A Depth-First Lipschitz Based Global Maximization Algorithm[1]

by

Rua Murray and **Chris Stephens**
Department of Mathematics and Statistics,
University of Canterbury, Christchurch, New Zealand

No. 102 September, 1993

Abstract – A depth-first analog to the Lipschitz based Piyavskii-Shubert global maximization algorithm is presented. To within any given tolerance, the algorithm is shown to return a global maximum and maximizer for a uni-variate Lipschitz continuous function. This result is extended to a broader class of uni-variate functions. Empirical comparisons of Piyavskii-Shubert with several variations of the new algorithm are made.

[1] This paper is the result of research undertaken as part of MATH 457: Topic X.

A depth–first Lipschitz based global maximization algorithm*

Rua Murray Chris Stephens

September 24, 1993

Abstract

A depth–first analog to the Lipschitz based Piyavskii–Shubert global maximization algorithm is presented. To within any given tolerance, the algorithm is shown to return a global maximum and maximizer for a uni–variate Lipschitz continuous function. This result is extended to a broader class of uni–variate functions. Empirical comparisons of Piyavskii–Shubert with several variations of the new algorithm are made.

1 Introduction

This paper is concerned with the problem of estimating and locating the *global maximum* of a real–valued function on a closed interval.

Definition 1.1 *For $\epsilon > 0$, a* global maximum for f to within ϵ, *is a value y^* of f on $[a,b]$ such that*

$$y^* > \max_{[a,b]} f - \epsilon.$$

A value $x^* \in [a,b]$ for which $y^* = f(x^*)$ will be called a maximizer to within ϵ.

We seek an algorithm which will return, for any given ϵ, a global maximum for f to within ϵ, and a corresponding maximizer. When dealing with functions which are not necessarily continuous, it is common to replace max with *essential supremum* in this definition. It is assumed that the objective function f is not explicitly known to the algorithm, but that the function is treated as a *black–box* from which information about the function can be obtained by sampling. It is for this reason that we restrict attention to finding a global maximum to within ϵ, since the only way to guarantee to find an exact global maximum (if such exists) is to know the value assumed by f at each point of the domain; namely, to have explicit knowledge of the function. In fact, no algorithm can *guarantee* to return a global maximum for a totally general function; there are many references to this in the literature, see for instance [3, pp 82-84] or [8]. It is therefore necessary to restrict the class of functions for which any specific algorithm is guaranteed to be correct.

The Piyavskii-Shubert algorithm (see Shubert [7] or Piyavskii [6]) is a global maximization algorithm for Lipschitz continuous functions with Lipschitz constant L. As specified by Shubert, this algorithm is of a width–first variety. Width–first algorithms inherently have large memory usage in comparison with their depth–first analogs, and it is often the case that no more work is required, on average, for a depth–first analog to a width–first algorithm. Note, the measure of *work* in this paper is taken to be the number of function evaluations.

In section 2 we present a simple *depth–first* Lipschitz–based global maximization algorithm. Some subtleties are included in section 3 to give the new algorithm. It is shown that the memory requirements of the new algorithm are much less than required for the Piyavskii–Shubert algorithm. A result concerning the robustness of the new algorithm under inaccuracies in the user–supplied Lipschitz constant is also proved. Finally, in section 4, empirical comparisons of several variations of the new algorithm are made with Piyavskii–Shubert.

Note, it is assumed throughout this paper that $f : [a,b] \to R$ is Lipschitz continuous with Lipschitz constant L.

*This paper is the result of research undertaken as part of MATH 457: Topic X.

2 Conceptual Algorithm

We begin with a discussion of the conceptual foundation of our algorithm, and present a basic depth–first algorithm.

The following lemma (see figure 1) provides the basis for an algorithm's stopping criterion. Namely on an interval of length less than $\frac{2\epsilon}{L}$, no function value in the interval can exceed a bound on an endpoint by ϵ or more. For example, a grid search of $\lceil \frac{(b-a)L}{2\epsilon} \rceil$ uniformly spaced points over $[a, b]$, is sufficient to give a global maximizer and maximum to within ϵ.

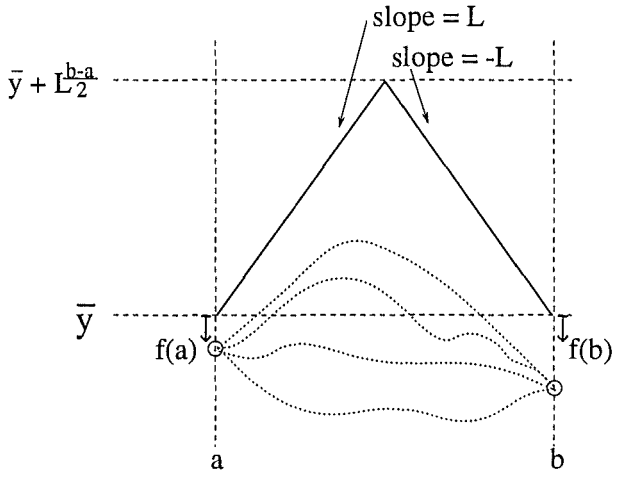


Figure 1 : Diagram showing how a Lipschitz constant L can provide an upper bound for a function on $[a, b]$ when $f(a)$ and $f(b)$ are known to be bounded above by $\bar{y}$; this is formalized by Lemma 2.1. Some possible functions f are illustrated.

Lemma 2.1 *Let $f : [a, b] \rightarrow R$ be Lipschitz continuous with Lipschitz constant L, and suppose that $f(a)$ and $f(b)$ are bounded above by $\bar{y}$. Then for all $x \in [a, b]$,*

$$f(x) \leq \bar{y} + L\frac{b-a}{2}.$$

Proof: It follows immediately from Lipschitz continuity that for $x \in [a, b]$,

$$f(x) \leq \bar{y} + L(x-a) \quad , \tag{1}$$
$$f(x) \leq \bar{y} + L(b-x) \quad . \tag{2}$$

Adding inequalities (1) and (2), and dividing by 2 yields the result. ■

The second lemma allows regions of the domain where a global–maximum cannot possibly lie to be excluded from the search; see Figure 2.

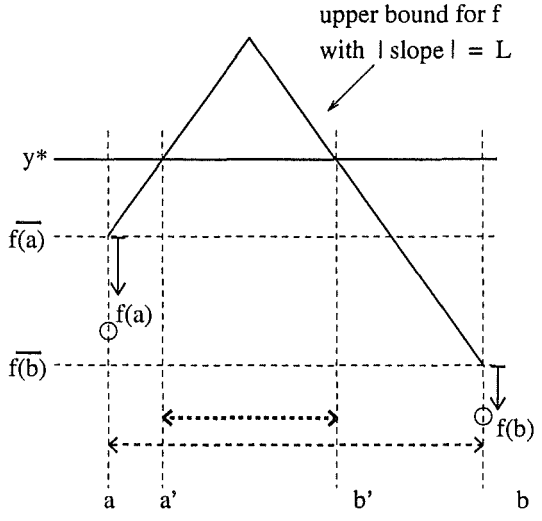


Figure 2 : Diagram of interval reduction justification. When bounds on a Lipschitz continuous function f are known at the endpoints of $[a, b]$, any value of f in that interval which exceeds some y^* must lie in $[a', b']$, as specified in Lemma 2.2 below.

Lemma 2.2 *Let $f : [a,b] \to R$ be Lipschitz continuous with Lipschitz constant L, and suppose that $f(a)$ and $f(b)$ are bounded above by $\bar{f}(a)$ and $\bar{f}(b)$ respectively. Suppose furthermore that $y^* \geq \max\{\bar{f}(a), \bar{f}(b)\}$. Then for all $x \in [a,b]$ such that $f(x) \geq y^*$,*

$$a' = \left(a + \frac{y^* - \bar{f}(a)}{L}\right) \leq x \leq \left(b - \frac{y^* - \bar{f}(b)}{L}\right) = b'. \tag{3}$$

Proof: It follows from (1) with $\bar{f}(a)$ replacing y^* that $f(x) < y^*$ for $x \in [a, a')$. Similarly, replacing y^* with $\bar{f}(b)$ in (2) gives $f(x) < y^*$ for $x \in (b', b]$. Thus, $\{x \in [a,b] : f(x) \geq y^*\} \subseteq [a', b']$. ■

The interval reduction $[a,b] \to [a',b']$ described in the above Lemma is used in our depth–first variant of the Piyavskii–Shubert algorithm. However, even the simple grid search described above can be improved using this result; see for instance Evtushenko [4].

In fact, a number of algorithms have been developed that basically employ these two lemmas; these tend to proceed by the construction of *saw–tooth covers*. The Evtushenko and Piyavskii–Shubert algorithms are of this form. A comparison of the performance of this class of algorithms is provided by Hansen et. al. [5]. Our algorithm is effectively also of this variety.

The following is a simple *divide and conquer* algorithm for globally maximizing f on [a,b]. The idea is to use the known Lipschitz constant to construct an upper bound for f on $[a,b]$, and to seek a function value which is within ϵ of this bound (this will clearly be a global–maximum to within ϵ for f on $[a,b]$); see figure 3.

Algorithm 2.3 (Conceptual depth–first) *Given $f : [a,b] \to R$, $f(a)$, $f(b)$ and a maximum–so–far, $y^* = f(x^*)$, then*

1. Let $P = (P_x, P_y)$ be the intersection of the line through $(a, f(a))$ with slope L and the line through $(b, f(b))$ with slope $-L$; see figure 3.
   Stop if $P_y < y^* + \epsilon$, and return y^* and the corresponding domain value (x^*).
2. Evaluate f at P_x and update (x^*, y^*) if $f(P_x) > y^*$.
3. Repeat on each of the subintervals $[a, P_x]$ and $[P_x, b]$; and return the greater of the maxima obtained and the corresponding domain value.

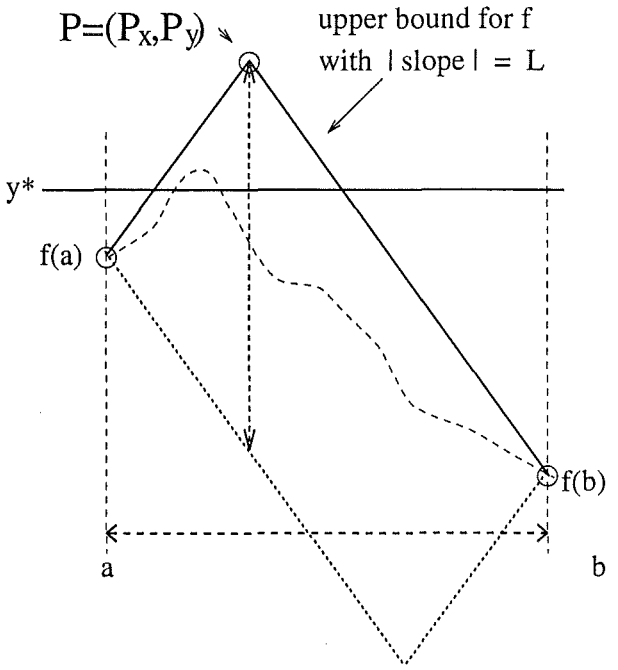


Figure 3 : Diagram of region in which Lipschitz continuous $f : [a,b] \to R$ can lie. By Lemma 2.1 P_y is the largest value that f could possibly assume on $[a,b]$. Clearly, only values of f greater than y^* are of interest in the sequential search for $\max_{[a,b]} f$. The vertical arrow denotes the range in which $f(P_x)$ could lie.

The first step of Algorithm 2.3 constructs an upper envelope for f on $[a,b]$ using the known Lipschitz constant (see Figure 3). The validity of the stopping criterion is seen by

applying Lemma 2.2 to $[a,b]$ with $\bar{f}(a) = f(a)$ and $\bar{f}(b) = f(b)$, and then Lemma 2.1 to the $[a',b']$ thus obtained.

To understand the convergence of the algorithm, let P_l and P_r be the points described in step one of Algorithm 2.3 applied to $[a, P_x]$ and $[P_x, b]$ respectively. Let $y^{*\prime} = \max\{y^*, f(P_x)\}$ be the maximum–so–far for each of these subintervals. Then $P_{l_y} = P_{r_y}$, and

$$P_{l_y} - y^{*\prime} = \frac{(P_y - y^*) - |f(P_x) - y^*|}{2} \leq \frac{P_y - y^*}{2}. \tag{4}$$

Algorithm 2.3 can be seen to converge geometrically in view of (4). Furthermore, the equality part of (4) implies that $|f(P_x) - y^*|$ close to $(P_y - y^*)$ gives a large reduction in the difference between P_y (the upper bound for f on $[a,b]$) and the best largest known function value y^*, and hence speeds convergence. This corresponds to $f(P_x)$ either "very high" or "very low", relative to its possible range.

A very high function evaluation at some stage brings the maximum–so–far closer to a maximum for f. But also, a very low function evaluation at some stage in some reduces the potential maximum, hence bringing the maximum of f closer to the maximum–so–far.

3 The Algorithm

The conceptual algorithm described above makes an arbitrary choice as to whether it recurses upon the left subinterval or the right subinterval first. This could possibly make a large difference to the number of function evaluations required. The following algorithm, written in a pseudo-computer language uses a one–step look–ahead to allow a more informed choice as to which side to recurse on. Also, the explicit form of Algorithm 3.1 is that of *interval length reduction*. To this end, rather than passing the actual function evaluation points, a reduced interval and upper bound on the end–points are passed.

Algorithm 3.1 (Depth–first Piyavskii–Shubert) *Let $f : [a,b] \to R$ be Lipschitz continuous with Lipschitz constant L. Let $\epsilon > 0$, $y^* = f(x^*)$ be a maximum so far and let $\bar{y}$ be an upper bound for $f(a)$ and $f(b)$.*

`define function global-max(`f`,`$[a,b]$`,`$\bar{y}$`, (`$\frac{a+b}{2}, f(\frac{a+b}{2})$`), (`$x^*, y^*$`),`$L,\epsilon$`)`

$a_l \leftarrow a + \frac{y^* - \bar{y}}{L}$;

$b_l \leftarrow \frac{a+b}{2} - \frac{y^* - f(\frac{a+b}{2})}{L}$;

$a_r \leftarrow \frac{a+b}{2} + \frac{y^* - f(\frac{a+b}{2})}{L}$;

$b_r \leftarrow b - \frac{y^* - \bar{y}}{L}$;

Assertion (1) : If $x \in [a,b]$ and $f(x) \geq y^*$, then $x \in [a_l, b_l] \cup [a_r, b_r]$.

Assertion (2) : y^* is an upper bound for f on $\{a_l, b_l, a_r, b_r\}$.

`if` $b_l - a_l < \frac{2\epsilon}{L}$,

Assertion (3) : If $x \in [a_l, b_l] \cup [a_r, b_r]$, then $f(x) < y^* + \epsilon$.

`return` (x^*, y^*) ;

`else`

$\bar{y} \leftarrow y^*$;

$y^* \leftarrow \max\{f(\frac{a_l+b_l}{2}), f(\frac{a_r+b_r}{2}), y^*\}$;

$x^* \leftarrow$`argmax`$\{f(\frac{a_l+b_l}{2}), f(\frac{a_r+b_r}{2}), y^*\}$;

(x^*, y^*) = `global-max(`f`,`$[a_l,b_l]$`,`$\bar{y}$`, (`$\frac{a_l+b_l}{2}, f(\frac{a_l+b_l}{2})$`), (`$x^*, y^*$`),`$L,\epsilon$`);`

`return global-max(`f`, `$[a_r,b_r]$`,`$\bar{y}$`, (`$\frac{a_r+b_r}{2}, f(\frac{a_r+b_r}{2})$`), (`$x^*, y^*$`), `$L,\epsilon$`);`

`end if;`

`end define global-max;`

Note : The *assertions* in the algorithm are to assist understanding of the proof of the algorithm.

Remarks:

1. The interval midpoints ($\frac{a+b}{2}$) and function values there are passed as parameters to prevent re-evaluation there after the recursive call; the conceptual algorithm 2.3 does not explicitly utilize this economy.
2. A consistent interchange of the roles of l and r can be made throughout the algorithm.
3. Because the heights of the triangles above the reduced subintervals are equal, it will normally be the case that function evaluations on both the left and right subintervals will be required. Thus, we can bring forward those function evaluations and use this information to make a more informed decision as to which subinterval to do first.

   Observe however that the recursive call in algorithm 3.1 is expicitly on the *left* then *right* subintervals; this is for notational convenience only, and *choosing-rules* will generally be employed to determine which recursion to do first.
4. If $f(\frac{a_l+b_l}{2})$ is within ϵ of the peak it would be unnecessary to evaluate $f(\frac{a_r+b_r}{2})$. An additional check may be added to prevent this second evaluation occurring. However, an implementation with this addition made no difference to the number of function evaluations required in all trials we ran.
5. Refer to Figure 4 for more illumination of the interval reduction.

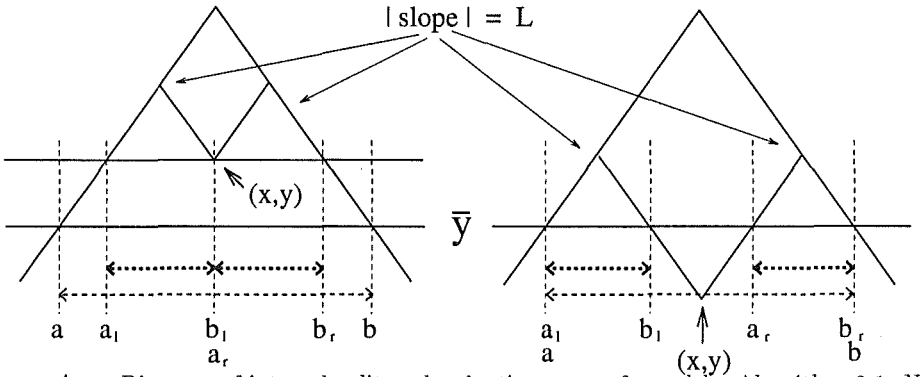


Figure 4 : Diagram of interval split and reduction, as performed by Algorithm 3.1. Note that $\bar{y}$ is a bound on the function at the endpoints of the original interval $[a,b]$, $x = \frac{a+b}{2}$, and the nature of the split to $[a_l,b_l]$ and $[a_r,b_r]$ depends on the value of $y = f(x)$: the left–hand diagram illustrates $y > \bar{y}$; the right–hand diagram illustrates $y < \bar{y}$; and if $y = \bar{y}$, the reduction is trivial (i.e.. $[a,b] = [a_l,b_l] \cup [a_r,b_r]$). The two possible situations illustrate the dual benefit of high and low values of f beneath the peak.

We now prove that Algorithm 3.1, with appropriate conditions on parameters, will halt, returning a global maximum for f to within ϵ, and a corresponding maximizer.

Theorem 3.2 *Let $f : [a,b] \to R$ be Lipschitz continuous with Lipschitz constant L, and let $\epsilon > 0$. Set*

$\bar{y} = \max\{f(a), f(b)\}$,

$a' = a + \frac{\bar{y}-f(a)}{L}$,

$b' = b - \frac{\bar{y}-f(b)}{L}$,

$y^* = \max\{f(a), f(b), f(\frac{a'+b'}{2})\}$,

$x^* =$ *domain value corresponding to* y^* *and* $\epsilon > 0$.

Then `global-max(f, [a',b'], y*, (`$\frac{a'+b'}{2}, f(\frac{a'+b'}{2})$`), (x*,y*), L, ε)` *will halt and return (X,Y) such that $Y = f(X)$ and Y is a global maximum to within ϵ for f.*

Proof: It follows immediately from Lemma 2.2 that $\max\limits_{[a,b]} f$ lies in $[a',b']$. So without loss of generality let $a = a'$ and $b = b'$.

The theorem is clearly implied for all intervals $[a,b]$ if it can be shown to be true for intervals where $b - a < 2^n \frac{2\epsilon}{L}$ for an arbitrary natural number n. The argument is thus inductive.

First, observe that: $y^* \geq \bar{y}$, $y^* \geq f(\frac{a+b}{2})$, $\bar{y} \geq f(a)$ and $\bar{y} \geq f(b)$.

Now, *assertion (1)* in Algorithm 3.1 justifies the splitting of the interval $[a, b]$ into two subintervals, which are then searched recursively.

Proof of assertion (1): Since $\bar{y}$ is an upper bound on $f(a)$ and y^* is an upper bound on $f(\frac{a+b}{2})$ (by the remarks above), Lemma 2.2 applied to $[a, \frac{a+b}{2}]$ shows that for any $x \in [a, \frac{a+b}{2}]$ such that $f(x) \geq y^*$, $x \in [a_l, b_l]$. This, together with a similar argument on $[\frac{a+b}{2}, b]$ proves the assertion.

Proof of assertion (2): Suppose, for example, that $f(a_l) > y^*$. Then by continuity of f, there exists $x < a_l$ such that $f(x) > y^*$; this would contradict *assertion (1)*.

Next, it follows from the remarks regarding y^* and $\bar{y}$ that $a \leq a_l$, $b_l \leq \frac{a+b}{2}$, $\frac{a+b}{2} \leq a_r$ and $b_r \leq b$. Thus,

$$b_l - a_l \leq \frac{b-a}{2} \quad \textit{and} \quad b_r - a_r \leq \frac{b-a}{2}. \tag{5}$$

The inductive basis may now be established. Let $n = 1$. Then, $b - a < \frac{4\epsilon}{L}$, so that by (5), $b_l - a_l \leq \frac{2\epsilon}{L}$, and y^* is returned by Algorithm 3.1. It thus remains to prove that y^* is a maximum to within ϵ for f on $[a, b]$. By *assertion (1)*, only the intervals $[a_l, b_l]$ and $[a_r, b_r]$ need be considered. The basis is now given by *assertion (3)*.

Proof of assertion (3): By *assertion (2)*, y^* is an upper bound for $f(a_l)$ and $f(a_r)$. Thus, by Lemma 2.1, for $x \in [a_l, b_l]$,

$$f(x) \leq y^* + L\frac{b_l - a_l}{2} < y^* + \epsilon.$$

The same argument applies to $[a_r, b_r]$.

Suppose now that for $n = k$, `global-max` returns a global–maximum to within ϵ (and corresponding maximizer) for all intervals of length less than $2^k \frac{2\epsilon}{L}$. Let $[a, b]$ be such that $b - a < 2^{k+1}\frac{2\epsilon}{L}$. Then, setting up as in the hypotheses of the theorem, the interval split is undertaken as normal. By (5), each of these subintervals has length less than $2^k \frac{2\epsilon}{L}$, and by *assertion (2)* each of the endpoints of these subintervals is bounded above by y^*. Hence, the algorithm is called recursively on $[a_l, b_l]$ and $[a_r, b_r]$ with this value as a bound on the interval endpoints. The maximum so far (y^*) is also updated to preserve the relations at the beginning of the proof.

It now follows from the induction hypothesis that `global-max` returns global maxima to within ϵ on each subinterval, so that the result follows for $n = k + 1$.

The theorem now follows by the principle of induction. ■

Corollary 3.3 *Algorithm 3.1 has* $O(\ln \frac{1}{\epsilon})$ *memory requirements. Furthermore,*

$$2^{\lceil \log_2 [L\frac{(b-a)}{2\epsilon}] \rceil} - 2 \tag{6}$$

provides an upper bound on the number of function evaluations required.

Proof: We see that Algorithm 3.1 is a **binary recursion**, and the interval length at each level of the binary tree is at most half of the length of its parent's interval. The algorithm halts when the interval length is less than $\frac{4\epsilon}{L}$. Therefore the depth of the recursion is no more than

$$k = \lceil \log_2 [L\frac{(b-a)}{4\epsilon}] \rceil.$$

It is thus easily seen that halving ϵ increases maximum depth by 1, so that memory requirements are linear in the number of binary significant figures of ϵ.

Secondly, there are two function evaluations at all nodes of the tree except the leaves. Since the tree has maximum depth k (as above), the tree contains at most $2^k - 1$ nodes that are not leaves. The maximum number of function evaluations required is $2^{k+1} - 2$. ■

Observe that the number of function evaluations required is in fact three more than that given by equation (6). This is because the hypotheses of Theorem 3.2 include knowledge of three function values.

We have shown that Algorithm 3.1 does indeed return a global maximizer and global maximum to with ϵ in a finite number of function evaluations, but this does not show the

true power of the algorithm. While for a constant function, the bounds on work and memory requirements are reached because the search interval reduction described by Lemma 2.2 is always trivial, Algorithm 3.1 reduces the search interval non-trivially in other cases. This means that the number of function evaluations required is often less than that for a uniform grid search. This is investigated further in the empirical section of this paper.

It should be noted that **no** algorithm can *guarantee* the global maximum to within ϵ with less than $\lceil \frac{(b-a)L}{2\epsilon} \rceil$ function evaluations, given only the Lipschitz constant, L, for the constant function. To see this, one only needs to see that if the algorithm has probed less than $\lceil \frac{(b-a)L}{2\epsilon} \rceil$ times , then there is at least one interval of length greater than $\frac{2\epsilon}{L}$ and hence from Lemma 2.1 the algorithm has not *guaranteed* that it has found the global maximum to within ϵ. Hence, this is precisely the same bound that can most generally be given for the Piyavskii–Shubert algorithm.

The second corollary shows that Algorithm 3.1 will halt for any bounded function g, but the *accuracy* of the "global–maximum" returned will depend on the properties of g.

Corollary 3.4 *Let $f : [a,b] \rightarrow R$ be Lipschitz continuous with Lipschitz constant L. If $g : [a,b] \rightarrow R$ is such that for $\delta > 0$,*

$$f(x) \leq g(x) \quad \forall x \in [a,b] \,, \tag{7}$$

and

$$g(x^*) < f(x^*) + \delta \tag{8}$$

where x^* is the global maximizer for g. Then **`global-max`** *called on g with parameters as in Theorem 3.2 (except that g is not necessarily Lipschitz continuous but L is given as above), halts and returns (X,Y) such that $Y = g(X)$ and Y is a global maximum for g to within $\delta + \epsilon$.*

Proof: Consider the proof of Theorem 3.2. However, replace g by f in the assertions (all function evaluations are still of g). Now all deductions in the proof follow through, but the algorithm returns with (X,Y) such that $Y = g(X)$, and $Y > f(x) - \epsilon$ for all $x \in [a,b]$. But $x^* \in [a,b]$, so that by (8) and definition of x^*,

$$\begin{aligned} Y &> f(x^*) - \epsilon \\ &> g(x^*) - (\epsilon + \delta) \\ &\geq g(x) - (\epsilon + \delta) \quad \forall x \in [a,b] \,. \end{aligned}$$

■

This result is advantageous, since it dispenses not only with the requirement that a function g be Lipschitz continuous with a known Lipschitz constant, but with the requirement that g be continuous at all. Provided that a function $g : [a,b] \rightarrow R$ can be bounded below on $[a,b]$ by a Lipschitz continuous function f with a known constant which is within δ of g at the true global maximum of g, a global maximum for g to within $\delta + \epsilon$ can still be obtained by `global-max`, *without even knowing what the continuous function is*. This is actually the best that it seems reasonable to expect of any algorithm, since equation (8) allows g to possess discontinuities of up to δ.

In fact, for functions f and g as above, the function f can be thought of as a *cutter* for g, as in Baritompa [1].

Furthermore, the above corollary demonstrates that Algorithm 3.1 will not fail if an incorrect Lipschitz constant is given, it will merely result in a loss of accuracy: for provided g is bounded, given any $L > 0$, there will always exist some Lipschitz continuous function of Lipschitz constant L, and $\delta > 0$ such that (7) and (8) hold.

4 Empirical results

As mentioned in section 1, our chosen measure of the amount of work an algorithm is required to do is the number of evaluations it makes of the objective function. It is generally

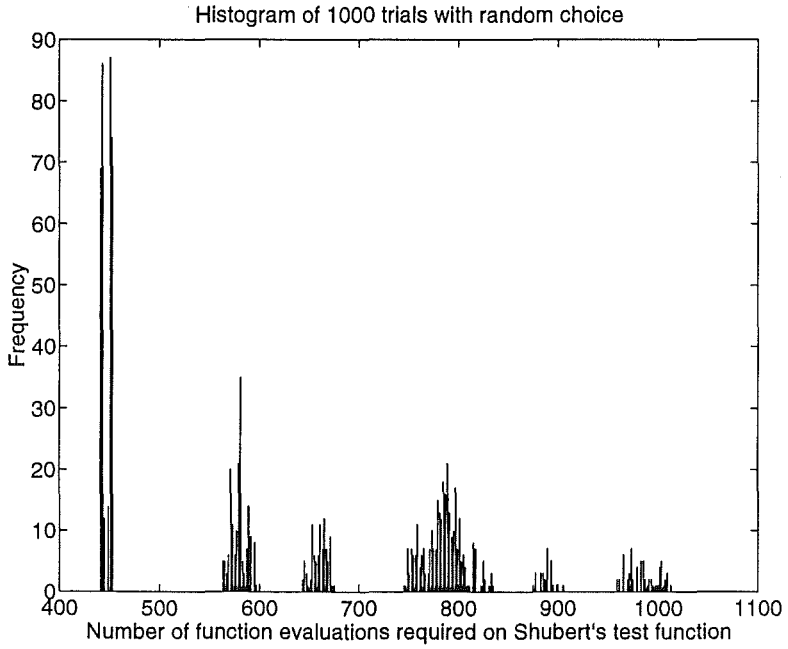
assumed that this is a reasonable measure of work. As such, it is the number of function evaluations required which forms the basis of our comparison of depth–first and width–first algorithms.

The first observation is that for an arbitrary Lipschitz continuous function with Lipschitz constant L, the least upper bound which can be given on the number of function evaluations required for a Lipschitz based Piyavskii–Shubert algorithm is simply that given in equation 6. For this reason, our algorithm will perform no worse than the width–first scheme described in [7] in the worst case. It is therefore relevant to examine how the algorithm's compare in the "better" cases.

An example of a good case was given by Shubert:

$$f(x) = \sum_{k=1}^{5} k \sin((k+1)x + k)$$

The width–first implementation he cited sought a global maximum to within 0.01, and when supplied with a Lipschitz constant of 70 on the interval $[-10, 10]$, returned an answer in 444 function evaluations [7, p387]. (The number of function evaluations required by different implementations of Shubert's scheme may differ by a small amount due to arbitrary choices made in deciding where to evaluate next when there are two or more global maximizers of the envelope function, and due to rounding errors.) However, the depth–first implementation produced, on this function, some interesting results. On this function, we ran Algorithm 3.1, 1000 times, each time making a random choice as to whether to solve the left or right sub–problem first. This yielded a median of 591 function evaluations, with the smallest number required coinciding at 441. See the below plot:



This picture shows that the number of function evaluations required by depth–first Piyavskii–Shubert is dependent on the left/right choices which are made.

It can be shown that on most functions, depth-first Piyavskii–Shubert will evaluate a super–set of the points evaluated by Piyavskii–Shubert, except for possibly a few points, (see Figure 5). These exceptional points are due to the possibility of having two or more peaks of the saw–tooth envelope at the same height; sometimes the selection of one such peak for first function evaluation will render it unnecessary to evaluate under the other peaks, hence the arbitrary choice made by any particular implementation of Piyavskii–Shubert cannot generally be dismissed as ineffectual. Indeed, it is possible to construct functions on which the number of these exceptional points is arbitrarily large, but in most cases this difference is insignificant.

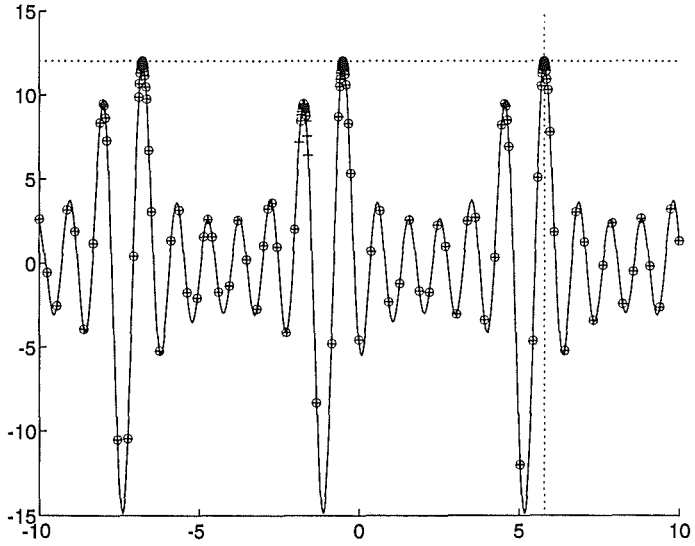


Figure 5 : 571 function evaluations were required for depth–first and 441 for width–first maximization routines on this function. Evaluation points marked by + and o respectively. Note that depth–first evaluated at many points where width–first did not evaluate.

To compare the performance of Depth–First Piyavskii-Shubert *DFPS* to Piyavskii-Shubert *PS* (and also to evaluate which choosing rule is the best in an average case) we ran the algorithms on randomly chosen splines. The cubic splines were created by choosing 53 random range values, uniformly distributed between 0 and 1 on a uniform domain grid between -.02 and 1.02, and using the not–a–knot strategy to deal with the endpoints; see Beatson and Chacko [2]. The trials were run over the interval $[0, 1]$ (the additional points at -.02 and 1.02 were to eliminate the atypical behavior that cubic splines with not–a–knot conditions exhibit on the outer sections of a uniform mesh). A theoretical Lipschitz constant of 50×1.73205 was used, this was obtained from Beatson and Chako's analysis of the derivatives of such splines.

Below are the results of 1000 trials with ϵ chosen to be .01. Algorithm 3.1 was run on each function, with three different choosing rules.

1. Arbitrary ($DFPS_R$): Randomly choose which side to recurse on first.
2. Highest ($DFPS_H$): Always recurse first on the subinterval which has the highest function evaluation at its mid–point.
3. Lowest ($DFPS_L$): Always recurse first on the subinterval which has the lowest function evaluation at its mid–point.

Note that the function evaluations performed before Algorithm 3.1 recurses are precisely the midpoints of the two subintervals. It is thus easy for a decision to be made as to which subinterval to recurse upon first. The purpose of these options is to try and determine whether there is some preferred choosing rule. The Piyavskii-Shubert algorithm was also run on each function.

Since the explicit behavior of each function will determine the choosing–rule which is optimal for that function, we have sought to remove this dependence by expressing the results as ratios of the number of function evaluations required, to the theoritical minimum number required. This minimum was computed using the algorithm described in [5]. The data in the last column are for the uniform grid search (described earlier) which guarantees to find the global maximum to within ϵ when supplied with the same Lipschitz constantas used in our trials; this is called by Hansen et. al. [5] and others, the *passive* algorithm.

Method	PS	$DFPS_R$	$DFPS_H$	$DFPS_L$	$PASSIVE$
Mean	1.4423	1.8710	1.8588	1.9040	30.8189
Standard Deviation	0.0347	0.3845	0.3907	0.3947	5.0971

Figures 6 show the distributions of the above data.

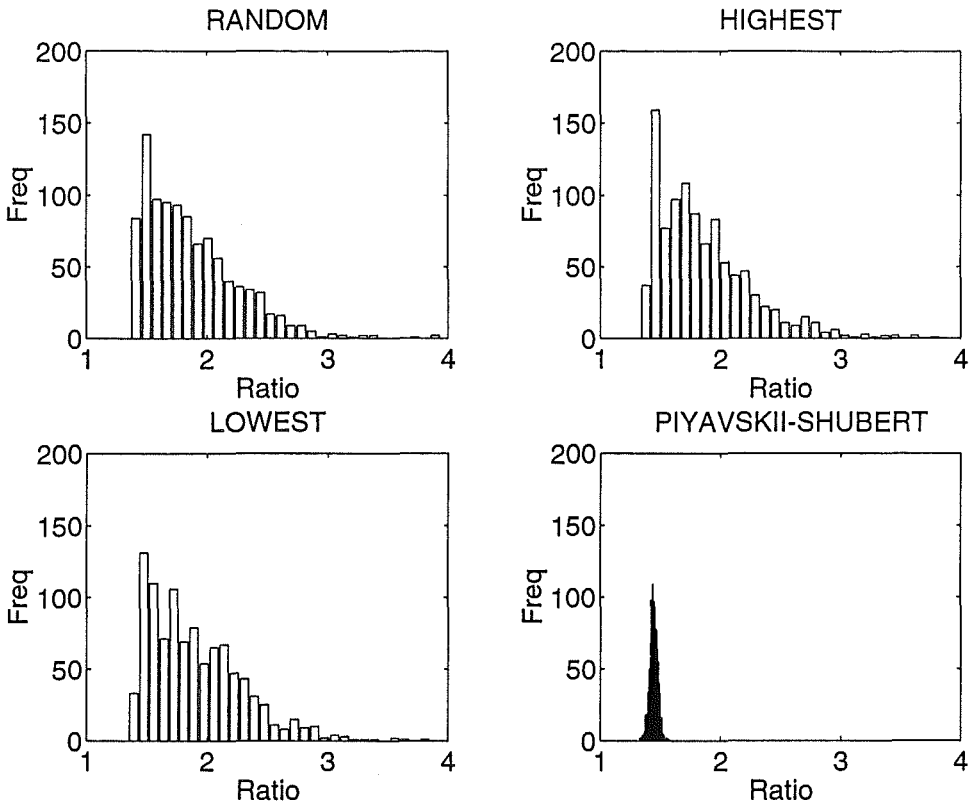


Figure 6 : Observe that the Piyavskii–Shubert data are approximately normally distributed, while the Depth–first results appear to be a superimposition of that normal distribution and a negative exponential distribution.

Also the choice of $\epsilon = 0.01$ above was arbitrary, so for 5 specific splines, the above algorithms were run on each function for ϵ over a range of values. The results are presented in Figure 7.

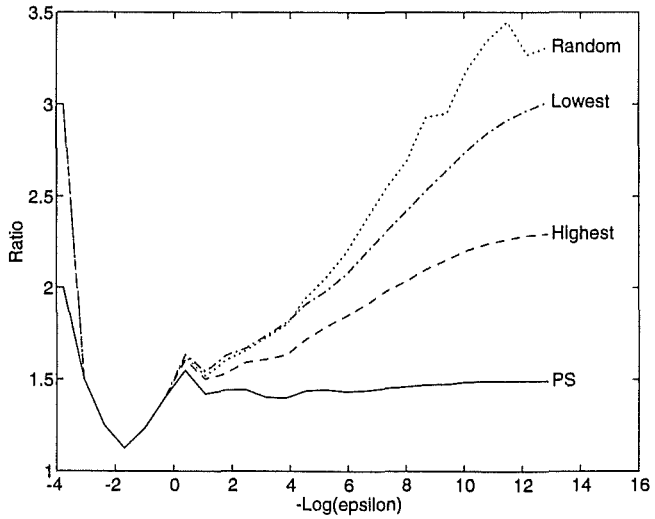


Figure 7 : Average ratios of number of function evaluations required for PS and various implementations of $DFPS$, compared to best possible, as ϵ varies.

Observe that the ratio for Piyavskii–Shubert appears to stabilize to about 1.4. It appears

that the ratios for the depth–first implementations also stabilize, but this occurs at a much finer accuracy. However, the fact that these averages were taken over only five functions means that the results have a high standard error. Because of this, it is inappropriate to draw a conclusion about which choosing rule is best as accuacy increases.

Since $n = 1000$, the results in the above table and the order shown in Figure 7 tend to suggest that $DFPS_H$ has the best choosing rule.

5 Summary

We have demonstrated that the algorithm, within its stated constraints, will produce accurate global optima for uni–variate Lipschitz continuous functions. Also, we have shown empirically that the number of function evaluations required by a depth–first Lipschitz based Piyavskii-Shubert algorithm is reasonable compared to Piyavskii-Shubert. Additionally, it requires substantially less memory.

6 Acknowledgements

First and foremost, we must thank our supervisers for this work, Bill Baritompa and John Hannah of the University of Canterbury Mathematics Department for their supportive and helpful guidance. Graham Woods and Rick Beatson provided valuable assistance in obtaining the empirical results for the new algorithm. The computing facilities and other resources provided by the University of Canterbury Mathematics Department proved utterly indispensible. Thank–you to all involved.

References

[1] W P Baritompa. Customizing methods for global optimization - a geometric viewpoint. *J. Global Optimization*, 3:193–212, 1993.

[2] R K Beatson and E Chacko. Which cubic spline should one use? *SIAM J. Sci. Stat. Comput.*, 13:1009–1024, 1992.

[3] R P Brent. *Algorithms for minimization without derivatives.* Prentice-Hall, Inc., 1973.

[4] Y G Evtushenko. Numerical methods for finding global extrema (case of a non–uniform mesh). *USSR Comp. Math. and Math. Phys.*, 11:38–54, 1971.

[5] P Hansen, B Jaumard, and S-H Lu. Global optimization of uni–variate Lipschitz functions: II. New algorithms and computational comparison. *Mathematical Programming*, 55:273–292, 1992.

[6] S A Piyavskii. An algorithm for finding the absolute extremum of a function. *USSR Comp. Math. amd Math. Phys.*, 12:57–67, 1972.

[7] B O Shubert. A sequential method seeking the global maximum of a function. *SIAM J. Numer. Anal.*, 9:379–388, 1972.

[8] F J Solis and R J-B. Wets. Minimization by random search techniques. *Mathematics of Operation Research*, 6:19–30, 1981.