RANDOM BINARY SEARCH: A RANDOMIZING ALGORITHM FOR GLOBAL OPTIMIZATION IN $\mathbb{R}^*$

EITAN ZEMEL

Northwestern University and Tel Aviv University

Randomizing (stochastic) global optimization algorithms can be viewed as sampling procedures, where at each iteration a local optimum is sampled from the set of local optima. The effectiveness of such an algorithm depends crucially on the sampling distribution, i.e., on the probabilities of sampling the various local optima. There exists a very large body of research on stochastic global optimization. However, very little is known about the sampling distribution itself and, in particular, on the specific way in which it depends on the function being optimized, on the region in which the search takes place, and on the optimization routine. In this paper we set out to examine these issues in some detail and present some first results in this direction. The case analyzed in this paper is the optimization of a one-dimensional function using a randomizing version of binary search. We give an effective (computable in quadratic time) formula for the sampling distribution and obtain various interesting properties of this distribution which are relevant to the behavior of the algorithm in practice. We also discuss some issues related to adaptive search.

1. Introduction. Randomizing (stochastic) global optimization algorithms can be viewed as sampling procedures, where at each iteration a local optimum is sampled from the set of local optima. The effectiveness of such an algorithm depends crucially on the sampling distribution, i.e., on the probabilities of sampling the various local optima. There exists a very large body of research on stochastic global optimization (cf. [1, 2, 4, 6, 7, 10, 11]). (See also the recent excellent surveys and extensive reference lists in the theses of Boender [3] and Timmer [13] and in the survey of Rinnooy Kan and Timmer [9].) However, very little is known about the sampling distribution itself and, in particular, on the specific way in which this distribution depends on the function being optimized, on the region in which the search takes place and on the optimization routine. In this paper we set out to examine this issue in some detail and present several results which could be viewed as first steps in this direction.

The case analyzed in this paper is the relatively simple case of optimizing a function of one variable. In practice, of course, one is more likely to face a multivariate function. Nevertheless, the one-dimensional case constitutes an important subroutine of numerous multidimensional global optimization routines. A challenging open problem is to extend the results of this paper to the multivariate case.

The algorithm we consider is an obvious randomizing version of binary search. We present a detailed analysis of the (stochastic) performance of this algorithm and present an effective (computable in quadratic time) formula for the sampling distribution. We also examine several properties of this distribution which allow us to reach some surprising conclusions concerning this algorithm. It would be very interesting to

*Received October 12, 1984; revised September 5, 1985.
OR/MS Index 1978 subject classification. Primary: 661 Programming/nonlinear/unconstrained.
Key words. Global optimization, nonconvex optimization, randomizing algorithms.
†Research for this paper was supported in part by National Science Foundation Grant No. ECS-8121941 to Northwestern University.

651

0364-765X/86/1104/0651$01.25
Copyright © 1986, The Institute of Management Sciences/Operations Research Society of America
observe whether similar analysis can be carried out for other randomizing one dimensional algorithms.

The structure of this paper is the following. In §2 we present the necessary preliminaries. The analysis is contained in §3. §4 concerns the computation of the sampling probabilities. Their properties are discussed in §5. §6 analyzes the completely symmetric case. Finally, in §7, we explore some issues related to adaptive search.

2. Notations and preliminaries. Let $f$ be a continuous, almost everywhere differentiable function of one variable defined over an interval $u = [a, b]$. We wish to find the global minimum of $f$ over $u$, i.e., a point $x^* \in u$ such that $f(x^*) \leq f(x)$ for every $x \in u$. The deterministic version of our algorithm runs as follows:

**Algorithm binary search (BS).**

Let $u_1 = u$.

For $i = 1, 2, \ldots$

1. Let $z_i$ be the center of the interval $u_i$.
2. If $f'(z_i) > 0$ set $u_{i+1} = u_i \cap [a, z_i]$. 
3. If $f'(z_i) < 0$ set $u_{i+1} = u_i \cap [z_i, b]$. 
4. Else stop.

End For

In the randomizing version of the algorithm (RBS), we replace the deterministic selecting of $z_i$ (line (1)) by the randomizing step:

(1*) Let $z_i$ be chosen using the uniform distribution over $u_i$.

It is obvious that each of the intervals $u_i$ defined by either (BS) or (RBS) must contain at least one local minimum (possibly at one of its end points). Before we investigate (RBS) in more detail we have to clarify several points regarding our conventions.

(i) The algorithm requires evaluation of the sign of $f'(z)$ at arbitrary points $z \in u$. We will regard the computational effort associated with this task as one unit.

(ii) The algorithm terminates prematurely when $f'(z_i) = 0$, not necessarily with a local minimum on hand. We will assume that $f'(z) \neq 0$ almost everywhere in $u$ so that such termination hardly ever occurs for (RBS). We also assume that $f$ has a finite number of local optima over $u$.

(iii) Finally, we note that the algorithm contains an infinite loop and, as stated, would run forever. This convention is convenient in the analysis of §§3–7. In practice, one would include a certain stopping rule to ensure finiteness. A typical rule is to stop the algorithm when the size of $u_i$ is less than some user specified constant $\delta > 0$ (see [5] for a discussion of this point). The time required to achieve that by (BS) is $\log_2((b - a)/\delta)$. For (RBS), the time required is a random variable which cannot be bounded a priori. However, it can be shown to be of a similar order of magnitude to within any arbitrary level of certainty:

**Theorem 1.** Let $r = \log_2((b - a)/\delta)$ and let $m > 1$. Also, let $t(m) = 3(m - 1)^2/8m$, and let $|u_i|$ be the length of the interval $u_i$. Then

$$\Pr[|u_{3mr}| \geq \delta] \leq 2e^{-rt(m)}.$$

**Proof.** Call an iteration (i.e., one scan of line 1*, 2, 3) a success if $|u_{i+1}|/|u_i| \leq 2/3$. The probability of success is at least 1/3. Also, after $r$ successes, the size of $u$ cannot exceed $\delta$. The right side of the assertion is an upper bound on the probability that $3mr$ trials would result in less than $r$ successes. It is based on the normal approximation to the binomial due to Rénye [13] (see [8] or [15] for a more detailed analysis for a similar approximation). A similar result also can be obtained from an approximation due to Chernoff.
It follows from Theorem 12, by letting $\delta$ go to zero, that the infinite sequence $z_i$, $i = 1, \ldots$, obtained by the algorithm (RBS) converges to a local minimum of $f$ over $u$ with probability one. Obviously, the identity of this minimum is random, with some minima being more probable than others. We are interested in computing these probabilities explicitly. Before proceeding further the reader may find it instructive to examine the four simple examples (Figures 1–4) and establish on his own (at least qualitatively) the probabilities of the algorithm terminating at each of the local minima. We will examine the correct answers in §5.
3. **Analysis.** Consider the interval \( u = [a, b] \) and denote the number of local minima in this interval by \( n \) (possibly including the end points of the interval). Let the positions of these minima be at \( X = (x_1, x_2, \ldots, x_n) \) with \( a \leq x_1 < x_2, \ldots, < x_n \leq b \). Then the number of local maxima of \( f \) in \( u \) is \( n - 1 \), \( n \), or \( n + 1 \), depending on whether zero, one or two of the minima of \( f \) are obtained at end points. We use the following convention: if \( a \) or \( b \) is a local minimum of \( f \) then we include it also in the list of local maxima. Under this convention the number of local maxima is always \( n + 1 \). Denote their positions by \( Y = (y_1, y_2, \ldots, y_{n+1}) \) with \( y_1 < y_2 < y_3, \ldots, < y_{n+1} \). Thus, \( a = y_1 \leq x_1 < y_2 < \ldots, < x_n \leq y_{n+1} = b \). Let \( L_i = (y_i, x_i) \) and \( R_i = (x_i, y_{i+1}) \), \( i = 1, \ldots, n \), be the open intervals surrounding the local minima \( x_i \) to its left and right, respectively. We denote the lengths of these intervals by \( l_i \) and \( r_i \). The general case is depicted in Figure 5. Figure 6 depicts the situation when \( a \) is a local minimum. This causes \( x_1 \) to coincide with \( y_1 \) and thus \( L_1 = \emptyset \), \( l_1 = 0 \). For \( i = 1, \ldots, n \) let \( t_{2i} = l_i, \quad t_{2i+1} = r_i \), \( T = (t_1, \ldots, t_{2n}) \). For each interval \( v = [c, d] \subseteq u \) let \( |v| \) denote its length.

![Figure 5](image)

![Figure 6](image)

The only information on the function \( f \) used during the execution of an iteration of (RBS) is the sign of the derivative of \( f \) at the random points \( z_k \). That sign is positive at points of \( R = \bigcup_{i=1}^{n} R_i \) and negative at \( L = \bigcup_{i=1}^{n} L_i \). Thus, the progress of the algorithm is completely specified by the positions and lengths of the intervals \( R_i, L_i, \) \( i = 1, \ldots, n \). This means that there is no loss of generality if we view \( f \) as a piecewise linear function with consecutive segments having derivatives of alternating signs, as depicted in Figures 1–6. Moreover, since the position of the interval \( u \) on the \( x \) axis is obviously irrelevant, the probability sought depends only on the vector \( T = (t_1, \ldots, t_{2n}) \). Denote the probability that the algorithm converges to the minimum \( x_i \) by \( p_i(T) \) and let \( p(T) = (p_1(T), p_2(T), \ldots, p_n(T)) \) with \( \sum_{i=1}^{n} p_i(T) = 1 \). We now examine these probabilities in more detail.

Let \( z \in u \) be arbitrary. Assume we split \( u \) at \( z \) obtaining two subintervals \( ur(z) = [z, b] \) and \( ul(z) = [a, z] \). Let the corresponding \( T \) vectors for each of these subintervals be \( TR(z) \) and \( TL(z) \), respectively. Splits play a crucial role in our analysis since each iteration of (RBS) is essentially a split at a random point \( z \). In fact, we can easily
state a recursive expression for \( p_i(T) \) which is based on the observation that if \( z \) falls in \( R_j \), \( j \leq i \) then \( x_i \) becomes the \( i - j + 1 \) minimum of \( ur(z) \) and if \( z \) falls in \( L_j \), \( j > i \) then \( x_i \) becomes the \( i^{th} \) minimum in \( ul(z) \). Thus, we have

**Theorem 2.**

\[
\begin{align*}
P_i(T) &= \frac{1}{|U|} \left[ \left( \sum_{j=1}^{i} \int_{y_j}^{x_j} p_{i-j+1}(TR(z)) \, dz \right) + \sum_{j=i}^{n} \int_{x_j}^{y_j} p_j(TL(z)) \, dz \right].
\end{align*}
\]  

(4)

This looks like a hopeless formula. However, we will shortly be in position to simplify it considerably. To that end, we concentrate on a very special type of splits, namely, at the actual minima \( x_i, \ i = 1, \ldots, n \). Such splits occur in practice with probability 0. However, they possess some very special and useful properties. We assume henceforth that \( x_i \) is fixed and suppress references to its identity in our various constructs. Thus, we write \( ul \) for \( ul(x_i) \) and similarly for \( ur, TR, \) and \( TL \). Note that \( TL = (l_1, r_1, \ldots, l_n, 0) \) and \( TR = (0, r_1, \ldots, l_n, r_n) \). Obviously, \( x_i \) is the last (1\(^{st} \)) minimum in \( ul \) and the first in \( ur \). Denote the local minima to which the algorithm actually converges by \( x^* \).

**Theorem 3.**

(i) \( p_i(T) = p_i(TR) \cdot p_i(TL) \).

(ii) \( p_j(T, x^* \in \{ x_1, \ldots, x_i \}) = p_j(TL), \ for \ every \ j \leq i \).

(iii) \( p_j(T, x^* \in \{ x_1, \ldots, x_i \}) = p_j(TR), \ for \ every \ j \geq i \).

In words, (i) the probability of converging to \( x_i \) in \( u \) is equal to the product of the probabilities of doing so when the search is restricted to the left and right segments generated from \( u \) by a split at \( x_i \); (ii) the conditional probability of converging to \( x_j \), given that we converge to a minimum to the left of \( x_i \), is equal to the unconditional probability of converging to that minimum when the search is restricted to \( ul \)—similarly for (iii).

**Proof.** Consider the infinite sequence of points generated by the algorithm \( Z = (z_1, z_2, \ldots) \) and the sequence of intervals \( U = (u_1 \supseteq u_2, \ldots) \). Recall that \( z_k \) is uniform over \( u_k \) and that \( u_k \) depends on the initial segment of \( Z \), \( Z_{i-1} = (z_1, \ldots, z_{i-1}) \). It follows that \( z_i \) depends on \( Z_{i-1} \) in a rather complex fashion.

Partition the sequence \( Z \) into two subsequences \( ZR = (zr_1, zr_2, \ldots) \) and \( ZL = (zl_1, zl_2, \ldots) \) according to whether a given point \( z_k \) is to the right or left of \( x_i \). Also, partition each interval \( u_k \) into two subintervals \( ur_k = u_k \cap [x_i, b] \) and \( ul_k = u_k \cap [a, x_i] \). Then \( z_k \in ZR \) exactly if \( z_k \in ur_k \) and vice versa for \( ZL \). The probability of the former event is, of course, \( |ur_k|/|u_k| \). A typical sequence \( Z = (z_1, z_2, \ldots) \) is composed of alternating subsequences of \( ZR \) and \( ZL \), e.g., \( Z = (zr_1, zr_2, zl_1, zr_3, zl_2, \ldots) \).

The evolution of the process \( Z \) could be described in the following equivalent way. First, choose one of the two intervals \( ur_k \) or \( ul_k \) with probabilities \( |ur_k|/|u_k| \) and \( |ul_k|/|u_k| \) respectively. After this interval is chosen, pick \( z_k \) using the uniform distribution over that interval. \( z_k \) is classified as the next element of the \( ZR \) or \( ZL \) sequence in the obvious way. Finally, update the intervals \( ur_k \) and \( ul_k \) as follows: If \( z_k \in ur_k \cap L \) then \( ur_{k+1} = ur_k \cap [z_k, b] \) and \( ul_{k+1} = \emptyset \); and, if \( z_k \in ur_k \cap R \), then \( ur_{k+1} = ur_k \cap [a, z_k] \) and \( ul_{k+1} = ul_k \). Similarly, we handle the case of \( z_k \in ul_k \). Note that the only dependence between the sequences \( ZL \) and \( ZR \) is through the interval update in the case \( z_k \in ur_k \cap L \) or \( z_k \in ul_k \cap R \). In particular, \( z_k \in ur_k \cap L \) causes \( ZL \) to terminate (\( Z \) continues to coincide with \( ZR \) from this point on) and similarly for the other case.

Call the first iteration which satisfies one of these conditions a **pivot**. Note that the pivot satisfies \( z_k \in ul_k \cap R \) (a pivot the left) precisely when the process \( Z \) converges to a point strictly to the left of \( x_i \) and similarly to the right. The process converges to \( x_i \) precisely if the infinite sequence of iterations contains no pivot.
Define a process \( W = (w_1, w_2, \ldots) \) which is closely related to \( Z \) but easier to analyze as its right and left subsequences are completely independent. Specifically, we start with intervals \( v_1 = u_1, v_1 = u_1, v_1 = u_1 \). At each step we choose \( vr_k \) or \( vl_k \) with probabilities proportional to their length and pick \( w_k \) uniformly from the chosen interval. We include \( w_k \) in the \( WR \) sequence if it comes from \( vr_k \) and similarly for \( WL \). The only new feature here is the update of \( vr_k \) and \( vl_k \). If \( w_k \) is chosen from \( vr_k \cap R \) then \( vl_{k+1} = vl_k \) and \( vr_{k+1} = vr_k \cap [x_i, w_k] \) as is the case for the intervals \( U \). However, if \( w \in vr_k \cap L \) then we leave \( vl_{k+1} = vl_k \) (instead of setting it to \( \emptyset \) in the case of \( U \)) and set \( vr_{k+1} = vr_k \cap [w_k, b] \). Similarly, we update only \( vl_k \) if \( w \in vl_k \). Naturally, the sequence \( W \) is not implementable algorithmically since \( x_i \) is typically unknown.

The processes \( WL \) and \( WR \) are both infinite sequences which are completely independent. Furthermore, the distribution of \( WL \) relative to \( u \) is identical to the distribution of \( Z \) relative to \( ul \) and similarly for \( WR \) and \( ur \). Also, the processes \( W \) and \( Z \) are identical until the occurrence of the first pivot. If that pivot is to the left, then \( Z \) (which becomes \( ZL \) from this point on) continues to coincide with \( WL \) after this pivot. If the pivot is the right, then \( Z \) coincides with \( WR \) after the pivot.

We can now finalize the argument. For (i) note that the probability that both \( WR \) and \( WL \) converge to \( x_i \) is the right-hand side of the assertion since the two processes are independent. Also, since \( Z \) and \( W \) are identical until the occurrence of a pivot, the probability of a pivot occurring is identical in both processes. Finally, no pivot occurs in \( Z \) precisely if \( Z \) converges to \( x_i \), and no pivot occurs in \( W \) precisely if both \( WL \) and \( WR \) converge to \( x_i \). For (ii) observe that \( Z \) converges to \( x_i \) or a point to the left of \( x_i \) precisely if \( Z \) does not have any pivot to the right. Similarly for (iii).

**Remark 1.** The reader may note that part (ii) holds without any modification if the split is taken at any point \( x \in R \) and similarly for part (iii) and \( x \in L \).

**Remark 2.** Theorem 3 does not hold if one replaces the uniform distribution over \( u_i \) (step 1* of RBS) by another distribution.

An immediate but rather surprising corollary of Theorem 3(i) is the following:

**Corollary 3.1.** \( p_i(t_1, t_2, \ldots, t_{2n}) = p_i(0, t_2, \ldots, t_{2n-1}, 0) \).

In other words, the sampling probabilities \( p_i \) are independent of the length of the first and last subintervals of \( T \), \( l_1 \) and \( r_0 \) (1 and \( 2n \)).

**Proof.** Consider a split at \( x_i(x_n) \) and apply Theorem 3(i). \( \blacksquare \)

**Remark.** We note that although Corollary 3.1 guarantees that \( P_i(T) \), \( i = 1, \ldots, n \), is independent of \( t_1 \) and \( t_{2n} \), the progress of the algorithm (RBS) itself does depend on these parameters.

In the sequel we set, without loss of generality, \( l_1 = r_n = 0 \). Note that in this case \( |u| \) is simply \( x_n - x_1 \). Applying Corollary 3.1 to formula (4), we get the following.

**Theorem 4.**

\[
p_i(T) = \frac{1}{x_n - x_1} \left[ \sum_{j=2}^{i} l_j p_{i-j+1}(TR(x_j)) + \sum_{j=i}^{n-1} r_j p_{i}(TL(x_j)) \right]
\]

\[
= \frac{1}{x_n - x_1} \left[ \sum_{j=2}^{i} (x_j - y_j) p_{i-j+1}(0, r_j, \ldots, l_n, 0)
+ \sum_{j=1}^{n-1} (y_{j+1} - x_j) p_{j}(0, r_1, \ldots, l_j, 0) \right]. \quad \blacksquare
\]
(6) is a finite recursion which involves only subvectors of $T$. It can be used to calculate the entire vector $p(T)$ in $O(n^3)$ operations. However, using Theorems 3 and 4 together, one can do much better. This will be discussed in the following section.

### 4. The computation of $p(T)$

We are now in a position to obtain an explicit expression for $p_i(T)$ which can be evaluated in $O(n)$ computational steps. We start by examining the expression (6) for the special case $i = 1$. This yields

$$
p_1(0, r_1, l_2, \ldots, l_n, 0) = \frac{1}{x_n - x_1} \sum_{j=1}^{n-1} (y_{j+1} - x_j) p_1(0, r_1, l_2, \ldots, l_j, 0).
$$

Define $q_j = p_1(0, r_1, l_2, \ldots, l_j, 0)$ for $j = 2, \ldots, n$, $q_1 = 1$. Note that $p_1(T) = q_n$. Thus, (7) becomes

$$
q_n = \frac{1}{x_n - x_1} \sum_{j=1}^{n-1} (y_{j+1} - x_j) q_j.
$$

Applying the formula recursively we get for $k = 1, \ldots, n - 1$

$$
q_{k+1} = \frac{y_{k+1} - x_1}{x_{k+1} - x_1} q_k.
$$

This implies

$$
q_k = \frac{y_k - x_1}{x_k - x_1} \frac{y_{k-1} - x_1}{x_{k-1} - x_1} \cdots \frac{y_2 - x_1}{x_2 - x_1}
$$

so that

$$
p_1(T) = q_n = \frac{y_n - x_1}{x_n - x_1} \frac{y_{n-1} - x_1}{x_{n-1} - x_1} \cdots \frac{y_2 - x_1}{x_2 - x_1}.
$$

In a symmetric fashion we get

$$
p_n(T) = \frac{x_n - y_2}{x_n - x_1} \frac{x_n - y_3}{x_n - x_2} \cdots \frac{x_n - y_n}{x_n - x_{n-1}}.
$$

Finally, using Theorem 3(i) we put (10) and (11) together to obtain:

$$
p_j(T) = \prod_{1 < k < j} \frac{x_j - y_k}{x_j - x_{k-1}} \prod_{j < k \leq n} \frac{y_k - x_j}{x_k - x_j}.
$$

Obviously, $p_j(T)$ can be computed in linear time for each index $j$. Thus, the entire vector $p(T)$ can be computed in $o(n^2)$ time.

We can now return to the examples presented in §2 and examine them in detail.

**Example 1.** ($y_1 = 0, x_1 = 1, y_2 = 5, x_2 = 6, y_3 = 10$). The two minima here may look symmetric, as $T = (1, 4, 1, 4)$. However, by Corollary 3.1, we can replace $t_1$ and $t_4$ by 0 so that we can take $T = (0, 4, 1, 0)$. This is obviously a situation biased in favor of $x_1$. In fact, using (7) and (8) we get

$$
p_1 = \frac{y_2 - x_1}{x_2 - x_1} = \frac{t_2}{t_1 + t_2} = \frac{4}{5}, \quad p_2 = \frac{x_2 - y_2}{x_2 - x_1} = \frac{t_3}{t_1 + t_2} = \frac{1}{5}.
$$
Example 2. \((y_1 = 0, x_1 = 1, y_2 = 3, x_2 = 5, y_3 = 9)\). This is, in one sense, an opposite example to Example 1. On first glance it appears that \(x_1\) is more likely than \(x_2\) as \(T = (1, 2, 2, 4)\). However, using Corollary 3.1 we take \(T = (0, 1, 1, 0)\) which yields \(p_1 = p_2 = 1/2\).

Example 3. \((y_1 = 0, x_1 = 1, y_2 = 2, x_2 = 3, y_3 = 4, x_3 = 5, y_4 = 6)\). Here \(T = (1, 1, 1, 1, 1, 1)\) which is reduced to \((0, 1, 1, 1, 1, 1)\). Although the function is symmetric, the probabilities are not equal. In fact, it is easy to obtain:

\[
p_1 = p_3 = \frac{1}{2} \times \frac{2}{4} = \frac{1}{4}, \quad p_2 = \frac{1}{2} \times \frac{1}{2} = \frac{1}{4}.
\]

The general symmetric case is handled in §6.

Example 4. For \(M\) equals 1, this example is identical to Example 3. For the general case we get from (10), (11) and (12):

\[
P_1 = \frac{M + 2}{(M + 1)(M + 3)}, \quad P_2 = \frac{M + 2}{2(M + 3)}, \quad P_3 = \frac{M}{2(M + 1)}.
\]

This example is reconsidered at the end of §5.

5. Properties. In this section we investigate some properties of the vector \(p\) which are directly relevant to the performance of random binary search in practice. Clearly, what we care about when the algorithm is actually applied is the probability \(q = q(f, u)\) of picking the global optimum in one iteration as this probability is inversely related to the expected number of iterations required before that optimum is generated. We take the typical approach used in worst case analysis, i.e., we assume that the algorithm is applied to the “worst” instance in a certain class \(C\) of problem instances and seek to assess a bound \(q(C)\) on \(q(f, u)\) which is valid for all instances in the class \(C\). Note that \(p\) depends on the vector \(T\) only, but that for a given \(T\) one can easily construct a function whose global minimum is any of the local minimum \(x_i, \ i = 1, \ldots, n\). Thus, for each vector \(T\) the worst choice of \(f\) is such that \(q = \min_{i=1, \ldots, n} p_i(T)\). An obvious class to examine is the class of functions with \(n\) or less local minima over the appropriate interval. Clearly in this case \(q(C) \ll 1/n\) but as one would expect, it is not possible to derive a lower bound of \(q\) in terms of \(n\). This is demonstrated by the example of Figure 7 for the case \(n = 3\). Here \(T = (0, 1, \epsilon, \epsilon, 1, 0)\), \(p_2 = (\epsilon/(1 + \epsilon))^2\) so that \(q\) can be made arbitrarily small by choosing a small \(\epsilon\). A more useful characterization of \(C\) is in terms of the minimal (normalized) distance between adjacent minima and maxima

\[
\delta_0 = \frac{1}{|u|} \left\{ \min_{i=1, \ldots, n} \min \{r_i, l_i\} \right\}.
\]

![Figure 7](image-url)
or the size of the smallest "valley" in \(u\),

\[
\delta_i = \frac{1}{|u|} \left( \min_{i=1, \ldots, n} \{ r_i + l_i \} \right).
\]

Ideally, one would have liked the probability of sampling a given minimum to equal the size of the appropriate "valley" as this would have meant that each subinterval of \(u\) is being sampled proportionally to its length and independently of its location inside \(u\). This is, however, not the case as for a given size of \(\delta_0\) or \(\delta_1\), \(q\) can be arbitrarily small. (Examine Figure 7 again and note that \(\delta_0 = \epsilon/(1 + \epsilon)\) and \(\delta_1 = \epsilon/(1 + \epsilon)\).) However, a quadratic lower bound on \(q\) in terms of \(\delta_0\) (but not \(\delta_1\)) can be obtained:

**Theorem 5.** \(q \geq 4\delta_0^2\).

**Proof.** For any vector \(T\) and any minimum \(x_i\), \(p_i(T)\) is at least equal to the probability that the first \(z\) to the right of \(x_i\) is in \(l_i\) and the first \(z\) to its left is in \(l_i\).

Thus,

\[
p_i(T) \geq \frac{l_i \cdot r_i}{(x_i - a)(b - x_i)} \geq 4\delta_0^2.
\]

Note that the bound of Theorem 5 is actually attained by the function of Figure 7. The quadratic behavior means that, at least for that function, some regions of \(u\) are undersampled relative to their size and consequently other regions are oversampled. In the next section we study this phenomenon more closely.

We conclude this section with a certain desirable and intuitive monotonicity result. Consider a vector \(T'\) obtained from \(T\) by the relation \(l_i = l_i', i = 1, \ldots, n, r_i = r_i', i = 1, \ldots, n, i \neq j, r_j < r_j', i.e., T' is identical to T except that the "valley" surrounding \(x_i\) in \(T'\) is larger to the right. Surely, one expects \(p_i(T) \leq p_i(T')\). This seems hard to show directly from (4) or (5), but a more general result falls easily out of (12).

**Theorem 6.** \(p_i(T) \leq p_i(T')\) for \(i = 1, \ldots, f\).

**Proof.** Direct from formula (12). \(\blacksquare\)

Note that the theorem does not make any assertion with respect to the relation between \(p_k(T)\) and \(p_k(T')\) for \(k > j\).

To illustrate Theorem 6, reconsider Example 4. The stipulations of the theorem correspond to increasing the value of \(M\). For \(M \to 0\), we have \(p \to (2/3, 0, 2/3)\) while, for \(M \to \infty\), \(p \to (0, 1/2, 1/2)\). The theorem asserts that as \(M\) scans the interval \((0, \infty)\), \(p_2\) and \(p_3\) increase monotonically from their initial values 0 and 1/3 to their final values 1/2 and 1/2.

**6. The symmetric case.** The probability \(p_i(T)\) of sampling \(x_i\) depends on the relative sizes of the different "valleys" surrounding the local minima and on the "position" of \(x_i\) inside \(u\). In this section we study the second effect by concentrating on completely symmetric vectors \(T\), namely \(t_i = 1, i = 1, \ldots, 2n\). In spite of the symmetry in sizes, we will shortly discover that the various local minima have very different probabilities of being sampled.

Let \(T_n\) be the vector of \(2n\) ones. Let \(g_n(i) = p_i(T_n)\). We wish to study this function. Let \(h(n) = p(T_n) = g_n(1)\). From Theorem 3 we have \(g_n(i) = h(i) \cdot h(n - i + 1)\). Thus, it is enough to study the function \(h(\cdot)\). From (9) we get the following simple recursion relation

\[
h(n + 1) = \frac{2n - 1}{2n} h(n).
\]
Clearly, \( h(1) = 1 \). Thus

\[
h(n) = \frac{1}{2} \times \frac{1}{3} \times \frac{1}{4} \times \cdots \times \frac{2n - 3}{2n - 2}.
\]

We now wish to find a closed form approximation for \( h(n) \). Consider the function \( f(n) = 1/\sqrt{n} \). Using the first term in the Taylor expansion of this function we get

\[
\frac{1}{\sqrt{n + 1}} = \frac{1}{\sqrt{n}} - \frac{1}{2n\sqrt{n}} + o\left(\frac{1}{n^2\sqrt{n}}\right) = \frac{1}{\sqrt{n}} \left(\frac{2n - 1}{2n} + o\left(\frac{1}{n^2}\right)\right),
\]

that is

\[
f(n + 1) = \left(\frac{2n - 1}{2n} + o\left(\frac{1}{n^2}\right)\right)f(n),
\]

with \( f(1) = 1 \).

Comparing the recursions for \( h \) and \( f \) we observe that for large values of \( n \), \( h(n) \) behaves approximately like \( 1/\sqrt{n} \) and thus

\[
g_n(i) \approx \frac{c}{\sqrt{i\sqrt{n} - i + 1}}. \quad (14)
\]

The actual values \( g_n(i) \) together with their approximation \( c/\sqrt{i\sqrt{n} - i + 1} \) normalized to sum to 1 for \( n = 30 \) are shown in Figure 8. (The constant \( c \) was chosen so that the approximate probabilities sum to one.)

For an asymptotically large \( n \), we can consider the limiting behavior of the probabilities \( p_i(T_n) \). Consider the following process defined on the interval \( u = u_i = [0,1] \). At each step, pick a point \( z_k \) uniformly distributed over \( u_k = [a_k, b_k] \) and let \( u_{k+1} \) be \([a_k, z_k]\) or \([z_k, b_k]\) with probability \( 1/2 \), respectively. It can be shown that the process \( z = (z_1, z_2, \ldots) \) converges to a point \( z \in [0,1] \) with probability one and that the density of the limit point \( z \) is given by

\[
f(z) = \frac{1}{\sqrt{z\sqrt{1-z}}}, \quad 0 \leq z \leq 1. \quad (15)
\]

Clearly (15) is in agreement with (14). The implication of (14) or (15) is clear. Random binary search tends to oversample at the sides of the interval \( u \) and to undersample at

![Figure 8. \( g_n(i) \) for \( n = 30 \) (△) and Its Approximation \( \frac{c}{\sqrt{i\sqrt{n} - i}} \) (+).](image)
the center. This may be helpful if the function involved is likely to have its global optimum at the edges, but, in general, this bias is bothersome. To offset it (but only partially) one can replace the uniform distribution over $u_k$ by another distribution, heavier at the center and lighter at the tails. It would be interesting to find out whether there exists a distribution of $z_i$ over $u_i$ which would make $g_n(i) = 1/n$ for $i = 1, \ldots, n$. I conjecture that this is not possible.

7. Adaptive search. The simplest way to search the interval $u$ using randomizing binary search is to apply each iteration independently to the interval $u$. This means that the information generated during the execution of the previous iterations is completely ignored. However, one can expect to do better if this information is somehow used to guide the search. For instance, say that the minimum located in the first iteration is at $x^*$. One would certainly like to bias the search in subsequent iterations so that the likelihood of $x^*$ being selected again is decreased. Naturally, the question arises whether this can be achieved. A simple way to try is the following. Split $u$ at $x^*$ and consider the two subintervals $ur(x^*)$ and $ul(x^*)$. Put these two intervals in a candidate list and attach a probability to each. At each iteration, pick an interval from the list according to its probability and apply random binary search to this interval. Then, split the subinterval at the new minimum found, modify the probabilities, and repeat the process. Leaving aside momentarily the question of how one assigns these probabilities, we examine below the question of whether this approach can be advantageous. Theorem 3 supplies the necessary vehicle for analyzing this issue.

Let $p + q = 1$, and $p, q > 0$. Assume we chose $ur$ with probability $p$ and $ul$ with probability $q$ and apply the search to the interval chosen. Let $\alpha = \alpha(p)$ be the probability of sampling $x^*$ with this strategy. The question is whether we can pick $p$ so that $\alpha(p) < p_i(T)$ where $x^* = x_i$. This is answered in the negative by

**Theorem 8.** \( \alpha(p) \geq p_i(T) \) for every \( 0 \leq p \leq 1 \).

**Proof.**

\[
\alpha(p) = q \cdot p_i(TR) + \alpha(1 - q) \geq \min\{p_i(TR), p_i(TL)\}
\]

\[
\geq p_i(TR) \cdot p_i(TL) = p_i(T).
\]

The question of whether one can devise other strategies for adaptive search which will increase the probabilities of sampling new minima at each iteration is a central issue in stochastic global optimization (cf. [4, 9, 13]). It is still an open issue whether this can be done for the case of randomizing binary search.

**Acknowledgement.** I wish to thank Isaac Meilijson for a very interesting discussion concerning the solution (15) to the asymptotic process discussed at the end of §6, and to Colin McDiarmid and the two referees for some very useful remarks concerning an earlier version of this manuscript.

**References**


J. L. KELLOGG GRADUATE SCHOOL OF MANAGEMENT, NORTHWESTERN UNIVERSITY, EVANSTON, ILLINOIS 60201
LEON RECANATI SCHOOL OF BUSINESS ADMINISTRATION, TEL AVIV UNIVERSITY, TEL AVIV, ISRAEL