The Tantalizing New Prospect of Index-Based Diversified Retrieval

George Tsatsanifos
supervised by Timos Sellis
School of Electrical & Computer Engineering
National Technical University of Athens
gtsat@dblab.ece.ntua.gr

ABSTRACT

In this paper, we propose efficient algorithms for result diversification over indexed multi-dimensional data. We develop algorithms under the prism of a centralized approach, as in a database. Specifically, we rely on widely used multi-dimensional indexes, like the R-tree. In principle, our schemes adopt a maximal marginal relevance (MMR) ranking strategy and leverage interchange and greedy diversification techniques. Hitherto, mostly combinatorial aspects of this problem have been considered which require scanning the entire data, and therefore, existing solutions are costly.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*Retrieval models, Search process*

General Terms

Algorithms; Design; Experimentation; Performance

Keywords

Search Result Diversification; Query Processing; Branch and Bound; Best-First Search

1. INTRODUCTION

Result diversification has attracted recently great scientific interest for it addresses a variety of problems simultaneously. First, modern information retrieval research focuses on result diversification as a means of counteracting intrinsic IR problems, like the over-specialization problem of retrieving too homogeneous results. Therefore, new search techniques had to be improvised in order to ameliorate user satisfaction and decrease query abandonment. Second, diversification reduces the risk that none of the returned results satisfies a user's query intent. Third, enabling diversity tacitly facilitates (near) duplicates elimination. Besides, conventional information retrieval models presume that given a similarity measure with different degrees of relevancy, the relevance of a document is independent of the relevance of other documents. In reality however, this *independence relevance* assumption rarely holds; the utility of retrieving one document may depend on which documents the user has already seen. Regarding this complementary aspect of diversification, it is in general insufficient to simply return a set of relevant results, since correlations among them are also important. More specifically, documents should be selected progressively according to the relevance of the document in terms with the documents that come before it, in the sense of comprising new information. The main challenge in diversification lies in that these two important goals are contradictory to each other. In essence, the goal of diversification is to identify a list of items that are dissimilar with each other, but nonetheless relevant to the user's search criteria. In other words, the diversity is defined by how much each item of the result-set differs from the others in terms of their attribute values. In effect, diversification is a multifaceted problem with many practical and theoretical extensions. A profound and rich link with combinatorics and approximation is exploited in research works where diversification is performed through heuristics and greedy algorithms mostly.

In this paper, we address the problem of search result diversification over multi-dimensional disk-resident data, through content-based definitions in order to maximize the satisfaction of different users. To this end, we employ the R-tree to serve as a indexing infrastructure but our methods are also applicable for its variants.

2. RELATED WORK

Content-based diversification aims at maximizing the minimum distance between any pair of objects in the result. In the seminal work of Maximal Marginal Relevance (MMR) [2], the authors introduced a tradeoff between diversity and the relevance of search results through the combination of two similarity functions. One for measuring the similarity among a document collection S, and the other the similarity between document and query $\vec{q}$. Parameter λ controls the degree of tradeoff.

$$f(S|\vec{q}) = \max_{\vec{x} \in S} \lambda d(\vec{q}, \vec{x}) - (1-\lambda) \min_{\vec{x},\vec{y} \in S} d(\vec{x}, \vec{y}) \tag{1}$$

[3] follows an axiomatic approach to develop a set of natural axioms that a diversification system is expected to follow. [4] focuses on structured data to propose an appropriate similarity function for a given ordering of the data attributes. In [6] starting with the K most relevant items; at each iteration, the item that contributes the least to the diversity, is substituted by the most promising that is not part of the answer set, till there are no more items with diversity higher than a given threshold.

[5] suggests that "novelty seeking" is not equivalent to "diversity seeking", and that the novelty preferences of individual users are directed towards finding more information on specific subtopics, rather than an undirected quest for any new information. In [1] this

approach is augmented with taking into account the relative importance of different nuggets (as distribution categories), and the fact that different documents containing the same nugget may satisfy the user to different extent.

3. MODELING DIVERSIFIED SEARCH

In this section we study the following sub-problem of the result diversification problem. Our solution to the partial problem will evolve in the following sections to a complete method for constructing diversified sets.

3.1 Definitions

Given a universe of objects U, its subset $S \subset U$, a query object $\vec{q}$, and a ranking function f, we want to find the object $\vec{p} \in U \setminus S$, which when added to S we have that $f(S \cup \{\vec{p}\}|\vec{q}) \leq f(S \cup \{\vec{x}\}|\vec{q}), \forall p, x \in U \setminus S$. In other words, we need to retrieve efficiently the object which when added to S it would cause the minimum increase in f from all objects in $U \setminus S$. In effect, this is translated into retrieving the closest stored tuples to query object $\vec{q}$ that also maximize their pairwise distance among them. Therefore, when examining an object before inserting it in the solution, we compute its similarity in terms of query object $\vec{q}$, and also, the resulting pairwise dissimilarity of all objects in S that are already part of the answer.

To elaborate, when adding an object, say $\vec{p}$, in an answer-set, say S, then f might increase depending on its distance from query object $\vec{q}$ and its distance from all previous objects in S. In particular, MMR ranking functions like f from Eq. 1 have a monotonically increasing behavior, as more items are inserted into S. According to certain criteria a set's rank might stay the same or deteriorate after an insertion, but never improve. Therefore, there is such an object $\vec{p_{\text{opt}}}$, which when added to S, it causes the minimal increase. More specifically, we discern four different cases. For the first case, the new object $\vec{p}$ is within range of the least relevant object from $\vec{q}$ in S, and farther from any object in S than the distance between the closest pair of objects in S. Therefore, the value of f for the augmented set does not change as the least relevant object and the least distant pair in the new set $S \cup \vec{p}$ remain the same. In the second case, the new object $\vec{p}$ is farther from $\vec{q}$ than any object in S, and farther from any object in S than the distance between the closest pair of objects in S. Therefore, the f-value of the augmented set will be increased by the relevance difference between $\vec{p}$ and the former farthest object from $\vec{q}$. The third scenario includes the cases where even though $\vec{p}$ is closer to $\vec{q}$ than the least relevant object in S, its closest distance from any object in S is less than the distance between the closest pair of objects in S. Hence, the score of the augmented set increases by this difference. Last, if $\vec{p}$ is less relevant than any point in S and its distance from any point in S is less than the distance between any pair of objects in S, then f increases by both the relevance and diversity loss caused by the insertion of $\vec{p}$. In effect, the new ranking function ϕ from Eq. 2 for items in $U \setminus S$ expresses the change in S's rank when an insertion takes place. All cases are included in Eq. 2 for quantifying object eligibility given a query $\vec{q}$ and a set S, as set $U \setminus S$ is totally ordered under ϕ since antisymmetry, transitivity and totality properties hold for all its elements. Thus, the object in $U \setminus S$ that minimizes Eq. 2 is also the one which solves the presented sub-problem of retrieving the optimal item to insert into the result-set.

Moreover, we can generalize the notion of ranking function ϕ to Minimum Bounding Rectangles (MBR), as well. This is of major significance since R-tree nodes are associated with a MBR and also comprise a number of child nodes whose respective rectangles are fully enclosed by the parent's MBR, thereby forming a hierarchy. Henceforth, we describe a hyper-rectangle with an interval over multiple dimensions, consisting of a lower and an upper bound. More formally, $\phi([\vec{\ell}, \vec{h}]|\vec{q}, S) = \min_{\vec{p} \in [\vec{\ell}, \vec{h}]} \phi(\vec{p}|\vec{q}, S)$ can be approximated with its lower bound as, $\phi([\vec{\ell}, \vec{h}]|\vec{q}, S) \geq \min_{\vec{x} \in [\vec{\ell}, \vec{h}]} d(\vec{x}, \vec{q}) - \max_{\vec{y} \in [\vec{\ell}, \vec{h}], \vec{s} \in S} d(\vec{y}, \vec{s})$. In other words, no object in hyper-interval $[\vec{\ell}, \vec{h}]$ can achieve a better score than this. Likewise, we deduce that $\phi([\vec{\ell}, \vec{h}]|\vec{q}, S) \leq \max_{\vec{x} \in [\vec{\ell}, \vec{h}]} d(\vec{x}, \vec{q})$.

3.2 Search Space Restrictions

Given a set S of multi-dimensional objects, Alg. 1 retrieves the object in $U \setminus S$ which when added in S, it is better ranked in terms of a ranking function f, than it would be if we had appended any other object. Our method relies on the intuition that for a given query $\vec{q}$, the current state of the answer-set S readily promotes objects from certain areas. In other words, objects originated from specific areas of the key-space are higher ranked than others, either due to being more relevant, or more diverse, or both. Most importantly, our scheme is in position of exploiting this crucial property and makes the most out of it by excluding from search all sub-optimal areas of the key-space. In practice, this is translated into preventing from reading certain R-tree nodes, and thereby, maximize the efficiency of our method, as only objects that contribute to the result are encountered.

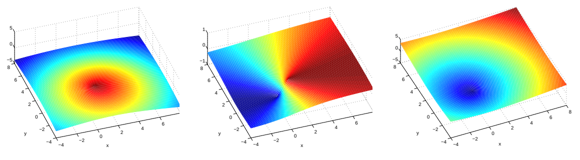
Algorithm 1 diversion($\vec{q}, S, \Phi_{\text{thres}}$,rtree)
```
1: h ← minHeap()
2: h.push(rtree.rootNode)
3: while not (h.isEmpty() or h.top().isLeaf()) do
4:   entry ← h.top()
5:   h.pop()
6:   for childEntry in entry.childNodes do
7:     $\vec{\ell}$ ← childEntry.MBR.lo
8:     $\vec{h}$ ← childEntry.MBR.hi
9:     if $\phi([\vec{\ell}, \vec{h}]|\vec{q}, S \setminus \{\vec{s}\}) < \Phi_{\text{thres}}$ then
10:      heap.push(childEntry)
11:    end if
12:    $\Phi_{\text{thres}} \leftarrow \min\{\Phi_{\text{thres}}$, maxdist$(\vec{q}, [\vec{\ell}, \vec{h}])\}$
13:  end for
14: end while
15: if not h.isEmpty() and h.top().isLeaf() then
16:   return h.top().MBR
17: end if
18: return nil
```

To elaborate, Alg. 1 directs the search using the heap h in line 1, which comprises R-tree nodes sorted ascending on their lower bound for ϕ. It is safe to assume in line 9, Alg. 1 that the score achieved by any possible indexed object subsumed by an internal R-tree node associated with $[\ell, h]$, is higher than its lower bound, without any loss. As a result, starting from any internal node of the R-tree, when searching for objects that are better ranked than a given threshold, say Φ, then *only* the branches of the examined node that correspond to MBRs with lower bounds less than Φ should be further processed. For each of the accessed entries of the R-tree, sorted by their ϕ-values, a number of iterations is performed (lines 3–14, Alg. 1). The loop breaks when the popped processed node is a leaf, in which case the object corresponding to this node corresponds to the optimal object from $U \setminus S$, say p_j^*, that causes the minimum change in the rank of the result. Now, assuming that the processed node is not a leaf, the corresponding disk-page is read and each of the child nodes is checked whether its rank meets the threshold criteria in line 9. If it does, then the child node is inserted

$$\phi(\vec{p}|\vec{q},S) = \begin{cases} 0, \text{ if } d(\vec{p},\vec{q}) \le \max_{\vec{x}\in S} d(\vec{x},\vec{q}) \text{ and } \min_{\vec{x}\in S} d(\vec{p},\vec{x}) \ge \min_{\vec{y},\vec{z}\in S} d(\vec{y},\vec{z}), \\ \lambda(d(\vec{p},\vec{q}) - \max_{\vec{x}\in S} d(\vec{x},\vec{q})), \text{ if } d(\vec{p},\vec{q}) > \max_{\vec{x}\in S} d(\vec{x},\vec{q}) \text{ and } \min_{\vec{x}\in S} d(\vec{p},\vec{x}) \ge \min_{\vec{y},\vec{z}\in S} d(\vec{y},\vec{z}), \\ (1-\lambda)(\min_{\vec{x},\vec{y}\in S} d(\vec{x},\vec{y}) - \min_{\vec{z}\in S} d(\vec{p},\vec{z})), \text{ if } d(\vec{p},\vec{q}) \le \max_{\vec{x}\in S} d(\vec{x},\vec{q}) \text{ and } \min_{\vec{x}\in S} d(\vec{p},\vec{x}) < \min_{\vec{y},\vec{z}\in S} d(\vec{y},\vec{z}), \\ \lambda(d(\vec{p},\vec{q}) - \max_{\vec{x}\in S} d(\vec{x},\vec{q})) + (1-\lambda)(\min_{\vec{y},\vec{z}\in S} d(\vec{y},\vec{z}) - \min_{\vec{x}\in S} d(\vec{p},\vec{x})), \text{ otherwise.} \end{cases} \tag{2}$$

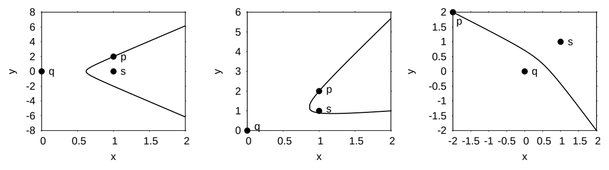
into the heap (line 10), otherwise the associated branch of the R-tree is pruned and none of its disk-pages are accessed. We note that not all of the inserted nodes in the heap are further processed. More specifically, the inserted nodes that succeed the first leaf entry in the heap, and thus are outranked, will never be processed to extract the associated sub-trees of the R-tree.

Furthermore, Figure 1 provides an insight of the score density according to function $\phi(.|\vec{q},\{\vec{s}\})$ for the two dimensional case. In our illustration, $\vec{q} = (0,0)$ for simplicity without any loss. The graph can naturally be shifted and centered to any query object. Since well diversified objects achieve low values, we assume that better ranked objects are located in the blue areas of our 3-dimensional illustration of ϕ. Then, given an object $\vec{p}$ with its ϕ-value equal to threshold Φ, Figure 2 depicts with a solid line the possible locations for the objects in U that achieve the same score as $\vec{p}$. In practice, ranking function $\phi(.|\vec{q},\{\vec{s}\})$ serves as a discriminant function dividing the key-space into two distinct parts. The first part consists of the region at the left of the dividing curve, where all enclosed objects are better ranked than $\vec{p}$. The rest of the key-space, the region in the right enclosed by the hyperbolas contains objects worse that $\vec{p}$. Therefore, when searching for a better object than $\vec{p}$, Alg. 1 excludes from search all disk-pages that are responsible for areas of the key-space that do not overlap with the part of the key-space which is located in the left. We note that the better score object $\vec{p}$ achieves, a larger area of the key-space is pruned.



(a) $\lambda = 0.3$ (b) $\lambda = 0.5$ (c) $\lambda = 0.7$

Figure 1: Score density distribution for $\vec{s} = (1,0)$.



(a) $\vec{s} = (1,0)$, $\vec{p} = (1,2)$ (b) $\vec{s} = (1,1)$, $\vec{p} = (1,2)$ (c) $\vec{s} = (1,1)$, $\vec{p} = (-2,2)$

Figure 2: Dividing curves for $\lambda = 0.5$.

Next, we show how the dividing curves in Figure 2 were computed. This corresponds to restricting a certain part of the key-space according to a certain threshold Φ associated with the score of a retrieved item (or the upper bound for the score of an indexed object subsumed by a retrieved internal node). Our computations consider low dimensionality (spatial) spaces but can be generalized for higher dimensional spaces, as well. For simplicity, we consider that $\vec{q} = (0,0)$, and $S = \{\vec{s}\}$, where $\vec{s} = (s_x, s_y)$, without any loss. Object $\vec{p_0}$ stands for an object achieving score equal to the Φ-threshold. After finding the set of points that achieve the same score as $\vec{p_0}$ (locus), we determine the area of the key-space with objects that are better ranked.

Nevertheless, in the above we consider sets with only one element. When more items are comprised in S, then we search for better ranked R-tree entries in internal and leaf-nodes, exclusively in the area which corresponds to the *intersection* of all areas with improved objects for each element $\vec{s_i} \in S$. Last, our proposals generalize to other multi-dimensional spaces of higher dimensionality.

4. RANK-AWARE QUERY PROCESSING

The algorithm presented in this section starts from an initial set of size K and prevents from accessing all disk-pages that cannot improve the result in its current state. The main idea behind our scheme is computing a lower bound for each entry of the R-tree of the data objects, and prune the sub-trees whose lower bound is greater or equal to a certain threshold which corresponds to the score of best answer already found. More formally, given an initial set S, let sub-sets $S_1, S_2, \cdots, S_K$, where $S_j = S \setminus \{\vec{s_j}\}$. In principle, we investigate whether there is a better object, say $\vec{p_j} \in U \setminus S$, such that $f((S \setminus \{\vec{s_j}\}) \cup \{\vec{p_j}\}) < f(S)$. In particular, from all the possible items $\vec{p_j}$ that improve the initial set's S score, we want to retrieve the one, say $\vec{p_j}^*$, such that $f((S \setminus \{\vec{s_j}\}) \cup \{\vec{p_j}^*\}) \le f((S \setminus \{\vec{s_j}\}) \cup \{\vec{p_j}\}), \forall \vec{p_j}, \vec{p_j}^* \in U \setminus S$. We also assume an order in the turn we examine the objects in S. More specifically, an object $\vec{s_i}$ from S precedes s_j, denoted as $\vec{s_i} \preceq \vec{s_j}$, if and only if, $\phi(\vec{s_i}|\vec{q},S_i) \le \phi(\vec{s_j}|\vec{q},S_j)$. Eq. 3 indicates that if $\vec{s_i} \preceq \vec{s_j}$ in S, then S_j is a better ranked answer-set than S_i. In essence, our approach involves starting from the best sub-set of S, and retrieving the best substitute for the replaced element. Therefore, the replacement of the next sub-set of S should be such that it would outrank the previously enhanced answer-set (consisting of a better sub-set and the optimal replacement from the universe), and thus, imposing progressively stricter restrictions due to past improvements made.

$$\begin{aligned} f((S \setminus \{\vec{s_i}\}) \cup \{\vec{s_i}\}) &= f((S \setminus \{\vec{s_j}\}) \cup \{\vec{s_j}\}) \\ f(S_i \cup \{\vec{s_i}\}) &= f(S_j \cup \{\vec{s_j}\}) \\ f(S_i) + \phi(\vec{s_i}|\vec{q},S_i) &= f(S_j) + \phi(\vec{s_j}|\vec{q},S_j) \\ f(S_i) &\ge f(S_j) \end{aligned} \tag{3}$$

Alg. 2 retrieves $\vec{p_j}^*$ from an R-tree by accessing *only* the disk-pages with items that improve answer-set's S score when replacing $\vec{s_j}$, as all other pages excluded from search. Moreover, our algorithm takes into account the selection made at the previous steps. We want $\vec{p_j}^*$ not only to make the new answer-set S' to be better ranked than the original answer S, but also to be better ranked than the ones retrieved in the previous steps. This is done sequentially when searching for $\vec{p_j}^*$, where $j > 1$, we consider the score achieved when including in the answer S the best retrieved item so far, $\vec{p_i}^*$, instead of $\vec{s_i}$, with $j > i$. For instance, when searching for possible replacements of $\vec{s_1}$, we want for the new result-set,

$$\begin{aligned} f(S_1 \cup \{\vec{p_1}\}) &< f(S_1 \cup \{\vec{s_1}\}) \\ f(S_1) + \phi(\vec{p_1}|\vec{q}, S_1) &< f(S_1) + \phi(\vec{s_1}|\vec{q}, S_1) \\ \phi(\vec{p_1}^*|\vec{q}, S_1) \leq \phi(\vec{p_1}|\vec{q}, S_1) &< \phi(\vec{s_1}|\vec{q}, S_1) \end{aligned} \tag{4}$$

Specifically, in line 8, a threshold variable is set according to Eq. 4. This threshold is compared with accessed tree nodes and *only* the child nodes whose score is lower than the threshold will be further processed. Generally, if such an item $\vec{p_i}$ is found as a possible replacement for $\vec{s_i}$, we want for $\vec{s_j}$'s replacement,

$$\begin{aligned} f(S_j \cup \{\vec{p_j}\}) &< f(S_i \cup \{\vec{p_i}\}) \\ f(S_j) + \phi(\vec{p_j}|\vec{q}, S_j) &< f(S_i) + \phi(\vec{p_i}|\vec{q}, S_i) \\ \phi(\vec{p_j}^*|\vec{q}, S_j) \leq \phi(\vec{p_j}|\vec{q}, S_j) &< \phi(\vec{p_i}|\vec{q}, S_i) + \underbrace{f(S_i) - f(S_j)}_{\delta \leq 0} \end{aligned} \tag{5}$$

Again, the threshold variable Φ_{thres} is updated in line 6, Alg. 2 according to Eq. 5. Now, only the nodes of the R-tree that meet the updated criteria are further processed. Moreover, the δ-factor in Eq. 5, essentially strengthens the restrictions imposed by $\vec{p_i}$, to compensate for the differences in the initial conditions set to objects $\vec{p_i}$ and $\vec{p_j}$ by S_i and S_j, respectively. We note that the order in which the elements of the answer-set are examined is defined in line 3, Alg. 2, where they are sorted according to their ϕ-values in descending order. Therefore, as search areas become smaller and more limited, Alg. 2 comes forth as an efficient diversification method, since all pages that contain elements that cannot outrank either S, or $S_i \cup \{\vec{p_i}^*\}$ are disregarded. Thereby, the previously retrieved result $\vec{p_i}$, indirectly affects with its score the pruning that takes place in all subsequent steps. Therefore, our progressive search resembles a continuous process, rather than a sequence of distinct steps whose numerous results should be compared at the end to opt for the best. In effect, at the j-th iteration of Alg. 2, none R-tree node is read that cannot contain items with better score than either $\vec{s_j}$, or replacement $\vec{p_i}^*$ from a previous iteration, if any.

Moreover, at each iteration, the retrieved object $\vec{p_j}^*$ corresponds to a better replacement for $\vec{s_j}$, than the previously retrieved substitute $\vec{p_i}^*$ of $\vec{s_i}$. Therefore, the current solution progressively replaces a previous one. When Alg. 2 terminates, it has found the best pair combination of objects, consisting of an element $\vec{s} \in S$ and an item $\vec{p}^*_{+\infty} \in U \setminus S$, according to which if $\vec{s}$ is replaced with $\vec{p}^*_{+\infty}$ to form a new answer that is better ranked than it would be if any object from S had been replaced with any other object in $U \setminus S$.

In Alg. 3 we present the pseudo-code of an algorithm that solves the full diversification problem. To elaborate, it invokes the method `diversify-singlepass` and improves the initial answer-set by replacing with each invocation a single element of the set with the best substitute in U. Alg. 3 keeps refining the result and it terminates either when none improvement can take place as the result-set converges, or the maximum number of iterations is reached.

Furthermore, regarding the initialization of the answer-set S, there is no clear trend in the literature. Some works start with a random result, while others initialize the answer-set with the K closest neighbors of the query $\vec{q}$. Since efficiency is the main incentive behind this work, we choose to initialize S with the items that are comprised in the disk-page responsible for the query object $\vec{q}$. If more items are needed,then the neighboring disk-pages are accessed. Specifically, having enacted a lookup process for the query object in the R-tree, we fill S with the objects in the encountered leaf nodes of the tree, until K elements are congregated. These objects are very close to the query but not necessarily the closest, as they are greedily retrieved by the way they are stored in the disk.

Algorithm 2 `diversify-singlepass`($\vec{q}$,S,`rtree`)

```
1: $\vec{p}_{\text{best}} \leftarrow$ **nil**
2: $\vec{s}_{\text{repl}} \leftarrow$ **nil**
3: $S_{\text{sorted}} \leftarrow$ sort($S, \phi$, reverse) //$S_{\text{sorted}} \leftarrow \vec{s_1} \succeq \cdots \succeq \vec{s_K}$
4: for $\vec{s}$ in $S_{\text{sorted}}$ do
5:     if $\vec{p}_{\text{best}} \neq$ nil and $\vec{s}_{\text{repl}} \neq$ nil then
6:         $\Phi_{\text{thres}} \leftarrow \phi(\vec{p}_{\text{best}}|\vec{q}, S \setminus \{\vec{s}_{\text{repl}}\}) + f(S \setminus \{\vec{s}_{\text{repl}}\}) - f(S \setminus \{\vec{s}\})$
7:     else
8:         $\Phi_{\text{thres}} \leftarrow \phi(\vec{s}|\vec{q}, S \setminus \{\vec{s}\})$
9:     end if
10:    $\vec{p}_{\text{best}} \leftarrow$ diversion($q, S \setminus \{\vec{s}\}, \Phi_{\text{thres}}$, rtree)
11:    if $\vec{p}_{\text{best}} \neq$ nil then
12:        $\vec{s}_{\text{repl}} \leftarrow \vec{s}$
13:    end if
14: end for
15: if $\vec{p}_{\text{best}} \neq$ nil and $\vec{s}_{\text{repl}} \neq$ nil then
16:     return $S \setminus \{\vec{s}_{\text{repl}}\} \cup \{\vec{p}_{\text{best}}\}$
17: end if
18: return $S$    // none improvement for $S$ was found
```

Algorithm 3 `diversify-multipass`($\vec{q}$,K,`rtree`)

```
1: $S_{\text{curr}} \leftarrow$ initializeResult ($\vec{q}$,$K$,rtree)
2: for passes $\leftarrow 1$ to MAX_PASSES do
3:     $S_{\text{prev}} \leftarrow S_{\text{curr}}$
4:     $S_{\text{curr}} \leftarrow$ diversify-singlepass($\vec{q}$, $S_{\text{prev}}$,rtree)
5:     if $|S_{\text{prev}} \cap S_{\text{curr}}| = K$ then
6:         break    // none change took place
7:     end if
8: end for
```

5. EXPERIMENTAL EVALUATION

In this section, we verify the effectiveness of our schemes by comparing them against baseline methods that rely on exhaustive search. The efficiency of all centralized schemes is evaluated according to the execution time required by a query and the total number of disk accesses. In particular, we consider *random access time*[*], and the *transfer time* to actually read the requested data. Random access time takes 3 ms to 6 ms on high-end modern disks and dominates access times for single accesses. Next, the requested data are transferred at a rate of 100-180 MB/s. We also disable the system cache in order to focus thoroughly on the IO characteristics.

Parameter	Range	Default
dataset size $\|U\|$	100K, 500K, 1M, 5M, 10M	1M
dimensionality $\|D\|$	$2, 3, 5, 7, 11, 13$	3
result-size K	10,20,30,40,50,60,70,80,90,100	30

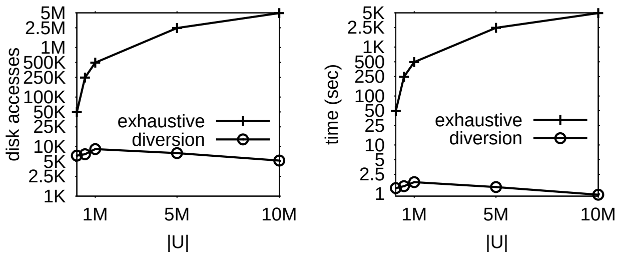
Table 1: System parameters for centralized schemes.

Specifically, we investigate how both metrics scale with the parameters in Table 1 Approximately $10K$ clusters constitute the synthetic dataset used. We also used real spatial datasets from `www.rtreeportal.org` to obtain similar results. When we vary one parameter, all others are set at their default values. For our scheme datasets are indexed using R-trees using a page-size of $4KB$ resulting in node capacities between 200 for two dimensions and about 30 for thirteen dimensions.

[*]the time to physically move the disk head to the right sector ("seek"), and the time to get the addressed area to a place where it can be accessed ("rotational delay")

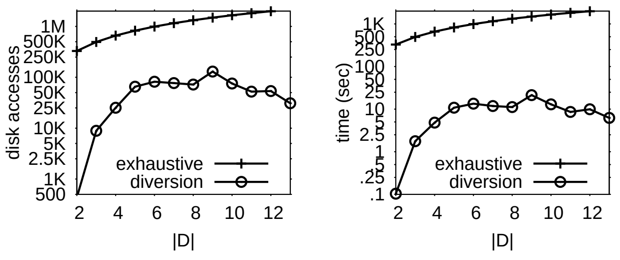
5.1 Results

In Figure 3, we show how all metric scale with regard to varied data-set size. Evidently, as data-set becomes bigger, so does the R-tree. In particular, additional leaf-nodes need to be created since R-tree nodes can fit a specific maximum number of entries, and thereby, more internal nodes need to be created in order to fully enclose the subsumed lower levels which also get larger. Therefore, more disk-pages are read, as shown in Fig. 3(a), as part of the diversification process. For the same reasons execution time increases with $|U|$ in Figure 3(b). Nonetheless, the exhaustive search baseline performs many scans until it reaches its final solution, as it accesses all disk-pages and this requires more time of course.



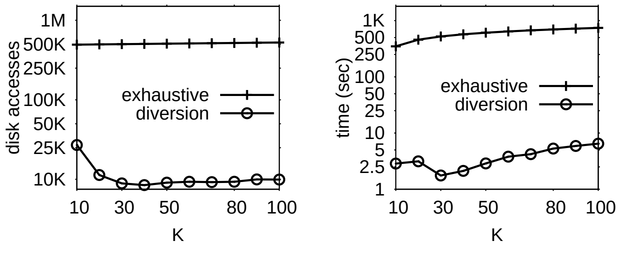
(a) Disk accesses (b) Execution time

Figure 3: in terms of $|U|$



(a) Disk accesses (b) Execution time

Figure 4: in terms of $|D|$



(a) Disk accesses (b) Execution time

Figure 5: in terms of K

In Figure 4, as dimensionality increases, R-tree entries occupy more space, and therefore, less MBRs or objects are comprised in internal and leaf nodes, respectively. Therefore, the R-tree becomes bigger in size in order to congregate the same number of data items. Increasing the data-set size and the dimensionality degree have a similar impact since both cause the R-tree to grow bigger and occupy more disk space, and therefore, more disk-pages are read until highly ranked data are retrieved. This is evident in both Figures 4(a) and 4(b) for the required I/Os and time, respectively.

The increase of the result-size has a bilateral impact on performance. Specifically, more items need to be examined in the result-set and whether they should be replaced, and thus, we would expect a linear increase in I/Os and time with K that would impair performance, due to the additional consecutive operations for computing all possible replacements. However, this is not the case as shown in Figures 5(a) and 5(b), where the combined impact of two contradicting phenomena is revealed. To elaborate, since we examine only one item from S at a time, there are $K-1$ other items restricting the searched area of the key-space. We note, that only the R-tree entries that intersect with the intersection of all $K-1$ restricted search areas are accessed. Therefore, as K increases, there are more restrictions imposed which effectively make the search area shrink, and therefore, less disk-pages are access. As a result, performance improves due to the more limited search operations. However, all these beneficial effects ceased to help when K took high values well over 100 and the processing cost was the dominant performance factor. Which of the two phenomena will prevail and where the turning point can be found at each time depends on the effectiveness of our pruning policy, which in turn is affected by the data distribution, how node splitting takes place, how the hierarchy is formed, and other less significant factors.

6. CONCLUSIONS AND FUTURE WORK

To recapitulate, we address the problem of search result diversification for multi-dimensional disk-resident data that cannot fit into main memory. To the best of our knowledge, we are the first to tackle this important problem in the context of database applications relying on widely used multi-dimensional indexes, like the R-tree and its variants. Hitherto, mostly combinatorial aspects of this problem have been considered which require scanning the entire data input when computing a diversified result. Above all, existing solutions are costly in terms of both I/Os and processing time. On the other hand, our method is I/O optimal in a sense that it does not access disk-pages that cannot contribute and improve the result-set, it does not require any pre-computation besides building the R-tree, it has limited main-memory requirements, and is easy to implement as it leverages best-first search. Moreover, the results verified the efficiency of our method as it outperforms a competitor scheme and a baseline based on exhaustive search.

Moreover, we are currently investigating the impact of different result initialization techniques on performance. In fact, we are in the process of designing alternative problem-specific initialization methods that reduce the required number of disk accesses and time to process a query without harming the quality of the result. Thereby, such a policy would involve the presented method to operate at a second stage of a more complex scheme. Last but not least, significant efforts have been made towards incorporating our techniques in distributed query processing algorithms.

7. REFERENCES

[1] R. Agrawal, S. Gollapudi, A. Halverson, and S. Ieong. Diversifying search results. In *WSDM*, pages 5–14, 2009.

[2] J. G. Carbonell and J. Goldstein. The use of mmr, diversity-based reranking for reordering documents and producing summaries. In *SIGIR*, pages 335–336, 1998.

[3] S. Gollapudi and A. Sharma. An axiomatic framework for result diversification. *IEEE Da. Eng. Bul.*, 32(4):7–14, 2009.

[4] E. Vee, J. Shanmugasundaram, and S. Amer-Yahia. Efficient computation of diverse query results. *IEEE Data Eng. Bull.*, 32(4):57–64, 2009.

[5] Y. Xu and H. Yin. Novelty and topicality in interactive information retrieval. *JASIST*, 59(2):201–215, 2008.

[6] C. Yu, L. V. S. Lakshmanan, and S. Amer-Yahia. It takes variety to make a world: diversification in recommender systems. In *EDBT*, pages 368–378, 2009.