Regionally Influential Users in Location-Aware Social Networks

Panagiotis Bouros
Department of Computer Science
Humboldt-Universität zu Berlin
Berlin, Germany
bourospa@informatik.hu-berlin.de

Dimitris Sacharidis
Institute for the Mgmt. of Information Systems
"Athena" Research Center
Athens, Greece
dsachar@imis.athena-innovation.gr

Nikos Bikakis
School of Electrical and Computer Engineering
NTU of Athens
Athens, Greece
bikakis@dblab.ntua.gr

ABSTRACT

The ubiquity of mobile location aware devices and the proliferation of social networks have given rise to Location-Aware Social Networks (LASN), where users form social connections and make geo-referenced posts. The goal of this paper is to identify users that can influence a large number of important other users, within a given spatial region. Returning a ranked list of regionally influential LASN users is useful in viral marketing and in other per-region analytical scenarios. We show that under a general influence propagation model, the problem is #P-hard, while it becomes solvable in polynomial time in a more restricted model. Under the more restrictive model, we then show that the problem can be translated to computing a variant of the so-called closeness centrality of users in the social network, and devise an evaluation method.

Categories and Subject Descriptors

H.2 [**Database Management**]: Database Applications

General Terms

Algorithms

Keywords

location-aware services, social networks, propagation model, influence maximization, graph closeness centrality

1. INTRODUCTION

The proliferation of mobile location-aware devices (e.g., smartphones, tablets, GPS devices) and the current trend for services based upon the social interactions of their users have given rise to the so-called *Location-aware Social Networks* (LASN). In the most predominant LASNs, such as Foursquare or Twitter (geo-tagged tweets), a user can become friend with another, forming thus a social network, and more importantly can *check-in* at various places, i.e., share in public (or to her friends/followers) her current location and activity, for example eating at a restaurant, attending a concert.

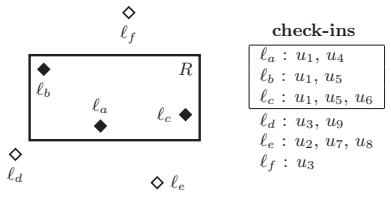
In LASNs it is often useful to find users that are highly influential within a specific geographical region. Consider for example the organizer of a city-wide festival looking to attract people across city districts. The organizer could utilize an LASN to determine the most influential user within each district, and then, recruit her to locally advertise the festival. Contrary to recruiting a group of *globally* influential users, targeting regionally influential users has increased chances to draw attendance from *all* districts. As another example, consider a natural disaster, where affected people often turn to LASNs as a source of prompt information as well as a means for self-organizing community-driven help and support. The government seeking to reward the most active and helpful civilians in the aftermath, would locate the most influential LASN users within the affected area. In such scenarios, the common theme is ranking LASN users according to their computed geo-social influence.

In this work, we introduce the *top-k Regionally Influential LASN users* (k-RIL) problem. A user *checks-in* at a location ℓ when she makes a geo-tagged post from ℓ. Thus, given a spatial region R, we say that a user is *regional* if she has checked-in at least once at a location within R. Moreover, the *locality* of a regional user u models the probability that u will check-in at a location within R, and, in a sense, generalizes for regions the concept of "mayorship" in Foursquare. Assuming an information propagation model for social networks [7], the *regional influence* of a user is defined as the (expected) total locality of users she influences. Under these, k-RIL returns the k regional users with the highest regional influence.

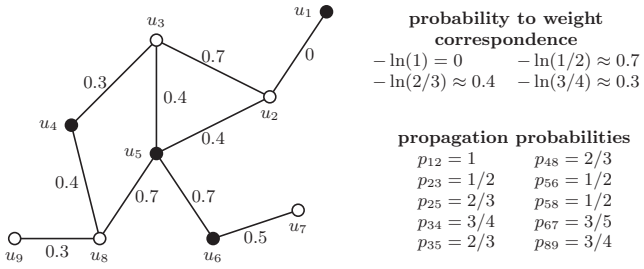
The k-RIL problem is related to the problems of *Influence Maximization* (IM) in social networks, and *Graph Closeness Centrality* (GCC). In IM, given an information propagation model, e.g., the *Independent Cascade* (IC) described in [7], the goal is to select a group of users, termed seeds, that *collectively* influence the largest number of other users. The most computationally challenging (#P-hard) task in IM problems is computing the probability of a user being influenced. Note that even though the k-RIL problem definition is similar to the case of a single seed in IM, the #P-hardness still holds. Therefore, a common strategy in IM problems is to simplify the underlying model. In this spirit, the *Maximum Influence Arborescence* (MIA) model [9] restricts IC with respect to the following two assumptions: (i) a user may influence another only through third users which lie on the path that maximizes the aggregate propagation probability, and (ii) only such paths with aggregate propagation probability above a pre-defined threshold are

SIGSPATIAL'14, November 04 - 07 2014, Dallas/Fort Worth, TX, USA

ACM 978-1-4503-3131-9/14/11 ...$15.00
http://dx.doi.org/10.1145/2666310.2666489



(a) Locations and check-ins



(b) Social network and propagation probabilities

Figure 1: Running example of 9 users and 6 locations

considered. In our work, we adopt a similar strategy introducing a restricted version of the IC model, termed MIAwoT, which is however significantly less restrictive than MIA. Finally, a very recent work [8] solves an IM problem in a similar context to ours, However, this work targets the fundamentally different problem of selecting a group of k users that *collectively* maximize influence within a region R, whereas k-RIL seeks to *rank* users. In addition, it makes some unrealistic assumption, e.g., each LASN user has a known fixed location, and the proposed solution relies on extensive pre-computations, which makes it unsuitable for k-RIL.

Under the MIAwoT model, we show that it is possible to compute the regional influence of a user deterministically, by carefully assigning weights to edges of the social graph and computing network distances between users. As a result, k-RIL becomes similar to the *Graph Closeness Centrality* (GCC) problem [1], i.e., find the node that minimizes the sum of distances to all other nodes. However, the state-of-the-art method for GCC [6] optimizes only the case of directed graphs and thus cannot be applied in our setting. Therefore, we present a preliminary solution to k-RIL termed DRIC, that calculates all pairwise network distances (after computing the appropriate weights) and, then determines the regional influence of users. Our experiments show that DRIC can be efficient when both the size of the social network and the number of regional users are small, as it essentially computes the influence of all regional users.

2. PROBLEM DEFINITION

Before formally introducing the k-RIL problem, we present some necessary definitions.

Location-Aware Social Network (LASN). Let U denote the set of users, L the set of locations, and C the set of check-ins, where a check-in (u, ℓ) means that user u has checked-in at location ℓ; $C(u)$ denotes the set of locations user u has checked-in. The social graph $G(U, E)$ contains an (undirected) edge $(u_i, u_j) \in E$ indicating that u_i and u_j are friends.

Propagation Model. Each edge $(u_i, u_j) \in E$ is associated with a propagation probability $p_{ij} > 0$, which quantifies the degree of influence between the two users. This value is calculated directly from the users' check-ins, e.g., using the Jaccard similarity:

$$p_{ij} = \frac{|C(u_i) \cap C(u_j)|}{|C(u_i) \cup C(u_j)|},$$

or it can also be determined by external parameters, such as the users' profiles and their friendship duration.

The propagation model we adopt differs from the maximum influence arborescence (MIA) model in that it does not enforce an influence threshold (i.e., we do not require the second assumption discussed in Section 1). This model, which we refer to as MIAwoT, is a restricted version of the Independent Cascade (IC) model but is significantly less restrictive than MIA.

The concept of maximum influence paths is principle in MIAwoT. Let π_{st} denote a simple path $u_{\pi[1]} \cdots u_{\pi[m]}$ on the social graph from user $u_{\pi[1]} \equiv u_s$ to $u_{\pi[m]} \equiv u_t$, and define the path propagation probability of π_{st} as

$$p(\pi_{st}) = \prod_{i=1}^{m-1} p_{\pi[i]\pi[i+1]} \tag{1}$$

A path from u_s to u_t is called the *maximum influence path* (mip), and denoted by π^*_{st}, if it has the highest path propagation probability among all other paths from u_s to u_t. As there can be multiple paths that maximize the path propagation probability, the maximum influence path is selected as one of them subject to the restriction that all its subpaths are also mips (as in MIA [9]).

Users in MIAwoT, as in IC, can be in two possible states, influenced and not influenced; once a node becomes influenced it remains so. Propagation on G under MIAwoT proceeds as follows. Let S_t represent the set of users influenced at step t. At step $t = 0$, the influenced users S_0 are also called the seeds. Then at step $t+1$, each user u_i that was influenced at step t, i.e., $u_i \in S_t \setminus S_{t-1}$, may influenced her neighbor u_j with probability p_{ij}, *only if* edge (u_i, u_j) lies on some maximum influence path starting from a seed. This last clause is what differentiates MIAwoT from IC. Note that each user is given only one chance to influence her neighbors, at the step right after she became influenced. Propagation ends at the step when no new user is influenced. We denote the set of eventually *influenced users* as $\Phi(S_0)$; when S_0 is a single user u we simply denote it as $\Phi(u)$.

Problem Statement. Given a spatial region R, we define the set of *regional users* $U_R \subseteq U$ as the users who have checked-in at least once at a location inside R, i.e, $U_R = \{u | u \in U, \exists \ell \in C(u) : \ell \in R\}$. Moreover, for an LASN user u we define the ratio $\gamma_R(u)$ of u's local check-ins in R over the total as the *locality* of the user:

$$\gamma_R(u) = \frac{|C(u) \text{ inside } R|}{|C(u)|} \tag{2}$$

Intuitively, the locality captures the prior probability of a user checking in at some location inside a given region R.

Given a region R, the *regional influence* of a user u is defined as the expected sum of localities of users influenced by u (expectance $\mathbb{E}$ is on the random set $\Phi(u)$ over the propagation probabilities):

$$I_R(u) = \mathbb{E}\left(\sum_{u' \in \Phi(u)} \gamma_R(u') \right). \tag{3}$$

Note that non-regional users have localities equal to zero, and thus they do not contribute to the regional influence. This means that equivalently, the summation in Equation 3 could be restricted over users in $\Phi(u) \cap U_R$.

We next state the *top-k Regionally Influential LASN users* (k-RIL) problem.

Problem k-RIL. Given a spatial region R, return a set U_R^k of k regional users that have the highest regional influence, i.e., $|U_R^k| = k$ and $\forall u \in U_R^k, \forall u' \in U_R \setminus U_R^k$ it holds that $I_R(u) \geq I_R(u')$.

Example 1. Figure 1 presents an example LASN of 9 users and 6 locations. Consider an 1-RIL instance with a spatial region R. Figure 1a depicts the locations as diamonds, draws the region R, and also shows the check-ins grouped by location. The locations inside R are drawn with filled diamonds, and their corresponding check-ins are enclosed in a box. From the check-in lists, we derive that u_1, u_4, u_5, u_6 are the regional users. Without loss of generality, we simplify formulas setting $\gamma_R = 1$ for all regional users. Figure 1b depicts the social graph, where regional users are shown as filled circles. The bottom right part of the figure contains the propagation probabilities p_{ij} between users u_i and u_j. For the sake of the example, we assume that these probabilities are given and thus do not correspond to Jaccard similarities computed from the check-ins. ■

3. METHODOLOGY

First, in Section 3.1, we present an efficient method for computing the regional influence. Next, in Section 3.2 we outline an algorithm for solving k-RIL.

3.1 Computing the Regional Influence

We first prove the #P-hardness of k-RIL under the general Independent Cascade (IC) propagation model.

Theorem 1. The k-RIL problem under the IC propagation model is #P-hard.

Proof. The problem of computing the influence spread of a single user, which we call SIS, under the IC model is shown to be #P-hard in [9]. We reduce SIS to the k-RIL problem. Given an SIS instance, we create a k-RIL instance where the social graph is identical, the region R is equal to the entire space, k is equal to $|U|$, and the locality γ_R is equal to 1 for all users. Then, it is easy to determine the answer to the SIS instance by solving the k-RIL instance. Essentially, in order to rank the users, you need to compute the regional influence of all users, which in turn means that you have solved the SIS instance, as the influence spread of a user in SIS equals its regional influence in the particular k-RIL instance. □

This result justifies our adoption of a model more restricted than IC, namely the MIAwoT propagation model. Under MIAwoT, it is possible to solve k-RIL in polynomial time. To reach this conclusion, we first show that the regional influence of a user can be computed exactly using a closed form deterministic formula.

Lemma 1. The regional influence of a user u_s under MIAwoT is:

$$I_R(u_s) = \sum_{u_i \in U_R} p(\pi_{si}^*) \cdot \gamma_R(u_i)$$

where π_{si}^* is the maximum influence path from u_s to u_i.

Proof. For any user u_i, let $X(u_i)$ be an indicator random variable such that $X(u_i) = 1$ when $u_i \in \Phi(u_s)$, and $X(u_i) = 0$ otherwise. Then, Equation 3 can be rewritten as:

$$I_R(u_s) = \mathbb{E}\left(\sum_{u_i \in \Phi(u_s) \cap U_R} \gamma_R(u_i)\right) = \mathbb{E}\left(\sum_{u_i \in U_R} X(u_i) \cdot \gamma_R(u_i)\right)$$
$$= \sum_{u_i \in U_R} \mathbb{E}\left(X(u_i)\right) \cdot \gamma_R(u_i)$$

Under MIAwoT, a user u_i can be influenced by u_s only via the maximum influence path π_{si}^* from u_s to u_i. This means that u_i is influenced with probability equal to this path's propagation probability $p(\pi_{si})$. Therefore, $\mathbb{E}(X(u_i)) = 1 \cdot p(\pi_{si}^*) + 0 \cdot (1 - p(\pi_{si}^*)) = p(\pi_{si}^*)$, and the theorem follows. □

Lemma 1 shows that the regional influence of a user u_s can be directly computed from the path propagation probabilities of the maximum influence paths from u_s to any other regional user. Therefore, the challenge is how to efficiently compute the propagation probabilities. Towards this goal, inspired by [9], we define a set W of *edge weights* for the social graph such that weight

$$w_{ij} = -\ln p_{ij} \tag{4}$$

is assigned to edge (u_i, u_j). Moreover, let $d(u_s, u_t)$ denote the *social distance*, i.e., the sum of weights of the shortest path on G from user u_s to u_t. We emphasize that the social distance of two users is not related to the spatial locations of their check-ins and is only based on their proximity on the social graph.

Example 2. Returning to our running example of Figure 1, we note that the top right part of the figure shows the value of the edge weights for all propagation probability values, as computed using Equation 4. The numbers along the graph edges correspond to the weights. For illustration purposes and easy distance computations the weight values are rounded to one decimal place; we remark that this rounding does not affect the correctness of the 1-RIL result in all evaluation methods. ■

The following lemma shows an alternative way for computing path propagation probabilities using the edge weights.

Lemma 2. The path propagation probability of the maximum influence path from u_s to u_i can be computed from the social distance of u_s and u_i as:

$$p(\pi_{si}^*) = e^{-d(u_s, u_i)}$$

Proof. Let $\pi_{si}^* = u_{\pi^*[1]} \cdots u_{\pi^*[m]}$ denote the maximum influence path from u_s to u_i. Since $p(\pi_{si}^*) = \prod_{k=1}^{m-1} p_{\pi^*[k]\pi^*[k+1]} = \exp(\ln(\prod_{k=1}^{m-1} p_{\pi^*[k]\pi^*[k+1]})) = \exp(-\sum_{k=1}^{m-1} w_{\pi^*[k]\pi^*[k+1]}) = \exp(-d(u_s, u_i))$, the lemma follows. □

Combining the results of Lemmas 1 and 2, we obtain the following formula for the regional influence of a user u_s.

$$I_R(u_s) = \sum_{u_i \in U_R} e^{-d(u_s, u_i)} \cdot \gamma_R(u_i) \tag{5}$$

Moreover, it is easy to show the tractability of the k-RIL problem under MIAwoT.

Theorem 2. The k-RIL problem under the MIAwoT propagation model is solvable in polynomial in $|U|$ time.

Proof. Computing Equation 5 for each user $u_s \in U_R$, i.e., at most $|U|$ times, clearly solves k-RIL. Moreover, computing Equation 5 requires finding all shortest path from u_s according to the edge weights on the social graph. This task is accomplished in $O(|E| + |U| \log |U|)$ amortized time using Dijkstra's algorithm, where $|E| = O(|U|^2)$ is the number of edges in the graph. Hence, there exists an algorithm that solves k-RIL in time at most cubic in $|U|$. □

3.2 The Algorithm

The discussion in the previous section implies the following evaluation method to solve k-RIL, called Dijkstra-based Regional Influence Computation and denoted as DRIC. The basic idea is to compute the shortest path between any pair of regional users on the social graph. Then, for each regional user, DRIC computes her regional influence and finally, sorts the regional users according to their influence and returns the k most influential.

Algorithm 1: DRIC

Input: social graph $G(U, E)$; set of weights W; set of locations L; set of check-ins C; spatial region R; value k
Output: top-k list $\mathcal{T}$
Variables: set of regional users U_R, social distance matrix D

1 $U_R \leftarrow$ `GetRegionalUsers`(U, L, C, R);
2 **foreach** $u_i \in U_R$ **do**
3 $\quad D \leftarrow$ `Dijkstra`(u_i, G, W, U_R);
4 $\quad I_R(u_i) \leftarrow$ `ComputeRegionalInfluence`(u_i, U_R, D);
5 $\quad$ **push** u_i to $\mathcal{T}$;
6 **return** $\mathcal{T}$;

	u_1	u_4	u_5	u_6
u_1	0	1	0.4	1.1
u_4	1	0	0.7	1.4
u_5	0.4	0.7	0	0.7
u_6	1.1	1.4	0.7	0

(a) Distance matrix D

$I_R(u_1) = 1 + 3/8 + 2/3 + 1/3 = 2.375$
$I_R(u_4) = 3/8 + 1 + 1/2 + 1/4 = 2.125$
$I_R(u_5) = 2/3 + 1/2 + 1 + 1/2 = 2.666$
$I_R(u_6) = 1/3 + 1/4 + 1/2 + 1 = 2.083$

(b) Regional influence

Figure 2: DRIC computations

Algorithm 1 illustrates the pseudocode of the method. DRIC receives as inputs an LASN, i.e., a social graph $G(U, E)$ with a set of weights W, a set of spatial locations L and a set of check-ins C, and a k-RIL query, i.e., a spatial region R and an integer k. It returns the list $\mathcal{T}$ of the top-k most influential regional users. DRIC utilizes two data structures: (i) the set of regional users U_R and (ii), the social distance matrix D which is a $|U_R| \times |U_R|$ symmetric matrix that stores inside every cell $D[u_i][u_j]$ the length of the shortest path on the social graph $G(U, E)$ between regional users u_i and u_j, i.e., $D[u_i][u_j] = d(u_i, u_j)$.

In the beginning, DRIC invokes the `GetRegionalUsers` function to define the set of regional users U_R (Line 1). For every user u_i of U_R the total number of her check-ins inside R is also calculated to determine her locality $\gamma_R(u_i)$. The implementation details of `GetRegionalUsers` are outside the scope of this paper; any index for spatial range queries, e.g., the R-tree [5], can be employed. Then, in Lines 2–5, the algorithm examines every regional user u_i in U_R to calculate her regional influence $I_R(u_i)$ calling functions `Dijkstra` and `ComputeRegionalInfluence`, and inserts u_i into list $\mathcal{T}$. `Dijkstra` computes the shortest path from u_i to all regional users in U_R and stores its length inside the social distance matrix D, while `ComputeRegionalInfluence` computes $I_R(u_i)$ using Equation 5 and matrix D.

Example 3. In the 1-RIL example of Figure 1b, there exist 4 regional users, u_1, u_4, u_5, u_6. DRIC calls `Dijkstra` once for each regional user to compute entries of the social distance matrix D, one row at a time. The resulting matrix is shown in Figure 2a. After each `Dijkstra` invocation, DRIC computes the regional influence of the examined user from Equation 5. Consider user u_1 for example. Her influence is $I_R(u_1) = e^0 + e^{-1} + e^{-0.4} + e^{-1.1} = 1 + 3/8 + 2/3 + 1/3 = 2.375$. Finally, after all regional influences are computed, depicted in Figure 2b, DRIC returns u_5, having the highest regional influence, as the answer to 1-RIL. ■

Complexity. The DRIC algorithm performs exactly U_R iterations. Each iteration invokes Dijkstra's algorithm, which performs $|E|$ edge relaxations and $|U|$ deheap operations. Assumming a Fibonacci heap, each of these operations require $O(1)$, and $O(\log |U|)$ amortized time. Note that an iteration also computes the regional influence, which however takes $O(|U|)$ time and is thus dominated by Dijkstra's running time. Therefore, the total (amortized) running time of DRIC is $O(|U_R||E| + |U_R||U| \log |U|)$.

4. EXPERIMENTS AND CONCLUSIONS

We finally present a preliminary experimental evaluation of our methodology for identifying the top-k regionally influential users.

Table 1: Datasets characteristics

Characteristic	Datasets			
	Gowalla [2]	Brightkite [2]	Foursquare1 [3]	Foursquare2 [4]
Users $\|U\|$	197K	58K	18K	11K
Edges $\|E\|$	950K	214K	116K	47K
Locations $\|L\|$	1.3M	773K	43K	187K
Check-ins $\|C\|$	6.4M	4.5M	2M	1.4M

Table 2: Response time (sec) varying query selectivity, $k = 5$

$\|U_R\|/\|U\|$ (%)	Gowalla	Brightkite	Foursquare1	Foursquare2
0.1	140.6	9.6	0.9	0.2
0.2	262.1	17.9	2.3	0.4
0.3	432.5	26.8	3.2	0.6
0.5	590.6	42.1	5.8	0.9
1	1148.6	71.5	11.2	1.9

Our analysis involves 4 datasets from real-world LASNs. Table 1 summarizes the characteristics of these datasets. The evaluation is carried out on an 2.67Ghz Intel Xeon CPU E5640 with 32GB of RAM running Debian Linux and DRIC was written in C++.

To assess the performance of DRIC algorithm, we measure its average response time over 500 k-RIL queries, varying query selectivity $|U_R|/|U|$, i.e., the number of regional users over the total number of LASN users. Note that we choose to directly vary the selectivity of a query instead of the size of its spatial region R as the most time consuming step of the method (the Dijkstra algorithm) is related to the number of users checked-in at a location inside R and not to how large this region is.

Table 2 reports the response time of DRIC while varying query selectivity $|U_R|/|U|$. The results verify the complexity analysis of Section 3.2. The response time increases linearly to the number of regional users $|U_R|$. Naturally, DRIC slows down with the increase of the size of the social graph. Note that the performance of DRIC is not affected by the number of returned users, so k is set to 5.

DRIC can efficiently solve k-RIL in case of small size social networks or small number of regional users. Motivated by this, in the future we plan to devise more efficient methods that will avoid computing the influence for all regional users by examining them in descending order of their expected influence.

Acknowledgements This research was partially supported by the German Research Foundation (DFG) through the Research Training Group METRIK, grant no. GRK 1324, and by the European Union (European Social Fund – ESF) and Greek national funds through the Operational Program "Education and Lifelong Learning" of the National Strategic Reference Framework (NSRF) – Research Funding Program: Thales. Investing in knowledge society through the European Social Fund.

5. REFERENCES

[1] A. Bavelas. Communication patterns in task-oriented groups. *Journal of the acoustical society of America*, 1950.
[2] E. Cho, S. A. Myers, and J. Leskovec. Friendship and mobility: user movement in location-based social networks. In *KDD*, 2011.
[3] H. Gao, J. Tang, and H. Liu. Exploring social-historical ties on location-based social networks. In *ICWSM*, 2012.
[4] H. Gao, J. Tang, and H. Liu. gscorr: modeling geo-social correlations for new check-ins on location-based social networks. In *CIKM*, 2012.
[5] A. Guttman. R-trees: A dynamic index structure for spatial searching. In *SIGMOD*, 1984.
[6] P. W. O. Jr., A. G. Labouseur, and J.-H. Hwang. Efficient top-k closeness centrality search. In *ICDE*, 2014.
[7] D. Kempe, J. M. Kleinberg, and É. Tardos. Maximizing the spread of influence through a social network. In *KDD*, 2003.
[8] G. Li, S. Chen, J. Feng, K. lee Tan, and W.-S. Li. Efficient location-aware influence maximization. In *SIGMOD*, 2014.
[9] C. Wang, W. Chen, and Y. Wang. Scalable influence maximization for independent cascade model in large-scale social networks. *Data Min. Knowl. Discov.*, 25(3):545–576, 2012.

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.

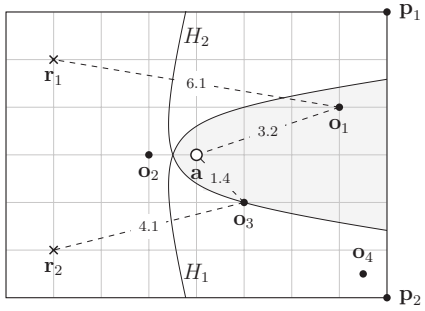


Figure 1: Cohesion query with one attractor a and two repellers $\mathbf{r}_1$, $\mathbf{r}_2$; object $\mathbf{o}_1$ has the largest cohesion of around 2.9

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 $d^+(N,\mathbf{r}) - \lambda \cdot d^-(N,\mathbf{a}) < \tau$.

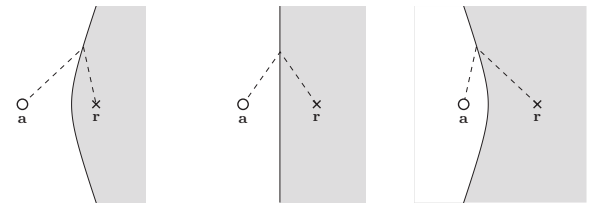
Proof. From the definitions of $d^+(N,\mathbf{a})$ and $d^-(N,\mathbf{r})$, we derive that $\forall \mathbf{a} \in \mathcal{A}$, $\exists \mathbf{r} \in \mathcal{R}$ such that $d^r(\mathbf{o},\mathbf{r}) - \lambda \cdot d^a(\mathbf{o},\mathbf{a}) \leq d^+(N,\mathbf{r}) - \lambda \cdot d^-(N,\mathbf{a}) < \tau$ 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 $d(\mathbf{x}, \mathbf{r}) - d(\mathbf{x}, \mathbf{a}) < \tau$

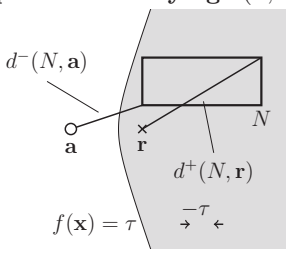


Figure 3: Criterion 2 for $\lambda = 1$ and $\tau < 0$

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.

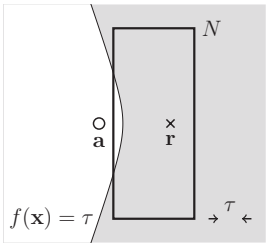


Figure 4: Non-convex pruning area for $\lambda = 1$, $\tau > 0$

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.

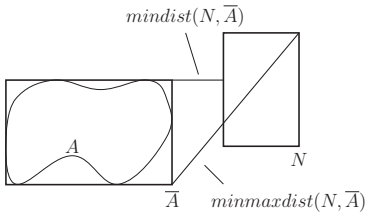


Figure 5: Distance bounds of node N to area A

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

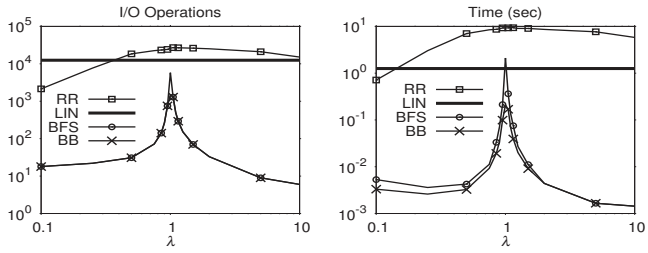
Effect of λ. We first study the effect of the weight λ as it varies from 0.1 up to 10 at the FACTUAL and SYNTH datasets. The remaining parameters obtain their default values, depicted in Table 3.

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

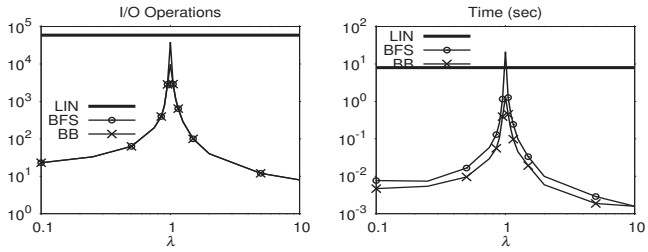
Table 5: Pruning Power ($\lambda = 1$)

$\|\mathcal{R}\|$	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

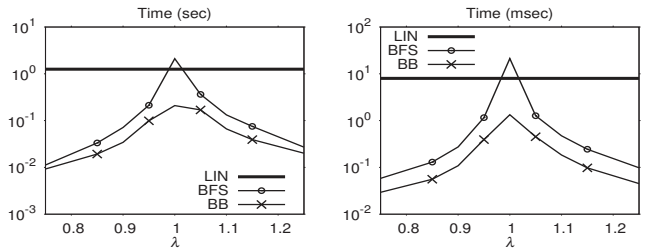
Figure 7: Effect of λ, SYNTH

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.

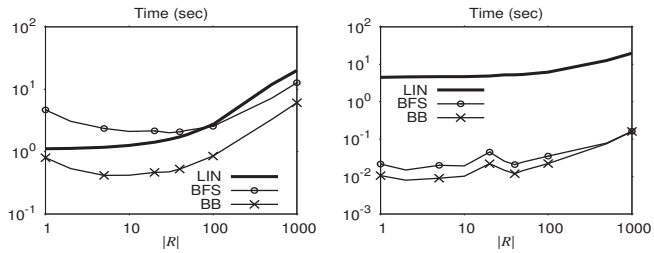
To better illustrate the running time improvements, the important area around $\lambda = 1$ is depicted in Figure 8a, but in this case the x-axis is in linear scale. Figure 8b shows the corresponding results for the SYNTH dataset (using the default parameter values). Our BB algorithm is more than 6 times 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

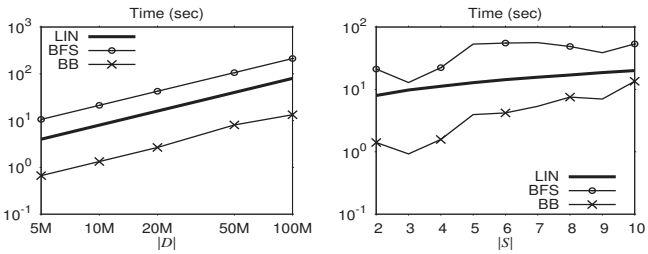
Figure 9: Effect of $|\mathcal{R}|$

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.

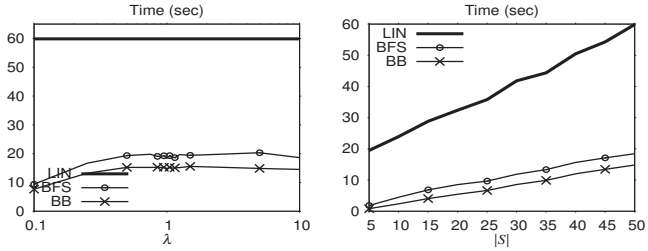
Effect of $|\mathcal{D}|$. In this experiment, we measure the efficiency of queries using the synthetic SYNTH dataset, while varying the cardinality $|\mathcal{D}|$ from 5M to 100M. As always, the remaining parameters obtain their default values. Figure 10a depicts the total processing time as a function 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.

Acknowledgements. This work was supported by the European Social Fund and Greek National Funds through the NSRF Research Program Thales

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.

A Study on External Memory Scan-based Skyline Algorithms

Nikos Bikakis[1,2], Dimitris Sacharidis[2], and Timos Sellis[3]

[1] National Technical University of Athens, Greece
[2] IMIS, "Athena" R.C., Greece
[3] RMIT University, Australia

Abstract. Skyline queries return the set of non-dominated tuples, where a tuple is dominated if there exists another with better values on all attributes. In the past few years the problem has been studied extensively, and a great number of external memory algorithms have been proposed. We thoroughly study the most important scan-based methods, which perform a number of passes over the database in order to extract the skyline. Although these algorithms are specifically designed to operate in external memory, there are many implementation details which are neglected, as well as several design choices resulting in different flavors for these basic methods. We perform an extensive experimental evaluation using real and synthetic data. We conclude that specific design choices can have a significant impact on performance. We also demonstrate that, contrary to common belief, simpler skyline algorithm can be much faster than methods based on pre-processing.

Keywords: Experimental evaluation, experimental survey, disk-based algorithm.

1 Introduction

The *skyline query*, or skyline operator as it was introduced in [3], has in the past few years received great attention in the data management community. Given a database of objects, the skyline query returns those objects which are not dominated. An object *dominates* another, if it has better values on all attributes, and strictly better value on at least one. Finding the skyline is also known as the Pareto-optimal set, or maximal vectors problem in multi-objective optimization research, where it has been studied extensively in the past, but only for in-memory computations. For example the well-known divide and conquer algorithm of [7] has complexity $O(N \log^{d-2} N)$, for $d \geq 2$, where N is the number of objects, and d their dimensionality; the algorithm is optimal for $d = 3$.

The interest in external memory algorithms has sparked after the seminal work in [3]. The most efficient method in terms of worst-case Input/Output (I/O) operations is the algorithm in [15], which requires in the worst case $O\left((N/B)\log^{d-2}_{M/B}(N/B)\right)$ I/Os, where M is the memory size and B the block (minimum unit of transfer in an I/O operation) size in terms of objects. However,

in practice, other external-memory algorithms proposed over the past years can be faster.

This work studies in detail an important class of practical algorithms, the *scan-based skyline algorithms*. An algorithm of this class performs multiple passes over an input file, where the input file in the first pass is the database, and in a subsequent pass it is the output of the previous pass. The algorithm terminates when the output file remains empty after a pass concludes. Generally speaking, during each pass, the algorithm maintains in main memory a small *window* of incomparable objects, which it uses to remove dominated objects from the input file. Any object not dominated is written to the output file.

Although the studied algorithms are specifically designed to operate in external memory, little attention has been given to important implementation details regarding memory management. For example, all algorithms assume that the unit of transfer during an I/O operation is the object, whereas in a real system is the block, i.e., a set of objects. Our work addresses such shortcomings by introducing a more realistic I/O model that better captures performance in a real system. Furthermore, by thoroughly studying the core computational challenge in these algorithms, which is the management of the objects within the window, we introduce several novel potentially interesting policies.

Summarizing, the contributions of our study are the following:

- Based on a standard external memory model [1], we appropriately adapt four popular scan-based algorithms, addressing in detail neglected implementation details regarding memory management.
- We focus on the core processing of scan-based algorithms, the management of objects maintained in the in-memory window. In particular, we introduce various policies for two tasks: traversing the window and evicting objects from the window. Both tasks can have significant consequences in the number of required I/Os and in the CPU time.
- We experimentally evaluate concrete disk-based implementations, rather than simulations, of all studied algorithms and derive useful conclusions for synthetic and real datasets. In particular, we demonstrate that, in many cases and contrary to common belief, algorithms that pre-process (typically, sort) the database are not faster.
- We perform an extensive study of our proposed policies, and reach the conclusion that in some settings (dimensionality and dataset distribution) these policies can reduce the number of dominance checks by more than 50%.

2 Preliminaries

2.1 Definitions

Let $\mathcal{O}$ be a set of *d-dimensional objects*. Each object $o \in \mathcal{O}$ is represented by its *attributes* $o = (o^1, o^2, \ldots, o^d)$. The *domain* of each attribute, is the positive real numbers set $\mathbb{R}^+$. Without loss of generality, we assume that an object o_1 is *better* than another object o_2 on an attribute j, iff $o_1^j < o_2^j$. An object o_1

dominates another object o_2, denoted by $o_1 \succ o_2$, iff (1) $\forall i \in [1, d]$, $o_1^i \leqslant o_2^i$ and (2) $\exists j \in [1, d]$, $o_1^j < o_2^j$. The *skyline* of an object set $\mathcal{O}$, denoted as $SL(\mathcal{O})$, is the set of objects in $\mathcal{O}$ that are not dominated by any other object of $\mathcal{O}$. Formally, $SL(\mathcal{O}) = \{o_i \in \mathcal{O} \mid \nexists o_k \in \mathcal{O} : o_k \succ o_i\}$.

2.2 External Memory I/O Model

This section describes an external memory model, similar to that of [1]. The unit of transfer between the main memory and the external memory (i.e., the disk) is a single *block*.[4] Any external memory algorithm, like the skyline methods, read/write blocks from/to disk *files*. We assume that files are stored contiguously on disk, and therefore a new block is written always at the end of a file.

We denote as $N = |\mathcal{O}|$ the size of the database, i.e., N is the total number of objects to be processed. We measure the fixed size B of a block in terms of objects (tuples). Similarly, main memory can fit M objects, with the requirements that $M < N$ (and often much smaller) to justify the need for external memory algorithms, and $M > 2B$ to support basic in-memory operations.

We next discuss Input/Output (I/O) operations. We assume no input or output buffers, so that blocks from the disk are transferred directly to (resp. from) the disk from (resp. to) the main memory. Equivalently, the input/output buffers share the same memory of size M with the algorithm.

We categorize I/O operations in two ways. Naturally, a *read* transfers data from the disk, whereas a *write* transfers data to the disk. The second categorization is based on the number of blocks that are transferred. Note that a read (resp. write) operation transfers at least one block and at most $\lfloor \frac{M}{B} \rfloor$ blocks into main memory (resp. disk). We also remark that in disks, the *seek time*, i.e., the time it takes for the head to reach the exact position on the ever spinning disk where data is to be read or written, is a crucial parameter in disk performance. Reading or writing k consecutive blocks on the disk is much faster than reading or writing k blocks in arbitrary positions on the disk. The reason is that only one seek is required in the first case, compared to the k seeks for the second. Therefore, we distinguish between *sequential* and *random* I/Os. A random I/O incorporates the seek time, whereas a sequential I/O does not. For example, when a procedure reads k blocks sequentially from the disk, we say that it incurs 1 random read and $k - 1$ sequential reads.

3 A Model for Scan-based Skyline Algorithms

3.1 Design Choices

All skyline algorithms maintain a set of objects, termed *window*, which consists of possible skyline objects, actual skyline objects, or some arbitrary objects in general. A common procedure found in all algorithms is the following. Given some candidate object not in the window, *traverse* the window and determine if

[4] [1] assumes that P blocks can be transferred concurrently; in this work we set $P = 1$

the candidate object is dominated by a window object, and, if not, additionally determine the window objects that it dominates. Upon completion of the traversal and if the candidate is not dominated, the skyline algorithm may choose to insert it into the window, possible *evicting* some window objects.

In the aforementioned general procedure, we identify and focus on two distinct design choices. The first is the *traversal policy* that determines the order in which window objects are considered and thus dominance checks are made. This design choice directly affects the number of dominance checks performed and thus the running time of the algorithm. An ideal (but unrealistic) traversal policy would require only one dominance check in the case that the candidate is dominated, i.e., visit only a dominating window object, and/or visit only those window objects which the candidate dominates.

The second design choice is the *eviction policy* that determines which window object(s) to remove so as to make room for the candidate object. This choice essentially determines the dominance power of the window, and can thus indirectly influence both the number of future dominance checks and the number of future I/O operations.

We define four window traversal policies. The *sequential traversal policy* (sqT), where window objects are traversed *sequentially*, i.e., in the order they are stored. This policy is the one adopted by all existing algorithms. The *random traversal policy* (rdT), where window objects are traversed in *random* order. This policy is used to gauge the effect of others. The *entropy-based traversal policy* (enT), where window objects are traversed in ascending order of their *entropy* (i.e., $\sum_{i=1}^{d} \ln(o^i + 1)$) values. Intuitively, an object with a low entropy value has greater dominance potential as it dominates a large volume of the space.

In addition to these traversal policies, we define various ranking schemes for objects, which will be discussed later. These schemes attempt to capture the dominance potential of an object, with higher ranks suggesting greater potential. Particularly, we consider the following traversal policies. The *ranked-based traversal policy* (rkT), where window objects are traversed in descending order based on their *rank* values. Moreover, we consider three hybrid random-, rank-based traversal policies. The *highest-random traversal policy* ($hgRdT$), where the k objects with the highest rank are traversed first, in descending order of their rank; then, the random traversal policy is adopted. The *lowest-random traversal policy* ($lwRdT$), where the k objects with the lowest rank are compared first, before continuing with a random traversal. Finally, the *recent-random traversal policy* ($rcRdT$), where the k most recently read objects are compared first, before continuing with a random traversal.

Moreover, we define three eviction policies. The *append eviction policy* (apE), where the *last* inserted object is removed. This is the policy adopted by the majority of existing algorithms. The *entropy-based eviction policy* (enE), where the object with the *highest entropy* value is removed. Finally, the *ranked-based eviction policy* (rkE), where the object with the *lowest rank* value is removed. In case of ties in entropy or rank values, the most recent object is evicted.

We next discuss ranking schemes used in the ranked-based traversal and eviction policies. Each window object is assigned a rank value, initially set to zero. Intuitively, the rank serves to identify "promising" objects with high dominance power, i.e., objects that dominate a great number of other objects. Then, the skyline algorithm can exploit this information in order to reduce the required dominance checks by starting the window traversal from promising objects, and/or evict non-promising objects.

We define three ranking schemes. *r0R*: the rank of an object o at a time instance t, is equal to the number of objects that have been dominated by o until t. In other words, this ranking scheme counts the number of objects dominated by o. *r1R*: this ranking is similar to *r0R*. However, it also considers the number of objects that have been dominated by the objects that o dominates. Let $rank(o)$ denote the rank of an object o. Assume that object o_1 dominates o_2, Then, the rank of o_1 after dominating o_2 is equal to $rank(o_1) + rank(o_2) + 1$. *r2R*: this ranking assigns two values for each object o, its *r1R* value, as well as the number of times o is compared with another object and none of them is dominated (i.e., the number of incomparable dominance checks). The *r1R* value is primarily considered to rank window objects, while the number of incomparable check is only considered to solve ties; the more incomparable checks an object has, the lower its rank.

3.2 Algorithm Adaptations for the I/O Model

BNL. The *Block Nested Loop* (BNL) [3] algorithm is one of the first external memory algorithms for skyline computation. All computations in BNL occur during the window traversal. Therefore, BNL uses a window as big as the memory allows. In particular, let W denote the number of objects stored in the window, and let O_b denote the number of objects scheduled for writing to disk (i.e., in the output buffer). The remaining memory of size $I_b = M - W - O_b$ serves as the input buffer, to retrieve objects from the disk. Note that the size of the I/O buffers I_b and O_b vary during the execution of BNL, subject to the restriction that the size of the input buffer is always at least one disk block, i.e, $I_b \geq B$, and that the output buffer never exceeds a disk block, i.e., $O_b \leq B$; we discuss later how BNL enforces this requirements.

We next describe memory management in the BNL algorithm. BNL performs a number of passes, where in each an input file is read. For the first pass, the input file is the database, whereas the input file in subsequent passes is created at the previous pass. BNL terminates when the input file is empty. During a pass, the input file is read in *chunks*, i.e., sets of blocks. In particular, each read operation transfers into main memory exactly $\lfloor \frac{I_b}{B} \rfloor$ blocks from disk, incurring thus 1 random and $\lfloor \frac{I_b}{B} \rfloor - 1$ sequential I/Os. On the other hand, whenever the output buffer fills, i.e., $O_b = B$, a write operation transfers into disk exactly 1 block and incurs 1 random I/O.

We now discuss what happens when a chunk of objects is transfered into the input buffer within the main memory. For each object o in the input buffer,

BNL traverses the window, adopting the sequential traversal policy (sqT). Then, BNL performs a two-way dominance check between o and a window object w. If o is dominated by w, o is discarded and the traversal stops. Otherwise, if o dominates w, object w is simply removed from the window.

At the end of the traversal, if o has not been discarded, it is appended in the window. If W becomes greater than $M - O_b - B$, BNL needs to move an object from the window to the output buffer to make sure that enough space exists for the input buffer. In particular, BNL applies the append eviction policy (apE), and selects the last inserted object, which is o, to move into the output buffer. If after this eviction, the output buffer contains $O_b = B$ objects, its contents are written to the file, which will become the input file of the next pass.

A final issue is how BNL identifies an object o to be a skyline object, BNL must make sure that o is dominance checked with all surviving objects in the input file. When this can be guaranteed, o is removed from the window and returned as a result. This process is implemented through a *timestamp mechanism*; details can be found in [3].

SFS. The *Sort Filter Skyline* (SFS) [4] algorithm is similar to BNL with one significant exception: the database is first sorted by an external sort procedure according to a monotonic scoring function. SFS can use any function defined in Section 3.1.

Similar to BNL, the SFS algorithm employs the sequential window traversal policy (sqT) and the append eviction policy (apE). There exist, however, two differences with respect to BNL. Due to the sorting, dominance checks during window traversal are one-way. That is an object o is only checked for dominance by a window object w. In addition, the skyline identification in SFS is simpler than BNL. At the end of each pass, all window objects are guaranteed to be results and are thus removed and returned.

LESS. The *Linear Elimination Sort for Skyline* (LESS) [5] algorithm improves on the basic idea of SFS, by performing dominance checks during the external sort procedure. Recall that standard external sort performs a number of passes over the input data. The so-called zero pass (or sort pass) brings into main memory M objects, sorts them in-memory and writes them to disk. Then, the k-th (merge) pass of external sort, reads into main memory blocks from up to $\lfloor M/B \rfloor - 1$ files created in the previous pass, merges the objects and writes the result to disk.

LESS changes the external sort procedure in two ways. First, during the zero pass, LESS maintains a window of size W_0 objects as an elimination filter to prune objects during sorting. Thus the remaining memory $M - W_0$ is used for the in-memory sorting. The window is initially populated after reading the first $M - W_0$ objects by selecting those with the lowest entropy scores. Then for each object o read from the disk and before sorting them in-memory, LESS performs a window traversal. In particular, LESS employs the sequential traversal policy (sqT) performing a one-way dominance check, i.e., it only checks if o is dominated. Upon comparing all input objects with the window, the object with

the lowest entropy o_h is identified. Then, another sequential window traversal (sqT) begins, this time checking if o_h dominates the objects in the window. If o_h survives, it is appended in the window, evicting the object with the highest entropy score, i.e., the entropy-based eviction policy (enE) is enforced.

The second change in the external sort procedure is during its last pass, where LESS maintains a window of size W objects. In this pass, as well as any subsequent skyline processing passes, LESS operates exactly like SFS. That is the sequential traversal policy (sqT) is used, one-way dominance checks are made, and window objects are removed according to the append eviction policy (epE).

RAND. In the Randomized multi-pass streaming (RAND) algorithm [13], each pass in RAND consists of three phases, where each scans the input file of the previous pass. Therefore, each pass essentially corresponds to three reads of the input file. In the first phase, the input file is read and a window of maximum size $W = M - B$ is populated with randomly sampled input objects (using reservoir sampling).

In the second phase, the input file is again read one block at a time, while the window of W objects remain in memory. For each input object o, the algorithm traverses the window in sequential order (sqT), performing one-way dominance checks. If a window object w is dominated by o, w is replaced by o. Note that, at the end of this phase, all window objects are skyline objects, and can be returned. However, they are not removed from memory.

In the third phase, for each input object o, RAND performs another sequential traversal of the window (sqT), this time performing an inverse one-way dominance check. If o is dominated by a window object w, or if o and w correspond to the same object, RAND discards o. Otherwise it is written on a file on the disk, serving as the input file for the next pass. At the end of this phase, the memory is cleaned.

4 Related Work

External memory skyline algorithms can be classified into three categories: (1) scan-based, (2) index-based, and (3) partitioning-based algorithms.

The *scan-based* approaches perform multiple passes over the dataset and use a small window of candidate objects, which is used to prune dominated objects. The algorithms of this category can be further classified into two approaches: with and without pre-processing. Algorithms of the first category, directly process the set of objects, in the order in which they are stored, or produced (e.g., in the case of pipelining multiple operators). The BNL [3] and RAND [13] algorithms, detailed in Section 3.1, lie in this category. On the other hand, methods in the second category perform an external sort of the objects before, or parallel to the skyline computation. The SFS [4] and LESS [5], also detailed in Section 3.1, belong to this category. Other algorithm, include Sort and Limit Skyline algorithm (SaLSa) [2], which is similar to SFS and additionally introduces a condition for early terminating the input file scan, and Skyline Operator

on Anti- correlated Distributions (SOAD) [14], which is also similar to SFS but uses different sorting functions for different sets of attributes.

In *index-based* approaches, various types of indices are used to guide the search for skyline points and prune large parts of the space. The most well-known and efficient method is the Branch and Bound Skyline (BBS) [12] algorithm. BBS employs an R-tree, and is shown to be I/O optimal with respect to this index. Similarly, the Nearest Neighbor algorithm (NN) [6] also uses an R-tree performing multiple nearest neighbor searches to identify skyline objects. A bitmap structure is used by Bitmap [16] algorithm to encode the input data. In the Index [16] algorithm, several B-trees are used to index the data, one per dimension. Other methods, e.g., [9,10], employ a space-filling curve, such as the Z-order curve, and use a single-dimensional index. The Lattice Skyline (LS) algorithm [11] builds a specialized data structure for low-cardinality domains.

In the *partitioning-based* approaches, algorithms divide the initial space into several partitions. The first algorithm in this category, D&C [3] computes the skyline objects adopting the divide-and-conquer paradigm. A similar approach with stronger theoretical guarantees is presented in [15]. Recently, partitioning-based skyline algorithms which also consider the notion of incomparability are proposed in [17,8]. OSP [17] attempts to reduce the number of checks between incomparable points by recursively partition the skyline points. BSkyTree [8] enhances [17] by considering both the notions of dominance and incomparability while partitioning the space.

5 Experimental Analysis

5.1 Setting

Datasets. Our experimental evaluation involves both synthetic and real datasets. To construct synthetic datasets, we consider the three standard distribution types broadly used in the skyline literature. In particular, the distributions are: anti-correlated (ANT), correlated (CORR), and independent (IND). The synthetic datasets are created using the generator developed by the authors of [3].

We also perform experiments on three real datasets. *NBA* dataset consists of 17,264 objects, containing statistics of basketball players. For each player we consider 5 statistics (i.e., points, rebound, assist, steal blocks). *House* is 6-dimensional dataset consists of 127,931 objects. Each object, represents the money spent in one year by an American family for six different types of expenditures (e.g., gas, electricity, water, heating, etc.). Finally, *Colour* is a 9-dimensional dataset, which contains 68,040 objects, representing the first three moments of the RGB color distribution of an image.

Implementation. All algorithms, described in Section 3.1, were written in C++, compiled with gcc, and experiments were performed on a 2.6GHz CPU. In order to accurately convey the effect of I/O operations, we disable the operating system caching, and perform direct and synchronous I/O's.

The size of each object is set equal to 100 bytes, as was the case in the experimental evaluation of the works that introduced the algorithms under investigation. Finally, the size of block is set to 2048 bytes; hence each block contains 20 object.

Metrics. To gauge efficiency of all algorithms, we measure: (1) the number of disk I/O operations, which are distinguished into four categories, read, write operations, performed during the pre-processing phase (i.e., sorting) if any, and read, write operations performed during the main computational phase; (2) the number of dominance checks; (3) the time spent solely on CPU processing denoted as CPU Time and measured in seconds; (4) the total execution time, denoted as Total Time and measured in seconds; In all cases the reported time values are the averages of 5 executions.

5.2 Algorithms Comparison

Table 1 lists the parameters and the range of values examined. In each experiment, we vary a single parameter and set the remaining to their default (bold) values. SFS and LESS sort according to the entropy function. During pass zero in LESS, the window is set to one block.

Table 1. Parameters

Description	Parameter	Values
Number of Objects	N	50k, 100K, **500K**, 1M, 5M
Number of Attributes	d	3, **5**, 7, 9, 15
Memory Size	$M/N(\%)$	0.15%, 0.5% **1%**, 5%, 10%

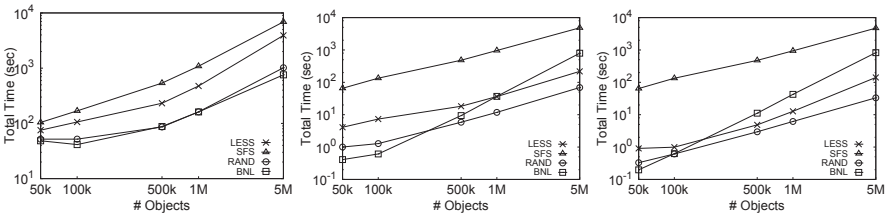
Varying the number of objects. In this experiment, we vary the number of objects from 50K up to 5M and measure the total time, number of I/O's and dominance checks, and CPU time, in Figures 1–4.

The important conclusions from Figure 1 are two. First, RAND and BNL outperform the other methods in anti-correlated datasets. This is explained as follows. Note that the CPU time mainly captures the time spent for the following task: dominance checks, data sorting in case of LESS/SFS, and skyline identification, in case of BNL. From Figure 4 we can conclude that BNL spends a lot of CPU time in skyline identification. BNL requires the same or more CPU time than RAND, while BNL performs fewer dominance checks than RAND. This is more clear in the case of independent and correlated datasets where the cost for dominance checks is lower compared to the anti-correlated dataset. In these datasets, the BNL CPU time increased sharply as the cardinality increases.

The second conclusion is that, in independent and correlated datasets, the performance of BNL quickly degrades as the cardinality increases. This is due to the increase of the window size, which in turn makes window maintenance and skyline identification more difficult.

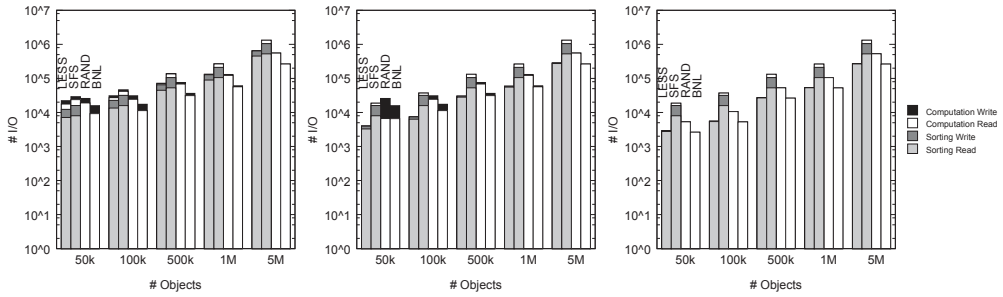
Figure 2 shows the I/O operations performed by the algorithms. We observe that BNL outperforms the other methods in almost all settings. Particularly, in the correlated dataset, LESS is very close to BNL. Also, we can observe that, in general, the percentage of write operations in LESS and SFS is much higher than in BNL and RAND. We should remark that, the write operations are generally more expensive compared to the read operations. Finally, for LESS and SFS, we

can observe that the larger amount of I/O operations are performed during the sorting phase.



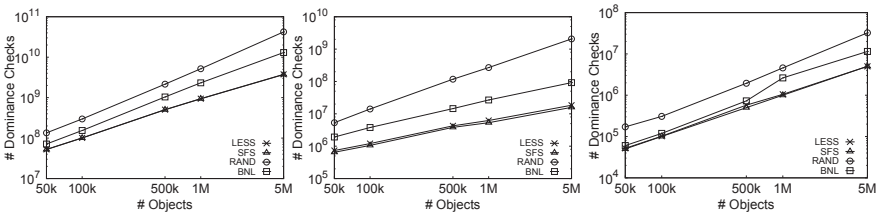
(a) Anti-correlated (b) Independent (c) Correlated

Fig. 1. Total Time: Varying Number of Objects



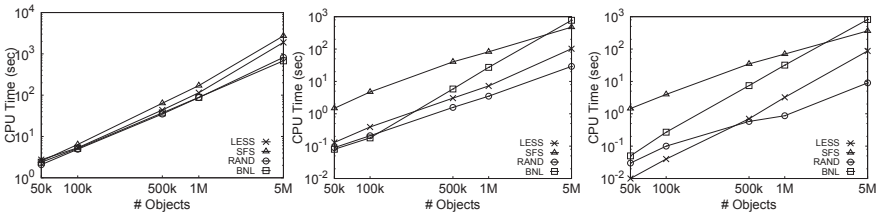
(a) Anti-correlated (b) Independent (c) Correlated

Fig. 2. I/O Operations: Varying Number of Objects



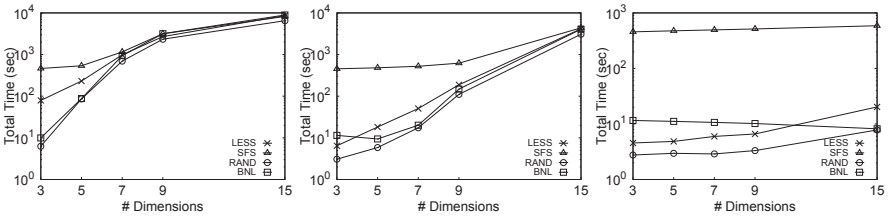
(a) Anti-correlated (b) Independent (c) Correlated

Fig. 3. Dominance Checks: Varying Number of Objects



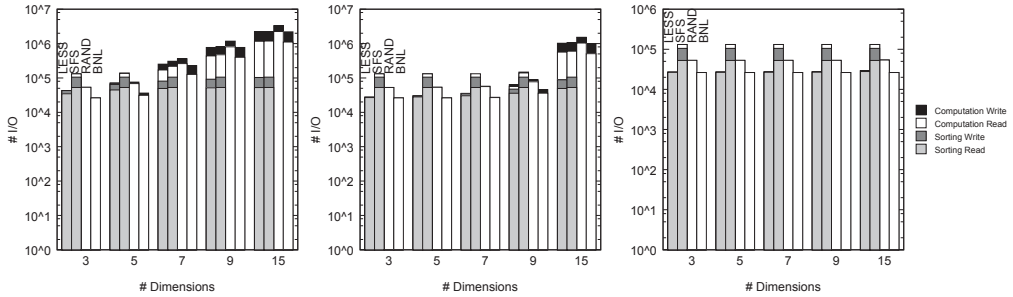
(a) Anti-correlated (b) Independent (c) Correlated

Fig. 4. CPU Time: Varying Number of Objects



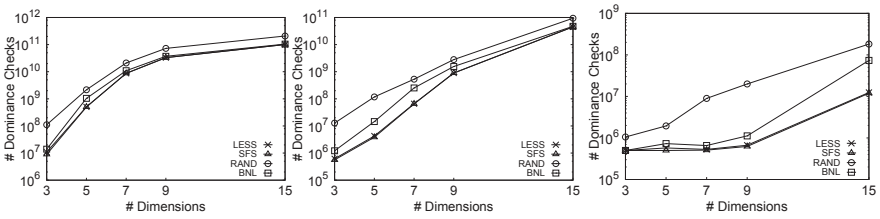
(a) Anti-correlated (b) Independent (c) Correlated

Fig. 5. Total Time: Varying Number of Attributes



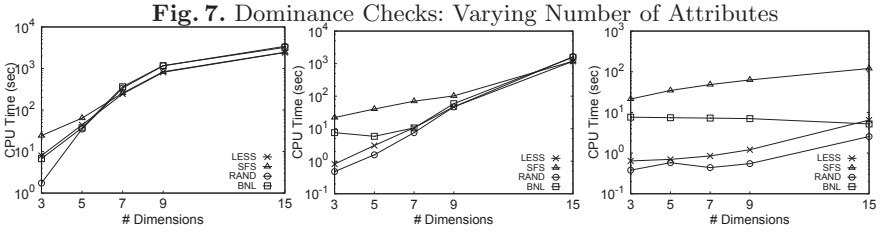
(a) Anti-correlated (b) Independent (c) Correlated

Fig. 6. I/O Operations: Varying Number of Attributes



(a) Anti-correlated (b) Independent (c) Correlated

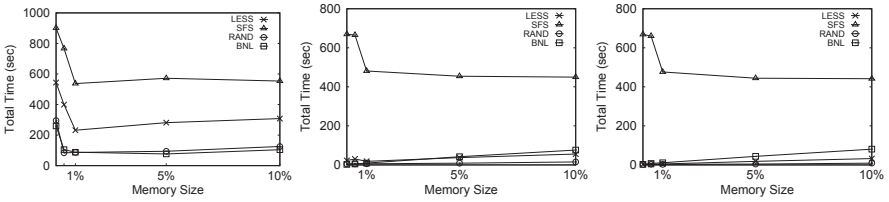
Fig. 7. Dominance Checks: Varying Number of Attributes



(a) Anti-correlated (b) Independent (c) Correlated

Fig. 8. CPU Time: Varying Number of Attributes

Regarding the number of dominance checks, shown in Figure 3, LESS and SFS perform the fewest, while RAND the most, in all cases. Figure 4 shows the CPU time spent by the methods. SFS spends more CPU time than LESS even



(a) Anti-correlated (b) Independent (c) Correlated

Fig. 9. Total Time: Varying Memory Size

though they perform similar number of dominance checks; this is because SFS sorts a larger number of object than LESS. Finally, as previously mentioned, BNL spends considerable CPU time for skyline identification.

Varying the number of dimensions. In this experiment we investigate the performance as we vary the number of dimensions from 3 up to 15. In Figure 5 where the total time is depicted, the performance of all methods become almost the same for anti-correlated and independent datasets, as the dimensionality increases. In the correlated dataset, the skyline can fit in main memory, hence BNL and RAND require only a few passes, while SFS and LESS waste time sorting the data.

Regarding I/O's (Figure 6), BNL outperforms all other methods in all cases, while LESS is the second best method. Similarly, as in Figure 2, LESS and SFS performs noticeable more write operations compared to BNL and RAND. Figure 7 shows that LESS and SFS outperforms the other method, performing the same number of dominance checks. Finally, CPU time is presented in Figure 8, where once again the cost for skyline identification is noticeable for BNL.

Varying the memory size. In Figure 9, we vary the size of the available memory. In general, the total time here, follows the trend of I/O operations. We observe that the required time of all methods decreased sharply for memory sizes up to 1%. However, beyond this point, the time is almost stable as the memory size increases, with the exception of BNL, where the time slightly increases (due to the skyline identification cost w.r.t. window size).

Real Datasets. In this experiment, we evaluate our methods using the real datasets described in Section 5.1. Table 2 summarizes the results, presenting the total time required by all methods. We observe that BNL outperforms the other methods in all datasets in terms of total time. RAND outperforms the other methods in all cases, while SFS is the worst. Note that, in House and Colour datasets, RAND performs more dominance checks, and more I/O operations, than LESS. However, LESS requires more total time, due to larger number of write operations, and the CPU time spend for sorting.

Table 2. Real Datasets: Total Time (sec)

Dataset	LESS	SFS	RAND	BNL
House	30.11	178.21	15.25	4.98
Colour	14.43	90.73	3.70	1.28
NBA	9.45	26.68	0.71	0.41

5.3 Policies Evaluation

In this experiment, we study the effect of different window policies in scan-based skyline algorithms. Particularly, we use BNL and SFS algorithms and we employ several traversal and eviction and policies, in conjunction with different ranking schemes. The effect of policies in LESS are similar to those in SFS and are not shown. Regarding RAND, only the window traversal policy affects its performance; its effect is not dramatic and hence it is also not shown.

All results are presented w.r.t. the original algorithms. That is, let m be a measurement for the original algorithm, and m' be the corresponding measurement for an examined variation. In this case, the measurement presented for the variation is $1 + (m' - m)/m$.

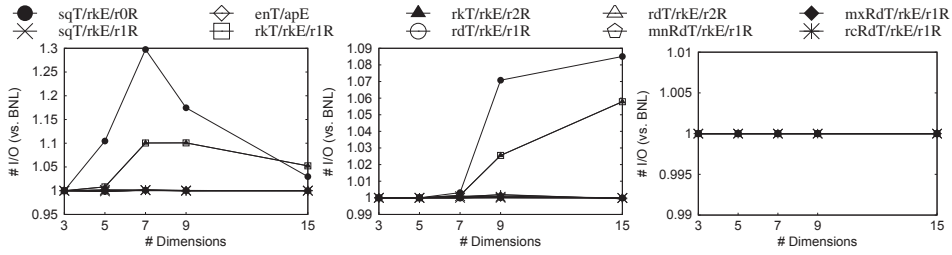
BNL. We first study the performance of BNL under the 10 most important policy and ranking scheme combinations. Figure 10 shows the I/O operations performed by the BNL flavors. As we can see, none of the examined variations performs significant better than the original algorithm. In almost all cases, the I/O performance of most variations is very close to the original. The reason is that the append eviction policy (apE), adopted by the original BNL already performs very well for two reasons. First, the apE policy always removes objects that have not dominated any other object. This way, the policy indirectly implements a dominance-oriented criterion. Second, the apE policy always removes the most recently read object, which is important for BNL. A just read object, requires the most time (compared to other objects in the window) in order to be identified as a skyline, thus propagated to the results and freeing memory. Hence, by keeping "older" objects we increase the probability of freeing memory in the near future. Still it is possible to marginally decrease the number of I/Os.

Figure 11 shows the number of dominance checks performed. We can observe that, in several cases, the variants that adopt rank-based traversal, perform significant fewer dominance checks than the original. Particularly, the rkT/rkE/r1R and rkT/rkE/r2R variants outperform the others in almost all cases, in independent and correlated datasets, by up to 50%. Similar results also hold for low dimensionalities in the anti-correlated dataset. However, this does not hold in more dimensions, due to the explosion of skyline objects in anti-correlated datasets.

SFS. Here, as in the previous experiment, we examine the performance of SFS algorithm adopting several policies. Similar to BNL, none of SFS variants perform noticeable fewer I/O operations (Figure 12). Regarding the dominance checks (Figure 13), in anti-correlated and independent datasets, most of variants have similar performance to the original algorithm. Only for correlated datasets, ranked-based policies exhibit significant performance gains.

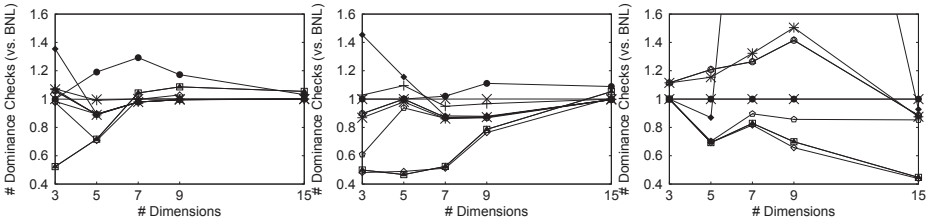
5.4 Discussion

In an I/O-sensitive setting, i.e., when I/O operations cost significantly more than CPU cycles, BNL seems to be the ideal choice, as it performs less I/O operations



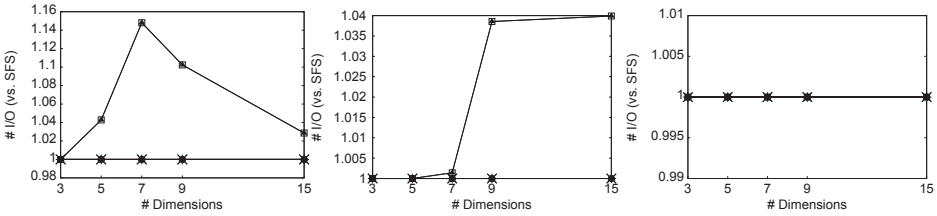
(a) Anti-correlated (b) Independent (c) Correlated

Fig. 10. BNL Policies (I/O Operations): Varying Number of Attributes



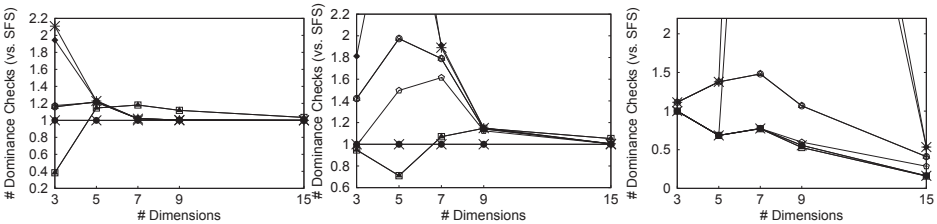
(a) Anti-correlated (b) Independent (c) Correlated

Fig. 11. BNL Policies (Dominance Checks): Varying Number of Attributes



(a) Anti-correlated (b) Independent (c) Correlated

Fig. 12. SFS Policies (I/O Operations): Varying Number of Attributes



(a) Anti-correlated (b) Independent (c) Correlated

Fig. 13. SFS Policies (Dominance Checks): Varying Number of Attributes

than all other methods in almost all settings. Additionally, BNL and RAND perform less write operation than the other methods. On the other hand, in a CPU-sensitive setting, LESS and RAND seem to be good choices. LESS performs the fewest dominance checks, while RAND doesn't spend time for sorting the

data, or for skyline identification. Finally, regarding the policies tested, the rank-based ones show significant gains but only in CPU-sensitive settings.

Acknowledgements

This research has been co-financed by the European Union (European Social Fund – ESF) and Greek national funds through the Operational Program "Education and Lifelong Learning" of the National Strategic Reference Framework (NSRF) – Research Funding Program: Thales. Investing in knowledge society through the European Social Fund.

References

1. Aggarwal, A., Vitter, J.S.: The input/output complexity of sorting and related problems. Commun. ACM 31(9) (1988)
2. Bartolini, I., Ciaccia, P., Patella, M.: Efficient sort-based skyline evaluation. TODS 33(4) (2008)
3. Börzsönyi, S., Kossmann, D., Stocker, K.: The skyline operator. In: ICDE (2001)
4. Chomicki, J., Godfrey, P., Gryz, J., Liang, D.: Skyline with presorting. In: ICDE (2003)
5. Godfrey, P., Shipley, R., Gryz, J.: Algorithms and analyses for maximal vector computation. VLDBJ 16(1) (2007)
6. Kossmann, D., Ramsak, F., Rost, S.: Shooting stars in the sky: An online algorithm for skyline queries. In: VLDB (2002)
7. Kung, H.T., Luccio, F., Preparata, F.P.: On finding the maxima of a set of vectors. Journal of the ACM 22(4) (1975)
8. Lee, J., Hwang, S.w.: Bskytree: scalable skyline computation using a balanced pivot selection. In: EDBT (2010)
9. Lee, K.C.K., Zheng, B., Li, H., Lee, W.C.: Approaching the skyline in z order. In: VLDB (2007)
10. Liu, B., Chan, C.Y.: Zinc: Efficient indexing for skyline computation. VLDB 4(3) (2010)
11. Morse, M.D., Patel, J.M., Jagadish, H.V.: Efficient skyline computation over low-cardinality domains. In: VLDB (2007)
12. Papadias, D., Tao, Y., Fu, G., Seeger, B.: Progressive skyline computation in database systems. TODS 30(1) (2005)
13. Sarma, A.D., Lall, A., Nanongkai, D., Xu, J.: Randomized multi-pass streaming skyline algorithms. VLDB 2(1) (2009)
14. Shang, H., Kitsuregawa, M.: Skyline operator on anti-correlated distributions. vol. 6 (2013)
15. Sheng, C., Tao, Y.: Worst-case i/o-efficient skyline algorithms. TODS 37(4) (2012)
16. Tan, K.L., Eng, P.K., Ooi, B.C.: Efficient progressive skyline computation. In: VLDB (2001)
17. Zhang, S., Mamoulis, N., Cheung, D.W.: Scalable skyline computation using object-based space partitioning. In: SIGMOD (2009)

Maximum Attraction, Minimum Repulsion Queries

Dimitris Sacharidis
IMIS, R.C. Athena
Athens, Greece
dsachar@imis.athena-innovation.gr

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

ABSTRACT

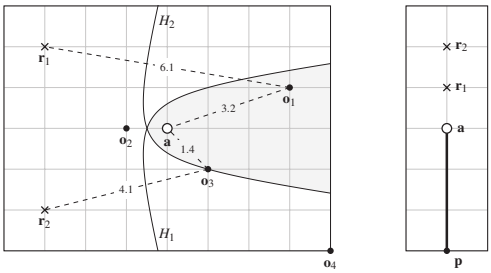
Given an attractor, a set of repellers and a database of objects, all embedded in a metric space, the maximum attraction, minimum repulsion (MAMR) query returns the object that maximizes the weighted difference of its distance to the attractor and its aggregate distance to the repellers, i.e., it is as close to the attractor and as far from the repellers as possible. MAMR queries arise in various optimization problems, such as the query results diversification problem, where the goal is to retrieve a set of objects that are relevant to a given query and at the same time dissimilar to each other. In the basic MAMR query, similar to the nearest neighbor (NN) query, the attractor is the optimal point in space. However, unlike NN queries, the answer to a MAMR query could be very far from the attractor. While MAMR queries require only linear time in the worst case, we show that it is possible to devise a much faster algorithm in practice, by studying the properties of the objective function and introducing pruning criteria. Using a multi-dimensional index structure, drawn from a large variety of supported indexes, we are able to process MAMR queries for a broad class of objective functions and distance metrics, I/O efficiently and orders of magnitude faster than a simple linear scan.

1. INTRODUCTION

This paper introduces the *maximum attraction, minimum repulsion* (MAMR) query. Assume a metric space $\mathbb{S}$, and consider a database of objects $\mathcal{D}$, an attractor $\mathbf{a}$, and a set of repellers $\mathcal{R}$, which are all points embedded in $\mathbb{S}$. The *attraction* of an object $\mathbf{o} \in D$ is the opposite of its distance to $\mathbf{a}$, while the *repulsion* of $\mathbf{o}$ is the opposite of its minimum distance to any repeller in $\mathcal{R}$. The MAMR query returns the object $\mathbf{o}^* \in D$ that maximizes an objective score defined as the weighted difference of its attraction and repulsion. Intuitively, MAMR returns an object that is both close to the attractor and far from the repellers.

The motivation for MAMR queries comes from various optimization problems; we next describe three examples. Consider the top-k diversification problems, where the general goal is to determine a set of k documents that are relevant/similar to a given query and at the same time dissimilar to each other. Since these problems are generally NP-hard, a standard heuristic approach is to build the result set incrementally [13]. Then, the sub-problem at the n-th step can be seen as a MAMR query. The database consists of all documents, the attractor is the query, and the repellers are the set of documents selected at the previous steps.

As another motivational example, consider a competitive facility location [7], where the goal is to determine the optimal location for opening a new department store. In this setting, the database consists of possible building sites, the attractor represents a desirable location, e.g., the city center, and the repellers represents the competitor stores. Then, the MAMR query returns the location that is close to the city center and faces the least competition.



(a) General position (b) Special case

Figure 1: MAMR queries with two repellers

In other optimization problems, the attractor could be implied. For example, assume an NBA team that decides which player to pick so as to better complement its existing roster. Here, the database is the set of eligible players, the set of repellers is the team's current players, and the attractor is the "ideal" player having perfect statistics. Similar problems also appear in other domains, e.g., a company that looks for a new product to launch, where the attractor would have minimum production cost and best specifications, and the repellers would be the competing products.

MAMR queries are related to nearest neighbor (NN) and aggregate farthest neighbor (AFN) queries. The NN query retrieves the object that is closest to an attractor, while the AFN query the one that is farthest from a group of repellers. Contrary, the MAMR query returns the object that strikes the perfect balance between the attraction and the repulsion forces, and as a result, the MAMR answer could significantly differ from the NN or AFN answers.

To illustrate this, consider the example in Figure 1a, which 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 distance (we draw a grid for ease of reference), and seek to maximize the difference between attraction and repulsion. Observe that object $\mathbf{o}_2$ is the NN of the attractor, while $\mathbf{o}_4$ (at the lower right part of the figure) is the min-aggregate FN of $\{r_1, r_2\}$. The answer to the MAMR query is $\mathbf{o}_1$ with score about 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 score about only 2.7.

In basic MAMR queries, where the weights are equal, the attractor is an optimal point that maximizes the objective score. In other words, in the continuous case of MAMR queries (where $\mathcal{D}$ contains all points in the space), the answer is the attractor. This is a desirable property for NN variants, which is used to guide the search

given that the result may lie near the optimal point. For example, in NN queries, the optimal point is the attractor, whereas for AFN queries, it is a vertex in the bounded Voronoi diagram constructed on the set of repellers [10]. However, in MAMR queries the result may not be close to the attractor. In Figure 1, both objects $\mathbf{o}_2$ and $\mathbf{o}_3$ are closer to $\mathbf{a}$ than $\mathbf{o}_1$, and yet have suboptimal scores.

To further complicate things, there exist cases (depending on the distance metric and the position of the attractor and repellers) where no single optimal point (or a finite set) maximizing the objective score exists, but rather a locus. In the example of Figure 1b, the locus is the line segment from the attractor $\mathbf{a}$ towards the edge of the space at point $\mathbf{p}$, drawn with a thick line. Any point on this segment has the highest attainable score 1.

The simplest method for answering a MAMR query is to perform a linear scan on the database $\mathcal{D}$ and compute the score of each object. The computational cost is tolerable, as $|\mathcal{D}| \cdot (|\mathcal{R}|+1)$ distance computations are necessary (a linear cost assuming only a few repellers). A better approach, though, is to process in parallel the corresponding constituent NN, AFN queries and progressively retrieve answers from each until a common object is seen (or a threshold is exceeded), similar to Fagin's algorithms [9].

We propose a different approach for MAMR query processing by studying the properties of the objective function. Returning to the example of Figure 1a, examine object $\mathbf{o}_3$ with score of about 2.7. Considering only repeller $\mathbf{r}_2$, the locus of points that have objective score 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). Hyperbola branch H_1 is similarly defined with respect to repeller $\mathbf{r}_1$. Based on these hyperbolas, it is possible to characterize the space with respect to the score of $\mathbf{o}_3$. In Figure 1a, the shaded area, defined as the intersection of the interior of the two hyperbolas, contains points in space that have score greater than $\mathbf{o}_3$'s, and thus may contain a better object, in our case $\mathbf{o}_1$. So, given an object's score, it is possible to define pruning criteria that eliminate parts of the space containing objects with lower score.

Given an appropriate tree index for the space, we compute bounds for the score of all objects within sub-trees. Using these results, we propose a generic index-based algorithm termed *Branch and Bound MAMR* (BBM) that, in practice, processes MAMR queries significantly faster than a linear scan and a baseline approach.

The BBM algorithm applies for a broad range of distance metrics, including the popular L_p metrics (e.g., Euclidean and Manhattan distances), cosine distance, and histogram intersection. Moreover, BBM works with any index based on minimum bounding rectangles (MBR) or spheres (MBS), including vector space indices, such as the R-tree variants [14, 2] (MBR-based), SS-tree [27] (MBS-based), A-tree [24] (MBR-based), X-tree [3], but also generic metric space indices, such as the M-tree [6]. Other indices, such as the SR-tree [18] combine characteristics from both MBR-based and MBS-based indices and, while they can be supported by our techniques, require some special attention (discussed at the end of Section 5). The contributions of this work are the following:

- We introduce and study the maximum attraction, minimum repulsion query on metric spaces.
- We study the characteristics of the MAMR query and introduce pruning criteria to determine parts of the space with non-promising objects.
- We propose the BBM algorithm for processing MAMR queries for a broad range of distance metrics and a large family of hierarchical indices.
- We perform a detailed experimental study on real and synthetic data, showing that BBM is orders of magnitude faster than a simple linear scan and significantly more efficient than more elaborate techniques based on NN query processing.

Roadmap. Section 2 reviews related work. Section 3 defines all concepts and formally states the MAMR query. Section 4 presents baseline approaches. Section 5 details our methodology for processing MAMR queries. In Section 6 we compute bounds for index nodes, an essential tool for our algorithm. Section 7 presents our experimental study, and Section 8 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., [16], and high-dimensional data, e.g., [26], have been proposed.

For a set of query points, the *aggregate nearest neighbor* (ANN) query [22] 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 [21, 20].

A MAMR 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 MAMR 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. Rather MAMR queries are more 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 [11] proposes an R-tree based algorithm for processing AFN queries for the SUM, MAX, MIN functions. Again, a MAMR 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 MAMR queries. This is the baseline approach described in Section 4. We note that the MAMR 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 [25, 19].

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 [13], which categorizes common diversification objectives and proves NP-hardness.

Note that, in general, diversification problems seek a *group* of

Table 1: Notation

Symbol	Definition
$\mathcal{D}, \mathbf{o}$	database of objects, an object
$\mathcal{A}$	set of attributes
$\mathbf{a}$	the attractor
$\mathcal{R}, \mathbf{r}$	the set of repellers, a repeller
$a(\mathbf{o})$	attraction of object $\mathbf{o}$ by the attractor $\mathbf{a}$
$r(\mathbf{o})$	repulsion of object $\mathbf{o}$ by the set of repellers in $\mathcal{R}$
$s(\mathbf{o})$	objective score of $\mathbf{o}$ given $\mathbf{a}$ and $\mathcal{R}$
ρ	repulsion reach
d_x^a	distance metric x is used for attraction
d_x^r	distance metric x is used for repulsion
τ	a threshold on objective scores

documents that are relevant and diverse. Thus, they are not directly related to MAMR queries. However, sub-problems similar to MAMR 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.

There are other types of diversification, such as coverage based approaches [1, 8], which however are not related to our problem. A more relevant line of work combines diversification and NN queries. [17] 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, [23] 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 MAMR queries.

The most related diversification problem appears in [10]. Their iterative approach solves a sub-problem identical to a restricted version of the MAMR query, when we restrict MAMR to Euclidean space and distance. Using MAMR terminology, the key idea of [10] is to alternate between NN retrievals from the attractor and from certain points computed using the Voronoi diagram of the repellers. We note that [10] is proposed for a more restrictive setting, where only NN modules are available, and thus its application to MAMR queries does not result in a strong competitor.

3. PROBLEM DEFINITION

We now present the necessary definitions and formally introduce the maximum attraction, minimum repulsion query. Table 1 gathers the most important symbols used throughout this paper.

Consider a finite set of objects $\mathcal{D}$, termed the database. An object $\mathbf{o} \in \mathcal{D}$ is defined over a set of numerical attributes $\mathcal{A}$ that form the metric space $\mathbb{S}$. Given an attractor $\mathbf{a} \notin \mathcal{D}$, the *attraction* of an object $\mathbf{o} \in \mathcal{D}$ is defined as:

$$a(\mathbf{o}) = -d^a(\mathbf{o}, \mathbf{a}),$$

where d^a is a distance metric on $\mathbb{S}$. The smaller the distance to $\mathbf{a}$ is, the greater the attraction. The object that maximizes the attraction is the nearest neighbor (NN) of $\mathbf{a}$. Given a set of repellers $\mathcal{R}$, such that $\mathcal{R} \cap \mathcal{D} = \varnothing$, the *repulsion* of object $\mathbf{o} \in \mathcal{D}$ is defined as:

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

where d^r is a distance metric on $\mathbb{S}$. 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 (min-AFN) of $\mathcal{R}$.

Regarding notation, note that the superscripts a, r indicate that the distance metric is used for attraction or repulsion, respectively. Thus, d_{cos}^a, d_2^r signifies that cosine distance is used for attraction and Euclidean distance (L_2) for repulsion. The superscripts a, r are omitted when the distinction between attraction and repulsion is clear, or when the same distance metric is used for both.

Given an attractor $\mathbf{a}$, a set of repellers $\mathcal{R}$ and a sanity bound ρ, termed as the *repulsion reach*, that seeks to limit the effect of distant repellers, we define the objective score of an object $\mathbf{o}$ to be equal to the weighted difference of its attraction and repulsion:

$$\begin{aligned} s(\mathbf{o}) &= \lambda \cdot a(\mathbf{o}) - \max\{-\rho, r(\mathbf{o})\} \\ &= \min\left\{\rho, \min_{\mathbf{r} \in \mathcal{R}} d^r(\mathbf{o}, \mathbf{r})\right\} - \lambda \cdot d^a(\mathbf{o}, \mathbf{a}), \end{aligned} \quad (1)$$

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

To understand the role of the repulsion reach, consider the extreme case depicted in Figure 1b, where no ρ is set. As a result, points $\mathbf{a}$ and $\mathbf{p}$ have the same score, even though the latter is at the edge of the space. The reason is that, as we walk along the thick line away from $\mathbf{a}$, the attraction and repulsion decrease by the same amount. The repulsion reach ρ sets a limit to how small repulsion can become. As a result, walking away from $\mathbf{a}$, we reach a point where repulsion no longer decreases and thus the score (attraction minus repulsion) will start to decrease, so that $\mathbf{p}$ eventually gets a score lower than $\mathbf{a}$. In many applications, a meaningful value for ρ is $\min_{\mathbf{r}_i, \mathbf{r}_j} d^r(\mathbf{r}_i, \mathbf{r}_j)$, which implies that the repulsion reach is equal to the maximum existing repulsion among the repellers. For example, in the competitive facility location example, a new store is assumed to have acceptable competition when it is opened farther to its nearest competitor than the closest distance between existing stores. Of course, in other applications, different ρ values may make more sense; ρ can even be set to ∞ to cancel its effect.

The problem we address in this work is the Maximum Attraction, Minimum Repulsion (MAMR) query: how to efficiently find the object that has the greatest objective score.

Problem 1. [Maximum Attraction, Minimum Repulsion (MAMR) Query] Given an attractor $\mathbf{a}$ and a set of repellers $\mathcal{R}$, find an object $\mathbf{o}^* \in \mathcal{D}$ such that $\mathbf{o}^* = \operatorname{argmax}_{\mathbf{o} \in \mathcal{D}} s(\mathbf{o})$.

4. BASELINE METHODS

A baseline processing technique is to decompose a MAMR query into a nearest neighbor (NN) query on the attractor 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 NN and the AFN search until a termination condition is met. Therefore, we require modules capable of *progressively* processing NN and AFN queries. For NN queries, we use the DF algorithm [15], whereas for AFN queries the algorithm of [11]. Algorithm 1 presents the pseudocode of this algorithm, which we call RR.

The algorithm maintains two threshold values, τ_a, τ_r, which represent the smallest possible distance of an object not yet retrieved by the NN search, and the largest possible aggregate distance of an object not yet seen in the AFN search, initially set to 0 and ∞, respectively (Line 1). Also RR initializes the result $\mathbf{o}^*$ to null and the next search module to NN (Line 1). Then, RR begins a loop with progressive objective retrievals, until the largest attainable score by retrieving additional objects, which is $\min\{\rho, \tau_r\} - \lambda \cdot \tau_a$, drops below the current best score (Line 2). At each iteration, a single object is retrieved (Line 4 or 8), the appropriate threshold is updated

Algorithm 1: RR

Input: index T; attractor $\mathbf{a}$; repellers $\mathcal{R}$; repulsion reach ρ
Output: $\mathbf{o}^*$ the answer to the MAMR query
Variables: objective threshold τ

1 $\mathbf{o}^* \leftarrow \emptyset$; $\tau_a \leftarrow 0$; $\tau_r \leftarrow \infty$; *search* $\leftarrow$ NN
2 **while** $\mathbf{o}^* = \emptyset$ *or* $\min\{\rho, \tau_r\} - \lambda \cdot \tau_a > s(\mathbf{o}^*)$ **do**
3 **if** *search* = NN **then**
4 $\mathbf{o} \leftarrow$ NN.getNext()
5 $\tau_a \leftarrow d^a(\mathbf{o}, \mathbf{a})$
6 *search* $\leftarrow$ AFN
7 **else**
8 $\mathbf{o} \leftarrow$ AFN.getNext()
9 $\tau_r \leftarrow \min_{\mathbf{r} \in \mathcal{R}} d^r(\mathbf{o}, \mathbf{r})$
10 *search* $\leftarrow$ NN
11 **if** $s(\mathbf{o}) > s(\mathbf{o}^*)$ **then**
12 $o^* \leftarrow \mathbf{o}$

(Line 5 or 9), and the other search method is set for the next retrieval (Line 6 or 10). Because the search modules are progressive, the thresholds set are a lower bound on the distance to $\mathbf{a}$, and an upper bound on the aggregate distance to $\mathcal{R}$ of any object not seen in NN and AFN search, respectively. The answer object is updated if an object with better score is retrieved (Lines 11–12).

As another baseline, we adapt the SPP algorithm from [10], which was proposed for a top-k diversification setting. Similar to RR, our SPP implementation employs an NN 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.[1] As we show in our experimental study, 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.

5. METHODOLOGY

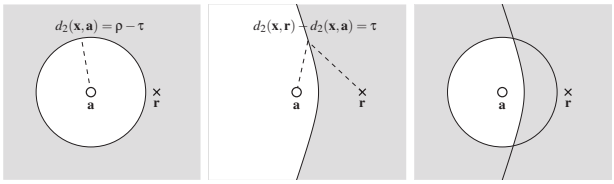
Our techniques for efficiently processing MAMR queries assume the existence of an index over the database of objects $\mathcal{D}$. In Section 5.1, we establish space pruning criteria for eliminating unpromising objects. We then exploit (Section 5.2) the underlying index and show how to apply our criteria to prune entire groups of objects contained within index nodes. In Section 5.3 we derive bounds on the score of all objects contained within an index node. The BBM algorithm for MAMR queries is described in Section 5.4.

5.1 Pruning Criteria

We hereafter seek to simplify notation by introducing the following concepts. For an object $\mathbf{o} \in \mathcal{D}$, we define $\mathbf{r}_{NR}$ to be its nearest repellent amongst set $\mathcal{R}$, i.e., $\mathbf{r}_{NR} = \text{argmin}_{\mathbf{r} \in \mathcal{R}} d^r(\mathbf{o}, \mathbf{r})$; thus Equation 1 simplifies to: $s(\mathbf{o}) = \min\{\rho, d^r(\mathbf{o}, \mathbf{r}_{NR})\} - \lambda \cdot d^a(\mathbf{o}, \mathbf{a})$.

Given an objective threshold τ, the goal of this section is to characterize the metric space, i.e., determine the sub-space containing objects that have score less than τ, in order to define *pruning criteria* for processing a MAMR query. Our algorithm retrieves objects until it finds the one with the maximum objective score. Assume that τ is the score of an object already retrieved (or it is the lower bound on the score of some non-retrieved object). Clearly, the objects that fall in the less than τ sub-space need not be considered.

[1] The authors also propose a more elaborate strategy, but with no significant gains.



(a) Criterion 1 (b) Criterion 2 (c) Pruned space

Figure 2: Pruned space with score less than τ (Euclidean distances, $\lambda = 1$, $\tau > 0$)

Important note. For illustration purposes, all examples hereafter assume that (1) $\mathbb{S} = \mathbb{R}^2$, i.e., the Euclidean plane, (2) attraction and repulsion are defined based on the Euclidean distance d_2, and (3) $\lambda = 1$. We emphasize that the pruning criteria that we introduce apply *without any changes* to other metrics and weight values λ as well. The only thing that changes is the geometric interpretation of these criteria. Moreover, our pruning criteria, which utilize upper and lower bounds on the score of objects lying inside an index node (which in turn depend on the distance bounds computed in Section 6), can be applied in our problem *without* requiring us to draw their geometric interpretation.

Criterion 1. Given an attractor $\mathbf{a}$, a set of repellers $\mathcal{R}$ and an objective threshold τ, any object $\mathbf{o} \in \mathcal{D}$ such that $d^a(\mathbf{o}, \mathbf{a}) > \frac{1}{\lambda}(\rho - \tau)$ has score less than τ.

Proof. The score of an object under the condition of the criterion is: $s(\mathbf{o}) = \min\{\rho, d^r(\mathbf{o}, \mathbf{r}_{NR})\} - \lambda \cdot d^a(\mathbf{o}, \mathbf{a}) < \rho + (\tau - \rho) = \tau$. □

Consider the geometric interpretation of Criterion 1 for the Euclidean case. Any object $\mathbf{o}$ outside the circle with center $\mathbf{a}$ and radius $\frac{1}{\lambda}(\rho - \tau)$ cannot have score more than τ. Figure 2a draws shaded the pruning area for $\lambda = 1, \rho > 0$. Observe that the size of the pruning area increases with the threshold, as higher τ values mean smaller radii (i.e., smaller $\rho - \tau$ values).

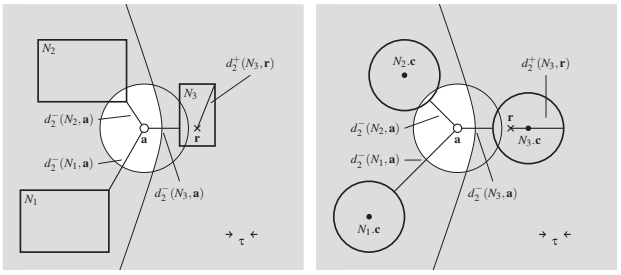
Criterion 2. Given an attractor $\mathbf{a}$, a repeller $\mathbf{r} \in \mathcal{R}$ and an objective threshold τ, any object $\mathbf{o} \in \mathcal{D}$ such that $d^r(\mathbf{o}, \mathbf{r}) - \lambda \cdot d^a(\mathbf{o}, \mathbf{a}) < \tau$ has score less than τ.

Proof. For any object $\mathbf{o} \in \mathcal{D}$ and its nearest repeller $\mathbf{r}_{NR}$, it holds that $d^r(\mathbf{o}, \mathbf{r}) \geq d^r(\mathbf{o}, \mathbf{r}_{NR})$. Since $d^r(\mathbf{o}, \mathbf{r}_{NR}) \geq \min\{\rho, d^r(\mathbf{o}, \mathbf{r}_{NR})\}$, the score of object $\mathbf{o}$ is $s(\mathbf{o}) \leq d^r(\mathbf{o}, \mathbf{r}_{NR}) - \lambda \cdot d^a(\mathbf{o}, \mathbf{a}) \leq d^r(\mathbf{o}, \mathbf{r}) - \lambda \cdot d^a(\mathbf{o}, \mathbf{a})$. Therefore, any object $\mathbf{o}$ satisfying the criterion must have $s(\mathbf{o}) < \tau$. □

Note that Criterion 2 holds, and is in fact stronger, when we substitute the repeller $\mathbf{r}$ with $\mathbf{o}$'s nearest repeller $\mathbf{r}_{NR}$. Consider the geometric interpretation of Criterion 2 for the Euclidean case. Observe that the locus of points $\mathbf{x}$ satisfying equation $d_2(\mathbf{o}, \mathbf{a}) - \lambda \cdot d_2(\mathbf{o}, \mathbf{r}) = \tau$ defines one of the two branches of a hyperbola-like curve with foci the attractor $\mathbf{a}$ and the repeller $\mathbf{r}$. Particularly, we distinguish two cases with respect to τ's value.

(a) When $\tau > 0$, the locus is the branch around the attractor $\mathbf{a}$. Criterion 2 states that any object that lies outside this branch (i.e., the part of space containing focus $\mathbf{r}$) has score less than τ.
(b) When $\tau \leq 0$, the locus is the branch around $\mathbf{r}$. Criterion 2 states that any object that lies inside this branch (i.e., the part of space containing focus $\mathbf{r}$) has score less than τ.

The size of the pruning area increases with τ. A higher τ value causes the locus to move closer to $\mathbf{a}$, and the corresponding branch to become narrower. Figure 2b portrays case (a) of Criterion 2 for $\lambda = 1$ and $\tau > 0$. Any object that lies in the shaded area, i.e., outside the hyperbola branch around focus $\mathbf{a}$, has score less than τ. Figure 2c illustrates the pruning area of both criteria for the depicted attractor $\mathbf{a}$ and repeller $\mathbf{r}$. We emphasize that we can derive



(a) MBR-based nodes (b) MBS-based nodes

Figure 3: Pruning criteria for nodes (Euclidean distances, $\lambda = 1$, $\tau > 0$). The same pruning criteria hold on both subfigures.

bounds on the score of objects grouped at an index node and apply our pruning criteria without their exact geometric interpretation.

5.2 Pruning Criteria for Nodes

We assume a tree structure 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. 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 order to prune entire index nodes during MAMR processing, we need to adapt the pruning criteria to take into account the aggregate information of nodes. This, in turn, requires computing lower ($d^-(N)$) and upper ($d^+(N)$) bounds on the distance of a node N, i.e., bounds on the distance of any object within the node ($d(\mathbf{o})$, $\forall \mathbf{o} \in N$). Section 6 presents bounds for MBR-based and MBS-based nodes for the family of L_p distance metrics, cosine distance and histogram intersection distance. We emphasize that our space pruning framework, presented in Section 5.1, is generic and applies to *any* distance metrics d^a, d^r. Given distance bounds, we apply the criteria for index nodes as follows. First, consider Criterion 1.

Lemma 1. Given an objective threshold τ, a node N contains objects with score less than τ, if $d^{a-}(N,\mathbf{a}) > \frac{1}{\lambda}(\rho - \tau)$.

Proof. Any object $\mathbf{o} \in N$ has $d^a(\mathbf{o},\mathbf{a}) \geq d^{a-}(N,\mathbf{a})$. From the condition of the lemma, we obtain $d^a(\mathbf{o},\mathbf{a}) > \frac{1}{\lambda}(\rho - \tau)$. Thus, Criterion 1 applies for all objects $\mathbf{o} \in N$. □

Figure 3a shows a Euclidean plane example of Lemma 1 for three MBR-based nodes N_1, N_2 and N_3, assuming $\lambda = 1$ and $\tau \geq 0$. Exactly the same example for MBS-based nodes N_1, N_2 and N_3 is presented in Figure 3b. Observe that $d_2^-(N_1,\mathbf{a})$ is greater than $\rho - \tau$, i.e., N_1 lies completely outside the circle with radius $\rho - \tau$ around $\mathbf{a}$, and thus Lemma 1 applies. On the other hand, the lemma does not apply to N_2 or N_3. The next lemma applies Criterion 2 for a node N.

Lemma 2. Given an objective threshold τ, a node N contains objects with score less than τ, if there exists a repeller $\mathbf{r} \in \mathcal{R}$ such that $d^{r+}(N,\mathbf{r}) - \lambda \cdot d^{a-}(N,\mathbf{a}) < \tau$.

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

Note that the object within N that has attraction equal to $d^{a-}(N,\mathbf{a})$ and the object that has repulsion (from $\mathbf{r}$) equal to $d^{r+}(N,\mathbf{r})$ do not coincide in general. In other words, the score of any object within N may be much lower than the value $d^{r+}(N,\mathbf{r}) - \lambda \cdot d^{a-}(N,\mathbf{a})$. As a result, Lemma 2 might not succeed in pruning a node N that only contains objects with scores less than τ, if for that node $d^{r+}(N,\mathbf{r}) - \lambda \cdot d^{a-}(N,\mathbf{a}) \geq \tau$.

Figure 3 shows an example where Lemma 1 does not apply, but Lemma 2 does. Let us examine repeller $\mathbf{r}$ and MBR-based node N_3. Observe that the condition of Lemma 1 does not apply for N_3, but Lemma 2 applies for N_3 and $\mathbf{r}$, since $d_2^+(N_3,\mathbf{r}) - d_2^-(N_3,\mathbf{a})$ is positive, but smaller than τ (the size of τ is depicted at the lower right part of the figure). Thus, Lemma 2 allows us to prune N_3.

5.3 Objective Score Bounds for Nodes

Using the distance bounds on index nodes, computed in Section 6, we can also compute bounds on the score of any object within a tree node. Such bounds are then used to compute an objective threshold and guide the search during MAMR processing.

Lemma 3. Given a non-leaf node N, the score of an object $\mathbf{o} \in \mathcal{D}$ within N cannot be more than $s^+(N) = \min_{\mathbf{r} \in \mathcal{R}}\{\rho, d^{r+}(N,\mathbf{r})\} - \lambda \cdot d^{a-}(N,\mathbf{a})$.

Proof. By the definitions of d^{a-} and d^{r+}, for any object $\mathbf{o} \in N$, we have $d^{a-}(N,\mathbf{a}) \leq d^a(\mathbf{o},\mathbf{a})$ and $d^{r+}(N,\mathbf{r}) \geq d^r(\mathbf{o},\mathbf{r})$. We thus derive $s(\mathbf{o}) = \min_{\mathbf{r} \in \mathcal{R}}\{\rho, d^r(\mathbf{o},\mathbf{r})\} - \lambda \cdot d^a(\mathbf{o},\mathbf{a}) \leq \min_{\mathbf{r} \in \mathcal{R}}\{\rho, d^{r+}(N,\mathbf{r})\} - \lambda \cdot d^{a-}(N,\mathbf{a}) = s^+(N)$ for any object $\mathbf{o} \in N$. □

Lemma 3 implies that the non-leaf node with the highest $s^+()$ value is more likely to contain the answer object, and thus provides the means to direct the search.

Lemma 4. Given a non-leaf node N, the score of an object $\mathbf{o} \in \mathcal{D}$ within N cannot be less than $s^-(N) = \min_{\mathbf{r} \in \mathcal{R}}\{\rho, d^{r-}(N,\mathbf{r})\} - \lambda \cdot d^{a+}(N,\mathbf{a})$.

Proof. By definition, it holds that $d^{a+}(N,\mathbf{a}) \geq d^a(\mathbf{o},\mathbf{a})$ for any object $\mathbf{o} \in N$. Furthermore, it holds that $d^{r-}(N,\mathbf{r}) \leq d^r(\mathbf{o},\mathbf{r})$ for any object $\mathbf{o} \in N$ and any $\mathbf{r} \in \mathcal{R}$. Therefore $s^-(N) \leq s(o)$. □

Lemma 4 provides a lower bound $s^-(N)$ on the score of the answer object for any non-leaf node N. This bound can be used to define an objective threshold and thus apply the pruning criteria.

5.4 Algorithm

Putting the results of the previous sections together, we propose the *Branch and Bound MAMR* (BBM) algorithm for index-based processing of MAMR queries. BBM requires a hierarchical index on the set of objects $\mathcal{D}$. We emphasize that BBM can be used with any MBR-based or MBS-based index for vector spaces as well as any MBS-based index for general metric spaces.

Algorithm 2 shows the pseudocode of the BBM algorithm. It takes as input the index T storing all objects in the database $\mathcal{D}$, the attractor $\mathbf{a}$, the set of repellers $\mathcal{R}$, and the repulsion reach ρ. BBM returns the maximum attraction, minimum repulsion object $\mathbf{o}^*$.

BBM maintains an objective threshold τ, which corresponds to a lower bound of the highest possible score. This value is used to prune the space and avoid visiting non promising nodes. BBM directs the search using the heap H, which contains index nodes and is sorted descending on their upper bound on score, computed according to Lemma 3. BBM initializes τ to $-\infty$, and H to empty (Line 1). The BBM algorithm performs a number of iterations (Lines 3–15). At the end of each iteration the node N_x at the top of the heap is popped (Line 15); for the first iteration N_x is set to the root node of T (Line 2). BBM terminates when node N_x is a non-index node, corresponding thus to an object, which in this case is the answer $\mathbf{o}^*$ (Line 16). Assuming that N_x is an index node, BBM reads this node from disk (Line 4) and checks if Lemmas 1–2 apply for its children (Lines 5–11). Particularly, it first checks if

Algorithm 2: BBM

Input: index T; attractor $\mathbf{a}$; repellers $\mathcal{R}$; repulsion reach ρ
Output: $\mathbf{o}^*$ the answer to the MAMR query
Variables: H a heap with nodes sorted by $s^+()$; objective threshold τ

1 $\tau \leftarrow -\infty$; $H \leftarrow \varnothing$
2 $N_x \leftarrow N_{root}$ ▷ root node of T
3 **while** N_x *is an index node* **do**
4 read node N_x
5 **foreach** *child* N *of* N_x **do**
6 $pruned \leftarrow false$
7 **if** $d^{a-}(N,\mathbf{a}) > \frac{1}{\lambda}(\rho-\tau)$ **then** ▷ Lemma 1
8 $pruned \leftarrow true$; **continue**
9 **foreach** $\mathbf{r} \in \mathcal{R}$ **do**
10 **if** $d^{r+}(N,\mathbf{r}) - \lambda \cdot d^{a-}(N,\mathbf{a}) < \tau$ **then** ▷ Lemma 2
11 $pruned \leftarrow true$; **break**
12 **if** *not pruned* **then**
13 **if** $s^-(N) > \tau$ **then** $\tau \leftarrow s^-(N)$; ▷ Lemma 4
14 push(H,N)
15 $N_x \leftarrow$ pop(H)
16 $\mathbf{o}^* \leftarrow N_x$

Table 2: Bounds provided in this paper

Index Type	L_p	Cosine distance	Histogram intersection distance
MBR-based	Tight upper and lower bounds for any finite value of p as well as $p=\infty$	Non-tight upper and lower bounds provided	Non-tight upper and lower bounds provided
MBS-based Bounding spheres built in L_2	Tight upper and lower bounds for $p=1,2,\infty$. Non-tight bounds also computed for $p \notin \{1,2,\infty\}$	Tight upper and lower bounds provided	Non-tight upper and lower bounds provided
MBS-based Bounding spheres in metric space	Non-tight upper and lower bounds provided for same metric space, as the one in which the bounding spheres are constructed		

Lemma 1 applies for a child N (Lines 7–8). If not, BBM examines all repellers within the set $\mathcal{R}$ (Lines 9–11). For each repeller $\mathbf{r}$, the algorithm checks if Lemma 2 (Lines 10–11) applies for node N. If not, (Lines 12–14), then N is pushed in the heap (Line 14), and the score threshold is appropriately updated (Line 13) using the lower bound on the score of N, according to Lemma 4. The next theorem prove the correctness of BBM.

Theorem 1. The BBM algorithm returns the maximum attraction, minimum repulsion object.

Proof. We show that BBM cannot miss the answer object $\mathbf{o}^*$. BBM prunes index nodes based on Lemmas 1–2 and the threshold computed based on Lemma 4. Therefore, by the correctness of these lemmas, $\mathbf{o}^*$ cannot be in any pruned node.

BBM 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 3, it holds that $s(\mathbf{o}_x) \geq s^+(N)$ for all nodes in H. Therefore $\mathbf{o}_x$ has higher score than any object within any node in the heap and thus any object in $\mathcal{D}$. □

Table 2 provides a synopsis of all the bounds provided in Section 6 for both MBR-based and MBS-based indexes. At this point we need to note that our techniques can also support the SR-tree index, in which an index node specifies a region that is the intersection of a bounding sphere and a bounding rectangle, can be adopted in our techniques, by using as a lower (upper) bound for an index node the maximum (minimum) of the bounds derived for its bounding sphere and rectangle. The corresponding bounds are, though, not tight for the SR-tree.

6. DISTANCE BOUNDS FOR NODES

6.1 Minimum Bounding Rectangles

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

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

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

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 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 $\mathbf{o}[i] \in [N.\ell[i], N.\mathbf{u}[i]]$. 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. □

Consider the *cosine distance metric*: $d_{cos}(\mathbf{o},\mathbf{p}) = 1 - \frac{\langle \mathbf{o},\mathbf{p}\rangle}{\|\mathbf{o}\|\cdot\|\mathbf{p}\|}$, where $\langle \mathbf{x},\mathbf{y}\rangle = \sum_{i=1}^{|\mathcal{A}|}(\mathbf{x}[i]\cdot\mathbf{y}[i])$ is the inner product of $\mathbf{x}$ and $\mathbf{y}$, $\|\mathbf{x}\| = \sqrt{\sum_{i=1}^{|\mathcal{A}|}\mathbf{x}[i]^2}$ is the norm of $\mathbf{x}$, and $\|\mathbf{o}\| \neq 0$ for any object $\mathbf{o}$ in N and also $\|\mathbf{p}\| \neq 0$.

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

$$\mathbf{x}_{cos}^-[i] = \begin{cases} N.\mathbf{u}[i] & \text{,if } \mathbf{p}[i] < 0 \\ N.\ell[i] & \text{,if } \mathbf{p}[i] \geq 0 \end{cases}, \mathbf{x}_{cos}^+[i] = \begin{cases} N.\ell[i] & \text{,if } \mathbf{p}[i] < 0 \\ N.\mathbf{u}[i] & \text{,if } \mathbf{p}[i] \geq 0 \end{cases}.$$

Then, based on these points, the bounds d_2^-, d_2^+ on the L_2 (Euclidean) distance metric, and the zero vector $\mathbf{0}$, i.e., $\mathbf{0}[i]=0$ for all $i \in [1,|\mathcal{A}|]$, define values: $d_{cos}^-(N,\mathbf{p}) = \max\{0, 1 - \frac{\langle \mathbf{x}_{cos}^+,\mathbf{p}\rangle}{d_2^-(N,\mathbf{0})\cdot\|\mathbf{p}\|}\}$ and $d_{cos}^+(N,\mathbf{p}) = 1 - \frac{\langle \mathbf{x}_{cos}^-,\mathbf{p}\rangle}{d_2^+(N,\mathbf{0})\cdot\|\mathbf{p}\|}$ for which the following lemma holds.

Lemma 6. The values $d_{cos}^-(N,\mathbf{p})$, $d_{cos}^+(N,\mathbf{p})$ are a lower and an upper bound, respectively, on $d_{cos}(\mathbf{o},\mathbf{p})$ for any object $\mathbf{o} \in N$ and an arbitrary object $\mathbf{p}$.

Proof. The cosine distance of any object of N and object p is given by: $d_{cos}(\mathbf{o},\mathbf{p}) = 1 - \frac{\langle \mathbf{o},\mathbf{p}\rangle}{\|\mathbf{o}\|\cdot\|\mathbf{p}\|}$. Therefore, it holds that a upper (resp. lower) bound on the numerator and an lower (resp. upper) bound on the denominator result in a lower (resp. upper) bound on $d_{cos}(\mathbf{o},\mathbf{p})$. It thus follows that $d_{cos}(\mathbf{o},\mathbf{p}) \geq d_{cos}^-(N,\mathbf{p})$. Similarly, we derive $d_{cos}(\mathbf{o},\mathbf{p}) \leq d_{cos}^+(N,\mathbf{p})$. □

Finally, we discuss the *histogram intersection distance metric*:

$$d_{hi}(\mathbf{o},\mathbf{p}) = 1 - \frac{\sum_{i=1}^{|\mathcal{A}|}\min\{\mathbf{o}[i],\mathbf{p}[i]\}}{\min\left\{\sum_{i=1}^{|\mathcal{A}|}\mathbf{o}[i], \sum_{i=1}^{|\mathcal{A}|}\mathbf{p}[i]\right\}},$$

where $\mathbf{o}[i] \geq 0$ for any object $\mathbf{o} \in N$ and $\mathbf{p}[i] \geq 0$. Define values:

$$d_{hi}^{-}(\mathbf{o},\mathbf{p}) = \max\{0, 1 - \frac{\sum_{i=1}^{|\mathcal{A}|} \min\{N.\mathbf{u}[i], \mathbf{p}[i]\}}{\min\left\{\sum_{i=1}^{|\mathcal{A}|} N.\ell[i], \sum_{i=1}^{|\mathcal{A}|} \mathbf{p}[i]\right\}}\}$$

$$d_{hi}^{+}(\mathbf{o},\mathbf{p}) = 1 - \frac{\sum_{i=1}^{|\mathcal{A}|} \min\{N.\ell[i], \mathbf{p}[i]\}}{\min\left\{\sum_{i=1}^{|\mathcal{A}|} N.\mathbf{u}[i], \sum_{i=1}^{|\mathcal{A}|} \mathbf{p}[i]\right\}},$$

for which the following lemma holds.

Lemma 7. The values $d_{hi}^{-}(N,\mathbf{p})$, $d_{hi}^{+}(N,\mathbf{p})$ are a lower and an upper bound, respectively, on $d_{hi}(\mathbf{o},\mathbf{p})$ for any object $\mathbf{o} \in N$ and an arbitrary object $\mathbf{p}$.

Proof. The proof follows from the fact that the functions $f(\mathbf{x}) = \sum_{i=1}^{|\mathcal{A}|} \min\{\mathbf{x}[i], \mathbf{p}[i]\}$ and $g(\mathbf{x}) = \min\left\{\sum_{i=1}^{|\mathcal{A}|} \mathbf{x}[i], \sum_{i=1}^{|\mathcal{A}|} \mathbf{p}[i]\right\}$ are non-decreasing monotonous in every dimension i, and the fact that $0 \leq N.\ell[i] \leq \mathbf{o}[i] \leq N.\mathbf{u}[i]$. □

6.2 Minimum Bounding Spheres

An MBS-based node N is associated with a sphere that has center $N.\mathbf{c}$ and radius $N.r$. In the following, we abuse notation and use the symbols $\mathbf{c}$ and r instead for the center and radius of the sphere. The sphere may be constructed either in a general metric space (as in the M-trees [6]), or, more commonly, in L_2, thus creating spheres in the Euclidean plane (as in the SS-tree [27]). In the latter case, the distance metrics used between any point and an attractor or a repeller can be expressed in a different metric distance than L_2.

6.2.1 Bounding Spheres in Metric Spaces

When the bounding sphere is built in a generic metric space, the triangle inequality can be used to derive the desired bounds. In such a case, for a metric m, the values $d_m^{+}(N,\mathbf{p}) = d_m(\mathbf{c},\mathbf{p}) + r$, while $d_m^{-}(N,\mathbf{p}) = d_m(\mathbf{c},\mathbf{p}) - r$. The bounds are not tight. The proof is straightforward and omitted for brevity.

6.2.2 Bounding Spheres Defined in L_2

More often, the bounding sphere is constructed in L_2, rather than a general metric distance. In what follows, we seek to bound $d(\mathbf{o},\mathbf{p})$ for some distance metric d, where $\mathbf{o}$ is an object within N, and $\mathbf{p}$ is some object in the metric space. A straightforward way to achieve this is to (a) Consider the minimum bounding rectangle of the sphere, and (b) Compute the bounds using this minimum bounding rectangle, as described in Section 6.1. For the histogram intersection distance metric, this is the approach that we adopt. However, in many cases we can derive tighter bounds.

Lemma 8. For the L_∞ distance metric, the value $d_\infty^{+}(N,\mathbf{p}) = d_\infty(\mathbf{c},\mathbf{p}) + r$ is a tight upper bound, while a tight lower bound $d_\infty^{-}(N,\mathbf{p})$ can be computed by an efficient iterative algorithm in $O(|\mathcal{A}|^2)$ time.

Proof. Let $j \in \{1,\ldots,|\mathcal{A}|\}$ denote the dimension with the maximum $|\mathbf{p}[j] - \mathbf{c}[j]|$ value. Consider the point $\mathbf{o}$ on the boundary of the sphere N: $\mathbf{o}[i] = \mathbf{p}[i]$, for $i \neq j$ and $\mathbf{o}[j] = \mathbf{p}[j] + r \cdot \text{sign}(\mathbf{p}[j] - \mathbf{c}[j])$, otherwise. Obviously $\mathbf{o}$ is the point inside the sphere which determines the tight upper bound $d_\infty^{+}(N,\mathbf{p}) = r + d_\infty(\mathbf{p},\mathbf{c})$.

For the lower bound d_∞^{-}, it is trivial to see that the point $\mathbf{o}$ with the minimum $d_\infty(\mathbf{o},\mathbf{p})$ value must lie on the boundary of the sphere (otherwise, pushing it towards $\mathbf{p}$ could decrease its distance from $\mathbf{p}$ in the dimensions with the max $|\mathbf{o}[i] - \mathbf{p}[i]|$ values). We thus seek to determine $u_i \geq 0$ values, such that we minimize: $\max_{i=1}^{|\mathcal{A}|}\{|\mathbf{c}[i] - \mathbf{p}[i] - u_i \cdot \text{sign}(\mathbf{c}[i] - \mathbf{p}[i])|\}$, subject to $\sum_{i=1}^{|\mathcal{A}|} u_i^2 = r^2$. Assume that we sort the differences $|\mathbf{c}[i] - \mathbf{p}[i]|$ in descending order and let $d_i, i \in \{1,\ldots,|\mathcal{A}|\}$ denote the sorted list of differences. It is easy to see that what must be accomplished by the optimal algorithm is to start devoting the budget r^2 so as to reduce the maximum d_1 difference, until we reach the second largest d_2 value, then start to equally devoting budget to both d_1 and d_2 until we reach d_3 etc. The process terminates when the available budget is exhausted. Assume that the first $1 \leq k \leq |\mathcal{A}|$ sorted differences are affected when the budget is exhausted and that budget x is devoted at the last step. Thus, $u_i^2 = (x + \sum_{j=i}^{k-1}(d_j - d_{j+1}))^2$ for $i \leq k$ and $u_i = 0$, for $i > k$. Then, $\sum_{i=1}^{k}(x + \sum_{j=i}^{k-1}(d_j - d_{j+1}))^2 = r^2 \Rightarrow \sum_{i=1}^{k}(x + d_i - d_k)^2 = r^2$ yields the solution for x. If $x > d_k - d_{k+1}$, then the algorithm should assign budget $d_k - d_{k+1}$ to the first k dimensions and proceed with dimension $k+1$. The bound is obviously tight. The process requires $O(|\mathcal{A}|^2)$ time to determine the optimal assignment of u_i values. □

Lemma 9. The values $d_p^{-}(N,\mathbf{p}) = \max\{d_p(\mathbf{c},\mathbf{p}) - r|\mathcal{A}|^{\frac{1}{p}-\frac{1}{2}}, 0\}$ and $d_p^{+}(N,\mathbf{p}) = d_p(\mathbf{c},\mathbf{p}) + r|\mathcal{A}|^{\frac{1}{p}-\frac{1}{2}}$ are a lower and an upper bound, respectively, on $d_p(\mathbf{o},\mathbf{p})$ for any object $\mathbf{o} \in N$ and an arbitrary object $\mathbf{p}$ for the L_p distance metric, when p is finite. For the L_1 and L_2 distance metrics the provided upper and lower bounds are tight. The tight lower bound for L_1 is different from the generic formula for d_p^{-}, but can be derived by an analytical formula in $O(|\mathcal{A}|^2)$ time.

Proof. For finite p values, we prove the case for the upper bound, since the case for the lower bound is similar. For any point $\mathbf{o}$ inside N, due to the triangle inequality, $d_p(\mathbf{o},\mathbf{p}) \leq d_p(\mathbf{c},\mathbf{p}) + d_p(\mathbf{c},\mathbf{o})$, subject to $d_2(\mathbf{c},\mathbf{o}) \leq r$. To derive the bounds, we thus seek to determine the maximum possible value of $d_p(\mathbf{c},\mathbf{o})$.

For finite p values, it is trivial to see that the point $\mathbf{o}$ with the maximum $d_p(\mathbf{c},\mathbf{p})$ value must lie on the boundary of the sphere (otherwise, pushing it outwards would increase its distance from $\mathbf{c}$). Maximizing $(\sum_{i=1}^{|A|} |\mathbf{o}[i] - \mathbf{c}[i]|^p)^{\frac{1}{p}}$ subject to $(\sum_{i=1}^{|A|} |\mathbf{o}[i] - \mathbf{c}[i]|^2)^{\frac{1}{2}} = r$ yields (using Lagrange multipliers) that each $|\mathbf{o}[i] - \mathbf{c}[i]|$ be equal to $\frac{r}{\sqrt{|\mathcal{A}|}}$. The upper bound then follows trivially. The case for the lower bound is symmetric.

For the L_2 metric the provided bounds are tight. Consider the vector $\mathbf{d} = \mathbf{c} - \mathbf{p}$ and $\hat{\mathbf{d}} = \frac{\mathbf{d}}{\|\mathbf{d}\|}$. The points $\mathbf{c} - r \cdot \hat{\mathbf{d}}$ and $\mathbf{c} + r \cdot \hat{\mathbf{d}}$ are the ones that determine the d_2^{-} and d_2^{+} values, making the bounds tight. For the L_1 metric, the point $\mathbf{o}$ with coordinates $\mathbf{o}[i] = \mathbf{c}[i] + \frac{r}{\sqrt{|\mathcal{A}|}} \cdot \text{sign}(\mathbf{c}[i] - p[i])$ does have $d_1(\mathbf{o},\mathbf{p}) = d_1(\mathbf{c},\mathbf{p}) + r \cdot |\mathcal{A}|$, making the upper bound tight. The provided lower bound is not tight, but we can reach a tight lower bound with a more intelligent approach.

For the lower bound $d_1^{-}(N,\mathbf{p})$, this is trivially 0 if $\mathbf{p}$ is inside the sphere. Otherwise, we seek to determine a point of the sphere N, such that we minimize $\sum_{i=1}^{|\mathcal{A}|} |\mathbf{p}[i] - \mathbf{c}[i] - u_i \cdot \text{sign}(\mathbf{p}[i] - \mathbf{c}[i])|$, subject to $\sum_{i=1}^{|\mathcal{A}|} u_i^2 \leq r^2$. Again, it is easy to see that the minimum must occur for a point at the boundary of the sphere. Consider all differences $d_i = |\mathbf{p}[i] - \mathbf{c}[i]|$ and sort them in ascending order. Assume for ease of presentation that $d_1 \leq d_2 \leq \cdots \leq d_{|\mathcal{A}|}$. We claim that the optimal assignment of u_i values is $u_i = \min\{d_i, \sqrt{\frac{r^2 - \sum_{j=1}^{i-1} u_j^2}{|A|+1-i}}\}$. Therefore, u_i values are non-decreasing, and once we discover the first k index such that u_k is not capped by d_k, then all u_i values for $k \leq i \leq |\mathcal{A}|$ are equal to $\sqrt{\frac{r^2 - \sum_{j=1}^{k-1} u_j^2}{|A|+1-k}}$. We prove that this is the optimal assignment by contradiction. Assume that there exists a different optimal assignment U^* of u_i^* values with a lower overall d_1^{-} value. Assume that h is the first index where $u_h < u_h^*$.

This obviously cannot happen if $h < k$ (i.e., $u_h == d_h$) – otherwise U^* would have created a non-zero difference for the corresponding index, while wasting budget. Let l denote the first index where $u_l > u_l^*$. If $l < h$, then $u_l \leq u_h \Rightarrow u_l^* < u_h^*$. If $l > h$, then $u_l == u_h \Rightarrow u_l^* < u_h^*$.

For a sufficiently small $\varepsilon > 0$, an alternative assignment U^+ from U^*, such that $u_l^+ = u_l^* + \varepsilon$ and $u_h^+ = u_h^* - \varepsilon$ results in the same d_1^- value as U^*, but requires less budget, as $(u_l^+)^2 + (u_h^+)^2 = (u_l^*)^2 + (u_h^*)^2 + 2\varepsilon(\varepsilon - (u_h^* - u_l^*)) < (u_l^*)^2 + (u_h^*)^2$, thus meaning that U^+ can exploit the extra budget to reduce its d_1^- value. Thus, U^* cannot be optimal, which is a contradiction. □

Lemma 10. Let $\hat{\mathbf{p}} = \frac{\mathbf{p}}{\|\mathbf{p}\|}$, $\mathbf{z} = \langle \hat{\mathbf{p}}, \mathbf{c} \rangle \cdot \hat{\mathbf{p}}$ denote the projection of the sphere's center onto $\mathbf{p}$. The $d_{cos}^-(N, \mathbf{p}) = 1 - \cos(\max\{0, \arccos(\frac{\|\mathbf{z}\|}{\|\mathbf{c}\|}) - \arcsin(\frac{r}{\|\mathbf{c}\|})\})$ and $d_{cos}^+(N, \mathbf{p}) = 1 - \cos(\arccos(\frac{\|\mathbf{z}\|}{\|\mathbf{c}\|}) + \arcsin(\frac{r}{\|\mathbf{c}\|}))$ values are a tight lower and a tight upper bound, respectively, on $d_{cos}(\mathbf{o}, \mathbf{p})$ for any object $\mathbf{o} \in N$ and an arbitrary object $\mathbf{p}$.

Proof. The angle between $\hat{\mathbf{p}}$ and $\mathbf{c}$ is $\arccos(\frac{\|\mathbf{z}\|}{\|\mathbf{c}\|})$. The cone with vertex at the beginning of the coordinate system to which the sphere of N is tangent creates a maximum angle $\arcsin(\frac{r}{\|\mathbf{c}\|})$ between the center of the sphere and any of the tangent points of the cone to the sphere. Let $dist = \|\mathbf{c} - \mathbf{z}\|$. If $dist \leq r$, then the extension of $\mathbf{p}$ intersects the sphere and, thus, $d_{cos}^-(N, \mathbf{p}) = 0$ and $d_{cos}^+(N, \mathbf{p}) = 1 - \cos(\arccos(\frac{\|\mathbf{z}\|}{\|\mathbf{c}\|}) + \arcsin(\frac{r}{\|\mathbf{c}\|}))$ provide tight lower and upper bounds for the cosine distance metric.

If $dist \leq r$, then the extension of $\mathbf{p}$ does not intersects the sphere and, thus, $d_{cos}^-(N, \mathbf{p}) = 1 - \cos(\arccos(\frac{\|\mathbf{z}\|}{\|\mathbf{c}\|}) - \arcsin(\frac{r}{\|\mathbf{c}\|}))$ and $d_{cos}^+(N, \mathbf{p}) = 1 - \cos(\arccos(\frac{\|\mathbf{z}\|}{\|\mathbf{c}\|}) + \arcsin(\frac{r}{\|\mathbf{c}\|}))$ provide tight lower and upper bounds for the cosine distance metric. The proof is thus complete. □

7. EXPERIMENTAL EVALUATION

7.1 Experimental Setting

Methods. We implement our proposed BBM algorithm, discussed in Section 5, as well as the two baseline approaches, SPP and RR, discussed in Section 4, for processing MAMR queries. Moreover, we also implement the naïve algorithm, LIN, which performs an exhaustive linear scan over the database of objects, computing their scores. All algorithms are implemented in C++ and executed on a 3GHz machine.

Datasets. Our evaluation includes real and synthetic datasets, whose characteristics are shown in Table 3. 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 with which a cluster center attracts objects is drawn from a Zipfian distribution with degree 0.8. The number of attributes in the SYNTH datasets varies from 2 up to 8, while the total number of objects varies from 500K up to 20M. The Euclidean distance is used for both attraction and repulsion.

The real dataset FACTUAL is a collection of 2,120,732 locations of places[2] (restaurants, shops, etc.) in the U.S. The Euclidean distance is used for both attraction and repulsion.

The second real dataset, denoted as MIRFLICKR, is a collection of 1 million images used in the evaluation of content-based image retrieval methods.[3] In our experiments, we use the first 200,000 images and the first 50 buckets (out of 150) of edge histogram descriptors, of the MPEG-7 specification [28], as the feature vector. Histogram intersection is used for both attraction and repulsion, as per [28].

Table 3: Dataset Characteristics

Dataset	Cardinality	Dimensions	Attraction	Repulsion
SYNTH	$0.5 - 20 \cdot 10^6$	2 – 8	$-d_2$	$-d_2$
FACTUAL	2,120,732	2	$-d_2$	$-d_2$
MIRFLICKR	200,000	50	$-d_{hi}$	$-d_{hi}$
NBA	21,961	4	$-d_{cos}$	$-d_2$
JESTER	50,691	5	$-d_{cos}$	$-d_2$

The third real dataset, denoted as NBA, is a collection of NBA players statistics[4] covering seasons 1946 until 2009. For our experiments, we use the points, rebounds, assists and blocks per game attributes. Cosine similarity is used for attraction and Euclidean distance for repulsion.

The fourth real dataset, denoted as JESTER, is a collection of ratings from 50,691 users on a set of jokes (Dataset 3),[5] and is widely used in the collaborative filtering literature. In our experiments, we use four popular jokes, which are also used as a gauge in [12]. Cosine similarity is used for attraction and Euclidean distance for repulsion.

Indices. We have implemented our BBM algorithm over R*-trees, the most popular and efficient MBR-based vector space index, and over M-trees, a MBS-based metric space index. We use the R*-tree for all datasets; for MIRFLICKR we also use the M-tree.

Parameters, queries and metrics. We study the performance of the algorithms by varying four parameters: (1) the number of objects $|\mathcal{D}|$, from 500K up to 20M in SYNTH, (2) the number of attributes $|A|$, from 2 up to 8 in SYNTH and from 5 up to 50 in MIRFLICKR, (3) the number of repellers $|\mathcal{R}|$ from 5 up to 100, and (4) the weight parameter λ from 0.01 up to 100.

In each experiment, the attractor is an object uniformly selected from the dataset at random. For the NBA dataset, the attractor is a tuple with attributes values the highest in all statistics. The set of $|\mathcal{R}|$ repellers is constructed progressively by posing $|\mathcal{R}|$ MAMR queries and inserting the answer to $\mathcal{R}$. To quantify performance, we measure the number of examined objects, the number of I/O operations, and the processing time for a MAMR query. All reported quantities for all algorithms are measured *after* the $\mathcal{R}$ repellers have been chosen. The reported values are the averages of 10 distinct queries.

7.2 Results

Effect of λ. We first study the effect of the weight λ as it varies from 0.01 up to 100. Initially, in Figure 4, we fix $|\mathcal{R}| = 5$ and execute all methods on a small subset of FACTUAL containing only 500K tuples. The low cardinality is chosen so that the weakest algorithm (SPP) terminates within reasonable time.

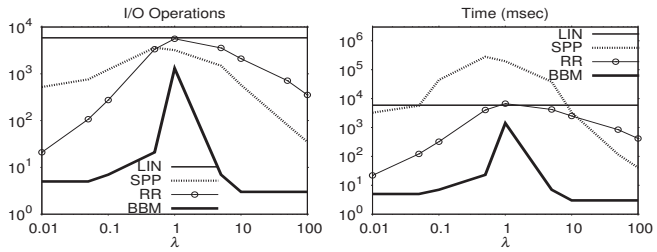
Figure 4a shows (using a logarithmic y-axis) that the number of I/O operations in LIN is independent of λ. The effect of λ in all other methods is similar. All algorithms perform the most I/Os when λ is 1 (or around 1), while they perform the least I/Os at extreme λ values (either very small or very large). This behavior is inherent in MAMR queries and explains why they are more challenging than NN or AFN queries. Large λ values assign more weight to attraction, and thus MAMR query processing resembles NN search. On the other hand, small λ values assign more weight to repulsion resembling AFN search. Values around 1 for λ mean

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

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

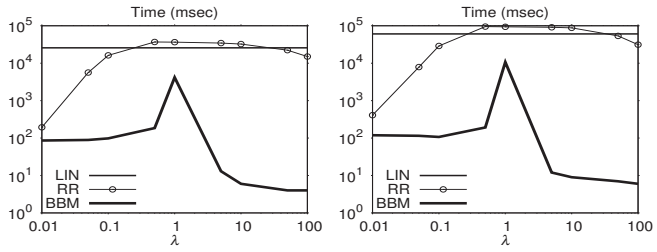
[4] Available at `http://www.basketball-reference.com`

[5] Available at `http://eigentaste.berkeley.edu/dataset/`



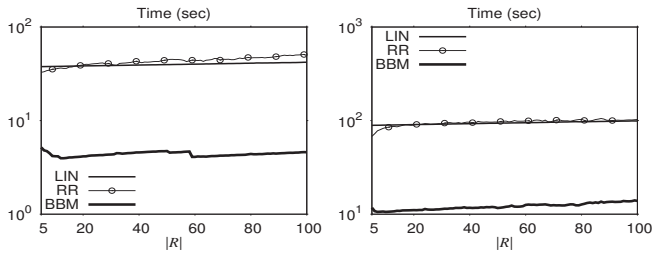
(a) I/Os operations (b) Processing time

Figure 4: Effect of λ, FACTUAL ($|\mathcal{D}| = 500$K, $|\mathcal{R}| = 5$)



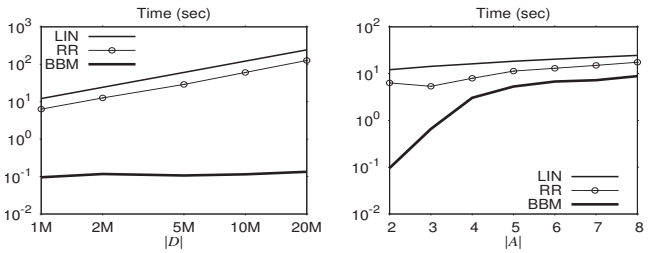
(a) FACTUAL (b) SYNTH ($|\mathcal{A}|=2$, $|\mathcal{D}|=5$M)

Figure 5: Effect of λ ($|\mathcal{R}| = 50$)



(a) FACTUAL (b) SYNTH ($|\mathcal{A}|=2$, $|\mathcal{D}|=5$M)

Figure 6: Effect of $|\mathcal{R}|$ ($\lambda = 1$)



(a) SYNTH ($|\mathcal{A}| = 2$) (b) SYNTH ($|\mathcal{D}| = 1$M)

Figure 7: Effect of $|\mathcal{D}|$ and $|\mathcal{A}|$ ($\lambda = 0.1$, $|\mathcal{R}| = 50$)

that attraction and repulsion are equally strong, making it harder to identify the best object.

BBM consistently outperforms all competitors, in terms of I/Os, by up to 2-3 orders of magnitude (when $\lambda = 5$). Even in its worst performance setting ($\lambda = 1$), BBM performs about 3-4 times less I/Os than its competitors. These results attest that BBM is able to guide the search towards the MAMR answer, effectively pruning non-promising objects. Regarding the baseline methods, they have a similar performance in terms of I/O operations, with RR outperforming SPP in small weights.

Figure 4b draws the total processing time of MAMR queries as a function of λ. The thing to notice is that the dominant factor for LIN, RR and BBM is the I/O cost, whereas for SPP is the CPU cost. SPP is impractical for the majority of the tested λ values, as it is much slower than a linear scan. BBM, on the other hand, is significantly faster than all competitors, up to 2-3 orders of magnitude, while being 3-4 times faster in the worst case. In the remaining experiments, we increase both the database size and the number of repellers and, consequently, we do not consider SPP any further. Moreover, we only report the total processing time.

Figure 5a presents running times on the entire FACTUAL dataset, setting the number of repellers to $|\mathcal{R}| = 50$ and varying λ. The graph shows the same behavior for all methods as that in Figure 4b. BBM is the dominant algorithm significantly outperforming the others in the entire range of λ values. For $\lambda = 1$, BBM is almost an order of magnitude faster, and for $\lambda = 100$, it becomes almost four orders of magnitude faster. Note that RR achieves its best performance for $\lambda = 0.01$; still, BBM is more than two times faster at that extrene setting. Figure 5b shows processing times on the SYNTH dataset with 5M objects and 50 repellers. The findings are analogous to the FACTUAL dataset.

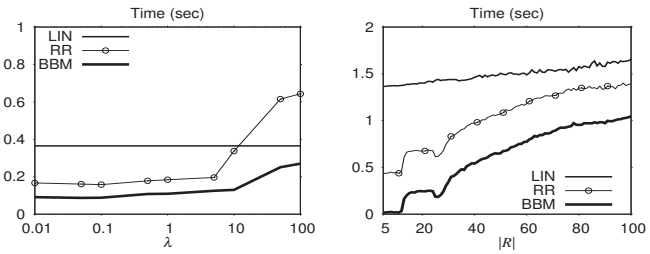
Effect of $|\mathcal{R}|$. We now study the effect of the number $|\mathcal{R}|$ of repellers, while fixing the weight at $\lambda = 1$. First, in Figure 6a we present results on the FACTUAL dataset. As the number of repellers increases the performance of all methods slightly worsens as more distance computations need to be performed. The performance deterioration is only marginal, because, due to the MIN in the MAMR objective function, only one repeller (the nearest) determines the score of an object. The same trends appear in Figure 6b for the SYNTH dataset.

Effect of $|\mathcal{D}|$. In this experiment, we measure MAMR efficiency using synthetic datasets ($|\mathcal{A}| = 2$ attributes) of varying cardinality $|\mathcal{D}|$ from 500K up to 20M, while we fix $\lambda = 0.1$ and $|\mathcal{R}| = 50$. Figure 7a depicts the total processing time as a function of $|\mathcal{D}|$. All methods scale linearly with the dataset cardinality. The effect is apparent for LIN and RR (both axes are in logarithmic scale), but is less pronounced in BBM, whose running time increases at a smaller rate. As a result, the gap between BBM and its competitors increases with $|\mathcal{D}|$.

Effect of $|\mathcal{A}|$. We next study the effect of dimensionality in MAMR processing, using SYNTH ($|\mathcal{D}| = 1$M objects) and vary the number of attributes $|\mathcal{A}|$ from 2 up to 8, while we fix $\lambda = 0.1$ and $|\mathcal{R}| = 50$. Figure 7b depicts the total processing time as a function of $|\mathcal{A}|$. The efficiency of all methods decreases as dimensionality increases. The effect is more pronounced for BBM due to the performance degradation of the underlying index (see, e.g., the study in [3]). This degradation with dimensionality appears in all indices based on bounding surfaces (MBR or MBS), as we also verify in a subsequent experiment, and stems from the fact that node boundaries are forced to occupy more space in higher dimensions (curse of dimensionality). Still, BBM remains more than two times faster than RR and threes time faster than LIN.

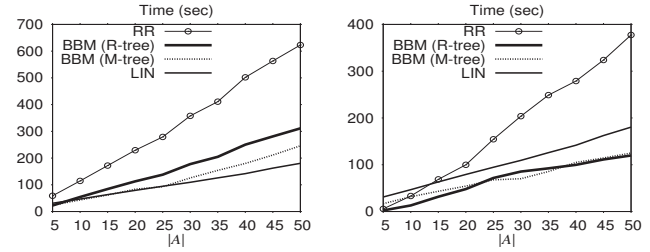
Effect of distance metrics. In this experiment, we investigate the performance of our framework using the real datasets NBA, JESTER, and define attraction using the cosine distance, as depicted in Table 3. A different distance metric, namely histogram intersection distance, will be later used in the experiment of Figure 9.

Figure 8a shows the effect of weight λ on MAMR queries over NBA. The worst-case performance of RR and BBM comes when λ takes its highest value, meaning that attraction (defined in terms of cosine distance) plays the dominant role in defining the MAMR answer. For $\lambda = 100$, BBM is almost three times faster than RR and 1.4 times faster than LIN. A thing to notice here is that the dataset



(a) Effect of λ, NBA ($|\mathcal{R}|=50$) (b) Effect of $|\mathcal{R}|$, JESTER ($\lambda=1$)

Figure 8: Effect of distance metrics



(a) MIRFLICKR ($\lambda = 1$) (b) MIRFLICKR ($\lambda = 0.1$)

Figure 9: Effect of index type ($|\mathcal{R}| = 50$)

is very small (a few tens of thousands of tuples) and thus a linear scan is already a very fast option. As we have discussed, the gain of BBM over its competitors increases with the data cardinality.

Figure 8b shows the effect of the number of repellers on MAMR queries over JESTER. The performance of all methods deteriorates with $|\mathcal{R}|$. In the worst setting BBM is about 1.3 and 1.5 times faster than RR and LIN, respectively. The previous discussion regarding the data cardinality applies in this experiment as well.

Effect of index type. Finally, we study the effect of using other indices in the BBM framework. In particular we test an R*-tree and an M-tree based implementation of BBM using the high-dimensional MIRFLICKR dataset, where attraction and repulsion are defined using the histogram intersection distance. We set the number of repellers to 50 and increase dimensionality from 5 up to 50. Due to the curse of dimensionality, the performance of any index structure is expected to degrade rapidly while the number of attributes increases, making the linear scan method LIN the only option. Still, as Figure 9 shows, BBM performs comparably to LIN and even outperforms it in the case of $\lambda = 0.1$, shown in Figure 9b. Note that the M-tree BBM variant exhibits better scalability in $\lambda = 1$ outperforming the R-tree variant for $|\mathcal{A}| > 5$, and also (slightly) in $\lambda = 0.1$ for $|\mathcal{A}| > 25$.

8. CONCLUSIONS

This work introduced the maximum attraction, minimum repulsion query, which finds applications in various optimization problems, including top-*k* diversification and facility location. By studying the characteristics of the problem, we proposed pruning criteria, and derived upper and lower bounds on the score of objects, which we then applied in an index-based approach, suitable for a large variety of indices and distance metrics. The resulting BBM algorithm is shown to be up to four orders of magnitude faster than baseline approaches and a linear scan of the database.

Acknowledgements. This research has been co-financed by the European Union (European Social Fund – ESF) and Greek national funds through the Operational Program "Education and Lifelong Learning" of the National Strategic Reference Framework (NSRF) – Research Funding Program: Thales. Investing in knowledge society through the European Social Fund.

9. 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] P. Ciaccia, M. Patella, and P. Zezula. M-tree: An efficient access method for similarity search in metric spaces. In *VLDB*, 1997.

[7] Z. Drezner. *Facility location: a survey of applications and methods*. Springer Verlag, 1995.

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

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

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

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

[12] K. Goldberg, T. Roeder, D. Gupta, and C. Perkins. Eigentaste: A constant time collaborative filtering algorithm. *Information Retrieval*, 4(2), 2001.

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

[14] A. Guttman. R-trees: A dynamic index structure for spatial searching. In *SIGMOD*, 1984.

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

[16] 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.

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

[18] N. Katayama and S. Satoh. The SR-tree: An index structure for high-dimensional nearest neighbor queries. In *SIGMOD*, 1997.

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

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

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

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

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

[24] Y. Sakurai, M. Yoshikawa, S. Uemura, and H. Kojima. The A-tree: An index structure for high-dimensional spaces using relative approximation. In *VLDB*, 2000.

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

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

[27] D. A. White and R. Jain. Similarity indexing with the SS-tree. In *ICDE*, 1996.

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