Direction Matters in High-Dimensional Optimisation

Cara MacNish, *Member, IEEE*, Xin Yao, *Fellow, IEEE*

***Abstract*— Directional biases are evident in many benchmarking problems for real-valued global optimisation, as well as many of the evolutionary and allied algorithms that have been proposed for solving them. It has been shown that directional biases make some kinds of problems easier to solve for similarly biased algorithms, which can give a misleading view of algorithm performance.**

In this paper we study the effects of directional bias for high-dimensional optimisation problems. We show that the impact of directional bias is magnified as dimension increases, and can in some cases lead to differences in performance of many orders of magnitude.

We present a new version of the classical evolutionary programming algorithm, which we call *unbiased evolutionary programming* (UEP), and show that it has markedly improved performance for high-dimensional optimisation.

I. Introduction

Many of the problems used to benchmark real-valued global optimisation algorithms are subject to biases [1], which may in turn give misleading results for the algorithms used to solve them. An example that has shown significant performance differences is *directional bias*, most commonly in the form of *axial bias*, where features of a problem surface are aligned with the Cartesian co-ordinate system in which it is defined. Some solution algorithms are able to exploit this alignment. It has been shown in [2], [3], for example, that traditional bitstring encoded genetic algorithms show improved performance on functions with axial biases, and that performance deteriorates when the function is rotated with respect to the co-ordinate system.

An extreme example of directional bias is problems that are *linearly separable*. An n-dimensional function $f : R^n \mapsto R$ is linearly separable iff:

$$f(x_1, \ldots, x_n) = \sum_{i=1}^{n} f_i(x_i)$$

for some f_i (which may be non-linear), that is it can be defined as a sum of functions of the individual variables. More generally, a function can be defined as decomposable or *separable* [4] iff:

$$\arg\min_{x_1,\ldots,x_n} f(x_1, \ldots, x_n) = \\ \big(\arg\min_{x_1} f(x_1, \ldots), \ldots, \arg\min_{x_n} f(\ldots, x_n)\big)$$

The separability of a problem has a significant impact on algorithm performance.

Cara MacNish is with the School of Computer Science & Software Engineering, University of Western Australia, Nedlands 6009, Australia (cara@csse.uwa.edu.au). Xin Yao is with the School of Computer Science, University of Birmingham, Birmingham B15 2TT, U.K (x.yao@cs.bham.ac.uk).

In this paper we study the effects of directional bias for *high-dimensional* optimisation problems by examining variants of a classical evolutionary programming algorithm [5], [6], [7]. Following [8] we are particularly interested in the range 100 to 1000 dimensions, although many of the trends in our results continue to higher dimensions.

We study two primary variants of the classical algorithm. The first is designed to exploit separability, and demonstrate that such high-dimensional problems can be beneficially reduced to a number of 1-dimensional problems. This highlights the synergy gains that a more "generalist" algorithm needs to make to outperform this reductionist approach.

The second is a new algorithm designed to evolve without directional bias. We show that this leads to significant performance improvements over the classical algorithm, in some cases several orders of magnitude within the range of dimensions studied.

In order to further elucidate the sizeable difference in performance between the algorithms, we additionally examine three "intermediate" algorithms that share features of the primary algorithms. We find, however, that the unbiased algorithm continues to significantly outperform the others on non-separable problems, and all but the reductionist algorithm on separable problems, at high dimensions.

The algorithms studied in this paper are described in Section II. The problem set used to study the algorithms is taken from [8], and is briefly described in Section III. Section IV summarises the main results of the study and discusses their implications. The paper is concluded in Section V.

II. Algorithms

In this paper we are concerned with bounded, constraint-free, real-valued optimisation. A problem is a pair (S, f) where $S \subseteq R^n$ is a bounded set on R^n, and $f : S \mapsto R$ is an n-dimensional *fitness function*. Without loss of generality we assume all problems are stated as minimisation problems. Ideally our goal is to find a point $\boldsymbol{x}_{\min} \in S$ such that $f(\boldsymbol{x}_{\min})$ is a global minimum on S, that is

$$\forall \boldsymbol{x} \in S : f(\boldsymbol{x}_{\min}) \leq f(\boldsymbol{x}).$$

In practice, the goal of our algorithm will be to find the smallest value (closest to $\boldsymbol{x}_{\min}$) that it can before reaching a stopping criterion.

In the following we assume, as specified in [8], that the stopping criterion for the algorithms is a specified maximum number of fitness function evaluations, $E_{\max}$.

Following common practice we will refer to a point $\boldsymbol{x}$ as an *individual*, and a set of individuals as a *population*. When we use information in an individual to generate a new individual, we refer to this as *mutation*, with the former

978-1-4244-1823-7/08/$25.00©2008 IEEE

individual constituting the *parent* and the latter the *offspring*. Choosing a new population from a set of parents and offspring is known as *selection*.

A. Overview of the Algorithms

Our aim is to investigate the effects of directional biases as they apply to high-dimensional problems, and to see how they interact with properties of the optimisation landscapes, in particular separability and modality. To achieve this we use as our "standard" a simple and well-known evolutionary programming algorithm which mutates real-valued vectors using normally distributed random deviates [5], [6]. We will refer to this as *Classical Evolutionary Programming (CEP)* after [7]. The CEP algorithm is also self-adaptive — as well as mutating the objective variables it also mutates the standard deviations which apply to them. Offspring are chosen using tournament selection. The algorithm is provided in Subsection II-B.

Of the seven problems specified in [8], three are separable. Our second standard algorithm is designed to exploit this property. Separability ensures that each of the variables can be solved independently with the others fixed — if the minimum is found for each independent variable then the overall minimum has been determined. Other authors have noted that such problems are more easily solved by certain kinds of algorithms [2], [3]. However this issue is particularly relevant to high-dimensional optimisation, since it allows us to completely avoid the complexity brought about by the high-dimensionality. We simply solve each variable in turn, while arbitrarily fixing all the others. The trade-off is that we have fewer evaluations available for each variable — assuming the evaluations are shared equally between variables, each can be allocated only $E_{\max}/n$ evaluations.

The algorithm, which we denote *Independent Variable EP (IVEP)*, is described in Subsection II-C. The IVEP algorithm provides a good benchmark for the separable problems — for an algorithm to beat it, it must be able to achieve some "synergy" between the variables that outweighs the complexity of solving in higher dimensions. We expect this to be rare on separable problems.

Our third primary algorithm is a new algorithm designed to address the inherent axial bias of CEP. Like many evolutionary approaches, CEP is based on a "genotype" that encodes the information in an individual in terms of components (or unit vectors) in the axial directions of a Cartesian co-ordinate system. These values are mutated in axial directions and then combined to give the "phenotype". This has the effect of biasing the phenotype. As a very simple example of this, consider an individual (point) at the origin of a Cartesian co-ordinate system. Assume we generate an offspring by adding a uniformly distributed random deviate between 0 and 1 in each co-ordinate direction. An equivalent way to say this is that we multiply each axial unit vector, or versor (the genotypes), by a random deviate and obtain an offspring by vector addition (the phenotype). If the resulting offspring were to fall on one of the axes, it would have a maximum displacement of 1 from the parent. If, on the other hand, the offspring were to fall on the diagonal between the axes, it would have a maximum displacement of $\sqrt{2}$. It also follows that the density of points falling along the diagonal is lower than that on the axis.

The directional biases become more extreme as dimension increases. In the n-dimensional case the ratio of maximum "step size" along the axes to maximum step size on the diagonal grows with $\sqrt{n}$. The issue of step size in high dimensions is in fact a very interesting topic in its own right and includes additional affects which we do not have space to address in the present work.

To avoid the axial biases we construct an algorithm, called *Unbiased EP (UEP)*, in which the genotype mutations are encoded as a magnitude and $n-1$ angles, ensuring step sizes in all directions are (on average) equal. The angles are chosen from uniform distributions, ensuring that all directions are selected with equal probability. The magnitude is mutated with a Gaussian distribution. The algorithm is described in Subsection II-D.

The algorithms in Subsections II-E to II-G are derivatives of the three primary algorithms aimed at reducing the differences between them and thereby isolating the causes of differences in performance.

B. Classical Evolutionary Programming (CEP)

The classical evolutionary programming algorithm proceeds as follows:

1) Generate an initial population of p individuals, where each individual is a pair of real-valued vectors $(\boldsymbol{x}_i, \boldsymbol{\sigma}_i)$, $i = 1, \ldots, p$. Each $\boldsymbol{x}_i$ represents an n-dimensional vector of objective variables. Each $\boldsymbol{\sigma}_i$ represents an n-dimensional vector of standard deviations.
   Choose each element $x_i(j)$ of $\boldsymbol{x}_i$, $j = 1, \ldots, n$, randomly from a uniform distribution between the problem bounds. Set each element $\sigma_i(j)$ of $\boldsymbol{\sigma}$, $j = 1, \ldots, n$, to the value σ_{initial}.
   Set the maximum number of generations, $k_{\max} = E_{\max}/p$, and the current generation, $k = 1$.
2) Evaluate the fitness score of each individual $(\boldsymbol{x}_i, \boldsymbol{\sigma}_i)$ according to the fitness function $f(\boldsymbol{x}_i)$.
3) Each parent $(\boldsymbol{x}_i, \boldsymbol{\sigma}_i)$ creates a single offspring $(\boldsymbol{x}'_i, \boldsymbol{\sigma}'_i)$ as follows:

$$x'_i(j) = x_i(j) + N_{ij}(0, \sigma_i(j)) \tag{1}$$
$$\sigma'_i(j) = \sigma_i(j) e^{\tau' N_i(0,1) + \tau N_{ij}(0,1)} \tag{2}$$

   where $x_i(j)$, $x'_i(j)$, $\sigma_i(j)$ and $\sigma'_i(j)$ denote the j-th components of vectors $\boldsymbol{x}_i$, $\boldsymbol{x}'_i$, $\boldsymbol{\sigma}_i$ and $\boldsymbol{\sigma}'_i$ respectively. $N(\mu, \sigma)$ denotes a normally distributed random number with mean μ and standard deviation σ. N_i indicates that the random number is generated anew for each value of i, while N_{ij} is generated anew for each value of i and j. The factors τ and τ' are commonly set to $(\sqrt{2\sqrt{n}})^{-1}$ and $(\sqrt{2n})^{-1}$ [5], [6].
4) Calculate the fitness of each offspring $(\boldsymbol{x}'_i, \boldsymbol{\sigma}'_i)$.
5) Conduct pairwise comparison over the union of parents $(\boldsymbol{x}_i, \boldsymbol{\sigma}_i)$ and offspring $(\boldsymbol{x}'_i, \boldsymbol{\sigma}'_i)$, $i = 1, \ldots, n$. For each individual, q opponents are chosen uniformly at random from all the parents and offspring. For each

comparison, if the individual's fitness is no smaller than the opponent's, it receives a "win".

6) Select the p individuals out of $(\boldsymbol{x}_i, \boldsymbol{\sigma}_i)$ and $(\boldsymbol{x}'_i, \boldsymbol{\sigma}'_i)$ that have the most wins to be the parents of the next generation.
7) Set $k \leftarrow k+1$. Stop if the halting criterion ($k = k_{\max}$) is satisfied, otherwise go to Step 3.

C. Independent Variable EP (IVEP)

IVEP is designed as closely as possible to CEP, except that it solves each variable separately. The algorithm simply calculates the number of evaluations available for each dimension, and then uses CEP to solve for that dimension while mutations in other dimensions are set to zero.

1) Generate an initial population of p individuals as per CEP Step 1.
2) Evaluate the fitness of the initial population as per CEP Step 2.
3) Set the current dimension, $d = 1$. Set the number of generations per dimension, $k_{d,\max} = E_{max}/p/n$.
4) If $d > n$ halt, otherwise set:

$$\sigma(i,j) = \begin{cases} \sigma_{\text{initial}} & j = d \\ 0 & \text{otherwise} \end{cases}$$

for $i = 1, \ldots, p$, $j = 1, \ldots, n$. Set $k = 1$.

5) Create a single offspring for each parent as per CEP Step 3. (Note that only x_d will be mutated due to the zeros in $\boldsymbol{\sigma}$.)
6) Calculate the fitness of each offspring $(\boldsymbol{x}'_i, \boldsymbol{\sigma}'_i)$ as per CEP Step 4.
7) Conduct pairwise comparison as per CEP Step 5.
8) Select the p best individuals for the next generation as per CEP Step 6.
9) Set $k \leftarrow k+1$. If $k = k_{d,\max}$ set $d \leftarrow d+1$ and go to Step 4, otherwise continue from Step 6.

D. Unbiased EP (UEP)

UEP differs from CEP in the way that mutations are performed. Rather than generating Gaussian deviates in axial directions, a set of angles is randomly generated. These are resolved into a direction vector in the co-ordinate system, which is then multiplied by a Gaussian deviate.

1) Generate an initial population of p individuals, where each individual is a triple $(\boldsymbol{x}_i, \boldsymbol{\phi}_i, \sigma_i)$, $i = 1, \ldots, p$. $\boldsymbol{x}_i$ represents an n-dimensional vector as in CEP, randomly initialised in the same way. $\boldsymbol{\phi}$ represents an $(n-1)$-dimensional vector of angles, initialised randomly from a uniform distribution between 0 and 2π, and σ_i represents a standard deviation, initially set to to the value σ_{initial}.
   Set the maximum number of generations, $k_{\max} = E_{\max}/p$, and the current generation $k = 1$.
2) Evaluate the fitness score of each individual as per CEP Step 2.
3) Create a set of p offset vectors, $\boldsymbol{\Delta x}_i$, $i = 1, \ldots, p$, as follows:
   a) For each value of i, set $prevSin = 1$. Then for each value of j, $j = 1, \ldots, n-1$, set:

$$\begin{aligned} \Delta x_i(j) &= \cos(\phi_i(j)) \times prevSin \\ prevSin &\leftarrow \sin(\phi_i(j)) \times prevSin. \end{aligned}$$

   Finally set $\Delta x_i(n) = prevSin$.
   b) Randomly permute the elements of $\boldsymbol{\Delta x}_i$. This is necessary because the above efficient method of resolving angles into Cartesian components generates the co-ordinates in monotonically decreasing order (and hence sets its reference frame accordingly).
4) Each parent $(\boldsymbol{x}_i, \boldsymbol{\phi}_i, \sigma_i)$ creates a single offspring $(\boldsymbol{x}'_i, \boldsymbol{\phi}'_i, \sigma'_i)$ as follows:

$$x'_i(j) = x_i(j) + N_i(0, \sigma_i)\Delta x_i(j) \tag{3}$$

$$\phi_i(j)' = U(0, 2\pi) \tag{4}$$

$$\sigma'_i = \sigma_i e^{(\tau' + \tau)N_i(0,1)} \tag{5}$$

   where $U(a,b)$ denotes uniform random distribution between a and b.
5) Calculate the fitness of each offspring $(\boldsymbol{x}'_i, \boldsymbol{\phi}'_i, \sigma')$ as per CEP Step 4.
6) Conduct pairwise comparison as per CEP Step 5.
7) Select the p best individuals for the next generation as per CEP Step 6.
8) Set $k \leftarrow k+1$. Stop if the halting criterion ($k = k_{\max}$) is satisfied, otherwise go to Step 3.

E. Normalised CEP (CEPN)

CEPN is identical to CEP, except that the initial step size is normalised against dimension. This makes the average step size identical to UEP (which is independent of dimension), ruling out average step size (prior to mutation) as a factor in the comparison. Only the first step changes:

1) Generate 1000 points in n-dimensional space by choosing each co-ordinate from $N(0,1)$. Calculate the mean length μ_n of the vectors from the origin to each of the 1000 points. Next generate 1000 points in 1-dimensional space and calculate the average length μ_1 from the origin to the points. Reset:

$$\sigma_{\text{initial}} \leftarrow \frac{\mu_1}{\mu_n}\sigma_{\text{initial}}.$$

   Then continue as per CEP.

Note that the above can be efficiently approximated using

$$\sigma_{\text{initial}} \leftarrow \frac{1}{\sqrt{n}}\sigma_{\text{initial}}.$$

F. Normalised CEP with Limited Self-adaptation (CEPLS)

Another of the differences between CEP and UEP is that whereas CEP maintains a separate self-adaptation parameter $\sigma_i(j)$ for each parameter $x_i(j)$, UEP mutates a single magnitude, and therefore maintains a single adaptation parameter σ_i, for each individual.

The separate adaptation approach used by CEP may have a disadvantage in high dimensional space. Imagine a worst case scenario where a population is making its way down a valley aligned with one axis, belonging to say the k^{th} parameter. This will have the effect of increasing $\sigma_i(k)$ and reducing (asymptotically towards zero) $\sigma_i(j), j \neq k$, for all population members ($i = 1, \ldots, p$). Imagine the valley now turns in an orthogonal direction k'. Since the $\sigma_i(k')$ required for mutation in the correct direction have tended to zero there is little scope for progress. Meanwhile the $\sigma_i(k)$ are obstructing progress by unsuccessfully directing mutations up the valley walls, leading to a reduction in population diversity. It may take some time for the population to adjust to the new direction. While this is an extreme example, it is possible to envisage many scenarios of this kind.

The CEPLS algorithm addresses this issue by storing and adapting a single mutation parameter for each individual in the same way as UEP, in addition to normalising for step size. The CEPLS algorithm is therefore identical to CEPN except for two changes.

1) In Step 1 generate a population of pairs $(\boldsymbol{x}_i, \sigma_i)$, where $\boldsymbol{x}_i$ and σ_i are as defined in Step 1 of UEP.
3) In Step 3, Equations 1 and 2 are replaced by:

$$\begin{aligned} x_i'(j) &= x_i(j) + N_i(0, \sigma_i) \\ \sigma_i' &= \sigma_i e^{(\tau' + \tau) N_i(0,1)} \end{aligned}$$

resembling Equations 3 and 5 in UEP Step 4.

G. Interleaved IVEP (IVEPI)

Our last algorithm attempts to bridge the gap between CEP and IVEP and can be regarded as a derivative of either. It proceeds by mutating the population in a single dimension in each iteration, cycling through the dimensions.

From the perspective of IVEP we wish to see whether we can improve performance, particularly on non-separable functions such as Schwefel's (see F2 in Section III), by solving parameters in an interleaved fashion rather than one after another. We expect IVEP to perform poorly on F2 since any 1-dimensional slice will be flat over much of the parameter range. Solving in that dimension may therefore result in setting the corresponding parameter anywhere in the flat portion. This may in turn correspond to a poor starting point in subsequent dimensions.

From the perspective of CEP this may help to ameliorate some of the difficulties faced by CEP on multimodal separable landscapes (in particular that the peaks are higher on diagonals then in axial directions) that IVEP does not suffer from. The downside is that it will not have access to the synergies that can be obtained in some situations by modifying multiple parameters at once.

The algorithm is identical to CEP with the following exceptions:

1) Set $d = 0$. Proceed as per CEP Step 1.
3) Set $d \leftarrow (d \mod n) + 1$.
Each parent $(\boldsymbol{x}_i, \boldsymbol{\sigma}_i)$, $i = 1, \ldots, p$, creates a single offspring $(\boldsymbol{x}_i', \boldsymbol{\sigma}_i')$ as follows:

$$\begin{aligned} x_i'(d) &= x_i(d) + N_i(0, \sigma_i(d)) \\ \sigma_i'(d) &= \sigma_i(d) e^{\tau' N_i(0,1) + \tau N_i'(0,1)} \end{aligned}$$

That is, both $\boldsymbol{x}_i$ and $\boldsymbol{\sigma}_i$ are mutated only in the d^{th} dimension.

III. Experiments

For comparison we tested the algorithms on the seven functions specified in [8]. The first six are elementary mathematical functions:

$$\begin{aligned} F_1(\boldsymbol{x}) &= \sum_{i=1}^{n} z_i^2 + b_1 \\ F_2(\boldsymbol{x}) &= \max_{i=1}^{n} |z_i| + b_2 \\ F_3(\boldsymbol{x}) &= \sum_{i=1}^{n-1} (100(z^2 - z_{i+1})^2 + (z_i - 1)^2) + b_3 \\ F_4(\boldsymbol{x}) &= \sum_{i=1}^{n} (z_i^2 - 10\cos(2\pi z_i) + 10) + b_4 \end{aligned}$$

$$\begin{aligned} F_5(\boldsymbol{x}) &= \sum_{i=1}^{n} \frac{z_i^2}{4000} - \prod_{i=1}^{n} \cos(\frac{z_i}{\sqrt{i}}) + 1 + b_5 \\ F_6(\boldsymbol{x}) &= -20\exp(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} z_i^2}) \\ &\quad - exp(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi z_i)) + 20 + e + b_6 \end{aligned}$$

where the $\boldsymbol{z} = \boldsymbol{x} - \boldsymbol{o}$, for some offset $\boldsymbol{o}$, are "shifted" objective variables, and $b_1, \ldots, b_6$ are fixed biases. These are respectively known as *Sphere*, *Schwefel's Problem 2.21*, *Rosenbrock's function*, *Rastrigin's function*, *Griewank's function* and *Ackley's function*. The seventh is a *fast fractal* function, *"DoubleDip"*, from the Fractal Function benchmarking suite [1], [9]. Functions 1, 4 and 6 are separable. Function 7 is close to separable in that each parameter is dependent only on one of the $n - 1$ other parameters (different for each). For plots of the functions see [8].

Following [8], all functions have been evaluated at $50n$, $500n$ and $5000n$ function evaluations, averaged over 25 runs. While [8] requires evaluation at 100, 500 and 1000 dimensions, so that we can better see the trends in behaviour we have evaluated all functions at 1, 2, 5, 10, 20, 50, 100, 200, 500 and 1000 dimensions. Convergence graphs have also been obtained for 1000 dimensions averaged over 25 runs.

In line with [7] all experiments set the tournament size $q = 10$, population size $p = 100$, and initialise the standard deviation $\sigma_{\text{initial}} = 3.0$.

IV. Results and Discussion

Figures 1, 2 and 3 show the overall performance of each algorithm on the test functions.

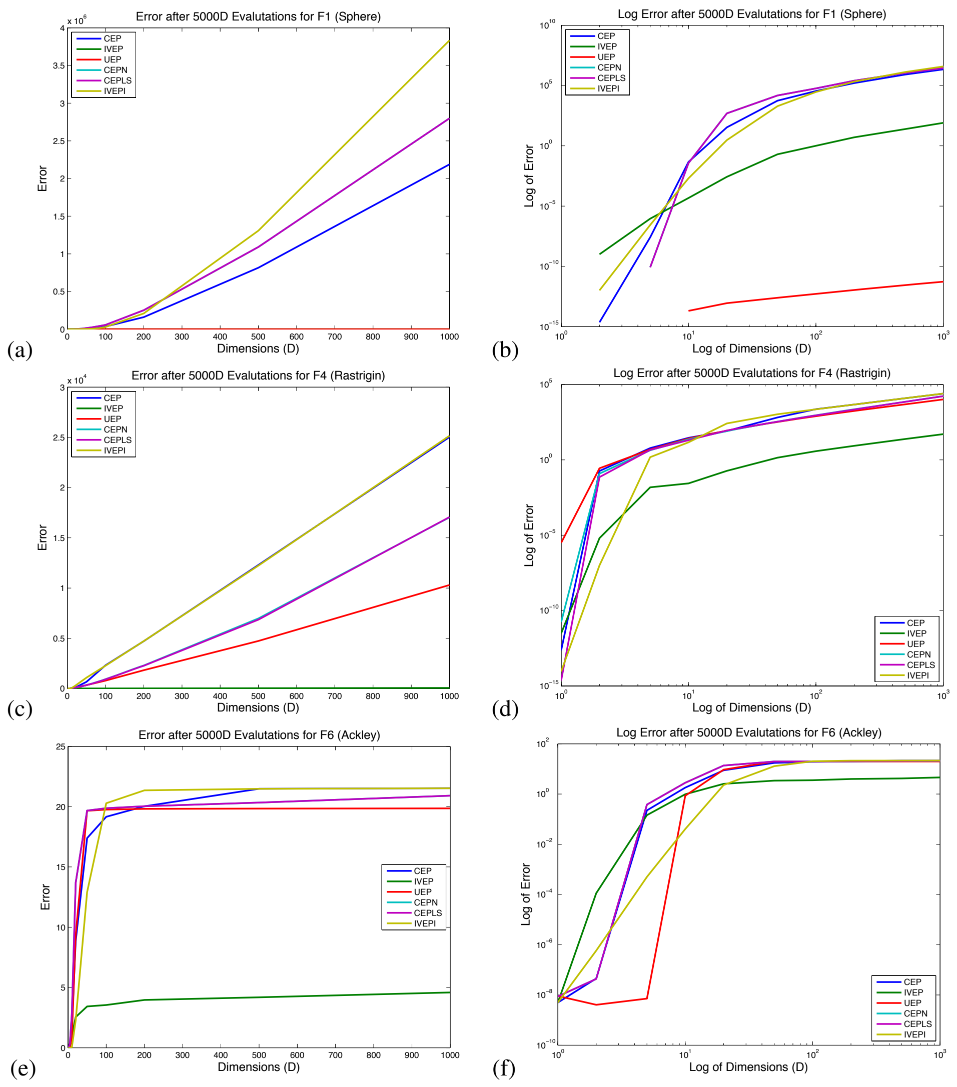


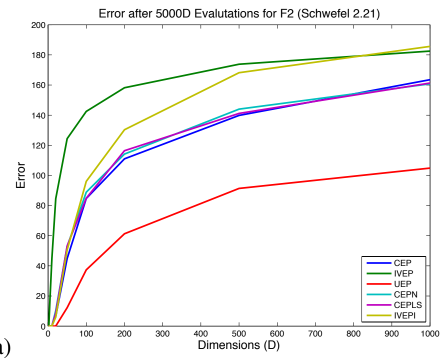
Fig. 1. Smallest error achieved versus dimension (measured at 1, 2, 5, 10, 20, 50, 100, 200, 500 and 1000 dimensions) for separable functions 1, 4 and 6, averaged over 25 runs. Where CEPN is occluded it lies beneath CEPLS. In (a) IVEP lies beneath UEP. The truncated points in (b) can be effectively regarded as zero to 64-bit floating point resolution.

It can be seen from the figures that Unbiased Evolutionary Programming outperforms CEP, normalised CEP, normalised CEP with limited self-adaptation and interlaced IVEP across all functions. UEP also outperforms IVEP on all non-separable functions, with the exception of the near-separable Function 7, and on the separable function F1. In some cases the difference in performance is striking — for functions 1, 3 and 5, for example, the difference is many orders of magnitude.

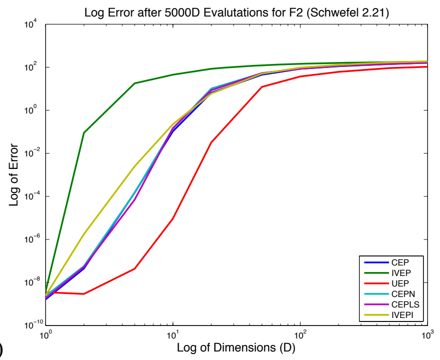
A second general point to notice is that performance at low dimensions is a poor indicator of high-dimensional performance. Most functions take at least 100 dimensions before what appear to be longer term trends establish themselves, with some such as F2 and F3 taking significantly longer. This suggests that some algorithms rely on the constraints implicit in low-dimensional problems, and the conclusions of many works in the literature on global optimisation will not generalise to higher dimensions.

A. Observations on Separable Problems

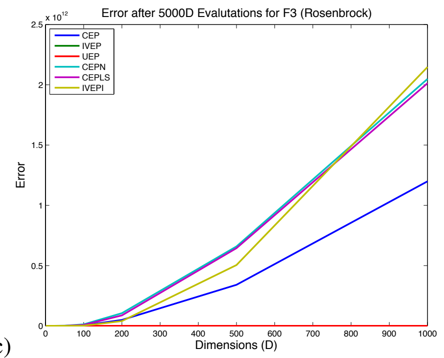
It is arguable whether separable problems should appear in benchmarking problem sets for selecting algorithms designed for real-world applications. First, it is unlikely that many significant real-world problems have independent parameters. Secondly, since the parameters are independent, we can simply solve each of them independently, transforming an n-



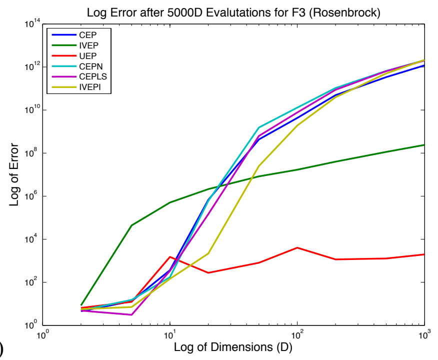
(a)



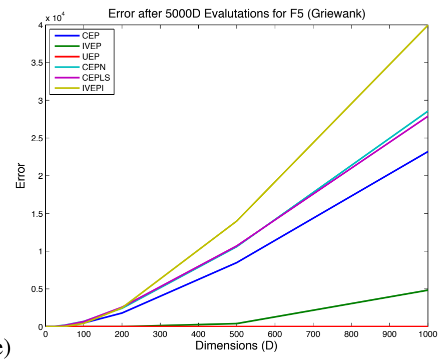
(b)



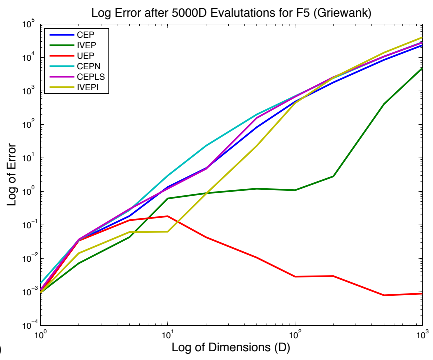
(c)



(d)



(e)



(f)

Fig. 2. Smallest error achieved versus dimension (measured at 1, 2, 5, 10, 20, 50, 100, 200, 500 and 1000 dimensions) for non-separable functions 2, 3 and 5, averaged over 25 runs. In (c) IVEP lies beneath UEP.

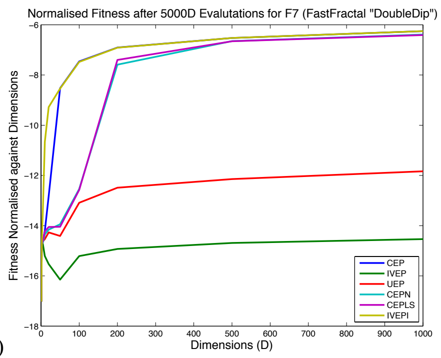
dimensional problem to n 1-dimensional problems.

From a theoretical point of view, however, they provide for some interesting comparisons. In particular, it is interesting to see whether generic solvers which do not take advantage of the separability can outperform those that do, and if so, for what kinds of problems.

Figure 1 (a) and (b) demonstrates that a generic solver can indeed outperform the algorithms that solve each of the dimensions independently. Although traditional CEP and its derivatives cannot match the performance of IVEP for greater than 5 dimensions, UEP outperforms IVEP by approximately 9 to 12 orders of magnitude from 10 to 1000 dimensions.

Looking at the more difficult multimodal problems in Figures 1 (c) to (f), however, we see that for dimensions above about 10 even UEP is not able to outperform IVEP. This is as we anticipated — the increase in complexity with dimension for the more general algorithms outweighs the reduction in the number of dimensions available to IVEP.

The same is true of the "near-separable" multimodal function F7 in Figure 3. For this function it is impossible for IVEP to find the minimum, since 1-dimensional "slices" in any axial direction will have distinct minima depending on the value of another parameter. There is no order in which the parameters can be solved that will lead to a global minimum. Nevertheless IVEP outperforms the more general algorithms. The error caused by small dependencies between the variables is outweighed by the high-dimensional complexity that the more general algorithms must deal with.

(a)


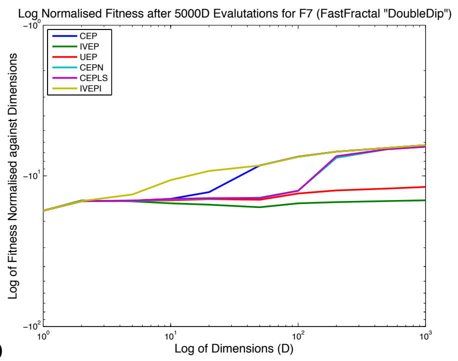
(b)


Fig. 3. Smallest error achieved versus dimension (measured at 1, 2, 5, 10, 20, 50, 100, 200, 500 and 1000 dimensions) for function 7, averaged over 25 runs. Since the depth of the minimum for this function increases with dimension we have normalised against dimension. Results shown are for the parameter range [-1 1] specified in [8]. Results for the intended range of the function, [0 1], are very similar.

There are at least two effects that we can conjecture come into play. First, in order to outperform IVEP an algorithm must be able to find synergies from improving multiple parameters at once. One way to look at this is that it must be able to find "shortcuts" across the terrain compared with algorithms that solve the parameters independently. For the "sphere" problem (F1) this is relatively easy — it is just as easy to descend towards the minimum down a diagonal as it is along an axis. Heading down the diagonal however has the potential to solve multiple parameters at once compared with IVEP which solves only a single parameter at a time.

For multimodal functions, however, the picture changes. For separable problems the peaks that must be traversed between one minimum and the next will tend to be higher on a diagonal than in an axial direction. This is certainly the case for all linear separable problems, where the fitness can be expressed as the sum of functions of the individual parameters. A clear example of this is Rastrigin's function (F4). In this case an unbiased algorithm will waste many evaluations sampling regions that are subsequently selected out of the population, compared with an algorithm such as IVEP which will only sample along the "path of least resistance".

The second but related issue is the way that probability density functions change with increasing dimensions. As n increases the number of evaluations that IVEP has available for each 1-dimensional search decreases with $\frac{1}{n}$. The hypervolume over which a general algorithm must search however increases exponentially. Intuitively therefore therefore we can conjecture that the general algorithm search becomes sparse at a faster rate, and the ability to adequately sample a specific region (such as along the axes) decreases with increasing dimension. More work is needed to verify this effect.

B. *Observations on Non-separable Problems*

For the non-separable problems in Figure 2 the algorithm without directional bias, UEP, outperforms all others. This is particularly evident in functions 3 and 5 where UEP outperforms the other algorithms by up to 10 orders of magnitude at 1000 dimensions.

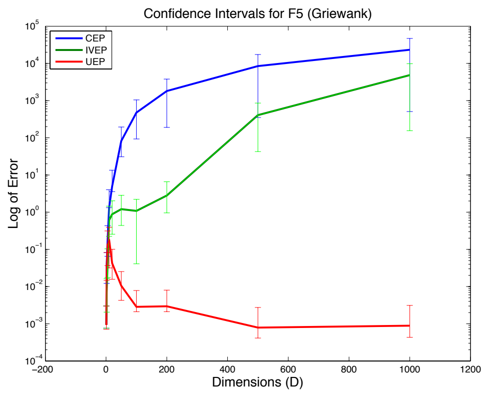


Fig. 4. Semilogarithmic graph for F5 showing 95% confidence intervals.

The most remarkable result, however, is Function F5 in Figure 2 (f). For this function, after approximately 10 dimensions, performance per number of evaluations (which, recall, increases linearly with dimension) actually improves with increasing dimensions. This is the only algorithm that was able to exhibit a decreasing trend on any of the problems.

To confirm this result we took additional measurements (these were carried out for all functions but there is not room to include them all here). Figure 4 shows confidence intervals for F5 for the three primary functions at the 95% level of confidence. It shows a significant difference between UEP and the other algorithms. Figure 5 shows convergence plots for F5 for 1000 dimensions. The plots indicate that all algorithms are steadily improving over the 5 million evaluations. Apart from IVEP which has the expected linear improvement as it cycles through the variables, the algorithms approximate decaying exponentials (linear in the semilogarithmic graph) as expected. UEP however reduces error by a much larger proportion of the remaining error on each iteration.

This stark contrast in performance was the main motivation for the intermediate algorithms CEPN and CEPLS, which

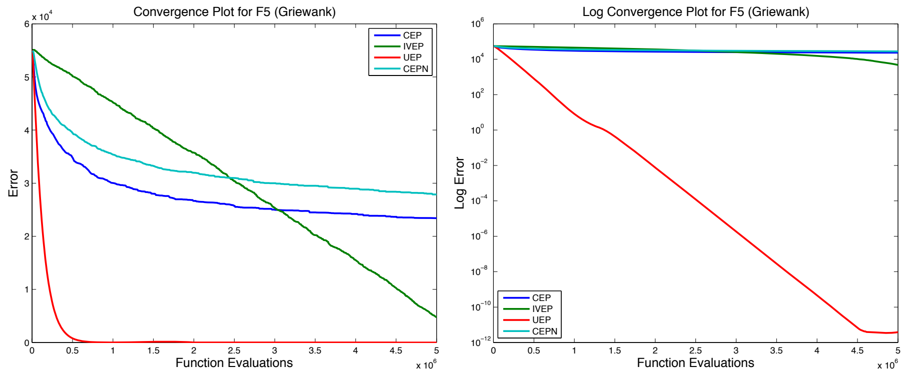


Fig. 5. Convergence plots for Function F5 for 1000 dimensions.

match CEP more closely to UEP by adjusting for initial step size and maintaining step size for the whole population, thus preventing progress in preferred directions diminishing step size in others. Figures 1 to 3 show that adjusting for these factors has little impact on the performance of CEP, in some cases making the performance worse. To date, therefore, we have not been able demonstrate an alternative explanation for the exceptional improvement in performance of UEP relative to CEP other than the choice of direction.

V. Conclusion

Many global optimisation algorithms are developed with implicit biases, often brought about by the expedience of specifying them in the Cartesian co-ordinate system which we are most used to, without an extensive study of the implications of those choices [10]. This paper demonstrates that these implications are often masked at low dimensions, and increase significantly as we extend the domain of application to higher dimensions. This is shown vividly in results such as Figure 2 (f), where our unbiased algorithm outperforms the classical algorithm by many orders of magnitude.

Our results have been generated by developing an unbiased evolutionary programming algorithm (in terms of mutation direction) that is based on a classical EP algorithm. We have attempted to control for other factors that may play a part in the improved performance by examining intermediate algorithms that adopt several aspects of UEP. While these have failed to match UEP for performance, it should be cautioned that these are not exhaustive, and further detailed studies are needed to examine the relative behaviour of UEP in high-dimensional spaces that lead to such a striking improvement in performance.

Finally, it is worth noting that we are not claiming these algorithms to be the *best* for high-dimensional optimisation, and more recent alternatives such as [11] deserve further comparision. CEP is deliberately chosen as a simple standard in an attempt to isolate the differences brought about by changes to the mutation process. It is anticipated that the lessons from this study will be applicable in other high-dimensional optimisation algorithms.

Acknowledgment

The experimentation for this work was carried out while the first author was on study leave at the Centre of Excellence for Research in Computational Intelligence and Applications (CERCIA) at the University of Birmingham, who's support she gratefully acknowledges.

References

[1] C. MacNish, "Towards unbiased benchmarking of evolutionary and hybrid algorithms for real-valued optimisation," *Connection Science*, vol. 19, no. 4, pp. 361–385, 2007.

[2] R. Salomon, "Re-evaluating genetic algorithm performance under coordinate rotation of benchmark functions," *BioSystems*, vol. 39, no. 3, pp. 263–278, 1996.

[3] A. Czarn, C. MacNish, K. Vijayan, and B. Turlach, "Statistical exploratory analysis of genetic algorithms: The detrimentality of crossover," in *Twentieth Australian Joint Conference on Artificial Intelligence (AI07)*, ser. Lecture Notes in Artificial Intelligence. Springer, 2007, to appear.

[4] M. Schoenauer, "Bio-inspired continuous optimization: The coming of age," in *IEEE Congress on Evolutionary Computation (CEC'07)*. Invited Lecture, IEEE Press, 2007.

[5] D. B. Fogel, "An introduction to simulated evolutionary optimization," *IEEE Trans. Neural Networks*, vol. 5, pp. 3–14, 1994.

[6] T. Back and H. P. Schwefel, "An overview of evolutionary algorithms for parameter optimization," *Evol. Comput.*, vol. 1, no. 1, pp. 1–23, 1993.

[7] X. Yao, Y. Liu, and G. Lin, "Evolutionary programming made faster," *IEEE Transactions on Evolutionary Computation*, vol. 3, no. 2, pp. 82–102, 1999.

[8] K. Tang, X. Yao, P. N. Suganthan, C. MacNish, Y. P. Chen, C. M. Chen, and Z. Yang, "Benchmark functions for the CEC'2008 special session and competition on large scale global optimization," Nature Inspired Computation and Applications Laboratory, USTC, China, Tech. Rep., 2007. [Online]. Available: http://nical.ustc.edu.cn/cec08ss.php

[9] B. Sendhoff, M. Roberts, and X. Yao, "Evolutionary computation benchmarking repository," *IEEE Computational Intelligence Magazine*, vol. 1, no. 4, pp. 50–51, November 2006. [Online]. Available: http://www.cs.bham.ac.uk/research/projects/ecb/

[10] T. Schnier and X. Yao, "Using multiple representations in evolutionary algorithms," in *Proc. 2000 Congress on Evolutionary Computation*. IEEE Press, 2000, pp. 479–486.

[11] Z. Yang, K. Tang, and X. Yao, "Differential evolution for high-dimensional function optimization," in *Proc. of the IEEE Congress on Evolutionary Computation (CEC'07)*. IEEE Press, 2007, pp. 3523–3530.