The second problem has arisen as a result of the success of many-body, semi-empirical, potential functions in modelling near equilibrium, bulk crystal properties. A parameterisation of the potential is assumed which is based on physical considerations. The free parameters are then chosen using a least squares fit, to a large number of crystal properties, by global optimization. This has been achieved for face-centered cubic and diamond lattice materials (Daw and Baskes, 1983, 1984; Tersoff, 1988, 1988a) but little work has been done on body centered cubic (bcc) materials. The approach adopted here fits the bcc crystal structure, as the preferred minimum energy configuration for tungsten, and also fits the dimer energetics and the elastic properties of crystalline tungsten.

5.2 Investigation of small Cluster Energetics by Global Optimization

Empirical many-body potentials are becoming an increasingly important means of investigating high and low energy processes in both metals and semiconductors. For example, the 'embedded atom' potential has been used to investigate the molecular dynamics simulation (Garrison et al., 1988) of $Rh\{111\}$ (Rhodium). While such potentials may be less accurate than those determined using *ab initio* methods, because of their relative simplicity, they are invaluable for use with MD simulations.

Many of these empirical many-body potentials have been designed with the bulk material properties in mind. For example, the Tersoff potentials (Tersoff 1988, 1988a) for *Si* (silicon), accurately fit the diamond lattice structure as the minimum potential configuration with the correct binding energy and lattice spacing. They also model the elastic properties with reasonable accuracy. However, the fitting procedure for the potential ignored the small cluster energetics. More recently developed potentials (Brenner 1990 and Smith 1992) have included small cluster properties in the fitting process and indeed in one case (Brenner, 1990), which develops a many-body *C-H* (carbon-hydrogen) potential, the energetics of a large number of small clusters have been accurately reproduced.

Our purpose is to calculate the minimum energy of small clusters predicted by 'Tersoff' potentials for *Si* and 'Tersoff-like' potentials (Smith, 1992) for *As* (arsenic). The properties of small clusters predicted by these potentials are discussed in Ali and Smith (1993). When using MD simulations on *Si*, the resulting clusters must have the correct energetics or the calculated proportion of dimers and trimers energetics will be incorrect. It might seem natural to use MD to try to calculate these structures but this is not easy because the form of the potentials gives rise to a large number of local minima. We therefore use global optimisation algorithms to find the minimum energy. We have chosen up to six particles. We use the global optimization algorithms described in the earlier Chapters to see if they give comparable results and compare the performance of each method with respect to the number of function evaluations and cpu time.

Problem Formulation

Before we define the underlying function to be optimized we need to describe the following terms explicitly.

The binding energy in the Tersoff formulation (see, Tersoff 1988a) is written as a sum over atomic sites in the form

$$E_i = \frac{1}{2}\sum_{j\neq i} f_c(r_{ij})\big(V_R(r_{ij}) - B_{ij}V_A(r_{ij})\big)\ , \quad \forall i \tag{5.1}$$

where r_{ij} is the distance between atoms i and j, V_R is a repulsive term, V_A is an attractive term, $f_c(r_{ij})$ is a switching function and B_{ij} is a many-body term that depends on the positions of atoms i and j and the neighbours of atom i. More details of each of these quantities can be found in (Tersoff 1988; Brenner 1990; Smith 1992 and Ali and Smith 1993). The term B_{ij} is given by

$$B_{ij} = (1 + \gamma^{n_1}\xi_{ij}^{n_1})^{-1/2n_1} \tag{5.2}$$

where n_1 and γ are known fitted parameters (Tersoff, 1988a). The term ξ_{ij} for atoms i and j is given by

$$\xi_{ij} = \sum_{k \neq i,j} f_c(r_{ik}) g(\theta_{ijk}) \tag{5.3}$$

where θ_{ijk} is the bond angle between bonds ij and ik and g is given by

$$g(\theta_{ijk}) = 1 + c^2/d^2 - c^2/[d^2 + (h - \cos\theta_{ijk})^2] \; . \tag{5.4}$$

The quantities c, d and h which appear in (5.4) are also known fitted parameters. The terms $V_R(r_{ij})$ and $V_A(r_{ij})$ are given by

$$\begin{aligned} V_R(r_{ij}) &= A \exp[-\lambda_1 r_{ij}] \\ V_A(r_{ij}) &= B \exp[-\lambda_2 r_{ij}] \end{aligned} \tag{5.5}$$

where A, B, λ_1 and λ_2 are given fitted constants. The switching function $f_c(r_{ij})$ is given by

$$f_c(r_{ij}) = \begin{cases} 1 \; , & r_{ij} \leq R - D \\ \frac{1}{2} - \frac{1}{2}\sin[\pi(r_{ij} - R)/(2D)] \; , & R - D \leq r_{ij} \leq R + D \\ 0 \; , & r_{ij} \geq R + D \end{cases} \tag{5.6}$$

Each of the parameters appearing in the above terms has three values representing two Si potentials, $Si(B)$ and $Si(C)$, given by Tersoff (1988, 1988a) and an As potential derived by Smith (1992). These values are given in Table 5.1.

Each atom, say atom i, has its own potential energy, E_i, given by (5.1). The sum in (5.1) is taken over j, all neighbours of i. Therefore to determine the potential energy of a single particle one has to calculate (5.1) which involves the calculation of (5.2)-(5.6) for each neighbour of that particle. Notice that the energy of a particle depends upon the distances and angles subtended with respect to the other particles and therefore different particles have different energies. The objective function becomes the total energy for all atoms, i.e.,

$$f = \sum_i E_i, \quad \forall i \; . \tag{5.7}$$

It is clear from (5.7) that f is a function of atomic coordinates. Therefore we consider the atomic positions in two and three dimensional space as variables. We describe more details of all variables involved in the next section.

Potential Parameters for *Si(B)*, *Si(C)* and *As*

Table 5.1

	As	Si(B)	Si(C)
c	5.2731318	4.8381	1.0039×10^{-5}
d	0.75102662	2.0417	16.216
h	0.15292354	0.0000	-0.59826
n_1	0.60879133	22.956	0.78734
γ	0.00748809	0.33675	1.0999×10^{-6}
λ_1	6.739581257	$3.2394 \times CS_i$	$2.4799 \times CS_i$
λ_2	4.886847795	$1.3258 \times CS_i$	$1.7322 \times CS_i$
A	10.45561332	$3.2647E3$	$1.8308E3$
B	14.41961332	$9.5373E1$	$4.7118E2$
R	$3.50/CA_s$	$3.0/CS_i$	$2.85/CS_i$
D	$0.15/CA_s$	$0.20/CS_i$	$0.15/CS_i$

$CA_s = 5.6537/2$, $CS_i = 5.4307/2$

Calculation of the Potential Minima

In order to calculate the minimum potential energy for small numbers of particles, we first fix a particle at the origin and choose our second particle to lie on the positive x-axis. The third particle is chosen to lie in the x-y plane. Therefore the variables involving the third particle are radial distance of the position of the particle from origin and its polar angle. Since the position of the first particle is always fixed and the second particle is restricted to the positive x-axis, this gives a minimisation problem involving three variables for three particles (P3). For four particles (P4), a further three variables (the cartesian co-ordinates of the 4-th particle) are required to give a minimisation problem in six independent variables. Further particles (P5 and P6) are added to determine the energetics of small clusters. Therefore, for clusters of 4, 5 and 6 particles the number of dimensions become 6, 9 and 12 respectively. The physics involved in the problems imposes the following restrictions on the variables. The first two variables are taken to lie in $[0, 1.16]$ for *Si* and $[0, 1.30]$ for *As*. The third variable for all cases is taken to lie in, $[0, \pi]$, and all other variables are specified on $[-1.5, 1.5]$ for both *Si* and *As*. However, we invoke symmetry arguments to constrain the third coordinates of the fourth and fifth particles to be non-negative and non-positive respectively. We do not constrain the coordinates of the sixth particle, preferring instead to calculate all the symmetric structures (with respect

to the 6th particle) which are identical apart from rotational symmetry as a check on the accuracy of the calculation.

To demonstrate the complex nature of the objective function for this problem we now give this function explicitly for four atoms i, j, k and l. Therefore, for P4, let x_1 be the x-coordinate of the 2nd particle, x_2 be the radial distance from the origin to the 3rd particle, x_3 the angle the 3rd particle subtends with the x-axis, x_4, x_5, x_6 the coordinates of the 4th particle. Then

$$f = E_i + E_j + E_k + E_l = f(x_1, \ldots, x_6)$$

and the E_i are given by

$$\begin{aligned}
E_i &= 0.5\Big[f_c(r_{ij})\left[V_R(r_{ij}) - B_{ij}V_A(r_{ij})\right] + f_c(r_{ik})\left[V_R(r_{ik}) - B_{ik}V_A(r_{ik})\right] + \\
&\qquad f_c(r_{il})\left[V_R(r_{il}) - B_{il}V_A(r_{il})\right]\Big] \\
&= 0.5\Big[f_c(r_{ij})\left[Ae^{-\lambda_1 r_{ij}} - Be^{-\lambda_2 r_{ij}}\left\{1 + \gamma^{n_1}\left(f_c(r_{ik})g(\theta_{ijk}) + f_c(r_{il})g(\theta_{ijl})\right)^{n_1}\right\}^{-\frac{1}{2n_1}}\right] \\
&\quad + f_c(r_{ik})\left[Ae^{-\lambda_1 r_{ik}} - Be^{-\lambda_2 r_{ik}}\left\{1 + \gamma^{n_1}\left(f_c(r_{ij})g(\theta_{ikj}) + f_c(r_{il})g(\theta_{ikl})\right)^{n_1}\right\}^{-\frac{1}{2n_1}}\right] \\
&\quad + f_c(r_{il})\left[Ae^{-\lambda_1 r_{il}} - Be^{-\lambda_2 r_{il}}\left\{1 + \gamma^{n_1}\left(f_c(r_{ij})g(\theta_{ilj}) + f_c(r_{ik})g(\theta_{ilk})\right)^{n_1}\right\}^{-\frac{1}{2n_1}}\right]\Big] \\
E_j &= 0.5\Big[f_c(r_{ji})\left[V_R(r_{ji}) - B_{ji}V_A(r_{ji})\right] + f_c(r_{jk})\left[V_R(r_{jk}) - B_{jk}V_A(r_{jk})\right] + \\
&\qquad f_c(r_{jl})\left[V_R(r_{jl}) - B_{jl}V_A(r_{jl})\right]\Big] \\
&= 0.5\Big[f_c(r_{ji})\left[Ae^{-\lambda_1 r_{ji}} - Be^{-\lambda_2 r_{ji}}\left\{1 + \gamma^{n_1}\left(f_c(r_{jk})g(\theta_{jik}) + f_c(r_{jl})g(\theta_{jil})\right)^{n_1}\right\}^{-\frac{1}{2n_1}}\right] \\
&\quad + f_c(r_{jk})\left[Ae^{-\lambda_1 r_{jk}} - Be^{-\lambda_2 r_{jk}}\left\{1 + \gamma^{n_1}\left(f_c(r_{ji})g(\theta_{jki}) + f_c(r_{jl})g(\theta_{jkl})\right)^{n_1}\right\}^{-\frac{1}{2n_1}}\right] \\
&\quad + f_c(r_{jl})\left[Ae^{-\lambda_1 r_{jl}} - Be^{-\lambda_2 r_{jl}}\left\{1 + \gamma^{n_1}\left(f_c(r_{jk})g(\theta_{jlk}) + f_c(r_{ji})g(\theta_{jli})\right)^{n_1}\right\}^{-\frac{1}{2n_1}}\right]\Big] \\
E_k &= 0.5\Big[f_c(r_{ki})\left[V_R(r_{ki}) - B_{ki}V_A(r_{ki})\right] + f_c(r_{kj})\left[V_R(r_{kj}) - B_{kj}V_A(r_{kj})\right] + \\
&\qquad f_c(r_{kl})\left[V_R(r_{kl}) - B_{kl}V_A(r_{kl})\right]\Big] \\
&= 0.5\Big[f_c(r_{ki})\left[Ae^{-\lambda_1 r_{ki}} - Be^{-\lambda_2 r_{ki}}\left\{1 + \gamma^{n_1}\left(f_c(r_{kj})g(\theta_{kij}) + f_c(r_{kl})g(\theta_{kil})\right)^{n_1}\right\}^{-\frac{1}{2n_1}}\right] \\
&\quad + f_c(r_{kj})\left[Ae^{-\lambda_1 r_{kj}} - Be^{-\lambda_2 r_{kj}}\left\{1 + \gamma^{n_1}\left(f_c(r_{ki})g(\theta_{kji}) + f_c(r_{kl})g(\theta_{kjl})\right)^{n_1}\right\}^{-\frac{1}{2n_1}}\right] \\
&\quad + f_c(r_{kl})\left[Ae^{-\lambda_1 r_{kl}} - Be^{-\lambda_2 r_{kl}}\left\{1 + \gamma^{n_1}\left(f_c(r_{ki})g(\theta_{kli}) + f_c(r_{kj})g(\theta_{klj})\right)^{n_1}\right\}^{-\frac{1}{2n_1}}\right]\Big] \\
E_l &= 0.5\Big[f_c(r_{li})\left[V_R(r_{li}) - B_{li}V_A(r_{li})\right] + f_c(r_{lj})\left[V_R(r_{lj}) - B_{lj}V_A(r_{lj})\right] + \\
&\qquad f_c(r_{lk})\left[V_R(r_{lk}) - B_{lk}V_A(r_{lk})\right]\Big] \\
&= 0.5\Big[f_c(r_{li})\left[Ae^{-\lambda_1 r_{li}} - Be^{-\lambda_2 r_{li}}\left\{1 + \gamma^{n_1}\left(f_c(r_{lj})g(\theta_{lij}) + f_c(r_{lk})g(\theta_{lik})\right)^{n_1}\right\}^{-\frac{1}{2n_1}}\right] \\
&\quad + f_c(r_{lj})\left[Ae^{-\lambda_1 r_{lj}} - Be^{-\lambda_2 r_{lj}}\left\{1 + \gamma^{n_1}\left(f_c(r_{li})g(\theta_{lji}) + f_c(r_{lk})g(\theta_{ljk})\right)^{n_1}\right\}^{-\frac{1}{2n_1}}\right] \\
&\quad + f_c(r_{lk})\left[Ae^{-\lambda_1 r_{lk}} - Be^{-\lambda_2 r_{lk}}\left\{1 + \gamma^{n_1}\left(f_c(r_{li})g(\theta_{lki}) + f_c(r_{lj})g(\theta_{lkj})\right)^{n_1}\right\}^{-\frac{1}{2n_1}}\right]\Big]
\end{aligned}$$

Hence in terms of the variables $x_i,\ i = 1, 2, \ldots, 6,$

$$f = 0.5\Big[f_c(r_{ij} = x_1)\Big[Ae^{-\lambda_1 x_1} - Be^{-\lambda_2 x_1}\Big\{1 + \gamma^{n_1}\big(f_c(r_{ik} = x_2)(1 + c^2/d^2 - c^2/[d^2$$
$$+(h - \cos x_3)^2]) + f_c(r_{il} = \sqrt{(}x_4^2 + x_5^2 + x_6^2))(1 + c^2/d^2 - c^2/[d^2 + (h -$$
$$\frac{x_4}{\sqrt{(}x_4^2 + x_5^2 + x_6^2)})^2]))^{n_1}\Big\}^{-\frac{1}{2n_1}}\Big] + f_c(r_{ik})\Big[Ae^{-\lambda_1 x_2} - Be^{-\lambda_2 x_2}\Big\{1 +$$
$$\gamma^{n_1}\big(f_c(r_{ij})(1 + c^2/d^2 - c^2/[d^2 + (h - \cos x_3)^2]) + f_c(r_{il})(1 + c^2/d^2 - c^2/[d^2$$
$$+(h - \frac{x_4 \cos x_3 + x_5 \sin x_3}{\sqrt{(}x_4^2 + x_5^2 + x_6^2)})^2]))^{n_1}\Big\}^{-\frac{1}{2n_1}}\Big] + f_c(r_{il})\Big[Ae^{-\lambda_1\sqrt{(}x_4^2+x_5^2+x_6^2)}$$
$$-Be^{-\lambda_2\sqrt{(}x_4^2+x_5^2+x_6^2)}\Big\{1 + \gamma^{n_1}\big(f_c(r_{ij})(1 + c^2/d^2 - c^2/[d^2 + (h -$$
$$\frac{x_4}{\sqrt{(}x_4^2 + x_5^2 + x_6^2)})^2]) + f_c(r_{ik})(1 + c^2/d^2 - c^2/[d^2 + (h -$$
$$\frac{x_4 \cos x_3}{\sqrt{(}x_4^2 + x_5^2 + x_6^2)} - \frac{x_5 \sin x_3}{\sqrt{(}x_4^2 + x_5^2 + x_6^2)})^2]))^{n_1}\Big\}^{-\frac{1}{2n_1}}\Big]\Big]$$
$$+0.5\Big[f_c(r_{ji})\Big[Ae^{-\lambda_1 x_1} - Be^{-\lambda_2 x_1}\Big\{1 + \gamma^{n_1}\big(f_c(r_{jk} = \sqrt{(}x_1^2 + x_2^2 - 2x_1x_2\cos x_3))$$
$$(1 + c^2/d^2 - c^2/[d^2 + (h - \frac{(x_1 - x_2\cos x_3)}{\sqrt{(}x_1^2 + x_2^2 - 2x_1x_2\cos x_3)})^2])$$
$$+f_c(r_{jl} = \sqrt{(}x_1 - x_4)^2 + x_5^2 + x_6^2))$$
$$(1 + c^2/d^2 - c^2/[d^2 + (h - \frac{(x_1 - x_4)}{\sqrt{(}(x_4 - x_1)^2 + x_5^2 + x_6^2)})^2]))^{n_1}\Big\}^{-\frac{1}{2n_1}}\Big] + f_c(r_{jk})$$
$$\Big[Ae^{-\lambda_1\sqrt{(}x_1^2+x_2^2-2x_1x_2\cos x_3)} - Be^{-\lambda_2\sqrt{(}x_1^2+x_2^2-2x_1x_2\cos x_3)}\Big\{1 + \gamma^{n_1}\big(f_c(r_{ji})$$
$$(1 + c^2/d^2 - c^2/[d^2 + (h - \frac{(x_1 - x_2\cos x_3)}{\sqrt{(}x_1^2 + x_2^2 - 2x_1x_2\cos x_3)})^2]) + f_c(r_{jl})(1 + c^2/d^2 - c^2/[d^2$$
$$+(h - \frac{(x_1 - x_4)(x_1 - x_2\cos x_3) + x_2x_5\sin x_3}{\sqrt{(}x_1^2 + x_2^2 - 2x_1x_2\cos x_3)\sqrt{(}(x_4 - x_1)^2 + x_5^2 + x_6^2)})^2]))^{n_1}\Big\}^{-\frac{1}{2n_1}}\Big]$$
$$+f_c(r_{jl})\Big[Ae^{-\lambda_1\sqrt{((}x_4-x_1)^2+x_5^2+x_6^2)} - Be^{-\lambda_2\sqrt{((}x_4-x_1)^2+x_5^2+x_6^2)}$$
$$\Big\{1 + \gamma^{n_1}\big(f_c(r_{jk})(1 + c^2/d^2 - c^2/[d^2 + (h -$$
$$\frac{(x_1 - x_4)(x_1 - x_2\cos x_3) + x_2x_5\sin x_3}{\sqrt{(}x_1^2 + x_2^2 - 2x_1x_2\cos x_3)\sqrt{(}(x_4 - x_1)^2 + x_5^2 + x_6^2)})^2]) + f_c(r_{ji})(1 +$$
$$c^2/d^2 - c^2/[d^2 + (h - \frac{(x_1 - x_4)}{\sqrt{(}(x_4 - x_1)^2 + x_5^2 + x_6^2)})^2]))^{n_1}\Big\}^{-\frac{1}{2n_1}}\Big]\Big]$$

$$+0.5\Bigg[f_c(r_{ki})\Big[Ae^{-\lambda_1 x_2}-Be^{-\lambda_2 x_2}\Big\{1+\gamma^{n_1}\big(f_c(r_{kj})(1+c^2/d^2-c^2/[d^2+(h$$

$$-\frac{(x_2\cos x_3-x_1)\cos x_3+x_2\sin^2 x_3}{\sqrt{(x_1^2+x_2^2-2x_1x_2\cos x_3)}})^2])$$

$$+f_c(r_{kl}=\sqrt{(x_2^2+x_4^2+x_5^2+x_6^2-2x_2x_4\cos x_3-2x_2x_5\sin x_3)})(1+c^2/d^2$$

$$-c^2/[d^2+(h-\frac{\cos x_3(x_2\cos x_3-x_4)+\sin x_3(x_2\sin x_3-x_5)}{\sqrt{(x_2^2+x_4^2+x_5^2+x_6^2-2x_2x_4\cos x_3-2x_2x_5\sin x_3)}})^2])\big)^{n_1}\Big\}^{-\frac{1}{2n_1}}\Big]$$

$$+f_c(r_{kj})\Big[Ae^{-\lambda_1\sqrt{(x_1^2+x_2^2-2x_1x_2\cos x_3)}}-Be^{-\lambda_2\sqrt{(x_1^2+x_2^2-2x_1x_2\cos x_3)}}\Big\{1+\gamma^{n_1}\big(f_c(r_{ki})$$

$$(1+c^2/d^2-c^2/[d^2+(h-\frac{(x_2\cos x_3-x_1)\cos x_3+x_2\sin^2 x_3}{\sqrt{(x_1^2+x_2^2-2x_1x_2\cos x_3)}})^2])$$

$$+f_c(r_{kl})(1+c^2/d^2-c^2/[d^2+(h-$$

$$\frac{(x_4-x_2\cos x_3)(x_1-x_2\cos x_3)+x_2\sin x_3(x_2\sin x_3-x_5)}{\sqrt{(x_1^2+x_2^2-2x_1x_2\cos x_3)}\sqrt{((x_2^2+x_4^2+x_5^2+x_6^2-2x_2x_4\cos x_3-2x_2x_5\sin x_3)}})^2])\big)^{n_1}$$

$$\Big\}^{-\frac{1}{2n_1}}\Big]+f_c(r_{kl})\Big[Ae^{-\lambda_1\sqrt{((x_2^2+x_4^2+x_5^2+x_6^2-2x_2x_4\cos x_3-2x_2x_5\sin x_3)}}$$

$$-Be^{-\lambda_2\sqrt{((x_2^2+x_4^2+x_5^2+x_6^2-2x_2x_4\cos x_3-2x_2x_5\sin x_3)}}$$

$$\Big\{1+\gamma^{n_1}\big(f_c(r_{ki})(1+c^2/d^2-c^2/[d^2+(h-$$

$$\frac{\cos x_3(x_2\cos x_3-x_4)+\sin x_3(x_2\sin x_3-x_5)}{\sqrt{(x_2^2+x_4^2+x_5^2+x_6^2-2x_2x_4\cos x_3-2x_2x_5\sin x_3)}})^2])+f_c(r_{kj})(1+c^2/d^2-c^2/[d^2+$$

$$(h-\frac{(x_4-x_2\cos x_3)(x_1-x_2\cos x_3)+x_2\sin x_3(x_2\sin x_3-x_5)}{\sqrt{(x_1^2+x_2^2-2x_1x_2\cos x_3)}\sqrt{((x_2^2+x_4^2+x_5^2+x_6^2-2x_2x_4\cos x_3-2x_2x_5\sin x_3)}})^2$$

$$])\big)^{n_1}\Big\}^{-\frac{1}{2n_1}}\Big]\Bigg]$$

$$+0.5\Bigg[f_c(r_{li})\Big[Ae^{-\lambda_1\sqrt{(x_4^2+x_5^2+x_6^2)}}-Be^{-\lambda_2\sqrt{(x_4^2+x_5^2+x_6^2)}}\Big\{1+\gamma^{n_1}\big(f_c(r_{lj})(1+c^2/d^2$$

$$-c^2/[d^2+(h-\frac{x_4^2+x_5^2+x_6^2-x_4x_1}{\sqrt{(x_1^2+x_4^2+x_5^2+x_6^2-2x_1x_4)}\sqrt{(x_4^2+x_5^2+x_6^2)}})^2])$$

$$+f_c(r_{lk})(1+c^2/d^2-c^2/[d^2+(h-$$

$$\frac{x_4^2+x_5^2+x_6^2-x_2x_4\cos x_3-x_2x_5\sin x_3}{\sqrt{(x_2^2+x_4^2+x_5^2+x_6^2-2x_2x_4\cos x_3-2x_2x_5\sin x_3)}\sqrt{(x_4^2+x_5^2+x_6^2)}})^2])\big)^{n_1}\Big\}^{-\frac{1}{2n_1}}\Big]$$

$$+f_c(r_{lj})\Big[Ae^{-\lambda_1\sqrt{(x_1^2+x_4^2+x_5^2+x_6^2-2x_1x_4)}}-Be^{-\lambda_2\sqrt{(x_1^2+x_4^2+x_5^2+x_6^2-2x_1x_4)}}\Big\{1+\gamma^{n_1}$$

$$(f_c(r_{li})(1+c^2/d^2-c^2/[d^2+(h-$$

$$\frac{x_4^2+x_5^2+x_6^2-x_4x_1}{\sqrt{(x_1^2+x_4^2+x_5^2+x_6^2-2x_1x_4)}\sqrt{(x_4^2+x_5^2+x_6^2)}})^2])$$

$$+f_c(r_{lk})(1+c^2/d^2-c^2/[d^2+(h-$$

$$\frac{(x_4-x_1)(x_4-x_2\cos x_3)+x_5(x_5-x_2\sin x_3)+x_6^2}{\sqrt{(x_1^2+x_4^2+x_5^2+x_6^2-2x_1x_4)}\sqrt{(x_2^2+x_4^2+x_5^2+x_6^2-2x_2x_4\cos x_3-2x_2x_5\sin x_3)}})^2])$$

$$)^{n_1}\Big\}^{-\frac{1}{2n_1}}\Big]$$

$$+f_c(r_{lk})\Big[Ae^{-\lambda_1\sqrt{(x_2^2+x_4^2+x_5^2+x_6^2-2x_2x_4\cos x_3-2x_2x_5\sin x_3)}}$$

$$-Be^{-\lambda_2\sqrt{(x_2^2+x_4^2+x_5^2+x_6^2-2x_2x_4\cos x_3-2x_2x_5\sin x_3)}}$$

$$\Big\{1+\gamma^{n_1}\big(f_c(r_{li})(1+c^2/d^2-c^2/[d^2+(h-$$

$$\frac{x_4(x_4-x_2\cos x_3)+x_5(x_5-x_2\sin x_3)+x_6^2}{\sqrt{(x_4^2+x_5^2+x_6^2)}\sqrt{(x_2^2+x_4^2+x_5^2+x_6^2-2x_2x_4\cos x_3-2x_2x_5\sin x_3)}})^2])$$

$$+f_c(r_{lj})(1+c^2/d^2-c^2/[d^2+(h-$$

$$\frac{(x_4-x_2\cos x_3)(x_4-x_1)+x_5(x_5-x_2\sin x_3)+x_6^2}{\sqrt{(x_1^2+x_4^2+x_5^2+x_6^2-2x_1x_4)}\sqrt{(x_2^2+x_4^2+x_5^2+x_6^2-2x_2x_4\cos x_3-2x_2x_5\sin x_3)}})^2$$

$$])\big)^{n_1}\Big\}^{-\frac{1}{2n_1}}\Big]\Big].$$

5.3 Numerical Considerations and Comparison Studies

In this section we compare the performances of all the global optimization algorithms previously discussed. The purpose of the comparisons is to see the merits of the modified algorithms over their original versions and to see which method is most efficient in finding the best local minimum. As far as the implementations of SA and ASA are concerned we invoked local search only for two iterations both in the generation mechanism Alternative B (see section 3.3.2 in Chapter 3) and to find the aspiration solution. The values of the common parameters of the SA and ASA algorithms were kept the same (see sections 3.2.3 and 3.4.3 in Chapter 3) but we used $\delta_{\min}=0.05$, $\delta_{\max}=0.8$,$n_a=5$,$\epsilon_r=10^{-3}$, $\mu=0.1$ and the length of Markov chain defined by (3.69) for the implementation of the ASA algorithm. In the CRS5 algorithm only a single iteration of the local search (E04UCF) is used except

at the final stage where a complete local search is carried out. The tolerance for the local search was 10^{-5} in all implementations as no numerical difficulties were encountered for this tolerance. The tolerance for the CRS algorithms was $\epsilon = 10^{-4}$ except that for CRS5 we took $\epsilon = 10^{-1}$ and $t = 0.5$ (see section 4.4 in Chapter 4). The value of N (the number of initial points) was taken as $10(n+1)$, where n is the dimension of the problem. In all applications of CRS4, we used γ=0.1 and 50%M (see section 4.3 in Chapter 4). We first examined the performance of the CRS algorithms for up to six particles for both *Si* and *As* and the results are given in Tables 5.2 and 5.3.

Results of CRS for all Problems

Table 5.2

CRS2		CRS3		CRS4		CRS5			
FE	cpu	FE	cpu	FE	cpu	FE	cpu		
1123	0.48	1094	0.76	755	0.34	808	0.39	P3	*Si(B)*
4304	5.33	3724	6.26	2418	2.25	2967	2.64	P4	
11111	16.74	18348	38.72	3971	5.66	4321	6.72	P5	
37677	82.07	64067	244.97	44980	91.99	42134	87.17	P6	
54215	104.62	87233	290.71	52124	108.24	50230	94.92		Total
998	0.26	1092	0.58	705	0.19	612	0.22	P3	*Si(C)*
9829	4.96	11446	16.48	7369	3.24	8521	3.50	P4	
58063	46.73	45173	82.00	12479	9.13	10362	8.90	P5	
182720	235.21	329089	1199.38	156916	311.48	175233	222.1	P6	
251610	287.16	386800	1298.44	177469	324.04	194728	234.72		Total
952	0.42	1281	1.16	951	0.44	820	0.41	P3	*As*
4075	3.92	5111	7.70	1694	1.62	2113	2.06	P4	
8877	15.58	43173	101.56	6130	9.55	9124	17.11	P5	
38381	101.52	67241	293.38	33192	85.48	30784	79.20	P6	
52285	121.44	116806	403.80	41967	97.09	42841	98.78		Total

From the totals in Table 5.2 it is clear that CRS5 works surprisingly well especially in terms of cpu time. CRS4 and CRS5 are better than CRS2 and CRS3 with CRS4 better than CRS5 in terms of FE but the reverse holding in terms of cpu time. CRS3 is by far the worst of all four algorithms. It is interesting that *Si(C)* is the most difficult problem.

Of course, Table 5.2 gives the value of FE and cpu obtained on satisfaction of the stopping condition but it is also important to consider the quality of the approximations to the global minimum found by the four algorithms. The values of f^* are shown in Table 5.3.

Best minima found by CRS for all Problems

Table 5.3

CRS2	CRS3	CRS4	CRS5		
f^*	f^*	f^*	f^*		
-7.87	-7.87	-7.87	-7.87	P3	*Si(B)*
-15.70	-15.70	-15.70	-15.70	P4	
-20.39	-20.31	-20.39	-15.70	P5	
-24.51	-24.51	-26.51	-24.51	P6	
-5.33	-5.33	-5.33	-5.33	P3	*Si(C)*
-7.99	-7.99	-7.99	-7.99	P4	
-10.66	-11.30	-10.57	-10.57	P5	
-15.10	-15.19	-15.88	-15.10	P6	
-6.85	-6.85	-6.85	-6.85	P3	*As*
-10.65	-10.65	-10.65	-10.65	P4	
-14.78	-14.78	-14.78	-14.78	P5	
-19.57	-19.57	-18.49	-16.51	P6	

The results here are remarkably uniform and perhaps the only real conclusion is that CRS5 is the worst of the four algorithms for locating the smaller values for f^* which, of course, are not known theoretically.

We next tested the MSL and TMSL algorithms. The results for MSL and TMSL are given in Tables 5.4 and 5.5 respectively, where IT represents the number of iterations. To represent the complexity of the problem as far as the number of local minima is concerned we also give the number, LM, of local minima found. The true number of local minima for the problem is unknown but numerical experience suggests that it is quite high. We therefore fixed the number of iterations at 5 for MSL and TMSL. In other words, these algorithms were allowed to run for a maximum of five iterations only. In all implementations of the MSL algorithm we used a sample of size 100 but varied γ and σ. The results for TMSL in Table 5.5 were obtained by setting σ equal to 4. However, different values for the number of nearest neighbours g and the number of sample points N were tested.

Results of MSL for all Problems

Table 5.4

γ	σ	f^*	FE	cpu	LM	IT		
0.1	2	-7.87	438	0.18	2	2	P3	Si(B)
0.1	4	-7.87	438	0.18	2	2		
0.2	2	-7.87	259	0.14	2	1		
0.2	4	-7.87	210	0.17	2	1		
0.1	2	-15.70	6352	4.36	27	5	P4	
0.1	4	-15.70	5437	3.84	25	5		
0.2	2	-15.70	7516	5.37	25	5		
0.2	4	-15.70	5312	3.80	21	5		
0.1	2	-20.39	15415	14.70	66	5	P5	
0.1	4	-20.39	12954	11.94	55	5		
0.2	2	-20.39	18568	17.97	58	5		
0.2	4	-20.39	16082	15.53	49	5		
0.1	2	-24.51	18666	27.30	48	5	P6	
0.1	4	-24.51	19184	25.96	50	5		
0.2	2	-24.51	24216	32.73	65	5		
0.2	4	-24.51	23088	31.51	63	5		
0.1	2	-5.33	376	0.14	1	1	P3	Si(C)
0.1	4	-5.33	159	0.19	1	1		
0.2	2	-5.33	376	0.17	1	1		
0.2	4	-5.33	159	0.05	1	1		
0.1	2	-7.99	5067	3.01	26	5	P4	
0.1	4	-7.99	3978	2.24	21	5		
0.2	2	-7.99	5217	3.18	29	5		
0.2	4	-7.99	3820	2.39	23	5		
0.1	2	-10.66	11975	8.46	64	5	P5	
0.1	4	-10.66	10023	7.12	51	5		
0.2	2	-10.66	9698	6.92	57	5		
0.2	4	-10.66	8202	6.10	48	5		
0.1	2	-13.24	14231	11.78	49	5	P6	
0.1	4	-13.24	14027	11.70	48	5		
0.2	2	-13.24	17355	14.54	69	5		
0.2	4	-13.24	17302	14.29	67	5		
0.1	2	-6.85	690	0.50	2	2	P3	As
0.1	4	-6.85	151	0.08	1	1		
0.2	2	-6.85	970	0.44	3	2		
0.2	4	-6.85	151	0.10	1	1		

γ	σ	f^*	FE	cpu	LM	IT	
0.1	2	-10.65	8785	6.10	24	5	P4
0.1	4	-10.65	7443	5.49	20	5	
0.2	2	-10.65	9173	6.44	25	5	
0.2	4	-10.65	7228	5.48	22	5	
0.1	2	-14.57	24431	25.05	61	5	P5
0.1	4	-14.57	20170	20.84	52	5	
0.2	2	-14.57	25152	25.54	65	5	
0.2	4	-14.57	20340	20.60	54	5	
0.1	2	-18.35	35147	48.42	67	5	P6
0.1	4	-18.35	39862	55.82	72	5	
0.2	2	-18.35	43917	61.27	81	5	
0.2	4	-18.35	37946	51.79	78	5	

Results of TMSL for all Problems

Table 5.5

N	g	f^*	FE	cpu	LM	IT		
10n	n-2	-7.87	212	0.14	3	1	P3	Si(B)
10n	n-1	-7.87	212	0.18	3	1		
15n	n-2	-7.87	227	0.26	2	1		
15n	n	-7.87	227	0.30	3	1		
10n	n-2	-15.70	3369	5.66	11	5	P4	
10n	n-1	-15.70	2520	4.47	9	4		
15n	n-2	-14.04	3784	14.34	21	5		
15n	n	-13.11	2896	16.27	17	5		
10n	n-2	-18.97	3702	15.04	12	4	P5	
10n	n-1	-18.97	3594	15.81	11	4		
15n	n-2	-20.39	6653	55.91	23	5		
15n	n	-18.97	4962	60.58	18	5		
10n	n-2	-24.44	4579	20.43	9	2	P6	
10n	n-1	-24.44	4139	22.54	8	2		
15n	n-2	-25.53	13531	169.34	28	5		
15n	n	-24.44	5921	68.14	12	2		
10n	n-2	-5.33	254	0.15	1	1	P3	Si(C)
10n	n-1	-5.33	254	0.14	1	1		
15n	n-2	-5.33	216	0.25	2	1		
15n	n	-5.33	269	0.30	1	1		
10n	n-2	-7.99	2557	4.88	14	5	P4	
10n	n-1	-7.99	1850	3.76	9	4		
15n	n-2	-7.99	3723	13.61	24	5		
15n	n	-7.99	1606	2.82	5	1		
10n	n-2	-11.30	5198	18.84	20	5	P5	
10n	n-1	-11.30	4442	19.54	18	5		
15n	n-2	-12.08	7858	57.21	29	5		
15n	n	-12.08	5542	61.11	22	5		
10n	n-2	-12.64	2612	10.10	5	1	P6	
10n	n-1	-12.64	2328	11.20	5	1		
15n	n-2	-13.32	10212	165.30	31	5		
15n	n	-13.32	8944	78.47	24	5		
10n	n-2	–6.85	231	0.39	2	1	P3	As
10n	n-1	-6.85	231	0.35	2	1		
15n	n-2	-6.85	228	0.32	1	1		
15n	n	-6.85	228	0.52	3	1		

N	g	f^*	FE	cpu	LM	IT	
10n	n-2	-10.65	4363	7.10	15	5	P4
10n	n-1	-10.65	2347	3.78	8	3	
15n	n-2	-10.65	5903	16.07	21	5	
15n	n	-10.65	1682	5.63	6	2	
10n	n-2	-14.35	6315	21.54	18	5	P5
10n	n-1	-14.35	3648	12.77	9	3	
15n	n-2	-14.35	7394	59.71	26	5	
15n	n	-14.35	1831	11.27	7	1	
10n	n-2	-15.62	4900	15.12	6	1	P6
10n	n-1	-15.62	3878	14.51	5	1	
15n	n-2	-15.62	11856	163.70	21	5	
15n	n	-15.62	6064	37.63	8	1	

The Tables 5.4 and 5.5 show that the number of local minima found by MSL is much higher than that found by TMSL. In a total of 48 runs MSL continued up to 5 iterations in 36 runs and TMSL reached up to 5 iterations in 19 runs. Evidently the total number of function evaluations for MSL is much higher than that for TMSL. In the following Table 5.6 we show the total number of function evaluations and cpu time obtained by MSL and TMSL. Each total here is taken for all three potentials, for P3, P4, P5 and P6 and for the particular values of γ and σ shown in Table 5.6.

Table 5.6

MSL				TMSL				
Total				Total				
γ	σ	FE	cpu	N	g	FE	cpu	*Si(B)+Si(C)+As*
0.1	2	141573	150.00	10n	n-2	38292	119.39	P3+P4+P5+P6
0.1	4	133826	145.40	10n	n-1	29443	109.05	
0.2	2	162417	174.71	15n	n-2	71585	716.02	
0.2	4	139840	151.81	15n	n	40172	343.04	

Table 5.6 shows that TMSL is much superior to MSL in terms of FE but is less so in terms of cpu. Table 5.7 compares the results in terms of the best local minima obtained by MSL and TMSL.

The Best Results for MSL and TMSL for all Problems

Table 5.7

MSL				TMSL					
FE	f^*	LM	cpu	FE	f^*	LM	cpu		
210	-7.87	2	0.17	212	-7.87	3	0.14	P3	*Si(B)*
5312	-15.70	21	3.80	2520	-15.70	9	4.40	P4	
12954	-20.39	55	11.94	6653	-20.39	23	55.91	P5	
18666	-24.51	48	27.30	13531	-25.53	28	169.39	P6	
37142		126	43.21	22916		63	229.84	Total	
159	-5.33	1	0.14	254	-5.33	1	0.15	P3	*Si(C)*
3820	-7.99	23	2.39	1606	-7.99	5	2.82	P4	
8202	-10.66	48	6.10	5542	-12.08	22	61.11	P5	
14027	-13.24	48	11.70	8944	-13.32	24	78.47	P6	
26208		120	20.33	16346		52	142.55	Total	
151	-6.85	2	0.08	228	-6.85	3	0.52	P3	*As*
7228	-10.65	22	5.48	1682	-10.65	6	5.63	P4	
20170	-14.57	52	20.84	1831	-14.35	7	11.27	P5	
35147	-18.35	67	25.05	3878	-15.62	5	14.51	P6	
62696		143	51.45	7619		21	31.93	Total	

There is little to choose between the two methods in terms of the quality of the minima found; TMSL does slightly better for *Si(B)*P6, *Si(C)*P5 and *Si(C)*P6 and MSL for *As*P5 and *As*P6. The number of local minima found is much higher for MSL than for TMSL with a consequent high number of function evaluations for the former. The cpu time for TMSL, however, is much worse than that for MSL. In Table 5.8 we compare the results of the SA and ASA algorithms.

Results of SA and ASA for all Problems

Table 5.8

ASA						SA					
FE	cpu	f^*	R	t	T_o	FE	cpu	f^*	t		
13992	5.23	-7.87	4	54	5556.20	13978	5.91	-7.87	62	P3	*Si(B)*
27988	16.03	-15.70	1	31	7227.13	46458	49.30	-15.70	65	P4	
36417	35.87	-20.31	0	29	7052.22	112951	141.97	-20.31	81	P5	
151109	175.52	-25.98	3	64	10399.23	206166	315.71	-23.12	89	P6	
229506	232.65					379553	512.89			Total	
6073	2.69	-5.33	0	24	2760.72	12768	4.27	-4.95	62	P3	*Si(C)*
53485	20.49	-7.99	4	54	3583.92	56365	25.82	-7.91	81	P4	
113077	66.68	-10.14	4	67	35257.54	114438	85.13	-11.30	80	P5	
66274	47.63	-13.03	0	29	5120.84	272927	254.92	-10.57	90	P6	
238909	137.49					456498	370.14			Total	
20017	8.96	-6.85	4	68	1960.00	14938	6.56	-6.85	63	P3	*As*
60994	50.97	-10.65	4	70	2514.15	52499	58.51	-10.65	53	P4	
34285	39.90	-14.78	0	32	2486.85	108406	149.74	-14.57	81	P5	
10462	149.67	-18.49	1	60	3698.28	205151	319.80	-18.49	89	P6	
125758	249.50					380994	534.61			Total	

The results in this Table clearly indicate that ASA is much superior to SA in terms of both FE and cpu and also in finding better values for f^*.

The best results for all algorithms tested are compared in Table 5.9 and 5.10 with Table 5.9 giving FE and cpu values and Table 5.10 giving the best values for f^*.

Table 5.9

CRS2	CRS3	CRS4	CRS5	ASA	SA	TMSL	MSL			
1123	1094	755	808	13992	13978	212	210	P3	FE	*Si(B)*
4304	3724	2418	2967	27988	46458	2520	5312	P4		
11111	18348	3971	4321	36417	112951	3594	12954	P5		
37677	64067	44980	42134	151109	206166	4139	18666	P6		
54215	87233	52124	50230	229506	379553	10465	37142	Total		
0.48	0.76	0.34	0.39	5.23	5.91	0.14	0.17	P3	cpu	
5.33	6.26	2.25	0.64	16.03	49.30	4.40	3.80	P4		
16.74	38.72	5.66	6.72	35.87	141.97	15.81	11.94	P5		
82.07	244.97	91.99	87.17	175.52	315.71	22.54	27.30	P6		
104.62	290.71	108.24	94.92	232.65	512.89	42.89	43.21	Total		
998	1092	705	612	6073	12768	254	159	P3	FE	*Si(C)*
9829	11446	7369	8521	53485	56365	1606	3820	P4		
58063	45173	12479	10362	113077	114438	5542	8202	P5		
182720	329089	156916	175233	66274	272927	8944	14027	P6		
251610	386800	177469	194728	238909	456498	16346	26208	Total		
0.26	0.58	0.19	0.22	2.69	4.27	0.15	0.14	P3	cpu	
4.96	16.48	3.24	3.50	20.49	25.82	2.82	2.39	P4		
46.73	82.00	9.13	8.90	66.68	85.13	61.11	6.10	P5		
235.21	1199.38	311.48	222.10	47.63	254.92	78.47	11.70	P6		
287.16	1298.44	324.04	234.72	137.49	370.14	142.55	20.33	Total		
952	1281	951	820	20017	14938	228	151	P3	FE	*As*
4075	5111	1694	2113	60994	52499	1682	7228	P4		
8877	43173	6130	9124	34285	108406	1831	20170	P5		
38381	67241	33192	30784	10462	205151	3878	35147	P6		
52285	116806	41967	42841	125758	380994	7619	62696	Total		
0.42	1.16	0.44	0.41	8.96	6.56	0.52	0.08	P3	cpu	
3.92	7.70	1.62	2.06	50.97	58.51	5.63	5.48	P4		
15.58	101.56	9.55	17.11	39.90	149.74	11.27	20.84	P5		
101.52	293.38	85.48	79.20	149.67	319.80	14.51	25.05	P6		
121.44	403.80	97.09	98.78	249.50	534.61	31.93	51.45	Total		
358110	590839	271560	287799	594173	1217045	34430	126046	G-T	FE	
513.22	1992.95	529.37	408.84	619.64	1417.04	217.37	88.59	G-T	cpu	

G-T : Grand Total

The best minima found

Table 5.10

CRS2	CRS3	CRS4	CRS5	ASA	SA	TMSL	MSL	
								Si(B)
-7.87	-7.87	-7.87	-7.87	-7.87	-7.87	-7.87	-7.87	P3
-15.70	-15.70	-15.70	-15.70	-15.70	-15.70	-15.70	-15.70	P4
-20.39	-20.31	-20.39	-15.70	-20.31	-20.31	-18.97	-20.39	P5
-24.51	-24.51	-26.51	-24.51	-25.98	-23.12	-24.44	-24.51	P6
								Si(C)
-5.33	-5.33	-5.33	-5.33	-5.33	-4.95	-5.33	-5.33	P3
-7.99	-7.99	-7.99	-7.99	-7.99	-7.91	-7.99	-7.99	P4
-10.66	-11.30	-10.57	-10.57	-10.14	-11.30	-12.08	-10.66	P5
-15.10	-15.19	-15.88	-15.10	-13.03	-10.57	-13.32	-13.24	P6
								As
-6.85	-6.85	-6.85	-6.85	-6.85	-6.85	-6.85	-6.85	P3
-10.65	-10.65	-10.65	-10.65	-10.65	-10.65	-10.65	-10.65	P4
-14.78	-14.78	-14.78	-14.78	-14.78	-14.57	-14.35	-14.57	P5
-19.57	-19.57	-18.49	-16.51	-18.49	-18.49	-15.62	-18.35	P6

A critical comparison of the results in Table 5.9 shows that in terms of FE and cpu the algorithms can be listed in the following order of merit.

	FE	cpu
1	TMSL	MSL
2	MSL	TMSL
3	CRS4	CRS5
4	CRS5	CRS2
5	CRS2	CRS4
6	CRS3	ASA
7	ASA	SA
8	SA	CRS3

5.4 A Short Ranged Many-Body Potential for Modelling bcc Metal

The energy and structure of defects such as vacancies and interstitials in metals attracts a considerable experimental and theoretical research effort from the physics and material science communities. There have been a number of theoretical approaches to calculating the energetics of such defects and one of the most popular has been the use of interatomic potential functions. These interatomic potentials can be determined from *ab initio* calculations or by empirical means. Semi-empirical pair potentials have been used to represent accurately the cohesive energy of fcc metals at the correct lattice spacing but have been found to be inappropriate for use in calculating energetics of defects because the elastic properties of the material were inaccurately represented. They also cannot be used for investigating the effects of chemically active impurities. To overcome these problems many-body approaches were adopted. Two of the most successful have been the potentials of the Finnis-Sinclair or embedded atom (Foiles, 1985) types. These potentials have been used with various degrees of success to study the properties of surfaces, point defects and cracks. Many-body potentials have also been developed for semiconductor materials using the ideas of bond order and preferred angular directions. The approaches used were based on entirely different considerations than from those for metals but Brenner (Brenner, 1989) has shown that the embedded atom method for metals and the Tersoff/Abell (Tersoff, 1988) approach for covalent materials are mathematically equivalent.

Most of the work for metals has been for the closed-packed fcc and hcp configurations. This is because under pair potential interactions a large collection of particles will always move to an energetically favourable distribution which is closed-packed. This is clearly inappropriate for bcc materials. We attempt to overcome this problem by fitting a short-ranged many-body potential whose functional form is designed to give the bcc structure as the preferred potential energy minimum. This potential description is aimed at ensuring that the bcc arrangement is favoured over the fcc, hcp and diamond structures. The potential description contains a number of free parameters which are optimized to fit the cohesive energy, elastic constants and the bulk modulus of bcc materials. Although a model of a crystal, based on pair-wise interaction, is insufficient to model a stable bcc lattice or fit the elastic constants of the material, our potential description is a modification of two particle interaction which takes account of the fact that the interaction must be affected by near neighbours. The Tersoff/Abell approach considers the attractive term to be modified by the presence of near neighbours. The functional form of the many-body potential is, therefore, based on the Tersoff/Abell approach where the many-body term is chosen to

give the bcc structure as the preferred minimum. This is described in more detail in the next section.

The Model

The bcc unit cell is shown in figure 5.1. The 'bond' angle is defined as the angle formed by any two atoms in the bcc cell with the central-most atom.

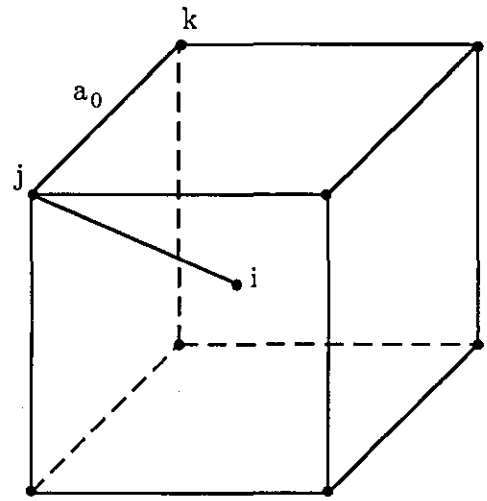


Figure 5.1

We label the central atom i and its distance from atom j is defined as r_{ij}. The angle that is subtended by atoms j and k with atom i is named θ_{jik}. Within the bcc cell there are three values of θ_{jik}, $\cos^{-1}(\frac{1}{3})$, $\cos^{-1}(-\frac{1}{3})$ and $\cos^{-1}(-1)$, corresponding to $r_{jk}=a_o$, $r_{jk}=\sqrt{2}a_o$ and $r_{jk}=\sqrt{3}a_o$ respectively. Here a_o, the lattice constant, is the length of the 9 particle unit cell and $r_{ik}=\frac{\sqrt{3}}{2}a_o$. The three angles all satisfy the equation

$$\cos(\theta_{jik}) = 1 - \frac{2}{3}\frac{r_{jk}}{a_o^2} \,. \tag{5.8}$$

The basic form of two-body interactions between atoms i and j is chosen to be similar to the 'Morse' potential, i.e.,

$$V_{ij} = Ae^{-n_2\beta r_{ij}} - Be^{-\beta r_{ij}} \tag{5.9}$$

where A, B, n_2 and β are fitting constants. This first order approximation to the interaction potential V_{ij} between two particles has a singularity as $r_{ij} \to 0$, a minimum value (dimer binding energy) at some fixed (equilibrium position) separation and $V_{ij} \to 0$ as $r_{ij} \to \infty$. The many-body nature of the potential is taken into account by modifying the attractive part of the potential by multiplying B by a function which is chosen to make the second

term decrease in magnitude away from the bcc configuration. Thus the chosen form for the total potential energy of the central-most atom i is

$$E = \frac{1}{2}\sum_{j\neq i} f_c(r_{ij})\big(Ae^{-n_2\beta r_{ij}} - Be^{-\beta(r_{ij}+\alpha_1 f_1+\alpha_2 f_2)}\big) \tag{5.10}$$

where the sum is taken over all neighbours of i. For the bcc cell i has 8 neighbours situated at the eight corners of the cube, for fcc it has 12 neighbours and for the diamond cell it has 5 neighbours. Notice that we are interested in calculating the energy of the central-most atom of the bcc cube with respect to its eight nearest neighbours. The switching function $f_c(r_{ij})$ has been chosen so that the potential drops to zero as $r_{ij} \to \infty$ from its equilibrium position. The chosen form is

$$f_c(r_{ij}) = \begin{cases} 1, & r_{ij} \leq R - D \\ 1 + \sin\left[\pi(r_{ij} - R - 3D)/(4D)\right], & R - D \leq r_{ij} \leq R + D \\ 0, & r_{ij} \geq R + D \end{cases} \tag{5.11}$$

The values of R and D are chosen as 3.06Å and 0.1Å respectively. The functions f_1 and f_2 in (5.10) are given by

$$f_1 = \sum_{k\neq i,j} r_{ij}^p f_c(r_{ik})\big(\cos(\theta_{jik}) - 1 + \frac{2}{3}\frac{r_{jk}^2}{a_o^2}\big)^2 , \tag{5.12}$$

$$f_2 = \sum_{k\neq i,j} r_{ij}^p f_c(r_{ik})(r_{jk} - \sqrt{3}a_o)^2(r_{jk} - \sqrt{2}a_o)^2(r_{jk} - a_o)^2/(1 + (r_{ik} - a_o)^6) , \tag{5.13}$$

where p is a known parameter given by

$$p = \big(\sum_{k\neq i,j}^{n_3-1} \frac{r_{ij}}{r_{ik}} - 1\big)/3 \tag{5.14}$$

and n_3 is the number of remaining nearest neighbours of the centralmost atom when the pair (i,j) is considered. For bcc, fcc and diamond lattices the value of n_3 is 7, 11 and 4 respectively. Thus the values of p for bcc, fcc and diamond lattices are $2.0, 3.33$ and 1.0 respectively. Within the bcc cell $r_{ij} = \frac{\sqrt{3}}{2}a_o$ for all j and $r_{jk} = \sqrt{2}a_o, \sqrt{3}a_o$ or a_o, where a_o is the lattice constant. Therefore the penalty functions f_1 and f_2 vanish in the equilibrium configuration and increase as we move away from the bcc structure. At the equilibrium each pair (i,j) contributes the same energy. The parameters $A, B, n_2, \beta, \alpha_1$ and α_2 must be chosen to fit the cohesive energy and other elastic properties of the material. This is achieved using least squares and global optimization.

The numerical calculations presented here, fit the data for tungsten. The free parameters are optimized to fit the cohesive energy, elastic constants and the bulk modulus of bcc materials. The objective function we have chosen, consists of the sum of squares of the differences between the required and calculated quantities. The function implicitly depends on the free parameters and is given by

$$F = (E - E')^2 + (BM - BM')^2 + (c_{11} - c_{11}')^2 + (c_{44} - c_{44}')^2 \tag{5.15}$$

where the symbols are defined as :
Required cohesive energy, $E = -8.90$ eV ,
Required bulk modulus, $BM = 1.86$ eV/Å^3 ,
Required elastic constant, $c_{11} = 3.12$ eV/Å^3 ,
Required elastic shear constant, $c_{44} = 0.94$ eV/Å^3
and the 'dashed' terms represent the calculated values of the various material properties. To reduce the number of free parameters needed in the computation, we use the following arguments. The dimer energy for most metals is not known but the cohesive energy for some metals is known, for instance the cohesive energy for tungsten (W) is -8.90 with cell length $a_o = 3.16469$Å. The model described above will give minimum energy when the penalty functions f_1 and f_2 are zero, that is, when the total potential is the sum of the individual pair potentials. We now use the additional assumption that the dimer energy = $\frac{1}{8}$cohesive energy. From the two body term

$$V = Ae^{-n_2\beta r} - Be^{-\beta r} , \tag{5.16}$$

the equilibrium distance (r_d) can be found by setting $\frac{dV}{dr}$=0, giving

$$r_d = \frac{1}{(n_2 - 1)\beta} \ln \frac{n_2 A}{B} , \tag{5.17}$$

with the corresponding dimer binding energy

$$V_d = \frac{B}{n_2}(1 - n_2)e^{-\beta r_d} . \tag{5.18}$$

This implies that n_2 must exceed 1 as $V_d < 0$. Using (5.17) we can write

$$\begin{aligned} e^{-\beta r_d} &= e^{-\frac{1}{n_2 - 1} \ln \frac{n_2 A}{B}} \\ &= \left(\frac{n_2 A}{B}\right)^{-\frac{1}{n_2 - 1}} . \end{aligned}$$

Again from (5.17) if we fix $r_d = \frac{\sqrt{3}}{2}a_o$ we get

$$A = \frac{B}{n_2} e^{\frac{\sqrt{3}}{2}a_o(n_2-1)\beta} . \tag{5.19}$$

Now (5.18) becomes

$$V_d = \frac{B}{n_2}(1-n_2)\left(\frac{n_2 A}{B}\right)^{-\frac{1}{n_2-1}} . \tag{5.20}$$

After substituting the value of A from (5.19) into (5.20) and setting

$$\text{Dimer Energy } (V_d) = \frac{1}{8}(\text{Cohesive Energy at the equilibrium distance } r_d = \frac{\sqrt{3}}{2}a_o),$$

we get

$$B = \frac{1.1125 n_2}{(n_2-1)}\left[e^{\frac{\sqrt{3}}{2}a_o(n_2-1)\beta}\right]^{\frac{1}{(n_2-1)}} . \tag{5.21}$$

By fixing n_2 and β in (5.21) values of A and B can be found from (5.19) and (5.21) making F a function of n_2, β, α_1 and α_2 only. However, preliminary numerical investigation suggests that β should be approximately 0.42 and n_2 should be approximately 6.81. Taking these values for β and n_2 and the corresponding values of A and B gives F as a function of α_1 and α_2 only and we have optimized F with respect to these parameters using all the global optimization algorithms taking $[0,1]^2$ as the search region. The optimized values of the parameters are given in Table 5.11 and the calculated values of the elastic constants and bulk modulus are shown in Table 5.12.

The optimized parameter values

Table 5.11

Parameter	Values
A	513.48133
B	4.15243
n_2	6.81547
β	0.42267
α_1	0.03082
α_2	0.00490

Bulk Modulus and Elastic Constants

From the general theory of solid state physics, we know that the bulk modulus of a cubic solid is the energy required to produce a given deformation. From the relationship

$$\text{Bulk Modulus} = \frac{c_{11} + 2 \times c_{12}}{3} \tag{5.22}$$

it is clear that if we calculate the bulk modulus and the elastic constant c_{11}, the remaining elastic constant c_{12} can be found. The bulk modulus is given by

$$\mathrm{BM} = \lim_{\delta V_p \to 0} V_p \Big[\frac{E(V_p + \delta V_p) - 2E(V_p) + E(V_p - \delta V_p)}{\delta V_p^2} \Big] \tag{5.23}$$

where V_p is the volume per particle (Ashcroft and Mermin, 1976). The expression for c_{11} is similar to bulk modulus but the shear stress,

$$c_{44} = \lim_{\phi \to 0} 2 \Big[\frac{E(\phi) - 2E(0) + E(-\phi)}{\phi^2 a_o^3} \Big] \tag{5.24}$$

where ϕ is the shearing angle (Ashcroft and Mermin, 1976) and $E(\phi)$ is the energy for the deformation due to ϕ.

Table 5.12

Correct Value		Calculated Value	
BM	1.866 eV/Å^3	BM′	1.853 eV/Å^3
c_{11}	3.126 eV/Å^3	c_{11}'	3.136 eV/Å^3
c_{44}	0.944 eV/Å^3	c_{44}'	0.958 eV/Å^3

The computations for the elastic constants and bulk modulus are carried out by fixing the central most atom at the origin and eight corners are chosen in the three dimensional space according to the lattice constant a_o. A similar scheme was also implemented for the calculation of energies of fcc and diamond lattices. For calculation of the bulk modulus the coordinates of the corner atoms are extended and contracted by a magnitude of 0.001. Similarly for c_{44} the x-coordinates of the top four atoms are extended and those of the bottom four are contracted and vice versa to form a small angle of magnitude ϕ with the vertical axis. This deformation, however does not change the volume of the cube.

Before going on to compare the relative merits of the algorithms we use the optimum values of the parameters (see Table 5.11) to test the model for different lattices. The comparison of the energetics is demonstrated in Table 5.13.

Minimum Energy

Table 5.13

Lattice	Minimum Energy	Separation (r_d)
bcc	−8.90eV	2.740Å
fcc	−8.189eV	2.954Å
Diamond	−4.45eV	2.740Å

From Table 5.13 it is clear that proposed model for bcc has its energy lower than fcc and for the diamond lattice. Table 5.12 shows that the calculated elastic constants are in good agreement with their correct values. Consequently we can infer that the model is acceptable. In the next section we compare the numerical results obtained by all algorithms.

Numerical Comparison of the Algorithms

We have implemented all algorithms with the best of the user supplied parameters described earlier. We used $N = 50$ and $\sigma = 4$ for both MSL and TMSL but we took $\gamma = 0.2$ for MSL and $g = 8$ for TMSL. We took the local search tolerance as 10^{-4} for this problem. TMSL performed 2 local searches and MSL 3 but only the global minimum was found. We summarize the results of all algorithms in Table 5.14.

Table 5.14

	CRS2	CRS3	CRS4	CRS5	ASA	SA	TMSL	MSL
FE	1327	1159	1083	1596	3582	3951	749	895
cpu	9.96	12.68	8.03	11.32	28.42	31.73	6.21	6.37
F^*	1.7E-3	2.5E-3	1.9E-3	2.2E-4	1.5E-3	2.3E-3	1.5E-4	1.1E-4

Table 5.14 shows that TMSL is the best algorithm with respect to function evaluations and cpu time and the CRS algorithms are better than the SA-type algorithms. It also shows that CRS4 performs better than the other CRS methods both in terms of the number of function evaluations and cpu time and ASA is superior to SA.

CHAPTER 6

The Optimal Control of Vehicle Suspension Systems

6.1 Introduction

This Chapter describes a model of an active (computer-controlled) vehicle suspension system and a new non-linear controller design methodology is presented. Based on the principles of optimal control, it permits the use of more general cost functions than the standard linear optimal design techniques and hence increases the freedom of the designer. It implements the control with an optimal, non-linear feedback function. Having designed this non-linear, closed loop, feedback control, simulations of the suspension system are carried out. Feedback is optimized both with and without the imposition of constraints. Comparisons between the open loop, the unconstrained closed loop and the constrained systems are given. The global optimization algorithms discussed in previous Chapters are used to minimize the underlying cost function and their performances are compared.

6.2 Design of Vehicle Suspension System

The modelling of vehicle suspension systems and the design of suspension control strategies for the purpose of giving a 'smoother' ride is a problem that has attracted much interest over the years (Thompson, 1976; Frühauf, et al, 1985 and Sharp and Crola, 1987). In the field of active (computer-controlled) suspension control strategy design, a number of suspension systems have been produced in recent years (Karnopp, 1983; Gordon, et al, 1990 and Gordon, et al, 1991). To date most of the models (Thompson, 1984; Hac, 1985 and Wilson, et al, 1986) have used linear optimal control theory to solve the optimization problem. The linear optimal control approach, or LQG (linear quadratic Gaussian) as it is commonly known, assumes unconstrained actuation of the control which is likely to be undesirable. This approach leads to a linear feedback law and a closed loop system. Although this method provides an analytical solution with relatively low computational time, it places unsatisfactory limits on system performance, because the cost function must be a quadratic function of the state and control variables (Hac, 1987, Marsh, et al, 1989 and Gordon, et al, 1990). Disadvantages of the use of a quadratic cost function are well explained in Marsh (1992). Hac (1987) introduced the idea of an adaptive linear strategy, but it is demonstrated in Gordon, et al, (1990) that this approach could provide poor performance in some situations, such as when potholes are encountered. A more general non-linear methodology which does not restrict the designer to quadratic cost functions, allows greater freedom for the expression of performance requirements and is potentially

able to produce controllers which enable inherent adaptation of the characteristics of the vehicle. This fundamental advantage of a nonlinear methodology is likely to be more effectively demonstrated for an ideal active suspension system since this offers the greatest freedom of actuation. Our purpose therefore is to solve the problem for a non-linear feedback control in order to see how the performance characteristics might be improved by using a non-quadratic cost function. The motivation and justifications for such an approach are well-explained in Marsh (1992). Recently, Gordon et al. (1990) and Gordon et al. (1991) implemented a non-linear feedback control with a non-quadratic cost function, where the control is a fifth degree polynomial in the state variables, with 91 free parameters, which are the polynomial coefficients. The problem was formulated using Pontryagin's maximum principle and a fitting method used to find the polynomial coefficients, the numerical scheme however required that optimal open loop data be generated for 4000 initial conditions which is computationally very expensive. A further drawback of their approach is that only an approximate method was used to solve the open loop problem. In addition, a controller design subject to realistic constraints on the system was not studied. We solve the open loop problem using a stable numerical method and design a nonlinear feedback control. A model of a car suspension system with realistic constraints is also examined.

6.3 Unconstrained Problem Model and Optimization

A quarter vehicle model is the simplest that can represent the dynamics of a suspension system and possesses particular advantages over more complex models (see, for example Sharp and Crolla 1987). This model is to be used to design and test the suspension system because of its simplicity and ability to model the most fundamental aspects of the system performance. The system is shown schematically in figure 6.1, which also defines the variables used in the description of the problem. The suspension system is controlled by a force generator situated between the wheel and the body. The state equations for the system are Newton's laws of motion, namely;

$$\begin{aligned} \dot{x}_1 &= -x_3 \ , \\ \dot{x}_2 &= x_3 - x_4 \ , \\ \dot{x}_3 &= (k_t x_1 - u)/m \ , \\ \dot{x}_4 &= u/M \ , \end{aligned} \tag{6.1}$$

where $x = (x_1, \ldots, x_4)$ is a vector of state variables of the system and $u(t)$ is the control input to the system.

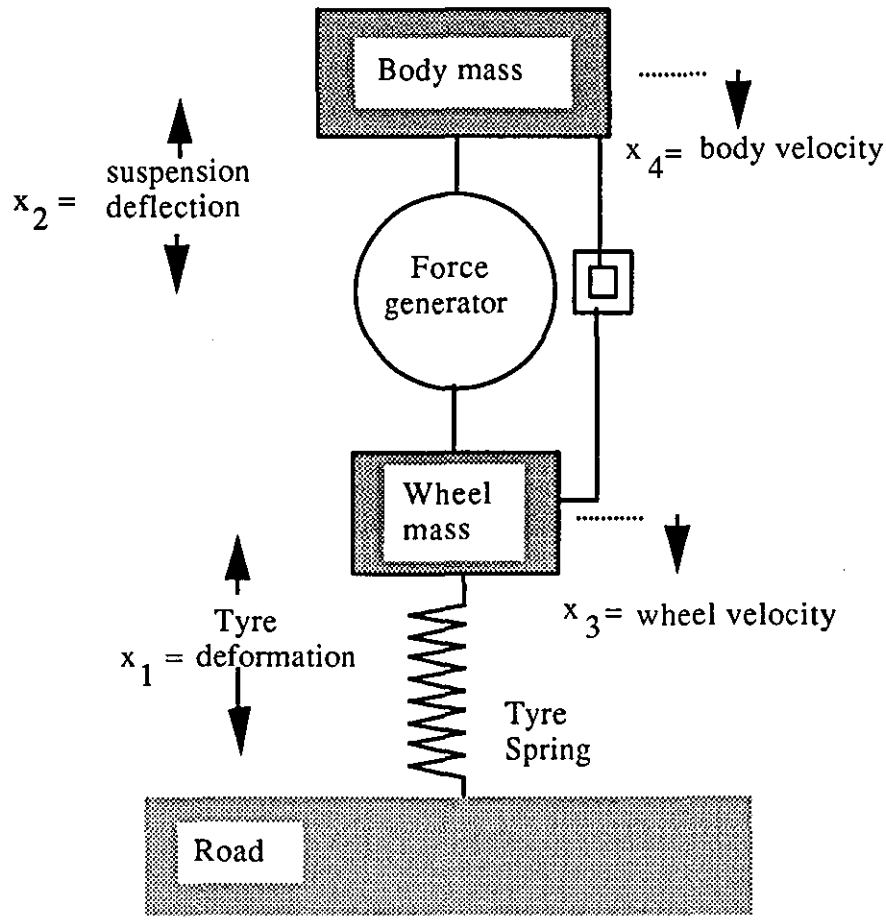


Figure 6.1

Typical values for the constants are M (body mass)=320 kg, m (wheel mass)=40 kg and k_t (tyre stiffness)=$2.0 \times 10^5\ N/m$, (see, for example, Marsh, et al, 1989). For given initial conditions the performance of the system is assessed via a cost function, $L(u,x)$, which is a function of both state variables and control. This is chosen to accumulate all the undesirable effects on the states caused by the disturbances and the costs of the control action into the a single function (Marsh, 1992). The cost function is then integrated over the time period to form the dynamic cost, I. Therefore, the dynamic cost functional to be minimized is

$$I = \int_0^{t_{max}} L(u,x)\, dt \tag{6.2}$$

with $x(0) = x_0$. $L(x,u)$ is a positive definite function of the state variables and control inputs given by,

$$L(x,u) = J_1(x_1) + J_2(x_2) + \dot{x}_4^2 \tag{6.3}$$

where

$$J_1(x_1) = \begin{cases} 4000x_1^2 & \text{if } |x_1| \leq 0.007 \\ 98500x_1^2 - 1323|x_1| + 4.6305 & \text{if } 0.007 \leq |x_1| \leq 0.009 \\ 25000x_1^2 - 1.323 & \text{if } |x_1| \geq 0.009 \end{cases}$$

$$J_2(x_2) = \begin{cases} 500x_2^2 & \text{if } |x_2| \leq 0.079 \\ 385250x_2^2 - 60790.5|x_2| + 2401.22475 & \text{if } 0.079 \leq |x_2| \leq 0.081 \\ 10000x_2^2 - 60.7905 & \text{if } |x_2| \geq 0.081 \end{cases} \tag{6.4}$$

The distances are in metres and this will be used as the unit of length subsequently. The cost functions J_1 and J_2 are quadratic splines chosen to have high values if the amplitude of the state disturbances is large. Their form is chosen to be approximately the same as that given by Gordon, et al, (1990) except that they are continuous with continuous first derivatives with respect to x_1 and x_2 respectively. The graphs of J_1 and J_2 are shown below in Figure 6.2.

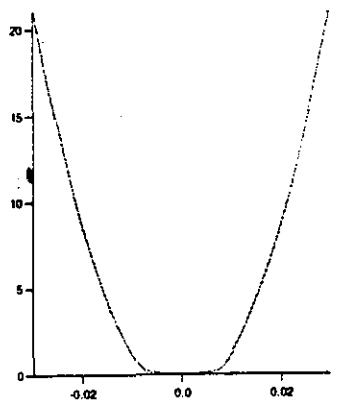


Figure 6.2(a) $J_1(x_1)$.

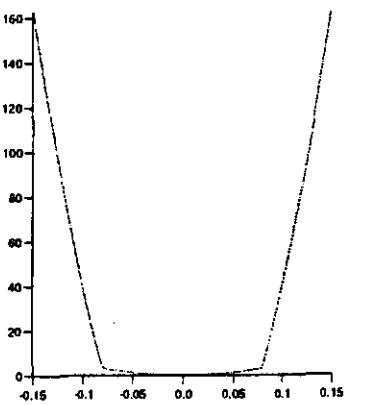


Figure 6.2(b) $J_2(x_2)$.

The cost function is chosen to depend on x_1, x_2 and x_4 because (1) the tyre deformation, x_1, should be kept small: (2) the work space between the body and the wheel should be kept constant; (3) the body acceleration should be as small as possible. This is a much more realistic cost function than that for the LQG model, which is a continuous quadratic function and cannot impose a high penalty for large amplitude disturbances. For the purpose of numerical calculation t_{max} is taken to be 2 secs. This is approximately equal to the infinite time problem (see Marsh, 1992). The workspace between the body and the wheel is limited by the design criterion. A typical workspace size is given by $-0.1 \leq x_2 \leq 0.1$. The tyre deformation is also limited and typically lies in the range $-0.025 \leq x_1 \leq 0.025$. The cost function $L(x, u)$ has been chosen with these design criteria in mind and has a very large value if x_1 and x_2 lie outside these limits. For the system (6.1) with performance index (6.2) the open loop control is determined using Pontryagin's maximum principle (see, for example, Bryson and Ho, 1969). This gives the optimal control $u(t)$ that minimizes (6.2).

6.3.1 Open Loop Optimization

We apply Pontryagin's maximum principle to find the optimum open loop costs and controls for specified initial conditions. In the usual way by introducing a 5-th state variable, we can write

$$\begin{aligned}\dot{x}_5 &= J_1(x_1) + J_2(x_2) + \dot{x}_4^2 \\ &= J_1(x_1) + J_2(x_2) + u^2/M^2 \ ,\end{aligned}$$

with $x_5(0) = 0$. This gives rise to the system

$$\begin{aligned}\dot{x}_1 &= -x_3 \ , \\ \dot{x}_2 &= x_3 - x_4 \ , \\ \dot{x}_3 &= (k_t x_1 - u)/m \ , \\ \dot{x}_4 &= u/M \ , \\ \dot{x}_5 &= J_1(x_1) + J_2(x_2) + u^2/M^2 \ .\end{aligned} \tag{6.5}$$

Now, the problem becomes : minimize

$$I = x_5(t_f) \ , \tag{6.6}$$

subject to (6.5) and $x(0) = (x_o, 0)$. The Hamiltonian H of the system is

$$H = \sum_{i=1}^{5} \lambda_i f_i \ ,$$

where

$$\dot{\lambda}_i(t) = -\frac{\partial H}{\partial x_i} \quad i = 1, \ldots, 5 \quad ,$$

and $\lambda(t_f) = (0, 0, 0, 0, 1)$. Here f_i are right hand sides of the system (6.5) and $\lambda = (\lambda_1, \ldots, \lambda_5)$ is the co-state vector. The co-state equations are therefore as follows:

$$\begin{aligned}\dot{\lambda}_1(t) &= -\lambda_5 \frac{dJ_1(x_1)}{dx_1} - \frac{\lambda_3 k_t}{m} \ , \\ \dot{\lambda}_2(t) &= -\lambda_5 \frac{dJ_2(x_2)}{dx_2} \ , \\ \dot{\lambda}_3(t) &= \lambda_1 - \lambda_2 \ , \\ \dot{\lambda}_4(t) &= \lambda_2 \ , \\ \dot{\lambda}_5(t) &= 0 \ .\end{aligned} \tag{6.7}$$

It is seen that the non-quadratic nature of the cost function causes the costate and state equations to be non-linear and hence yields a non-linear two point boundary value problem.

We solve this problem by the steepest descent approach (see, for example, Rosenbrock and Storey, 1966). We define

$$\begin{aligned} g(u) &= \frac{\partial H}{\partial u} \\ &= \frac{2u\lambda_5}{M^2} + \frac{\lambda_4}{M} - \frac{\lambda_3}{m} . \end{aligned} \tag{6.8}$$

Given an approximation $u_o(t)$ to the optimal control then a better approximation, in the sense of taking a step along the path of steepest descent of the functional $x_5(t_f)$, can be shown (Rosenbrock and Storey, 1966) to be

$$u_1 = u_o - \epsilon g(u(t)) \tag{6.9}$$

where ϵ determines the length of step taken. The line search along the negative gradient used a quadratic interpolation (see, for example, Rao, 1978) which does not use derivatives. Our numerical scheme proceeds as follows. The time interval is discretized as $0 = t_o, t_1, \ldots, t_n = 2$. With an initial guess $u_o(t)$ at the control the state equations are integrated from 0 to 2 and the co-state equations are integrated from 2 to 0. At each time stage of the integration the nominal control $u_o(t)$ is adjusted by the rule given in (6.9). The whole process is then repeated for the new control $u_1(t)$ and the process continues until the convergence criterion is met. The stopping criterion was $||g(u_i)|| < 10^{-4}$. We used an integration step $h = 0.005$ and solved the equations (6.5) and (6.7) using a Runge-Kutta 4th order method (Sanchez, et. al, 1988) with initial control $u_o(t) = 1.5$.

In practical control problems the open loop approach is very inefficient because the control is not an instantaneous function of the state variables. To determine $u(t)$ requires that a series of differential equations be solved over the time interval $(0, t_{max})$ before the control $u(t)$ can be determined and then applied. An attempt has been made here to design a feedback control to overcome the problem. There are many possible formats in which to express this feedback control (Marsh, 1992). Concern for simplicity leads us to consider the implementation of the control strategy via a continuous analytical feedback function to be known as the feedback law. This feedback law overcomes the computational problems of solving for $u(t)$ by specifying the control $u(t)$ to be a function of the state variables $u(t) = U(x(t), k)$ which contains some free parameters k which are chosen so that $u(t)$ gives optimal responses. Often $u(t)$ is chosen as a low order polynomial (Marsh, 1992) in the state variables x. The unknown coefficients in the polynomial are the parameters k. The problem here requires that the controller performs the same operation, irrespective of whether the system is displaced from its equilibrium condition through positive or negative values. Clearly the controller should be an odd function of the state variables. We have therefore chosen $U(x(t), k)$ as a 12 term polynomial, in the state variables, containing first and third degree terms only. Motivation for such a choice of the feedback function can be

found in Marsh (1992). For the car suspension problem a full state feedback requires a non-linear function $U(x(t),k)$ of 4 independent variables (2 displacements and 2 velocities). Therefore our chosen feedback is given by

$$\begin{aligned} U(x(t),k) &= k_1x_2 + k_2x_3 + k_3x_4 + k_4x_1^2x_3 + k_5x_1x_2x_3 + k_6x_2^3 \\ &+ k_7x_2^2x_4 + k_8x_2x_3x_4 + k_9x_4^2x_2 + k_{10}x_3^3 + k_{11}x_3^2x_4 + k_{12}x_4^2x_3 \end{aligned} \tag{6.10}$$

The parameters k_i in (6.10) are found by optimizing a combined cost which is the sum of the costs incurred due to several initial conditions evenly distributed throughout the state space. We have, in fact, selected four representative points $x(0)$ in the space of initial conditions and call them the design sample. To enable the feedback law to give a good representation across the practically available region of the state space, the design sample needs to provide information over all areas of the state space. The size of the design sample must be curtailed however, due to the computational cost. But clearly more points could be chosen, if necessary, at the expense of increased computing cost. The chosen set of initial conditions x_o are

$$\begin{aligned} &(0.025, 0.1, 0, 0, 0), (-0.025, 0.1, 0, 0, 0), \\ &(0.01, 0.04, 0, 0, 0), (0.01, -0.04, 0, 0, 0)\ . \end{aligned} \tag{6.11}$$

Note that inclusion of the image points $-x_o$ to distribute the initial conditions evenly in the state space will contribute the same cost as x_o and these are therefore not included here. However it will be shown, a posteriori, by simulations of the optimal feedback law that the values of the cost function calculated at other points do not differ significantly from the corresponding open loop values.

6.3.2 Closed Loop Optimization

Closed loop optimization implements the control (6.10) with chosen parameter values $k = (k_1, k_2, k_3, k_4, \ldots, k_{12})$ as opposed to open loop which pre-calculates $u(t)$ to minimize the cost. As a result the cost function depends implicitly on k. Closed loop optimization does not involve the co-state equations (6.7) and the new cost function is defined as follows:

Cost function **C**= sum of the dynamic costs incurred by using each of the four initial conditions in 6.11 whilst implementing the feedback law with parameters assigned, i.e.,

$$\mathbf{C} = \sum_{j=1}^{4} \text{cost}_j \tag{6.12}$$

where cost_j=cost with the j-th initial condition from 6.11. In order to determine the control $u(t)$ we must determine the optimal parameter set $k = (k_1, k_2, k_3, k_4, \ldots, k_{12})$. This is done by using the global optimization algorithms described in earlier Chapters. However, it was

felt that to produce numerically reasonable values for k_i the state variables should be normalized by dividing each by its maximum practically available value, thus:

$$y_1 = \frac{x_1}{0.025}, y_2 = \frac{x_2}{0.1}, y_3 = \frac{x_3}{2.5}, y_4 = x_4 \ . \tag{6.13}$$

The corresponding new state equations become

$$\begin{aligned} \dot{y}_1 &= -100y_3 \ , \\ \dot{y}_2 &= 25y_3 - 10y_4 \ , \\ \dot{y}_3 &= (0.025k_t y_1 - u)/(2.5m) \ , \\ \dot{y}_4 &= u/M \ , \\ \dot{y}_5 &= J_1(y_1) + J_2(y_2) + u^2/M^2 \ . \end{aligned} \tag{6.14}$$

The cost functions J_1 and J_2 become

$$\begin{aligned} J_1(y_1) &= \begin{cases} 2.5y_1^2 & \text{if } |y_1| \leq 0.28 \\ 61.5625y_1^2 - 33.075|y_1| + 4.6305 & \text{if } 0.28 \leq |y_1| \leq 0.36 \\ 15.625y_1^2 - 1.323 & \text{if } |y_1| \geq 0.36 \end{cases} \\ J_2(y_2) &= \begin{cases} 5y_2^2 & \text{if } |y_2| \leq 0.79 \\ 3852.5y_2^2 - 6079.05|y_2| + 2401.22475 & \text{if } 0.79 \leq |y_2| \leq 0.81 \\ 100y_2^2 - 60.7905 & \text{if } |y_2| \geq 0.81 \end{cases} \end{aligned} \tag{6.15}$$

Note that this transformation of the system does not affect the open loop cost at all and the same open loop control will be produced for a particular initial condition. The individual normalised costs, cost_j, are now found by solving the system (6.14) only with the normalised initial conditions of (6.11). We have found by extensive exploratory numerical work the following region of parameter $(k_i, i = 1, 2, \ldots, 12)$ space within which the system is well defined.

$$\begin{aligned} &[0, 2.0E3] \times [0, 2.0E3] \times [-5.0E3, 1.0E2] \times [-5.0E3, 5.0E3] \times \\ &[-6.0E3, 5.0E2] \times [0, 1.5E3] \times [-2.0E3, 2.0E3] \times [0, 5.0E3] \times \\ &[0, 1.5E3] \times [-5.0E3, 5.0E3] \times [-5.0E3, 1.0E3] \times \\ &[-5.0E3, 1.5E3] \end{aligned}$$

The above region was therefore chosen as our search region for all the subsequent calculations. Seven different local minima were found within the above region but each of these local minima have the same function value. The function C has a lowest value of 12.69. The seven individual local minimizers are given in Table 6.1 below. (Not all the local minima are found by all of the algorithms. The values in Table 6.1 are representative in those cases where more than one methods found the same local minima.)

Table 6.1

Local minimizer

i	k_i^1	k_i^2	k_i^3	k_i^4	k_i^5	k_i^6	k_i^7
1	536.12	530.20	536.32	523.09	534.49	539.90	540.38
2	539.70	621.23	520.11	601.54	596.21	541.69	576.11
3	-2080.86	-2040.57	-2098.46	-1990.09	-2066.86	-2089.35	-2076.80
4	3829.17	4080.68	3814.65	3660.40	3891.00	3921.36	3752.10
5	170.05	486.76	239.91	83.42	257.45	383.22	187.48
6	1301.01	1181.22	1282.58	1311.17	1286.93	1246.56	1244.62
7	-1941.02	-1114.49	-1989.10	-1711.17	-1870.07	-1679.93	-1404.76
8	4899.68	3768.93	4895.90	4878.77	4988.70	4120.65	3954.50
9	1242.84	31.90	1499.89	407.16	1218.00	998.65	447.10
10	3665.00	3286.64	3718.66	3261.95	3398.93	3841.81	3639.27
11	-3574.96	-2385.87	-1414.28	-3466.25	-3607.44	-2154.98	-3399.40
12	-47.73	-435.58	52.69	-194.85	-3275.65	129.79	1084.71

A set of initial conditions is chosen from the state space at which to test a controller from Table 6.1. Since all sets of parameters in Table 6.1 give rise to the same optimal cost any set k_i can be used in the feedback law. However, we have used the parameter set in column 1 of Table 6.1 in the feedback law. These results are shown in Table 6.2 where IC stands for initial conditions and OLC and CLC stand for open and closed loop cost respectively. The Table shows that the agreement is quite good in all cases.

Table 6.2

IC	OLC	CLC
(0.025,0.1,0,0)	9.51	9.67
(-0.025,0.1,0,0)	2.23	2.36
(0.01,0.04,0,0)	0.41	0.42
(0.01,-0.04,0,0)	0.22	0.22
(0.015,0.1,0,0)	6.61	6.75
(0.015,0.05,0,0)	0.76	0.80
(-0.015,0.05,0,0)	0.42	0.44
(0.025,0.05,0,0)	1.53	1.60
(0.025,-0.05,0,0)	0.88	0.95
(-0.015,0.1,0,0)	2.15	2.30
(0.02,0.09,0,0)	5.49	5.66
(0.015,-0.08,0,0)	0.91	1.00
(-0.025,0.08,0,0)	1.31	1.40

A comparison of the difference between open and closed loop costs for a variety of initial conditions

Finally in the next section a constrained model is considered where the damping is subject to a delay. The problem is similar to the closed loop problem but has an extra state variable. This problem is also solved and the results compared with the open loop and unconstrained closed loop models described above.

6.4 Constrained Model and Optimization

The optimal feedback controller given above has been calculated for a system which is not constrained by limits on the forces and in which the controller acts with immediate effect to counteract any disturbances to the system. In practice, however, the suspension system is a spring-damper combination and the controller $u(t)$ is constrained. The disturbances to the system are controlled by the damper situated between the body and the wheel while the spring acts as a support to the system. The total force therefore is the sum of the damper force F_d and the spring force $k_s x_2$, i.e., $u(t) = F_d + k_s x_2$, where the spring stiffness $k_s = 1.8 \times 10^4 N/m$. The damper force F_d is dynamically constrained and is dependent on the relative velocity between wheel and body. As the damper force F_d is constrained so is the total force $u(t)$. The constrained system model has been derived in consultation with Dr. T. J. Gordon of the Department of Transport Technology, Loughborough University. The model includes an extra state variable x_6, known as the damper current of the system which satisfies

$$\frac{dx_6}{dt} = \frac{1}{c_1}(I_o - x_6)(1 + \frac{c_1}{c_2}|I_o - x_6|) \ , \tag{6.16}$$

where I_o is known as the signal current and c_1 and c_2 are constants depending on the system in question. Typical values for c_1 and c_2 are $3 \times 10^{-2} s$ and $5 \times 10^{-3} s^2$ respectively. The signal current I_o and the damper current x_6 always lie between 0 and 1. The damper force F_d is bounded by a maximum value when $x_6 = 0$ and a minimum value when $x_6 = 1$. There is a well defined relation between the damper force F_d, the relative velocity $(x_3 - x_4)$ and the damper current. This relationship is given numerically in Table 6.3. We call this relationship the damper map P_m and it maps between the damper force and the damper current for a given relative velocity.

Table 6.3

	F_d							
$(x_3 - x_4)$	-1.5	-1.0	-0.5	-0.2	0.0	0.2	0.5	1.0
current=0.0	-2350	-1750	-1500	-800	0.0	900	1300	2400
current=0.5	-2000	-1450	-550	-250	0.0	250	650	1350
current=1.0	-1450	-900	-300	-200	0.0	200	350	800

Relationship between the damper force, the damper current and the relative velocity across the damper

This mapping between F_d, $(x_3 - x_4)$ and current is typical of a realistic automobile control problem and can be found for a real-life situation by system identification. (The data of Table 6.3 was supplied by Dr. T. J. Gordon.) Notice that for fixed values of $(x_3 - x_4)$ and current, there is a unique value of the damper force F_d. The constrained closed loop optimization problem is the same as it was for the unconstrained closed loop optimization with the same cost function, the only difference is that the system now includes an extra state equation. Thus for the constrained system the new normalized state equations are

$$\begin{aligned}
\dot{y}_1 &= -100y_3 \\
\dot{y}_2 &= 25y_3 - 10y_4 \ , \\
\dot{y}_3 &= (0.025k_t y_1 - u(t))/(2.5m) \ , \\
\dot{y}_4 &= u(t)/M \ , \\
\dot{y}_5 &= J_1(y_1) + J_2(y_2) + u(t)^2/M^2 \ , \\
\dot{y}_6 &= \frac{1}{c_1}(I_o - y_6)(1 + \frac{c_1}{c_2}|I_o - y_6|) \ ,
\end{aligned} \tag{6.17}$$

where $y_6 = x_6$. The representative set of initial conditions is the same as that taken for the unconstrained system with the exception that the value of $y_6(0)$ was chosen as 0.5, half way between the upper and lower bounds. To integrate the above system at each time step, $i = 0, 1, \ldots, n-1$, the control $u^{(i)}(t)$ is found from the state variables at the begining of the step with the assigned values of k_i. Now the damper force is found from $F_d^{(i)} = u^{(i)} - k_s x_2^{(i)}$ and the signal current is found by inverse interpolation in Table 6.3, i.e., $I_o^{(i)} = P_m^{-1}\left(x_3^{(i)} - x_4^{(i)}, F_d^{(i)}\right)$. If $I_o^{(i)} \in (0,1)$ the integration step proceeds with $u^{(i)} = F_d^{(i)} + k_s x_2^{(i)}$with a new damper force $F_d^{(i)} = P_m\left(x_3^{(i)} - x_4^{(i)}, y_6^{(i)}\right)$. If however $I_o^{(i)} \leq 0$, or ≥ 1, it is fixed ('clipped') at 0 or 1 respectively and a new $F_d^{(i)c}$ is found by interpolation in Table 6.3, i.e., $F_d^{(i)c} = P_m\left(x_3^{(i)} - x_4^{(i)}, 0 \text{ or } 1\right)$. $u^{(i)}$ is itself then found from $u^{(i)c} = F_d^{(i)c} + k_s x_2^{(i)}$ and again integration proceeds. Notice that the damper force must be obtained with the original (non-normalized) variables.

The global optimization algorithms were used to calculate the optimized parameters of the feedback law (6.10) for the constrained system. The global minimum was obtained at 13.95 and there were a number of local minima. The results are summarized in Table 6.4.

Table 6.4

Local minimizer

i	k_i^1	k_i^2	k_i^3	k_i^4	k_i^5	k_i^6	k_i^7
1	651.51	623.87	791.28	1490.98	1083.38	231.76	640.25
2	1452.86	1310.18	1071.81	674.27	1182.76	607.44	1499.52
3	-2023.49	-2072.60	-2143.48	-4799.13	-4537.21	-4937.39	-2003.49
4	-4395.83	1640.91	-3389.28	-4420.82	1382.17	-4928.69	-4543.22
5	-4456.98	-5259.94	-3518.50	-4752.03	-4739.95	-2181.32	-5750.17
6	18.56	107.60	0.0025	303.97	897.98	1458.36	0.00
7	-1732.77	-1209.61	-2000.00	-1945.80	-335.46	-1772.17	-2000.00
8	4161.25	3285.16	2921.95	1500.55	4488.5	1447.26	5000.00
9	1053.03	1061.18	213.70	503.21	422.36	361.51	1500.00
10	-2532.61	-3959.59	-619.18	3821.33	-1058.33	1823.26	-3723.96
11	-3643.94	-4369.33	-1702.10	-3138.17	-4573.44	-2962.64	-4948.54
12	217.81	240.10	628.25	489.93	-1254.67	-3401.59	1287.83
cost	13.95	13.96	13.96	14.11	14.06	14.09	13.95

We have compared the performance of the two feedbacks represented by the parameters given in column 1 in Tables (6.1) and (6.4) on the constrained system. Table 6.5 represents the comparison which shows that the constrained controller provides lower cost for almost all of the state space. In Table 6.5, CUP and CCP represent costs due to the optimized parameters for unconstrained and constrained feedback respectively on the constrained system.

Table 6.5

IC	CUP	CCP
(0.025,0.1,0,0,0,0.5)	9.91	9.89
(-0.025,0.1,0,0,0,0.5)	3.09	2.99
(0.01,0.04,0,0,0.5)	0.70	0.69
(0.01,-0.04,0,0,0.5)	0.38	0.38
(-0.01,0.04,0,0,0,0.5)	0.39	0.39
(-0.01,-0.04,0,0,0,0.5)	0.68	0.67
(-0.025,-0.1,0,0,0,0.5)	10.04	10.01
(0.025,-0.1,0,0,0,0.5)	3.04	2.97
(0.025,0.05,0,0,0,0.5)	2.29	2.26
(-0.025,-0.05,0,0,0,0.5)	2.29	2.25
(-0.025,0.05,0,0,0,0.5)	1.23	1.15
(0.025,-0.05,0,0,0,0.5)	1.28	1.14
(0.0125,0.1,0,0,0,0.5)	6.25	6.25
(-0.0125,-0.1,0,0,0,0.5)	6.33	6.32
(0.0125,-0.1,0,0,0,0.5)	2.90	2.80
(-0.0125,0.1,0,0,0,0.5)	2.92	2.84
(0.0125,0.05,0,0,0,0.5)	1.10	1.09
(-0.0125,-0.05,0,0,0,0.5)	1.07	1.06
(0.0125,-0.05,0,0,0,0.5)	0.61	0.60
(-0.0125,0.05,0,0,0,0.5)	0.63	0.62
(0.0225,0.09,0,0,0,0.5)	6.57	6.56
(-0.0225,-0.09,0,0,0,0.5)	6.63	6.61
(0.0225,-0.09,0,0,0,0.5)	2.26	2.21
(-0.0225,0.09,0,0,0,0.5)	2.30	2.23
(0.015,0.08,0,0,0,0.5)	2.95	2.94
(-0.015,-0.08,0,0,0,0.5)	2.94	2.93
(0.015,-0.08,0,0,0,0.5)	1.52	1.49
(-0.015,0.08,0,0,0,0.5)	1.57	1.53
(0.0,0.1,0,0,0,0.5)	3.92	3.76
(0.0,-0.1,0,0,0,0.5)	3.95	3.75
(0.025,0.0,0,0,0,0.5)	1.20	1.11
(-0.025,0.0,0,0,0,0.5)	1.21	1.14
(0.02,0.09,0,0,0,0.5)	5.90	5.88

A comparison between the 'constrained' controller and the 'unconstrained' controller on the constrained system

6.5 Implementation and Comparison of Algorithms

In this section we discuss the numerical applications of the global optimization algorithms described in the previous Chapters both to the unconstrained (UP) and to the constrained (CP) problems. We have implemented the algorithms with the values of the user defined parameters used in the previous Chapter, the only difference here is that the local search tolerance was 10^{-6}. All algorithms successfully located the global minima for both UP and CP. In Table 6.6 we first compare the results of the MSL and TMSL algorithms. The number of function evaluations, cpu times and the optimal values are used for comparison purposes. Preliminary runs showed that it was only necessary to run MSL and TMSL for one iteration.

The Results of MSL and TMSL

Table 6.6

MSL						TMSL						
N	γ	f^*	FE	cpu	σ	N	g	f^*	FE	cpu	σ	
100	0.2	12.69	8751	320.10	2	$10n$	n	12.69	4775	219.45	2	UP
100	0.2	12.69	7291	278.32	4	$10n$	n	12.69	3921	189.94	4	
150	0.1	12.69	5211	179.71	2	$15n$	$n+1$	12.69	3296	180.27	2	
150	0.1	12.69	4331	152.21	4	$15n$	$n+1$	12.69	3624	189.59	4	
200	0.05	12.69	4265	148.34	2	$10n$	$n+1$	12.69	3834	145.46	2	
200	0.05	12.69	3117	122.87	4	$10n$	$n+1$	12.69	2991	137.32	4	
			32966	1201.55					22441	1062.03		Total
100	0.2	13.95	10209	472.49	2	$10n$	n	13.95	6075	379.30	2	CP
100	0.2	13.95	9257	452.52	4	$10n$	n	13.95	5129	280.09	4	
150	0.1	13.95	6217	234.18	2	$15n$	$n+1$	13.95	5896	285.17	2	
150	0.1	13.95	5801	229.68	4	$15n$	$n+1$	13.95	5206	277.75	4	
200	0.05	13.95	5336	168.11	2	$10n$	$n+1$	13.95	4478	228.01	2	
200	0.05	13.95	4543	159.74	4	$10n$	$n+1$	13.95	3924	183.20	4	
			41363	1716.72					30708	1633.52		Total

From the total figures in Table 6.6 it is clear that TMSL uses fewer function evaluations and less cpu time than MSL. It is also clear from this Table that for two iterations $\sigma = 4$ has always produced the best results for both algorithms.

Since all eight algorithms successfully found the global minimum therefore in Table 6.7 we compare them only in terms of cpu time and number of function evaluations. In Table 6.7, the data for MSL is $N = 200$, $\gamma = 0.05$ and $\sigma = 4$. The data for TMSL is $N = 10n$, $g = n + 1$ and $\sigma = 4$.

Table 6.7

CRS2	CRS3	CRS4	CRS5	SA	ASA	MSL	TMSL		
7072	6658	4341	3891	14942	13647	3117	2991	UP	FE
4241	5110	4079	3902	16240	15858	4543	3924	CP	
243.5	213.2	151.4	123.6	588.00	587.64	122.87	137.32	UP	cpu
220.0	245.9	214.5	178.3	702.20	663.98	159.74	183.20	CP	

The results indicate that CRS5 is the best algorithm with TMSL and MSL close runners up. The SA-type are the worse performing algorithms both in terms of number of function evaluations and cpu time, however, ASA has always exhibited superiority over SA. For UP some numerical difficulties were encountered. This is because for a few sets of parameters the dynamic cost for UP became very high and overflow occurred. When this happened we assumed the cost to be 10^{20} corresponding to that parameter set. If at the i-th step of the Runge-Kutta method, the calculated cost exceeded 10^{20}, we assumed the final cost to be 10^{20}. Notice that this does not happen for CP, because the signal current, I_o corresponding to F_d^c is always maintained within its bounds.

CHAPTER 7

Application of Global Optimization Algorithms to some Problems in Control and Statistics

7.1 Introduction

In this Chapter we have used three more practical problems to examine the performances of the stochastic global optimization algorithms. Two of the problems are from optimal control and have arisen in the field of chemical engineering. These control problems have multiple local optima and the global optimum is sought. For these optimal control problems, we also compare the results obtained from a special kind of dynamic programming implemented by Luus (1989). The third problem is a global optimization problem which has arisen in applied statistics.

7.2 Comparative Studies and Discussion

In this section we discuss the numerical results obtained and make a critical comparison of all algorithms used on the three problems. These results have been obtained by using the same user supplied parameters as were used in Chapters 5 and 6. For the implementation of all methods, on the control problems, we discretize the time interval so that the number of time steps becomes the number of variables, n, and the constant controls used for each time step become the variables. For both control problems we used a variable step and variable order Runge-Kutta routine, D02CAF, from the NAG library for integration. The routine therefore uses constant control, $u(i-1)$, to integrate the system from time step t_{i-1} to step t_i, $i = 1, 2, \ldots, n$.

7.2.1 Tank Reactor Problem:

This is a model of a nonlinear continuous stirred tank reactor which involves two different local minima. The problem was studied by Luus and Galli (1991). The equations describing the chemical reactor are:

$$\begin{aligned} \dot{x}_1 &= -(2+u)(x_1+0.25) + (x_2+0.5)\exp\left(\frac{25x_1}{x_1+2}\right) , \\ \dot{x}_2 &= 0.5 - x_2 - (x_2+0.5)\exp\left(\frac{25x_1}{x_1+2}\right) , \\ \dot{x}_3 &= x_1^2 + x_2^2 + 0.1\,u^2. \end{aligned} \tag{7.1}$$

The control u is unconstrained and the performance index is given by

$$f = x_3(0.78) . \tag{7.2}$$

The initial condition is $x(0) = (0.09, 0.09, 0.0)$ and the interval of integration is $0 \leq t \leq 0.78$. It is required to compute the control variable $u(t)$ which will minimize the performance index $x_3(0.78)$. The problem has a global minimum $x_3(0.78) = 0.13309$ and a local minimum $x_3(0.78) = 0.24442$.

The first method we used was the CRS method. We began by checking the numerical accuracy by finding the number of numerical integration step lengths required to obtain a reasonable accuracy. The number of time steps (variables) taken were 3, 6 and 13. The results are given in Table 7.1.

Table 7.1

CRS2		CRS3		CRS4		CRS5		
FE	f^*	FE	f^*	FE	f^*	FE	f^*	P
1260	0.172	1122	0.172	487	0.173	542	0.172	3
2635	0.142	3918	0.142	1570	0.142	876	0.141	6
12136	0.135	14905	0.136	8997	0.136	1342	0.245	13

It is evident from Table 7.1 that for 13 time steps a reasonably good approximation for the global minimum is achieved, therefore, for comparison purpose we only use results obtained for this number of time steps. We now compare the performance of the CRS algorithms in Table 7.2.

Table 7.2

	FE	f^*	cpu
CRS2	12136	0.135	99.9
CRS3	14905	0.136	156.6
CRS4	8997	0.136	67.8
CRS5	1342	0.245*	33.7

* Local minimum

This Table shows that CRS5 could not find the global minimum and that the overall performance of CRS4 is much better than that of CRS2 and CRS3.

In Table 7.3 the results for the MSL algorithm for some best runs are given. From Table 7.3 it is clear that MSL successfully found the global minimum for all values of its parameters. The local search tolerance used was 10^{-6}.

Table 7.3

γ	σ	N	f^*	FE	LS	LM	cpu
0.2	2	100	0.137	9613	18	2	229.47
0.2	4	100	0.136	9613	18	2	229.47
0.2	4	50	0.135	6900	10	2	147.32
0.1	2	100	0.137	3760	8	2	97.89
0.1	4	100	0.137	3760	8	2	97.89

This Table shows that the best result was obtained when $N = 100$ and $\gamma = 0.1$ and that σ had little effect. The results for TMSL are shown in Table 7.4. For all values of g the TMSL algorithm also successfully located a reasonable approximation to the global minimum but it produced worse results for values of g that were low compared with N.

Table 7.4

g	σ	N	f^*	FE	LS	LM	cpu
n	2	50	0.148	832	1	1	16.38
n	4	50	0.148	832	1	1	16.38
$n+1$	2	50	0.148	832	1	1	16.38
$n+1$	4	50	0.148	832	1	1	16.38
6	2	100	0.155	1728	4	2	54.04
6	4	100	0.155	1728	4	2	54.04
$n+1$	2	100	0.148	882	1	1	27.57
$n+1$	4	100	0.148	882	1	1	27.57

If we compare the results of Table 7.3 and 7.4, we see that both TMSL and MSL located global minima with MSL the more accurate of the two. However, in terms of cpu time and the number of function evaluations TMSL is much better than MSL.

We tried next the SA algorithm. In the solution generation mechanism, Alternative B (see section 3.3.2 in Chapter 3) Dekkers and Aarts (1991) suggested $t_o = 0.75$, however, to see the effect of t_o on this problem we examined several values of t_o. The results are given in Table 7.5.

Table 7.5

f^*	FE	t_o	cpu
0.145	157931	0.75	2382.11
0.173	129675	0.85	1853.93
0.204	52294	0.95	639.19
0.204	21025	0.99	211.52

From this Table it is clear that as the number of local searches decreases the number of function evaluations and cpu time (to satisfy the stopping condition) decrease but the accuracy of the solution falls off. $t_o = 0.75$ does seem a reasonable compromise. Finally we used the ASA algorithm with $t_o = 0.75$ and the best results of ASA together with the results of the other methods are summarized in Table 7.6.

Final Comparison of the best Results found

Table 7.6

	CRS2	CRS3	CRS4	CRS5	SA	ASA	MSL	TMSL	DP
FE	12136	14905	8997	1342	157931	101357	6900	832	-
cpu	99.9	156.6	67.8	33.7	2382.1	1380.9	147.3	16.3	44.7
f^*	0.135	0.136	0.136	0.245*	0.141	0.144	0.135	0.148	0.134

* Local minimum

From above results it is clear that the minimum number of function evaluations and the cpu time were obtained by TMSL but in terms of the accuracy of the solution it is not quite so good as MSL and the CRS methods. The CRS5 method failed to locate the global minimum and CRS4 was easily the best of the other successful CRS methods. The overall performance of ASA is better than SA but in terms of the accuracy the SA-type methods are worse than MSL and the CRS methods. We also used the iterative dynamic programming procedure, DP, (see Appendix 7A) designed by Luus (1989). The results obtained are quite good, especially in terms of accuracy but these results are deceptive since to obtain them a lot of preliminary work is needed for the determination of the appropriate values of the parameters involved (Luus, et al, 1991).

7.2.2 Bifunctional Catalyst Reactor Problem:

This is a difficult optimal control problem, with a multiplicity of local maxima, originally discussed by Luus, et al, (1991). A chemical reactor with a bifunctional catalyst is described by the following 7 differential equations:

$$\begin{aligned}
\dot{x}_1 &= -k_1 x_1 \\
\dot{x}_2 &= k_1 x_1 - (k_2 + k_3) x_2 + k_4 x_5 \\
\dot{x}_3 &= k_2 x_2 \\
\dot{x}_4 &= -k_6 x_4 + k_5 x_5 \\
\dot{x}_5 &= k_3 x_2 + k_6 x_4 - (k_4 + k_5 + k_8 + k_9) x_5 + k_7 x_6 + k_{10} x_7 \\
\dot{x}_6 &= k_8 x_5 - k_7 x_6 \\
\dot{x}_7 &= k_9 x_5 - k_{10} x_7
\end{aligned} \tag{7.3}$$

The initial condition is $x(0) = (1,0,0,0,0,0,0)$, the integration interval is $0 \leq t \leq 2000$, the rate constants are cubic functions of the catalyst blend u, given by

$$k_i = c_{i1} + c_{i2}u + c_{i3}u^2 + c_{i4}u^3, \qquad i = 1,2,\ldots,10, \tag{7.4}$$

where the constants c_{ij} can be found in Luus, et al, (1991) and in Appendix 7B. It is required to compute $u(t)$ so that $x_7(2000)$ is maximized for values of u satisfying the constraints

$$0.60 \leq u \leq 0.90\ . \tag{7.5}$$

The performance index to be optimized is given by

$$f = x_7(2000)\ . \tag{7.6}$$

For convenience we define

$$f = 10^3 \times x_7(2000)\ . \tag{7.7}$$

Luus et al showed, using recursive quadratic programming, that the problem has 25 local maxima and then went on to find the global maximum $f^* = 10.094$ using iterative dynamic programming. In order to solve this optimal control problem, in all applications 10 time stages are used. Therefore, there are 10 equal sections each of length 200 and piecewise constant controls in each section $u(0), u(1), \ldots, u(9)$ to maximize the performance index are sought. (Ten time stages were found to be sufficiently accurate by Luus, et al (1991)). In Table 7.7 we compare the CRS algorithms.

Table 7.7

	FE	f^*	cpu
CRS2	210450	10.06*	401779
CRS3	200339	9.97*	411623
CRS4	-	-	-
CRS5	119373	10.05*	313210

* Local maximum, - Results not available

This Table shows the best results for the CRS algorithms which all failed to find the global maximum. However, both CRS2 and CRS5 found the second best local maximum and in terms of cpu time CRS5 is much more efficient than CRS2. For this problem CRS4 completely failed to converge.

We now compare the performances of the MSL and TMSL algorithms. The results of some best runs are given in Tables 7.8 and 7.9. The local search tolerance was 10^{-7}.

Results of MSL

Table 7.8

γ	σ	N	f^*	FE	LS	LM	cpu
0.2	2	100	9.90*	7613	19	9	30112.50
0.2	4	100	9.90*	6521	19	9	27106.10
0.05	4	500	9.90*	11613	21	12	35495.30
0.05	4	1000	10.05*	20759	43	17	46635.40
0.02	2	2000	9.99*	10621	35	18	41758.40
0.02	2	1200	9.99*	6529	22	14	27305.90
0.02	2	1000	10.04*	5393	19	13	22386.70

* Local maxima

Clearly none of the runs for MSL could find the global maximum for this problem. However, the second best minimum was obtained when $\gamma = 0.05$ and $\sigma = 4$. The results for TMSL show that it does obtain the global maximum

Results of TMSL

Table 7.9

g	σ	N	f^*	FE	LS	LM	cpu
n	2	150	10.09	5798	15	11	13882.0
n	4	150	10.09	5798	15	11	13882.0
7	2	175	10.09	8915	23	12	21397.8
7	4	175	10.09	8915	23	12	21397.8
n	4	250	10.09	10336	26	15	25488.5
7	2	200	10.09	10484	26	13	24899.9

for all runs. Moreover, in terms of cpu time and FE TMSL is much better than MSL. To try to make the comparison more fair we took $N = 150$ and $\sigma = 2$ for both MSL and TMSL, $\gamma = 0.2$ for MSL and $g = 6$ for TMSL and ran both algorithms and the results are shown in the Table 7.10. The results for MSL are the average of four runs, 2 of which produced local maxima of 9.90 and 9.86 and the other two the correct global maximum.

Table 7.10

	N	FE	cpu	f^*
MSL	150	11252	28676	10.094 (9.90,9.86)
TMSL	150	6013	14361	10.094

This Table shows that TMSL still exhibits superiority in both FE and cpu time.

We show the effect of t_o on the SA algorithm, for this problem, in Table 7.11.

Table 7.11

f^*	FE	t_o	cpu
9.641*	171543	0.75	414400
9.027*	123900	0.85	296317
7.895*	27890	0.95	92939
9.645*	13618	0.99	38836

* Local maximum

Table 7.11 indicates that the effect of local search on accuracy is not so clear for this problem as it was for the first control problem. However the more local searches the greater the number of function evaluations and $t_o = 0.75$ still seems a reasonable value to use.

The results of all methods are now summarized in Table 7.12.

Final Comparison of Best Results Found

Table 7.12

	CRS2	CRS3	CRS4	CRS5	SA	ASA	MSL	TMSL	DP
FE	210450	200339	-	119373	171543	166317	20759	5798	-
cpu	401779	411623.5	-	313210.0	414400	391517	46635.4	13882.0	1622.3
f^*	10.06*	9.97*	-	10.05*	9.641*	9.640*	10.05*	10.094	10.094

* Local maxima

From this Table it is clear that the minimum number of function evaluations and cpu time were obtained by TMSL. Moreover, TMSL was the only algorithm to successfully find the global maximum. MSL, CRS2 and CRS5 found the second best maximum MSL the number of function evaluations is very high for CRS2 and CRS5 with CRS5 the better of the two. Both SA and ASA produced the same maxima but in terms of FE and cpu time ASA is better than SA. Again we tried dynamic programming and it successfully found the global maximum with the best cpu time but much work was needed to determine the values of some parameters as in the previous problem.

7.2.3 Pig-Liver Likelihood Function

This example arises from a statistical analysis of the elimination rates of flowing substrates in pigs liver. The problem is to estimate the parameters of a model of steady-state elimination by the standard statistical procedure of maximum likelihood estimation. The full details of the mathematical model are given in Robinson, et al.(1983). Experimental measurements of elimination rate, V_{ij} (for j-th experiment on the i-th pig-liver), on 5 pig livers, each measured under four or five different conditions are given in table 7.13. The statistical model fitted to this data has 12 parameters, only two of which are of interest,

the other 10 are nuisance parameters. This model was first investigated by Robinson, et al.,(1983), who considered a Bayesian approach to the problem, using uninformative priors, and obtained the marginal posterior densities of the two main parameters of interest by quadrature methods. The two important parameters of interest are the Michaelis constant for the enzyme-substrate interaction (k_m) and the coefficient of variation for the properties of the capillaries assumed to make up the liver (ϵ^2). The 10 nuisance parameters are, for each pig-liver i, the standard deviation σ_i of $\ln V_{ij}$ and the maximum elimination rate, $V_{\max_i}$, of the whole liver.

Table 7.13

Experimental values of V_{ij}

Liver 1	0.09	0.23	0.23	0.33	0.38
Liver 2	0.05	0.11	0.17	0.24	0.35
Liver 3	0.26	0.36	0.55	0.57	
Liver 4	0.15	0.21	0.36	0.41	0.41
Liver 5	0.16	0.33	0.67	0.70	0.74

Therefore, the log-likelihood function to be maximized can be written as

$$f\left(k_m, \epsilon^2, \sigma_1, \ldots, \sigma_5, V_{\max_1}, \ldots, V_{\max_5}\right) = -\sum_{i=1}^{5}(n_i \ln \sigma_i + R_i^2/(2\sigma_i^2)) \qquad (7.8)$$

where n_i is the number of experiments on the i-th pig-liver (see Schagen, 1986). The data is analyzed using Bayesian statistical techniques with normal errors assumed i.e. the error distribution of $\ln V_{ij}$ is assumed to be normal (Robinson, et al.,1983). Therefore,

$$R_i^2(V_{\max_i}, k_m, \epsilon^2) = \sum_{j=1}^{n_i}\left(\ln V_{ij} - \ln \hat{V}_{ij}\right)^2 \qquad (7.9)$$

where $\hat{V}_{ij}$=modelled value for j-th experiment on the i-th pig-liver, based on $k_m, \epsilon^2, V_{\max_i}$, etc and again n_i is the number of experiments on the i-th pig-liver. The model values of $\hat{V}_{ij}$ are found by solving the following non-linear equation

$$\frac{V_{\max_i}}{\hat{V}_{ij}}\left(1 - \left(\epsilon^2 \frac{V_{\max_i}}{2F_i k_m}\right) \Big/ \left[1 + \frac{\hat{V}_{ij}/F_i k_m}{\exp\{(V_{\max_i} - \hat{V}_{ij})/F_i k_m\} - 1}\right]^2\right) = \frac{k_m}{\hat{c}_{ij}} + 1 \qquad (7.10)$$

where, $\hat{c}_{ij}$, the logarithmic average of c_i and $\bar{c}_o$, is given by

$$\hat{c}_{ij} = \frac{c_i - \bar{c}_o}{\ln(c_i/\bar{c}_o)} \quad , \qquad (7.11)$$

F_i, c_i and $\bar{c}_o$ are known constants whose values are given in Table 7.14. It is clear from (7.10) that $\hat{V}_{ij}$ is no longer expressible explicitly and must be determined by numerical solution of (7.10) with nonzero ϵ^2.

Table 7.14

Experiment	F						
2	1.03	c_i	0.14	0.43	0.46	1.69	3.69
		$\bar{c}_o$	0.04	0.20	0.23	1.30	3.20
4	0.94	c_i	0.08	0.15	0.28	0.43	1.65
		$\bar{c}_o$	0.02	0.04	0.09	0.17	0.95
6	0.96	c_i	0.36	0.63	2.63	26.5	
		$\bar{c}_o$	0.09	0.26	1.92	26.0	
8	1.14	c_i	0.23	0.36	0.80	1.47	4.12
		$\bar{c}_o$	0.08	0.13	0.48	1.14	3.78
9	1.22	c_i	0.16	0.35	2.0	18.4	24.4
		$\bar{c}_o$	0.04	0.13	1.53	18.0	23.6

Robinson, et al.,(1983) integrated out the nuisance parameters and found marginal posterior modes at $k_m = 0.225$ and $\epsilon^2 = 0.165$. Schagen (1986) considered an alternative, maximum likelihood, approach, finding a maximum likelihood value of 23.264 by optimizing the full 12 parameter likelihood function over $[0,1]^{12}$. Schagen found that the maximum likelihood estimates of these parameters of interest are somewhat different from the marginal posterior modes and means obtained by Robinson, et al.,(1983). The best maximum value he found was 23.983 in a reduced region where the ranges of variables taken were $V_{\max_i} \in [0.1, 1]$ and $\sigma_i, k_m, \epsilon^2 \in [0.5, 1]$. Hence it would seem that nuisance parameters have a considerable effect on the maximum likelihood estimates of the parameters of interest. We used our methods to check this conclusion by recalculating the maximum likelihood estimates. In fact we found that this 12-parameter-optimization problem has a global maximum value 59.84 with a number of local maxima. However, we used the region $[0.03, 1]^{12}$ as the region of optimization in all implementations. Schagen's results are given in table 7.15. Noticeably Schagen's estimates of the parameters of interest are significantly different from those of Robinson et al indicating that the nuisance parameters may have some influence on the estimates.

Table 7.15

$k_m = 0.186,\ \epsilon^2 = 0.239,\ f^* = 23.983$

i	σ_i	$V_{\max_i}$
1	0.046	0.391
2	0.448	0.367
3	0.739	0.539
4	0.440	0.416
5	0.736	0.732

All CRS algorithms have been applied to this problems and two different local maxima and the global maximum have been found. The result are given in Table 7.16.

Table 7.16

	FE	f^*	cpu
CRS2	42796	54.67*	212.6
CRS3	272790	59.84	1350.3
CRS4	11291	59.84	63.8
CRS5	8367	59.84	57.0

* Local maximum

This Table shows that CRS2 could not find the global maximum, CRS5 is much more efficient than the other methods and the overall performance of CRS4 is much better than that of CRS2 and CRS3.

In the next Table we give the results of MSL for several runs. We took the local search tolerance for this problem as 10^{-5}.

Table 7.17

γ	σ	N	f^*	FE	LS	LM	cpu
0.2	4	100	59.84	8847	14	1	45.60
0.2	2	100	59.84	9531	15	1	47.87
0.1	4	100	59.84	4443	7	1	22.85
0.1	2	100	59.84	4443	7	1	23.15

Table 7.17 shows that the global minimum was obtained for all runs and the best results were achieved for $\gamma = 0.1$ and $\sigma = 4$. The results of TMSL for this problem are given in Table 7.18. As before TMSL successfully located the global minimum with fewer function evaluations and less cpu time. Surprisingly however the effect of g and σ were not significant.

Table 7.18

g	σ	N	f^*	FE	LS	LM	cpu
n	2	100	59.84	2011	4	2	21.85
n	4	100	59.84	2011	4	2	21.85
$n+1$	2	100	59.84	2011	4	2	21.85
$n+1$	4	100	59.84	2011	4	2	21.85

Finally, we used the SA algorithm. The results are shown in Table 7.19. For all values of t_o the global minimum was obtained. Once again FE and cpu time decrease as t_o increases but the effect of t_o on accuracy is less marked than before with $t_o = 0.85$ being the best value.

Table 7.19

f^*	FE	t_o	cpu
59.80	268568	0.75	1396.56
59.81	131222	0.85	686.74
59.79	96634	0.95	502.31
58.69	103321	0.99	511.49

In Table 7.20 the results of all methods are summarized. This Table shows that within the CRS algorithms CRS5 is the best and in the overall comparison TMSL and SA are respectively the best and the worse algorithms.

Final Comparison of Best Results found

Table 7.20

	CRS2	CRS3	CRS4	CRS5	SA	ASA	MSL	TMSL
FE	42796	272790	11291	8367	268568	89265	4443	2011
cpu	212.6	1350.3	63.8	57.0	1396.5	572.3	22.8	21.8
f^*	54.67*	59.84	59.84	59.84	59.80	59.84	59.84	59.84

* Local maxima

For this problem, the model values of $\hat{V}_{ij}$ together with maximum likelihood estimates are given in Table 7.21 and 7.22 respectively. The maximized likelihood value found is clearly superior to that found by Schagen's routine. The estimates of the parameters of interest k_m and ϵ^2 are more closely in agreement with the estimates of Robinson et al, indicating that integrating out the nuisance parameters does not have a significant effect on the values of these estimates. In our investigation of the problem, none of the local maxima found by Schagen was located. The only maxima found were 59.84, 54.67 and -23.025. The maximum 54.67 was located by CRS2 and that of -23.025 was located by TMSL.

Table 7.21

Model values ($\hat{V}_{ij}$)

Liver 1	0.093	0.217	0.228	0.345	0.374
Liver 2	0.058	0.1003	0.165	0.222	0.3362
Liver 3	0.2379	0.3655	0.548	0.598	
Liver 4	0.1612	0.2147	0.3313	0.3884	0.4328
Liver 5	0.1657	0.3193	0.6516	0.729	0.7312

Table 7.22

$k_m = 0.224,\ \epsilon^2 = 0.168\ f^* = 59.84$

i	σ_i	$V_{\max_i}$
1	0.035	0.398
2	0.091	0.428
3	0.051	0.604
4	0.06	0.457
5	0.031	0.738

During the optimization, the model values $\hat{V}_{ij}$ are found by solving (7.10) which is used in (7.9) to evaluate each function value defined by (7.8). However, the $\hat{V}_{ij}$ have to be determined by numerical solution of the transcendental equation (7.10). The data needed in solving (7.10) are $\hat{c}_{ij}$ and F_i. The values of $\hat{c}_{ij}$ are given by (7.11). Therefore, the data values are available from Tables 7.13 and 7.14. A PASCAL function for the calculation of function values for the pig-liver function including a PASCAL subroutine for solving equation (7.10) is given in appexdix 7C.

CHAPTER 8

Conclusion

8.1 Conclusion

In this thesis some recent stochastic global optimization algorithms have been studied and modifications have been proposed. The modified algorithms have been tested on some well-known test problems as well as on a number of practical problems. In this chapter we summarize our conclusions and briefly indicate some areas for future research.

We have proposed a new algorithm, TMSL, which combines MSL with the topographical global optimization algorithm. The main differences between MSL and TMSL are that instead of using pseudo-random numbers in sampling the search region, as in MSL, TMSL uses a Halton sequence for sampling and instead of using sample reduction to find the starting points for local searches, TMSL uses a topograph to find graph minima and then carries out local searches from a subset of these. The effects of the user supplied parameters for TMSL have been investigated and suggestions for their selection have been given. TMSL was found to be much superior to MSL in terms of the number of function evaluations but not so competitive in terms of cpu time. This is because of the extra work required for finding the graph minima in the TMSL algorithm. We have found that a great advantage of TMSL is that it can avoid finding unnecessary local minima whose function values are higher than the global minimum value.

The conventional SA algorithm cannot memorize the best solution during its execution. In ASA we have introduced a self-regulatory mechanism so that the best solution is retained. This mechanism adapts the cooling schedule in such a way that the lengths of the Markov chains and the rate of decrement of the temperature can both vary. We have also incorporated a criterion in the stopping condition which may allow the temperature to increase its value to a certain level. The effects of this re-annealing have been investigated and we have found that this feature is an important attribute of the ASA algorithm. We have clearly demonstrated the marked superiority of ASA over SA.

The new CRS4 algorithm modifies the CRS algorithm by introducing the Hammersley sequence for sampling and a periodic feature for generating a small number of points using a β-distribution whenever a best point is evolved through the CRS2 algorithm. We have also proposed the CRS5 algorithm which replaces a simplex-type local search in the CRS3 algorithm with a gradient-based local search. We have carried out various implementations of the CRS4 and CRS5 algorithms and have found that in each case they are much superior to their original versions both in terms of cpu time and the number of function evaluations.

We have also investigated the accuracy of the final solution which has suggested that there is a need to incorporate a local search, as in CRS5, to refine the final solution of the non-gradient CRS algorithms.

We have also judged the importance of the new algorithms on a number of practical problems. In our opinion a real conclusion is very difficult to draw without knowing a great deal about the problems. However our preliminary investigations have shown that for problems with a small to a moderate number of local minima TMSL is the best algorithm followed by MSL. Again this conclusion has to be considered with caution as one may encounter numerical difficulties for noisy functions.

For problems with a large numbers of local minima, for example the problem considered in Chapter 5, TMSL is still better than MSL. However, for this problem the performances of the CRS algorithms, especially the CRS4 algorithm, were satisfactory, especially in terms of accuracy of the final solutions. However, CRS4 always achieved an overall superiority over the rest of the CRS algorithms. Clearly the CRS algorithms may be preferable to TMSL and MSL for problems with many minima and problems which are discontinuous and/or extremely noisy. It is therefore clear that there are circumstances in which the CRS4 algorithm could have an important role to play.

Additional difficulties with the MSL and TMSL algorithms are that they have not only to perform multiple local searches but also they have to store all distinct local minima and minimizers obtained. This becomes expensive both in terms of cpu time and storage for problems with a large number of local minima. The CRS algorithms can partially overcome this drawback as CRS4 does not perform local searches at all and CRS5 only needs to perform a complete local search to refine the final solution. However the CRS algorithms are purely heuristic algorithms. For this reason the ASA algorithm, which is superior to SA, may be preferable, firstly because even at low temperatures the algorithm remains exploratory and secondly the amount of data that has to be stored is small. In essence, therefore, ASA may be preferable for problems where the user has little knowledge about their complexity.

Research could be continued in several directions such as, the choice of user supplied parameters for TMSL and MSL especially for practical problems and the effect of different stopping conditions and their possible improvement. The stopping condition for the CRS algorithms also remains an important research area. Research could also be continued towards the derivation of a more appropriate cooling schedule for ASA.

There is also a need to thoroughly investigate the practical problems we have considered in this thesis and consider other practical problems to further test the algorithms. Finally, we claim that we have clearly shown that some of the best of the recent stochastic global optimization algorithms can be substantially improved even on very complex practical problems.

References

References

Aarts, E.H.L. and van Laarhoven, P.J.M. (1985), A new Polynomial-Time Cooling Schedule, Proceedings IEEE International Conference on CAD, ICCAD-85, Santa Clara, CA, November, 1985, pp 206-208.

Aarts, E.H.L. and van Laarhoven, P.J.M. (1985a), Statistical Cooling: A General Approach to Combinatorial Optimization Problems, Philips Journal of Research, Vol-40, pp 193-226.

Aarts, E.H.L. and Korst, J.H.M. (1989), Simulated Annealing and Boltzmann Machines, John Wiley and Sons Ltd., New York.

Ali, M.M. and Smith, R.(1993), The Structure of small Clusters Ejected by Ion Bombardment of Solids, Vacuum, Vol.44, Number 3/4, pp 377-379.

Ali, M.M. and Storey, C.(1994), Topographical Multilevel Single Linkage, to appear in the Journal of Global Optimization, 1994.

Ali, M.M. and Storey, C.(1994a), Aspiration based Simulated Annealing Algorithm, to be submitted to the Journal of Global Optimization, September, 1994.

Ali, M.M. and Storey, C. (1995), Modified Control Random Search Algorithms, International Journal of Computer Mathematics, Vol.54, Number 3/4.

Aluffi-Pentini, F., Parisi, V. and Zirilli, F. (1985), Global Optimization and Stochastic Differential equations, Journal of Optimization Theory and Applications, Vol.47, pp 1-16.

Anily, S. and Federgruen, A. (1987), Simulated Annealing methods with General Acceptance Probabilities, Journal of Applied Probability, Vol.24, pp 657-667.

Ashcroft, A.S. and Mermin, D.S. (1976), Solid State Physics, Academic Press, London.

Boender, C.G.E. (1984), The Generalised Multinomial Distribution: A Bayesian Analysis and Applications, Ph.D. Dissertation, Erasmus University Rotterdam.

Boender, C.G.E. and Rinnooy Kan, A.H.G.(1983), A Bayesian Analysis of the Number of Cells of a Multinomial Distribution, The Statistician, Vol.32, pp 240-248.

Boender, C.G.E. and Rinnooy Kan, A.H.G.(1985), Bayesian Stopping Rules for a Class of Stochastic Global Optimization Methods, Technical Report, Econometric Institute, Erasmus University Rotterdam.

Boender, C.G.E. and Rinnooy Kan, A.H.G.(1987), Bayesian Stopping Rules for Multistart Global Optimization Methods, Mathematical programming, Vol.37, pp 59-80.

Bohachevsky, M.E., Johson, M.E. and Stein, M.L.(1986), Generalized Simulated Annealing for function Optimization, Technometrics, Vol.28, Number 3, pp 209-217.

Branin, F.H. (1972), Widely Convergent Methods for Finding Multiple Solutions of Simultaneous Nonlinear Equations, IBM Journal of Research Developments, pp 504-522.

Branin, F.H. and Hoo, S.K. (1972), A Methods for Finding Multiple Extrema of a Function of n Variables, in: Lootsma, F.A., (eds.), Numerical Methods of Nonlinear Optimization, Academic Press, London, pp 231-237.

Bremermann, H. (1970), A method of unconstrained global optimization, Mathematical Biosciences, Vol.9, pp 1-15.

Brenner, D. W. (1989), Relationship between the Embedded-Atom Method and Tersoff Potentials, Physics Review Letter, Vol.63, No.9, pp 1022.

Brenner, D. W. (1990), Empirical Potential for Hydrocarbons for use in Simulating the Chemical Vapor Deposition of Diamond films, Physics Review B., Vol.42, No.15, pp 9458-9471.

Brooks, S.H. (1958), Discussion of random methods for locating surface maxima, Operations Research, Vol.6, pp 244-251.

Bryson, A., E. and Ho, Y. C. (1969), Applied Optimal Control, Blaisdell, Waltham Mass..

Catthoor, F., de Man, H. and Vandewalle, J. (1988), SAMURAI: A general and efficient Simulated Annealing Schedule with fully Adaptive Annealing Parameters, Integration, the VLSI Journal, Vol.6, pp 147-178.

Cheng R.C.H. (1978), Generating Beta Variates with Nonintegral Shape Parameters, Communications of the ACM, Vol.21, Number 4, pp 317-322.

Chiang, T.S., Hwang, C.R. and Sheu, S.J. (1987), Diffusion for Global Optimization in $\mathbb{R}^n$, SIAM Journal of Control and Optimization, Vol.25, Number 3, pp 737-753.

Chiang, T.S. and Chow, Y. (1988), On Eigenvalues and Annealing Rates, Mathematics of Operations Research, Vol.13, pp 508-511.

Collins, N.E., Eglese, R.W. and Golden, B.L. (1988), Simulated Annealing - An Annotated Bibliography, American Journal of Mathematical and Management Sciences, Vol.8, No.3/4, pp 209-308.

Byrd, R.H., Eskow, E. and Schnabel, R.B. (1992), A new large-scale global optimization method and its application to Lennard-Jones problems, Department of Computer Science, University of Colorado at Boulder, Report No. CU-CS-630-92.

Connolly, D.T. (1988), An improved Annealing Scheme for the QAP, European Journal of Operational Research, Vol.46, pp 93-100.

Daw, M.S. and Baskes, M.I. (1983), Semi-empirical, Quantum Mechanical Calculation of Hydrogen Embrittlement in Metals, Physics Review Letter, Vol.50, No.17, pp 1285-1288.

Daw, M.S. and Baskes, M.I. (1984), Embedded-atom method : Derivation and application to impurities, surface and other defects in Metals, Physics Review B, Vol.29, No.12, pp 6443-6453.

De Biase, L. and Frontini, F. (1978), A Stochastic Method for Global Optimization : its structure and numerical performance, in Towards Global Optimization 2, Dixon, L.C.W., and Szegö, G.P. (eds.), North-Holland, Amsterdam, Holland, pp 85-102.

Dekkers, A. and Aarts, E. (1991), Global Optimization and Simulated Annealing, Mathematical programming, Vol.50 pp 367-393.

Dixon, L.,C.W. and Szegö, G.P., (eds.), (1978), Towards Global Optimization 2, North-Holland, Amsterdam, Holland.

Dugundji, J. (1966), Topology, Publisher: Allyn and Bacon, Boston.

Edwards, S. (1983), New Ideas on Old Problems Emerging from Statistical Mechanics, Institute of Mathematics and Its Applications, Vol.19, pp 162-164.

Evtushenko, Yu.G. (1971), Numerical methods for finding global extrema, USSR Comp. Math. and Math. Phys, Vol.11, pp 1390-1403.

Faigle, U. and Schrader, R. (1988), On the Convergence of Stationary Distributions in Simulated Annealing Algorithms, Information Processing Letters, Vol.27, pp 189-194.

Floudas, A. and Pardalos, M.: (eds.), (1992), Recent Advances in Global Optimization; Princeton University Press, Princeton, USA.

Foiles, S.M. (1985), Calculation of the surface segregation of Ni-Cu alloys with the use of the embedded-atom method, Physics Review B, Vol.32, No.12, pp 7685-7693.

Frühauf, F., Kasper, R., and Lückel, J. L. (1985), Design of an Active Suspension for a Passenger Vehicle Model Using Input Processes with Time Delays, Vehicle System Dynamics, Vol.14, Numbers 1-3 , pp 115-120.

Garrison, B. J., Winograd, N., Deavon, D.M, Reimann, C.T., Lo, D. Y. Tombrello, T. A. , Harrison, D. E. Jr. and Shapiro, M. H. (1988), Many-body Embedded-atom Potential for describing the Energy and angular distributions of Rh atoms distorted from ion-bombarded Rh{111}, Physics Review B., Vol.37, No.13, pp 7197-7204.

Geman, S. and Hwang, C.R. (1986), Diffusion for Global Optimization, SIAM Journal of Control and Optimization, Vol.24, Number 5, pp 1031-1043.

Gidas, B. (1985), Global Optimization via the Langevin Equation, Proceedings 24th IEEE Conference on Decision and Control, Fort Lauderdale, FL, December 1985, pp 774-778.

Glover, F. (1989), Tabu Search-Part I, ORSA Journal on Computing, Vol.1, Number 3, pp 190-204.

Glover, F. and Greenberg, H.J. (1989), New Approaches for Heuristic Search: A Bilateral Linkage with Artificial Intelligence, European Journal of Operational Research, Vol.39, pp 119-130.

Goldstein, A.A. and Price, J.F. (1971), On descent from local minima, Mathematics of Computation, Vol.25, pp 569-574.

Gordon, T. J, Marsh, C., and Milsted, M. G. (1990), Control Law Design for Active and Semi-active Automobile Suspension Systems, VDI International Congress: Numerical Analysis in Automotive Engineering, Würzburg, 1990, VDI Berichte Vol. 816, pp 537-546.

Gordon, T. J, Marsh, C., and Milsted, M. G. (1991), A Comparison of Adaptive LQG and Nonlinear Controllers for Vehicle Suspension Systems, Vehicle System Dynamics, Vol.20, Number 6, pp 321-340.

Greene, J. (1984), Simulated Annealing Without Rejected Moves, Proceedings of IEEE International Conference on Computer Design: VLSI in Computers, ICCD-84, Port Chester, NY, pp 658-664.

Greene, J. and Supowit, K. (1986), Simulated Annealing Without Rejected Moves, IEEE Transactions on CAD, Vol.CAD-5, Number 1, pp 221-228.

Hac, A. (1985), Suspension Optimization of a 2-DOF Vehicle Model using a Stochastic Optimal Control Technique, Journal of Sound and Vibration, Vol.100, Number 3, pp 343-357.

Hac, A. (1987), Adaptive Control for Vehicle Suspension, Vehicle System Dynamics, Vol.16, pp 57-74.

Hajek, B. (1988), Cooling Schedule for Optimal Annealing, Mathematics of Operations Research, Vol.13, pp 311-329.

Halton, J.H. (1960), On the Efficiency of certain Quasi-random Sequences of Points in Evaluating Multi-dimensional Integrals, Numerische Mathematik, Vol.2, pp 84-90.

Hammersley, J.M. and Handscomb, D.C. (1964), Monte Carlo Methods, Methuen, London.

Hansen, E. (1979), Global Optimization using Interval analysis-the One Dimensional Case, Journal of Optimization Theory and Applications, Vol.29, pp 331-344.

Hansen, E. (1980), Global Optimization using Interval analysis-the Multi-dimensional Case, Numerische Mathematik, Vol.34, pp 247-270.

Hassan, A.M. and Storey, C. (1981), Numerical Determination of Domains of Attraction for electrical power systems using the Method of Zubov, International Journal of Control, Vol.34 Number 2, pp 371-381.

Hassan, A.M. (1982), Extensions of Zubov's Method for the Determination of Domains of Attraction, Ph.D. Thesis, Loughborough University of Technology.

Hoffman, K.L. (1981), A Method for Globally Minimizing Concave Function over a Convex Set, Mathematical Programming, Vol.20, pp 22-32.

Horst, R., and Tuy, H. (1990), Global Optimization; Deterministic Approaches, Springer-Verlag, Berlin.

Huber, K. P. and Herzberg, G. (1979), Constants of Diatomic Molecules, Van Nostrand Reinhold, New York.

Huang, M.D., Romeo, F. and Sangiovanni-Vincentelli, A.L. (1986), An Efficient General Cooling Schedule for Simulated Annealing, Proceedings International Conference on CAD, ICCAD-86, Santa Clara, CA, pp 381-384.

Ichida, K. and Fujii, Y. (1979), An Interval Arithmetic Method for Global Optimization, Computing, Vol.23, pp 85-97.

Ingber, L. (1989), Very Fast Simulated Re-annealing, Mathematical and Computer Modelling, Vol.12, pp 967-973.

Karnopp, D. (1983), Active Damping in Road Vehicle Suspension System, Vehicle System Dynamics, Vol.12, pp 291-316

Khachaturyan, A. (1986), Statistical Mechanics Approach in Minimizing a Multivariable Function, Journal of Mathematical Physics, Vol.27, Number 7, pp 1834-1838.

Kirkpatrick, S. Gelatt, C.D. and Vecchi, M.P. (1983), Optimization by Simulated Annealing, Science, Vol.220, Number 4598, pp 671-680.

Kushner, H.J. (1987), Asymptotic Global Behaviour for Stochastic Approximation and Diffusions with Slowly Decreasing Noise Effects: Global Minimization via Monte Carlo, SIAM Journal of Applied Mathematics, Vol.47, Number 1, pp 169-185.

Lam, J. and Delosme, J.M.(1986), Logic Minimization Using Simulated Annealing, Proceedings of the IEEE International Conference on CAD, ICCAD-86, Santa Clara, CA, pp 348-351.

Lapidus, L. and Luus, R. (1967), Optimal Control of Engineering Processes, Blaisdell, Waltham, Mass., pp 243-273.

Lasdon, L. S. , Mitter, S. K., and Waren, A. D. (1967), The Conjugate Gradient Method for Optimal Control Problems,IEEE Transactions on Automatic Control, Vol.Ac-12, Number 2, pp 132-138.

Leong, H.W. and Liu, C.L. (1985), Permutation Channel Routing, Proceedings IEEE Conference on Computer Design: VLSI in Computers, ICCD-85, Port Chester, NY, October 1985, pp 579-584.

Leong, H.W., Wong, D.F. and Liu, C.L. (1985), A Simulated Annealing Channel Router, Proceedings IEEE Conference on CAD, ICCD-85, Santa Clara, CA, November 1985, pp 226-228.

Levy, A. and Govez, S. (1980), The Tunneling Method for the Global Optimization Problem of Constrained functions, Technical Report 231, Universidad, National Autonoma de Mexico.

Levy, A. and Montalvo, A. (1985), The Tunneling Algorithm for Global minimization of functions, SIAM Journal of Scientific and Statistical Computing, Vol.6, pp 15-29.

Lundy, M. (1984), Global Optimization and the Simulated Annealing Algorithm, Ph.D. Thesis, Statistical Laboratory, Department of Pure Mathematics and Mathematical Statistics, University of Cambridge, UK.

Lundy, M. and Mees, A. (1986), Convergence of an Annealing Algorithm, Mathematical Programming, Vol.34, pp 111-124.

Luus, R. and Cormack, D.E. (1972), Multiplicity of Solutions Resulting from the Use of Variational Methods in Optimal Control Problems, Canadian Journal of Chemical Engineering, Vol.50, pp 309-311.

Luus, R. (1989), Optimal Control by Dynamic Programming using Accessible Grid Points and Region Reduction, Hungarial Journal of Industrial Chemistry, Vol.17, pp 523-543.

Luus, R. (1990), Optimal Control by Dynamic Programming using Systematic Reduction in Grid Size, International Journal of Control, Vol.51, number 5, pp 995-1013.

Luus, R. and Galli, M. (1990), Multiplicity of Solutions in using Dynamic Programming for Optimal Control, Workshop on Chemical Engineering Mathematics and Computation, Göttingen, West Germany; July, pp 15-20.

Luus, R. and Galli, M. (1991), Multiplicity of Solutions in using Dynamic Programming for Optimal Control, Hungarian Journal of Industrial Chemistry, in press.

Luus, R., Dittrich, J. and Keil, F.J. (1991), Multiplicity of Solutions in the Optimization of a Bifunctional Catalyst Blend in a Tubular Reactor, International Workshop on Chemical Engineering Mathematics, Göttingen, Germany, July 7-11.

Marsh, C. (1992), A nonlinear Control design Methodology for Computer-controlled Vehicle Suspension System, Ph.D. Disertation, Department of Transport Technology, Loughborough University of Technology.

Marsh, C., Gordon, T. J., and Milsted, M. G. (1989), Design of Optimal Non-linear Control Schemes for Vehicle Suspensions, Internal Report, Department of Transport Technology, Loughborough University of Technology.

Masri, S.F., Bekey, G.A. (1980), A Global Optimization Algorithm Using Adaptive Random Search, Applied Mathematics and Computation, Vol.7, pp 353-375.

Metropolis, N., Rosenbluth, A.W., Rosenbluth, M.N., Teller, A.H. and Teller, E. (1953), Equation of State Calculations by Fast Computing Machines, Journal of Chemical Physics, Vol.21, Number 6, pp 1087-1092.

Moore, R.E. (1966), Interval Analysis, Prentice-Hall, Englewood Cliffs.

Nelder, J.A.and Mead, R. (1965), A Simplex Method for function minimization, The Computer Journal, Vol.7, pp, 308-313.

Otten R.H.J.M. and van Ginneken L.P.P.P. (1984), Floorplan Design Using Simulated Annealing, Proceedings IEEE International Conference on CAD, ICCAD-84, Santa Clara, CA, November, 1984, pp 96-98.

Palosaari, S., Reunanen, J. and Miyahara, M. (1992), Nonlinear Constrained Global Optimization with Random Search and Simplex Booster, Proceedings of the 42nd Canadian Chemical Engineering Conference, pp 283-284.

Papadimitriou, C.H. and Steiglitz, K. (1982), Combinatorial Optimization, Prentice Hall, Englewood Cliffs, NJ, pp 366-371.

Price, W.L. (1977), A Controlled Random Search Procedure for Global Optimization, Computer Journal, Vol.20, Number 4, pp 367-370.

Price, W.L. (1978), A Controlled random Search Procedure for Global Optimization, in Towards Global Optimization 2, Dixon, L.C.W., and Szegö, G.P. (eds.), North-Holland, Amsterdam, Holland, pp 71-84.

Price, W.L. (1983), Global Optimization by Controlled Random Search, Journal of Optimization theory and Applications, Vol.40, pp 333-348.

Price, W.L. (1987), Global Optimization Algorithm for a CAD workstation, Journal of Optimization theory and Applications, Vol.55, pp 133-146.

Pronzato, L., Walter, E., Venot, A. and Lebruchec, J. (1984), A General Purpose Global Optimizer: Implementation and Applications, Mathematics and Computers in Simulation, Vol.26, pp 412-422.

Rao, S. S. (1978), Optimization Theory and Applications, Wiley Eastern Limited, New Delhi, India.

Ratschek, H., and Rokne, J. (1988), New Computer Methods for Global Optimization, Ellis Horwood, Chichester.

Renpu, G.E. (1990), A Filled Function Method for Finding a Global Minimizer of a Function of Several Variables, Mathematical Programming, Vol.46, pp 191-204.

Rinnooy Kan, A.H.G. and Timmer, G.T. (1984), Stochastic Methods for Global Optimization, American Journal of Mathematical and Management Sciences, Vol.4, pp 7-40.

Rinnooy Kan, A.H.G. and Timmer, G.T. (1987), Stochastic Global Optimization Methods; Part I : Clustering Methods, Mathematical Programming, Vol.39, pp 27-56.

Rinnooy Kan, A.H.G. and Timmer, G.T. (1987a), Stochastic Global Optimization Methods; Part II : Multilevel Methods, Mathematical Programming, Vol.39, pp 57-78.

Robinson, P.J., Pettitt, A.N., Zornig, J. and Bass, L. (1983), A Bayesian Analysis of Capillary Heterogeneity in the Intact Pig Liver, Biometrics, Vol.39, pp 61-69.

Romeo, F. and Sangiovanni-Vincentelli, A. L. (1985), Probabilistic Hill Climbing Algorithms: Properties and Applications, Proceedings 1985, Chapel Hill Conference on VLSI, Chapel Hill, NC, 1985, pp 393-418.

Romeo, F., Sangiovanni-Vincetenlli, A. and Sechen, C. (1984), Research on Simulated Annealing at Berkeley, Proceedings IEEE Conference on Computer Design: VLSI in Computers, ICCD-84, ICCD-85, Port Chester, NY, October 1984, pp 652-657.

Rosen, J.B. (1981), Parametric Global Minimization for Large Scale Problems, Technical Report, University of Minnesota.

Rosenbrock H. H. and Storey, C. (1966), Computational Techniques for Chemical Engineers, Pergamon Press, N. Y.

Sanchez, D. A., Allen, Jr., R. C. and Kyner, W. T. (1988), Differential Equations, Second Edition, Addison-Wesley Publishing Company, New york.

Schagen, I.P. (1980), Stochastic Interpolating Functions- Applications in Optimization, Journal of Institute of Mathematics and Applications, Vol.25, pp 93-101.

Schagen, I.P. (1986), Internal modelling of objective functions for Global Optimization, Journal of Optimization theory and Applications, Vol.51, Number 2, pp 345-353.

Sharp, R. S. and Crolla, D. A. (1987), Road Vehicle Suspension System Design - a Review, Vehicle Systems Dynamics, Vol.16, pp 57-74.

Shaw, J.E.H. (1988), A Quasirandom Approach to Integration in Bayesian Statistics, The Annals of Statistics, Vol.16, Number 2, pp 895-914.

Shubert, B.O. (1972), A sequential method seeking the global maximum of function, SIAM Journal on Numerical Analysis, Vol.9, pp 379-388.

Smith, R. (1992), A semi-empirical many-body interatomic Potential for Modelling Dynamical Processes in Gallium Arsenide, Nuclear Instruments and Methods in Physics Research B, Vol.67, pp 335-339.

Solis, F.J. and Wets, R.J.B (1981), Minimization by Random Search Techniques, Mathematics of Operations Research, Vol.6, Number 1, pp 19-30.

Thompson, A. G. (1976), An Active Suspension with Optimal Linear State Feedback, Vehicle System Dynamics, Vol.5, pp 187-203.

Thompson, A. G. (1984), Optimal and Suboptimal Linear Active Suspensions for Road Vehicles, Vehicle System Dynamics, Vol.13, pp 61-72.

Tersoff, J. (1988), New Empirical approach for the Structure and Energy of Covalent Systems, Physics Review B, Vol.37, No.12, pp 6991-7000.

Tersoff, J. (1988a), Empirical Interatomic Potential for Silicon with improved Elastic Properties, Physics Review B, Vol.38, No.14, pp 9902-9905.

Timmer, G.T. (1984), Global Optimization: A Stochastic Approach, Ph.D. Disertation, Econometric Institute, Erasmus University Rotterdam.

Törn, A. (1978), A Search Clustering Approach to Global Optimization in Towards Global Optimization 2, Dixon, L.C.W., and Szegö, G.P. (eds.), North-Holland, Amsterdam, Holland, pp 49-62.

Törn, A. and Viitanen, S. (1992), Topographical Global Optimization, in Recent Advances in Global Optimization, C. A. Floudas and P. M. Pardalos (eds.), Princeton University Press, Princeton USA, pp 384-398.

Törn, A., and Žilinskas, A. (1989), Global Optimization. Springer-Verlag, Berlin.

Tsitsiklis, J.N. (1989), Markov Chain with Rare Transitions and Simulated Annealing, Mathematics of Operations Research, Vol.14, pp 70-90.

Vanderbilt, D. and Louie, S.G. (1984), A Monte Carlo Simulated Annealing approach to Optimization over Continuous Variables, Journal of Computational Physics, Vol.56, pp 259-271.

Van Laarhoven, P.J.M. and Aarts, E.H.L. (1987), Simulated Annealing: Theory and Applications, Kluwer Academic Publisher, Dordrecht, Netherlands.

White, P. (1979), Computation of Domains of Attraction of Ordinary Differential Equations using Zubov's Methods, Ph.D. Thesis, Loughborough University of Technology.

White, S.R. (1984), Concepts of Scale in Simulated Annealing, American Institute of Physics Conference Proceedings, Number 122, pp 261-270.

Wilson, D. A., Sharp, R. S. and Hassan, S. A. (1986), The Application of Linear Optimal Control Theory to the Design of Active Automotive Suspensions, Vehicle Systems Dynamics, Vol.15, pp 105-118.

Yamashita, H. (1979), A Continuous Path Method of Optimization and its Application to Global Optimization, in: Prekopa, A., (eds.), Survey of Mathematical Programming 1 (Budapest) pp 539-546.

Zubov, V.I. (1964), Methods of A.M. Liapunov and their application, Noordhoff, Groningen, The Netherlands.

Appendices

Appendix 2A

```
program halton(input,output);

const
        ndims=10;
        npoints=500;
type
    float=longreal;
    mat=array[1..npoints,1..ndims] of float;
    vec=array[1..ndims] of float;
    posint = 0..maxint;
var
    npts, ndim, i, j, k :  posint;
    p,prim :  vec;
    xpt :  mat;
    train :text;

procedure qrhal(var xpt :  mat):
var
    i, j, k :  posint;
    r,f,g,h :  float;
begin
  for i:=1 to ndim do
    begin
      r:=1/prim[i];
      for j:=1 to npts do
        begin
          if j >1 then
            f:=1.0-xpt[j-1,i]
          else
            f:=1.0-p[i];
          g:=1.0; h:=r;
          while f-h <1.0L-3 do
            begin
              g:=h; h:=h*r;
            end;
          xpt[j,i]:=g+h-f;
        end;
    end;
end;

begin
    rewrite(train, 'hal.dat');
    npts:=10; ndim:=2; prim[1]:=2; prim[2]:=3;
    p[1]:=0; p[2]:=0;
    qrhal(xpt);
    for i:=1 to npts do
    begin
      for j:=1 to ndim do
      write(train, xpt[i,j]:=14,'  '); writeln(train);
    end;
end.
```

Appendix 2B

The following test functions were chosen from Dixon and Szegö (1978).

GP (Goldstein and Price)

$f(x_1, x_2) = [\, 1 + (x_1 + x_2 + 1)^2(19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2) \,]$
$[\, 30 + (2x_1 - 3x_2)^2(18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2) \,]$

$\Omega = \{\, x \in \mathrm{R}^2 \mid -2 \le x_i \le 2,\ i = 1, 2 \,\},\ x^* = (0, -1),\ f(x^*) = 3$

There are four local minima

BR (Branin)

$f(x_1, x_2) = a(x_2 - bx_1^2 + cx_1 - d)^2 + e(1 - f)\cos(x_1) + e$ where $a = 1, b = 5.1/(4\pi^2)$, $c = 5/\pi, d = 6, e = 10, f = 1/(8\pi)$

$\Omega = \{\, x \in \mathrm{R}^2 \mid -5 \le x_1 \le 10, \text{and } 0 \le x_2 \le 15 \,\}$

$x^* = (-\pi, 12.275);\ (\pi, 2.275);\ (3\pi, 2.475),\ f(x^*) = 5/(4\pi).$

There are no more minima.

H3 and H6 (The Hartmann family)

$f(x) = -\sum_{i=1}^{m} c_i \exp\left(-\sum_{j=1}^{n} a_{ij}(x_j - p_{ij})^2\right)$

Table 2a

H3 ($n = 3$ and $m = 4$)

i	a_{ij}			c_i	p_{ij}		
1	3	10	30	1	0.3689	0.1170	0.2673
2	0.1	10	35	1.2	0.4699	0.4387	0.7470
3	3	10	30	3	0.1091	0.8732	0.5547
4	0.1	10	35	3.2	0.038150	0.5743	0.8828

Table 2b

H6 (n = 6 and m = 4)

i	a_{ij}						c_i	p_{ij}					
1	10	3	17	3.5	1.7	8	1	0.1312	0.1696	0.5569	0.0124	0.8283	0.5886
2	0.05	10	17	0.1	8	14	1.2	0.2329	0.4135	0.8307	0.3736	0.1004	0.9991
3	3	3.5	1.7	10	17	8	3	0.2348	0.1451	0.3522	0.2883	0.3047	0.6650
4	17	8	0.05	10	0.1	14	3.2	0.4047	0.8828	0.8732	0.5743	0.1091	0.0381

$\Omega = \{x \in \mathrm{R}^n \mid 0 \le x_i \le 1,\ 1 \le i \le n\}$. These functions both have 4 local minima, $x_{loc} \approx (p_{i1}, \ldots, p_{in})$, $f(x_{loc}) \approx -c_i$

S5, S7 and S10 (The Shekel family)

$f(x) = -\sum_{i=1}^{m}((x - a_i)^T(x - a_i) + c_i)^{-1}$

with the dimension n=4, m=5,7,10 for S5, S7, S10, respectively, $x = (x_1, ..., x_n)^T$ and $a_i = (a_{i1}, ..., a_{in})^T$.

Table 2c

$S5,\ S7\ , S10$

i	a_{ij}				c_i
1	4	4	4	4	0.1
2	1	1	1	1	0.2
3	8	8	8	8	0.2
4	6	6	6	6	0.4
5	3	7	3	7	0.4
6	2	9	2	9	0.6
7	5	5	3	3	0.3
8	8	1	8	1	0.7
9	6	2	6	2	0.5
10	7	3.6	7	3.6	0.5

$\Omega = \{x \in \mathrm{R}^4 \mid 0 \le x_j \le 10, 1 \le j \le 4\}$. These functions have 5, 7 and 10 local minima for S5, S7 and S10, respectively, $x_{loc} \approx a_i$, $f(x_{loc}) \approx 1/c_i$ for $1 \le i \le m$.

Appendix 3A

```
procedure heatup(var starttemp,fval:datatype;
          var x:vector);

var
          m1,m2,m :integer;
          tdcm1,tdcm2,advm1,advm2 :  datatype;
          w,tempnext,diffcost,bot :  datatype;
          i,j,k:  integer;

begin
  starttemp:=0.0;
  tempnext:=0.0;
  m:=0;
  m1:=0;
  m2:=0;
  tdcm1:=0;
  tdcm2:=0;
  advm1:=0;
  advm2:=0;
  fval:=obj(x);
  nits:=nits+1;
  repeat
    starttemp:=tempnext;
      moverslt:=allowmove(starttemp,diffcost,fval,x);
      if(moverslt=ACCEPT)then begin
        m1:=m1+1;
        tdcm1:=tdcm1+diffcost
      end else
      begin
        m2:=m2+1;
        tdcm2:=tdcm2+diffcost;
      end;
      if(m2>0) then
        advm2:=tdcm2/m2;
        if(m1>0) then
        advm1:=tdcm1/m1;
      if(moverslt<>REJECT)then m:=m+1;
        bot:=m2*Chi - (m1*(1-Chi));
      if(bot>0) then
        tempnext:=advm2/ln(m2/bot);
  until (m1+m2)=10*ndim;
  T:=starttemp;
  To:=T;
end;

function allowmove(var temp, diffcost,fval :  datatype;
          var x :  vector) :  moversltttype;
label 31;
var
          w,prob,fp,fold,pp :  datatype;
          i,j,k :  integer;
```

```
        y :  vector;
        flag1 :  boolean;
  begin
    fold:=fval;
    w:=random;
    if w<= 0.75 then begin
    w:=random;
    for k:=1 to ndim do
    y[k]:=w*xlower[k]+(1-w)*xupper[k];
    fp:=obj(y);
    nits:=nits+1;
  end
  else
  begin
    grad(x,g);
31:  for k:=1 to ndim do y[k]:=x[k]-silon*g[k];
    fp:=obj(y);
    nits:=nits+1;
    if fp>fold then begin
    silon:=0.5*silon;
    goto 31 end;
  end;
    diffcost:=fp-fold;
    if diffcost <= 0.0 then
    begin
      prob:=1;
    end
    else
      prob:=exp(-diffcost/temp);
    w:=random;
    if(w>prob) then
      allowmove:=REJECT
    else
    begin
      for k:=1 to ndim do
      x[k]:=y[k];
      fval:=fp;
      if diffcost <=0 then
      allowmove:=ACCEPT
      else
      allowmove:=PROBABILISTIC
    end;
  end;
```

Appendix 4A

```
program hammersley(input,output);

const
        ndims=10;
        npoints=500;
type
    float=longreal;
    mat=array[1..npoints,1..ndims] of float;
    vec=array[1..ndims] of float;
    posint = 0..maxint;
var
    npts, ndim, i, j, k :  posint;
    p,prim :  vec;
    xpt :  mat;
    train :text;

procedure qrpham(var xpt :  mat; prim :  vec);
var
    i, j, k :  posint;
    r,f,g,h :  float;
begin
  for i:=2 to ndim do
    begin
      r:=1/prim[i];
      for j:=1 to npts do
        begin
          if j >1 then
            f:=1.0-xpt[j-1,i]
          else
            f:=1.0-p[i];
          g:=1.0; h:=r;
          while f-h <1.0L-15 do
            begin
              g:=h; h:=h*r;
            end;
          xpt[j,i]:=g+h-f;
          xpt[j,1]:=(j)/(npts+1)
        end;
    end;
end;

begin
    rewrite(train, 'ham.dat');
    npts:=50; ndim:=2; prim[2]:=2; prim[3]:=3;
    p[1]:=0; p[2]:=0;
    qrpham(xpt,prim);
    for i:=1 to npts do
    begin
      for j:=1 to ndim do
      write(train, xpt[i,j]:=14,'  '); writeln(train);
    end;
end.
```

Appendix 4B

```
function BETA(a1,b1,k1:longreal):longreal;
var
  v1,v,w,r,s  :longreal;
  p           :longreal;
  u1,u2,z,t   :longreal;
  a,b         :longreal;
  accept      :0..1;
  gamma,beta1  :longreal;
  alpha       :longreal;
  ii,jj       :integer;

function random:  longreal; external ftn77;
begin
     alpha:=a1+b1;
     if a1<b1 then a:=a1 else a:=b1;
     b:=alpha-a;
     beta1:=sqrt((alpha-2)/((2*a*b)-alpha));
     gamma:=a+(1/beta1);
accept:=0;
     repeat
          u1:=random;
          u2:=random;
          v:=beat1*ln(u1/ (1-u1));
          w:=a*exp(v);
          r:=(gamma*v)-1.3862944;
          s:=a+r-w;
          z:=u1*u1*u2;
          if (s+2.609438)>=(5*z) then accept:=1;
          if accept=0 then begin
            t:=ln(z);
            if s>=t then accept:=1;
            end;
          if accept=0 then begin
            if r+(alpha*ln(alpha/(b+w)))>=t then accept:=1;
            end;
     until (accept=1);
     if a=a1 then v1:=(k1*w)/(b+w) else v1:=(k1*b)/(b+w);
     Beta:=v1;
end;
```

Appendix 7A

Dynamic Programming

Having the x-grid at each time stage, and the allowable values for the control at each stage, the DP procedure as outlined by Luus (1989) can be summarized as follows:

Iterative dynamic programming algorithm

1. Divide the time interval t_f into P time stages, each of length L.
2. Choose the number of x-grid points N^d and the number of allowable values M for the control u.
3. Choose the region r for the control values.
4. By choosing N^d values of the control inside the allowable region, integrate (7.1) N^d times to generate the x-grid at each time stage.
5. Starting at the last time stage P, corresponding to $t_f - L$, for each x-grid point integrate (7.1) from $t_f - L$ to t_f for all the M allowable values of control. Choose the control that optimized the performance index and store the value of the control for use in step 6.
6. Step back to stage $P-1$, corresponding to time t_f-2L, and integrate (7.1) from t_f-2L to t_f-L for each x-grid point with the M allowable values of control. To continue integration from $t_f - L$ to t_f choose the control from step 5 that corresponds to the grid point nearest to the resulting x at $t_f - L$. Compare the M values of the performance index and store the value of control that gives the maximum value.
7. Continue the procedure until stage 1, corresponding to the initial time $t = 0$ is reached. Store the control policy that optimizes the performance index and store the corresponding x-trajectory.
8. Reduce the region for allowable control values by a factor ϵ_1; i.e.

$$r^{(j-1)} = (1 - \epsilon_1) r^{(j)}$$

where j is the iteration index. Use the optimal x-trajectory from step 7 as the mid-point for the x-grid at each time stage, and use the optimal control policy from step 7 as the midpoint for the allowable values for the control u.

9. Increment the iteration index j by 1 and go to step 4. Continue the iteration for a specified number of iterations (say 20) and examine the results.

Appendix 7B

Coefficients for the rate constants k_i

i	c_{i1}	c_{i2}	c_{i3}	c_{i4}
1	0.2918487E-02	-0.8045787E-02	0.6749947E-02	-0.1416647E-02
2	0.9509977E+01	-0.3500994E+02	0.4283329E+02	-0.1733333E+02
3	0.2682093E+02	-0.9556079E+02	0.1130398E+03	-0.4429997E+02
4	0.2087241E+03	-0.7198052E+03	0.8277466E+03	-0.3166655E+03
5	0.1350005E+01	-0.6850027E+01	0.1216671E+02	-0.6666689E+01
6	0.1921995E-01	-0.7945320E-01	0.1105666E+00	-0.5033333E-01
7	0.1323596E+00	-0.4696255E+00	0.5539323E+00	-0.2166664E+00
8	0.7339981E+01	-0.2527328E+02	0.2993329E+02	-0.1199999E+02
9	-0.3950534E+00	0.1679353E+01	-0.1777829E+01	0.4974987E+00
10	-0.250466E-04	0.1005854E-01	-0.1986696E-01	0.9833470E-02

Appendix 7C

```
function pig(var x:  datatype):float;

var zw,sum,v,e2,km,vmax :  float;
        i,j,k :  posint;

begin
  zw := 1.0; km := x[1]; e2 := x[2];
  for j := 1 to 5 do
  begin
    sum := 0.0;
    sigpig[j] := abs(x[j+2]);
    vmax := x[j+7];
    for i := 1 to numpig[j] do
    begin
      solve(v,vmax,e2,fpig[j],km,cval[i,j,1],cval[i,j,2]);
      sum := sum + sqr(ln(vpig[i,j])-ln(v));
    end;
    zw:=zw*exp(-sum/(2*sqr(sigpig[j])))/(exp(numpig[j]*
        ln(sigpig[j])));
  end;
    if zw > 1.0e-20 then pig := -ln(zw)
        else pig := 50.0;
    end;
  end;
end;


procedure solve(var v:float; vmax,e2,f,km,ci,co:float);

  var chat,fun,oldv,alpha :  float;
  begin
      chat := (ci-co)/ln(ci/co);
      alpha := vmax/(km/chat+1);
      v := 0.5*vmax;
      repeat
        fun := 1+(v/(f*km))/(exp((vmax-v)/(f*km))-1);
        fun := 1 - (e2*vmax/(2*f*km))/sqr(fun);
        oldv := v;
        v := 0.5*(oldv + alpha*fun);
      until abs(v-oldv) < 1.0e-6;
      if v < 0.03 then v := 0.03;
  end;
```