Categorical Range Queries in Large Databases

Alexandros Nanopoulos[1] and Panayiotis Bozanis[2]

[1] Dept. Informatics, Aristotle University
Thessaloniki, Greece
alex@delab.csd.auth.gr
[2] Dept. Computer Eng. and Telecom, University of Thessaly
Volos, Greece
pbozanis@inf.uth.gr

Abstract. In this paper, we introduce the categorical (a.k.a. chromatic) range queries (CRQs) in the context of large, disk-resident data sets, motivated by the fact that CRQs are conceptually simple and emerge often in DBMSs. On the basis of spatial data structures, and R-trees in particular, we propose a multi-tree index that follows the broad concept of augmenting nodes with additional information to accelerate queries. Augmentation is examined with respect to maximal/minimal points in subtrees, the properties of which are exploited by the proposed searching algorithm to effectively prune the search space. Detailed experimental results, with both real and synthetic data, illustrate the significant performance gains (up to an order of magnitude) due to the proposed method, compared to the regular range query (followed by the filtering w.r.t. categories) and to a naive R-tree augmentation method.

1 Introduction

Range queries, that find all tuples intersected (covered, etc.) by a query region, are commonplace in all kinds of database systems today. In a large number of applications, however, database objects can come aggregated in (disjoint) groups. Therefore, what becomes of interest is range queries searching for groups (instead of individual objects themselves) intersected by the query region. This type of query is denoted as *Categorical Range Query* (CRQ)[1] and the attribute that the grouping is based upon is called *categorical attribute*. CRQs have been comprehensively studied in the research fields of computational geometry and main-memory data structures [2, 9, 10, 11, 17, 18, 20, 23]. In contrast, much less attention has been given to supporting CRQs in large databases. Nevertheless, CRQs arise in many DBMSs. For instance: consider a set of locations, where each one is associated with its spatial coordinates and its soil type (a common case of thematic layers in GIS). A typical CRQ is to find all soil types of locations that are within a query window.

[1] Another common name is *Chromatic Range Query*, which corresponds to the same acronym.

T. Hadzilacos et al. (Eds.): SSTD 2003, LNCS 2750, pp. 122–139, 2003.

CRQs can be easily specified in any SQL-like language, including SQL extensions for spatial DBMSs. Nevertheless, the problem of query optimization for CRQs should be also taken into account. Despite their conceptual simplicity, CRQs present requirements for query optimization that are not addressed by current DBMSs. Considering the existing processing techniques, a CRQ will be processed by first executing the *regular range query*, i.e., the finding of individual objects (not categories) satisfying the range condition, followed by filtering its output set to select the distinct categories. The aforementioned approach does not take into account the difference between the selectivity (i.e., size of output set) of the CRQ and the one of the regular range query. In several real applications data tend to belong to categories and to be clustered along the categorical attributes. As a result, the selectivity of the CRQ can be much larger (i.e., smaller output size) than the one of the regular range query. By first processing the plain range query and then filtering with respect to the categorical attribute, a lot of cost is spent. Moreover, the domain size of the categorical attribute (i.e., the number of all its possible values) may be large enough, e.g., several hundreds or even thousands. This is prohibitive for a quick-fix solution that partitions objects based on their category (e.g., using independent indexes) and processes them separately.

The previously discussed issues have also been taken into account by approaches in computational geometry and main-memory data structures. Nevertheless, they have payed little attention to secondary memory, giving emphasis on worst-case asymptotic performance and requiring significant storage redundancy [19]. For large databases on secondary storage, high redundancy is prohibitive due to large space and update time overhead.

In this paper, we introduce the problem of categorical range queries in the context of large databases. Since CRQs find important applications in spatial data (e.g., GIS), we concentrate on this field, focusing, in such a way, on a concrete application framework. On the basis of spatial data structures, we develop a multi-tree index that integrates in an efficient way the spatial dimensions and the categorical attribute. This approach is based on the broad concept of the augmentation of nodes with additional information to accelerate queries [26]. The paper makes the following technical contributions:

- We develop novel techniques for spatial data structures and R-trees in particular (because they have been included in several commercial DBMSs [21]), which address the efficient query processing of CRQs in large databases.
- We provide a detailed experimental study, comparing the proposed method with two base-line algorithms: (i) the regular range query, and (ii) a naive method of incorporating the categorical attribute in the R-tree nodes. The results illustrate the significant gains (up to an order of magnitude) due to the proposed method.
- As another contribution, we discuss the issue of developing estimators for the selectivity of CRQs.

The rest of this paper is organized as follows. In Section 2 we give the related work. Section 3 describes the proposed approach, whereas Section 4 contains the

experimental results. Finally, Section 5 presents the conclusions and directions of future work.

2 Related Work

Jarardan and Lopez [20] introduced this new type of problems that are of significant theoretical interest and rich in applications (e.g., document retrieval, internet packet categorization). Solutions for standard problems yield output-insensitive solutions for their generalized chromatic counterparts. For this reason they attracted a lot of research [2, 9, 10, 17, 18, 14, 23, 24]. These papers present a number of efficient, mainly theoretical, algorithms that are tailored to the main-memory context. Disk-resident data were considered in [15] for specific chromatic problems that are reductions of problems related to the indexing of 2-dimensional strings. However, [15] simply uses regular range queries, which are sufficient in their context, since the distribution of the resulting (transformed) data sets guarantee singleton categories with very high probability (greater than 0.9). As it will be described in the following, in the context of spatial data this assumption may not hold, and the regular range query can present significant limitations in several cases.

[17, 20] provided a uniform framework that yields efficient solutions for chromatic problems on iso-oriented objects. Their techniques consist of geometric transformations of generalized problems into equivalent instances of some standard problems and the use of persistence as a method for imposing range restrictions to static problems. New improved bounds, as also extensions to the non-iso-oriented geometrical objects case, were given in [9, 18, 11], whereas [10] treated the red-blue categorical reporting problem (we are given a set Q_1 of "red" categorical objects and a set Q_2 of "blue" ones and we want to report all intersecting pairs between "red" and "blue" objects). Recently, Agarwal et al. [2] presented solutions for chromatic objects with grid co-ordinates. [15, 24] considered the applications of colors in document retrieval and [14] studied the chromatic queries in the case of internet packet categorization. Also, approximate colored nearest neighbor search queries were studied in [23].

Finally, loosely related to this work can be regarded papers considering either the adaptation of main-memory data-structuring techniques to the disk (I/O) context (see, for example, [4, 5, 12, 16]) or following the node augmentation paradigm to solve geometric intersection query problems (e.g., [27, 29]).

3 Proposed Method

In this section, we describe our solution to the problem of processing CRQs in large databases. Since a number of commercial database systems have developed spatial extensions and R-tree indexes for the management of spatial data, the usefulness of any solution is increased if it is based on this infrastructure.

3.1 Maximal/Minimal Points

Assuming that the points of a data set are indexed with an R-tree, a straightforward method to augment its nodes with information about categories is to store in the entries of intermediate nodes all categories that are present in the corresponding subtrees (using a bitmap, where each bit position corresponds to a category). Let q be a query rectangle. During a CRQ, we maintain an array A indicating the presence or absence of each category in rectangle q; initially, all categories are absent. Let also e be an entry in an intermediate node N. If q intersects e.MBR, then we find, through the bitmap of e, the categories that are present in the subtree of e. If the subtree contains any categories that are still absent from the result, then we descent to search in the subtree. Otherwise, we can prune the searching to this subtree. When reaching a leaf, we can determine the categories of its points that are covered by q, and, therefore, we can update A. Notice that analogous methods have been used in structures for regional data, e.g., quadtrees [22].

The aforementioned approach is simple and requires the addition of only one bitmap to each entry of intermediate nodes. It can avoid the searching to subtrees that are not going to contribute to the result, nevertheless it has the disadvantage that it can determine the actual existence of categories in the query rectangle only at the leaves. Due to this fact, one may not expect significant reductions in the I/O overhead, because the searching has to reach the leaf level many times. Also, the corresponding bitmaps will tend to be saturated and may indicate the presence of most of the categories. Henceforth, we denote the aforementioned approach as Naively Augmented R-tree (NAR).

To be able to determine the presence/absence of categories within the query rectangle at the upper levels as well, solutions in main-memory data structures and computational geometry are based on indexing schemes that, at each entry e of internal nodes, allow for the replication of all points stored at the subtree of e. Evidently, in the context of secondary storage and R-trees, such assumptions lead to prohibitive space and time overhead [19]. We, therefore, require a solution between these two extremes, i.e., the NAR approach, which does not store any points at internal nodes, and the aforementioned approaches which need to store all points of the subtrees. For this reason, we focus on the *maximal/minimal* points.

Definition 1 (Domination). *Let $p_1 = (x_1, \ldots, x_d)$ and $p_2 = (y_1, \ldots, y_d)$ be two d-dimensional points. We define that p_1 dominates maximally p_2 (denoted as $Dom_{\max}(p_1, p_2)$), when $x_i > y_i$, $\forall 1 \leq i \leq d$. We also define that p_1 dominates minimally p_2 (we denote $Dom_{\min}(p_1, p_2)$), when $x_i < y_i$, $\forall 1 \leq i \leq d$.*

Definition 2 (Maximal/Minimal Points). *Let S be a set of points. A point $p_i \in S$ is maximal (minimal, resp.), if there does not exist any other point $p_j \in S$ such that $Dom_{\max}(p_j, p_i)$ ($Dom_{\min}(p_j, p_i)$, respectively).*

Figure 1.a illustrates an example of a set of two-dimensional points and the corresponding maximal/minimal points. If the set S contains n points, then the

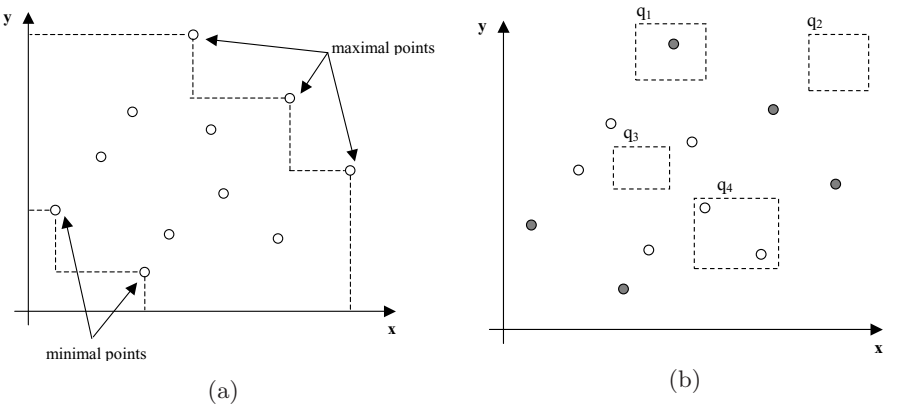


Fig. 1. Example of: (a) maximal/minimal points, (b) the different cases

calculation of maximal/minimal points can be done in $O(n \log n)$ time [8, 28]. Regarding the number of maximal (minimal, resp.) points, it is easy to see that in the worst case it is equal to n (for points placed in a diagonal line) and in the best case it is equal to one, i.e., a point at the upper-right (lower-left, resp.) corner. For the case of uniform placement, at random, of n points in a subset P of the plane, in [6] it was shown that the number of maximal points is normally distributed, with average value $\sim m\sqrt{\lambda}$, λ being the density and m a constant depending on P.[2]

For a set S of points, we denote as $S_{\max}$ and $S_{\min}$ the set of its maximal and minimal points respectively. $S_{\max}$ and $S_{\min}$ can be considered as a representation of S, which, given a rectangle q, allows for testing if q contains any points of S. Let q_{ll} and q_{ur} denote the lower-left and the upper-right corner of q. The following lemma is evident, since for each $p \in S_{\max}$ (equivalently, for each $p \in S_{\min}$) it also holds that $p \in S$.

Lemma 1. *If there exist a point p such that $p \in S_{\max}$ or $p \in S_{\min}$, and p is contained in a rectangle q, then q contains at least one point from S.*

For instance, see the case of rectangle q_1 in Figure 1.b, which contains a maximal point (maximal and minimal points are depicted as shaded). Due to the properties of maximal/minimal points, it is also easy to prove that:

Lemma 2. *If there does not exist any $p \in S_{\max}$ ($p \in S_{\min}$, resp.) such that $Dom_{\max}(p, q_{ll})$ ($Dom_{\min}(p, q_{ur})$, resp.), then there does not exist any point $s \in S$ such that s is contained in q.*

For an instance of the latter case regarding maximal points, see rectangle q_2 in Figure 1.b, where its lower-left corner dominates maximally all maximal points.

[2] The aforementioned results mainly focus on spaces with moderate dimensionality, which are the "target" of structures belonging to the R-tree family. In Section 5 we discuss the case of very high dimensionality.

In the cases of Lemmata 1 and 2, we get an exact answer to the question whether q contains any points of S. However, a third case exists, when none of the aforementioned conditions holds. Then, q may or may not contain any points from S. For instance, see rectangles q_3 and q_4 in Figure 1.b, where there exist points contained in q_4, but no point is contained in q_3. In order to avoid false-dismissals, we consider that q possibly contains points (evidently, this assumption may lead to a false-alarm, as it is depicted for the case of q_3). Therefore, from all previous cases, it follows that:

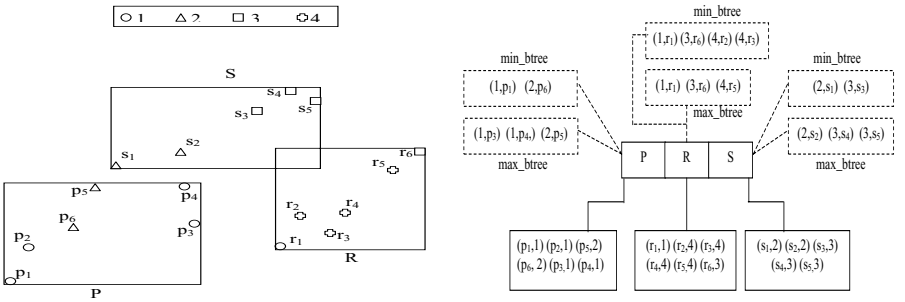
Theorem 1. *Given a set of points S and a rectangle q, $S_{\max}$ and $S_{\min}$ represent S without resulting to false-dismissals for the question whether q contains any points of S.*

The representation with the maximal/minimal points is between the two extreme cases (i.e., NAR and storing all points) and helps deciding about categories at internal nodes as well. Although in the worst case all points of a subtree may be maximal/minimal (leading to the second extreme), as explained, in the average case, only a small fraction of points are maximal/minimal. The details of how augmentation is applied with this type of representation are given in the following. Finally, we notice that the generalization for objects with spatial extent is immediate: after approximating them with iso-oriented geometrical objects (e.g., MBRs), one can work with the resulting upper and lower "corner" points; the choices can be made according to accuracy-time trade-offs.

3.2 The R-tree Augmentation Scheme

The proposed approach considers a multi-tree indexing scheme. It is based on a regular R-tree, which is augmented with auxiliary data structures that are maintained at each internal node entry. Each entry e in an internal R-tree node is of the form: $\langle e.\text{MBR}, e.\text{pointer}, e.\text{max\_btree}, e.\text{min\_btree}\rangle$, where e.max_btree and e.min_btree are pointers to the roots of B^+-trees that store the maximal and minimal, respectively, points in the subtree of e (the other two elements, e.MBR and e.pointer, have their regular meaning). It has to be noticed that each present category in the subtree has its own maximal/minimal points (i.e., maximality and minimality is determined only between points of the same category).

Within the B^+-trees, the maximal and minimal points are ordered according to their category values, which comprise the search-keys in each B^+-tree. Therefore, the two B^+-trees of a node entry e are probed with respect to category values, and the maximal and minimal points can be retrieved. Figure 2.a illustrates a sample data set (the description of categories is also depicted) and Figure 2.b the corresponding multi-tree index (R-tree nodes are depicted with solid line whereas the B^+-tree ones with dashed). The points in the leaves of the R-tree are stored along with their category, however, there is no ordering with respect to the category values. In contrast, in the B^+-tree nodes the entries are the maximal/minimal points and they are ordered with respect to their category value. For instance, in node P, the maximal points for category 1 are points p_3



(a) (b)

Fig. 2. (a) Example of a data set. (b) The corresponding augmented R-tree

and p_4, and for category 2 it is point p_5. For the same node, the minimal points are: p_1 for category 1 and p_6 for category 2.

Let Q be a query rectangle, where Q_{ll} and Q_{ur} denote its lower-left and upper-right corner. We maintain a boolean array $A[1..C]$, where C is the total number of categories (which, in general, is not constant). $A[i]$, $1 \leq i \leq C$, being true denotes that we need to search for category i in the currently visited node, otherwise we do not. The processing of a CRQ commences from the root of the R-tree, and initially all elements of A are set to true. When being at an internal node, we first find all its entries whose MBRs intersect Q. For each such entry e, we probe e.max_btree and e.min_btree to find the maximal and minimal points of all categories c for which $A[c]$ is true. According to the description in Section 3.1, we have to examine a set of cases for each such category c. The algorithm that processes CRQs with criteria based on the aforementioned cases is called M^{2}R (Maximal/Minimal R-tree) and is given in Figure 3 (the category of a point p is denoted as p.category, and the i-th entry of a node N is denoted as $N[i]$).

The categories included in Q are maintained in the global variable *outputSet* (which is assumed to be empty before the execution of M^{2}R). An array A is separately used for each node entry $N[i]$ (step 7 of M^{2}R), since the absence of a category in an entry does not induce the absence in the other entries as well. When descending to a lower level, the contents of array A are passed from the father to the child node through argument *FA*. However, step 13 affects all other entries of node N, since we do not need to search for this category any more. For this reason *FA* is updated at step 16. It has to be noticed that it is necessary to test, at steps 21 and 8, if a category has been included in the output set of the CRQ, due to the depth-search manner that the searching proceeds. (We have also tried a breadth-first variation, but it performed purely because it results to many false-alarms.)

3.3 Management of B$^+$-trees

Firstly, we estimate the overall space complexity of B$^+$-trees. As previously mentioned, in [6] was proven that in the case of uniform placement at random

Procedure $\mathrm{M}^2\mathrm{R}$(**Rect** Q, **Node** N, **Array** FA)
1. **if** N is leaf
2. **foreach** p in N such that Q contains p
3. outputSet $\leftarrow$ outputSet $\cup$ p.category
4. **else**
5. $SI = \{\mathsf{N}[i] \mid \mathsf{N}[i] \cap \mathsf{Q} \neq \emptyset\}$
6. **foreach** $\mathsf{N}[i] \in SI$
7. **Array** $A \leftarrow$ FA
8. **foreach** c such that $A[c]$ = **true and** $c \notin$ outputSet
9. $N^c_{\max}[i] = \{p$ in $\mathsf{N}[i]$.max_btree $\mid p$.category $= c\}$
10. $N^c_{\min}[i] = \{p$ in $\mathsf{N}[i]$.min_btree $\mid p$.category $= c\}$
11. **if** $N^c_{\max}[i] = \emptyset$ **and** $N^c_{\min}[i] = \emptyset$
12. $A[c] \leftarrow$ **false**
13. **else if** $\exists p \in N^c_{\max}[i] \cup N^c_{\min}[i]$ such that Q contains p
14. outputSet $\leftarrow$ outputSet $\cup$ c
15. $A[c] \leftarrow$ **false**
16. FA$[c] \leftarrow$ **false**
17. **else if** $(\nexists p \in N^c_{\max}[i] : Dom_{\max}(p, \mathsf{Q}_{\mathsf{ll}}))$**or**$(\nexists p \in N^c_{\min}[i] : Dom_{\min}(p, \mathsf{Q}_{\mathsf{ur}}))$
18. $A[c] \leftarrow$ **false**
19. **end if**
20. **end for**
21. **if** $\exists c$ such that $A[c]$ = **true and** $c \notin$ outputSet
22. $\mathrm{M}^2\mathrm{R}(\mathsf{Q}, \mathsf{N}[i]$.pointer$, A)$
23. **end for**
24. **endif**

end $\mathrm{M}^2\mathrm{R}$

Fig. 3. The $\mathrm{M}^2\mathrm{R}$ algorithm

of n points in a subset P of the plane, the number of maximal points is normally distributed, with average value $\sim \sqrt{\lambda}$, λ being the density. So, assuming w.l.o.g the normalized data space (where each coordinate is in the $[0, 1]$ range), one expects that the number of maximal points at level 0 (root) node is $O(\sqrt{n})$, at a level 1 node is $O(\sqrt{n/B})$ (B is the page capacity), at a level 2 node is $O(\sqrt{n/B^2})$, and generally, at a level i node is $O(\sqrt{n/B^i})$. Summing up we have the following lemma:

Lemma 3. *Let S be a set of n points stored in an augmented R-tree T. In case of uniformly distributed data, one expects that the B^+-trees of T demand $O(n/B^{1.5})$ pages.*

In the case of dynamic insertions/deletions of points, the proposed multi-tree indexing scheme can easily update the augmented information. For the insertion of a point p, it is initially inserted in the R-tree leaf N that is determined by the regular R-tree insertion algorithm. What has to be additionally taken into

account is that some of the maximal or minimal points in the father-entry of N may no longer be valid. This happens only in the case when p is a new maximal or/and minimal point in N.[3] In this case, we have to probe the corresponding entries of the maximal/minimal B^+-trees, to find the ones that have to be removed, and to insert p. Since p may also invalidate maximal/minimal points at upper levels as well, the previous procedure is repeated for all R-tree nodes invoked during the insertion of p. Considering a deletion of a point p from a leaf N, if it is a maximal or a minimal point in N, then we have to remove it from all B^+-trees in the root-to-leaf path corresponding to N, and additionally to insert the possible new maximal or minimal points from N, which may result after deleting p.

The case of semi-dynamic data sets (when only insertions are permitted) can be analyzed by standard amortization arguments [26, 4].[4] Omitting the proof due to space constraints, we have:

Lemma 4. *In the case of a semi-dynamic data set, an update operation has* $O(\log_B^2 n)$ *amortized cost.*

When the underlying data set is fully dynamic, then one cannot prove better than $O(n)$ worst-case bounds. However, in the average case, is easy to see that only an $O(\sqrt{n})$ fraction of inserted/deleted points will require an update of the augmented information leading to an $O(\sqrt{n})$ worst-case average performance. In practice one can experience much better performance.

As an additional optimization, we consider the case where in a node N, the roots of the B^+-trees of N's entries tend to have low utilization (their minimum allowed utilization is two entries). Therefore, when possible, such roots with few entries are kept packed into a single node. This reduces the required space and the processing time. The packing is done separately for maximal and minimal B^+-trees. Moreover, we utilized a scheme of storing, when possible, these packed roots at consecutive pages with that of node N, by trying to pre-allocate consecutive space for the two B^+-tree roots. Experimenting with the aforementioned optimizations, we found that they result to significant performance improvements, thus they are incorporated in the proposed method.

3.4 Selectivity Estimation

The *relative selectivity* (RS) is defined as the ratio between the sizes of output sets of the CRQ and the regular range query, when both are performed with the same query rectangle (RS is in the range [0, 1]). Let us consider the processing of a CRQ by filtering the output set of the regular range query so as to select the distinct categories in it —this method is henceforth denoted as Plain Range query Filtering (PRF). The performance difference between M^2R and PRF depends on the value of RS. When RS is high, PRF does not waste much cost,

[3] A point p can be *both* maximal and minimal when it is the only point of p.category.
[4] This case corresponds to data warehousing applications, where data are historical (i.e., they are not deleted).

whereas the processing of the augmented structures for M^{2}R may not pay-off. In contrast, when RS is not high, PRF spends a large cost to first find many points that satisfy the regular range query but will be filtered out later on. In most cases of interest (see Section 4), data are clustered with respect to the categorical attribute, and RS does not have high values.

It follows that the optimization of CRQs can be advocated by the estimation of RS. Although several methods have been proposed for estimating the selectivity of regular range queries, e.g., [1], up to our knowledge, no results have been reported for estimating the selectivity of a CRQ. As an initial approach towards this direction, we used the approximation of spatial data with index partitionings [1], which exploits the division of data space achieved by the R-tree (an analogous approach can be found in [3, 25]). By keeping track of the distinct categories within each partition, we can process all partitions, find those intersected by Q, and determine how many categories from each of them will be assigned to the output set.[5] Experiments with the aforementioned approach resulted to estimation error less than 1% for CRQs with large query rectangles, and around 15-25% for small ones. We are currently working on the development of an algorithm for RS estimation based on the Min-Skew algorithm [1], considering the distribution of the values of the categorical attribute instead of the density alone. Clearly, by using such an approach, a more accurate estimation is expected to be achieved.

Based on the RS estimation, the query optimizer will be able to decide whether the use of augmented structures will pay-off compared to the PRF method. In the extreme cases where there exist no actual grouping of points with respect to the categorical attribute, and so RS is high, the optimizer can reduce the overhead resulting from the augmented structures. This can be done by adapting M^{2}R to: (i) either consider only a fraction of them, e.g., by selecting every k-th augmented structure in the node instead of each one; a technique denoted as *reduction*), or (ii) entirely avoid to use them —in this case M^{2}R and PRF become equivalent.

4 Experimental Results

This section describes the results on the experimental comparison of: (i) the proposed algorithm (M^{2}R); (ii) the algorithm that first performs a plain range query and then filters w.r.t. category values (PRF); and (iii) the algorithm that is based on the naive augmentation of R-trees (NAR). We also examine why independent R-trees, one for each category, are not a viable solution. Our experiments consider both synthetic and real data sets. We study the impact of: the relative selectivity (RS) values, the domain size, the query and buffer sizes.

[5] For instance, we can first find the ratio r ($0 < r \leq 1$) between the volume of the intersection and the volume of an intersected partition. Next, if t is the number of distinct categories in the partition, we can select $r \cdot t$ categories (uniformity assumption), and filter those that have already been included in the output. Nevertheless, further discussion on this issue is out of the scope of this paper.

In the remaining of this section, we first describe the experimental setting and then we give the results.

4.1 Experimental Setting

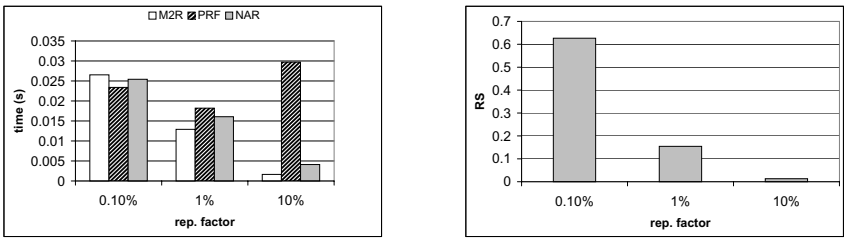
All examined methods were implemented in C using the same components. The experiments have been conducted on a computer with a Pentium III processor at 1.6 GHz. We used the R*-tree variant [7].

We used both synthetic and real data sets. The real data sets we consider are: Sequoia, Cover, and Census. The first one is from the Sequoia database[6] and contains 62,556 2-d points corresponding to California place names, along with their textual description. We cleaned the descriptions (e.g., by spell checking, removing empty ones, etc) and performed a grouping according to their type (e.g., a bar, beach, etc). This resulted to 180 distinct categories. The other two data sets belong to the UCI Machine Learning Repository[7]. The Cover data set contains 581,012 points and is used for predicting forest cover types from cartographic variables. From its 54 attributes, we used the spatial information to get the 2-d points corresponding to the locations. As categorical attribute we used the soil type information, which contains 40 distinct category values. Although Cover contains other possible categorical attributes, we selected soil because the others resulted to a much smaller number (less than ten) of distinct values. Finally, the Census-Income data set, denoted as Census, is a fragment of the US Census Bureau data and contains 199,523 records, from which we derived two separate sets: (i) the Census3d, having as dimensions the age, income and weeks worked; and (ii) the Census5d, having as additional dimensions the wage per hour and the dividends from stocks. As categorical attribute we selected, in both cases, the occupation type, that has 47 distinct values in total, because other possible ones would result to very small domains. Both the Cover and Census data sets have also been used elsewhere (e.g., in [13]).

To examine different characteristics of the data, we also considered synthetic data sets. Their generation was based on the following procedure. We specified a number of points that were the centers of overlapping 2-d normal (gaussian) distributions; since related work on main memory structures focused on the 2-d case, we use synthetic data sets of this dimensionality so as to examine the viability of the proposed method for this case, whereas we examine more dimensions with the Census data set. For each one of them, we generated a number of points that is given as a fraction of the total number of the points in the data set. This factor is called *replication factor* and corresponds to the size of each distribution. All points of a distribution are assigned to the same category. The total number of categories is pre-specified. We tested both the random (i.e., following uniform distribution) and skewed (i.e., following zipfian distribution) assignment of categories to points. However, both lead to qualitative similar results, thus we herein present results on the former. The coordinates in the data

[6] Available (also) at: `http://dias.cti.gr/~ytheod/research/datasets/spatial.html`

[7] Available at: `http://www.ics.uci.edu/~mlearn/MLRepository.html`



(a) (b)

Fig. 4. Execution time (a) and RS (b) w.r.t. replication factor, for query size 0.05%

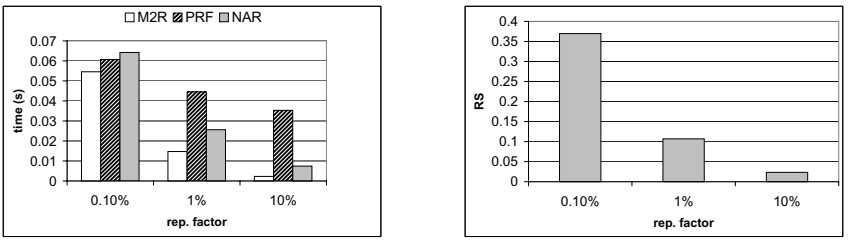
space were normalized in the range [0,1] and the standard deviation that was used in the normal distributions was equal to 5%.

The page size was set to 4 K and the default buffer size was set to the 20% of the data set size. For the queries we examined both uniform and biased workloads. With the former, queries corresponded to squares, the center of which follows a uniform distribution in the data set space. With the latter, the centers of the queries were points from the data set itself, thus they follow the data distribution. In our synthetic data sets we used uniform workloads, whereas for the real ones we used biased workloads. This is because the synthetic data have a good coverage of the entire data space, whereas the real ones are more skewed and the biased workload comprise a more realistic assumption. The query size is given as percentage of its size with respect to the size of the data space. The main performance metric was the total (wall-clock) execution time, measured in seconds, which includes I/O and CPU times for processing CRQs and the time to write the result.

4.2 Results

Our first experiment studies how the different values of RS affect CRQs. We used synthetic data sets with varying replication factor, that directly impacts RS. This happens because a larger replication factor results to a larger clustering with respect to the categorical value, therefore points that are close in space are more probable to belong to the same category. The data sets contained 100,000 points and 100 categories. Figures 4.a and 5.a depict the execution time (in seconds) with respect to replication factor, for query sizes 0.05% and 0.2% respectively. Figures 4.b and 5.b depict the corresponding RS values.

As expected, low values of replication factor result to large values of RS. In such cases, the PRF method performs well, since it does not spent too much extra cost. Nevertheless, the execution time of M^2R (also NAR) is similar to that of PRF, since it is not outperformed significantly by PRF for query size 0.05%,



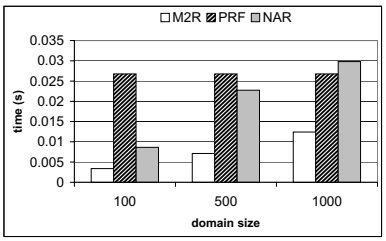
(a) (b)

Fig. 5. Execution time (a) and RS (b) w.r.t. replication factor, for query size 0.2%

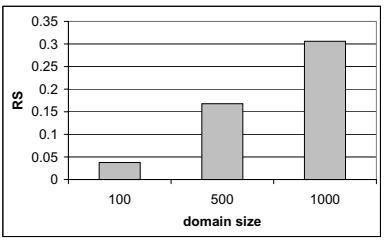
and is slightly better for query size 0.2%. The reason is that for these cases, M^2R exploits the reduction technique described in Section 3.4. When moving to medium and larger values of replication factor, RS reduces. The performance of PRF, however, is not affected —in some cases its execution time increases, whereas in others it reduces— since RS does not influence PRF. In contrast, as RS reduces, M^2R clearly outperforms PRF, whereas NAR comes second best. This is easily explained, since in these cases M^2R can perform a better pruning with respect to the categorical attribute. Another thing that can be noticed between Figures 4 and 5 is that in the latter, which corresponds to a larger query, RS values are relatively reduced. Although the output set of CRQ increases with increasing query size, the output set of the regular range query increases more rapidly with increasing query size. This results to the relative reduction in RS for the larger query size.

The aforementioned results verify the intuitive argument stated earlier in the paper, that the regular range query followed by filtering (i.e., PRF) is expected to waste a lot of cost in the cases when data are not randomly scattered along the categorical attribute, and that the performance gains due to M^2R are expected to be significant. Moreover, when RS values are high (random scattering), the results show that the overhead of M^2R is not significant. Fortunately, most of real data sets of interest (see the following) tend to be clustered along the categorical attributes. Therefore, in such cases the output size of the CRQ is a very small fraction of that of the regular range query.

Our next experiment examines the domain size of the categorical attribute, i.e., the total number of distinct values for the categorical attribute. We used synthetic data sets that were analogous to those used in the previous experiment. The replication factor was set to 5% and the query size was set to 0.1%. These values are selected so as to get a variety of different RS values w.r.t. the domain size. The execution time is depicted in Figure 6.a, whereas Figure 6.b illustrates the corresponding RS values.



(a)



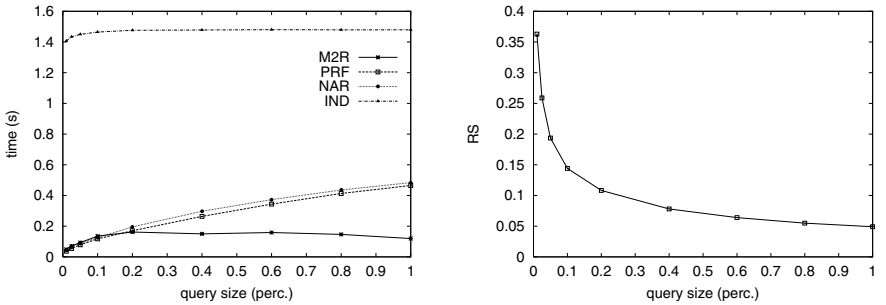
(a)

Fig. 6. Execution time (a) and RS (b) w.r.t. domain size of the categorical attribute

The increase in the domain size of the categorical attribute results to an increase in RS, since it increases the probability of more distinct categories to exist within the query region. As expected, PRF is independent from the domain size, because the result of the regular range query is the same in all cases. M^2R is only slightly affected by the increase in the domain size, and it outperforms the other two algorithms in all cases. In contrast, the impact is more clear on NAR, which for large domain sizes is outperformed by PRF. The aforementioned result indicates that M^2R scales well with respect to the domain size, even for very large values (in the order of thousand).

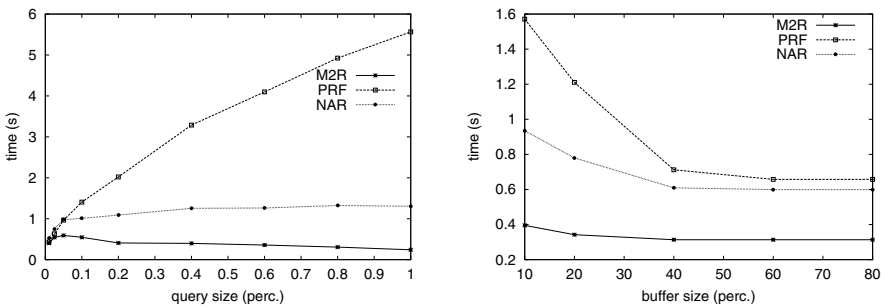
To validate the conclusions drawn with synthetic data, we also examined real ones. We first used the Sequoia data set. In this case, we also test the approach of having independent R-trees, each one storing separately the points of a category, which is denoted as IND. Figures 7.a and .b illustrate the execution time and RS, respectively, for varying query size, given as a percentage of the data set space. Similar to synthetic data sets, RS is decreased with increasing query size. It has to be noticed that the earlier stated argument, that the output of a CRQ is only a small fraction of the output of the regular range query, is verified by the values of RS in Figure 7.b. Regarding execution time, it is clearly noticed that IND presents the worst performance. This is because for every query it probes all categories although only few of them will belong to the output set. For this reason we do not examine IND in the following. Focusing on M^2R, NAR, and PRF, they perform similarly for very small query sizes. This is due to the higher RS values for these query sizes, and also due to the small size of the data set (about 62,000 thousand points) which renders all methods equivalent for such small query sizes. In contrast, for medium and large queries (0.5–1%), M^2R compares favorably to the other two methods.

As mentioned, the Sequoia data set is not large. To have a more clear comparison, we now move on to examine the Cover data set, that is much larger. Results on execution time for varying query size are depicted in Figure 8.a —RS values were similar to those in the previous experiment, thus they are omitted for



(a) (a)

Fig. 7. Execution time (a) and RS (b) w.r.t. query size for the Sequoia real data set



(a) (b)

Fig. 8. Execution time w.r.t. (a) query size, (b) buffer size, for the Cover real data set

brevity. As shown, with increasing query size, PRF clearly looses out. Therefore, the large size of data sets like Cover, renders the performance of PRF impractical, since for larger query sizes (1%) its execution time is an order of magnitude larger than that of M^{2}R. In contrast, M^{2}R performs very well in all cases. NAR comes second best, whereas the performance difference between M^{2}R and NAR is significant. Also, it has to be noticed that with increasing query size, the size of the output set for a CRQ increases, because its selectivity reduces (i.e., more categories are included in larger queries). Nevertheless, M^{2}R is beneficial regardless of the selectivity, since it can better determine containment (or not) at the higher R-tree levels, thus avoiding visits to the lower levels. This is the reason for the relative decrease in the execution time of M^{2}R for large query sizes. This can also be noticed in Figure 7.a.

Evidently, the performance of all examined methods depends on the provided memory size that is used for buffering. For this reason, we examined the impact of buffer size on the execution time. We used the Cover data set, and the query size was set to 0.1%. Figure 8.b illustrates the results for varying buffer size,

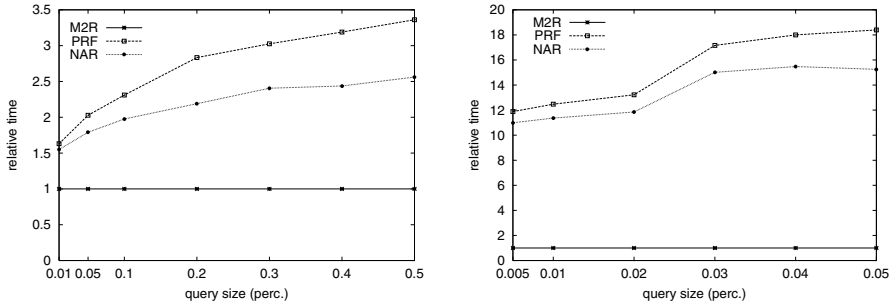


Fig. 9. Execution time w.r.t. query size for the: (a) Census3d and (b) Census5d real data sets

that is given as percentage of the data set size. Clearly, the execution time for all methods reduces with increasing buffer size. Since the performance of PRF is the worst among the three methods, it benefits more from increasing buffer size. Nevertheless, M^2R outperforms all other methods in all cases, even for very large buffer sizes.

Finally, we evaluated the proposed approach with the Census3d and Census5d data sets. The results for varying query size, given as a percentage of the data space, are depicted in Figure 9. The time axis corresponds to the relative execution times, normalized to those of M^2R, so as to more clearly present the performance differences between the two cases. For the Census3d (Figure 9.a), M^2R outperforms the other two methods by several factors, whereas NAR comes second best, performing similar to PRF for small query sizes and better for medium and larger ones. For the Census5d (Figure 9.b), the performance difference between M^2R and the other methods is much more pronounced. This is explained by the large skew that the additional two dimensions present, which result to a large cost paid by PRF even for small query sizes. In contrast, M^2R is not affected as much by the skewness, due to the early pruning with respect to categories.

5 Conclusions

This paper studied the problem of categorical range searching over large databases, that is, the finding of categories, instead of objects themselves, existing within the query region. We have developed M^2R, an algorithm that is based on the augmentation of R-trees with structures organizing the maximal/minimal points in subtrees. This approach follows the general paradigm of augmenting tree nodes in order to facilitate searching. M^2R is compared with the regular range query and with a naive method for augmenting R-tree nodes. Detailed experimental comparison with real and synthetic data illustrates the superiority of M^2R over the two base-line algorithms. Depending on the value of relative

selectivity of the CRQ, M^2R attains performance improvements up to an order of magnitude, for large real data sets.

There exist several directions of future work, for instance, the examination of data sets of very high dimensionality and the development of more sophisticated estimators for the selectivity of CRQs. Regarding the first issue, a decomposition scheme, based on multi-level B^+-trees that first narrow down the search space of the high dimensional space and then apply the solution presented in this work for moderate dimensions, seems a good candidate, demanding polylogarithmic overhead [26]. It is also interesting to consider that research on CRQs can be extended to other analogous queries, like the categorical nearest-neighbor queries [23] or the counting queries [17]. Such extensions can find interesting applications in large databases and, therefore, we consider the problem examined in this paper as a first attempt towards this research direction.

References

[1] S. Acharya, V. Poosala, S. Ramaswamy: "Selectivity Estimation in Spatial Databases". *Int. Conf. on Management of Data (SIGMOD'99)*, pp. 13-24, 1999. 131

[2] P. Agarwal, S. Govindarajan, S. Muthukrishnan: "Range Searching in Categorical Data: Colored Range Searching on Grid". *European Symp. on Algorithms (ESA'02)*, pp. 17-28, 2002. 122, 124

[3] G. Antoshenkov: "Random Sampling from Pseudo-ranked B^+-trees". *Int. Conf. on Very Large Databases (VLDB'92)*, pp. 375-382, 1992. 131

[4] L. Arge, J. S. Vitter: "Optimal Dynamic Interval Management in External Memory". *Symp. on Foundations of Computer Science (FOCS'96)*, pp. 560-569, 1996. 124, 130

[5] L. Arge, V. Samoladas, J. S. Vitter: "On Two-Dimensional Indexability and Optimal Range Search Indexing". *Symp. on Principles of Database Systems (PODS'99)*, pp. 346-357, 1999. 124

[6] A. D. Barbour, A. Xia: "The number of two-dimensional maxima", *Advanced Applications on Probability (SGSA)*, vol. 33, pp. 727-750, 2001. 126, 128

[7] N. Beckmann, H.-P. Kriegel, R. Schneider, B. Seeger: "The R^*-Tree: An Efficient and Robust Access Method for Points and Rectangles". *Int. Conf. on Management of Data (SIGMOD'90)*, pp. 322-331, 1990. 132

[8] M. de Berg, M. van Kreveld, M. Overmars, O. Schwarzkopf: *Computational Geometry*, Springer Verlag, 2nd ed., 2000. 126

[9] P. Bozanis, N. Kitsios, C. Makris, A. Tsakalidis: "New Upper Bounds for Generalized Intersection Searching Problems". *Int. Colloquium on Automata, Languages and Programming (ICALP'95)*, pp. 464-474, 1995. 122, 124

[10] P. Bozanis, N. Kitsios, C. Makris, A. Tsakalidis: "Red-Blue Intersection Reporting for Objects of Non-Constant Size". *The Computer Journal*, Vol. 39, No. 6, pp. 541-546, 1996. 122, 124

[11] P. Bozanis, N. Kitsios, C. Makris, A. Tsakalidis: "New Results on Intersection Query Problems". *The Computer Journal*, vol. 40, No. 1, pp. 22-29, 1997. 122, 124

[12] P. Bozanis, A. Nanopoulos, Y. Manolopoulos: "LR-tree: a Logarithmic Decomposable Spatial Index Method", *The Computer Journal, to appear*, 2003. 124

[13] N. Bruno, S. Chaudhuri, L. Gravano: "Top-k selection queries over relational databases: Mapping strategies and performance evaluation". *ACM Transactions on Database Systems*, Vol. 27, No. 2, pp. 153-187, 2002. 132

[14] D. Eppstein, S. Muthukrishnan: "Internet packet filter management and rectangle geometry". *Symp. on Discrete Algorithms (SODA'01)*, pp. 827-835, 2001. 124

[15] P. Ferragina, N. Koudas, S. Muthukrishnan, D. Srivastava: "Two-dimensional Substring Indexing". *Symp. on Principles of Database Systems (PODS'01)*, pp. 282-288, 2001. 124

[16] S. Govindarajan, P.K. Agarwal, L. Arge: "CRB-Tree: An Efficient Indexing Scheme for Range-Aggregate Queries". *Intl. Conf. on Database Theory (ICDT'03)*, pp. 143-157, 2003. 124

[17] P. Gupta, R. Janardan, M. Smid: "Further Results on Generalized Intersection Searching Problems: Counting, Reporting, and Dynamization". *Journal of Algorithms*, Vol. 19, No. 2, pp. 282-317, 1995. 122, 124, 138

[18] P. Gupta, R. Janardan, and M. Smid: "Efficient Algorithms for Generalized Intersection Searching on Non-Iso-oriented Objects". *Computational Geometry: Theory & Applications*, Vol. 6, No. 1, pp. 1-19, 1996. 122, 124

[19] J. Hellerstein, E. Koutsoupias, D. Miranker, C. Papadimitriou, V. Samolodas: "On a Model of Indexability and Its Bounds for Range Queries". *Journal of the ACM*, Vol. 19, No. 1, pp. 35-55, 2002. 123, 125

[20] R. Janardan, M. Lopez: "Generalized intersection searching problems". *Int. Journal on Computational Geometry and Applications*, Vol. 3, pp. 39-69, 1993. 122, 124

[21] K. Kanth, S. Ravada, D. Abugov. "Quadtree and R-tree indexes in oracle spatial: a comparison using GIS data". *Int. Conf. on Management of Data (SIGMOD'02)*, pp. 546-557, 2002. 123

[22] Y. Manolopoulos, E. Nardelli, G. Proietti, E. Tousidou: "A generalized comparison of linear representations of thematic layers". *Data and Knowledge Engineering*, Vol. 37, No. 1, pp. 1-23, 2001. 125

[23] D. Mount, N. Netanyahu, R. Silverman, A. Wu: "Chromatic nearest neighbor searching: A query sensitive approach". *Computational Geometry*, Vol. 17, No. 3-4, pp. 97-119, 2000. 122, 124, 138

[24] S. Muthukrishnan: "Efficient algorithms for document retrieval problems". *Symp. on Discrete Algorithms (SODA'02)*, pp. 657-666, 2002. 124

[25] A. Nanopoulos, Y. Theodoridis, Y. Manolopoulos: "An Efficient and Effective Algorithm for Density Biased Sampling". *Int. Conf. on Information and Knowledge Management (CIKM'02)*, pp. 63-68, 2002. 131

[26] M. H. Overmars: *The Design of Dynamic Data Structures*, Springer-Verlag, Berlin, Heidelberg, 1983. 123, 130, 138

[27] D. Papadias, Y. Tao, P. Kalnis, J. Zhang: "Indexing Spatio-Temporal Data Warehouses". *Int. Conf. on Data Engineering (ICDE'02)*, 2002. 124

[28] J.R. Sack, J. Urrutia (Eds.): *Handbook of Computational Geometry*, North-Holland, 2000. 126

[29] Y. Tao, D. Papadias: "The MV3R-Tree: A Spatio-Temporal Access Method for Timestamp and Interval Queries". *Intl. Conf. on Very Large Data Bases (VLDB'01)*, pp. 431-440, 2001. 124