

Tree-Direct: An Efficient Global Optimization Algorithm

Richard E. Haskell and Craig A. Jackson
Computer Science and Engineering Department
Oakland University
Rochester, Michigan 48309

Keywords: Global optimization, Derivative-free global search, Binary tree

Abstract

Tree-Direct (TD) is a general-purpose derivative-free optimization technique for real-valued functions in any dimension, which requires no parameter fine-tuning and exhibits relatively quick convergence to global optima. The algorithm uses a binary tree search space partitioning technique which systematically subdivides the hyperrectangular search space into smaller regions by bisecting regions associated with each node in the tree. Furthermore, through the use of mid-point sampling, scaling problems ordinarily associated with higher-order objective functions are mitigated. The balance between local and global search is accomplished by performing both local and global search simultaneously. This is done by selecting multiple leaf nodes at each iteration for further partitioning based on their relative expected potential of leading to a new optimal solution. The relative potential of each partition is characterized by two attributes: the value of the objective function at the mid-point of the partition, and the size of the partition. The former provides a measure of the partition's potential with respect to local search (i.e., partitions with better objective function values at their mid-point are more desirable than those with worse objective function values). The latter provides a measure of the partition's potential with respect to global search (i.e., larger partitions contain more unexplored territory, and therefore provide a greater opportunity for further exploration). Test results on several standard objective functions show the convergence rates of this new algorithm are quite competitive with other approaches.

Introduction

A number of derivative-free optimization techniques have been developed to address both continuous and discrete optimization problems where the objective function's derivative information is not available. The most widely used of these methods are Genetic Algorithms (GAs), Simulated Annealing, and Random Search [1–4]. Each of these techniques involves repeated evaluation of the objective function at locations that are determined through an intuitive heuristic that uses a stochastic mechanism. Because of this randomness, these methods often require a large number of function evaluations to converge to the optimal solution. Another set of approaches to optimization in the absence of exact derivative information are the Lipschitzian methods [5]. These techniques rely, not on the ability to determine the objective function's

derivative, but rather on the ability to determine a finite bound on the objective function's rate of change.

Knowing this bound, termed the Lipschitz constant, as well as the objective function's value at the upper and lower limits of each dimension of the region enables one to place a limit on the objective function within a given region. The standard Lipschitzian methods exploit the continuous nature of real-valued objective functions, and lessen the dependence on precise derivative information. However, one problem with these techniques is the number of evaluations required for higher order objective functions. A modified Lipschitzian method called the DIRECT algorithm, evaluates the objective function at the center of each region, thus minimizing the number of evaluations [6].

A central problem in any optimization algorithm is achieving an appropriate balance between global and local search. That is, the optimization technique must have a means of determining whether to continue searching near the current best location, or whether to sample some other region of the search space. For example, in GAs this is achieved through the mechanisms of selection, crossover and mutation. Likewise, simulated annealing uses the temperature parameter and the cooling schedule to control these decisions. The Lipschitz constant performs this role for the Lipschitzian methods. Often, the parameters that control this balance are highly problem-specific, and thus require a considerable amount of fine-tuning. An algorithm that places too much emphasis on local search will easily be trapped near a local optimum. Conversely, an algorithm that spends too much time performing global search will converge very slowly. Adjustment of the parameters that control this balance is itself often a non-linear optimization problem. Therefore, a technique that does not depend on such parameters would be desirable.

In this paper we present a new algorithm, called Tree-Direct (TD), that can find the global optimum of a real-valued function in any dimension. The name Tree-Direct refers to the fact that its integral structure is a binary tree and that it is partially based on the DIRECT algorithm

of Jones, Perttunen and Stuckman [6]. The algorithm uses a binary tree search space partitioning technique which systematically subdivides the hyperrectangular search space into smaller regions by bisecting regions associated with each node in the tree. By sampling the mid-point of each hyperrectangle, scaling problems ordinarily associated with higher-order objective functions are mitigated. The balance between local and global search is accomplished by performing both local and global search simultaneously. This is done at each iteration by selecting multiple leaf nodes for further partitioning, some of which are high in the tree (global search) while others are deep in the tree (local search). Only those leaf nodes with the best objective function value at a given level in the tree are considered for further partitioning. Of these only a small subset at the most global and most local ends of the spectrum are actually split at a given iteration.

The algorithm is presented in the following section and is then illustrated with a detailed example. Some test results are then presented for several standard objective functions which show that the convergence rates of this new algorithm are quite competitive with other approaches.

The Tree-Direct Algorithm

Let $f(\mathbf{x})$ be a real-valued function of n variables, $\mathbf{x} = (x_1, x_2, \dots, x_n)$. The goal is to find the global optimum (either a global maximum or a global minimum) of this function. The function f can be evaluated for any value of \mathbf{x} but no derivative information about f is known. The size of the search space, S , is defined by the upper and lower bounds of each of the n dimensions. Let these values be U_i and L_i , $i = 1$ to n . These can be thought of as the components of the vectors \mathbf{U} and \mathbf{L} . The midpoint of the search space is $\mathbf{M} = (\mathbf{U} + \mathbf{L})/2$.

Let $node_j$ be the j^{th} node of a binary tree. Each node of the tree is associated with a subspace s_j of the search space S . The size of the subspace s_j is given by the upper and lower bounds U_j and L_j . The midpoint of subspace s_j is $\mathbf{M}_j = (\mathbf{U}_j + \mathbf{L}_j)/2$. The value of the objective function associated with each node is $f(\mathbf{M}_j)$, i.e., the value of the function at the mid-point of the subspace. The root node of the binary tree represents the entire search space. At each successive level in the tree the size of the subspace, s_j , associated with $node_j$ is divided by 2.

The Tree-Direct algorithm is given in Figure 1. The functions *FindNodesToSplit* and *SplitNodes* are best described by means of an example.

Example -- Peaks function

The "peaks" function is the two-dimensional function shown in Figure 2. The upper and lower bounds are +3 and -3 in both dimensions. In testing this function we have added 7.0 to its value shown in Figure 2. The value of the global maximum is 15.106214. Figure 3 shows the first four iterations of TD for this function. Node 1 is the root node and the value of the function evaluated at the midpoint of the entire search space (0, 0) is 7.98. This value is indicated under the root node in the tree.

```
TD { Find global maximum of f(x)
    node1 = root_node
    value(node1) = f(M1)
    while not finished
        {
            FindNodesToSplit
            SplitNodes
        }
    }END_TD
```

Figure 1 The Tree-Direct algorithm, TD

Figure 2 The "peaks" function. (MATLAB file: *peaks.m*)

At the second iteration the search space is divided in half along the dimension with the largest range ($U_i - L_i$). In this case both dimensions have the same range so the first dimension is selected. (Had the second one been selected, we would have found the global maximum right away!) Nodes 2 and 3 are sampled at their midpoint and have function values 4.23 and 10.27 respectively. At level 2 the highest function value is 10.27 associated with node

3. Therefore, this is the node that is split at the third iteration.

The third iteration produces nodes 4 and 5 with midpoint function values of 6.56 and 8.19 at level 3. Level 2 still has one leaf node (node 4) with a function value of 4.23. The maximum value at level 3 is 8.19 for node 5. Therefore, at the next iteration we will split both

node 2 and node 5. These are the highest values in their respective levels. Even though node 2 has a function value of only 4.23 (compared with the value of 6.56 for node 4) its size is twice that of node 4 and therefore it has more of a chance to find a larger value. On the other hand, node 5 has already found a larger function value than node 4 for the same size search space.

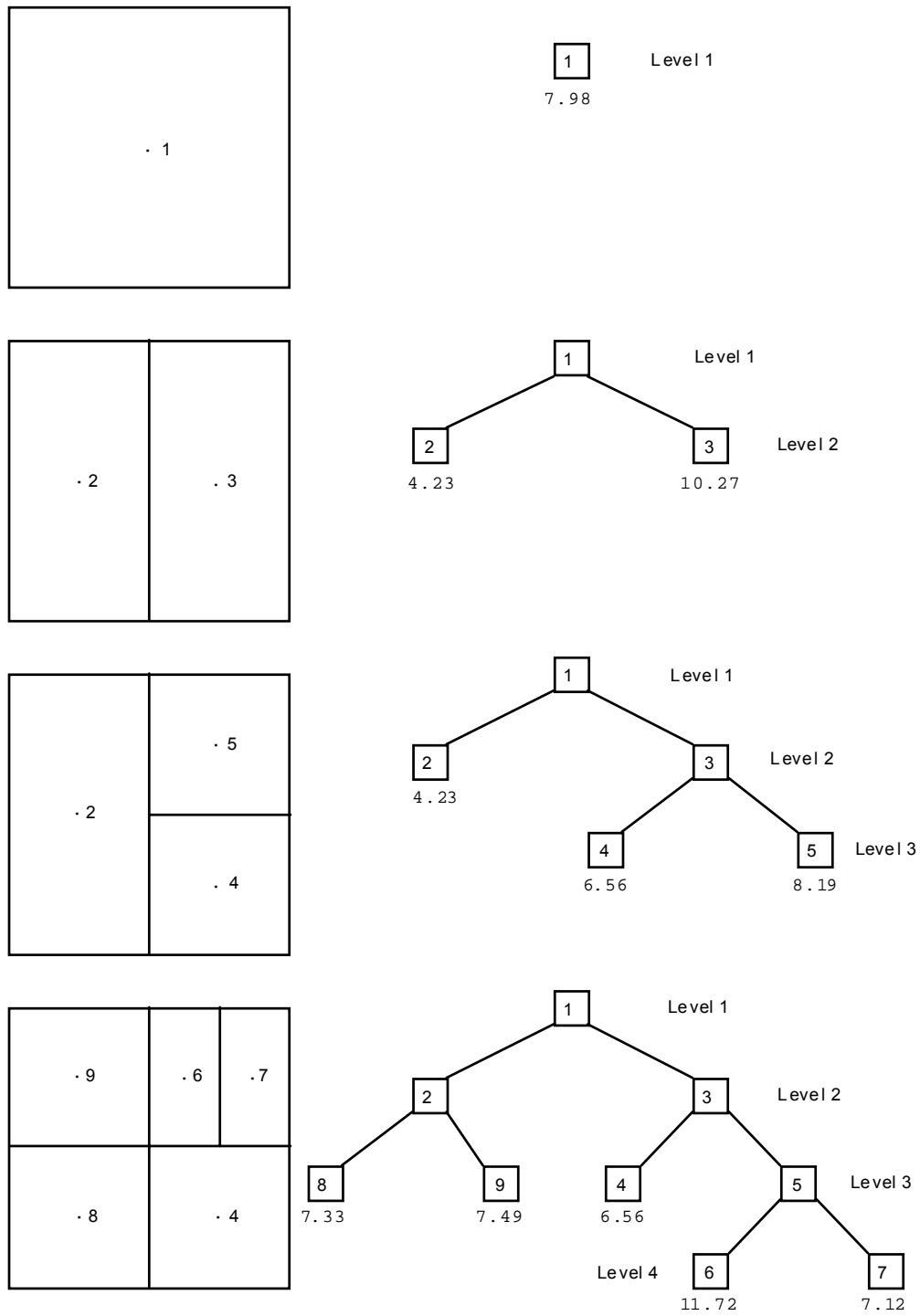


Figure 3 Binary tree representation of search space

Splitting nodes 2 and 5 leads to the partition and tree shown at the bottom of Figure 3. At this point the largest function value for leaf nodes at level 3 is 7.49 for node 9 and the largest function value for leaf nodes at level 4 is 11.72 for node 6. The next iteration would therefore split nodes 6 and 9.

At this point the function *FindNodesToSplit* in Figure 1 is to simply to select the leaf node at each level that has the maximum function value. However, this leads to a very inefficient algorithm in which too many unnecessary evaluations are made. The first improvement is to eliminate all leaf nodes at a particular level if the maximum function value at that level is lower than the best function value at a level above it in the tree. For example, if the function value on node 6 in Figure 3 were 5.34 instead of 11.72 then only node 9 would be split at the next iteration.

However, even this leaves too many nodes to be split at each iteration. To see how we can eliminate additional nodes consider Table 1 which shows the leaf nodes with the maximum function values after 17 nodes have been created. A plot of these three best fitness function values vs. leaf node level number is shown in Figure 4. Note that the value at level 4 falls below a line drawn from the level 3 value to the level 5 value. That is, the values for levels 3, 4, and 5 do not form a convex hull. To make a convex hull we need to eliminate the node at level 4. The idea is that even though the value at level 4 is larger than that of level 3, it is not enough larger considering that its size is smaller and the value of the best node at level 5 is significantly larger.

| Node No. | Level | Function value |
|----------|-------|----------------|
| 11 | 5 | 9.089436 |
| 7 | 4 | 7.123708 |
| 4 | 3 | 6.559508 |

Therefore, we will not split any node that is not on the convex hull of a plot of best fitness value vs. level number. For example, after 89 nodes have been added to the tree a plot of the best fitness value vs. level number will be as shown in Figure 5. After testing for the convex hull only the six nodes shown in Figure 6 are left as possible candidates to split.

Splitting only the best leaf nodes that fall on the convex hull produces satisfactory results as will be shown in the next section. However, the algorithm can be made even more efficient by noting that often the higher level number leaf nodes have almost the same best fitness

values as shown in Figure 6. This may be due to approaching a (possibly local) maximum. What is needed, in addition to splitting the best overall leaf node, is to split one or more of the most global (lowest level number) leaf nodes. We have found that splitting only three nodes at each iteration -- the most local node (highest level number) and the two most global nodes (lowest level number) from the leaf nodes on the convex hull produces very good results.

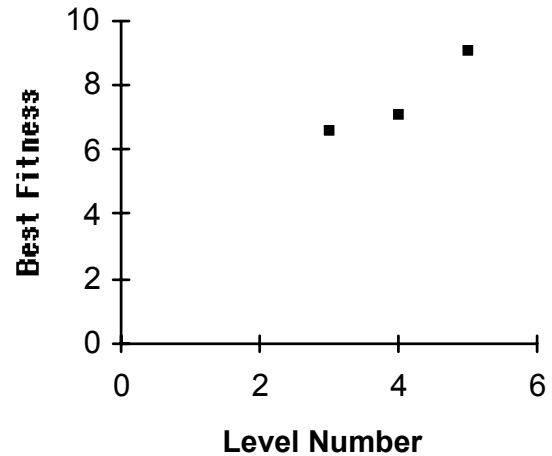


Figure 4 Level 4 is eliminated for not being on convex hull

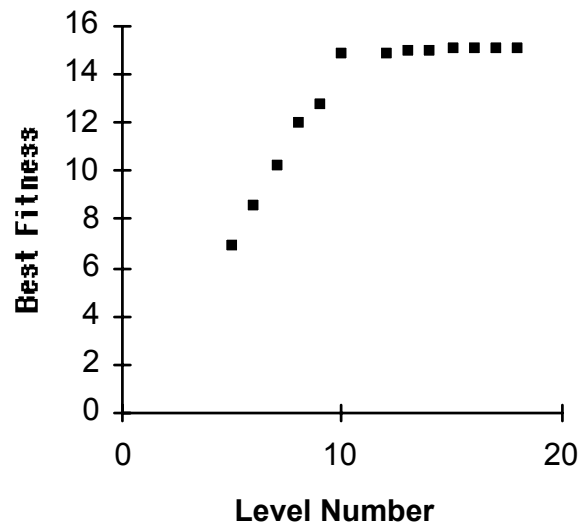


Figure 5 Best leaf nodes before testing for convex hull

Figure 7 shows a scatter plot of 103 evaluations of the "peaks" function. At this point the global maximum was found to be 15.104897 at the location (0.00293, 1.579102). This is within 0.01% of the true global maximum value. Additional results will be presented in the following section.

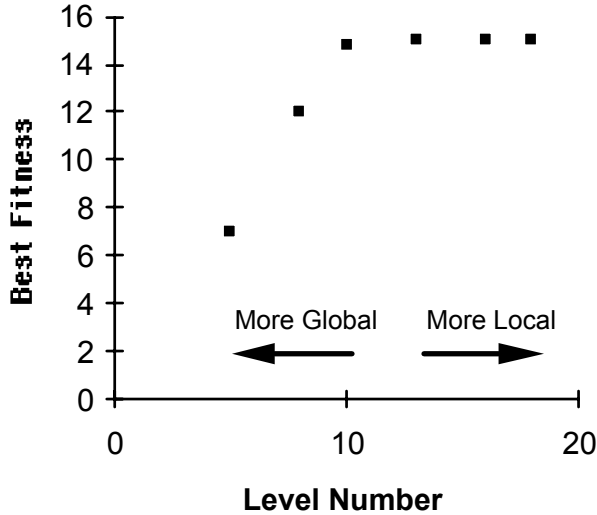


Figure 6 Best leaf nodes after testing for convex hull

Tests and Results

In addition to the Peaks function described in the previous section we tested the TD algorithm on the test functions listed in Table 2. These are the same test functions used by Jones, et. al [6]. The TD algorithm can handle either a global maximum or a global minimum. This is specified in a file read by the program.

Table 3 gives the number of function evaluations required to find the global optimum to within an error of less than 0.01% for each of the test functions listed in Table 2. The row labeled *DIRECT* contains the results reported by Jones, et al. [6] in which they compare their results with 18 other methods, most of which they beat handily. The row labeled *TD-ch* is the Tree-Direct method described above in which all the leaf nodes on the convex hull are split on each iteration. The last row, labeled TD-3, is the Tree-Direct method described above in which only three leaf nodes on the convex hull (the most local and the two most global) are split on each iteration. Notice that the TD-3 method does particularly well for the H3, H6, BR, GP, C6, and SHU test functions.

Peaks

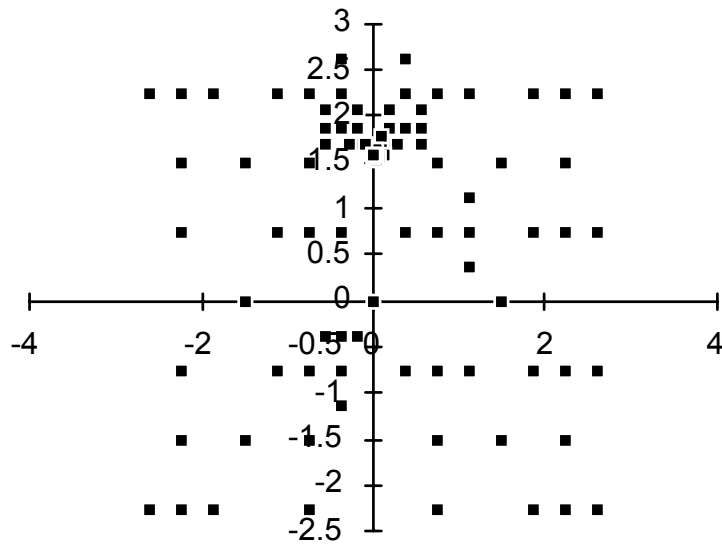


Figure 7 Scatter plot for "peaks" function -- 103 evaluations

| Test function | Abbreviation | Min or Max | Number of dimensions | Number of local optima | Number of global optima |
|-----------------|--------------|------------|----------------------|------------------------|-------------------------|
| Peaks | PK | max | 2 | 2 | 1 |
| Shekel 5 | S5 | min | 4 | 5 | 1 |
| Shekel 7 | S7 | min | 4 | 7 | 1 |
| Shekel 10 | S10 | min | 4 | 10 | 1 |
| Hartman 3 | H3 | min | 3 | 4 | 1 |
| Hartman 6 | H6 | min | 6 | 4 | 1 |
| Branin RCOS | BR | min | 2 | 3 | 3 |
| Goldstein-Price | GP | max | 2 | 4 | 1 |
| Six-Hump Camel | C6 | min | 2 | 6 | 2 |
| 2-D Shubert | SHU | min | 2 | 760 | 18 |

| Method | PK | S5 | S7 | S10 | H3 | H6 | BR | GP | C6 | SHU |
|--------|-----|-----|-----|-----|-----|-----|-----|-----|-----|------|
| DIRECT | NA | 155 | 145 | 145 | 199 | 571 | 195 | 191 | 285 | 2967 |
| TD-ch | 141 | 439 | 307 | 315 | 220 | 711 | 291 | 284 | 160 | * |
| TD-3 | 103 | 349 | 251 | 277 | 99 | 261 | 131 | 133 | 89 | 1927 |

* converged to local optima at evaluation number 454 out of a total of 16,135 evaluations

Summary

This paper has presented a new global optimization algorithm based on a binary tree in which each node of the tree represents a subspace of the search space. Each time a node of the tree is split, the size of the search space associated with that node is cut in half. The function to be optimized is evaluated at the center of each subspace region. At each iteration three leaf nodes are selected to be split. One is the most local node (deepest in the tree) with the best fitness value. The other two are the two most global levels that remain on the convex hull of best fitness leaf nodes vs. level number. The algorithm has the advantage of not requiring any parameter fine-tuning. The method has been shown to be efficient when applied to a number of specific test functions.

References

1. Jang, J.S.R., C.T. Sun, and E. Mizutani, *Neuro-Fuzzy and Soft Computing: A Computational Approach to Learning and Machine Intelligence*, Prentice-Hall, Upper Saddle River, New Jersey, 1997.
2. Goldberg, D. E., *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, Reading, MA, 1989.

3. Otten, R. H. J. M., and L. P. P. van Ginneken, *The Annealing Algorithm*, Kluwer Academic, 1989.
4. Solis, F. J., and J. B. Wets, *Minimization by Random Search Techniques*, Mathematics of Operations Research, 6(1), pp. 19-30, 1981.
5. Shubert, B., *A Sequential Method Seeking the Global Maximum of a Function*, SIAM Journal on Numerical Analysis, Vol. 9, pp. 379-388, 1972.
6. Jones, D.R., C.D. Perttunen, and B.E. Stuckman, *Lipschitzian Optimization Without the Lipschitz Constant*, Journal of Optimization Theory and Applications, Vol. 79, pp. 157-181, 1993.
7. Graham, R.I., *An Efficient Algorithm for Determining the Convex Hull of a Finite Planar Set*, Information Processing Letters, 1, pp. 132-133, 1972.

Acknowledgment

The authors are grateful to Donald R. Jones of General Motors Research and Development Center for many helpful discussions during the course of this research.