Precision, Local Search and Unimodal Functions

Martin Dietzfelbinger
Fakultät für Informatik und Automatisierung, Technische Universität Ilmenau
98684 Ilmenau, Germany
martin.dietzfelbinger@tu-ilmenau.de

Jonathan E. Rowe
School of Computer Science, University of Birmingham
Birmingham B15 2TT
United Kingdom
J.E.Rowe@cs.bham.ac.uk

Ingo Wegener
FB Informatik, Technische Universität Dortmund
44221 Dortmund, Germany
ingo.wegener@uni-dortmund.de

Philipp Woelfel
Department of Computer Science, University of Calgary
Calgary, Alberta T2N 1N4, Canada
woelfel@cpsc.ucalgary.ca

ABSTRACT

We investigate the effects of precision on the efficiency of various local search algorithms on 1-D unimodal functions. We present a $(1+1)$-EA with adaptive step size which finds the optimum in $O(\log n)$ steps, where n is the number of points used. We then consider binary and Gray representations with single bit mutations. The standard binary method does not guarantee locating the optimum, whereas using Gray code does so in $O((\log n)^2)$ steps. A $(1+1)$-EA with a fixed mutation probability distribution is then presented which also runs in $O((\log n)^2)$. Moreover, a recent result shows that this is optimal (up to some constant scaling factor), in that there exist unimodal functions for which a lower bound of $\Omega((\log n)^2)$ holds regardless of the choice of mutation distribution. Finally, we show that it is not possible for a black box algorithms to efficiently optimise unimodal functions for two or more dimensions (in terms of the precision used).

Categories and Subject Descriptors

F.2.1 [**Analysis of algorithms and problem complexity**]: Numerical Algorithms and Problems; G.1.6 [**Numerical analysis**]: Optimization

General Terms

Theory

Keywords

Local search, precision, computational complexity, unimodal functions

GECCO'08, July 12–16, 2008, Atlanta, Georgia, USA.

1. INTRODUCTION

Consider the problem of minimising a unimodal function $f : [a, b] \to \mathbb{R}$ defined on some interval $[a, b]$. We wish to study the effects of changing the *precision* with which we are working. On the one hand, increased precision will enable us to get closer to the actual optimum value. On the other hand, it increases the number of points that have to be considered during the search process. To study this, we imagine that we place n equally spaced points in the interval concerned, and label them $0, 1, \ldots, n-1$. We denote this set of points $[n]$. We say that such a point is an *optimum* with respect to f if it has a lower f-value than its neighbours, and we will be assuming that neighbouring points never have exactly the same f-value. A function is *unimodal* on $[n]$ if there is exactly one optimum point.

Algorithms for solving this problem efficiently (that is, in $O(\log n)$ time) are known and indeed date back to the 1950s. These algorithms, such as Fibonacci search [5], are based on the idea of using a binary search process to cut away large parts of the search space. In the following section we present a $(1+1)-EA$ that follows a somewhat similar approach, in that the mutation step size halves at each generation. We then consider the standard method of encoding the points as binary strings (using either the base-2 or Gray representations), and study local search based on flipping individual bits. It is well-known that using base-2 is not desirable since it can introduce new local optima under this search operator. However, local search with Gray code is guaranteed to find the optimum, and we derive upper and lower bounds on the time for doing so.

Ideally, we would like to create an algorithm which will work well on multimodal problems. However, the above techniques will not generally work well on such problems, as they will become trapped in a local optimum. There are several strategies for dealing with this situation. For example, one may employ random restarts, or use a tabu-list. Here we consider local search algorithms in which the mutation step size is drawn from a fixed probability distribution. Such algorithms at least have a chance to escape from local optima (especially if such optima are clustered together).

The question then remains as to how much this affects the algortihm's ability to locate the optimum when the function is actually unimodal. We present a $(1+1)-EA$ based on this idea, using a *scale free* distribution, and show that it optimises unimodal functions in $O((\log n)^2)$ steps. Recent research has shown that this is optimal (up to a constant factor) in that there exist unimodal functions for which the running time of such a local search algorithm is bounded below by $\Omega((\log n)^2)$ regardless of the choice of probability distribution.

We finish by discussing the problem of optimising unimodal functions in two (or more) dimensions, and conclude that black box algorithms are not efficient in this case.

2. BINARY SEARCH ALGORITHMS

Efficient algorithms for optimising unimodal functions are based on a form of binary search. The basic idea is to start by evaluating the end points 0 and $n-1$, and some point x_1 in between. We then evaluate a fourth point $x_2 > x_1$, and depending on whether its f-value is higher or lower than that of x_1 we can restrict the next step of the search to the interval $[0, x_2]$ or $[x_1, n-1]$ respectively. This process continues iteratively until the optimum is found in $O(\log n)$ steps. To maximise the number of points that can be excluded from the search at each step, it is necessary to place the new point in a way that balances the intervals created — it can be shown that using interval sizes in the ratio of the Fibonacci sequence is the best way to do this [5]. The same principle is utilised in the Quad Search algorithm [10], although a binary Gray code representation is used in the implementation (see also the following section).

Here we present an algorithm that is based on a somewhat similar idea, but takes the form of a (1+1)-EA, in which the mutation step size is halved at each iteration. The *halving algorithm* is as follows:

1: Let $m = 2^{\lceil \log_2 n \rceil}$
2: Let $k = 1$
3: Let x be chosen randomly from $[n]$
4: Let $y = x + m/2^k$
5: Let $z = x - m/2^k$
6: Let x be the best point from $\{x, y, z\}$.
7: Let $k = k + 1$
8: If $k > \lceil \log_2 n \rceil$ then stop, else go to 4.

In step 6, "best" means both legal and with the lowest f-value. It is clear that this algorithm iterates for $\lceil \log_2 n \rceil$ time steps and evaluates at most $2\lceil \log_2 n \rceil + 1$ points. We now show that it will terminate at an optimum point (even for a multimodal function) and so, in particular, will optimise a unimodal function.

THEOREM 1. *The halving algorithm terminates on an optimum point.*

PROOF. Let $T = \lceil \log_2 n \rceil$. Suppose the algorithm terminates at a point x_T and that this is not an optimum point. Without loss of generality, assume the point $w = x_T + 1$ has $f(w) < f(x_T)$. Then on the previous time step, the algorithm must have been in state $x_{T-1} = x_T - 1$, since the move was of step size 1, and it is impossible that it came from point w, or that it was a rejected move. Similarly, the time step before that must have been from state $x_{T-2} = x_{T-1} - 2 = x_T - 3$. In general, the state at time step $T - i$ must have been $x_{T-i} = x_{T-i+1} - 2^{i-1} = x_T - (2^i - 1)$. In particular, the initial point chosen must have been $x_0 = x_T - (2^T - 1)$. This means that $w - x_0 = x_T + 1 - x_T + 2^T - 1 = 2^T$. However, this is impossible since the total number of points is $n \leq 2^T$. □

COROLLARY 1. *The halving algorithm optimises a unimodal function in $O(\log n)$ steps.*

An advantage of this algorithm over the Fibonacci algorithm is that on multimodal functions, different optima may be found by performing restarts. It is also worth noting that one can implement a continuous version of this algorithm by taking m (in step 1) to be the size of the interval on which the function is defined.

3. BINARY AND GRAY CODE

One approach to constructing evolutionary algorithms for numerical optimisation is to represent numbers as binary strings, using either the standard base-2 representation, or a reflective Gray code. If we assume that $n = 2^L$, then we use bitstrings of length L to represent the points in the search space. The local search operator corresponds to randomly flipping bits one at a time.

If we have a unimodal function in which the optimum is at 0 or $n-1$, then using the base-2 encoding is quite efficient. The target is the all zeros string (or all ones string) and once a bit is set correctly, we do not accept moves which change it. Consequently, the problem is equivalent to the one-max problem and the running time for such a local search algorithm is $O(L \log L)$. However, these target strings are rather special in relation to the base-2 encoding. If we had some other optimum point, then local search (using this encoding) is not guaranteed to find the global optimum, as there may be several local optima (with respect to the Hamming neighbourhood). See [9] for details. Using a reflective Gray code, however, cannot possibly introduce new optima (since neighbouring points are guaranteed to remain neighbours in Gray code) and so, in particular, unimodal functions remain unimodal. Local search with Gray code is therefore guaranteed to optimise such functions. It has previously been proven that a *steepest descent* type of algorithm will locate the optimum in $O(L^2)$ function evaluations [7], and empirical data suggested that a *next descent* type of algorithm may perform significantly better. Here we show an upper bound of $O(L^2)$ for both steepest and next descent and a lower bound of $\Omega(L^2/\log L)$, although we conjecture, in contrast to [7], that $\Omega(L^2)$ steps are necessary.

THEOREM 2. *Next Descent and Steepest Descent algorithms using Gray code with L bits and with single bit flips require $O(L^2)$ function evaluations to optimise unimodal functions.*

Notation: The bits of the Gray code are $b_L \dots b_1$. The bijection between numbers $x \in [n]$ and Gray codes in $\{0,1\}^L$ can be described as follows: If x has binary representation $c_L \dots c_0$ (with $L+1$ bits, and hence $c_L = 0$), then $b_i = c_i \oplus c_{i-1}$. By $x^{(i)}$ we denote the integer obtained from x by flipping bit number i in the Gray code representation of x.

We can visualize the correspondence between numbers and Gray codes as a binary tree, like in Figure 1 for $L = 4$. This figure also includes a sketch of a unimodal function. The goal is to find the point m with $f(m) = \min\{f(x) \mid x \in [n]\}$.

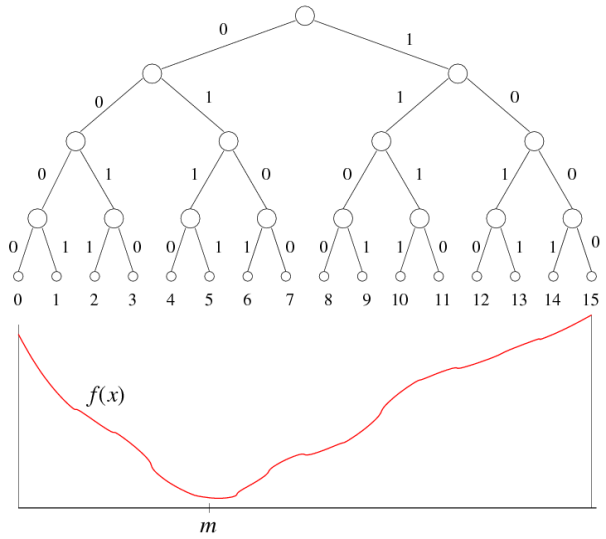

Figure 1: Gray codes as a binary tree

The *next descent* search strategy can be described as follows: Pick a starting point x_0 uniformly at random from $[n]$. Then iterate the following step, for $t = 1, 2, \ldots$:

Choose $i \in \{1, \ldots, L\}$ uniformly at random. Let $x' = x_{t-1}^{(i)}$. If $f(x') < f(x_{t-1})$, let $x_t = x'$, otherwise let $x_t = x_{t-1}$.

We are interested in the expectation of

$$T = \min\{t \mid x_t = m\}.$$

Our aim is to show that $\mathbf{E}(T) = O(L^2)$.

We associate an integer value $\Phi(x)$ with the situation where the process sits at x, as follows. Let $\Phi(m) = 0$. The set $A_x = \{y \mid f(y) \le f(x)\}$ forms an interval in $[n]$, which contains m. If $|A_x| = 1$, we are done; if $|A_x| = 2$ then m and x are immediate neighbors (as numbers), in which case the probability that the correct bit is chosen is $1/L$. In this case, we let $\Phi(x) = 2$. From here, we assume that $|A_x| \ge 3$. We look for the smallest pair of neighboring subtrees of the same height which include all of A_x. The term "neighboring" here does not refer to the tree structure but rather to the ordering of the natural numbers. For example, in Figure 1 the subtrees with leaf sets $\{4, \ldots, 7\}$ and $\{8, \ldots, 11\}$ are neighbors (height 2), and the subtrees with leaf sets $\{8, 9\}$ and $\{10, 11\}$ are neighbors (height 1). Each of the two trees has two subtrees in the standard sense. In this way, we obtain four subtrees of equal height h_x. (See Figure 2 for an example.) These subtrees are numbered $1, 2, 3, 4$, from left to right.

Case 1: A_x touches (the leaf sets of) all 4 subtrees. Then $\Phi(x) = 2h_x$.

Case 2: A_x touches (the leaf sets of) only 3 subtrees. Then $\Phi(x) = 2h_x - 1$.

For example, in Figure 2 Case 2 applies; the set A_x touches trees $1, 2, 3$. There are no other cases, since if only 2 trees out of $1, 2, 3, 4$ are touched by A_x, the trees are not minimal.

CLAIM 1. *Let $|A_x| \ge 2$. If $x_{t-1} = x$, then there is at least one i such that $\Phi(x_{t-1}^{(i)}) < \Phi(x_{t-1})$. (Hence* $\text{Prob}(\Phi(x_t) < \Phi(x_{t-1}) \mid x_{t-1} = x) \ge \frac{1}{L}$.*)*

Since $\Phi(x_0) \le 2L$, and Φ takes only integer values, $2L$ successful steps that decrease Φ are sufficient. It is then clear that the expected waiting time until Φ has become 0 is bounded by $2L/(1/L) = 2L^2$.

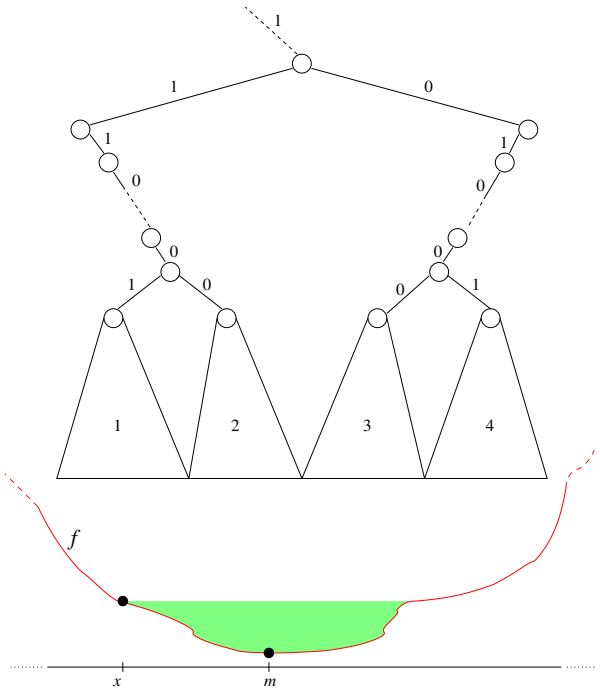

Figure 2: Four neighboring subtrees covering A_x

PROOF. The claim is proved by considering several cases. For symmetry reasons, we may assume that x is in tree 1 or tree 2. Note that then it is impossible that $m < x$, since then f would be increasing to the right of x, and hence A_x could not touch trees 3 or 4.

Case 1: x is in tree 1.

Consider the bit i that flips x to its "partner" x' in tree 2.

Case 1a: m is in tree 1, $x < m$. — Then f is increasing to the right of x', hence $f(x') < f(y)$ for all y in trees 3 and 4. Hence $A_{x'}$ is contained in the union of the trees 1 and 2. On the other hand, A_x touched tree 3 (at least). Hence $f(x') < f(x)$ and $\Phi(x') < \Phi(x)$.

Case 1b: m is in tree 2. — If $x' = m$, we have $\Phi(x') = 0 < \Phi(x)$. If $x' < m$, then f is decreasing to the left of x', and hence $A_{x'}$ does not touch tree 1, but A_x does. Hence $f(x') < f(x)$ and $\Phi(x') < \Phi(x)$. If $x' > m$, then f is increasing to the right of x', hence $A_{x'}$ is contained in the union of trees 1 and 2, and again $f(x') < f(x)$ and $\Phi(x') < \Phi(x)$.

Case 1c: m is in tree 3 or 4. — Since f is decreasing to the left of x', the set $A_{x'}$ does not touch tree 1, but A_x does. Hence $f(x') < f(x)$ and $\Phi(x') < \Phi(x)$.

Case 2: x is in tree 2.

Since $x < m$, the function f is decreasing to the left of x, and hence A_x does not touch tree 1. Hence A_x must touch trees 2, 3, and 4. Consider the bit i that flips x to its "partner" x'' in tree 3.

Case 2a: m is in tree 2, $x < m$. — Then f is increasing to the right of x'', hence $A_{x''}$ does not touch tree 4, hence $f(x'') < f(x)$, and $\Phi(x'') < \Phi(x)$.

Case 2b: m is in tree 3. — If $x'' = m$, we have $\Phi(x'') =$

$0 < \Phi(x)$. If $x'' < m$, then $A_{x''}$ does not touch tree 2. If $x'' > m$, then $A_{x''}$ does not touch tree 4. In all cases we have $f(x'') < f(x)$ and $\Phi(x'') < \Phi(x)$.

Case 2c: m is in tree 4. — Then $A_{x''}$ does not touch tree 2, hence $f(x'') < f(x)$ and $\Phi(x'') < \Phi(x)$.

The claim is proved. □

REMARK 1. The *steepest descent* algorithm [7] uses L function evaluations to determine the i that minimizes $f(x^{(i)})$. One such move will shrink the set A_x at least as much as the single good move identified in the "Claim". This yields an alternative proof for the fact that *steepest descent* needs at most $2L^2$ function evaluations.

An alternative strategy is the Random Bit Climber (RBC) algorithm [7], which tries bits in a randomized order. Once all bits have been tried, a new random order is generated. This strategy, therefore, runs in phases of L rounds each. Given $x = x_{(s-1)L}$, we check bit positions according to a random permutation $(i_1, \ldots, i_L)$, and accept whenever an improvement for f is found. The resulting sequence of new points is $x_{(s-1)L+1}, \ldots, x_{sL}$. After phase s, we choose a new random permutation. The idea is to try to avoid repeatedly picking previously tried, but unsuccessful, bits.

Consider the four trees 1, 2, 3, 4 that belong to x, as in the proof of the claim. We ignore bit positions that do not flip x_{t-1} to one of its "partners" in trees 1, 2, 3, or 4. Bit flips that lead to points outside these four trees are never accepted, bit flips that lead to other points in the same subtree may or may not be accepted.

If x is in tree 1, there are two cases:

Case 1: The bit flip that leads the sequence $x_{(s-1)L+1}, \ldots, x_{sL}$ into tree 4 is tried and accepted before the move into tree 2. Then the function Φ must decrease before or at the bit flip that moves the point from tree 4 into tree 3 (by the proof of the "Claim", applied to tree 4 by symmetry).

Case 2: The bit flip that leads the sequence $x_{(s-1)L+1}, \ldots, x_{sL}$ into tree 2 is carried out first. Then the function Φ must decrease before or at this step (by the proof of the claim).

Now consider the case that x is in tree 2. Since $x < m$, A_x does not touch tree 1, and no bit flip that leads into tree 1 will be accepted. Hence in $x_{(s-1)L+1}, \ldots, x_{sL}$ we will see the bit flip that leads into tree 3 first, and Φ must decrease before or at this bit flip. (We do not care whether or not the sequence moves into tree 4 afterwards.)

In any case, after one phase of testing the L bits the function Φ must have decreased by at least 1. Hence there can be at most $2L$ phases.

REMARK 2. This argument shows that for the upper bound to hold it is not necessary to change the order of the bit flips in each phase.

It is worth noting that the RBC algorithm offers only a small improvement (bounded by a constant) to the standard $(1+1)$-ES (in which bits are chosen uniformly at random). This is because if k of the L bits are improving moves then the $(1+1)$-ES finds one of them, on average, in L/k time steps, whereas the RBC takes $(L+1)/(k+1)$ steps, as the following lemma shows.

LEMMA 1. *If k distinct integers are picked randomly from the set $\{1, 2, \ldots, L\}$ then the expected value of the smallest integer chosen is $(L+1)/(k+1)$.*

PROOF. Since we are dealing with positive integers, then we can use the formula $\mathbf{E}[X] = \sum_i \text{Prob}(X \geq i)$. So the expected least integer is

$$\sum_{i=1}^{L-k+1} \frac{\binom{L-i+1}{k}}{\binom{L}{k}} = \sum_{m=k}^{L} \frac{\binom{m}{k}}{\binom{L}{k}} = \frac{\binom{L+1}{k+1}}{\binom{L}{k}} = \frac{L+1}{k+1} \qquad \square$$

The improvement is most when $k = 1$ in which case the RBC takes half the time to find the right bit to flip. As k increases, the improvement becomes less and less significant.

THEOREM 3. *If the "next descent" or the "RBC" algorithm is applied to the identity function $[2^L] \ni x \mapsto x \in [2^L]$, then $\mathbf{E}(T) = \Omega(L^2/\log L)$.*

PROOF (SKETCH). We look at the "next descent" algorithm and argue in terms of Gray codes as in the proof of the upper bound. The target point is $(0, \ldots, 0)$. To measure how far the process still has to go we take the position

$$h(x) = h((b_L, \ldots, b_1)) = \max\{i \mid b_i = 1\}$$

of the highest-valued 1-bit in the Gray code $x = (b_L, \ldots, b_1)$ as indicator.

We define one "phase" of the process as a maximal sequence of rounds in which $h(x_t) = \cdots = h(x_{t+s-1}) > h(x_{t+s})$. The expected number of rounds in a phase is exactly L, since the highest-valued 1-bit in the Gray code x_t must be hit, and the probability for this to happen is $1/L$. We claim that the average decrease of h in a phase is not larger than $\ln L$.

Assume $h(x) = i$, $b_i = 1$ and $b_{i-1} = \cdots = b_{i-k+1} = 0$ and $i - k = 0 \vee b_{i-k+1} = 1$. Let Z be the decrease of h in this phase. This means that $b_{i-1} = \cdots = b_{i-Z+1} = 0$ and $i - Z = 0 \vee b_{i-Z+1} = 1$ when bit i is chosen. Note that $\text{Prob}(Z \geq j) = \frac{1}{j}$ for $1 \leq j \leq k$ (the 1 in position i is the first to be chosen out of the positions $\{i, i-1, \ldots, i-j+1\}$). Hence

$$\mathbf{E}(Z) = \sum_{1 \leq j \leq k} \text{Prob}(Z \geq j) = \sum_{1 \leq j \leq k} \frac{1}{j} = H_k < 1 + \ln L.$$

Since the progress in each phase is at most $1 + \ln L$, and the average position of the leftmost 1 in the Gray code at the beginning is about $L-1$, the expected number of phases to reach the target 0 is at least $(L-1)/(1+\ln L)$. This can be seen by applying [4, Lemma 12] (a one-sided variant of Wald's identity).

The argument for RBC is similar; the expected length of a phase is $\frac{1}{2}(L+1)$, though. □

REMARK 3. We conjecture (but could not prove yet) that $\mathbf{E}(T) = \Omega(L^2)$ for the identity function. This is supported by the fact that for $L = 3, \ldots, 11$ the quantity $\mathbf{E}(T)$ (which depends on L) exhibits quadratic behaviour, as can be seen by calculating these figures by a computer program. This is in contrast to the conjecture in [7] that $O(L \log L)$ steps are sufficient. It is easy to see that *steepest descent* needs one move for every 1 in the Gray code representation of the starting point, whose expected number is $L/2$. Thus, *steepest descent* needs $\Omega(L^2)$ function evaluations on average.

4. FIXED DISTRIBUTION ALGORITHMS

We now consider the class of $(1+1)$-EAs in which the mutation step size is drawn from a fixed probability distribution μ, as follows:

1: Let x be a random point in $[n]$
2: Let $d \in \{1, 2, \ldots, n-1\}$ be drawn according to μ.
3: Let $y = x + d$
4: Let $z = x - d$
5: Let x be the best of $\{x, y, z\}$
6: Go to 2.

Again, "best" in line 5 indicates both legal and with smallest f-value. Note that bit mutations on binary strings do not belong to this class of algorithm, as whether or not you add or subtract the corresponding number depends on the current state.

The main motivation for studying algorithms of this form is that they are less prone to being trapped in local optima on multimodal functions. There is always some probability (depending on the distribution μ) of escaping to a better point. Of course the main problem is that one doesn't know where the better points are, and so one should find a balance between small and large jumps. A probability distribution which has this kind of scale free property is the Harmonic distribution:

$$\mu(d) = \frac{1}{dH_{n-1}}, \quad \text{where } H_{n-1} = \sum_{k=1}^{n-1} \frac{1}{k}.$$

The $(1+1)$-EA using this distribution is analysed in [1] for a particular unimodal function. Here we present an upper bound for general unimodal functions.

THEOREM 4. *The $(1+1)$-EA using the Harmonic distribution takes on average $O((\log n)^2)$ iterations to find the optimum of a unimodal function.*

PROOF. As previously, given our current point x, we let $A_x = \{y \mid f(y) \leq f(x)\}$, which forms an interval, containing the optimum, with x at one end. Now let B_x be the best half of these points (sorted by f-value). Then B_x is also an interval containing the optimum. We estimate the probability of moving from x to a point in B_x. Since the probability of moving a distance d is monotonically decreasing with d, it follows that the worst case is when the interval B_x is a distance $|B_x|$ away from x. The probability of moving to a point in this interval in one iteration is

$$\sum_{d=|B_x|}^{2|B_x|} \frac{1}{dH_{n-1}} = \frac{1}{H_{n-1}} \left(H_{2|B_x|} - H_{|B_x}\right) \approx \frac{\log 2}{\log(n-1)}$$

Any other moves can only shorten the distance to be jumped and so increase this probability. The expected waiting time to arrive in B_x is therefore $O(\log n)$. Since we need to repeat this at most $\log n$ times to locate the optimum, the result follows. □

This result shows that our Harmonic $(1+1)$-EA performs about as efficiently as local search using a Gray code representation, without the drawback of getting stuck in local optima. In fact a continuous version of this algorithm has been used successfully on a number of benchmark and real-world problems [6, 3]. One would like to know, however, if one couldn't do better by an even cleverer choice of probability distribution. The following result, reported in [1] indicates that improvements by only a constant scaling factor are possible.

THEOREM 5. *There exist unimodal functions for which the running time of a fixed distribution $(1+1)$-EA requires $\Omega((\log n)^2)$ steps to find the optimum.*

PROOF. The function concerned is simply $f(x) = x$, with the optimum at 0. See [1] for details. □

5. 2-D UNIMODAL FUNCTIONS

We have so far been concerned with one-dimensional functions and have seen that there exist efficient algorithms for solving unimodal problems in one dimension using black box algorithms such as EAs. It is natural to ask whether or not this continues to be the case if the search space is two-dimensional (or higher). It is known that black box search algorithms are not efficient for solving discrete unimodal functions when the complexity is considered as a function of the dimension of the problem, but where the precision (that is, the number of points per dimension) is fixed. This can be seen, for example, with the class of *long-path* problems defined on $\{0,1\}^n$ described in [2]. In contrast, here we consider problems with fixed dimension, but varying precision. We show that for two dimensions (and, therefore, for problems in higher dimensions as well), there are unimodal functions which black box algorithms cannot optimise efficiently.

Let $[n] = \{0, \ldots, n-1\}$. A point $p = (x, y) \in [n] \times [n]$ has up to four *neighbours*, $(x-1, y)$, $(x+1, y)$, $(x, y-1)$, and $(x, y+1)$. (Such a pair p' is not a neighbour, if $p' \notin [n] \times [n]$.) A point $p \in [n] \times [n]$ is a *maximum* of f, if for all neighbours p' of p it holds $f(p) \geq f(p')$. The function f is *unimodal*, it if has exactly one maximum. Let $\mathcal{F}_n$ be the family of unimodal functions $f : [n] \times [n] \to \mathbb{R}$.

THEOREM 6. *For any $n \in \mathbb{N}$ and any randomized black-box algorithm A, there exists a unimodal function $f \in \mathcal{F}_n$ such that A queries an expected number of $\Omega(n^{2/3})$ points, before querying the maximum of f.*

We use Yao's Min-Max principle (see for example Chapter 9 of [8]) to prove the theorem. In order to apply the principle, we have to restrict ourselves to a finite set of functions and to a finite set of deterministic black-box algorithms.

In the following, n is fixed, and w.l.o.g. $k = n^{1/3}$ is an odd integer. Let $\mathcal{F}'_n$ be the family of unimodal functions $f : [n] \times [n] \to \{-2n, \ldots, n^2\}$. Further, let $\mathcal{A}$ be the family of *deterministic* black-box algorithms for functions $f \in \mathcal{F}'_n$, that do not query any point twice. Clearly, $\mathcal{F}'_n$ and $\mathcal{A}$ are finite, and we can apply Yao's Min-Max principle. Moreover, it is easy to see that any black-box algorithm A' for $\mathcal{F}_n$ can be replaced by a black-box algorithm $A \in \mathcal{A}$ that finds the maximum of any function $\mathcal{F}'_n$ with at most as many queries as A'. (Whenever algorithm A' queries a point for the second time, A omits that query.)

For $A \in \mathcal{A}$ and a probability distribution μ over $\mathcal{F}'_n$, let $T_{A,\mu}$ be the random variable that denotes the number of queries A performs until it queries the maximum of f, if f is picked at random according to probability distribution μ. Theorem 6 follows immediately from the following lemma.

LEMMA 2. *There exists a probability distribution μ over $\mathcal{F}'_n$, such that for any black-box algorithm $A \in \mathcal{A}$*

$$E(T_{A,\mu}) = \Omega(k^2).$$

The rest of this section is devoted to the proof of the lemma. We define the probability distribution μ by constructing a random function $f \in \mathcal{F}'_n$ as described in the following.

Consider a path $p = (v_0, \ldots, v_\ell)$ in $[n] \times [n]$, such that

(i) $v_0 = (0, 0)$,

(ii) v_{i+1} *is a neighbour of* v_i *for* $0 \le i < \ell$, *and*

(iii) $v_i \neq v_j$ *for* $i \neq j$.

For such a path, we define function $f_p : [n] \times [n] \to \{-2n, \ldots, n^2\}$ by

$$f_p(x,y) = \begin{cases} i & \text{if } (x,y) = v_i,\ 0 \le i \le \ell, \text{ and} \\ -x - y & \text{if } (x,y) \text{ is not on the path } p. \end{cases} \quad (1)$$

It is easy to see that f_p is unimodal: Clearly v_ℓ is the unique maximum, because for every point $(x, y) = v_i \neq v_\ell$ on the path, the neighbour v_{i+1} has a larger function value, and for every point (x, y) not on the path either $(x, y - 1)$ or $(x - 1, y)$ is a neighbour and has a larger function value.

We now show how to construct path p at random. This way, we obtain a random construction for a function f_p, and thus a probability distribution μ over $\mathcal{F}'_n$.

Recall that $k = n^{1/3}$. First, we partition the grid into k^2 *sectors* $S^{a,b}$, $0 \le a, b < k$, where each sector is a $k^2 \times k^2$-sub-grid. Sector $S^{a,b}$ consists of the points (x, y), where $ak^2 \le x < (a + 1)k^2$ and $bk^2 \le y < (b + 1)k^2$.

It is helpful to visualize the grid as having the point $(0, 0)$ in the bottom left and the point $(n-1, 0)$ in the bottom right corner. Similarly, the sector $S^{0,0}$ is the $k \times k$ sub-grid in the bottom left corner of the $[n] \times [n]$ grid, and $S^{k-1,0}$ is in the bottom right corner. We now order the k^2 sectors as follows in a "snake-like" way. First comes sector $S^{0,0}$ and then the sectors $S^{1,0}, S^{2,0}, \ldots, S^{k-1,0}$ (i.e., the bottom sectors from left to right). Then follows the sector above $S^{k-1,0}$, namely $S^{k-1,1}$, and then all the sectors in the second row from right to left up to $S^{0,1}$. Then this continues with the sectors from the third row, ordered from left to right, and so on, until the last sector $S^{k-1,k-1}$ (recall that k is odd). Formally, the i-th sector $(0 \le i < k^2)$ is $S_i = S^{a,b}$, where $b = \lfloor i/k \rfloor$ and

$$a = \begin{cases} i \bmod k & \text{if } b \text{ is even, and} \\ (-i - 1) \bmod k & \text{if } b \text{ is odd.} \end{cases}$$

As a consequence, S_{i+1} is either to the left, to the right, or above of S_i. For convenience, we define $S_i = \emptyset$ for $i < 0$ and $i \ge k^2$.

The idea is the following: Our random path p passes through the sectors $S_1, S_2, \ldots$, and ends in sector S_{k^2-1}. The random construction of p will ensure, that if the algorithm has no information about sectors $S_{\alpha-1}, S_\alpha, S_{\alpha+1}$ for some $1 < \alpha < k^2 - 1$, i.e., if it has never queried any points from these sectors, then it is very unlikely that a query in S_α will hit the path p. Thus, in order to find the endpoint of the path, an algorithm must either follow the path from sector to sector (which requires at least $\Omega(k^2)$ queries due to the number of sectors), or it has to search for a point on p in sector S_{k^2-1} without having any information about the location of p in that sector. It will turn out, that the latter also takes an expected number of $\Omega(k^2)$ queries.

We now describe how to generate the random path p. In each sector S_i, $i \not\equiv 0 \pmod k$, we pick a point $q_i = (x_i, y_i)$ uniformly at random. We let $q_0 = (x_0, y_0) = (0, 0)$, and for $i \equiv 0 \pmod k$, $i > 0$, point q_i has the same x-coordinate as q_{i-1} and the same y-coordinate as q_{i+1}, i.e., $(x_i, y_i) = (x_{i-1}, y_{i+1})$. (Note that if $i \equiv 0 \pmod k$ and $i > 0$, then sector S_i is *above* sector S_{i-1}, so this definition ensures that $q_i \in S_i$.)

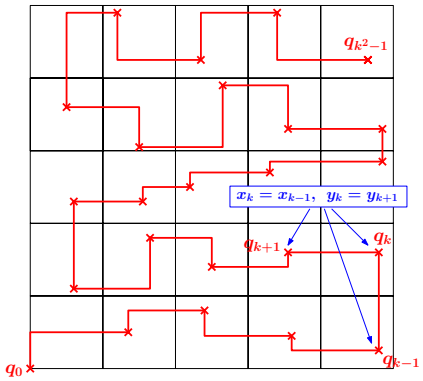

Figure 3: A random path for $k = 5$

Finally, we connect each pair of points q_i, q_{i+1}, $0 \le i < k^2 - 1$ by a subpath p_i, that first goes from (x_i, y_i) in y-direction until it reaches (x_i, y_{i+1}), and then in x-direction until it reaches (x_{i+1}, y_{i+1}). Hence, p_i contains exactly the points (x, y), where either $x = x_i$ and $\min\{y_i, y_{i+1}\} \le y \le \max\{y_i, y_{i+1}\}$, or $y = y_{i+1}$ and $\min\{x_i, x_{i+1}\} \le x \le \max\{x_i, x_{i+1}\}$. Note also that if S_i is above S_{i-1} (i.e., $i \equiv 0 \pmod k$ and $i > 0$), then the path from p_{i-1} to p_i goes only in y-direction, and the path from p_i to p_{i+1} goes only in x-direction. This way it is ensured that p_i and p_{i+1} have only their connecting point q_{i+1} in common. Finally, path p is the concatenation of $p_0, \ldots, p_{k^2-1}$. See Figure 3 for an example of a random path p for $k = 5$.

It is easy to see that Properties (i)–(iii) from above hold.

Claim 2. *Let* $u = (x, y) \in S_\alpha$ *for* $1 \le \alpha < k^2$ *be a point on path* p. *If* $\alpha \equiv k - 1 \pmod k$, *then* $y = y_\alpha$ *or* $x = x_\alpha$. *Otherwise,* $y \in \{y_\alpha, y_{\alpha+1}\}$ *or* $x = x_\alpha$.

Proof. If $\alpha \not\equiv k-1 \pmod k$, then the path p enters sector S_α at a point with y-coordinate y_α and leaves the sector at a point with y-coordinate $y_{\alpha+1}$. Inbetween, the path goes in y-direction only when its points have x-coordinate x_α. For $\alpha \equiv k - 1 \pmod k$, note that the path either ends at q_α (if $\alpha = k^2 - 1$), or goes from q_α to $q_{\alpha+1}$ only in y-direction. □

As mentioned above, this random construction of p defines a probability distribution μ over the functions in $\mathcal{F}'_n$. In the following, whenever we talk about a random path p or a random function f_p, we mean a path constructed according to this random experiment, or a function f_p picked according to distribution μ.

We now prove some essential properties of the random function f_p. Recall that by (1) a point $u \in [n] \times [n]$ is on path p if and only if $f_p(u) \ge 0$.

Claim 3. *For any point* u *in* S_α, $1 \le \alpha < k^2$ *it holds*

$$\operatorname*{Prob}_{\mu}\big(f_p(u) \ge 0\big) \le \frac{3}{k^2}.$$

Proof. Since each of the random numbers x_α, y_α, and $y_{\alpha+1}$, is uniformly distributed over a domain of size k^2, the claim follows immediately from Claim 2. □

We now show that whether or not a point is on the path in some sector S_α is independent of which points on the path

have been sampled in $S_1 \cup \cdots \cup S_{\alpha-2}$ and whether or not any points on the path have been sampled in $S_{\alpha+2} \cup \cdots S_{k^2-1}$.

For a real number $x \neq 0$ we define $\text{sign}(x) = x/|x|$, and $\text{sign}(0) = 1$.

CLAIM 4. *Let α, i, t be integers such that $1 \leq \alpha < k^2$, and $0 \leq r \leq t$. Let $u \in S_\alpha$, $1 < \alpha < k^2$, and let $u_1, \ldots, u_r$ be arbitrary points in $S_0 \cup \cdots \cup S_{\alpha-2}$, and $u_{r+1}, \ldots, u_t$ be arbitrary points in $S_{\alpha+2} \cup \cdots \cup S_{k^2-1}$. Finally, let $z_1, \ldots, z_r \in \mathbb{R}$ and $\sigma_{r+1}, \ldots, \sigma_t \in \{-1, 1\}$ such that the event*

$$\mathcal{E} : \forall 1 \leq j \leq r : f_p(u_j) = z_j \wedge \forall r < j \leq t : \text{sign}\big(f_p(u_j)\big) = \sigma_j$$

occurs with a positive probability. Then the events $\mathcal{E}$ and "$f_p(u) \geq 0$" are independent.

PROOF. Let $p' = p \cap (S_0, \ldots, S_{\alpha-2})$ and $p'' = p \cap (S_{\alpha+2} \cup \cdots \cup S_{k^2-1})$. The function values $f_p(u_j)$, $j > r$, can depend on the length of the subpath of p that goes through $S_{\alpha-1} \cup S_\alpha \cup S_{\alpha+1}$, but the sign of $f_p(u_j)$ only depends p''. On the other hand, the function values $f_p(u_j)$, $j \leq r$, are uniquely determined by p'. Hence, whether or not event $\mathcal{E}$ occurs depends only on p' and p''.

By Claim 2, the event "$f_p(u) \geq 0$" may depend only on the random variables x_α, y_α, and, if $\alpha \not\equiv k-1 \pmod k$, on $y_{\alpha+1}$. Even if we fix p' and p'' arbitrarily, point $q_\alpha = (x_\alpha, y_\alpha)$ can still be anywhere in sector S_α. Moreover, if $(\alpha+1) \not\equiv 0 \pmod k$, the choice of $y_{\alpha+1}$ is not restricted by p' or p'', either. Hence, the values x_α, y_α, and, if necessary also $y_{\alpha+1}$ are still uniformly distributed (over their respective domain of size k^2). Thus, whether event "$f_p(u) \geq 0$" occurs is independent from p' and p'', and thus also from $\mathcal{E}$. □

CLAIM 5. *Let $H = \{u_1, \ldots, u_t\} \subseteq [n] \times [n]$. Further, let $z_1, \ldots, z_t \in \mathbb{R}$, and $\alpha \in \{1, \ldots, k^2 - 1\}$, and $r \in \{0, \ldots, t\}$ such that*

1. *$\forall 1 \leq j \leq r$: $u_j \in S_1 \cup \cdots \cup S_{\alpha-2}$.*
2. *$\forall r < j \leq t$: $u_j \in S_{\alpha+2} \cup \cdots \cup S_{k^2-1}$ and $z_j < 0$.*

Then for any point $u \in S_{\alpha+2} \cup \cdots \cup S_{k^2-1}$ it holds[1]

$$\Prob_\mu(f_p(u) \geq 0 \,|\, \forall 1 \leq j \leq t : f_p(u_j) = z_j) \leq \frac{1}{k^2/3 - t + r}.$$

PROOF. Assume that $\text{Prob}(\forall 1 \leq j \leq t : f_p(u_j) = z_j) > 0$, because otherwise there is nothing to show. For any point u^*, if $f_p(u^*) < 0$, then u^* is not on the path and $f_p(u^*)$ is uniquely determined by the coordinates of u^*. Since $z_j < 0$ for $j > r$, we have

$$\forall r < j \leq t : \big(f_p(u_j) < 0 \;\Leftrightarrow f_p(u_j) = z_j\big). \tag{2}$$

Now consider the following events:

$\mathcal{E}$: $\forall 1 \leq j \leq r :\ f_p(u_j) = z_j$.

$\mathcal{E}'$: $\forall r < j \leq t :\ f_p(u_j) = z_j$.

By Claim 4, the event "$f_p(u) \geq 0$" is independent from event $\mathcal{E}$, and so by Claim 3,

$$\text{Prob}(f_p(u) \geq 0 \,|\, \mathcal{E}) \leq \frac{3}{k^2}. \tag{3}$$

For the same reason, for any $r < j \leq t$

$$\text{Prob}(f_p(u_j) \geq 0 \,|\, \mathcal{E}) \leq \frac{3}{k^2},$$

and thus

$$\text{Prob}(\mathcal{E}' \,|\, \mathcal{E}) \overset{(2)}{=} \text{Prob}\left(\bigwedge_{r<j\leq t} f_p(u_j) < 0 \,\middle|\, \mathcal{E} \right)$$

$$\geq 1 - \sum_{r<j\leq t} \text{Prob}\big(f_p(u_j) \geq 0 \,\big|\, \mathcal{E}\big) \overset{(3)}{\geq} 1 - (t-r)\frac{3}{k^2}. \tag{4}$$

This leads us to

$$\begin{aligned}
&\Prob_\mu(f_p(u) \geq 0 \,|\, \forall 1 \leq j \leq t : f(u_j) = z_j) \\
= \;& \Prob_\mu(f_p(u) \geq 0 \,|\, \mathcal{E} \wedge \mathcal{E}') \\
= \;& \frac{\text{Prob}_\mu(\mathcal{E}' \wedge f_p(u) \geq 0 \,|\, \mathcal{E})}{\text{Prob}_\mu(\mathcal{E}' \,|\, \mathcal{E})} \\
\leq \;& \frac{\text{Prob}_\mu(f_p(u) \geq 0 \,|\, \mathcal{E})}{\text{Prob}_\mu(\mathcal{E}' \,|\, \mathcal{E})} \\
\overset{(3),(4)}{\leq} \;& \frac{3/k^2}{1 - 3(t-r)/k^2} = \frac{1}{k^2/3 - t + r}
\end{aligned}$$

□

Now let $A \in \mathcal{A}$ be a black-box algorithm for functions in $\mathcal{F}'_n$. The maximum of f_p is v_ℓ, the last point on path p. Note that v_ℓ is in section S_{k^2-1}. We prove that the expected number of queries A needs until it finds v_ℓ is $\Omega(k^2)$.

After algorithm A has made t queries, the *query-history* H is the set $\{u_1, \ldots, u_t\}$ of points that A has queried. We define a potential function Φ that maps the set of possible query-histories to $\mathbb{N}$, and that measures the amount of progress algorithm A has made.

One way for the algorithm to make progress is to find a point on p in a sector S_i that is closer to the destination than all other sectors in which A has found points on p, yet. Hence, one component of our potential is $\varphi(H)$, the largest number of a sector, such that A has queried a point in that sector that is on path p. That is,

$$\varphi(H) = \max\{0, j \,|\, \exists u \in H \cap S_j : f_p(u) \geq 0\}.$$

Another way is to query points in sectors S_i, $i > \varphi(H)$, because this eliminates points that are not on p, and increases the chances of finding the path in a "higher" sector. Thus, the second component of our potential is $\gamma(H)$, which denotes the number of points queried in sectors "above" $S_{\varphi(H)+1}$, i.e.,

$$\gamma(H) = |H \cap (S_{\varphi(H)+1} \cup \cdots \cup S_{k^2-1})|.$$

The potential of the query-history H is a weighted sum of these two components, with a cap at $k^2/3$:

$$\Phi(H) = \min\left\{ \frac{k^2}{3},\ \varphi(H) + 4\gamma(H) \right\}.$$

Before A has queried a point, its query-history is empty, and $\Phi(\emptyset) = 0$. On the other hand, when A has queried the maximum v_ℓ, then a point on $p \cap S_{k^2-1}$ is in the query-history H, so $\varphi(H) = k^2 - 1$ and $\Phi(H) = k^2/3$. Therefore, during a run of algorithm A the potential of its query-history increases from 0 to $k^2/3$. We show that with each query the expected increase in potential is at most constant.

CLAIM 6. *Fix a sequence of points $H = \{u_1, \ldots, u_t\} \subseteq [n] \times [n]$, and a sequence of function values $z_1, \ldots, z_t \in$*

[1] We define $\text{Prob}(\mathcal{E}_1 | \mathcal{E}_2) = 0$ if $\text{Prob}(\mathcal{E}_2) = 0$.

$\mathbb{R}$. *Then for any* $u \in [n] \times [n]$ *it holds*

$$\mathop{E}_{\mu}\big(\Phi(H \cup \{u\}) - \Phi(H) \,\big|\, \forall 1 \le j \le t : f_p(u_j) = z_j\big) \le 5.$$

Proof. In the following we work under the condition that $f_p(u_j) = z_j$ for $1 \le j \le t$. Hence, $\varphi(H)$, $\gamma(H)$, and thus also $\Phi(H)$ are uniquely determined. All probabilities below are conditional probabilities, given the event "$\forall 1 \le j \le t : f_p(u_j) = z_j$" (and assuming that this event occurs with a probability larger than 0).

Let $i \in \{0, \ldots, k^2 - 1\}$ such that $u \in S_i$, and let $H' = H \cup \{u\}$. Clearly,

$$\varphi(H') = \begin{cases} i & \text{if } i > \varphi(H) \text{ and } f_p(u) \ge 0; \text{ and} \\ \varphi(H) & \text{otherwise.} \end{cases}$$

Hence, if $i \le \varphi(H)$, the potential can increase by at most $4(\gamma(H') - \gamma(H)) \le 4$. Thus, assume that $i > \varphi(H)$.

Let $s := |H \cap (S_{\varphi(H)+1} \cup \cdots \cup S_i)|$. In the event that $f_p(u) \ge 0$ and thus $\varphi(H') = i$, we have $\gamma(H') - \gamma(H) \le -s$ because in this case the s elements in $H \cap (S_{\varphi(H)+1} \cup \cdots \cup S_i)$ contribute to $\gamma(H)$ but not to $\gamma(H')$. Therefore, we have

$$\Phi(H') - \Phi(H) \begin{cases} = i - \varphi(H) - 4s & \text{if } f_p(u) \ge 0, \text{ and} \\ \le 4 & \text{otherwise.} \end{cases} \quad (5)$$

Hence, if $4s \ge \big(i - \varphi(H)\big) - 4$, the increase in potential is at most 4. Therefore, we assume now that $4s < (i - \varphi(H)) - 4$. Then

$$s < \frac{i - \varphi(H)}{4} - 1 \le \left\lfloor \frac{i - \varphi(H)}{4} \right\rfloor.$$

This implies that among the $i - \varphi(H)$ sets $S_{\varphi(H)+1}, \ldots, S_i$, there are 4 consecutive ones that do not intersect with H. Hence, there is at least one set S_α, $\varphi(H) + 1 < \alpha < i - 1$, such that $H \cap (S_{\alpha-1} \cup S_\alpha \cup S_{\alpha+1}) = \emptyset$. In other words, for all $1 \le j \le t$ it holds $u_j \notin S_{\alpha-1} \cup S_\alpha \cup S_{\alpha+1}$, and since $\varphi(H) < \alpha - 1$

$$\text{if } u_j \in S_{\alpha+2} \cup \cdots \cup S_{k-1}, \text{ then } f_p(u_j) < 0.$$

Moreover, $u \in S_{\alpha+2} \cup \cdots \cup S_{k^2-1}$ due to the choice of $\alpha < i - 1$. Thus, we are exactly in the same situation as in Claim 5, with $r \ge t - \gamma(H)$ (because at least $t - \gamma(H)$ of the t search points in H are actually in $S_1 \cup \cdots \cup S_{\alpha-2}$). Hence, we can conclude that (recall that the probabilities are conditional, given the event "$\forall 1 \le j \le t : f_p(u_j) = z_j$")

$$\epsilon := \text{Prob}(f_p(u) \ge 0) \le \frac{1}{k^2/3 - t + r} \le \frac{1}{k^2/3 - \gamma(H)}$$

Since $\Phi(H') \le k^2/3$ and $\Phi(H) \ge 4\gamma(H)$, we have in any case (no matter whether $f_p(u) \ge 0$ or not)

$$\Phi(H') - \Phi(H) \le k^2/3 - 4\gamma(H).$$

Moreover, if $f_p(u) < 0$, then $\Phi(H') - \Phi(H) \le 4$ according to (5). Hence,

$$\begin{aligned} E_\mu\big(\Phi(H') - \Phi(H)\big) &\le (1 - \epsilon)4 + \epsilon(k^2/3 - 4\gamma(H)) \\ &\le 4 + \frac{k^2/3 - 4\gamma(H)}{k^2/3 - \gamma(H)} \le 5. \end{aligned}$$

□

Utilizing [4, Lemma 12] it follows that A needs an expected number of $\Omega(k^2)$ queries until the potential has increased from 0 to $k^2/3$. This completes the proof of Lemma 2.

6. CONCLUSIONS

We investigated the problem of optimising unimodal functions in terms of the precision used. An evolutionary algorithm which halves its mutation step size at each iteration optimises such functions in $O(\log n)$ steps and is guaranteed to find a local optimum in multi-modal problems. If we use binary strings of length L to represent the points using the reflective Gray code then the running time is $O(L^2)$, using steepest and next descent algorithms. A (1+1)-ES was given in which the mutation step size is drawn from the Harmonic distribution. This optimises unimodal functions in $O((\log n)^2)$ steps. It has previously been shown that there exist unimodal functions for which this running time is optimal (up to a constant factor) for fixed distribution algorithms. Finally, we considered functions defined on two dimensions (and higher), and showed that black box algorithms cannot optimise unimodal functions efficiently (as a function of the precision) on such spaces.

7. REFERENCES

[1] M. Dietzfelbinger, J. E. Rowe, I. Wegener, and P. Woelfel. Tight bounds for blind search on the integers. In *Proc. 25th International Symposium on Theoretical Aspects of Computer Science*, 2008.

[2] S. Droste, T. Jansen, and I. Wegener. On the optimization of unimodal functions with the (1 + 1) evolutionary algorithm. In *Proc. 5th International Conference on Parallel Problem Solving from Nature*, pages 13–22. Springer-Verlag, 1998.

[3] D. Hidović and J. E. Rowe. Validating a model of colon colouration using an evolution strategy with adaptive approximations. In K. Deb, editor, *GECCO 2004, LNCS 3102*, pages 1005–1016. Springer-Verlag, 2004.

[4] J. Jägersküpper. Algorithmic analysis of a basic evolutionary algorithm for continuous optimization. *Theoretical Computer Science*, 279:329–347, 2007.

[5] J. Kiefer. Sequential minimal search for a maximum. *Proc. Amer. Math. Soc.*, 4:502–506, 1953.

[6] J. E. Rowe and D. Hidović. An evolution strategy using a continuous version of the gray-code neighbourhood distribution. In K. Deb, editor, *GECCO 2004, LNCS 3102*, pages 725–736. Springer-Verlag, 2004.

[7] J. E. Rowe, L. D. Whitley, L. Barbulescu, and J.-P. Watson. Properties of Gray and binary representations. *Evolutionary Computation*, 12(1):47–76, 2004.

[8] I. Wegener. *Complexity Theory: Exploring the Limits of Efficient Algorithms*. Springer-Verlag, 2005.

[9] L. D. Whitley. A free lunch proof for Gray versus binary encodings. In W. Banzhaf *et al*, editor, *GECCO 1999*, pages 726–733. Morgan Kaufmann, 1999.

[10] L. D. Whitley and J. E. Rowe. Gray, binary and real valued encodings: Quad search and locality proofs. In A. H. Wright, M. D. Vose, K. De Jong, and L. Schmitt, editors, *Foundations of Genetic Algorithms, Vol. 8. LNCS 3469*, pages 21–36. Springer-Verlag, 2005.