Computer Engineering and Intelligent Systems
ISSN 2222-1719 (Paper) ISSN 2222-2863 (Online)
Vol.4, No.12, 2013

Aggregation of Data by Using Top -K Spatial Query Preferences

Hanaa Mohsin Ali Al-Abboodi

College of Engineering , University of Babylon,Iraq

*Email:hanaamohsin77@Gmail.com

Abstract:

A spatial database is a database that is optimized to store and query data that represents objects defined in a geometric space. A spatial preference query ranks objects based on the qualities of features in their spatial neighborhood. For example, using a real estate agency database of flats for lease, a customer may want to rank the flats with respect to the appropriateness of their location, defined after aggregating the qualities of other features (e.g., restaurants, cafes, hospital, market, etc.) within their spatial neighborhood. Such a neighborhood concept can be specified by the user via different functions. It can be an explicit circular region within a given distance from the flat. Another intuitive definition is to assign higher weights to the features based on their proximity to the flat. In this paper, we formally define spatial preference queries and propose appropriate indexing techniques and search algorithms for them. Extensive evaluation of our methods on both real and synthetic data reveals that an optimized branch-and-bound solution is efficient and robust with respect to different parameters.

Index Terms: Query processing, spatial databases.

1. INTRODUCTION

Spatial database systems manage large collections of geographic entities, which apart from spatial attributes contain non-spatial information (e.g., name, size, type, price, etc.). In this paper, we study an interesting type of preference queries, which select the best spatial location with respect to the quality of facilities in its spatial neighborhood.

Given a set D of interesting objects (e.g., candidate locations), a top-k spatial preference query retrieves the k objects in D with the highest scores. The score of an object is defined by the quality of features (e.g., facilities or services) in its spatial neighborhood. As a motivating example, consider a real estate agency office that holds a database with available flats for lease. Here “feature” refers to a class of objects in a spatial map such as specific facilities or services. A customer may want to rank the contents of this database with respect to the quality of their locations, quantified by aggregating non-spatial characteristics of other features (e.g., restaurants, cafes, hospital, market, etc.) in the spatial neighborhood of the flat (defined by a spatial range around it). Quality may be subjective and query-parametric. For example, a user may define quality with respect to non-spatial attributes of restaurants around it (e.g., whether they serve seafood, price range, etc.).

As another example, the user (e.g., a tourist) wishes to find a hotel p that is close to a high-quality restaurant and a high quality cafe.

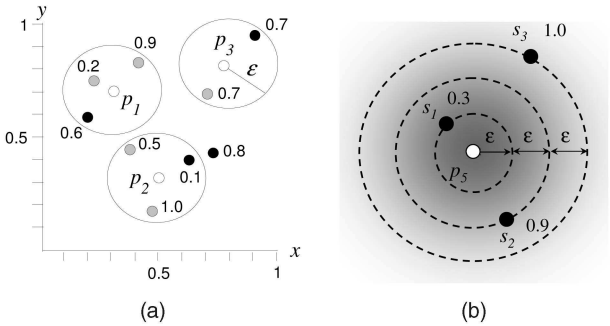


Fig. 1. Examples of top-k spatial preference queries. (a) Range score, ϵ = 0:2 km. (b) Influence score, ϵ = 0:2 km.

Fig. 1a illustrates the locations of an object data set D (hotels) in white, and two feature data sets: the set F_1 (restaurants) in gray, and the set F_2 (cafes) in black. Feature points are labeled by quality values that can be obtained from rating providers (e.g., http://www.zagat.com/). For the ease of discussion, the qualities are

normalized to values in [0; 1]. The score $\tau(p)$ of a hotel *p* is defined in terms of: 1) the maximum quality for each feature in the neighborhood region of *p*, and 2) the aggregation of those qualities.

A simple score instance, called the range score, binds the neighborhood region to a circular region at p with radius ϵ (shown as a circle), and the aggregate function to SUM. For instance, the maximum quality of gray and black points within the circle of p_1 are 0.9 and 0.6, respectively, so the score of p_1 is $\tau(p_1)$ = 0:9 + 0:6 = 1:5. Similarly, we obtain $\tau(p_2)$ = 1:0 + 0:1 = 1:1 and $\tau(p_3)$ = 0:7 + 0:7 = 1:4. Hence, the hotel p1 is returned as the top result.

In fact, the semantics of the aggregate function is relevant to the user's query. The SUM function attempts to balance the overall qualities of all features. For the MIN function, the top result becomes p_3, with the score $\tau(p_3)$ = min {0:7; 0:7} = 0:7. It ensures that the top result has reasonably high qualities in all features. For the MAX function, the top result is p_2, with $\tau(p_2)$ = max {1:0; 0:1} = 1:0. It is used to optimize the quality in a particular feature, but not necessarily all of them.

The neighborhood region in the above spatial preference query can also be defined by other score functions. A meaningful score function is the influence score (see Section 4). As opposed to the crisp radius _ constraint in the range score, the influence score smoothens the effect of ϵ and assigns higher weights to cafes that are closer to the hotel. Fig. 1b shows a hotel p_5 and three cafes s_1; s_2; s_3 (with their quality values). The circles have their radii as multiples of ϵ. Now, the score of a cafe si is computed by multiplying its quality with the weight 2^{-j}, where j is the order of the smallest circle containing si. For example, the scores of s_1, s_2, and s_3 are $0:3/2^1$ = 0:15, $0:9/2^2$ = 0:225, and $1:0/2^3$ = 0:125, respectively. The influence score of p_5 is taken as the highest value (0.225).

Traditionally, there are two basic ways for ranking objects: 1) spatial ranking, which orders the objects according to their distance from a reference point, and 2) non-spatial ranking, which orders the objects by an aggregate function on their non

spatial values. Our top-*k* spatial preference query integrates these two types of ranking in an intuitive way. As indicated by our examples, this new query has a wide range of applications in service recommendation and decision support systems.

To our knowledge, there is no existing efficient solution for processing the top-*k* spatial preference query. A brute force approach (to be elaborated in Section 3.2) for evaluating it is to compute the scores of all objects in D and select the top-k ones. This method, however, is expected to be very expensive for large input data sets. In this paper, we propose alternative techniques that aim at minimizing the I/O accesses to the object and feature data sets, while being also computationally efficient. Our techniques apply on spatial-partitioning access methods and compute upper score bounds for the objects indexed by them, which are used to effectively prune the search space. Specifically, we contribute the branch-and-bound (BB) algorithm and the feature join (FJ) algorithm for efficiently processing the top-k spatial preference query.

Furthermore, this paper studies three relevant extensions that have not been investigated in our preliminary work [1]. The first extension (Section 3.4) is an optimized version of BB that exploits a more efficient technique for computing the scores of the objects. The second extension (Section 3.6) studies adaptations of the proposed algorithms for aggregate functions other than SUM, e.g., the functions MIN and MAX. The third extension (Section 4) develops solutions for the top-k spatial preference query based on the influence score.

The rest of this paper is structured as follows: Section 2 provides background on basic and advanced queries on spatial databases, as well as top-k query evaluation in relational databases. Section 2 defines the top-*k* spatial preference query and presents our solutions. Section 3 studies the query extension for the influence score. In Section 3, our query algorithms are experimentally evaluated with real and synthetic data. Finally, Section 4 concludes the paper with future research directions.

2. SPATIAL DATA MODEL AND QUERY LANGUAGE

A spatial data model [5], [3] is a type of data-abstraction that hides the details of data-storage. There are two common models of spatial information: field-based and object based. The field-based model treats spatial information such as altitude, rainfall and temperature as a collection of spatial functions transforming a space-partition to an attribute domain. The object-based model treats the information space as if it is populated by discrete, identifiable, spatially referenced entities. The operations on spatial objects include distance and boundary. The operations on fields include local, focal, and zonal operations, as shown in Table 1.

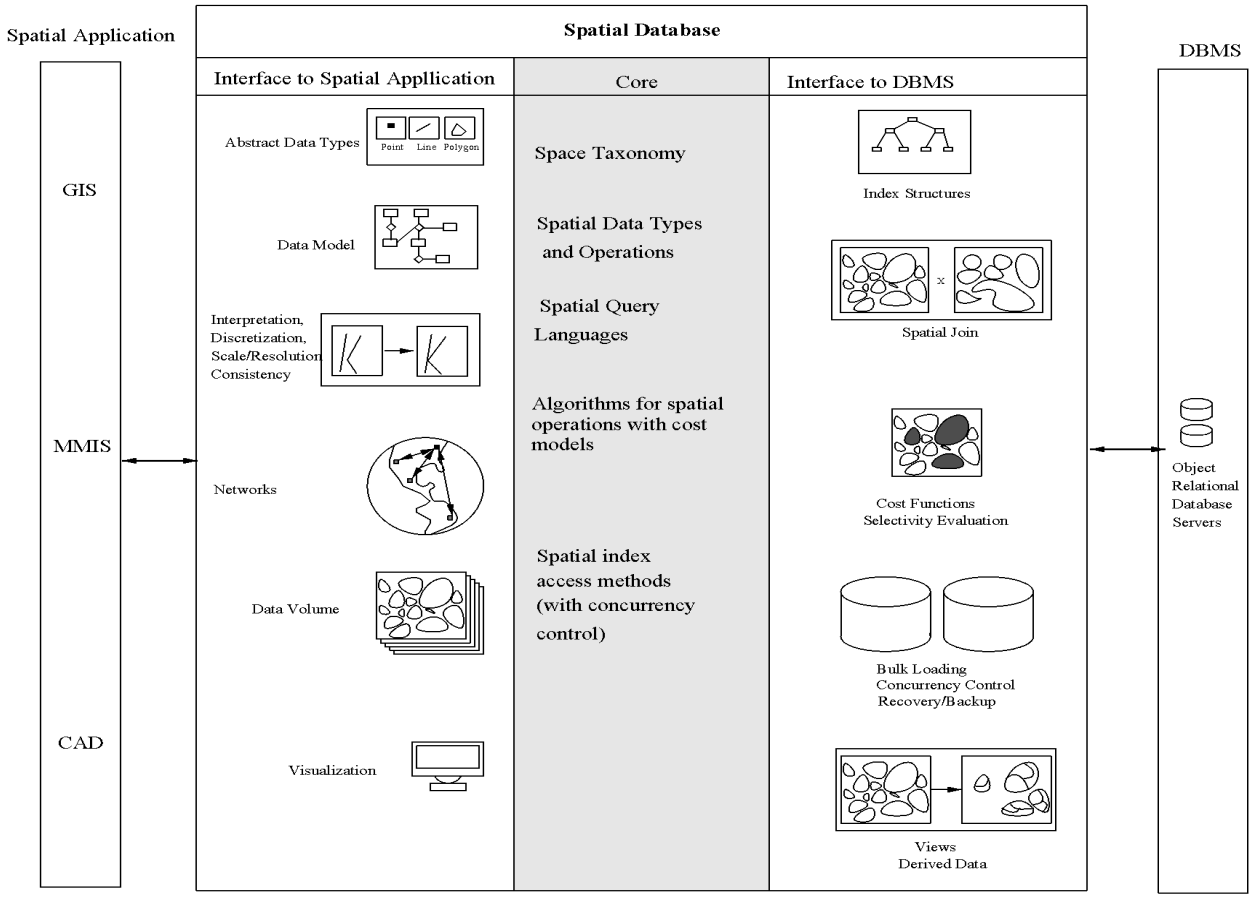


Fig. 2. Three-layer architecture.

TABLE 1
A SAMPLE OF SPATIAL OPERATIONS

Data model	Operator Group	Operation
Vector Object	Set-Oriented	equals, is a member of, is empty, is a subset of, is disjoint from, intersection, union, difference, cardinality
	Topological	boundary, interior, closure, meets, overlaps, is inside, covers, connected, components, extremes, is within
	Metric	distance, bearing/angle, length, area, perimeter.
	Direction	east, north, left, above, between.
	Network	successors, ancestors, connected, shortest-path
	Dynamic	translate, rotate, scale, shear, split, merge
Raster field	Local	Point-wise sums, differences, maximums, means, etc
	Focal	slope, aspect, weighted average of neighborhood
	Zonal	sum or mean or maximum of field values in each zone

The fields may be continuous, differentiable, discrete, and isotropic or anisotropic, with positive or negative autocorrelation. Certain field operations (slope or interpolation) assume certain field properties (differentiable or positive autocorrelation).

An implementation of a spatial data model in the context of object-relational databases consists of a set of spatial data types and the operations on those types. Much work has been done over the last decade on the design

of spatial Abstract Data Types (ADTs) and their embedding in a query language. Consensus is slowly emerging via standardization efforts, and recently the OGIS consortium [2] has proposed a specification for incorporating 2D geospatial ADTs in SQL.

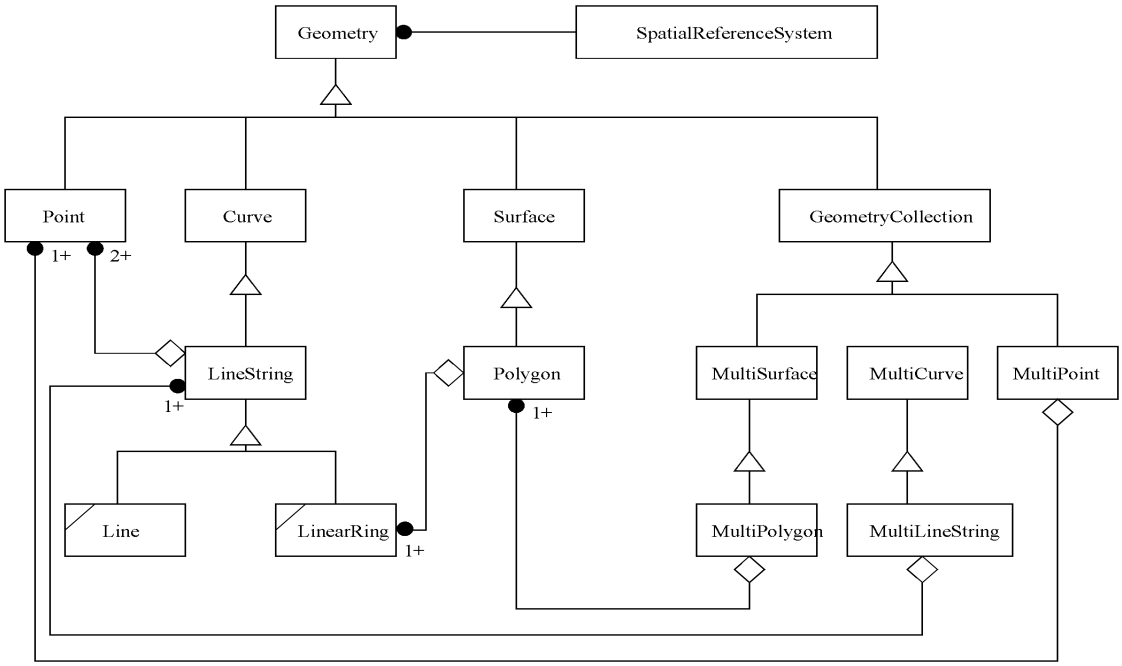


Fig. 4. Spatial data type hierarchy

Fig. 4, which illustrates this spatial data-type hierarchy, consists of Point, Curve, and Surface classes and a parallel class of Geometry Collection. The basic operations operative on all datatypes is shown in Table 2.

TABLE 2
REPRESENTATIVE FUNCTIONS SPECIFIED BY OGIS

Basic Functions	SpatialReference()	Returns the Reference System of the geometry
	Envelope()	The minimum bounding rectangle of the geometry
	Export()	Convert the geometry into a different representation.
	IsEmpty()	Tests if the geometry is a empty set or not
	IsSimple()	Returns True if the geometry is simple(no self-intersection)
	Boundary()	Returns the boundary of the geometry
Topological/ Set Operators	Equal	Tests if the geometries are spatially equal
	Disjoint	Tests if the geometries are disjoint
	Intersect	Tests if the geometries intersect
	Touch	Tests if the geometries touch each other
	Cross	Tests if the geometries cross each other
	Within	Tests if the given geomtry is within another given geometry
	Contains	Tests if the given geometry contains another given geometry
	Overlap	Tests if the geometry overlaps another geometry
Spatial Analysis	Distance	Returns the shortest distance between two geometries
	Buffer	Returns a geometry that represents all points whose distance from the given is less than or equal to the specified distance
	ConvexHull	Returns the convex hull of the geometry
	Intersection	Returns the intersection of two geometries
	Union	Returns the union of two geometries
	Difference	Returns the difference of two geometries
	SymDiff	Returns the symmetric difference of two geometries

The topological operations are based on the ubiquitous nine intersections model [10]. Using the OGIS specification, common spatial queries can be intuitively posed in SQL. For example, the query Find all lakes which have an area greater than 5 sq. km. and are within 20 km. from the campgrounds can be posed as shown in Fig. 3a. Other example GIS queries which can be implemented using OGIS operations are provided in Table 3. The OGIS specification is confined to topological and metric operations on vector data types. Other interesting classes of operations are network, direction, dynamic and the field operations of focal, local and zonal (see Table 1). While standards for field based raster data types are still emerging, specifically designed for cartographic modeling and for general multidimensional discrete objects (satellite images, X-rays, etc.), are important milestones.

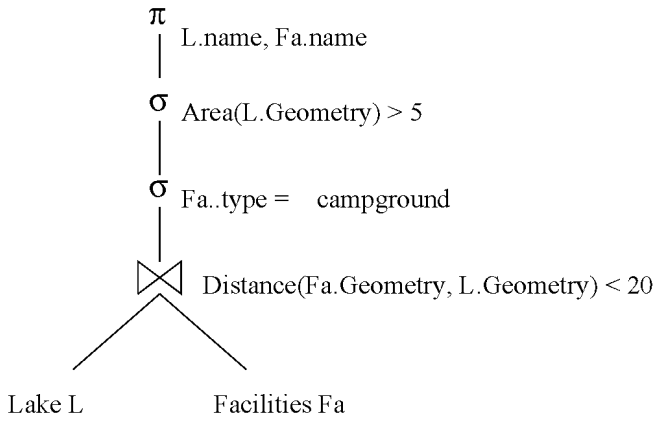
TABLE 3

TYPICAL SPATIAL QUERIES FROM GIS

Single Table Queries	
Grouping	Recode all land with silty soil to silt-loam soil
Isolate	Select all land owned by Steve Steiner
Classify	If the population density is less than 100 people / sq. mi., land is acceptable
Scale	Change all measurement to the metric system
Rank	If the road is an Interstate, assign it code 1; if the road is a state or US Highway, assign it code 2; otherwise assign it code 3
Evaluate	If the road code is 1, then assign it Interstate; if the road code is 2, then assign it Main Artery; if the road code is 3, assign it Local Road
Rescale	Apply a function to the population density
Multi-Table Queries	
Attribute Join	Join the Forest layer with the layer containing forest-cover codes
Zonal	Produce a new map showing state populations given county population
Registration	Align two layers to a common grid reference
Spatial Join	Overlay the land-use and vegetation layers to produce a new layer

```
SELECT    L.name, Fa.name
FROM      Lake L, Facilities Fa
WHERE     Area(L.Geometry) > 5 AND
          Fa.type =  campground  AND
          Distance(Fa.Geometry, L.Geometry) <  20
```

(a)



(b)

Fig. 3: (a) SQL query with spatial operators; (b) corresponding query tree.

2.1 Spatial Query Processing

The efficient processing of spatial queries requires both efficient representation and efficient algorithms. Common representations of spatial data in an object model include spaghetti, the node-arc-area (NAA) model, the doubly connected edge- list (DCEL), and boundary representation [7], some of which are shown in Fig. 5 using entity-relationship diagrams? The NAA model differentiates between the topological concepts (node, arc, areas) and the embedding space (points, lines, areas). The spaghetti-ring and DCEL focus on the topological concepts. The representation of the field data model includes a regular tessellation (triangular, square, hexagonal grid), as well as triangular irregular networks (TIN).

The spatial queries [7], shown in Table 3, are often processed using filter and refine techniques. Approximate geometry such as the minimal orthogonal bounding rectangle of an extended spatial object is first used to filter out many irrelevant objects quickly. Exact geometry is then used for the remaining spatial objects to complete the processing. Strategies for range-queries include a scan and index-search in conjunction with the plane-sweep algorithm [5]. Strategies for the spatial-join include the nested loop, tree matching [5], when indices are present on all participating relations, and space partitioning [2], in the absence of indices. To speed up computation for large spatial objects (it is common for polygons to have 1,000 or more edges), object indices are used in extended filtering. Strategies such as object approximation and tree matching originated in spatial-databases, and can potentially be applied in other

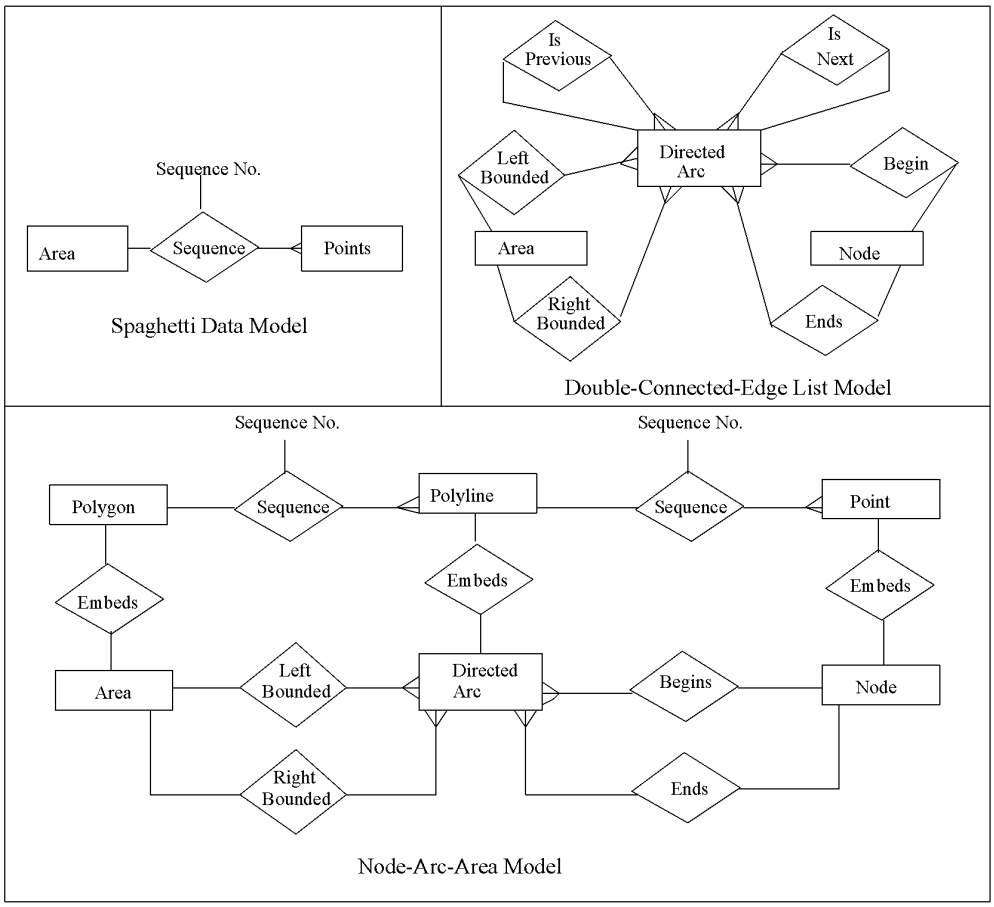


Fig.5. Entity relationship diagrams for common representations of spatial data.

2.2 Spatial File Organization and Indices

The physical design of a spatial database optimizes the instructions to storage devices for performing common operations on spatial data files. File designs for secondary storage include clustering methods as well as spatial hashing methods. The design of spatial clustering techniques is more difficult compared to the design of traditional clustering because there is no natural order in multidimensional space where spatial data resides. This is only complicated by the fact that the storage disk is a logical one-dimensional device. Thus, what is needed is a mapping from a higher dimensional space to a one dimensional space that is distance-preserving: So that elements that are close in space are mapped onto nearby points on the line, and one-one: no two points in the

space are mapped onto the same point on the line [2]. Several mappings, none of them ideal, have been proposed to accomplish this. The most prominent ones include row-order, z-order, and the Hilbert-curve (see Fig. 6).

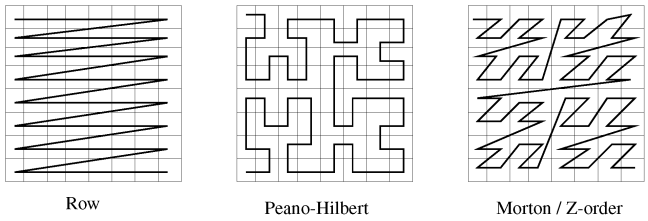


Fig.6. Space-filling curves to linearize a multidimensional space.

Metric clustering techniques use the notion of distance to group nearest neighbors together in a metric space. Topological clustering methods like connectivity-clustered access methods [7] use the min-cut partitioning of a graph representation to efficiently support graph traversal operations. The physical organization of files can be supplemented with indices, which are data-structures to improve the performance of search operations.

Classical one-dimensional indices such as the B+ tree can be used for spatial data by linearizing a multidimensional space using a space-filling curve such as the Z-order (see Fig. 6). A large number of spatial indices [3] have been explored for multidimensional Euclidean space. Representative indices for point objects include Grid files, multidimensional grid files [8], Point-Quad-Trees, and Kd-trees. Representative indices for extended objects include the R-tree family, the Field tree, Cell tree, BSP tree, and Balanced and Nested grid files.

One of the first access methods created to handle extended object. The R-tree is a height balanced natural extension of the B+ tree for higher dimensions. Objects are represented in the R-tree by their minimum bounding rectangles (MBRs). Nonleaf nodes are composed of entries of the form (R, child-pointer), where R is the MBR of all entries contained in the child pointer. Leaf nodes contain the MBRs of the data objects. To guarantee good space utilization and height-balance, the parent MBRs are allowed to overlap.

.

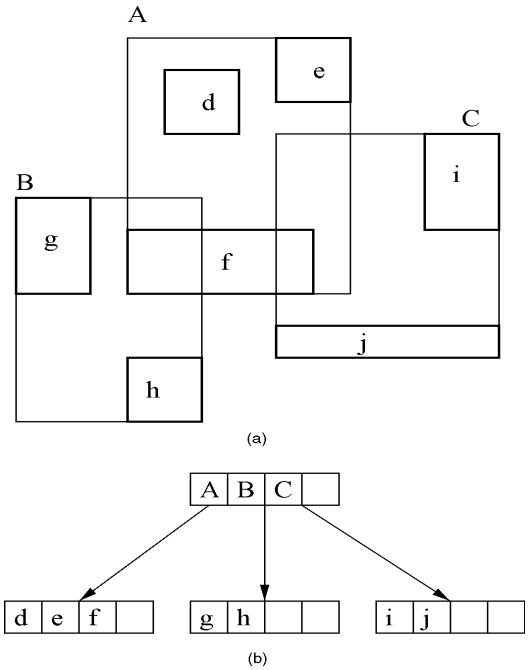


Fig. 7: (a) Spatial objects (bold) arranged in R-tree hierarchy; (b) R-tree file structure on disk.

Fig. 7a illustrates the spatial objects organized in an R-tree, while Fig. 7b shows the file structure where the nodes correspond to disk pages. Many variations of the R-tree structure exist whose main emphasis is on discovering new strategies to maintain the balance of the tree in case of a split and to minimize the overlap of the MBRs in order to improve the search time.

Concurrency control for spatial access methods [6] is provided by the R-link tree, which is a variant of the R-tree with additional sibling pointers that allow the tracking of modifications. Concurrency is provided during operations such as search, insert, and delete. The R-link tree is also recoverable in a write-ahead logging environment

2.3 Other Accomplishments

Spatial applications like NASA's Earth Observation System (EOS) have some of the largest data sets encountered in any application to date. This has prompted new research in database-file design for storage on tertiary storage devices such as juke-boxes. Representative results include those from the. High-performance spatial applications such as flight simulators with geographic accuracy have triggered the development of new parallel formalizations for the range query and the spatial join query, including declustering methods and dynamic-load balancing techniques for multidimensional spatial data [8], [9]. Other interesting developments include hierarchical algorithms for shortest path computation [4] and view materialization [6].

3. EXPERIMENTAL EVALUATION

In this section, we compare the efficiency of the proposed algorithms using real and synthetic data sets. Each data set is indexed by an R-tree with 4 K bytes page size. We used an LRU memory buffer whose default size is set to 0.5 percent of the sum of tree sizes (for the object and feature trees used). Our algorithms were implemented in C++ and experiments were run on a Pentium D 2.8 GHz PC with 1 GB of RAM. In all experiments, we measure both the I/O cost (in number of page faults) and the total execution time (in seconds) of our algorithms. Section 5.1 describes the experimental settings. Sections 3.2 and 3.3 study the performance of the proposed algorithms for queries with range scores and influence scores, respectively. We then present our experimental findings on real data in Section 3.4.

3.1 Experimental Settings

We used both real and synthetic data for the experiments. The real data sets will be described in Section 3.4. For each synthetic data set, the coordinates of points are random values uniformly and independently generated for different dimensions. By default, an object data set contains 200K points and a feature data set contains 100K points. The point coordinates of all data sets are normalized to the 2D space $[0; 10000]^2$.

For a feature data set Fc, we generated qualities for its points such that they simulate a real-world scenario: facilities close to (far from) a town center often have high (low) quality. For this, a single anchor point s? is selected such that its neighborhood region contains high number of points. Let $dist_{min}$ ($dist_{max}$) be the minimum (maximum) distance of a point in F_c from the anchor $s_\star$. Then, the quality of a feature point s is generated as

$$\omega(s) = \left(\frac{dist_{max} - dist(s, s_\star)}{dist_{max} - dist_{min}} \right)^{\vartheta},$$

where $dist(s, s_\star)$ stands for the distance between s and $s_\star$, and ϑ controls the skewness (default: 1.0) of quality distribution. In this way, the qualities of points in *Fc* lie in [0; 1] and the points closer to the anchor have higher qualities. Also, the quality distribution is highly skewed for large values of ϑ.

We study the performance of our algorithms with respect to various parameters, which are displayed in Table 4 (their default values are shown in bold). In each experiment, only one parameter varies while the others are fixed to their default values.

TABLE 4
Range of Parameter Values

Parameter	Values
Aggregate function	**SUM**, MIN, MAX
Buffer size (%)	0.1, 0.2, **0.5**, 1, 2, 5, 10
Object data size, $\|\mathcal{D}\|$ ($\times$1000)	100, **200**, 400, 800, 1600
Feature data size, $\|\mathcal{F}\|$ ($\times$1000)	50, **100**, 200, 400, 800
Number of results, k	**1**, 2, 4, 8, 16, 32, 64
Number of features, m	1, **2**, 3, 4, 5
Query range, ϵ	10, 20, **50**, 100, 200

3.2 Performance on Queries with Range Scores

This section studies the performance of our algorithms for top-k spatial preference queries on range scores.

TABLE 5
Effect of the Aggregate Function, Range Scores

SUM function	SP	GP	BB	BB*	FJ
I/O	350927	22594	2033	1535	489
Time (s)	635.0	32.7	3.0	2.0	1.3

MIN function	SP	GP	BB	BB*	FJ
I/O	235602	16254	611	615	47
Time (s)	426.8	22.7	0.9	0.8	0.2

MAX function	SP	GP	BB	BB*	FJ
I/O	402704	26128	228	186	8
Time (s)	742.8	38.2	0.3	0.2	0.1

Table 5 shows the I/O cost and execution time of the algorithms, for different aggregate functions (SUM, MIN, and MAX). GP has lower cost than SP because GP computes the scores of points within the same leaf node concurrently. The incremental computation technique (used by SP and GP) derives a tight upper bound score (of each point) for the MIN function, a partially tight bound for SUM, and a loose bound for MAX. This explains the performance of SP and GP across different aggregate functions. However, the costs of the other methods are mainly influenced by the effectiveness of pruning. BB employs an effective technique

to prune unqualified non-leaf entries in the object tree so it outperforms GP. The optimized score computation method enables BB* to save on average 20 percent I/O and 30 percent time of BB. FJ outperforms its competitors as it discovers qualified combination of feature entries early.

We ignore SP in subsequent experiments, and compare the cost of the remaining algorithms on synthetic data sets with respect to different parameters.

Next, we empirically justify the choice of using level-1 entries of feature trees Fc for the upper bound score computation routine in the BB algorithm. In this experiment, we use the default parameter setting and study how the number of node accesses of BB is affected by the level of Fc used.

TABLE 6
Effect of the Level of Fc Used for Upper Bound Score Computation in the BB Algorithm

Level	Node accesses (NA)			Upper bound score computation	
	Total	of $\mathcal{D}$	of $\mathcal{F}_c$	# of calls	NA of $\mathcal{F}_c$ per call
0	3350	53	3297	4	617.5
1	2365	130	2235	4	15
2	13666	930	12736	14	2

Table 6 shows the decomposition of node accesses over the tree D and the trees Fc, and the statistics of upper bound score computation. Each accessed non-leaf node of D invokes a call of the upper bound score computation routine.

When level-0 entries of *Fc* are used, each upper bound computation call incurs a high number (617.5) of node accesses (of *Fc*). On the other hand, using level-2 entries for upper bound computation leads to very loose bounds, making it difficult to prune the leaf nodes of D. Observe that the total cost is minimized when level-1 entries (of *Fc*) are used. In that case, the node accesses per upper bound computation call is low (15), and yet the obtained bounds are tight enough for pruning most leaf nodes of D.

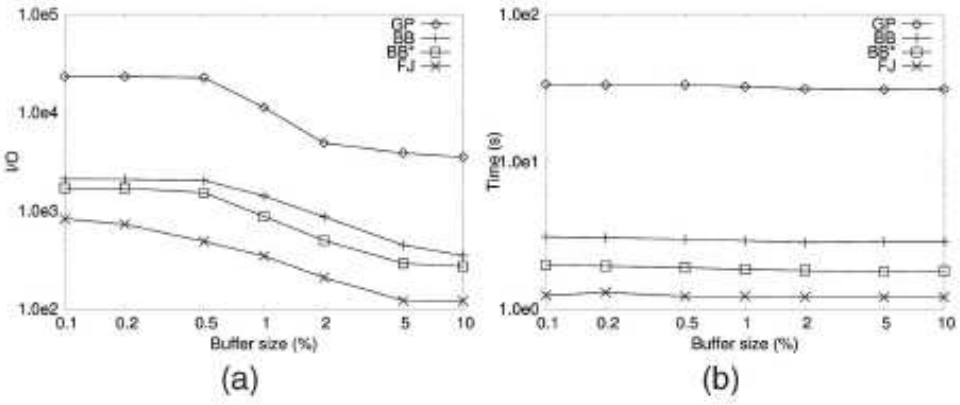


Fig. 8. Effect of buffer size, range scores. (a) I/O. (b) Time.

Fig. 8 plots the cost of the algorithms as a function of the buffer size. As the buffer size increases, the I/O of all algorithms drops. FJ remains the best method, BB* the second, and BB the third; all of them outperform GP by a wide margin. Since the buffer size does not affect the pruning effectiveness of the algorithms, it has a small impact on the execution time

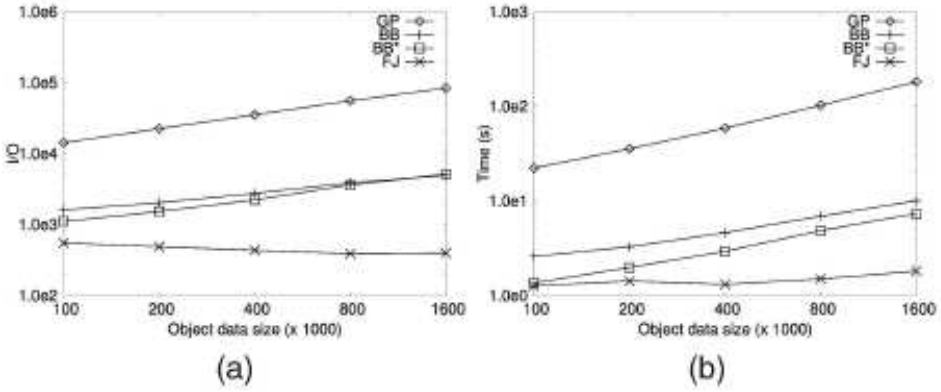


Fig. 9. Effect of $|\mathcal{D}|$, range scores. (a) I/O. (b) Time.

Fig. 9 compares the cost of the algorithms with respect to the object data size $|\mathcal{D}|$. Since the cost of *FJ* is dominated by the cost of joining feature data sets, it is insensitive to $|\mathcal{D}|$. On the other hand, the cost of the other methods (GP, BB, and BB*) increases with $|\mathcal{D}|$ as score computations need to be done for more objects in D.

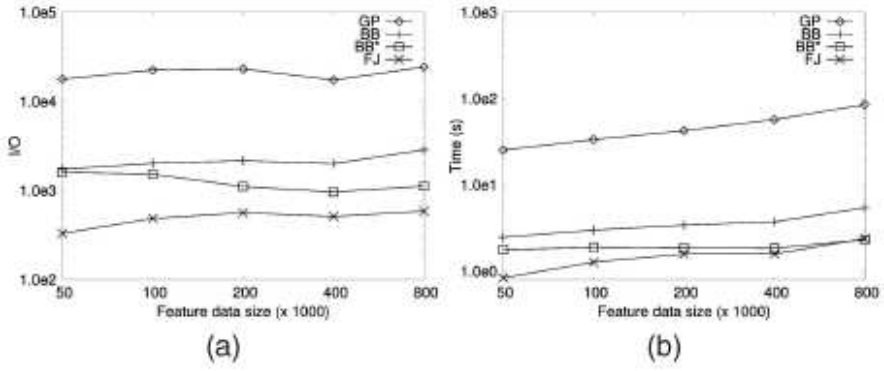


Fig. 10. Effect of $|\mathcal{D}|$, range scores. (a) I/O. (b) Time.

Fig. 10 plots the I/O cost of the algorithms with respect to the feature data size $|\mathcal{F}|$ (of each feature data set). As $|\mathcal{F}|$ increases, the cost of GP, BB, and FJ increases. In contrast, BB* experiences a slight cost reduction as its optimized score computation method (for objects and non leaf entries) is able to perform pruning early at a large $|\mathcal{F}|$ value.

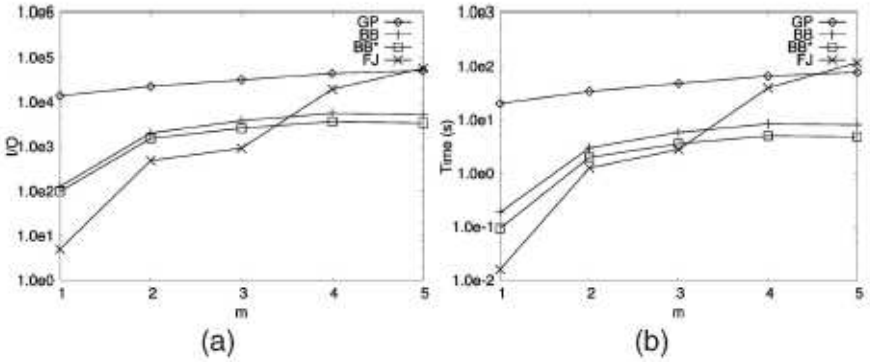


Fig. 11. Effect of *m*, range scores. (a) I/O. (b) Time.

Fig. 11 plots the cost of the algorithms with respect to the number m of feature data sets. The costs of GP, BB, and BB* rise linearly as m increases because the number of component score computations is at most linear to m. On the other hand, the cost of *FJ* increases significantly with m, because the number of qualified combinations of entries is exponential to *m*.

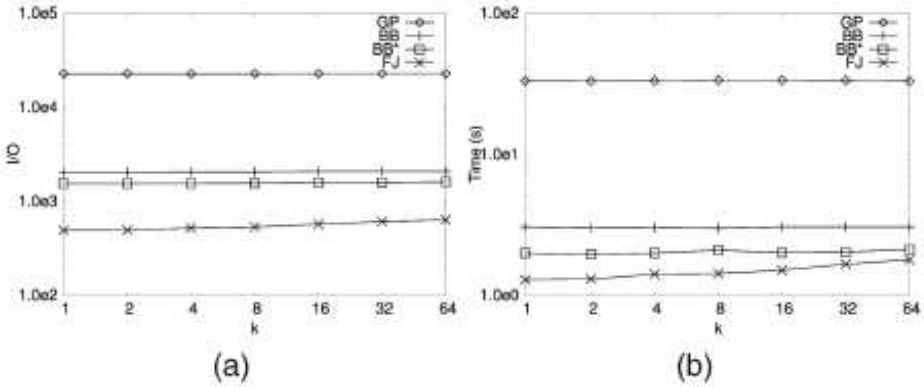


Fig. 12. Effect of *k*, range scores. (a) I/O. (b) Time.

Fig. 12 shows the cost of the algorithms as a function of the number *k* of requested results. GP, BB, and BB* compute the scores of objects in D in batches, so their performance is insensitive to *k*. As *k* increases, FJ has weaker pruning power and its cost increases slightly.

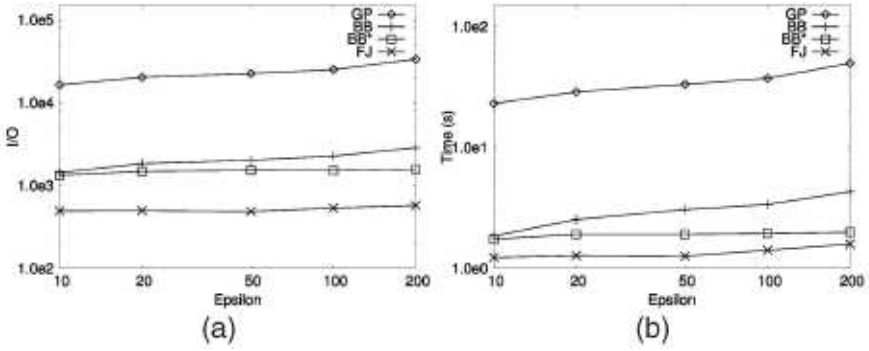


Fig. 13. Effect of *k*, range scores. (a) I/O. (b) Time.

Fig. 13 shows the cost of the algorithms, when varying the query range *k*. As *k* increases, all methods access more nodes in feature trees to compute the scores of the points. The difference in execution time between BB* and *FJ* shrinks as ϵ increases. In summary, although *FJ* is the clear winner in most of the experimental instances, its performance is significantly affected by the number *m* of feature data sets. BB* is the most robust algorithm to parameter changes and it is recommended for problems with large *m*.

4. CONCLUSION

In this paper, we studied top-k spatial preference queries, which provide a novel type of ranking for spatial objects based on qualities of features in their neighborhood. The neighborhood of an object p is captured by the scoring function: 1) the range score restricts the neighborhood to a crisp region centered at p, whereas 2) the influence score relaxes the neighborhood to the whole space and assigns higher weights to locations closer to p. We presented five algorithms for processing top-k spatial preference queries. The baseline algorithm SP computes the scores of every object by querying on feature data sets. The algorithm GP is a variant of SP that reduces I/O cost by computing scores of objects in the same leaf node concurrently. The algorithm BB derives upper bound scores for non-leaf entries in the object tree, and prunes those that cannot lead to better results. The algorithm BB* is a variant of BB that utilizes an optimized method for computing the scores of objects (and upper bound scores of nonleaf entries). The algorithm FJ performs a multi-way join on feature trees to obtain qualified combinations of feature points and then search for their relevant objects in the object tree. Based on our experimental findings, BB* is scalable to large data sets and it is the most robust algorithm with respect to various parameters. However, FJ is the best algorithm in cases where the number m of feature data sets is low and each feature data set is small. In the future, we will study the top-k spatial preference query on a road network, in which the distance between two points is defined by their shortest path distance rather than their euclidean distance. The challenge is to develop alternative methods for computing the upper bound scores for a group of points on a road network.

5. REFERENCES

[1] Man Lung Yiu, Hua Lu, Member, IEEE, Nikos Mamoulis, and Michail Vaitis, “Ranking Spatial Data by Quality Preferences”, IEEE Transactions On Knowledge And Data Engineering, Vol. 23, No. 3, March 2011.

[2] N. Bruno, L. Gravano, and A. Marian, “Evaluating Top-k Queries over Web-Accessible Databases,” Proc. IEEE Int’l Conf. Data Eng. (ICDE), 2002.

[3] A. Guttman, “R-Trees: A Dynamic Index Structure for Spatial Searching,” Proc. ACM SIGMOD, 1984.

[4] G.R. Hjaltason and H. Samet, “Distance Browsing in Spatial Databases,” ACM Trans. Database Systems, vol. 24, no. 2, pp. 265- 318, 1999.

[5] R. Weber, H.-J. Schek, and S. Blott, “A Quantitative Analysis and Performance Study for Similarity-Search Methods in High- Dimensional Spaces,” Proc. Int’l Conf. Very Large Data Bases (VLDB), 1998.

[6] K.S. Beyer, J. Goldstein, R. Ramakrishna, and U. Shaft, “When is ‘Nearest Neighbor’ Meaningful?” Proc. Seventh Int’l Conf. Database Theory (ICDT), 1999.

[7] R. Fagin, A. Lotem, and M. Naor, “Optimal Aggregation Algorithms for Middleware,” Proc. Int’l Symp. Principles of Database Systems (PODS), 2001.

[8] I.F. Ilyas, W.G. Aref, and A. Elmagarmid, “Supporting Top-k Join Queries in Relational Databases,” Proc. 29th Int’l Conf. Very Large Data Bases (VLDB), 2003.

[9] N. Mamoulis, M.L. Yiu, K.H. Cheng, and D.W. Cheung, “Efficient Top-k Aggregation of Ranked Inputs,” ACM Trans. Database Systems, vol. 32, no. 3, p. 19, 2007.

[10] D. Papadias, P. Kalnis, J. Zhang, and Y. Tao, “Efficient OLAP Operations in Spatial Data Warehouses,” Proc. Int’l Symp. Spatial and Temporal Databases (SSTD), 2001.

[11] S. Hong, B. Moon, and S. Lee, “Efficient Execution of Range Top-k Queries in Aggregate R-Trees,” IEICE Trans. Information and Systems, vol. 88-D, no. 11, pp. 2544-2554, 2005.

[12] T. Xia, D. Zhang, E. Kanoulas, and Y. Du, “On Computing Top-t Most Influential Spatial Sites,” Proc. 31st Int’l Conf. Very Large Data Bases (VLDB), 2005.