TERNARY TREES USAGE FOR COMPUTATIONAL EXPERIMENT DATA STORAGE

A.N. Kovartsev[1], D.A. Popova-Kovartseva[1], E.E. Gorshkova[1]

[1] Samara National Research University, Samara, Russia

Abstract. Repository for a great number of the suspected function runs must often be organized during the realization of numerical global optimization method or program modules testing methods. In this connection the problem of a fast-access search of required element arises. A way to solve the problem can be found through the organization of an effective data repository, which allows for rapid implementation of the search and insertion of new elements. The binary and ternary trees storage options are considered in the paper, their comparison is made, recommendations for use of analyzed data models are reported.

Keywords: binary trees, ternary trees, data structures, global optimization.

Citation: Kovartsev AN, Popova-Kovartseva DA, Gorshkova EE. Ternary trees usage for computational experiment data storage. CEUR Workshop Proceedings, 2016; 1638: 820-827. DOI: 10.18287/1613-0073-2016-1638-820-827

Introduction

Considerable volume of function "runs" must be stored during application of numerical global optimization method (GO) [1, 2] or program modules testing methods [3, 4]. The main requirement for GO algorithms and program modules testing methods is the speed of the algorithm. Let us assume that the computation time of function $f(x)$ is sufficiently large. This situation is typical for solving optimization problems in technical systems design process, such as gas turbine engines, aircraft, etc. Some tests in the known GO algorithms are repeated in increments (for example, in bisection method or its modified version [5]) and that leads to redundant computation of $f(x)$. A way to solve this problem can be found through the organization of an effective data repository allowing for rapid implementation of the search and insertion operations of new elements.

1 Disadvantages of traditional binary trees usage

Let us agree, that by the test we mean function $f(x)$ calculation at the point specified by optimization algorithm or function testing algorithm. Optimization methods complexity is the total number of algorithm steps [6]. However, taking into account the high advanced computers performance, by optimization algorithms complexity it's more accurate to mean the total number of calls to the optimizable function, since the algorithm steps can have different complexity, and each function call takes the same amount of time for any algorithm. On the other hand, the number of function calls is independent of the computer processing power that allows comparing algorithms to be implemented on computers with different performance.

At the same time, each function call may require significant computing time depending on the complexity of the test function.

This article deals with the problem of data repository organizing for GO algorithm based on the modified bisection method (BM) [5]. In the BM algorithm the search space partitioning strategy may be defined as non-uniform space division with the use of regular structures (see [1]). However, it is necessary to compute the function at the same points of independent variables set.

In order to avoid repeated calculations, it is expedient to store the point coordinates and function values at them in a separate data repository. Thus the data array may have considerable size. In this case there is the problem of finding an element in the array. For example, the average complexity of sequential search in an unordered and in an ordered array is $O(n)$ [7].

Practically, binary trees (b-trees) are generally used to increase the efficiency of the search algorithm [8, 9]. Binary search for balanced binary trees [10] has an average complexity proportional to $O(\log_2 n)$ and depends on the depth of a binary tree.

By the depth of a binary tree is generally understood [10] the number of nodes on the longest path from the root of the tree to its leaves. However, usage of binary trees for data storing at the test points of the tested or optimized function has its own aspects. In particular, a certain procedure of data entry or additional sorting of data is required to build an optimal balanced binary tree even for regular grids. For the irregular grids of experiment data sorting is essential.

Figure 2 shows the balanced binary tree built for the regular grid (see. Fig. 1) using one of the rational methods of construction of the balanced binary tree.

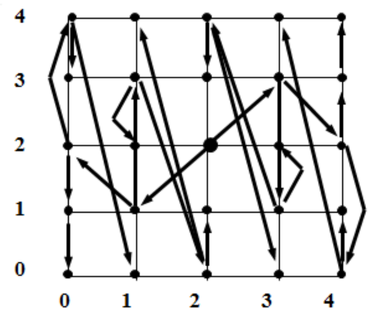

Fig. 1. Strategy of the balanced binary tree building on a regular grid

In addition, computational experiment must comply with the sequence shown in Figure 1. The calculations start from a point with coordinates (2, 2). A binary tree shown in Figure 2 corresponds to this plan.

However, the gradual detalization of the search area is peculiar for BM, and it starts from a large sample spacing with a gradual decrease in specific fragments in the searching range. The range of search "densifies" nonuniformly here. In this case, it's impractical to achieve the balance of the binary tree, and this leads to efficiency loss of search algorithms. Figure 3 shows a binary tree corresponding to the experiment shown in Figure 4.

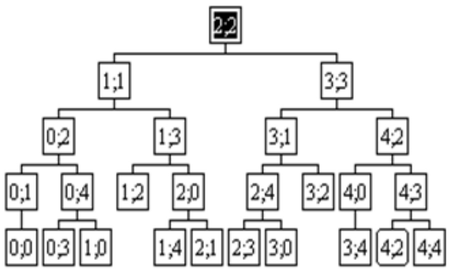

Fig. 2. Balanced binary tree for 5*5 experiment grid

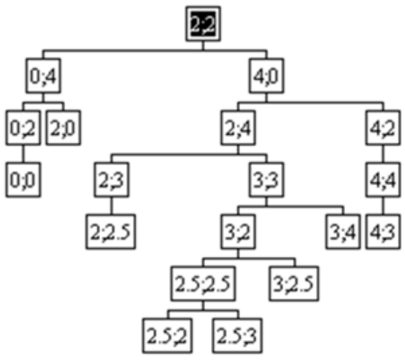

Fig. 3. Binary tree for BM

As can be seen from the figure binary tree becomes not balanced for irregular grid. One way to improve the efficiency of the search algorithm is to use more complex data structures, such as ternary trees (t-trees).

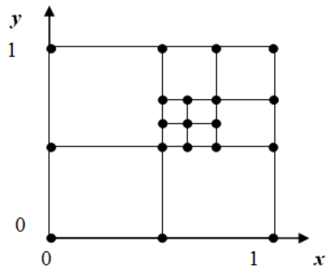

Fig. 4. Construction of the balanced binary tree

2 Ternary trees

Two ordering relationships ("<" and "> =" or "<=" and ">") are used to build a binary tree that allow to implement a procedure of binary division of the ordered set. Three conditions are required for the ternary tree construction which allow to split an ordered set into three parts. As such, will be the following conditions "<", "=" and ">".
Points of the experiment will be organized by the following criterion $z = x + y$, i.e. by the sum of the experimental points coordinates. In general, points with coordinates that correspond to the condition of equality, not often encountered; in this case, a ternary tree is converted into binary. However, there is a fair amount of such points for regular grids. Points that satisfy the condition of equality are located on the diagonals ($x + y = const$) for 5 * 5 grid in figure 5.

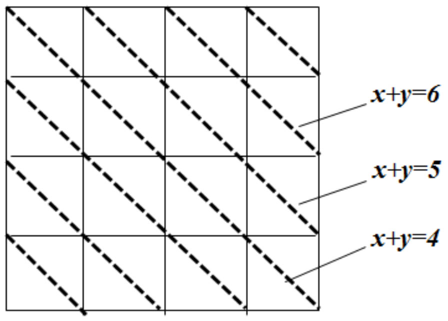

Fig. 5. Construction of the ternary tree

The result is a t-tree, shown in Figure 6.
The comparison between an average efficiency of search for b- and t-trees for different regular grids of partitioning the search space is of some interest.
Let n be a number of nodes of a regular grid for each coordinate, $N = n^2$ - the total number of grid nodes, k - the depth of a complete binary tree [10].

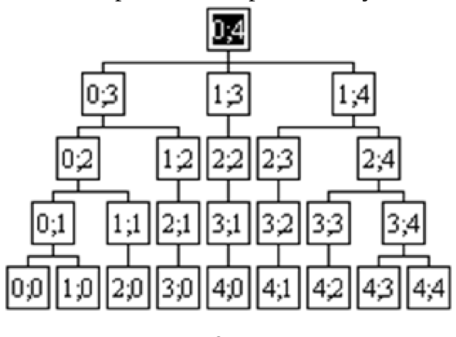

Fig. 6. Ternary tree for 5*5 experiment grid

A binary tree is "complete" if both subtrees of all nodes are "balanced", i.e. they have approximately the same number of nodes [10].
The total number of nodes for a "complete" (balanced) b-tree can be calculated from the formula [7] $2^k - 1$.

Suppose that the complete balanced tree is constructed for each regular grid, so that the first few levels except the last one are completely filled with nodes. Basically, it is impossible to build completely filled "balanced" trees for $n \times n$ grid, because the number of elements in the grid and in complete binary tree does not match.
For any number of elements (in this case the $n \times n$ grid) the "complete" tree depth can be calculated from the formula:

$$k = \begin{cases} [\log_2 N], & if \quad [\log_2 N] = \log_2 N, \\ [\log_2 N] + 1, & else. \end{cases}$$

On the assumption of equally probable binary tree filling with the information in the grid nodes we calculate the average complexity of the search on the "complete" b-tree. For a grid with the number of elements $N = n^2$, the elements of the "complete" tree up to the depth $k - 1$ will be completely filled. 2^k elements are located on each k level of the binary tree. It is easy to show that for equally probable distribution of nodes of a regular grid in a binary tree the average complexity of search can be calculated using the following formula:

$$B(N) = \frac{1}{n^2}(k\ n^2 + (k\ + 1) - 2^k\ ).$$

Unfortunately, "complete" balanced b-tree is generated only when the inclusion of grid elements into a binary tree has a certain order. In general, the search process can form linear structure with greater depth than the "complete" b-tree.
Knuth [11] has shown that, if all possible options of elements inclusion (N!) into the binary tree are assumed to be equally probable, then the average depth of a binary search is $\sqrt{N}$ and averagely $1.386 \log_2 N$ comparisons are required per search. Figure 7 shows graphs of average search complexity. It can be seen, that the average complexity of the search of the "complete" binary tree is much smaller than in case of arbitrary filling of b-tree with grid elements.
The ternary tree depth is almost the same as the dimension of the grid, as the tree depth cannot be less than the number of nodes on the grid diagonal because of the equality operation. It is much more difficult to calculate the average complexity of the search *T(N)* for a ternary tree. Figure 7 shows a graph of behavior *T(N)* of the number of grid elements resulted from the computing experiment for quasioptimal t-trees.

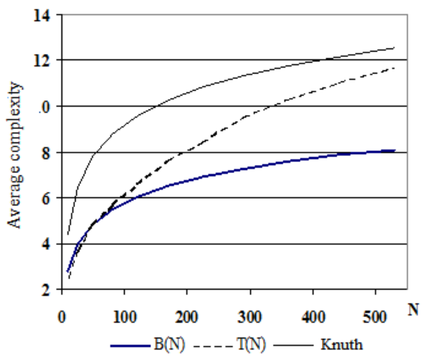

Fig. 7. Comparison of the average complexity of the search on the b and t trees

As can be seen from the figure a ternary tree has disadvantages in the description of the regular grid of the experiment in comparison with optimal binary tree. However, a margin of free branches in the tree that can be filled with new grid nodes is very important in the nonlinear search space partitioning method when changing the sample spacing of the search space. In this regard, the binary tree loses much to t-tree, since b-tree branches are located at the last level of the tree.

The reservation availability for new experimental points occurs in the t-tree, which do not increase the depth of search long enough. The latter happens because the operation of the equality "=" implements the partition of the grid points set into equivalence classes. All classes are comparable to operations "<" and ">" and form "branches" of equivalence.

Grid densifying in separate squares leads to the appearance of new branches, that quickly find an empty place on a tree. The upper levels of binary trees tend to be already completed by this time and further development is possible only by increasing the depth of the tree.

It is inappropriate to sort b-tree in order to improve its structure when the operations of element inserting and searching for an item in the tree are made at the same time. The fastest sorting algorithms require $N\log_2 N$ operations to improve the tree structure [12]. In this case, the efficiency of the dichotomous search on balanced b-tree will be lost due to the sort operations.

Thus, ternary tree can be used as the data model of experimental points placement during software module testing.

3 The efficiency of the search algorithm in ternary tree research

The efficiency of the search algorithm in ternary tree depends on the properties of the test function. We used 6 test functions with different topologies in the computational experiments and, therefore, different computational complexity in terms of solving the problem of GO. The functions are different: from relatively simple rational functions, more complex ravine functions, multiextremal functions (including discontinuous functions) to ROOT standard test functions offered on the site [13].

Figure 8 shows the results of computational experiments of the efficiency of the search algorithm in ternary tree research for the considered test functions. For clarity, dependency graphs of the average search length of optimal "complete" binary tree and evaluation of the same values for equally probable way of filling b-tree proposed by Knuth [11] are shown below.

As can be seen from the figure, in most cases, the efficiency of the search on the t-tree is comparable with the average complexity of the search at the optimal binary tree. The latter is achieved due to the large number of ternary tree free connections, located on the upper levels. In general, the average search length of t-tree has an acceptable value for large volumes of stored data. In comparison it should be noted that t-tree was discussed in unsorted form, while b-tree, for the same amount of information was optimal and balanced. The latter means the high efficiency of the search on t-trees.

If we consider the relative average search length (ratio of the average search length to the total number of elements of the tree), then with increasing of the size of t-tree search efficiency only increases (see Fig. 9).

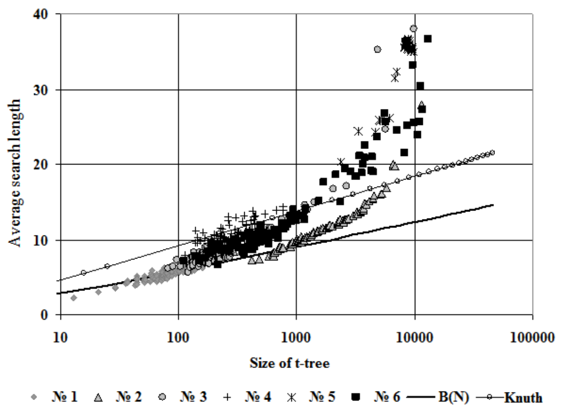

Fig. 8. The efficiency of the search on the t-trees

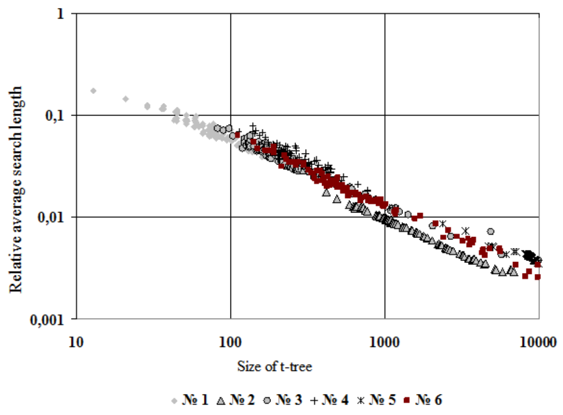

Fig. 9. The relative efficiency of the search on the t-trees

Conclusion

In this article, we considered the problem of data repository organization and data search algorithm on ternary trees, designed to serve the global optimization methods and function tests that require processing of large volumes of information. Using of ternary trees gives a number of advantages in comparison with traditional binary trees, as it does not require to perform a sort operation or balance the tree in order to improve the efficiency of the search on ternary trees, and the average complexity of the search is comparable to the efficiency of search on the binary optimal balanced tree.

Acknowledgements

This work was supported by the state Ministry of Education and Science of the Russian Federation as part of the activities of the Program of competitiveness improving of SSAU among the world's leading research and education centers for 2013-2020.

References

1. Kovartsev AN, Popova-Kovartseva DA. On efficiency of parallel algorithms for global optimization of functions of several variables. Computer Optics, 2011; 35(2): 256-261. [in Russian]
2. Kovartsev AN. A deterministic evolutionary algorithm for the global optimization of morse cluster. Computer Optics, 2014; 39(2): 234-240. [in Russian]
3. Kovartsev AN, Popova-Kovartseva DA, Abolmasov PV. Efficiency study of global parallel optimization for multivariable function. Vestnik NNGU, 2013; 3(1): 252-261. [in Russian]
4. Kovartsev AN. An efficient algorithm for testing the truth of assertions for real numbers expressed in relational signatures. Computer Optics, 2014; 38(3): 550-554. [in Russian]
5. Kovartsev AN, Popova-Kovartseva DA, Gorshkova EE. Software testing based on global search of several variables functions discontinuity. Proceedings of International Conference Information Technology and Nanotechnology (ITNT-2015), 2015: 389-396.
6. Nemirovskij AS, Yudin DB. The complexity of the problems and the effectiveness of optimization methods. Moscow: Nauka, 1979; 384 p. [in Russian]
7. Makkonel J. Analysis of algorithms. Introductory course. Moscow: Tekhnosfera, 2002; 304 p. [in Russian]
8. Adamenko AN, Kuchukov AM. Logic programming and Visual Prolog. St. Petersburg: BHV-Peterburg, 2003; 992 p. [in Russian]
9. Ivanov BN. Discrete mathematics. Algoritms and programs. Moscow: Laboratoriya bazovyh znanij, 2001; 288 p. [in Russian]
10. Meyer B, Baldwin K. Programming methods. V. 2 (3). Moscow: Mir, 1982; 386 p.
11. Knuth D. Art of Computer Programming. Sorting and searching. V. 3. Moscow: Mir, 1978; 848 p.
12. Holl P. Computational structure. Introduction to nonnumerical programming. Moscow: Mir, 1978; 216 p.
13. Global optimization algorithms. Sourse: <http:www.r-tech.narod.ru/gtest/html>