LA-UR-- 95-1196

TITLE: Intervals in Evolutionary Algorithms for Global Optimization

AUTHOR(S): Rajendra B. Patil
Computer Research and Applications
CIC-3, MS B256
Los Alamos National Laboratory, Los Alamos, NM 87545

SUBMITTED TO: APIC'95 Extended Abstracts, A Supplement to the International Journal of Reliable Computing, El Paso

LA-UR-95

Intervals in Evolutionary Algorithms For Global Optimization

Rajendra B. Patil
Computer Research and Applications
CIC-3, MS B256
Los Alamos National Laboratory
Los Alamos, NM 87545

Intervals in Evolutionary Algorithms for Global Optimization

Rajendra B. Patil

Abstract— Optimisation is of central concern to a number of disciplines. Interval Arithmetic methods for global optimisation provide us with (guaranteed) verified results [6]. These methods are mainly restricted to the classes of objective functions that are twice differentiable and use a simple strategy of eliminating a splitting larger regions of search space in the global optimisation process. This is done by estimating the bounds of the given function over intervals using methods from interval arithmetic. When applied to arbitrary complicated objective functions these methods compute overly pessimistic over-estimations of the bounds and therefore cannot be efficiently applied.

Evolutionary search algorithms are *heuristic* global optimization techniques in the sense that they do not provide a guaranteed (verified) result. Many models of the Evolutionary Algorithms are empirically proven to be robust and have demonstrated their capability to produce good solutions in many complex optimisation problems. These methods require very little knowledge about the structure of the search space of the problem at hand, so they are naturally applied to problems whose structure is poorly understood. This includes the cases where the exact function to be optimised is unknown. In such cases the Interval Methods of Global Optimisation cannot be applied. The disadvantages of Evolutionary Algorithms for global optimisation are that they are compute intensive, do not provided verified results and have complex dynamics making the theoretical proofs of their efficiency and convergence difficult.

An efficient approach that combines the efficient strategy from Interval Global Optimisation Methods and robustness of the Evolutionary Algorithms is proposed. In the proposed approach, search begins with randomly created interval vectors with interval widths equal to the whole domain. Before the beginning of the evolutionary process, fitness of these interval parameter vectors is defined by evaluating the objective function at the center of the initial interval vectors. In the subsequent evolutionary process the local optimisation process returns an estimate of the bounds (lower or upper for minimisation or maximisation) of the objective function over the interval vectors. Though these bounds may not be correct at the beginning due to large interval widths and complicated function properties, the process of reducing interval widths over time and a selection approach similar to simulated annealing helps in estimating reasonably correct bounds as the population evolves. The interval parameter vectors at these estimated bounds (local optima) are then subjected to crossover and mutation operators. This evolutionary process continues for predetermined number of generations in the search of the global optimum.

In the proposed approach, though the verified nature of the Interval Global Optimization Methods is not preserved, empirical results over standard benchmark functions has shown to have preserved the efficiency and robustness properties of the two approaches. The proposed approach is benchmarked against Parallel Genetic Algorithm and is observed to be more robust and in some cases order of magnitude efficient.

I. Introduction

Optimisation is of central concern to a number of disciplines in which numerical information is processed. Many problem solving methods in operations research, applied mathematics and artificial intelligence (AI) have an optimisation procedure as

The author is with Computing Research and Applications (CIC-3), MS - M986, Los Alamos National Laboratory, Los Alamos, NM 87545, email rbp1@.lanl.gov.

a subcomponent and the performance of these techniques hinges critically on the quality of the optimisation technique. The problem characteristics decides if the optimisations is constrained or unconstrained and whether the parameter space is continuous or discrete. In either case, the search method used to search for an optimum could be local or global, deterministic or stochastic, or in some cases a combination of these.

Optimisation is concerned with the characterization and computation of minima or maxima of a function $f(\vec{x})$. Given,

$$f : X \subseteq X_1 \times X_2 \times ... \times X_n \rightarrow R, \quad X \neq \emptyset \tag{1}$$

where f is called the objective function, the goal is to find a vector $\vec{x}^* \in X$ such that:

$$\forall \vec{x} \in X : f(\vec{x}) \geq f(\vec{x}^*) = f^* \tag{2}$$

where f^* is called the global minimum and $\vec{x}^*$ is the minimum location point or a set. Since $\max\{f(\vec{x})\} = -\min\{-f(\vec{x})\}$, the restriction to minimisation is without loss of generality. In general the optimisation task is complicated by the existence of non-linear objective function with multiple local optima. A local optimum $f' = f(\vec{x}')$ is defined by the following condition

$$\exists \epsilon > 0, \forall \vec{x} \in X : \| \vec{x} - \vec{x}' \| < \epsilon \Rightarrow f' \leq f(\vec{x}) \tag{3}$$

Even if there is only one optimum, it may be difficult to find a path towards it if there is a discontinuity in the objective function or its derivatives. Wide variety of methods for finding global solutions to nonlinear optimisation problems have been proposed. In this paper we will concentrate on global unconstrained optimisation using interval based evolutionary search.

Traditional calculus based methods assume that the objective function $f(\vec{x})$ (and constraints in constraint optimisation) are twice continuously differentiable functions of $\vec{x}$. These methods require explicit or implicit second derivative calculations of the objective function which in some cases can be ill-conditioned and cause the algorithm to fail. During the last 30 years there has been considerable research directed towards the nonlinear optimisation problems and progress has been made in theory and practice [12]

In general there is no known method for determining global maximum (or minimum) to the general nonlinear optimisation problem. Only if the objective function f (and the constraints c_i) satisfies certain properties, the global optimum can sometimes be found. algorithms for constraint problems are usually classified as indirect and direct methods. An indirect method solves the problems by extracting one or more linear problems from the original one, whereas a direct method tries to determine successive search points. This is usually done by converting the original problem into unconstrained one for which gradient methods are applied with some modifications [14].

There are many other problems connected with traditional optimisation techniques. For example, most proposed methods are local in scope, they depend on existence of derivatives, and they are insufficiently robust in discontinuities, vast multi-modal, or noisy search spaces [10]. Indeed, for many real world problems it may be impossible to find derivatives of f or in some situations f may not be known at all. In those cases the only way to get information about the function is to evaluate it at different values of $\vec{x}$. The performance of most optimisation techniques is thus compared using the number of function evaluations required to find the optimum. In this regard, it is important to investigate other heuristic methods which may prove useful in many real world problems. The restricted cases where the parameter space can be searched exhaustively or the objective function can be subjected to analytical methods are not considered here.

The main reasons for the failure of many optimisation algorithms are, the algorithm can get trapped in a local optimum (known as the foothill problem), it can get trapped in mostly flat surfaces with few sharp peaks (called the plateau problem), or it can get trapped because the direction of ascent (or descent) is not within the direction of search motion (known as the ridge problem) [7].

A. Evolutionary Computations

Evolutionary computation belongs to a large class of methods which attempts to optimise a function using a strategy essentially independent of the problem at hand. These methods require very little knowledge about the structure of the search space of the problem at hand, so they are naturally applied to problems whose structure is poorly understood.

Evolutionary algorithms are population based search strategies that maintain the locations of a set of probes in the function parameter space. The standard of comparison for which new probes are generated or old probes are discarded is a function of existing probe population. Combinations of individual probes in the population are used to derive new probes. The probe population is arbitrarily initialized, and it evolves towards better and better regions of the search space by means of randomised processes of *selection* (deterministic in some cases), *mutation*, and *recombination* (not used in some). The environment delivers a quality of information, the fitness value (objective function value) of the search probe, and the selection process favors those individual (probes) of higher fitness ("survival of the fittest") to reproduce more often than worse individuals. The recombination mechanism allows the mixing of parental information while passing it to their descendants, and mutation introduces innovation into the probe population. Thus, during the evolution process the average quality of population (probes, solutions) increases, hopefully leading to optimum solution.

The metaphoric description of evolutionary algorithms can be put into more formal terms. Here we use the formalism developed in [29].

Let $f : \Re^n \to \Re$ denote the objective function to be optimised. Let $\vec{x} \in \Re^n$ be a probe. The search algorithm may directly use $\vec{x}$ as vectors of real numbers, or use it in some encoded form, $\vec{a} = e(\vec{x})$ (e.g. binary, graycode). In some evolutionary algorithms, $\vec{x}$ is used as vector of reals (Evolutionary Strategy), while in others $\vec{x}$ is first encoded into binary vector $\vec{a}$, and is used in the process (Genetic Algorithms). In this paper we use $\vec{x}$ as a vector of intervals. Many other representations are possible. Solutions in the parent and offspring population sets are denoted I^μ and I^λ. The fitness function $\Phi : \vec{x} \to \Re$ or $\Phi : \vec{a} \to \Re$ defines the fitness value of an individual $\vec{x}$ or $\vec{a}$ in I respectively. In general, fitness function Φ and objective function f are not required to be identical, but f is always a part of Φ. While $\vec{a} \in I$ is used to denote the representation of an individual used in the process, $\vec{x} \in \Re^n$ indicates the object variable vector. In effect, $\vec{a} = e(\vec{x})$ where $e()$ is some encoding function. Correspondingly there is a decoding function to get the objective vector $\vec{x}$ from encoded vector $\vec{a}$. Furthermore, $\mu \gg 1$ and $\lambda \gg 1$ denote the parent and offspring population. A population at generation t, $P(t) = \{\vec{a}_1(t), ..., \vec{a}_\mu(t)\}$ consists of individual $\vec{a}_i(t) \in I$, and $r_{\Theta_r} : I^\mu \to I^\lambda$ denotes the recombination operator which might be controlled by additional parameters summarised in set Θ_r. Similarly the mutation operator $m_{\Theta_m} = I^\lambda \to I^\lambda$ modifies the offspring population controlled by some parameters Θ_m. Selection $s_{\Theta_s} : (I^\mu \cup I^{\mu+\lambda}) \to I^\mu$ is applied to choose the parent population of next generation. During the evaluation step the fitness function $\Phi : I \to \Re$ is calculated for all individuals of a population, and some form of termination criterion, T : {true, false} is used. The algorithmic description of a typical evolutionary algorithm is given below [29].

ALGORITHM 1: Evolutionary Search Algorithm

```
  t=0;
  *initialize* $P(0) = \{\vec{a}_1(0), ..., \vec{a}_\mu(0)\} \in I^\mu$;
  *evaluate* $P(0) : \{\Phi(\vec{a}_1(0)), ..., \Phi(\vec{a}_\mu(0))\}$
while ($T \neq$ true) do
    *recombine* $P'(t) = r_{\Theta_r}(P(t))$;
    *mutate* $P''(t) = m_{\Theta_m}(P'(t))$;
    *evaluate* $P''(t) = \{\Phi(\vec{a}''_1(t)), ..., \Phi(\vec{a}''_\lambda(t))\}$;
    select: $P(t+1) = s_{\Theta_s}\{P''(t) \cup Q\}$
    $t = t + 1$;
end while
```

Here, $Q \in \{\emptyset, P(t)\}$ is a set of individuals that are additionally taken into account during the selection step.

Some main-streams of Evolutionary Algorithms (EA), based on model of natural evolution are *Genetic Algorithms* (GA) [15], *Classifier Systems* (CS) [18] by Holland, the *Evolution Strategy* (ES) [20, 21] by Rechenberg and Schwefel, Evolutionary Programming (EP) by Fogel, Owens and Walsh [16] and *Genetic Programming* (GP) [19] by Kosa. The approaches mainly differ with respect to the structure of the individuals in the population which directly influences the recombination and mutation operators. The ES and GA are powerful methods for global optimisation, not needing any more information about the objective function than the actual value of the objective function. They do not use any predefined internal model of the objective function. Each of these mainstream algorithms have demonstrated their capability to yield good approximate solutions even in case of complicated multi-modal, discontinuous, non-differentiable, and even noisy or dynamic objective function optimisation problems. A variety of applications have been presented in [22] [23], [24], [25], [26], [27], [28], and an annotated bibliography collected in [29]. Due to their complexity, all EA algorithms lack mathematical proofs of general convergence and efficiency.

B. Interval search and optimization: Motivation

Interval methods are aimed at finding the global optimum of a twice differentiable objective function f defined on a hyper-rectangle X and having gradient ∇f and a Hessian $\nabla^2 f$ with only finite number of (isolated) zeros. Their essence is in evaluation of images $f(Z), \nabla f(Z), \nabla^2 f(Z)$ for hyper-rectangles $Z \subset X$ with the purpose of excluding those which cannot contain extremal points [4]. Interval methods for global optimisation find the global optimum and provide bounds on its value and location(s) that are guaranteed to be correct despite errors from rounding, approximation, and uncertain data [6]. Interval analysis also provides a means for doing sensitivity analysis in a more definitive way. Interval Newton methods have the disadvantage of needing gradient information in explicit or implicit form.

Until recently, it was thought that no numerical optimisation algorithm could guarantee having found the global solution of a general nonlinear optimisation problem. Many researchers said that such a guarantee was impossible [6]. Their argument was based on the observations that, optimisation algorithm can sample the objective function and perhaps some of its derivative at only finite number of distinct points. Hence, there is no way of knowing whether a function to be optimised dips to some unexpectedly small value between sample points. This is a reasonable argument and it is probably true that no algorithm using standard arithmetic will ever guarantee finding the global solution to a general problem. However interval methods do not sample at points. They obtain bounds for a function over an infinite continuum of points [6]. Consider the function (taken from [6]), $f(x) = x^4 - 4x^2$, with minima at $x = \pm 2^{1/2}$. Here $f(1) = -3$ and $f([3,4]) = [17,220]$. Thus, we know that $f(x) \geq 17, \forall x \in [3,4]$, the minimum value of f is no larger than -3. Therefore, a minimum value of f cannot occur in the interval [3,4]. To find this, only two "evaluations" of function f are needed. Notice that "evaluations" of f were over the intervals. Function evaluations over interval is an active area or research and is an important part of interval based search methods. Finding the bounds of a given function f in an interval is the restricted version of problem of optimisation but simply over a smaller (possibly) domain. The interval obtained when evaluating an interval function depends on the form of the function. Considerable effort has been expended by interval analysts in attempting to produce systematic methods for representing an interval function to most sharply bound the range of a given real function over an interval [5]. For monotonic functions these bounds are easy to find. For complicated functions Interval Arithmetic methods compute overly pessimistic over-estimations of the bounds and are useless in the present context.

The interval search approach outlined in this paper has been previously suggested in [8] using principles of genetic algorithms. Here we examine this approach in more detail and suggest some modifications for improving the efficiency and robustness. We also compare the previously proposed interval based GA to the proposed modified approach. In summary, the evolutionary interval search procedure is initialised with a population of interval vectors. Each such interval vector represents a hyper-rectangle in the parameter space. Function f to be optimised is then evaluated over these interval vectors. In the previously suggested approach [8] the evaluation of f is carried out at the center of the hyper-rectangle, in the proposed modification a local search methods is used. The initial population has largest interval widths vectors covering the whole domain. As the evolution of these interval proceed using principles of evolutionary algorithm, the size of hyper-rectangles is reduced. Principle similar to simulated annealing is used in the selection process. During the evolutionary process, the crossover evolutionary operator combines parts of these hyper-rectangles to form new hyper-rectangles. At times the crossover operator also generates new hyper-rectangles by taking intersections and unions of these search hyper-rectangles. The mutation creates new hyper-rectangles as search probes in the parameter search space.

II. Interval Evolutionary Search Algorithm (IESA)

Interval Evolutionary Search Algorithm (IESA) borrows idea from several other randomised methods. The structure of IESA is based on evolutionary algorithm with interval representations. It also combined ideas from simulated annealing and hill climbing (local optimisation) in the evolutionary operators. IESA consists of interval representations, initialisation, fitness evaluation, and evolutionary operators of Reproduction, Crossover, Mutation, Local optimisation and Selection.

A. Interval Representations

The schema theory proposed in [15] give valid motivation for the use of binary string for the representations of solutions in the population of solutions. This maximizes the number of schemata available in the evolutionary search. This also increase the size of the search space as each parameter in the objective function is encoded as a binary string. Higher precision can be achieved by increasing the number of bits. On the other hand, this does not lead to a natural coding of the problem. The need for better accuracy in the location of minimum suggests the use of real-valued solutions in the population of solutions. The representational issue in these, otherwise very similar algorithms has lead to the development of two evolutionary search paradigms, called Genetic Algorithms (GA) and evolution strategies (ES). The binary vs. real representational issue is addressed in [2].

In IESA the population of solutions, $P = \{\bar{\mathbf{a}}_1, ..., \bar{\mathbf{a}}_\mu\}$. Here P is a set of interval vectors $\bar{\mathbf{a}}_i$, where, vector $\bar{\mathbf{a}}_i = (\bar{a}_{i1}, ..., \bar{a}_{in})$ is a n dimension interval search vector of a n-variable function $f(\vec{x})$ to be optimised. Notice here that the function $f(\vec{x})$ being optimised is real-valued and the optimum $\vec{x}^*$ that we are interested in is also a real-valued vector. We are simply using interval vectors in the search process though the final answer is expected to be a real-valued vector. Each jth interval component, $\bar{a}_{ij}$ of an interval search vector, $\bar{a}_i$ is defined with three values as, $\bar{a}_{ij} = (\bar{a}^l_{ij}, \bar{a}^c_{ij}, \bar{a}^u_{ij})$, where $\bar{a}^l_{ij}, \bar{a}^c_{ij}$ and $\bar{a}^u_{ij}$ is the lower, center and upper point of an interval $\bar{a}_{ij}$, such that $\bar{a}^l_{ij} \leq \bar{a}^c_{ij} \leq \bar{a}^u_{ij}$. The parameter $w_{ij} = \bar{a}^u_{ij} - \bar{a}^l_{ij}$, is the width if interval $\bar{a}_{ij}$ and changes adaptively during the search process. The outline of the algorithms is given below.

B. Algorithm

```
initialize $P(0) = \{\bar{a}_1(0), ..., \bar{a}_\mu(0)\}$;
  set initial temperature, $T$;
  set initial widths, $w$;
  evaluate $P(0) : \{\Phi(\bar{a}^c_1(0)), ..., \Phi(\bar{a}^c_\mu(0))\}$;
  while ($\mathcal{T} \neq$ true) do
    crossover $P'(t) = r_{\Theta_r}(F(t))$;
    mutate/megre $P''(t) = m_{\Theta_r}(P'(t))$;
    local opt $P_{Lopt}(t) = \{L(\bar{a}^c_1(t)), ..., L(\bar{a}^c_\lambda(t))\}$;
    select $P(t+1) = s_{\Theta_s}\{P_{Lopt}(t)\}$;
  update widths, ,
  update temperature, $T$;
    $t = t + 1$;
  end while
```

C. Initialization

In the initial population, $P(0) = \{\bar{a}_1, ..., \bar{a}_\mu\}$ is constructed by choosing μ random interval search vectors. Each random interval search vector, $\bar{a}_i = (\bar{a}_{i1}, ..., \bar{a}_{in})$, is constructed by choosing n random centers, $\bar{a}^c_{ij}$, for each the interval components $\bar{a}_{ij}, j = 1, ..., n$. The initial widths, w_{ij}, of each interval component $\bar{a}_{ij}, j = 1, ..., n$, is set to the size of the whole domain over which $f(\vec{x})$ is to be optimised.

D. Fitness evaluation

At iteration t in the evolutionary process, fitness evaluation assigns a fitness value to each individual interval search vector, $\bar{a}_i = (\bar{a}_1, ..., \bar{a}_n)$, $i = 1, ..., \mu$, in the population $P(t)$. Fitness evaluation is a critical part of evolutionary search paradigm as this is the only information the evolutionary algorithms use in the search process. In the present context, we are to assign a fitness value to each interval search vector. This requires evaluating the objective function $f(\vec{x})$ with interval parameters. The estimation of bounds of the objective function using interval methods can be used as fitness value. To get sharper bounds, this process requires the function to be in centered form [6]. As no a priory assumptions are made about the form of the function, converting an arbitrary function to its centered form is difficult. Also In cases where function is not know, interval methods cannot be used to find the bounds.

A heuristic approach to bounds estimation is used here. Before the beginning of the evolutionary process, fitness of these interval parameter vectors is defined by evaluating the objective function at the center of the initial interval vectors. In the subsequent evolutionary process the local optimisation process returns an estimate of the bounds (lower or upper for minimisation or maximisation) of the objective function over the interval vectors. Though these bounds may not be correct at the beginning due to large interval widths and complicated function properties, the process of reducing interval widths over time and a selection approach similar to simulated annealing helps in estimating reasonably correct bounds as the population evolves. The interval parameter vectors at these estimated bounds (local optima) are then subjected to crossover and mutation operators. This evolutionary process continues for predetermined number of generations in the search of the global optimum.

Any efficient local search process can be used to estimate the bounds. Any such estimates will always improve the search. The critical issue here is how efficient it is to compute reasonably correct estimates to guide the search. Caution is to be taken in using the estimates as they could be totally wrong in the beginning of the process and potential search regions can be eliminated completely at the beginning of the evolution. Mutation operator helps recover from this situation.

E. Evolutionary Operators

Many types and versions of evolutionary operators have been proposed in the evolutionary search literature. We used the operators reproduction, crossover, merging, mutation and selection suggested in [8].

E.1. Reproduction

The reproduction operator, p_s, selects two parent interval search vectors, $\bar{a}_i$ and $\bar{a}_k$, to generate an instance of a new offspring interval vector to be used by the crossover operator. The selection of parent interval vectors for reproduction is made using Boltzmann distribution as,

$$p_s(\bar{a}_i) \propto \exp\left(-\frac{f(\bar{a}_i)}{T}\right) \qquad (4)$$

Here $f(\bar{a}_i)$ is the fitness value of the interval vector $\bar{a}_i$, and T is the temperature. Temperature T in the Boltzmann distribution controls the uniformity of the selection of parent interval search vectors. At the beginning of the search process, with high T, the parent selection is uniformly random over the whole domain. This results in having the same selection probability for all intervals in the population. As temperature T is decreased the selection of parents for reproduction narrows down to most promising regions and selects the parents only among those intervals which are better. Using Boltzmann distribution also has advantage of not having to scale the fitness function values [8].

E.2. Crossover

This evolutionary operator generates a new instance of interval search vector, $\bar{\mathbf{a}}_f$ from two parent interval search vectors, $\bar{\mathbf{a}}_i$ and $\bar{\mathbf{a}}_j$, selected for reproduction, by swapping their components. A multi-point crossover is used in generating new offspring interval search vector. The centers $\bar{a}^c_{fj}, j = 1, ..., n$, of each interval component $\bar{a}_{fj}$ in the offspring interval vector, $\bar{\mathbf{a}}_f$, is chosen with same probability from either the center, $\bar{a}^c_{ij}$ or $\bar{a}^c_{kj}$ of the parent interval vectors $\bar{\mathbf{a}}_i$ and $\bar{\mathbf{a}}_k$. The widths, w_{fj}, for offspring interval vector are assigned the values of the corresponding widths of the parents, selected in above process.

$$\bar{a}^c_{fj}, w_{fj} = \begin{cases} \bar{a}^c_{ij}, w_{ij} & \text{with probability } 0.5 \\ \bar{a}^c_{kj}, w_{kj} & \text{with probability } 0.5 \end{cases} \qquad (5)$$

E.3. Merging

Occasionally, with some probability, the merging operator is applied in the evolutionary process. A new offspring interval search vector is generated from two parent vectors selected by the reproduction operator by taking the intersection (when not empty) of the hyper-rectangles they represent. This operator points out most promising regions of the domain that the algorithm encounters in the search process [8]. Probability of application of this operator is kept low.

E.4. Mutation

This operator only modifies the center of interval components in a interval search vector and does not change the their widths. This generates interval vectors in the new regions of the search space. Mutation is the only operator that generates "new genetic material" in the evolutionary process. Crossover and merging use existing "genetic material". Each interval vector in the population is subjected to Mutation with mutation probability, p_c. Once selected for Mutation, each of its component is selected for Mutation with probability 0.5. If the component is selected for Mutation the center of the interval component is mutated within the associated interval or outside the interval (in the whole domain) with some probability. In our implementation within interval mutation probability is kept 0.8 and outside interval mutation probability is kept 0.2. Mutation within the interval (whose width reduces over time) or within the whole domain helps in overcoming the problem associated with fixed size population where new generation falls in the same old traps and slows the progress.

E.5. Local Optimization

During the first cycle of the evolutionary process, the local optimisation process starts from the initially generated random solutions. These initial points have interval widths as large as the whole domain. The local optimisation process returns the local optimum found. These local optimum become the solutions of next generation. In subsequent generations the local optimisation process is started with crossed and mutated versions of the local optimum found in the previous generations. The local optimisation process is similar to the one suggested in [7] except that the resolution parameter is made adaptive. The adaptive resolution parameter improves the efficiency of the local optimiser by reducing the number of function evaluations at the beginning of the evolutionary process, which in many cases are local optima. The resolution parameter is reduced slowly as number of generations increase and the main algorithm expects more and more resolution from the local optimiser procedure. The basis local optimiser used is given below. Here parameter *Resolution* is decreased slowly as the evolution proceeds.

```
initialise $\vec{v}$
while $|\vec{v}| \geq Resolution$
    $iter = 0$
    while $f(\vec{x} + \vec{v}) > f(\vec{x})$ and $iter \leq$ MAXITER
        $\vec{v}$ = NewDirection($\vec{v}$)
        $iter = iter + 1$
    if $f(\vec{x} + \vec{v}) > f(\vec{x})$
        $\vec{v} = \vec{v}/2$
    else
        $\vec{x} = \vec{x} + \vec{v}$
        $\vec{v} = 2\vec{v}$
return $\vec{x}$
```

Here $\vec{v}$ is a step vector and grows or shrinks according to the recent progress made. MAXITER determines iterations at each step before shrinking the vector. *Resolution* is the smallest step size allowed [7]

E.6. Selection

At iteration t of the evolutionary process, after applying reproduction, crossover, merging and mutation to the μ interval vectors in $P(t) = \{\bar{\mathbf{a}}_1(t), ..., \bar{\mathbf{a}}_\mu(t)\}$. After reproduction, crossover, merging and mutation, λ new offspring are created from $P(t)$. Let $P'(t) = \{\bar{\mathbf{a}}'_1(t), ..., \bar{\mathbf{a}}'_\mu(t), \bar{\mathbf{a}}'_{\mu+1}(t), ..., \bar{\mathbf{a}}'_{\mu+\lambda}(t)\}$. To create the population, $P(t+1)$, μ interval vectors form $P'(t)$

are chosen based on the selection operator using Metropolis criterion. If $\bar{\mathbf{a}}_n(t+1), n = 1, \ldots, \mu$, is the nth interval in next generation of interval search vectors and ξ is a random number in [0,1], then, $\bar{\mathbf{a}}_n(t+1)$ is

$$\bar{\mathbf{a}}_n(t+1) = \begin{cases} \bar{\mathbf{a}}_{\mu+n}(t) & \text{if } \xi \le p(\bar{\mathbf{a}}_n(t)) \\ \bar{\mathbf{a}}_n(t) & \text{if } \xi > p(\bar{\mathbf{a}}_n(t)) \end{cases} \qquad (6)$$

where $p(\bar{\mathbf{a}}_n(t))$ is

$$p(\bar{\mathbf{a}}_n(t)) = \exp\left(-\frac{f(\bar{\mathbf{a}}_{\mu+n}(t)) - f(\bar{\mathbf{a}}_n(t))}{T}\right), \qquad (7)$$

where $n = 1, \ldots, \mu$. The Metropolis criterion prefers interval search vectors with better fitness values, but allows local minima to be overcome by accepting uphill moves [8].

E.7. Annealing of Temperature

In the above evolutionary operators, temperature T, controls the reproduction and selection process. With high temperature, random search takes place while with very low temperature algorithm usually gets trapped in a local optimum. A simple method of adaptive temperature suggested in [8] is used. The temperature $T(t)$ is updated after N iterations as, $T(t+N) = T(t)/\alpha$. The parameter $\alpha < 1$ is fixed. Every time the temperature is updated the new temperature is compared with $T_{min} \cdot G$, where $T_{min} \ll 1$, is a small fixed parameter and G is the geometric mean of the differences between the fitness function values in the current population and the optimum fitness value, $f(\bar{\mathbf{a}}_*)$ up to the moment.

$$G = \left(\prod_{j=1}^{\mu}(f(\bar{\mathbf{a}}_j - f(\bar{\mathbf{a}}_*))\right)^{\frac{1}{\mu}} \qquad (8)$$

In this process, if $T(t+N) < T_{min} \cdot G$, then, $T(t+N) = \max(G, \mid f(\bar{\mathbf{a}}_*) \mid)$ is chosen and the search is continued. The same method is used in the initialisation process.

E.8. Updating of Widths

The widths of intervals components of interval search vectors are updated as every time an interval search vector $\bar{\mathbf{a}}_i$ is better than the current optimum $\bar{\mathbf{a}}_*$, its width is changed as follows [8]:

$$w_{ij} = w_{ij} \cdot \left(1 + \frac{\mid \bar{a}_{*j} - \bar{\mathbf{a}}_{ij} \mid}{\mu \cdot \max_j \mid \bar{a}_{*j} - \bar{\mathbf{a}}_{ij} \mid}\right) \qquad (9)$$

This increases the widths mostly in the direction of current optimum.

III. Experiments and Results

We demonstrate the performance of the interval based evolutionary search strategy on a set of well known optimisation problems. These functions have been typically used as benchmark functions in evaluating optimisation algorithms. Details on these test functions are given in section IV. The test function are referred to by their names or the authors who first proposed these problems as benchmark problems. The test suite contains functions that are

- Continuous and Discontinuous
- Convex and Concave
- Unimodal and Multi-modal
- Linear and nonlinear
- Low Dimensional and High Dimensional
- Deterministic and stochastic

Results reported over these test functions are averaged over 20 independent runs with different random initial populations. In all the example, global optimum is know and this information was used by termination criterion. In all cases, optimum found was required to be within 10^{-5} to the true global optimum along all coordinates.

Following parameter values were used in testing the performance. Population size $\mu = 20$, crossover probability $p_c = 0.2$, merging probability $p_{mr} = 0.005$, Number of generations before updating temperature $N = 100$, Number of generations before updating widths $= 50$, Temperature update coefficient $\alpha = 1.5$, and width update coefficient $\alpha = 2$. Same set of parameters were used in optimising all test functions.

It was observed that in most cases the algorithms is able to find good direction for search in very few generations. Most cycles are taken to get the required precision of the optimum. For example in Ackley's function f_{12}, the optimum close to true optimum by 10^{-1} was found consistently in few hundred function evaluations along all coordinates. The large number of evaluations required after this point are to get to the required quality of optimum which is as close as 10^{-5} to true optimum.

In comparison, a multi-population genetic algorithm, PGA v2.7 [31] was used. Following parameters were used in PGA. Number of populations = 5, Number of individuals in each population = 20, Number of bit per variable = 16, Rank selection was used. Mutation rate = 0.005, Two-point crossover was used and Migration interval = 10. Same set of parameters were used in optimising all test functions. No convergence situation is indicated by a "?".

The table summarises results over a set of benchmark functions. In the first column, $f(\vec{x})$ is the name of the functions, in the second column, n is the number of dimensions, in the third column, IESA indicates the number of function evaluations used by the Interval Evolutionary Search Algorithm (IESA) without the local optimisation process. The fitness in this algorithm is evaluated at the center of the interval vectors. In the fourth column, PGA indicates the number of function evaluations required by the Parallel Genetic Algorithm [31]. In the fifth column, IESA+HC indicates the number of function evaluations required by the Interval Evolutionary Search Algorithm (IESA) with the local optimisation process.

First thing to notice in the table is the robustness of the IESA+HC approach. It converges in all cases while the IESA and PGA do not. In many cases the IESA+HC approach is order of magnitude faster then the IESA and PGA approach. In the case of f_7 the bad performance of IESA+HC approach can be attributed to the large number of local optima. At the end of every generation, the population in IESA+HC approach consists of local optima. These are then crossed and mutated and sent to the local optimiser in the next generation. If the population size is made dynamic, the IESA+HC approach will collect more and more local optima at every generation. Any attempt in generating new points far away from these collected local optimum will increase the efficiency in this case. Due to fixed size population in the present implementation individuals in the new generations fall back into the same local optimum they started from and progress is halted or made slowly. Example of such situation is seen in Shekel's Foxhole function, f_5.

$f(\vec{x})$	n	IESA	PGA	IESA+HC
f_1	2	4388	2212	90
	15	21904	17525	551
	30	50012	50316	1313
f_2	2	5206	2550	343
	4	?	?	27172
f_3	5	2398	1987	889
f_4	2	2420	?	78
	4	3940	?	117
	8	5982	?	242
	16	35278	?	511
	32	?	?	1084
f_5	2	8386	2212	3481
f_6	2	?	?	92
	4	12236	7935	207
	8	16513	17240	837
	16	42749	45475	22970
	32	?	?	56297
f_7	2	?	?	32934
	4	18297	7900	42948
	8	28226	18620	50598
	16	?	?	59464
	32	?	?	1262
f_8	2	41901	2312	258
f_9	2	1956	4355	178
f_{10}	20	88981	117200	25407
f_{11}	2	7200	3811	113
	4	?	?	405
	8	?	?	1375
	20	?	170000 (?)	27130
f_{12}	15	26437	48033	1599
	30	?	113950	6057
f_{13}	4	28865	6675	22
f_{14}	2	11462	4762	675
	10	?	?	7627
f_{15}	4	41431	20870	75437
f_{16}	9	1670	?	975
	16	2280	?	4165

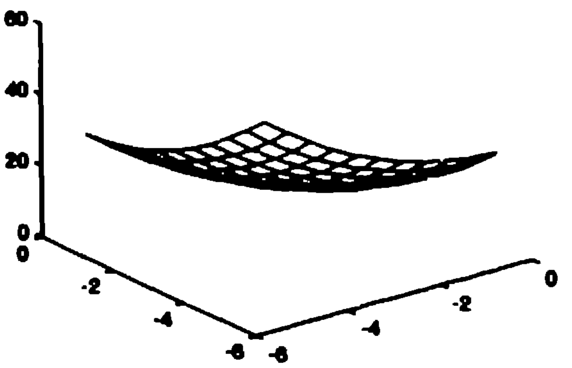
IV. Test Functions

The test functions used in testing the optimization algorithm are given below.

- f_1: *Sphere Model [1]*

$$f(\vec{x}) = \sum_{i=1}^{n} x_i^2 \qquad (10)$$

$n = 30; -5.12 \leq x_i \leq 5.12; \min(f) = f(0, ..., 0) = 0.$



The *sphere model* is smooth, unimodal, symmetric and does not have any problems of a ridge, plateau or a foothill. The performance on this model is a measure of the general efficiency of the optimisation algorithm.

- f_2: *Rosenbrock's function [1]*

$$f(\vec{x}) = \sum_{i=1}^{n-1} 100 \cdot (x_{i+1} - x_i^2)^2 + (x_i - 1)^2 \tag{11}$$

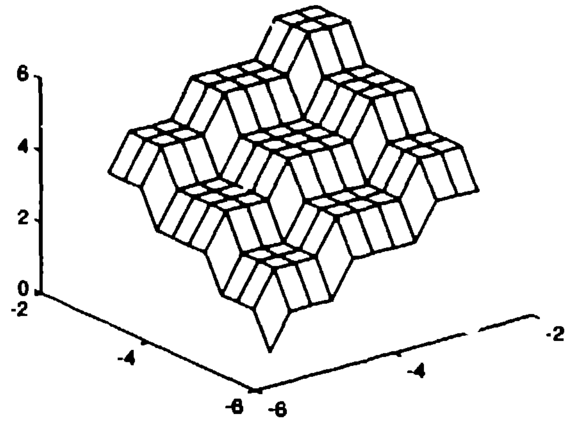
$-5.12 \le x_i \le 5.12; \min(f) = f(1, ..., 1) = 0.$

This is a continuous, unimodal and bi-quadratic function (Figure 1). It has a very narrow ridge with a very sharp tip and runs around a parabola. The progress of many algorithms is very slow and are unable to discover good direction.

- f_3: *Step Function [1]*

$$f(\vec{x}) = 6n + \sum_{i=1}^{n} \lfloor x_i \rfloor \tag{12}$$

$-5.12 \le x_i \le 5.12;$
$\min(f) = f([-5.12, -5), ..., [-5.12, -5)) = 0.$

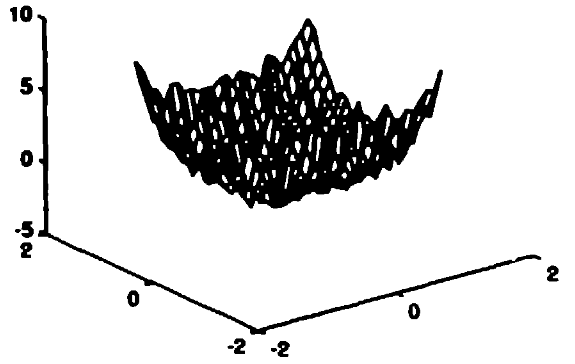


This is a representative of plateau problem with linear and discontinuous properties. Flat surfaces do not give any information as to which direction is favorable. If the algorithms does not have variable step size, optimising functions like this is a problem.

- f_4: *Quartic function with noise [1]*

$$f(\vec{x}) = \sum_{i=1}^{n} i x_i^4 + \text{gauss}(0, 1) \tag{13}$$

$-1.28 \le x_i \le 1.28; \min(f) = f(0, ..., 0) = 0$



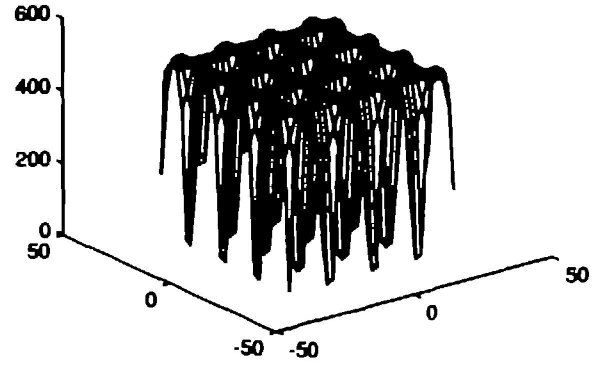
This is simple function padded with noise. The Gaussian noise results in different value of the function at the same point when sampled at separate instances. This makes it a good noise resistance test for optimisation algorithm which is a necessary part of many real world problems.

- f_5: *Shekel's Foxholes [1]*

$$\frac{1}{f(\vec{x})} = \frac{1}{K} + \sum_{j=1}^{25} \frac{1}{c_j + \sum_{i=1}^{2} (x_i - a_{ij})^6} \tag{14}$$

$$(a_{ij}) = \begin{pmatrix} -32 & -16 & 0 & 16 & 32 & -32 & .. & 0 & 16 & 32 \\ -32 & -32 & -32 & -32 & -32 & -16 & .. & 32 & 32 & 32 \end{pmatrix}$$

$K = 500; f(a_{1j}, a_{2j}) \approx c_j = j;$
$-65.536 \le x_i \le 65.536; \min(f) = f(-32, -32) = 1.$



This is a continuous, nonlinear, multimodal function with several local optima. It is difficult for optimisation algorithms because it has large plateau with equal function value with 25 narrow holes which have a different function value at their bottoms. Many algorithms get stuck in the first hole they find.

- f_6: *Plateau function [3]*

$$f(\vec{x}) = \sum_{j=1}^{4} 2500 \cdot \max_i \lfloor 1000 \cdot \mid x_i \mid \rfloor \qquad (15)$$

$(j-1)h < i \leq jh; h = n/4;$
$\min(f) = f(\max_{1 \leq i \leq n} \mid x_i \mid < 10^{-3}) = 0.$

The *Plateau* function has large number of flat regions whose value gradually decreases towards the global minimum near the origin.

- f_7: *Porcupine function [3]*

$$f(\vec{x}) = 10^4 \cdot (c + 1.5z) \qquad (16)$$

$c = 10^{-3} \sum_{i=1}^{n} \mid x_i \mid$
$z = 10^6(n-c) - 2 \cdot \left\lfloor \frac{10^6(n-c)}{2} \right\rfloor$
$\min(f) = f(0, ..., 0) = 0.$

In this function, every time $\lfloor 10^6(n-c) \rfloor$ is an even number there is a local minimum, leading to very large number of local minima. Figure. 2 shows the function and the location of the global optimum. The value of $f(\vec{x})$ decreases slowly towards the global minimum (the origin). The high density of local minima here is a major problem for many optimisation algorithms.

- f_8: *Sines function*

$$f(\vec{x}) = 1 + \sin^2(x_1) + \sin^2(x_2) - 0.1\exp(-x_1^2 - x_2^2) \qquad (17)$$

$-10 \leq x_1, x_2 \leq 10; \min(f) = f(0,0) = 0.9.$

This function is continuous and multi-modal with large number of local minima with very small different in their values. Figure. 3 shows the function and the location of the global optimum.

- f_9: *Goldstein and Price*

$$\begin{aligned} f(\vec{x}) &= [1 + (x_1 + x_2 + 1)^2(19 - 14x_1 + 3x_1^2 \\ &\quad -14x_2 + 6x_1x_2 + 3x_2^2)] \\ &\quad \cdot[30 + (2x_1 - 3x_2)^2(18 - 32x_1 \\ &\quad +12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2)] \qquad (18) \end{aligned}$$

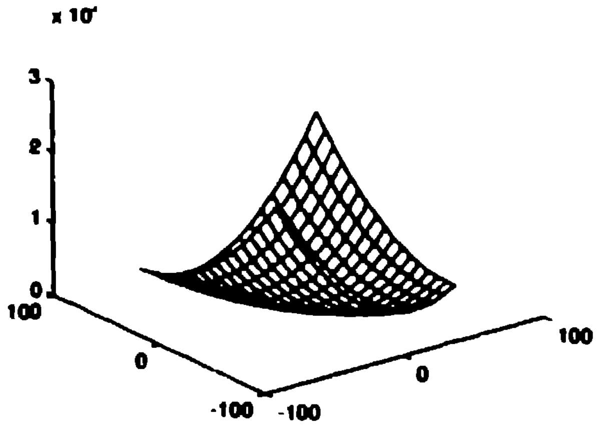
$-2 \leq x_1, x_2 \leq 2; \min(f) = f(0,-1) = 3.0.$

This function is continuous, multi-modal and nonlinear. Figure. 4 shows the function and the location of the global optimum. The problems optimisation algorithms face with this function is the peak response of about five orders of magnitude greater in the neighborhood of the optimum.

- f_{10}: *Schwefel's Function*

$$f(\vec{x}) = \sum_{i=1}^{n} \left(\sum_{j=1}^{i} x_j \right)^2 = x^T A x + b^T \qquad (19)$$

$n = 20; -65.536 \leq x_i \leq 65.536;$
$\min(f) = f(0, ..., 0) = 0.$



This is a continuous and unimodal function. The difficulty here is that searching along the coordinate axes only given poor rate of convergence, since the gradient of the function is not oriented along the axes. The valley here is much narrow (as compared to Rosenbrock's function) due to the worsening conditions of the form matrix **A** with increasing n [29].

- f_{11}: *Rastrigin's Function*

$$f(\vec{x}) = 10n + \sum_{i=1}^{n} x_i^2 - A\cos(2\pi x_i) \qquad (20)$$

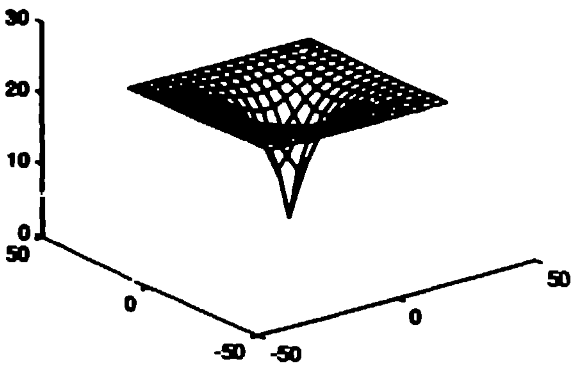
$n = 2; -5.12 \leq x_i \leq 5.12; \min(f) = f(0, ..., 0) = 0.$

This is a scalable, continuous, multi-modal test function which is made from the *Sphere model* by modulating it with $A\cos(2\pi x_i)$ [29]. Figure. 5 shows the function and the location of the global optimum. Far away from the origin this functions looks like the *Sphere model*, but with smaller x_i the effect of the modulation grows and dominates the shape. The multi-modality here present substantial difficulty to many optimization algorithms.

- f_{12}: *Ackley's Function [3]*

$$\begin{aligned} f(\vec{x}) &= -20\exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\right) \\ &\quad -\exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)\right) + 20 + e \qquad (21) \end{aligned}$$

$n = 30; -30 \leq x_i \leq 30; \min(f) = f(0, ..., 0) = 0$



This is a generalised variant of a multi-modal function by [3]. This function has been transformed such that the global minimum is located at the origin.

- f_{13}: *Nowhere Differentiable Function [17]*

$$f(\vec{x}) = \prod_{k=1}^{D}(1 + k\sum_{n=0}^{\beta} \frac{|\,2^n x_k - \lfloor 2^k x_k \rfloor\,|}{2^n} \qquad (22)$$

$n = 4; \beta = 60; -1000 \leq x_i \leq 1000;$
$\min(f) = f(0,...,0) = 1$. This function ($\beta = \infty$) is continuous but nowhere differentiable. It is a D dimension variation of contraction mapping defined in [17]. This test function has infinite number of local minima.

- f_{14}: *Griewank 1*

$$f(\vec{x}) = \frac{1}{d}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n}\cos\left(\frac{x_i}{\sqrt{i}}\right) + 1. \qquad (23)$$

$d = 200; n = 2; -100 \leq x_i \leq 100;$
$\min(f) = f(0,...,0) = 0.$
Figure. 6 shows the function and the location of the global optimum.

- f_{15}: *Colville*

$$\begin{aligned} f(\vec{x}) &= 100 \cdot (x_1 - (x_0^2)) \cdot (x_1 - (x_0^2)) & (24)\\ &+(1 - x_0) * (1 - x_0) + 90 \cdot (x_3 - (x_2^2)(x_3^2 & (25)\\ &+(1 - x_2)^2 + 10.1 * ((x_1 - 1.)^2 & (26)\\ &+(x_3 - 1)^2) + 19.8 * (x_1 - 1) * (x_3 - & (27) \end{aligned}$$

$n = 4; -1000 \leq x_i \leq 1000;$
$\min(f) = f(1,..,1) = 0.$

i	a_i	b_i^{-1}	i	a_i	b_i^{-1}
1	0.1957	0.25	7	0.0456	8
2	0.1947	0.5	8	0.0342	10
3	0.1735	1	9	0.0323	12
4	0.1600	2	10	0.0235	14
5	0.0844	4	11	0.0246	16
6	0.0627	6			

- f_{16}: *Neural Networks: Parity Function*

In this test, weights of a standard 2 input 2 hidden and one output node neural network architecture are found by finding the optimum of the error function defined over a model of the parity function. The total of 9 weights are used in the network. The error function is

$$f(\vec{w}) = \sum_{i=1}^{9}(t_i - o(\vec{w}))^2 \qquad (28)$$

is the sum of squared error between the actual output t_j and the produced output $o(\vec{w})$ is over 4 patterns of the parity (XOR) function. Similarly the weights of 4 input 4 hidden and one output node neural network were found to minimise the error for the 4 input parity function. The number of weights here are 25.

V. Conclusions and Future Research

One of our research goal is to use the advantages of interval computations in search and optimisation algorithms without any assumption about the underlying objective function. Many techniques from mathematical programming, statistics, machine learning, neural networks, and evolutionary algorithms have been proved to be robust in solving a wide variety of optimisation problems. These techniques can be extended to take advantage of techniques from interval computations.

An efficient approach that combines the efficient strategy from Interval Global Optimisation Methods and robustness of the Evolutionary Algorithms is proposed. In the proposed approach, search begins with randomly created interval vectors with interval widths equal to the whole domain. Before the beginning of the evolutionary process, fitness of these interval parameter vectors is defined by evaluating the objective function at the center of the initial interval vectors. In the subsequent evolutionary process the local optimisation process returns an estimate of the bounds (lower or upper for minimisation or maximisation) of the objective function within the interval vectors. Though these bounds may not be correct at the beginning due to large interval widths, the process of reducing interval widths over time and a selection approach similar to simulated annealing helps in estimating reasonably correct bounds. The interval parameter vectors at these estimated bounds (local optima) are then subjected to crossover and mutation. This evolutionary process continues for predetermined number of generations. Using an efficient method to find the bounds of the function in the sub-domain would add significant power to this approach. In the proposed approach a simple hill climber is used for this purpose.

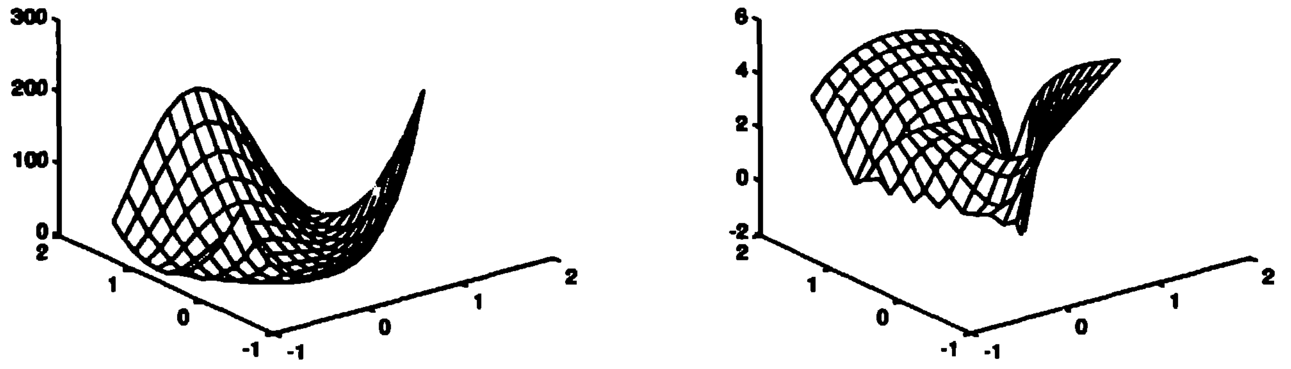


Figure 1: Rosenbrock's function

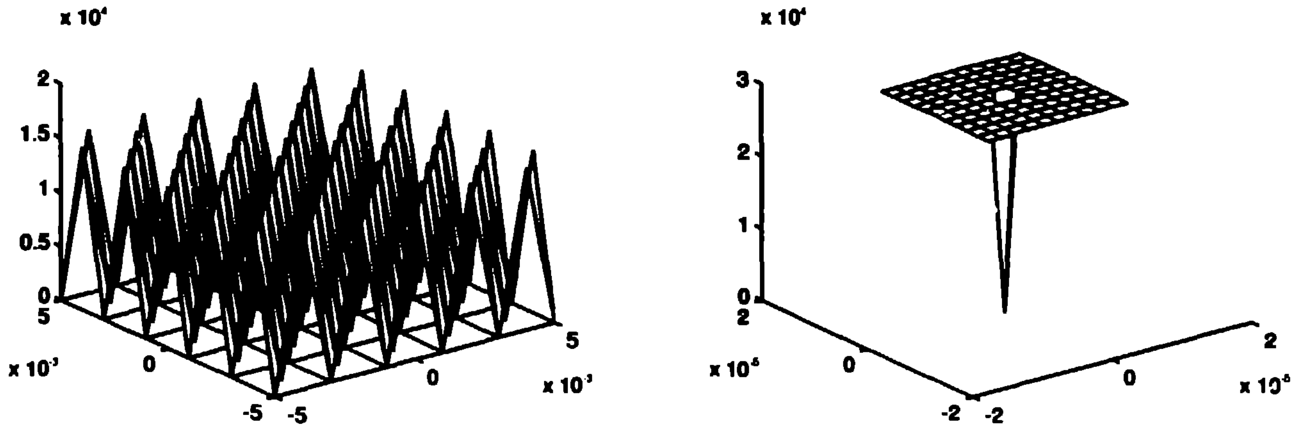


Figure 2: Porcupine function

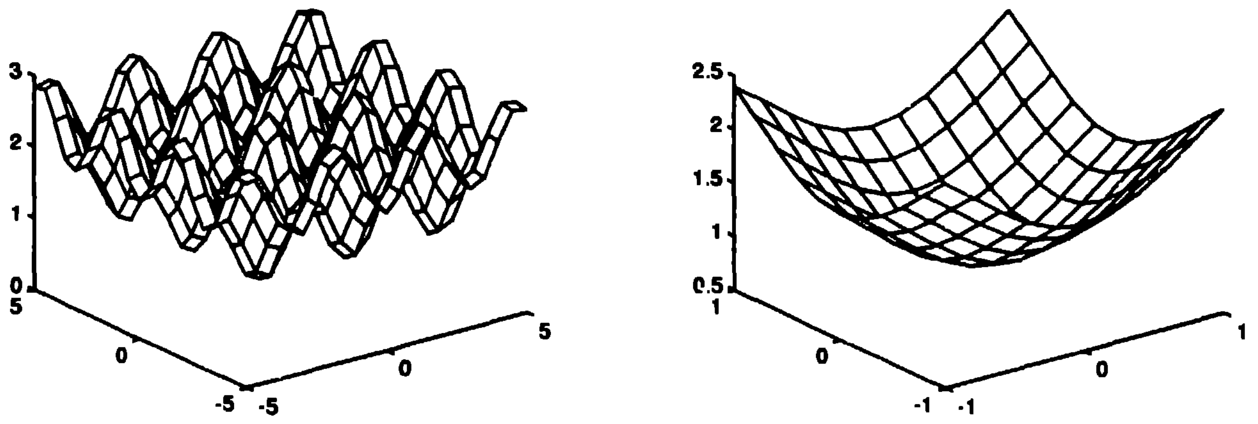


Figure 3: Sines function

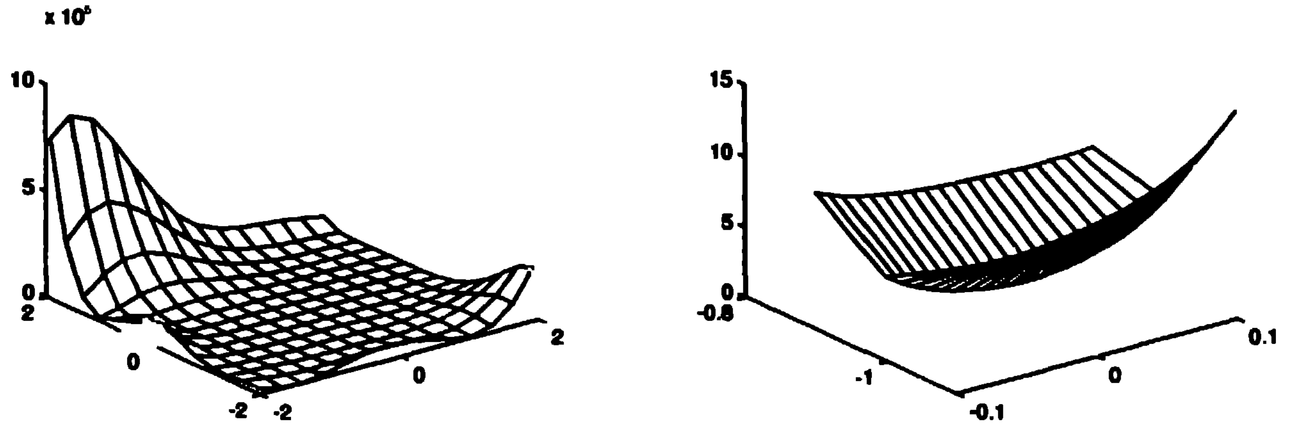


Figure 4: Goldstein and Price function

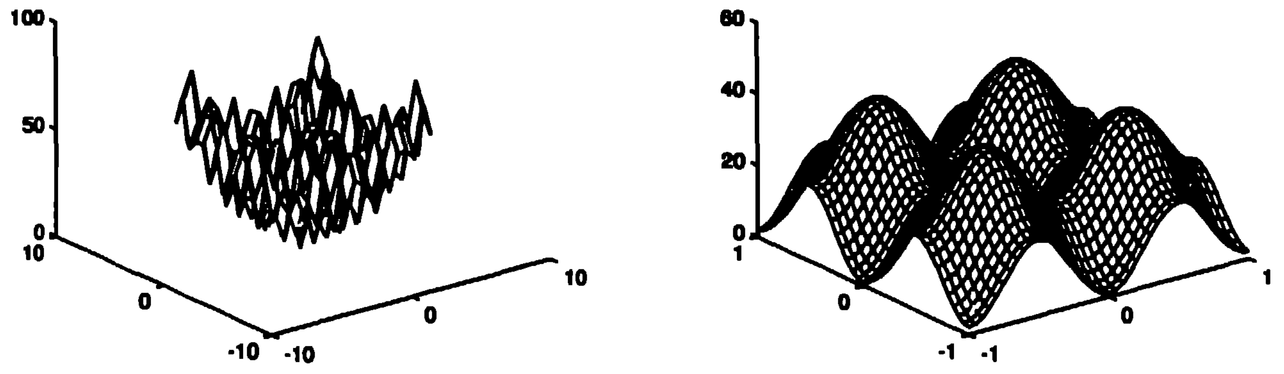


Figure 5: Rastrigin function

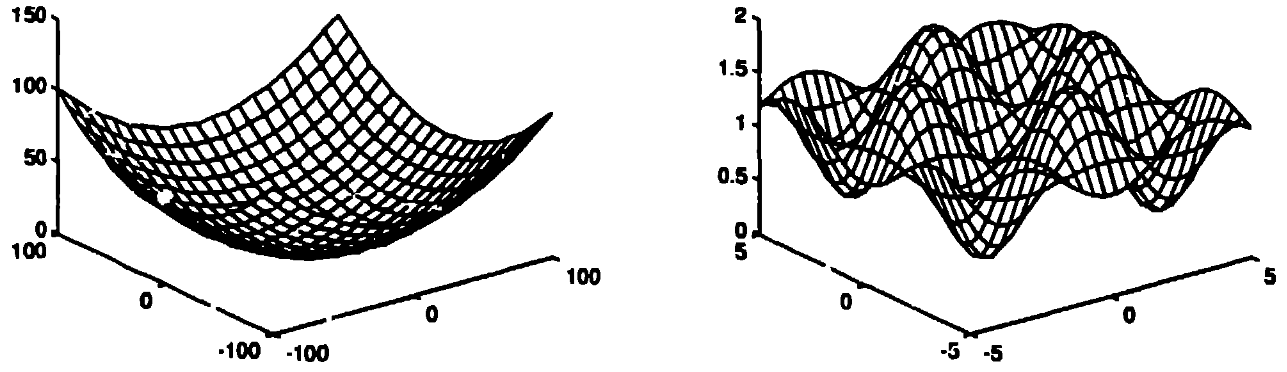


Figure 6: Griewank function

In the proposed approach, though the verified nature of the Interval Global Optimisation Methods is not preserved, empirical results over standard benchmark functions has shown to have preserved the efficiency and robustness properties of the two approaches. The proposed approach is benchmarked against Parallel Genetic Algorithm and is observed to be more robust and in some cases order of magnitude efficient.

In conclusion, it is observed that extension of one evolutionary algorithm with a local optimisation process, simulated annealing type selection process and interval mutation operator significantly improves the efficiency and robustness of the search process.

References

[1] K. De Jong, "An Analysis of the Behavior of a Class of Genetic Adaptive Systems", PhD thesis, *University of Michigan*, Diss. Abstr. Int. 36(10), 5140B, University Microfilms No 76-9381, 1975.

[2] D. E. Goldberg, "Real-Coded Genetic Algorithms, Virtual Alphabets, and Blocking", *Complex Systems*, 5, pp. 139-167, 1993.

[3] D. J. Ackley, "An Empirical Study of Bit Vector Function Optimisation", in *Genetic Algorithms and Simulated Annealing* edited by L. Davis, London, Pitman, pp. 194-200, 1987.

[4] A. A. Zhiglijavsky, **Theory of Global Random Search**, Kluwer Academic Publishers, 1991.

[5] H. Ratschek and J. Rokne, **Computer Methods for the Range of Functions**, Halsted Press, NY, 1984.

[6] E. Hansen, **Global Optimisation Using Interval Analysis**, Marcel Dekker, NY, 1994.

[7] D. Yuret, "From Genetic Algorithms to Efficient Optimisation", *Masters Thesis*, Dept. of Electrical Engineering, MIT, May 1994.

[8] M. Muselli and S. Ridella, "Global Optimisation of Functions with the Interval Genetic Algorithms", *Complex Systems*, 6, pp. 193-212, 1992.

[9] A. J. Keane, "Experiences with Optimisers in Structural Design", Dept. of Engineering Science *andy.keane@eng.ox.ac.uk*, University of Oxford, Oxford, UK, 1994.

[10] D. E. Goldberg, **Genetic Algorithms in Search, Optimisation and Machine Learning**, Addison Wesley, Reading, MA, 1989.

[11] D. E. Rumelhart, J. L. McClelland and the PDP Research Group, **Parallel Distributed Processing Vol 1** MIT press Cambridge, 1988.

[12] C.A. Floudas, and P.M. Pardolos, **Recent Advances in Global Optimisation** Princeton Series in Computer Science, Princeton University Press, Princeton, NJ, 1992.

[13] G. Alefeld and J. Hersberger, **Introduction to Interval Computations**, Academic Press, NY, 1983.

[14] R. Fletcher, **Practical Methods of Optimisation**, John Wiley & Sons, 1987.

[15] J. H. Holland, **Adaptation in Natural and Artificial Systems**, The University of Michigan Press, Ann Arbor, 1975.

[16] L. J. Fogel, A. J. Owens, and M. J Walsh, "Artificial Intelligence through Simulated Evolution", Jonh Wiely, New York, NY, 1966.

[17] H. Katsuura, "Continuous No-where Differentiable Function-An Application of Contraction Mappings", *The American Mathematical Monthly*, Vol. 98, no. 5, 1991.

[18] J. H. Holland, "Escaping Brittleness: The possibilities of General-Purpose Learning Algorithms Applied to Parallel Rule-Based Systems", In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, editors, *Machine Learning* Vol. II, chapter 20, pp: 593-623. Morgan Kaufmann Publishers, 1986.

[19] J. R. Kosa, "Hierarchical Genetic Algorithms Operating on Populations of Computer Programs", In N. S. Sridharan, editor, *Eleventh international joint conference on artificial intelligence*, Morgan Kaufmann Publishers, pp: 768-774, 1989.

[20] I. Rechenberg, "Evolutionsstrategie: Optimierung Technisher Systeme Nach Prinsippine der Biologischen Evolution", Frommann-Holsboog Verlag, Stuttgart, 1973.

[21] Hans-Paul Schwefel, **Numerical Optimisation of Computer Models**, Wiely, Chichester, 1981.

[22] J. J. Grefenstette, (Editor) *Proceedings of the First int'l Conference on Genetic Algorithms and Applications*, Hillsdale, NJ, Lawrence Erlbaum, 1985.

[23] J. J. Grefenstette, (Editor) *Proceedings of the Second int'l conference on genetic algorithms and applications*, Hillsdale, NJ, Lawrence Erlbaum, 1987.

[24] J. D. Schaffer, (Editor) *Proceedings of the Third int'l Conference on Genetic Algorithms and Applications*, San Mateo, CA, Mogan Kaufmann, 1989.

[25] R. K. Belew and L. B. Booker, (Editor) *Proceedings of the Fourth int'l Conference on Genetic Algorithms and Applications*, San Diego,

CA, Mogan Kaufmann, 1991.

[26] D. B. Fogel and J. W. Atmar, (Editors) *Proceedings of the First Annual Conference on Evolutionary Programming*, La Jolla, CA, Evolutionary Programming Society, 1992.

[27] H.-P. Schwefel and R. Männer, (Editors), *Parallel Problem Solving from Nature - Proc. 1st workshop PPSN I*, I, Vol. 496 of *Lecture notes in computer science*, Berlin, Springer, 1991.

[28] R. Männer and B. Manderick, (Editors), *Parallel Problem Solving from Nature*, 2, Amesterdam, Elsevier, 1992.

[29] T. Bäck, F. Hoffmeister and H.-P. Schwefel, "Applications of Evolutionary Algorithms", report of the Systems Analysis Research Group SYS-2/92. University of Dortmund, Department of Computer Science, 1992.

[30] T. Bäck, "A User's Guide to *GENEsYs 1.0*", report of the Systems Analysis Research Group, University of Dortmund, Department of Computer Science, 1992.

[31] P. Ross, "About PGA v2.7", Dept of AI, *University of Edinburgh*, June 1994.