UDC 004.93
doi:10.31799/1684-8853-2021-4-18-27

Articles

Indexing algorithm based on storing additional distances in metric space for multi-vantage-point tree

V. V. Fomin[a], *Dr. Sc., Tech., Professor, orcid.org/0000-0001-7040-5386, v_v_fomin@mail.ru*
I. V. Aleksandrov[b], *Post-Graduate Student, orcid.org/0000-0002-6258-8191*
[a]*Herzen State Pedagogical University of Russia, 48, Moika Emb., 191186, Saint-Petersburg, Russian Federation*
[b]*The Bonch-Bruevich Saint-Petersburg State University of Telecommunications, 22-1, Bolshevikov Pr., 193232, Saint-Petersburg, Russian Federation*

Introduction: *The similarity search paradigm is used in various computational tasks, such as classification, data mining, pattern recognition, etc. Currently, the technology of tree-like metric access methods occupies a significant place among search algorithms. The classical problem of reducing the time of similarity search in metric space is relevant for modern systems when processing big complex data. Due to multidimensional nature of the search algorithm effectiveness problem, local research in this direction is in demand, constantly bringing useful results.* ***Purpose:*** *To reduce the computational complexity of tree search algorithms in problems involving metric proximity.* ***Results:*** *We developed a search algorithm for a multi-vantage-point tree, based on the priority node-processing queue. We mathematically formalized the problems of additional calculations and ways to solve them. To improve the performance of similarity search, we have proposed procedures for forming a priority queue of processing nodes and reducing the number of intersections of same level nodes. Structural changes in the multi-vantage-point tree and the use of minimum distances between vantage points and node subtrees provide better search efficiency. More accurate determination of the distance from the search object to the nodes and the fact that the search area intersects with a tree node allows you to reduce the amount of calculations.* ***Practical relevance:*** *The resulting search algorithms need less time to process information due to an insignificant increase in memory requirements. Reducing the information processing time expands the application boundaries of tree metric indexing methods in search problems involving large data sets.*

Keywords *— complex data indexing, tree data structures, metric access methods, priority queue of nodes, multi-vantage-point tree.*

For citation: Fomin V. V., Aleksandrov I. V. Indexing algorithm based on storing additional distances in metric space for multi-vantage-point tree. *Informatsionno-upravliaiushchie sistemy* [Information and Control Systems], 2021, no. 4, pp. 18–27. doi:10.31799/1684-8853-2021-4-18-27

Introduction

When developing an applied analytical system [1] based on metric methods of artificial intelligence (methods of decision functions), the authors faced the classic problem of computational complexity. The task was to increase the speed of data processing by modifying the indexing algorithms. The emphasis is on modification of tree search algorithms as part of the task of overlapping tree nodes in order to reduce number of operations for calculating distances between objects. They use metric methods (K nearest neighbor's method, method of standards, method of correction increments, and others) to solve similarity search problems in practice. Distinctive features of these methods are: 1) any object from data set D is a point in space; 2) during the search, distances between two objects are repeatedly calculated; 3) distance processing takes place in metric space.

Let X — an arbitrary set. The function dist is called a metric on the X, if for all $x, y, z \in X$ the following conditions are satisfied: a) $\mathrm{dist}(x, y) \geq 0$ (non negativity); b) $\mathrm{dist}(x, y) = 0$, when $x = y$ (identity to itself); c) $\mathrm{dist}(x, y) = \mathrm{dist}(y, x)$ (symmetry); d) $\mathrm{dist}(x, y) \leq \mathrm{dist}(x, z) + \mathrm{dist}(z, y)$ (triangle inequality).

The presence of the triangle inequality (fourth property — subparagraph d) is inherent in its metric spaces. Therefore, the authors in the article did not consider datasets in which the triangle inequality does not performed. At the same time, we note that there is a study on the modification of the VP-tree for non-metric spaces using the Bregman distance, which does not satisfy the triangle inequality principle [2].

To find the nearest object among the set of data D in metric space, data indexing technology metric access methods (MAM) is used [3, 4]. MAM uses a special data structure to speed up the search for the nearest object in database D by reducing number of calculated distances.

We will highlight the following features of MAM technology:

1) the presence of a metric (distance function) for any two objects [5];

2) during the search, MAM algorithms use the principle of triangle inequality [6], which makes it possible to exclude the calculation of additional distances;

3) the computational complexity of the search algorithm depends significantly on the number of calculated distances between two objects [3].

There are many types of MAM tree algorithms, including GHT [7], D-index [8], BP [9], MM-tree [10], Onion-Tree [11] and others. One of the first MAM algorithms successfully applied in practice was the M-Tree [12–14]. Among the disadvantages of M-Tree, we highlight the problem of overlapping tree nodes, which increases the number of distance calculations. An alternative approach to using vantage points to the M-Tree is the OMNI algorithm [15]. OMNI uses strategically positioned vantage points to increase efficiency. Another classical algorithm Geometric Near Access Tree (GNAT) [16] suggests taking into account not only the minimum and maximum distances to objects, but also the distances between pairs of vantage points during the search.

A characteristic feature of the M-Tree, OMNI, and GNAT algorithms reviewed is the problem of overlapping nodes of the same level with each other. We can highlight the following M-Tree modifications to solve this problem:

— Slim-tree [17], which reduces the number of intersections of nodes of the same level with each other by using the slim-down algorithm. The disadvantage of the algorithm used is the need to re-insert objects;

— PM-tree [18], which combines a compact partition and a set of reference points. In the PM-tree, one can form the areas of nodes by the intersection of multiple hyperspheres defined by a global set of vantage points.

An alternative to the M-Tree family of algorithms is the VP-tree [19–21] and its modification with several reference points in each node — the multi-vantage-point (MVP) tree [22, 23]. The features of the structures are the balance tree based on the median data partition and the absence of intersection of nodes of the same level. The development of VP-tree ideas is the algorithm Fixed-Queries Tree (FQT) [24], which suggests using a single vantage point for all nodes of the same level to reduce calculations. The absence of the problem of intersection of nodes of the same level has a significant impact on the search efficiency. Thus, research [25] has shown in practice that searching based on disjoint nodes in an MVP-tree is more efficient than in an M-tree.

Currently, there are theoretical studies on the use of VP / MVP-trees as a basic component to improve the efficiency of various search methods [19, 21, 23]. Experimental results in [26, 27] confirmed the competitiveness of VP / MVP-trees compared to other metric data indexing methods. Analysis of the results obtained in practice for VP / MVP-trees proves their efficiency in metric spaces of small, medium [26] and large [20] dimensions.

Now, there is a demand in the development of search improvements for VP / MVP-trees, which is confirmed by various modern publications [20, 28]. Therefore, the research of reducing the number of calculated distances between two objects (search complexity) for the MVP-tree when solving problems of searching the nearest objects is relevant.

One of the ways to improve the search efficiency in the MVP-tree is to use a priority queue for processing nodes. As part of the adaptation of the priority queue to the MVP-tree, the authors have developed an algorithm for determining the intersection of the search area with the subtrees of a node: determination of intersection based on hierarchic processing of nodes (DIHPN).

DIHPN algorithm

Feature considered MVP-tree is logarithmic nearest neighbor search complexity depending on the number of objects in the original data set D. Without loss of generality, consider and analyze a search algorithm for a classical MVP-tree with the following characteristics:

1) $U = 2$ — branching of one vantage point;

2) $Y = 2$ — number of vantage points in one node of the tree;

3) $V = \{v1,\ v2\}$ — set of vantage points in each node of the tree, where $v1$ — the primary point, $v2$ — secondary;

4) $B = U^Y = 4$ — number of children for each node of the tree;

5) $A = \{A1, A2, A3, A4\}$ — set of subtrees in each node of the tree;

6) $R = \{r1, r2, r3\}$ — set of median distances (radii) that defines the boundaries of the four regions of the node: $A1 \in (v1, r1)$ && $A1 \in (v2, r2)$, $A2 \in (v1, r1)$ && $A2 \notin (v2, r2)$, $A3 \notin (v1, r1)$ && $A3 \in (v2, r3)$, $A4 \notin (v1, r1)$ && $A4 \notin (v2, r3)$, where (v, r) — hypersphere centered at point v and radius r;

7) $K = 1$ — branching of each leaf;

8) $P = 0$ — number of stored distances in the leaves from each leaf point to the set of vantage points.

The research [29] have identified the following features of the classical algorithm for finding the nearest object n for the query point q in the MVP-tree:

1. The search for the nearest object is recursive. During each iteration, it directly processes the data of only one node of the tree.

2. The node processing queue is formed based on the depth-first search principle. When a node is processed, the search area is constrained by the current node and all of its parent nodes. Each node implicitly takes into account its own set of hierarchical search rules.

3. Due to the architecture of building an MVP-tree, the descendants of $A1$, $A2$, $A3$, and $A4$ with-

in the same parent node do not intersect with each other.

4. Taking into account the set of hierarchical search rules and the architecture of building an MVP-tree allows you to set narrower boundaries for the search area.

5. The set of points D of the node *Node* of the tree T_{MVP} is defined by the intersection of two hyperspheres with centers at $\{v1, v2\}$ and the corresponding two radii from $R = \{r1, r2, r3\}$ of the parent node Parent.

6. Within the framework of parent node Parent, the fourth area $A4$ has only minimal borders.

7. The MVP-tree is a balanced tree.

Research [30] suggests using the priority queue $\underline{C}$ for processing nodes to improve efficiency of the classical search for the nearest object. Sorting nodes by increasing the distance from them to the query point q allows you to set the priority of processing areas. During research, when using the priority queue C in the search for the closest object for an MVP-tree, we identified the problem of additional computations. The emergence of additional calculations is due to the need to recreate the set of hierarchical search rules for each node. The nodes extracted from the priority queue for processing can belong to different parts and levels of the MVP-tree. Note that not taking into account the set of hierarchical search rules can lead to an increase in the number of overlapping nodes. Let us introduce the following notation and characteristics:

1) D — input set of points for indexing;
2) N^D_{points} — number of points in a D;
3) H — number of levels in the tree;
4) W — number of nodes at all levels in the tree;
5) L_k — k-th level in the tree.

Based on the notation introduced, we estimate influence of the number of overlapping nodes in the MVP-tree on the search efficiency. In previous author's studies [31], the following dependencies and estimates are given:

1) maximum estimate of the number of intersecting nodes NIL_k from the number k of the level L_k of the tree, when the set of hierarchical search rules is not taken into account:

$$\mathrm{NIL}(k) = B^{2k}, \tag{1}$$

where $B = U^Y = 4$ — number of children for each node in the tree;

2) based on formula (1), dependence of the total number of intersecting nodes NIT for the entire MVP-tree was calculated and presented as follows:

$$\mathrm{NIT} \approx B^H \sim N^D_{points}. \tag{2}$$

Studies have derived the dependence of the efficiency of finding the nearest object to the point q with the search radius r_q for tree structures MAM [3]:

$$O\left(NN(q, r_q)\right) \approx \frac{1}{r_0^{\Omega}} \sum_{k=0}^{H-1} \left(1 + \mathrm{fat}(T_{\mathrm{MVP}})(N^k_{nodes} - 1)\right) \times$$
$$\times \left(N^D_{points}\right)^{\frac{k}{H}} \times \left(r^k_{aver} + r_q\right)^{\Omega}, \tag{3}$$

where r_0 — radius required to cover the entire set of points at the root of the tree T_{MVP}; $\Omega \in R$ — internal dimension of the space; $\mathrm{fat}(T_{\mathrm{MVP}})$ — absolute fat factor of the tree T_{MVP}, which characterizes the degree of intersection of nodes with each other; N^k_{nodes} — average number of nodes at the k-th level of the tree; r^k_{aver} — average radius of the node coverage at the k-th level, which is calculated by the formula:

$$r^k_{aver} = \sqrt[\Omega]{\left(N^D_{points}\right)^{\frac{-k}{H}}}. \tag{4}$$

The formula for calculating the parameter absolute fat factor [3] is as follows:

$$\mathrm{fat}(T_{\mathrm{MVP}}) = \frac{I_C - H \times N^D_{points}}{N^D_{points}} \times \frac{1}{(W - H)}, \tag{5}$$

where I_C displays the total number of accesses to the tree nodes required to respond to a point request for each of the N^D_{points} objects in the tree T_{MVP}:

$$I_C \sim \mathrm{NIT}. \tag{6}$$

Taking into account the architectural features of the MVP-tree and formula (4), we rewrite formula (3) for the MVP-tree as the following dependency:

$$O(NN(q, r_q)) \sim \mathrm{fat}(T_{\mathrm{MVP}}). \tag{7}$$

Based on the fact that the range for the absolute factor $\mathrm{fat}(T_{\mathrm{MVP}})$ is [0..1] and analysis of formula (5), we get the dependence:

$$\mathrm{fat}(T_{\mathrm{MVP}}) \sim I_C \sim \mathrm{NIT}. \tag{8}$$

From formulas (1)–(8) it follows:

1) if the set of hierarchical search rules is not taken into account, there is a problem of overlapping nodes with each other;

2) the number of intersecting nodes within the entire tree depends on the number of indexed N^D_{points} in D;

3) search efficiency is inversely proportional to the number of intersecting nodes.

As part of solving the problem of additional computations, we propose to use a modification of the classical algorithm of the process of determining the fact of intersection of the search area with the current node Node. The following logical statement determines the fact that a node Node of type Type intersects with the search area (q, ln) based on the classical method:

$$overlap(Node,\ (q,\ ln)) = \left(\begin{array}{l} ((Type = A1)\,\&\,\&((ln + r1) \geq l1)\,\&\,\&((ln + r2) \geq l2))\,\| \\ (((Type = A2)\,\&\,\&((ln + r1) \geq l1)\,\&\,\&((ln + l2) > r2)))\,\| \\ (((Type = A3)\,\&\,\&((ln + r3) \geq l2)\,\&\,\&((ln + l1) > r1)))\,\| \\ (((Type = A4)\,\&\,\&((ln + l1) > r1)\,\&\,\&(ln + l2) > r2)) \end{array} \right), \tag{9}$$

where $l1$ — distance from point q to point Node.v1; $l2$ — distance from point q to point Node.v2.

Additional checks for intersections of the search area with parent nodes for Node at all higher levels of the tree is the center of the proposed modification — the algorithm DIHPN (Fig. 1). The algorithm DIHPN explicitly recreates the hierarchy of search rules, which allows you to set narrower boundaries of nodes and avoid overlapping nodes of the same level with each other. We present a generalized dependence of the number of overlapping nodes on the number of the current level k and the number of processed higher levels t:

$$\mathrm{NIL}(k,\ t) = (B^{k+1-t})^2. \tag{10}$$

Based on formula (10), we concluded that an increase in the number of processed higher-level parents (parameter t) leads to a decrease in the number of intersecting nodes in the entire tree. In the limiting case of the formula (10), when the algorithm requests all the higher parents for the current node, the number of intersecting nodes is equal to one (the root). Because no two nodes of the same level intersect each other due to the architectural.

The disadvantage of the DIHPN algorithm is the increase in the number of calculated distances between two objects. Determining the distances from the query point to the vantage points of the parent nodes is the reason for the increase in the computational complexity of the proposed algorithm. We can solve the problem by storing the previously calculated results in additional memory L in order to avoid repeated calculations.

As part of improving the efficiency of the DIHPN algorithm, we need to solve several problems:

1) the problem of effectively determining the distances from the query point q to the subtrees of node $A1$, $A2$, $A3$, $A4$. Note that based on the distance to the query point q, the priority of the node in queue C is calculated. To improve the efficiency of distance determination, the authors developed the algorithm: determination distance from request point to node area (DDRPNA);

2) the problem of increasing the efficiency of determining the fact of intersection of nodes with the search area [formula (1)]. Determining the fact of intersection is the basis of the procedure for adding a node to the priority queue C. We propose to modify the classic rules of intersection of MVP-tree regions [formula (9)] based on the use of minimum distances: modification of the rules for determining the intersection of nodes based on minimum distances (MRDINMD).

DDRPNA algorithm

To assign priorities that are more precise to nodes in queue C based on the distance from the query point q to the subtrees of the MVP-tree node, we suggest to use the author's DDRPNA algorithm. The problem appears by using only two reference points $\{v1,\ v2\}$ for four regions. Thus, the distances can only be determined to areas $A3$ and $A1/A2$. The choice of a specific second area depends on which region $A1$ or $A2$ the vantage point $v1$ belongs. Enter the following notation:

1) $AV1$ — a region of $A12 = A1 + A2$, which belongs to the vantage point $v1$;

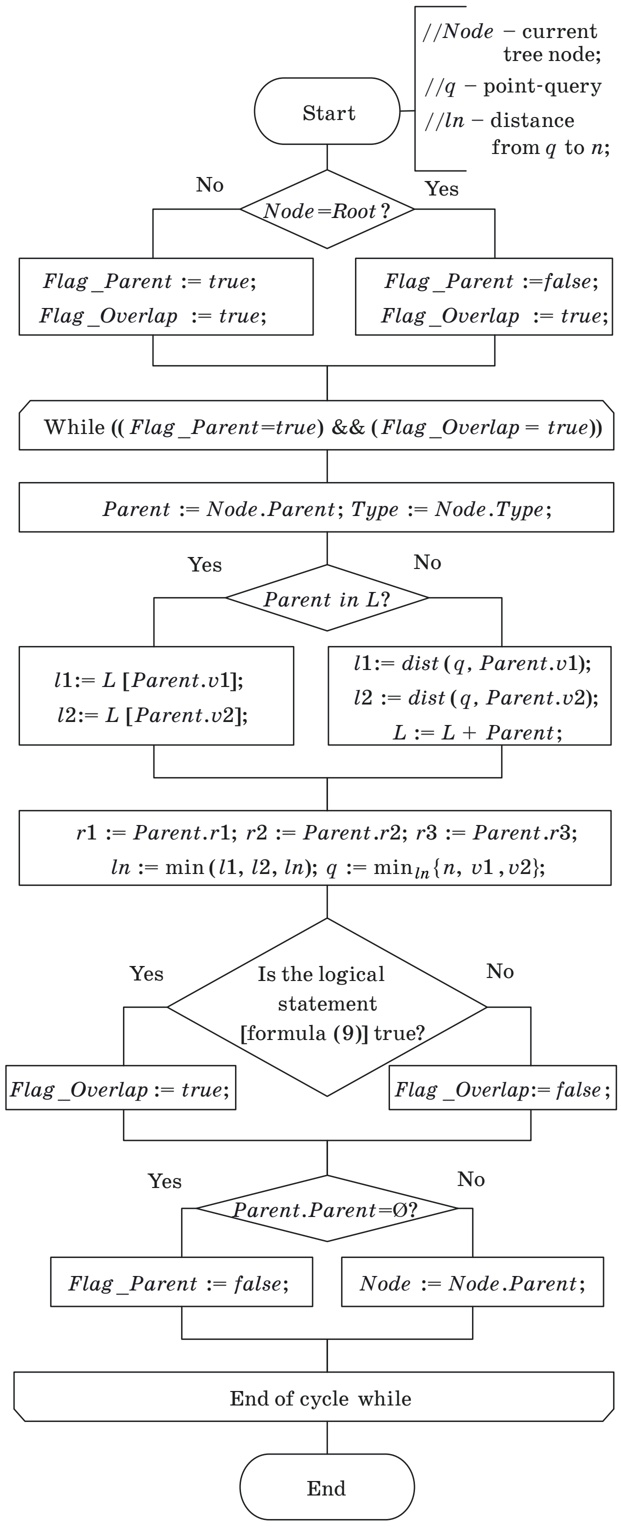
2) $AV2$ — the area of $A12$ that does not contain the reference point $v1$, i. e. $AV2 = A12 \cap AV1$.

To determine the distance from the query point q to the subtree Area of the node Node, the authors propose to use the following algorithm:

1) if the subtree Area is an area Node.AV1 and an internal node of the tree:

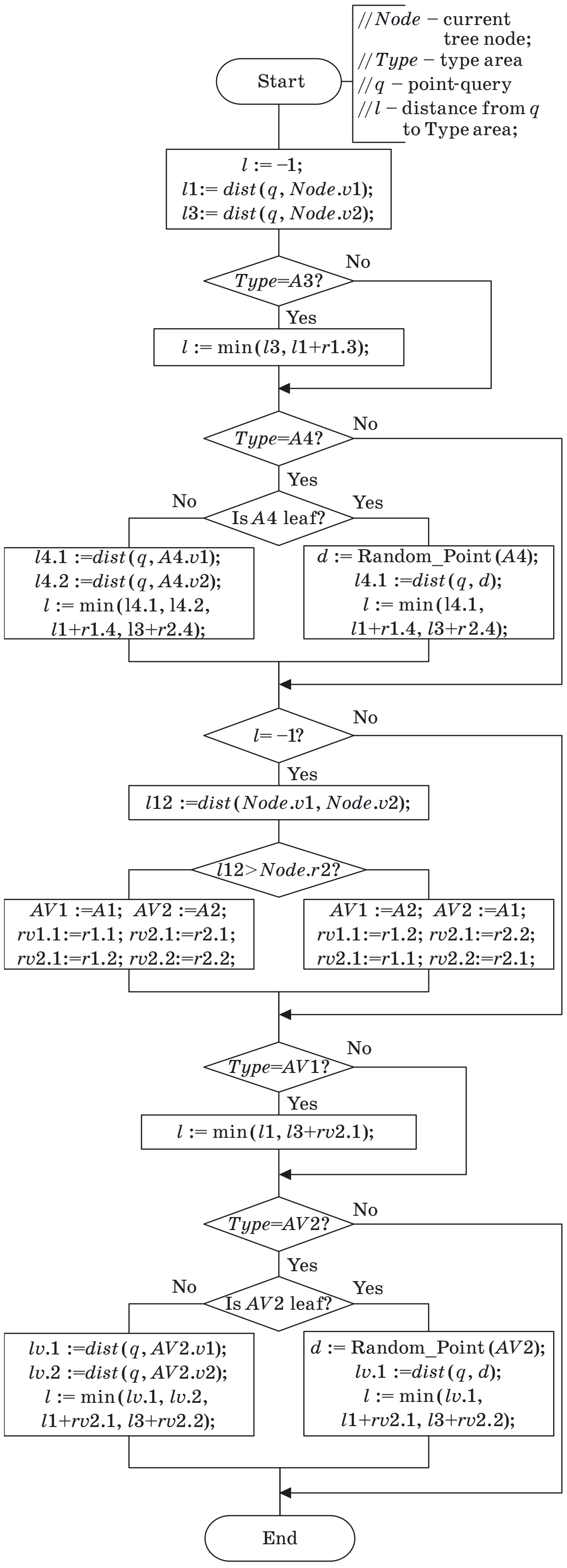
— determine the distance $l1.1$ from the query point q to the primary vantage point of the node Node.v1;

— set the desired distance $l1$ as the distance $l1.1$;



//Node – current tree node;
//q – point-query
//ln – distance from q to n;
Start
Node=Root?
No
Yes
Flag_Parent := true; Flag_Overlap := true;
Flag_Parent := false; Flag_Overlap := true;
While ((Flag_Parent=true) && (Flag_Overlap = true))
Parent := Node.Parent; Type := Node.Type;
Parent in L?
Yes
No
l1:= L [Parent.v1]; l2:= L [Parent.v2];
l1:= dist (q, Parent.v1); l2 := dist (q, Parent.v2); L := L + Parent;
r1 := Parent.r1; r2 := Parent.r2; r3 := Parent.r3;
ln := min (l1, l2, ln); q := min_ln{n, v1, v2};
Is the logical statement [formula (9)] true?
Yes
No
Flag_Overlap := true;
Flag_Overlap:= false;
Parent.Parent=Ø?
Yes
No
Flag_Parent := false;
Node := Node.Parent;
End of cycle while
End

■ ***Fig. 1.*** Flow chart of DIHPN algorithm



//Node – current tree node;
//Type – type area
//q – point-query
//l – distance from q to Type area;
Start
l := –1; l1:= dist (q, Node.v1); l3:= dist (q, Node.v2);
Type=A3?
No
Yes
l := min (l3, l1+r1.3);
Type=A4?
No
Yes
Is A4 leaf?
No
Yes
l4.1 :=dist (q, A4.v1); l4.2 :=dist (q, A4.v2); l := min (l4.1, l4.2, l1+r1.4, l3+r2.4);
d := Random_Point (A4); l4.1 :=dist (q, d); l := min (l4.1, l1+r1.4, l3+r2.4);
l= –1?
No
Yes
l12 :=dist (Node.v1, Node.v2);
l12>Node.r2?
AV1 :=A1; AV2 :=A2; rv1.1:=r1.1; rv2.1:=r2.1; rv2.1:=r1.2; rv2.2:=r2.2;
AV1 :=A2; AV2 :=A1; rv1.1:=r1.2; rv2.1:=r2.2; rv2.1:=r1.1; rv2.2:=r2.1;
Type=AV1?
No
Yes
l := min (l1, l3+rv2.1);
Type=AV2?
No
Yes
Is AV2 leaf?
No
Yes
lv.1 :=dist (q, AV2.v1); lv.2 :=dist (q, AV2.v2); l := min (lv.1, lv.2, l1+rv2.1, l3+rv2.2);
d := Random_Point (AV2); lv.1 :=dist (q, d); l := min (lv.1, l1+rv2.1, l3+rv2.2);
End

■ ***Fig. 2.*** Flow chart of DDRPNA algorithm

2) if the subtree Area is an area Node.AV2 and an internal node of the tree:
— determine the distance *l*2.1 from the query point *q* to the primary vantage point of the node Node.v1;
— calculate the distance *l*2.2 from the query point *q* to the secondary vantage point of the node Area.v2;
— select the minimum value among (*l*2.1, *l*2.2) as the desired distance *l*2;
3) if the subtree Area is an area Node.A3 and an internal node of the tree:
— determine the distance *l*3.1 from the query point *q* to the primary vantage point of the node Node.v2;
— set the desired distance *l*3 as the distance *l*3.1;
4) if the subtree Area is an area Node.A4 and an internal node of the tree:
— determine the distance *l*4.1 from the query point *q* to the primary vantage point of the node Area.v1;
— calculate the distance *l*4.2 from the query point *q* to the secondary vantage point of the node Area.v2;
— select the minimum value among (*l*4.1, *l*4.2) as the desired distance *l*4;
5) if the subtree Area is a leaf of the tree:
— select a random point *d* from the Area;
— set the value of the desired distance as the distance from the query point *q* to the point *d*.

To increase the efficiency of the algorithm, we propose to additionally calculate and use the minimum distances between the vantage points {*v*1, *v*2} and the subtrees of the node. Denote by *ri*, *j* the minimum distance between the *i*-th vantage point and the *j*-th area {*AV*1, *AV*2, *A*3, *A*4} of the current tree node. Then, for the regions *AV*1, *AV*2, *A*3 and *A*4, it is possible to determine more precise distances to the query point *q*:
1) for *AV*1, find the minimum among (*l*1, *l*3 + *r*2.1);
2) for *AV*2, find the minimum among (*l*2, *l*1 + r1.2, *l*3 + *r*2.2);
3) for *A*3, find the minimum among (*l*3, *l*1 + *r*1.3);
4) for *A*4, find the minimum among (*l*4, *l*1 + *r*1.4, *l*3 + *r*2.4).

We have implemented the proposed ideas in the DDRPNA algorithm (Fig. 2). The architectural changes associated with the DDRPNA algorithm in the MVP-tree (*Y**B* = 8 minimum distances at each node) increase the amount of memory for storing the indexing tree.

Modification of the node intersection rules MRDINMD

Another problem of the DIHPN algorithm is the problem of efficiently determining the intersection of the search area with the subtrees of the current node. The less intersections of the search area with the nodes of the MVP-tree, the higher the search efficiency. This problem can be solved by using the previously introduced minimum distances *ri*, *j*.

Studies [32] proposed a method to improve the efficiency of determining the intersection of the search area and nodes for a PM-tree. The features of the proposed method are:
1) using both minimum and maximum distances from vantage points to areas of the current node;
2) set of vantage points is global and common to all nodes of the tree.

The method proposed in [32] for determining the intersection of the search area and tree nodes by the authors is adapted for the MVP-tree.

The features of the adaptation are:
1) the use of only a set of minimum distances from the vantage points to the areas of the tree node;
2) using local vantage points for each node of the tree instead of a set of global control points.

The modification of the classical MVP-tree area intersection rules [formula (9)] based on the minimum distances MRDINMD is represented as

$$overlap(Node,\ (q,\ ln)) = \begin{cases} A1: (ln+r1) \ge l1\ and\ (ln+r2) \ge l2\ and\ (ln+l1) > r1.1\ and\ (ln+l2) > r2.1 \\ A2: (ln+r1) \ge l1\ and\ (ln+l2) > r2\ and\ (ln+l1) > r1.2\ and\ (ln+l2) > r2.2 \\ A3: (ln+r3) \ge l2\ and\ (ln+l1) > r1\ and\ (ln+l1) > r1.3\ and\ (ln+l2) > r2.3 \\ A4: (ln+l1) > r1\ and\ (ln+l2) > r2\ and\ (ln+l1) > r1.4\ and\ (ln+l2) > r2.4 \end{cases}. \qquad (11)$$

The developed modification MRDINMD by the authors replaces the classic rules of intersection of areas for MVP-tree in search algorithms and DIHPN (see Fig. 1).

The methods obtained during the research for reducing the number of calculated distances to improve the search efficiency in the MVP-tree have a theoretical basis [formulas (10) and (11)]. Thus, the purpose of the authors' practical research is to evaluate and analyze the performance gain of the modified search in the MVP-tree compared to the classical one, depending on the dimension of the space and the number of objects in the data.

Experiments

We have carried out an experimental evaluation of the efficiency of the proposed algorithms in the context of the nearest neighbor search problem. During the experiments, we tested: 1) algorithm DIHPN; 2) combination of DIHPN and DDRPAN; 3) combination of DIHPN + DDRPNA + MRDINMD algorithms.

Computer characteristics as a part of experiments: 64-bit Windows 7 operating system, Intel i5-2500K processor, 16 GB RAM. We wrote the program in Java and compiled it using the NetBeans IDE. We use the following MVP-tree parameters: 1) $U = 2$; 2) $Y = 2$; 3) $B = 4$; 4) $K = 1$; 5) $P = 0$;

Four data sets were selected as experimental data from repository ANN-Benchmarks [33]: 1) Last.fm (65 number of attributes); 2) SIFT (128 number of attributes); 3) MNIST (784 number of attributes); 4) GIST (960 number of attributes). Note that we have divided each dataset into two data categories: a set of SQ queries to find the nearest neighbor and the original set of objects.

Within the experiment, the MVP-tree has the following features: 1) we use the Euclidean distance as the distance metric; 2) we normalized each dimension of the data sets in the range [0, 1]; 3) we use a random selection heuristic to select the vantage points. We repeated each experiment 3 times, after which we averaged the results.

A measure of the computational complexity of the tested algorithm is the absolute number of operations (NumberDistances) for calculating the distances between two points during a search for a SQ set. To simplify understanding, all the results of the tested algorithms were normalized relative to the basic search algorithm. Based on the stated goal of the research, we have chosen the classical search algorithm MVP-tree [29] as the basic algorithm for comparison. We calculate the effectiveness of the proposed algorithms using the formula:

$$\mathrm{Eff}_{\mathrm{ALG}} = (\mathrm{NumberDistances}_{\mathrm{ALG}} / \mathrm{NumberDistances}_{\mathrm{MVP}}) \times 100\%, \qquad (12)$$

where $\mathrm{Eff}_{\mathrm{ALG}}$ — the efficiency of the tested algorithm ALG; $\mathrm{NumberDistances}_{\mathrm{ALG}}$ — the number of operations for calculating the distances between two points during the search for the SQ set of the algorithm ALG; $\mathrm{NumberDistances}_{\mathrm{MVP}}$ — the number of calculated distances between two points for the set SQ using the classical search algorithm MVP-tree.

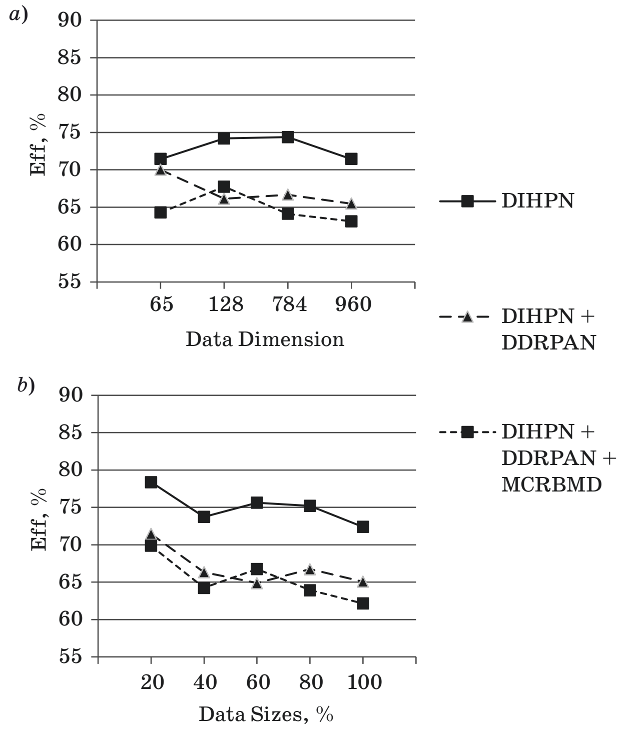
Thus, we obtained all the results of the work based on measuring and normalizing the absolute number of calculated distances between two objects (NumberDistances). Therefore, the evaluation of the efficiency of the tested algorithms does not depend on the programming language, the development environment and technical characteristics of computing resources.

We used the following data set parameters for the experiment: 1) the original set of 60 000 objects, the number of attributes in the object is 65, 128, 784 and 960; 2) a corresponding set of SQ queries of 3000 objects.

We presented two summary graphs: 1) the graph of the computational complexity of search algorithms for different data dimensions (the number of attributes), shown in Fig. 3, *a*; 2) the graph of the computational complexity of search algorithms for different original data sizes (the number of objects), which is shown in Fig. 3, *b*.

Based on the analysis of Fig. 3, we were made the following conclusions for the algorithms developed by the authors to reduce the number of computations for the nearest neighbor search in the MVP-tree: 1) the proposed algorithms increase the search performance in both small and large space dimensions; 2) with the growth of the data dimension (the number of attributes), the efficiency of the developed search algorithms increases; 3) with the growth of the number objects of data, the efficiency of the applied algorithms increases.

Thus, the results of the experiment confirm the effectiveness of the author's solutions for nearest neighbors searching in big data with a significant number of attributes. The proposed theoretical al-



■ ***Fig. 3.*** Efficiency of different data dimensions (*a*) and sizes (*b*) gorithms have found application in an applied classification system for solving pattern recognition problems based on precedent methods. The developed algorithms increase the computational performance of the nearest neighbor search procedure in the indexed set of original objects.

Conclusion

We presented algorithmic solutions related to the adaptation of the classical search algorithm to the priority queue of node processing, sorted by increasing the distances from the query point to the nodes.

The proposed solutions allow to:

— eliminate additional calculations when using the priority queue for processing nodes by reducing the number of intersecting nodes of the same level;

— identify and generalize the theoretical dependence of the number of intersections on the complexity of the search calculations (the number of additional processed levels);

— improve the efficiency of the priority queue for processing nodes by more accurately determining the distances from the query point to the subtrees of the tree node;

— use additional minimum distances between the vantage points and the subtrees of the node, which improves the search algorithm;

— reduce search time by reducing the number of processed nodes based on the use of minimum distances in the procedure for determining the intersection of the search area with the area of the tree node.

The developed algorithms of search procedures in tree structures have demonstrated promising results of their application to the class of data indexing problems in metric space.

A limitation of the proposed algorithms is the heuristic of random selection of vantage points. The use of random selection heuristics does not guarantee stable results for the same data. As further improvements to the search algorithms, we are planning to use combinations of greedy heuristics when selecting vantage points.

References

1. Fomin V. V., Duke V. A., Aleksandrov I. V. The use of machine learning methods for the determination of the fuel consumption of a gas turbine frigate. *Marine Intellectual Technologies*, 2019, vol. 4, no. 4(42), pp. 197–201.
2. Nielsen F., Piro P., Barlaud M. Bregman vantage point trees for efficient nearest Neighbor Queries. *2009 IEEE International Conference on Multimedia and Expo*, 2009, pp. 878–881. doi:10.1109/ICME.2009.5202635
3. Oliveira P. H., Traina C., Kaster D. S. CLAP, ACIR and SCOOP: Novel techniques for improving the performance of dynamic Metric Access Methods. *Information Systems*, 2017, vol. 72, pp. 117–135. doi:10.1016/j.is.2017.10.003
4. Zabot G. F., Cazzolato M. T., Scabora L. C., Traina A. J. M., Traina Jr. C. Efficient indexing of multiple metric spaces with spectra. *2019 IEEE International Symposium on Multimedia (ISM)*, 2019, pp. 169–1697. doi:10.1109/ISM46123.2019.00038
5. Calistru C. M., Ribeiro M. C., David G. T. Flexible descriptor indexing for multimedia databases. *Proceedings of the Fourth International Workshop on Content-Based Multimedia Indexing*, 2005, pp. 1–8.
6. Sehili Z., Rahm E. Speeding up privacy preserving record linkage for metric space similarity measures. *Datenbank-Spektrum*, 2016, vol. 16(3), pp. 227–236. doi:10.1007/s13222-016-0222-9
7. Chávez E., Ludueña V., Reyes N., Roggero P. Faster proximity searching with the distal SAT. *Information Systems*, 2016, vol. 59, pp. 15–47. doi:10.1007/978-3-319-11988-5_6
8. Hanyf Y., Silkan H., Labani H. Criteria and technique to choose a good ρ parameter for the D-index. *Intelligent Systems and Computer Vision (ISCV)*, 2015, pp. 1–6. doi:10.1109/ISACV.2015.7106169
9. Almeida J., Torres R. D. S., Leite N. J. BP-tree: An efficient index for similarity search in high-dimensional metric spaces. *Proceedings of the 19th ACM International Conference on Information and Knowledge Management*, 2010, pp. 1365–1368. doi:10.1145/1871437.1871622
10. Pola I. R. V., Traina Jr. C., Traina A. J. M. The MM-tree: a memory-based metric tree without overlap between nodes. *Proceedings of the 11th East European Conference on Advances in Databases and Information Systems*, 2007, pp. 157–171. doi:10.1007/978-3-540-75185-4_13
11. Carelo C. C. M., Pola I. R. V., Ciferri R. R., Traina A. J. M., Traina C. T., de Aguiar Ciferri C. D. Slicing the metric space to provide quick indexing of complex data in the main memory. *Information Systems*, 2011, vol. 36(1), pp. 79–98. doi:10.1016/j.is.2010.06.004
12. Guhlemann S., Petersohn U., Meyer-Wegener K. Reducing the distance calculations when searching an M-Tree. *Datenbank-Spektrum*, 2017, vol. 17(2), pp. 155–167. doi:10.1007/s13222-017-0258-5
13. Kokotinis I., Kendea M., Nodarakis N., Rapti A., Sioutas S., Tsakalidis A. K., Tsolis D., Panagis Y. NSM-tree: Efficient indexing on top of NoSQL databases. *International Workshop of Algorithmic Aspects of Cloud Computing*, 2016, pp. 3–14. doi:10.1007/978-3-319-57045-7_1

14. Donkó I., Szalai-Gindl J. M., Gombos G., Kiss A. An implementation of the M-tree index structure for PostgreSQL using GiST. *2019 IEEE 15th International Scientific Conference on Informatics*, 2019, pp. 189–194. doi:10.1109/Informatics47936.2019.9119265
15. Chen L., Gao Y., Zheng B., Jensen C. S., Yang H., Yang K. Pivot-based metric indexing. *Proceedings of the VLDB Endowment*, 2017, vol. 10(10), pp. 1058–1069. doi:10.14778/3115404.3115411
16. Maliki N., Silkan H., Maghri M. Efficient indexing and similarity search using the Geometric Near-neighbor Access Tree (GNAT) for face-images data. *Procedia Computer Science*, 2019, vol. 148, pp. 600–609. doi:10.1016/j.procs.2019.01.033
17. Razente H. L., Sousa R. M. S., Barioni M. C. N. Metric indexing assisted by short-term memories. *Similarity Search and Applications*, 2018, vol. 11223, pp. 107–121. doi:10.1007/978-3-030-02224-2_9
18. Chen L., Gao Y., Li X., Jensen C. S., Chen G. Efficient metric indexing for similarity search and similarity joins. *IEEE Transactions on Knowledge and Data Engineering*, 2017, vol. 29(3), pp. 556–571. doi:10.1109/TKDE.2015.2506556
19. Li S., Yu H., Yuan L. A Novel approach to remote sensing image retrieval with multi-feature VP-tree indexing and online feature selection. *IEEE Second International Conference on Multimedia Big Data*, 2016, vol. 1, pp. 133–136. doi:10.1109/BigMM.2016.11
20. Jiang D., Sun H., Yi J., Zhao X. The research on nearest neighbor search algorithm based on vantage point tree. *2017 8th IEEE International Conference on Software Engineering and Service Science*, 2017, pp. 354–357. doi:10.1109/ICSESS.2017.8342931
21. Su Y., Zhang Y., Wan C., Yu G. GDPC: A GPU-accelerated density peaks clustering algorithm. *Database Systems for Advanced Applications*, 2020, vol. 1, pp. 305–313. doi:10.1007/978-3-030-59410-7_21
22. Cheng H., Yang W., Tang R., Mao J., Luo Q., Li C., Wang A. Distributed indexes design to accelerate similarity based images retrieval in airport video monitoring systems. *Proceedings of the 12th International Conference on Fuzzy Systems and Knowledge Discovery*, 2015, pp. 1908–1912. doi:10.1109/FSKD.2015.7382239
23. Lin Y. Q., Fu Y. G., Su Q., Wang Y. M., Gong X. T. A rule activation method for extended belief rule base with VP-tree and MVP-tree. *Journal of Intelligent & Fuzzy Systems*, 2017, vol. 33(6), pp. 3695–3705. doi:10.3233/JIFS-17521
24. Figueroa K., Paredes R., Ibarrola J. A. C., Reyes N. *Fixed height queries tree permutation index for proximity searching*. In: *Pattern Recognition. MCPR 2017*. Carrasco-Ochoa J., Mart nez-Trinidad J., Olvera-L pez J. (eds). Lecture Notes in Computer Science, Springer, Cham, 2017. Vol. 10267. Pp. 74–83. doi:10.1007/978-3-319-59226-8_8
25. Mao R., Liu W., Iqbal Q., Miranker D. P. On index methods for an image database. *ACM International Workshop on Multimedia Databases*, 2003, pp. 1–9.
26. Boytsov L., Nyberg E. Pruning algorithms for low-dimensional non-metric k-NN search: A Case study. *Proceedings of the 12th International Conference on Similarity Search and Applications*, 2019, pp. 72–85. doi:10.1007/978-3-030-32047-8_7
27. Pandey V., Renen A., Kipf A., Kemper A. How good are modern spatial libraries? *Data Science and Engineering*, 2021, vol. 6(11), pp. 192–208. doi:10.1007/s41019-020-00147-9
28. Liu S., Wei Y. Fast nearest neighbor searching based on improved VP-tree. *Pattern Recognition Letters*, 2015, vol. 60, pp. 8–15. doi:10.1016/j.patrec.2015.03.017
29. Bozkaya T., Ozsoyoglu M. Distance-based indexing for high-dimensional metric spaces. *Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data*, 1997, pp. 357–368. doi:10.1145/253262.253345
30. Hjaltason G. R., Samet H. Index-driven similarity search in metric spaces. *ACM Transactions on Database Systems*, 2002, vol. 28, no. 4, pp. 517–580. doi:10.1145/958942.958948
31. Fomin V., Aleksandrov I., Gallyamov D., Kirichek R. Modified indexing algorithm based on priority queue in metric space for MVP tree. *The 4th International Conference on Future Networks and Distributed Systems (ICFNDS 2020)*, Saint-Petersburg, Russia, November 26–27, 2020, 8 p. doi:10.1145/3440749.3442617
32. Humberto L. R., Maria C. N. B. Storing data once in M-tree and PM-tree. *12th International Conference on Similarity Search and Applications SISAP*, 2019, pp. 18–31. doi:10.1007/978-3-030-32047-8_2
33. *Erik Bernhardsson — Ann-Benchmarks*. Available at: https://github.com/erikbern/ann-benchmarks (accessed 15 July 2021).

УДК 004.93
doi:10.31799/1684-8853-2021-4-18-27

Алгоритм индексации, основанный на хранении дополнительных расстояний в метрическом пространстве, в рамках структуры дерева множества опорных точек

В. В. Фомин[а], доктор техн. наук, профессор, orcid.org/0000-0001-7040-5386, v_v_fomin@mail.ru
И. В. Александров[б], аспирант, orcid.org/0000-0002-6258-8191
[а]Российский государственный педагогический университет им. А. И. Герцена, набережная р. Мойки, 48, Санкт-Петербург, 191186, РФ
[б]Санкт-Петербургский государственный университет телекоммуникаций им. проф. М. А. Бонч-Бруевича, Большевиков пр., 22-1, Санкт-Петербург, 193232, РФ

Введение: парадигма поиска по сходству применяется в различных вычислительных задачах, таких как классификация, интеллектуальный анализ данных, распознавание образов и др. В настоящее время среди алгоритмов поиска значительное место занимает технология древовидных метрических методов доступа. Классическая проблема сокращения времени поиска по сходству в метрическом пространстве является актуальной для современных систем при обработке больших сложных данных. Ввиду многоаспектности проблемы эффективности поисковых алгоритмов локальные исследования в этом направлении востребованы и продолжают приносить полезные результаты. **Цель:** снизить вычислительную сложность алгоритмов древовидного поиска в задачах, использующих метрическую близость. **Результаты:** разработан алгоритм поиска для структуры данных в виде дерева множества опорных точек, основанный на приоритетной очереди обработки узлов; математически формализованы проблемы дополнительных вычислений и способы их решения. Для повышения быстродействия поиска по сходству предложены процедуры формирования приоритетной очереди обработки узлов и уменьшения количества пересечений узлов одного уровня. Повышение эффективности происходит на основе изменения древовидной структуры данных и использования минимальных расстояний между опорными точками и поддеревьями узла. Уменьшение числа вычислений достигается за счет более точного определения расстояния до узлов от искомого объекта и факта пересечения области поиска с узлом дерева. **Практическая значимость:** полученным алгоритмам поиска требуется меньше времени для обработки информации за счет несущественного повышения требований к памяти. Снижение времени обработки информации расширяет границы применения древовидных метрических методов индексации в задачах поиска в больших массивах данных.

Ключевые слова — индексация сложных данных, древовидные структуры данных, метрические методы доступа, приоритетная очередь узлов, дерево множества опорных точек.

Для цитирования: Fomin V. V., Aleksandrov I. V. Indexing algorithm based on storing additional distances in metric space for multi-vantage-point tree. *Информационно-управляющие системы*, 2021, № 4, с. 18–27. doi:10.31799/1684-8853-2021-4-18-27
For citation: Fomin V. V., Aleksandrov I. V. Indexing algorithm based on storing additional distances in metric space for multi-vantage-point tree. *Informatsionno-upravliaiushchie sistemy* [Information and Control Systems], 2021, no. 4, pp. 18–27. doi:10.31799/1684-8853-2021-4-18-27