Spatial Cohesion Queries

Dimitris Sacharidis
Technische Universität Wien
Vienna, Austria
dimitris@ec.tuwien.ac.at

Antonios Deligiannakis
Technical University of Crete
Chania, Greece
adeli@softnet.tuc.gr

ABSTRACT

Given a set of attractors and repellers, the cohesion query returns the point in database that is as close to the attractors and as far from the repellers as possible. Cohesion queries find applications in various settings, such as facility location problems, location-based services. For example, when attractors represent favorable plases, e.g., tourist attractions, and repellers denote undesirable locations, e.g., competitor stores, the cohesion query would return the ideal location, among a database of possible options, to open a new store. These queries are not trivial to process as the best location, unlike aggregate nearest or farthest neighbor queries, may be far from the optimal point in space. Therefore, to achieve sub-linear performance in practice, we employ novel best-first search and branch and bound paradigms that take advantage of the geometrical interpretation of the problem. Our methods are up to orders of magnitude faster than linear scan and adaptations of existing aggregate nearest/farthest neighbor algorithms.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications—*Spatial databases and GIS*

Keywords

nearest neighbor, farthest neighbor

1. INTRODUCTION

This paper introduces the *spatial cohesion query*. Assume a database $\mathcal{D}$ of point objects, a set $\mathcal{A}$ of attractors, and a set $\mathcal{R}$ of repellers, all located within some area; attractors and repellers need not be points, but could also be arbitrarily shaped regions. Then, the *attraction* of an object $\mathbf{o} \in D$ is the opposite of its minimum distance to any attractor among $\mathcal{A}$, while the *repulsion* of $\mathbf{o}$ is the opposite of its minimum distance to any repeller in $\mathcal{R}$. The spatial cohesion query returns the object $\mathbf{o}^* \in D$ that has maximum *cohesion*, i.e., maximizes the weighted difference of its attraction and repulsion. Intuitively, the result is an object that is both close to the attractors and far from the repellers.

The motivation for cohesion queries comes from various spatial optimization problems that seek to balance opposing forces. For example, consider a scenario where the goal is to determine a profitable location, among a list of vacancies, for opening a new tourist shop. The cohesion query models this as follows: vacancies constitute the database of objects; touristic attractions, popular intersections, city landmarks, etc., act as attractors; while existing competitor shops and bad neighborhoods act as repellers.

Moreover, cohesion queries appear as submodules in some common but computationally hard analytic methods. For instance, in *k-means clustering*, the goal is to partition the data into k clusters so as to minimize the sum of each object's distance to its closest cluster mean. A standard heuristic [12] for this NP-hard problem is to perform multiple passes over the objects, and determine at each pass, the best object to relocate from its cluster to another. This sub-problem can be stated as a cohesion query, where the cluster objects are the database, the cluster mean acts as the single repeller, and the other cluster means act as attractors.

As another analytic tool, consider the *spatial diversification problem*, where the goal is to determine a set of k objects that are *relevant*, i.e., close, to some given query location, and *dissimilar*, i.e., far, from each other. The standard approach for this NP-hard problem is to progressively construct the result set, where at each step, choose for inclusion in the result set the object that minimizes a weighted combination of two components. The first component is the relevance to the query location, i.e., the single attractor, while the second is the aggregate distance to objects already in the result set, i.e., the repellers.

Cohesion queries are reminiscent of nearest neighbor (NN) variants, such as the aggregate NN (ANN) query, which retrieves the object that is closest to a group of attractors, and the aggregate farthest neighbor (AFN) query, which retrieves the object that is farthest from a group of repellers. In contrast, the cohesion query seeks for the object that strikes the perfect balance between attraction and repulsion forces, an object that can significantly differ from the ANN or AFN answers.

To illustrate this, consider Figure 1 that depicts an attractor $\mathbf{a}$, two repellers $\mathbf{r}_1$, $\mathbf{r}_2$, and four objects, $\mathbf{o}_1$ through $\mathbf{o}_4$. We assume that distances are measured by the Euclidean

SIGSPATIAL'15, November 03–06, 2015, Bellevue, WA, USA

ACM 978-1-4503-3967-4/15/11...$15.00
DOI: http://dx.doi.org/10.1145/2820783.2820834.

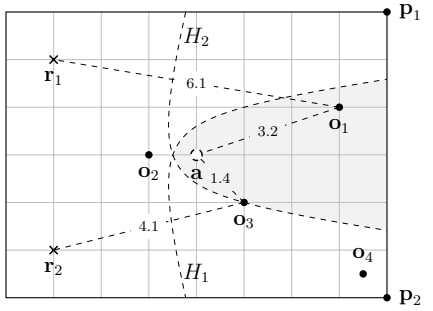


Figure 1: Cohesion query with one attractor a and two repellers $\mathbf{r}_1$, $\mathbf{r}_2$; object $\mathbf{o}_1$ has the largest cohesion of around 2.9

distance (we draw a grid for ease of reference), and seek to maximize the difference between attraction and repulsion (equal weights are assigned to the two forces). Observe that object $\mathbf{o}_2$ is the ANN of the attractor, while $\mathbf{o}_4$ is the AFN of $\{r_1, r_2\}$. The answer to the cohesion query is $\mathbf{o}_1$ with a cohesion of around 2.9, as the distance to its closest repeller (6.1) minus the distance to the attractor (3.2) is the largest; for example, $\mathbf{o}_3$ has a lower cohesion value of about $4.1 - 1.4 = 2.7$.

Moreover, the methodology used in processing NN variants does not apply for cohesion queries. These methods operate on the premise that the result lies near the optimal point in space under the corresponding objective function, and thus guide the search towards it. In ANN (resp. AFN) the point that minimizes (resp. maximizes) the min-aggregate distance to the attractors (resp. repellers) is an attractor (resp. a vertex in the bounded Voronoi diagram of the repellers; see [9]). For the cohesion query (under equal attraction, repulsion weights), the optimal point in space is also an attractor, but the actual result may not be close to the optimal point, or to the optimal points for ANN, AFN.

Returning to Figure 1, the optimal point in space for the cohesion query and ANN is the attractor $\mathbf{a}$, while the optimal points for AFN are $\mathbf{p}_1$, $\mathbf{p}_2$. Observe that the closest object to $\mathbf{a}$ is $\mathbf{o}_2$, while that closest to $\mathbf{p}_1$ or $\mathbf{p}_2$ is $\mathbf{o}_4$. However, neither object is the result ($\mathbf{o}_1$) to the cohesion query.

A simple method for processing a cohesion query is to perform a linear scan on the database $\mathcal{D}$ and compute the cohesion of each object. Another approach is to combine the ANN and AFN search, progressively retrieving objects from each until a common object is seen (or a threshold is exceeded), similar to top-k processing algorithms [7].

We propose a best-first search algorithm, termed BFS, suitable for tree-based space-partitioning indices, which, similar to other methods in its class (i.e., the BF algorithm of [13] for NN queries), defines an optimistic bound (admissible heuristic) for the cohesion function (e.g., *mindist* for NN queries), and uses it to guides the search. The efficiency of such a method depends on the tightness of the derived bounds. In fact, obtaining tight bounds on the cohesion of objects within sub-trees, would result in an index IO-optimal method, meaning that no other algorithm operating on the same index can perform fewer index node accesses. Unlike some NN variants though, it is computationally hard, if at all generally possible, to derive a tight bound for the cohesion function; it entails solving a non-smooth constrained optimization problem [8]. Nevertheless, our evaluation shows that a simple non-tight bound is able to achieve up to orders of magnitude better performance than existing methods.

Still, for some difficult settings, especially when the attraction and repulsion forces have equal weight, BFS performs poorly and at times worse than a linear scan. To address this issue, we take a branch and bound approach, termed BB, which introduces pruning criteria to eliminate objects that would be otherwise visited by BFS. As a result, BB is more efficient than BFS, often by a factor between 2 to 4. This is an interesting result, given than in other problems branch and bound may not outperform best-first search; e.g., for NN queries, the pruning criteria of [21] offer little benefit compared to the IO-optimal best-first search algorithm of [13]. The reason for the performance increase of BB is that the optimistic bounds we derive are not tight, and hence leave the door open for further pruning. Had they been tight, BB would not prune more objects than BFS, which is exactly the case in NN queries.

For the intuition behind BB's pruning, refer to Figure 1 and assume that object $\mathbf{o}_3$ has the best seen so far cohesion of about 2.7. Considering only repeller $\mathbf{r}_2$, the locus of points in space that have cohesion equal to that of $\mathbf{o}_3$'s defines the hyperbola branch H_2 (since the distance from $\mathbf{r}_2$ minus the distance from $\mathbf{a}$ is constant, given equal attraction, repulsion weights). Hyperbola branch H_1 is similarly defined for repeller $\mathbf{r}_1$ setting the difference of distances equal to the cohesion of $\mathbf{o}_3$. Based on these hyperbolas, it is possible to characterize the space with respect to best seen cohesion so far. In Figure 1, the shaded area, defined as the intersection of the interior of the two hyperbolas, contains points in space that have cohesion greater than $\mathbf{o}_3$'s, and thus may contain a better object, in our case $\mathbf{o}_1$. So, given any cohesion value (in our example 2.7), it is possible to define pruning criteria that eliminate parts of the space containing objects with lower cohesion. Then, the challenge is how to apply this idea to prune index sub-trees.

The contributions of this work are the following:

- We introduce and study the spatial cohesion query.
- We introduce a tree-based space-partitioning method, termed BFS, which follows the best-first search paradigm, by deriving optimistic bounds on the cohesion of objects within index subtree.
- We further introduce a branch and bound algorithm (BB) to further expedite cohesion query processing by introducing geometry-based pruning criteria.
- We extend our methods for non-point attractors and repellers, as well as for non-Euclidean distance metrics.
- We perform a detailed experimental study on real and synthetic data, showing that BFS is up to orders of magnitude faster than a simple linear scan and existing techniques based on NN query processing. Further, BB is shown to be about 2–4 times faster than BFS.

Outline. Section 2 reviews related work. Section 3 defines all concepts and formally states the cohesion query, while Section 4 presents baseline approaches. Then Section 5 introduces our best-first search algorithm, and Section 6 details our branch and bound approach. Section 7 discusses some extensions. Section 8 presents our experimental study, and Section 9 concludes this paper.

2. RELATED WORK

Nearest Neighbor Queries. There is an enormous body of work on the *nearest neighbor* (NN) query, also known as similarity search, which returns the object that has the smallest distance to a given query point; kNN queries output the k nearest objects in ascending distance. An overview of index-based approaches to accelerate the search can be found in [4]. Recently, more efficient approaches for metric spaces, e.g., [14], and high-dimensional data, e.g., [23], have been proposed.

For a set of query points, the *aggregate nearest neighbor* (ANN) query [19] retrieves the object that minimizes an aggregate distance to the query points. As an example, for the MAX aggregate function and assuming that the set of query points are users, and distances represent travel times, ANN outputs the location that minimizes the time necessary for all users to meet. In the case of the SUM function and Euclidean distances, the optimal location is also known as the Fermat-Weber point, for which no formula for the coordinates exists. The k-medoid problem is a generalization, which seeks a set of k objects that collectively minimizes an aggregate distance to the query points. The problem is NP-hard and has applications in clustering [17, 18].

A cohesion query, although also an optimization problem involving distances from a set of points (the repellers and the attractor), cannot be mapped to an ANN problem or its variants, and cannot be solved by adapting existing ANN algorithms. The reason is that the cohesion answer is an object that *maximizes* an aggregate distance to repellers (and minimizes the distance to the attractor), instead of minimizing an aggregate distance, as in ANN variants.

Cohesion queries are also related to aggregate farthest neighbor queries. The *farthest neighbor* (FN) query returns the object that has the largest distance to a given query point, and can be used, for example, to determine the minimum radius required to cover a set of points from a given location. Naturally, the *aggregate farthest neighbor* (AFN) query seeks the object that maximizes an aggregate distance to a set of query points. The work in [10] proposes an R-tree based algorithm for processing AFN queries for the SUM, MAX, MIN functions. Again, a cohesion query cannot be mapped to an AFN query and, thus, cannot be solved by algorithms for AFN queries, but an AFN algorithm together with an NN algorithm, can be used as a module for processing cohesion queries. This is the baseline approach described in Section 4. We note that the cohesion query is not related to reverse variants of the aforementioned problems, e.g., where the goal is to determine the objects that have a given query as their nearest neighbor [22, 16].

Diversification. The notion of (content-based) diversification first appears in information retrieval systems. The seminal work of [5] shows that a diversity-based reranking of the results list, which combines relevance and diversity similar to our formulation, achieves higher precision/recall values. An interesting study on diversification objectives is [11], which categorizes common diversification objectives and proves NP-hardness.

Note that, in general, diversification problems seek a *group* of documents that are relevant and diverse. Thus, they are not directly related to cohesion queries. However, subproblems similar to cohesion queries appear in many heuristic solutions to diversification problems. We emphasize that all referenced works in this section, unless stated otherwise, process these sub-problems by performing an exhaustive linear scan.

Table 1: Notation

Symbol	Definition
$\mathcal{D}, \mathbf{o}$	database of objects, an object
$\mathcal{A}, \mathbf{a}$	set of attractors, an attractor
$\mathcal{R}, \mathbf{r}$	the set of repellers, a repeller
$c(\mathbf{o})$	cohesion of $\mathbf{o}$ given $\mathcal{A}$ and $\mathcal{R}$
τ	a cohesion threshold

There are other types of diversification, such as coverage based approaches [1, 6], which however are not related to our problem. A more relevant line of work combines diversification and NN queries. [15] introduces the *k-nearest diverse neighbor* (kNDN) query, whose goal is to return a set of k objects that are as close as possible to a given query point, and at the same time no two objects have diversity below a given hard threshold. Similarly, [20] defines a variation of top-k search, adding the restriction that the result set must not contain any pair of objects having similarity above a user-specified hard threshold. In both these problems the hard threshold on diversity is fundamentally different than our notion of repulsion; hence such methods do not apply for cohesion queries.

A more related problem appears in [24], where relevance is defined as the distance to a query point, and diversity is defined either as the smallest distance (a formulation similar to cohesion queries) or the angle similarity to an object in the result. The proposed algorithms, however, avoid an exhaustive linear scan only for the angle-defined diversity. Moreover, the function that combines relevance and diversity is not a weighted combination, and thus their solutions do not apply for cohesion.

The most related diversification problem appears in [9]. Their iterative approach solves a sub-problem identical to a restricted version of the cohesion query, when we restrict cohesion to Euclidean space and distance. Using cohesion terminology, the key idea of [9] is to alternate between NN retrievals from the attractor and from certain points computed using the Voronoi diagram of the repellers. We note that [9] is proposed for a more restrictive setting, where only NN modules are available, and thus its application to cohesion queries does not result in a strong competitor.

3. PROBLEM DEFINITION

We now present the necessary definitions and formally introduce the cohesion query. Table 1 gathers the most important symbols used throughout this paper.

Consider a set $\mathcal{D}$, termed the *database*, of point objects. Given a set $\mathcal{A}$ of *attractors*, such that $\mathcal{A} \cap \mathcal{D} = \varnothing$, the *attraction* of an object $\mathbf{o} \in \mathcal{D}$ is defined as:

$$a(\mathbf{o}) = -\min_{\mathbf{a} \in \mathcal{A}} d(\mathbf{o}, \mathbf{a}),$$

where d denotes the Euclidean distance. Note that in the following, we assume that attractors and repellers are point objects; the discussion about non-point objects as well as the necessary changes to our methodology is deferred until Section 7.1. Moreover, the discussion about other distance metrics is in Section 7.2. The smaller the distance to the closest attractor is, the greater the attraction. The object that maximizes the attraction is the min-aggregate nearest

neighbor (ANN) of $\mathcal{A}$.

Given a set $\mathcal{R}$ of *repellers*, such that $\mathcal{R} \cap \mathcal{D} = \varnothing$, the *repulsion* of object $\mathbf{o} \in \mathcal{D}$ is defined as:

$$r(\mathbf{o}) = -\min_{\mathbf{r} \in \mathcal{R}} d(\mathbf{o}, \mathbf{r}).$$

The smaller the distance to the closest repeller is, the greater the repulsion. The object that minimizes the repulsion is the min-aggregate farthest neighbor (AFN) of $\mathcal{R}$.

Given a set of attractors $\mathcal{A}$ and a set of repellers $\mathcal{R}$, we define the *cohesion* of an object $\mathbf{o}$ to be equal to the weighted difference of its attraction and repulsion:

$$c(\mathbf{o}) = \lambda \cdot a(\mathbf{o}) - r(\mathbf{o}) = \min_{\mathbf{r} \in \mathcal{R}} d(\mathbf{o}, \mathbf{r}) - \lambda \cdot \min_{\mathbf{a} \in \mathcal{A}} d(\mathbf{o}, \mathbf{a}), \quad (1)$$

where the single weight λ controls the relative strength of attraction and repulsion.

In this work we answer the cohesion query: how to efficiently find the object that has the highest cohesion.

Problem 1. **[Cohesion Query]** Given attractors $\mathcal{A}$ and repellers $\mathcal{R}$, find an object $\mathbf{o}^* \in \mathcal{D}$ such that $\mathbf{o}^* = \operatorname{argmax}_{\mathbf{o} \in \mathcal{D}} c(\mathbf{o})$.

A final note concerns the extension of the previous definition to the corresponding top-k problem, i.e., determining the k objects with the highest cohesion values. Adapting our methods to process such a query in a progressive manner is straightforward and it is not further discussed.

4. BASELINE METHODS

A simple baseline approach is to exhaustively scan all objects, compute their cohesion, and then determine the object with the highest one. We refer to this method as LIN.

Another baseline processing technique is to decompose a cohesion query into an aggregate nearest neighbor (ANN) query on the set of attractors and an aggregate farthest neighbor (AFN) query on the set of repellers. The basic idea is to retrieve, in a round robin manner, objects from the ANN and the AFN search until a termination condition is met. Therefore, we require modules capable of *progressively* processing ANN and AFN queries. For ANN queries, we use the MBM algorithm [19], whereas for AFN queries the algorithm of [10]. We now present this method, which we term RR.

The algorithm maintains two threshold values, τ_a, τ_r, which represent the smallest possible aggregate distance of an object not yet retrieved by the ANN search, and the largest possible aggregate distance of an object not yet seen in the AFN search, initially set to 0 and ∞, respectively. Also RR initializes the result $\mathbf{o}^*$ to null and the next search module to ANN. Then, RR begins a loop with progressive object retrievals, until the largest attainable cohesion by retrieving additional objects, which is $\tau_r - \lambda \cdot \tau_a$, drops below the current best cohesion. At each iteration, a single object is retrieved, the appropriate threshold is updated, and the other search method is set for the next retrieval. Because the search modules are progressive, the thresholds set are a lower bound on the aggregate distance to $\mathcal{A}$, and an upper bound on the aggregate distance to $\mathcal{R}$ of any object not seen in ANN and AFN search, respectively. The answer object is updated if an object with better cohesion is retrieved.

As another baseline, we adapt the SPP algorithm from [9], which was proposed for a top-k diversification setting. Similar to RR, our SPP implementation employs an ANN search module, but unlike RR it does not use an AFN module. Instead, SPP determines probing locations around which the aggregate most farthest neighbor should lie and performs NN search around them. These probing locations are the vertices of the Voronoi diagram computed over the set of repellers. SPP uses a NN search module for each probing location and maintains a threshold for each. A round robin strategy for selecting the next search module is also used in SPP; the authors also examine more elaborate strategies, but with no significant performance gains. Our experimental study has shown that SPP performs much worse than RR, both in terms of I/O operations and CPU time, because the objects retrieved by the NN search around the probing locations are not guaranteed to maximize the aggregate distance from the repellers, and the thresholds used are CPU intensive, requiring the computation of intersections between circles and Voronoi edges.

Algorithm 1: RR

```
Input: index T; attractors A; repellers R
Output: o* the answer to the cohesion query
Variables: cohesion threshold τ
1  o* ← ∅;  τ_a ← 0;  τ_r ← ∞;  search ← ANN
2  while o* = ∅ or τ_r − λ · τ_a > c(o*) do
3      if search = ANN then
4          o ← ANN.getNext()
5          τ_a ← d(o, a)
6          search ← AFN
7      else
8          o ← AFN.getNext()
9          τ_r ← min_{r∈R} d(o, r)
10         search ← ANN
11     if c(o) > c(o*) then
12         o* ← o
```

5. THE BEST-FIRST SEARCH METHOD

This section describes an index-based *Best-First Search* algorithm, denoted as BFS, for processing cohesion queries. The method assumes a hierarchical space-partitioning index on the set of objects.

Therefore, we consider a tree structure T that indexes the database of objects $\mathcal{D}$. A node N of the index corresponds to a subtree rooted at N and hierarchically indexes all objects that reside in this subtree. In the following, we abuse notation and refer to N as the set of all objects that reside in N's subtree. To facilitate object retrieval, the index keeps aggregate information about the objects within N and stores it in an entry at the parent node of N; to simplify notation we simply refer to the entry for N as the node N. The aggregate information, which depends on the type of the tree, is typically the minimum bounding rectangle (MBR) or sphere (MBS) that covers all objects within N.

In the remainder of this paper, we assume that T is an R*-Tree [2], since it is perhaps the most well-known and studied spatial index. We note, that our methodology does not depend on the exact index type and is readily applicable to other indices.

As is characteristic to any best-first search method, BFS requires an optimistic bound (admissible heuristic) on the cohesion of objects contained within a particular subtree, represented by an index node. For cohesion queries, optimistic translates into an upper bound. Given a tree node N, let $\mathbf{o} \in N$ denote that object $\mathbf{o}$ is contained in the subtree rooted at N. Also, assume $d^+(N, \mathbf{x})$ (resp. $d^-(N, \mathbf{x})$) denote an upper (resp. lower) bound on the distance $d(\mathbf{o}, \mathbf{x})$

Algorithm 2: BFS

```
Input: index T; attractors 𝒜; repellers ℛ
Output: o* the answer to the cohesion query
Variables: H a heap with nodes sorted by c+()
1 H ← ∅
2 N_x ← N_root                                   ▷ root node of T
3 while N_x is an internal node do
4     read node N_x
5     foreach child N of N_x do
6         compute c+(N)                          ▷ Lemma 1
7         H.push(N, c+(N))
8     N_x ← H.pop()
9 o* ← N_x
```

of any object $\mathbf{o} \in N$ from point $\mathbf{x}$. Then, it is easy to construct an upper bound on the cohesion of any object within a node, as follows.

Lemma 1 (Upper Bound)**.** Given a non-leaf node N, the cohesion of an object $\mathbf{o} \in \mathcal{D}$ within N cannot be more than $c^+(N) = \min_{\mathbf{r}\in\mathcal{R}} d^+(N,\mathbf{r}) - \lambda \cdot \min_{\mathbf{a}\in\mathcal{A}} d^-(N,\mathbf{a})$.

Proof. Follows from the definitions of d^- and d^+. □

Note that even though the distance bounds d^-, d^+ are tight, the resulting cohesion bound of Lemma 1 may not be. This occurs because the point in the space contained by a node N that gives the distance bound for the attraction part may not coincide with the respective point for the repulsion part. Deriving a tight bound means finding the point in the space of N that maximizes the cohesion function. This essentially entails solving a non-smooth constrained optimization problem, which is computationally challenging and outside the scope of this paper. Nonetheless, our evaluation has shown that, in most cases, this bound suffices as it results in significant speedup over baseline methods.

Lemma 1 provides BFS with the means to guide the search, visiting more promising nodes first. Algorithm 2 shows the pseudocode of BFS. It takes as input the index T storing all objects in the database $\mathcal{D}$, the attractors $\mathcal{A}$, and the repellers $\mathcal{R}$, and returns the object $\mathbf{o}^*$ with the largest cohesion.

BFS directs the search using the heap H, which contains index nodes and is sorted descending on their upper bound on cohesion, computed according to Lemma 1; initially H is empty (Line 1). BFS performs a number of iterations (Lines 3–8). At the end of each iteration the node N_x at the top of the heap is popped (Line 8); for the first iteration N_x is set to the root node of T (Line 2). Index traversal terminates when node N_x is an external node, corresponding to an object, which in this case is the answer $\mathbf{o}^*$ (Line 9). Assuming that N_x is an internal node, BB reads this node from disk (Line 4), and for each child (Lines 5–8) it computes its upper cohesion bound (Line 6) and inserts it into the heap (Line 7). We next prove the correctness of BFS.

Theorem 1. The BFS algorithm returns the object with the largest cohesion.

Proof. BFS terminates when it pops from the heap a non-index node corresponding to object $\mathbf{o}_x$. As the heap contains nodes sorted by the upper bound of Lemma 1, it holds that $c(\mathbf{o}_x) \geq c^+(N)$ for all nodes in H. Therefore $\mathbf{o}_x$ has higher cohesion than any object within any node in the heap and thus any object in $\mathcal{D}$. □

6. THE BRANCH AND BOUND METHOD

This section describes an index-based *Branch and Bound* algorithm, denoted as BB, for processing cohesion queries. BB has the same index requirements as BFS and also uses an optimistic cohesion bound to direct the search towards promising nodes. In addition, BB applies the branch and bound paradigm to prune parts of the space (subtrees rooted at nodes) that may not contain the most cohesive object. In particular, BB: (1) computes pessimistic cohesion bounds on index nodes to derive threshold τ, which acts as a lower bound on the solution to the cohesion query, and (2) employs two pruning criteria to eliminate nodes containing objects with cohesion smaller than τ.

Computing the Threshold. Using the distance bounds d^-, d^+ of Section 5, we can also compute a lower bound on the cohesion of any object within a tree node.

Lemma 2 (Lower Bound)**.** Given a non-leaf node N, the cohesion of an object $\mathbf{o} \in \mathcal{D}$ within N cannot be less than $c^-(N) = \min_{\mathbf{r}\in\mathcal{R}} d^-(N,\mathbf{r}) - \lambda \cdot \min_{\mathbf{a}\in\mathcal{A}} d^+(N,\mathbf{a})$.

Proof. Follows from the definitions of d^+ and d^-. □

The threshold τ is set to the largest among the lower cohesion bound of any seen node and the cohesion values of any seen object.

Pruning Criteria. The discussion here assumes a cohesion threshold value τ is computed. The following theorem, which is a direct consequence of the coherence definition, determines whether an object $\mathbf{o}$ can be pruned given an attractor and a repeller.

Theorem 2. Given attractors $\mathcal{A}$, a repeller $\mathbf{r} \in \mathcal{R}$ and a cohesion threshold τ, any object $\mathbf{o} \in \mathcal{D}$ such that $d^r(\mathbf{o},\mathbf{r}) - \lambda \cdot \min_{\mathbf{a}\in\mathcal{A}} d^a(\mathbf{o},\mathbf{a}) < \tau$ has cohesion less than τ.

Theorem 2 can prune individual objects. However, we need a method to prune an entire subtree rooted at a given tree node. We thus consider the aggregation information stored within a node. Then, Criterion 1 holds.

Criterion 1. Given a cohesion threshold τ, a node N contains objects with cohesion less than τ, if there exists a repeller $\mathbf{r} \in \mathcal{R}$ such that for every attractor $\mathbf{a} \in \mathcal{A}$ it holds that $d^+(N,\mathbf{r}) - \lambda \cdot d^-(N,\mathbf{a}) < \tau$.

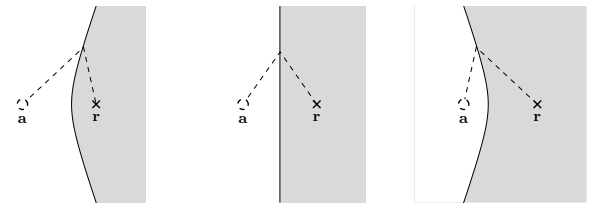
Proof. From the definitions of $d^+(N,\mathbf{a})$ and $d^-(N,\mathbf{r})$, we derive that $\forall \mathbf{a} \in \mathcal{A}$, $\exists \mathbf{r} \in \mathcal{R}$ such that $d^r(\mathbf{o},\mathbf{r}) - \lambda \cdot d^a(\mathbf{o},\mathbf{a}) \leq d^+(N,\mathbf{r}) - \lambda \cdot d^-(N,\mathbf{a}) < \tau$ and, thus, Theorem 2 applies for all objects within N. □

Criterion 1 is simple to check and succesfully prunes nodes. However, it is based on rather loose bounds on the cohesion of objects within nodes. To understand this, consider the geometric interpretation of Theorem 2 for the case of $\lambda = 1$, i.e., when attraction and repulsion forces are equally weighted, which is the most computationally challenging case for all algorithms as our experiments have shown.

We study the geometry of the function $f(\mathbf{x}) = d(\mathbf{x},\mathbf{r}) - d(\mathbf{x},\mathbf{a})$, where $\mathbf{x}$ is a point in the vector space. Then the locus of points $\mathbf{x}$ satisfying equation $f(\mathbf{x}) = \tau$, for a given constant τ, defines one of the two branches of a hyperbola curve with foci the attractor $\mathbf{a}$ and the repeller $\mathbf{r}$. Particularly, we distinguish three cases with respect to τ's value.

(a) When $\tau < 0$, the locus is the branch around $\mathbf{r}$. Theorem 2 implies that any object that lies inside this branch (i.e., in the part of space containing focus $\mathbf{r}$) has cohesion less than τ; see Figure 2a.

(b) When $\tau = 0$, the locus is the bisector of segment $\mathbf{ar}$. Theorem 2 implies that any object that lies closer to the



(a) $\tau < 0$ (b) $\tau = 0$ (c) $\tau > 0$

Figure 2: Geometric interpetation of Theorem 2 for $\lambda = 1$; locus of points x satisfying $d(\mathbf{x}, \mathbf{r}) - d(\mathbf{x}, \mathbf{a}) < \tau$

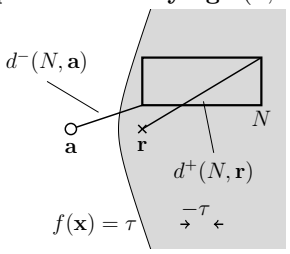


Figure 3: Criterion 2 for $\lambda = 1$ and $\tau < 0$

repeller $\mathbf{r}$ than the attractor $\mathbf{a}$ has cohesion less than τ; see Figure 2b.

(c) When $\tau > 0$, the locus is the branch around the attractor $\mathbf{a}$. Theorem 2 implies that any object that lies outside this branch (i.e., in the part of space containing focus $\mathbf{r}$) has cohesion less than τ; see Figure 2c.

The pruned space increases with τ. A higher τ value causes the locus to move closer to $\mathbf{a}$, and the corresponding branch to become narrower.

Now, let us turn our attention to the case of $\tau < 0$, and consider a node N, attractor $\mathbf{a}$, and repeller $\mathbf{r}$ as shown in Figure 3; the absolute value of τ depicted on the bottom right. Node N lies completely within the shaded area, and thus cannot contain any points with cohesion more that τ. Observe that Criterion 1 does not hold for N. The upper distance bound $d^+(N, \mathbf{r})$ to the repeller is greater than the lower distance bound $d^-(N, \mathbf{a})$ to the attractor, and thus $d^+(N, \mathbf{r}) - d^-(N, \mathbf{a}) > 0$, which is clearly greater than the negative threshold τ.

The reason for the previous observation is that the point within N that has attraction from $\mathbf{a}$ equal to $d^-(N, \mathbf{a})$ and the point that has repulsion from $\mathbf{r}$ equal to $d^-(N, \mathbf{r})$ do not coincide in general. In Figure 3, the former is the bottom left corner of N, while the latter is the top right corner of N. As a result, the value $d^-(N, \mathbf{r}) - d^-(N, \mathbf{a})$ is not a tight upper bound for the cohesion of any object within N.

In what follows, we derive a stronger criterion for pruning nodes when $\lambda = 1$ and $\tau \leq 0$. The key observation is that in this case the pruned space is convex.

Lemma 3. Given repeller $\mathbf{r}$, an attractor $\mathbf{a}$, and a cohesion threshold $\tau \leq 0$, the space defined by points $\mathbf{x}$ such that $d(\mathbf{x}, \mathbf{r}) - d(\mathbf{x}, \mathbf{a}) < \tau$ is convex.

Proof. The equation $d(\mathbf{x}, \mathbf{r}) - d(\mathbf{x}, \mathbf{a}) = \tau$ defines a hyperbola branch around $\mathbf{r}$ for $\tau < 0$. The inequality $d(\mathbf{x}, \mathbf{r}) - d(\mathbf{x}, \mathbf{a}) < \tau$ defines the space inside the branch, i.e., that contains $\mathbf{r}$, which is known to be convex. For $\tau = 0$ the branch reduces to the bisector and the half-space closer to $\mathbf{r}$ is convex. □

Convexity is desirable for its following property.

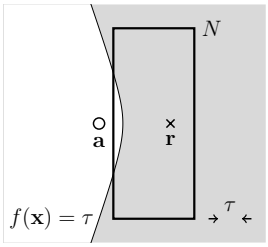


Figure 4: Non-convex pruning area for $\lambda = 1$, $\tau > 0$

Lemma 4. A rectangle R is completely inside a convex space S if and only if all its corners are inside the convex space S.

Proof. The $\Rightarrow$ direction is obvious. For the $\Leftarrow$ direction, assume otherwise that all corners of R are inside S, but there exists a point $\mathbf{p}$ inside R which is outside S. From the convexity of S, we derive that each edge of R is completely inside S (any point in an edge of R is a linear combination of two corners). With similar reasoning, we derive that any face of R is completely inside S. Then, since $\mathbf{p}$ can be written as a linear combination of two points on faces of R, point $\mathbf{p}$ must also be completely inside S — a contradiction. □

Combining the previous two lemmas we derive the following pruning criterion.

Criterion 2 ($\lambda = 1$, $\tau \leq 0$)**.** Given attractors $\mathcal{A}$, a repeller $\mathbf{r} \in \mathcal{R}$, and a cohesion threshold $\tau \leq 0$, a node N contains objects with cohesion less than τ for $\lambda = 1$, if for each corner $\mathbf{c}$ of N it holds that $d(\mathbf{c}, \mathbf{r}) - \min_{\mathbf{a} \in \mathcal{A}} d(\mathbf{c}, \mathbf{a}) < \tau$.

Proof. From Lemmas 3 and 4, we derive that for any point $\mathbf{x}$ inside R it holds that $d(\mathbf{x}, \mathbf{r}) - \min_{\mathbf{a} \in \mathcal{A}} d(\mathbf{x}, \mathbf{a}) < \tau$. Thus Theorem 2 holds for all points in R. □

Consider again the example of Figure 3, where the prerequisites ($\lambda = 1$, $\tau \leq 0$) of Criterion 2 hold. It is easy to see that all corners of node N are within the shaded are, and thus Criterion 2 prunes this node.

Unfortunately, the pruning space for $\tau > 0$ is not convex, meaning that Criterion 2 does not apply. Consider Figure 4, which depicts an example for $\lambda = 1$ and $\tau > 0$. Observe that while all corners of node N are within the shaded area, there exists a part of N that is outside; hence, the node cannot be pruned.

The BB Algorithm. Algorithm 3 shows the pseudocode of the BB algorithm. It takes as input the index T storing all objects in the database $\mathcal{D}$, the set of attractors $\mathcal{A}$, and the set of repellers $\mathcal{R}$. BB returns the object $\mathbf{o}^*$ with the largest cohesion.

BB operates largely similar to BFS. It also uses a data structure L containing index nodes sorted descending on their upper bound on cohesion (Lemma 1). BB requires sorted access, and thus L could be implemented as a binary search tree. In addition, BB maintains a cohesion threshold τ, which corresponds to a lower bound of the maximum cohesion, initially set to $-\infty$. BB operates similar to BFS with the following exceptions. BB updates the threshold (Lines 8–10) whenever a node with a higher upper bound or an object with a higher cohesion is seen. Also, BB marks this event raising a flag (Line 10). Subsequently, after all children nodes are inserted in L and if the flag was raised

Algorithm 3: BB

```
   Input: index T; attractors 𝒜; repellers ℛ
   Output: o* the answer to the cohesion query
   Variables: L a list with nodes sorted by c+(); cohesion
              threshold τ
1  τ ← −∞; L ← ∅
2  N_x ← N_root                                   ▷ root node of T
3  while N_x is an internal node do
4      read node N_x
5      foreach child N of N_x do
6          compute c+(N)                          ▷ Lemma 1
7          L.insert(N, c+(N))
8          if c−(N) > τ then                      ▷ Lemma 2
9              τ ← c−(N)
10             flag ← true
11     if flag = true then
12         flag ← false
13         foreach N ∈ L do
14             foreach r ∈ ℛ do
15                 pruned ← true
16                 foreach a ∈ 𝒜 do
17                     if d+(N, r) − λ · d−(N, a) ≥ τ then ▷ Criterion 1
18                         pruned ← false
19                         break
20                     if λ = 1 and τ ≤ 0 and ∃ corner c of N :
                       d(c, r) − d(c, a) ≥ τ then  ▷ Criterion 2
21                         pruned ← false
22                         break
23                 if pruned = true then
24                     L.erase(N)
25                     break
26     N_x ← L.pop()
27 o* ← N_x
```

Algorithm 4: BB_PruneCheck

```
   Input: node N; attractor a; repeller r
1  pruned ← d−(N, r) − λ · d−(N, a) < τ           ▷ Criterion 1
2  return pruned
```

(Line 11), the list L of nodes is traversed (Lines 13–25), and Criteria 1 and 2 are examined. If there exists a repeller $\mathbf{r}$ such that after considering all attractors the pruned flag remains true based on bothe criteria (Lines 17–22), then the currently examined node is pruned (Lines 23–25). The next theorem proves the correctness of BB.

Theorem 3. The BB algorithm returns the object with the largest cohesion.

Proof. BB operates as BFS with the addition of the pruning criteria. We only need to show that BB cannot miss the answer object $\mathbf{o}^*$ due to pruning. BB prunes index nodes based on Criteria 1 and 2, and the threshold computed based on Lemma 2. Therefore, by the correctness of these lemmas, $\mathbf{o}^*$ cannot be in any pruned node. □

7. EXTENSIONS

Section 7.1 discusses the case of non-point attractors and repellers, and Section 7.2 overviews other distance metrics.

7.1 Non-point Attractors and Repellers

We assume that attractor and repellers are not points and define a simply connected area, i.e., without holes. Consider such an area A. The distance $d(\mathbf{p}, A)$ of an arbitrary point $\mathbf{p}$ in the space to area A is defined as the minimum distance of $\mathbf{p}$ to any point within A.

Depending on the complexity of the area A, computing distances to A can be computationally hard. Therefore, we show how to use its minimum bounding rectangle, denoted as $\overline{A}$, to compute lower and upper bounds on the distance to A of points and tree nodes.

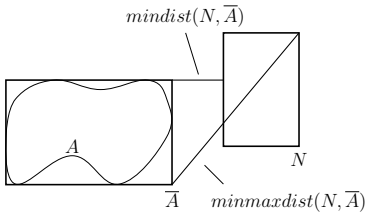


Figure 5: Distance bounds of node N to area A

We first start with a point $\mathbf{p}$ in space. It is easy to see that the following holds:

$$mindist(\mathbf{p}, \overline{A}) \leq d(\mathbf{p}, A) \leq minmaxdist(\mathbf{p}, \overline{A}),$$

where $mindist(\mathbf{p}, \overline{A})$ is the minimum possible distance of $\mathbf{p}$ to any point within $\overline{A}$, and $minmaxdist(\mathbf{p}, \overline{A})$ is the minimum across all faces of $\overline{A}$ of the maximum possible distance of $\mathbf{p}$ to any point on an $\overline{A}$ face.

We next consider the case of a node N and seek to bound the distance to area A of any object $\mathbf{o}$ within N. A similar result holds for any $\mathbf{o} \in N$:

$$mindist(N, \overline{A}) \leq d(\mathbf{o}, A) \leq minmaxdist(N, \overline{A}), \quad (2)$$

where $mindist(N, \overline{A})$ is the minimum possible distance of any point in N to any point in $\overline{A}$, and $minmaxdist(N, \overline{A})$ is the minimum across all faces of $\overline{A}$ of the maximum possible distance of any point within N to any point on an $\overline{A}$ face.

Figure 5 depicts the previous bounds for an area A bounded by $\overline{A}$ and node N. Note that the only necessary change to the algorithms is that whenever bounds $d^{\pm}(N, \cdot)$ of the distance of a node N to a non-point attractor or repellers are required, Equation 2 is employed. A final note is that Criterion 2 does not apply for non-point attractors and repellers, simply because the geometric interpretation of the pruning area is no longer defined by a hyperbola.

7.2 Other Distance Metrics

An MBR-based node N is associated with a rectangle defined by its lower $N.\boldsymbol{\ell}$ and upper $N.\mathbf{u}$ corners. In what follows, we seek to bound $d(\mathbf{o}, \mathbf{p})$ for an object $\mathbf{o}$ within N, and some object $\mathbf{p}$ in the space, for the L_p *distance metrics*:

$$d_p(\mathbf{o}, \mathbf{p}) = \left(\sum_{i=1}^{|\mathcal{A}|} |\mathbf{o}[i] - \mathbf{p}[i]|^p \right)^{1/p}.$$

Consider the points $\mathbf{x}_p^-$, $\mathbf{x}_p^+$ inside N's MBR defined as:

$$\mathbf{x}_p^-[i] = \begin{cases} N.\boldsymbol{\ell}[i] & \text{if } \mathbf{p}[i] < N.\boldsymbol{\ell}[i] \\ \mathbf{p}[i] & \text{if } N.\boldsymbol{\ell}[i] \leq \mathbf{p}[i] \leq N.\mathbf{u}[i] \\ N.\mathbf{u}[i] & \text{if } \mathbf{p}[i] > N.\mathbf{u}[i] \end{cases}, \text{ and}$$

$$\mathbf{x}_p^+[i] = \begin{cases} N.\mathbf{u}[i] & \text{if } \mathbf{p}[i] < \frac{1}{2}(N.\boldsymbol{\ell}[i] + N.\mathbf{u}[i]) \\ N.\boldsymbol{\ell}[i] & \text{if } \mathbf{p}[i] \geq \frac{1}{2}(N.\boldsymbol{\ell}[i] + N.\mathbf{u}[i]) \end{cases}.$$

Then, define values: $d_p^-(N, \mathbf{p}) = d_p(\mathbf{x}_p^-, \mathbf{p})$ and $d_p^+(N, \mathbf{p}) = d_p(\mathbf{x}_p^+, \mathbf{p})$ for which the following lemma holds.

Lemma 5. The values $d_p^-(N, \mathbf{p})$, $d_p^+(N, \mathbf{p})$ are a tight lower and a tight upper bound, respectively, on $d_p(\mathbf{o}, \mathbf{p})$ for any object $\mathbf{o} \in N$ and an arbitrary object $\mathbf{p}$.

Proof. The function $f(\mathbf{x}) = \left(\sum_{i=1}^{|\mathcal{A}|} |\mathbf{x}[i]|^p \right)^{1/p}$ is non-decreasing

Table 2: Dataset Characteristics

Dataset	Cardinality	Dimensions	Attraction	Repulsion
SYNTH	$5 \cdot 10^6 - 10^8$	2 – 10	d_2	d_2
FACTUAL	2,120,732	2	d_2	d_2
MIRFLICKR	1,000,000	50	d_1	d_1

monotonous in each dimension i. Therefore, it holds that the lowest (resp. highest) possible values of $\mathbf{x}$ in all dimensions gives a lower (resp. upper) bound for $f()$.

Observe that $|\mathbf{o}[i]-\mathbf{p}[i]| \geq |\mathbf{x}_p^-[i]-\mathbf{p}[i]|$ and $|\mathbf{o}[i]-\mathbf{p}[i]| \leq |\mathbf{x}_p^+[i]-\mathbf{p}[i]|$ for any $\mathbf{o}[i] \in [N.\boldsymbol{\ell}[i], N.\mathbf{u}[i]]$. Hence, the lemma follows from the monotonicity of the L_p distance metric and the fact that the specified $\mathbf{x}_p^-, \mathbf{x}_p^+$ points that determine the values of $d_p^-(N,\mathbf{p})$ and $d_p^+(N,\mathbf{p})$ reside within N. □

8. EXPERIMENTAL EVALUATION

Section 8.1 describes the experimental setting, while Section 8.2 presents the results.

8.1 Experimental Setting

Methods. We implement our proposed BFS and BB algorithms, discussed in Sections 5–6, for processing cohesion queries over R-Trees. Moreover, we also implement the baseline LIN algorithm, which performs an exhaustive linear scan over the database of objects, as well as methods RR and SPP described in Section 4. All algorithms are implemented in C++ and executed on a 3GHz machine. We note that SPP was consistenly slower than RR as it makes a series of expensive computations (finding intersections between Voronoi edges and circles defining the search frontier [9]); hence SPP is omitted from all figures. Moreover RR's performance was better than LIN only for very small or large values of weight λ. Since we focus on the hard cases ($\lambda = 1$ and close values), we only include RR in the first set of figures.

Datasets. Our evaluation includes both real and synthetic datasets, whose characteristics are shown in Table 2. The synthetic datasets, denoted as SYNTH, contain objects that are randomly distributed around 1,000 cluster centers, selected independently and uniformly at random. The probability that a cluster center attracts objects is drawn from a Zipfian distribution with skew (zipfian parameter) 0.8.

The real dataset FACTUAL is a collection of 2,120,732 locations of places[1] (restaurants, shops, etc.) in the U.S. To check the applicability of our methods for other distance metrics, we use another real dataset, denoted as MIRFLICKR, which is a collection of 1,000,000 images used in the evaluation of content-based image retrieval methods.[2] In our experiments, we use the first 50 buckets (out of 150) of edge histogram descriptors, of the MPEG-7 specification [25], as the feature vector. This is thus a high dimensional data set, where indices are expected to be less helpful in pruning the search space. For the MIRFLICKR dataset, the L_1 norm (d_1) is used as the distance metric.

Parameters, queries and metrics. We study the performance of the algorithms by varying four parameters: (1) the number of objects $|\mathcal{D}|$, from 5M up to 100M in SYNTH, (2) the dimensionality of the space $|\mathbb{S}|$, from 2 up to 10 in SYNTH and from 5 up to 50 in MIRFLICKR, (3) the number of repellers $|\mathcal{R}|$ from 1 up to 1000, and (4) the weight parameter λ from 0.1 up to 10. The default values of these parameters are specified in Table 3. Note that in all experiments we assume a single attractor, $|\mathcal{A}| = 1$. As we will demonstrate in our experimental evaluation, our selected value of $\lambda = 1$ is actually a worst-case scenario for our algorithms, since their I/O and running time improvements compared to LIN quickly improve as the value of λ deviates from 1 (either higher or lower values of λ).

[1] Retrieved using the API `http://www.factual.com/data/t/places`

[2] Available at `http://press.liacs.nl/mirflickr/`

Table 3: Parameters

Parameter	Symbol	Range	Default
Number of Repellers	$\|\mathcal{R}\|$	1 – 1000	10
Weight	λ	0.1 – 10	1
Cardinality (SYNTH)	$\|\mathcal{D}\|$	$10^6 - 10^8$	10^7
Dimensionality (SYNTH)	$\|\mathbb{S}\|$	2 – 10	2

Table 4: Pruning Power ($|\mathcal{R}| = 10$)

λ	Criterion 1	Criterion 2
0.5	75%	—
0.9	59%	—
1	59%	45%
1.1	60%	—
2	63%	—

In each experiment, the set of attractors is constructed by performing an $|\mathcal{A}|$-NN query on a point uniformly selected from the space at random. For the NBA dataset, this point is a tuple with attributes values the best in all statistics. The set of repellers is constructed progressively: we pose $|\mathcal{R}|$ cohesion queries with $\mathcal{A}$ constructed as before and $\mathcal{R}$ initially empty, inserting the result of each query to $\mathcal{R}$. After these steps, we obtain the set of attractors and repellers that will ultimately be used in our evaluation for cohesion queries. To quantify performance, we measure the number of I/O operations, and the processing time for a cohesion query. All reported quantities for all algorithms are measured *after* the attractors and repellers have been chosen. The reported values are in each case the averages of 10 distinct queries.

8.2 Results

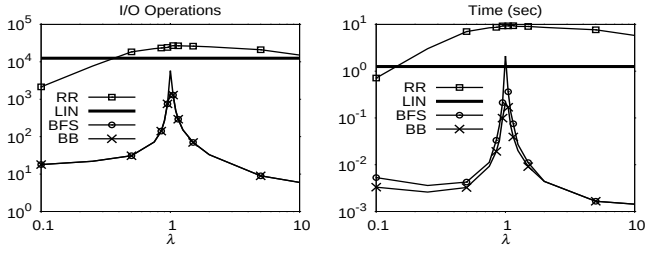
Effect of λ. We first study the effect of the weight λ as it varies from 0.1 up to 10 at the FACTUAL and SYNTH datasets. The remaining parameters obtain their default values, depicted in Table 3.

The results for the number of required I/O operations and the total running time (in seconds) of the algorithms are depicted in Figures 6 and 7 for FACTUAL and SYNTH, respectively. The findings are identical between the datasets. Please note that the y-axis in the figures is often in logarithmic scale. In this case, the x-axis is also logarithmic.

The number of I/O operations and running time in LIN is independent of λ. The performance of RR is often worse than LIN, especially its running time. Since RR was routinely outperformed by our algorithms, we omit it from the remaining experiments. Note that our adaptation of the SPP algorithm described in Section 2 performed even worse than RR and it is also omitted. The effect of λ in all other methods is similar. Our proposed algorithms offer significant (up to 3 orders of magnitude) I/O and running time savings over the LIN method, especially for values of λ much lower or higher than 1. This behavior is inherent in cohesion queries and explains why they are more challenging than NN or AFN queries. Large λ values assign more weight to attraction, and thus cohesion query processing resembles

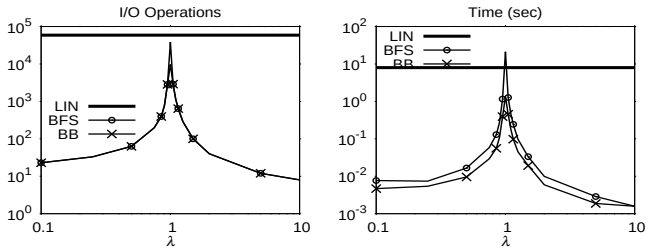
Table 5: Pruning Power ($\lambda = 1$)

$\lvert\mathcal{R}\rvert$	Criterion 1	Criterion 2
1	16%	4%
5	61%	28%
10	59%	45%
50	62%	40%
100	64%	35%



(a) Page Accesses (b) Time

Figure 6: Effect of λ, FACTUAL



(a) Page Accesses (b) Time

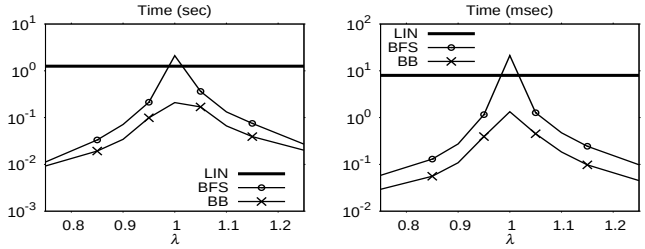
Figure 7: Effect of λ, SYNTH

NN search. On the other hand, small λ values assign more weight to repulsion resembling AFN search. Values around 1 mean that attraction and repulsion are equally strong, making it harder to identify the best object.

For the challenging case of $\lambda = 1$, BFS does not perform better than LIN. The reason is that it cannot guide the search towards the result as efficiently as in other cases (due to the inherent difficulty of $\lambda = 1$ and non-tight bounds), and consequently its computational overhead (computing bounds and prioritizing sub-trees) outweights the savings. On the other hand, when $\lambda = 1$, BB is almost an order of magnitude faster than LIN, thanks to its pruning criteria.

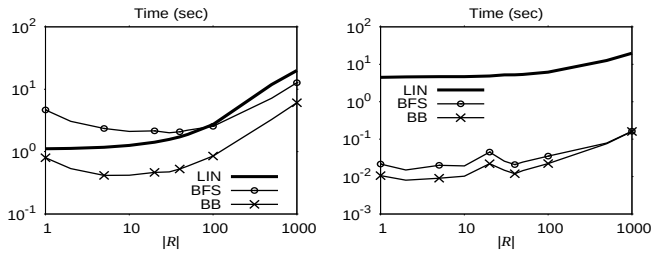
To better illustrate the running time improvements, the important area around $\lambda = 1$ is depicted in Figure 8a, but in this case the x-axis is in linear scale. Figure 8b shows the corresponding results for the SYNTH dataset (using the default parameter values). Our BB algorithm is more than 6 times faster than LIN, even when $\lambda = 1$, with the benefits quickly increasing for smaller or larger values of λ. BFS also provides significant improvements over LIN, but not around the value $\lambda = 1$. The superiority of BB is attributed to the additional pruning that can be achieved.

In order to assess the effectiveness of the individual pruning criteria in BB, Table 4 details the percentage of nodes pruned per criterion with respect to the number of nodes encountered during cohesion query processing. Empty cells indicate that the corresponding criterion does not apply to the specific setting. Please note that both criteria might prune the same node.



(a) FACTUAL (b) SYNTH

Figure 8: Effect of λ in range $[0.75, 1.25]$



(a) FACTUAL, λ=1 (b) FACTUAL, λ=0.75

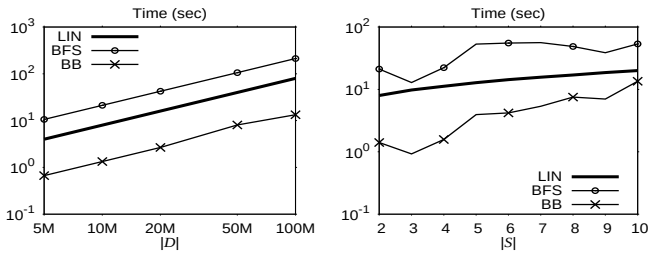
Figure 9: Effect of $|\mathcal{R}|$

Effect of $|\mathcal{R}|$. We now study the effect of the number of repellers $|\mathcal{R}|$, while fixing the weight at $\lambda = 1$. Recall that this is the worst case scenario for our three algorithms. In Figure 9a we present results on the FACTUAL dataset. As the number of repellers increases, so does the running time of all methods, as more distance computations need to be performed. While more repellers offer more chances for pruning, based on Theorem 2, what happens when $|\mathcal{R}|$ increases significantly is that each repeller ends up being close to other repellers, thus limiting the pruning power of new repellers (recall that we insert repellers incrementally). However, some small benefits are observed for BB, since a 100-fold increase in $|\mathcal{R}|$ results in a lower than 100-fold in its running time.

The same graph, but for $\lambda = 0.75$ is depicted in Figure 9b. As previously, when the value of λ deviates significantly from 1, both our algorithms provide running time improvements around 2 orders of magnitude over LIN. For large values of $|R|$, our algorithms have comparable performance. Table 5 shows the criteria pruning power for various $|\mathcal{R}|$ values.

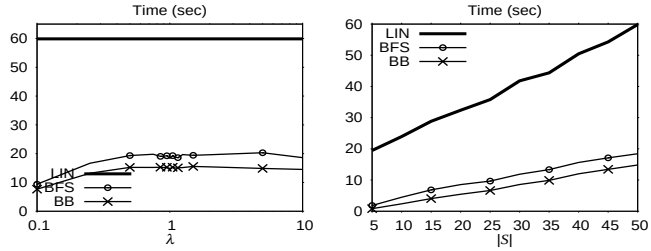
Effect of $|\mathcal{D}|$. In this experiment, we measure the efficiency of queries using the synthetic SYNTH dataset, while varying the cardinality $|\mathcal{D}|$ from 5M to 100M. As always, the remaining parameters obtain their default values. Figure 10a depicts the total processing time as a function of $|\mathcal{D}|$. All methods scale linearly with the dataset cardinality. Our BB algorithm actually scales slightly better than the other approaches, since its relative benefits slightly increase with the increase of $|\mathcal{D}|$. While not depicting the results, we note that for values of λ deviating significantly from 1, our BFS and BB algorithms showed the same trends as in previously experiments, significantly outperforming LIN.

Effect of $|\mathbb{S}|$. We next study the effect of dimensionality in processing cohesion queries, using the SYNTH dataset, and vary the number of attributes $|\mathbb{S}|$ from 2 up to 10. Figure 10b depicts the total processing time as a function of $|\mathbb{S}|$.



(a) Impact of $|\mathcal{D}|$, SYNTH (b) Impact of $|\mathbb{S}|$, SYNTH

Figure 10: Effect of $|\mathcal{D}|$ and $|\mathbb{S}|$, SYNTH



(a) Effect of λ (b) Effect of $|\mathbb{S}|$

Figure 11: MIRFLICKR, d_1

The efficiency of all methods decreases as the dimensionality increases. For BFS, the impact is smaller, as the algorithm has poor performance in all cases of $\lambda = 1$. The effect of $|\mathbb{S}|$ is more pronounced for BB due to the performance degradation of the underlying index (see, e.g., the study in [3]). Still, for small or medium dimensionalities, BB remains up to 2 times faster than LIN. We note that, while not depicting it due to space constraints, BB and BFS maintain significant benefits over LIN for values of λ that deviate significantly from 1, similarly to previous experiments.

Effect of distance metrics. In this experiment, we investigate the performance of our framework using the real dataset MIRFLICKR. As already mentioned, we use the histogram intersection distance. The used distance metrics make more sense for the specific data sets.

Figure 11a shows the effect of the weight λ on cohesion queries over the NBA dataset. In the chosen metrics, the value of λ does not have a significant impact on the running time of our BB and BFS algorithms. Both our algorithms are faster than LIN, typically by a factor of 3 (for BFS) and 4 (for BB). Figure 11b demonstrates the scalability of the tested algorithms when we vary the dimensionality ($|\mathbb{S}|$) from 5 to 50. The improvements of BB and BFS over LIN are important in all cases. For $|\mathbb{S}|$=50, BFS (resp. BB) is over 3 (resp. 4) times faster than LIN.

9. CONCLUSIONS

This work introduced the cohesion query, which given an attractor and a set of repellers, returns the object that is closer to the attractor and at the same time farther than the repellers. For this problem, best-first search and branch and bound algorithms were designed. The challenging case of equal weight between the attraction and repulsion forces is particularly studied and an optimized pruning criterion was proposed. All methods have shown to be up to orders of magnitude more efficient than a linear scan and existing methods based on NN search.

10. REFERENCES

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

[2] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The R*-tree: An efficient and robust access method for points and rectangles. In *SIGMOD*, 1990.

[3] S. Berchtold, D. A. Keim, and H.-P. Kriegel. The X-tree: An index structure for high-dimensional data. In *VLDB*, 1996.

[4] C. Böhm, S. Berchtold, and D. A. Keim. Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases. *ACM Comput. Surv.*, 33(3), 2001.

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

[6] M. Drosou and E. Pitoura. Disc diversity: result diversification based on dissimilarity and coverage. *PVLDB*, 6(1), 2012.

[7] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. In *PODS*, 2001.

[8] R. Fletcher. *Practical Methods of Optimization; (2Nd Ed.)*. Wiley-Interscience, New York, NY, USA, 1987.

[9] P. Fraternali, D. Martinenghi, and M. Tagliasacchi. Top-k bounded diversification. In *SIGMOD*, 2012.

[10] Y. Gao, L. Shou, K. Chen, and G. Chen. Aggregate farthest-neighbor queries over spatial data. In *DASFAA*, 2011.

[11] S. Gollapudi and A. Sharma. An axiomatic approach for result diversification. In *WWW*, 2009.

[12] J. A. Hartigan. *Clustering Algorithms*. Wiley series in probability and mathematical statistics: Applied probability and statistics. John Wiley & Sons Inc, 1975.

[13] G. R. Hjaltason and H. Samet. Distance browsing in spatial databases. *TODS*, 24(2), 1999.

[14] H. V. Jagadish, B. C. Ooi, K.-L. Tan, C. Yu, and R. Zhang. idistance: An adaptive b^+-tree based indexing method for nearest neighbor search. *TODS*, 30(2), 2005.

[15] A. Jain, P. Sarda, and J. R. Haritsa. Providing diversity in k-nearest neighbor query results. In *PAKDD*, 2004.

[16] F. Korn and S. Muthukrishnan. Influence sets based on reverse nearest neighbor queries. In *SIGMOD*, 2000.

[17] K. Mouratidis, D. Papadias, and S. Papadimitriou. Tree-based partition querying: a methodology for computing medoids in large spatial datasets. *VLDBJ*, 17(4):923–945, 2008.

[18] R. T. Ng and J. Han. Efficient and effective clustering methods for spatial data mining. In *VLDB*, 1994.

[19] D. Papadias, Y. Tao, K. Mouratidis, and C. K. Hui. Aggregate nearest neighbor queries in spatial databases. *TODS*, 30(2), 2005.

[20] L. Qin, J. X. Yu, and L. Chang. Diversifying top-k results. *Proc. of the VLDB*, 5(11), 2012.

[21] N. Roussopoulos, S. Kelley, and F. Vincent. Nearest neighbor queries. In *SIGMOD*, pages 71–79, 1995.

[22] I. Stanoi, D. Agrawal, and A. El Abbadi. Reverse nearest neighbor queries for dynamic databases. In *SIGMOD Workshop*, 2000.

[23] Y. Tao, K. Yi, C. Sheng, and P. Kalnis. Quality and efficiency in high dimensional nearest neighbor search. In *SIGMOD*, 2009.

[24] M. J. van Kreveld, I. Reinbacher, A. Arampatzis, and R. van Zwol. Multi-dimensional scattered ranking methods for geographic information retrieval. *GeoInformatica*, 9(1), 2005.

[25] C. S. Won, D. K. Park, and S.-J. Park. Efficient use of mpeg-7 edge histogram descriptor. *Etri Journal*, 24(1), 2002.