

Chapter 2

Background on Spatial Data Management and Exploration

Before studying mobility data, we have to make a short tour at the (stationary) spatial domain. For decades, spatial information has been studied thoroughly; from Cartography and Geodesy to Geographical Information Systems (GIS) and Spatial Database Management Systems (SDBMS); this is justified due to its importance and ubiquity in our everyday lives. The Database community has followed the paradigm of extended DBMS and provided inherent spatial functionality in geographical data collections by developing spatial data types, operators and methods for querying, as well as indexing techniques. At the exploration level, multi-dimensional online analytical processing (OLAP) and knowledge discovery in databases (KDD) have attracted excellent results at the spatial domain. In this chapter, we review spatial database management (modeling, indexing, query processing) and exploration aspects (data warehousing and OLAP analysis, data mining), followed by a short discussion on data privacy aspects. This is essential knowledge in order for the reader to get familiar with background terms and notions during the corresponding discussion in the mobility data domain, in the chapters that will follow.

2.1 Spatial Data Modeling

From modeling perspective, in the geographical space, we can recognize a set of basic spatial entities: *points*, *lines*, and *surfaces* or *polygons* (in several variations); see the respective illustrations in Fig. 2.1. These entities are assumed to be 2-dimensional since we are interested in the (x, y) - plane. If one is interested in the 3-dimensional (x, y, z) - space then these entities are still valid, also with the addition of a fourth entity, namely *volume*.

Exploiting on these types of spatial information, let us sketch a spatial database, call it *Countries-and-Cities*, to be used as a running example for the discussion in this section.

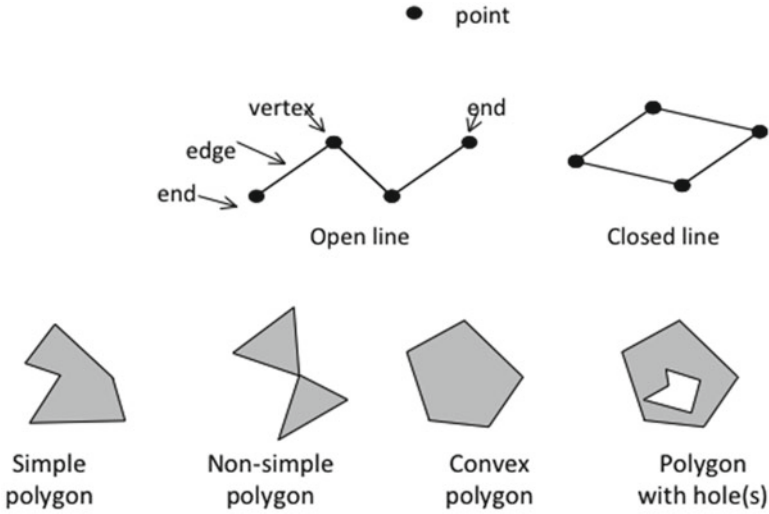


Fig. 2.1 Spatial entities in geographical space

Example 2.1. Countries-and-Cities database. *Countries-and-Cities* database is designed to support a geographical map consisting of countries (name, population, area, shape, etc.) and cities (name, population, shape, etc.). We recognize entities (a country named Greece, a city named Athens), relationships between entities (Athens is capital of Greece), alphanumeric (name of a country) vs. spatial attributes (geometry of a country, of a city), or even functions over spatial attributes of a single entity (the area of a country can be defined over its geometry) or of multiple entities (the length of borderline between two countries can be defined over their two geometries).

The question that arises is how can we model and efficiently manage information such as that of Example 2.1. Considering the different options of spatial entities presented in Fig. 2.1, ‘Country’ could be of polygon shape and ‘City’ could be of point shape. Figure 2.2 illustrates the design of *Countries-and-Cities* database at (a) conceptual and (b) logical level. The design at the conceptual level, which follows the *Extended Entity-Relationship* (EER) model, shows that we recognize two entities (Country, City) and one relationship between entities (capital-of). According to the design at the logical level, which follows the *Object-Relational* (OR) model, the database schema consists of two relations (Country, City).

Between spatial entities, we can define several relationships based on their relative positions, with *directional* and *topological* relationships being the most popular ones. Directional relationships could be absolute with respect to a Cartesian coordinate system (north-of, south-west-of, etc.) or relative with respect to a point of reference or another entity. On the other hand, topological relationships are based on the topology of entities in the geographical space and are invariant to topological transformations (shift, rotation, scaling).

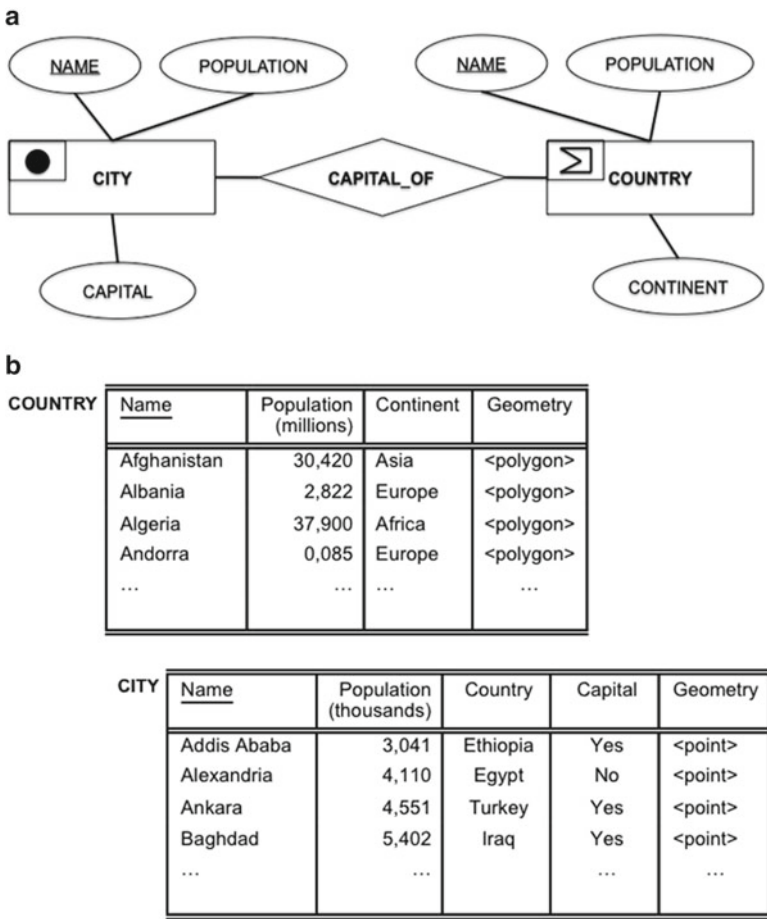


Fig. 2.2 The design of *Countries-and-Cities* database at (a) conceptual and (b) logical level

A quite popular set of topological relationships between regional objects is illustrated in Fig. 2.3 and includes:

- *Disjoint*: two regions A and B are disjoint iff A covers space which has nothing in common with the space covered by B;
- *Contain* (and its reverse, *Inside*): region A contains region B iff B is found inside A and their boundaries share no common space;
- *Cover* (and its reverse, *Covered by*): region A covers region B iff B is found inside A and their boundaries share common space;
- *Overlap*: two regions A and B overlap iff A and B share common space, A also covers some space outside B, and B also covers some space outside A;
- *Meet*: two regions A and B meet iff A and B share common space, which is found only at A's and B's boundaries;
- *Equal*: two regions A and B are equal iff A and B cover exactly the same space.

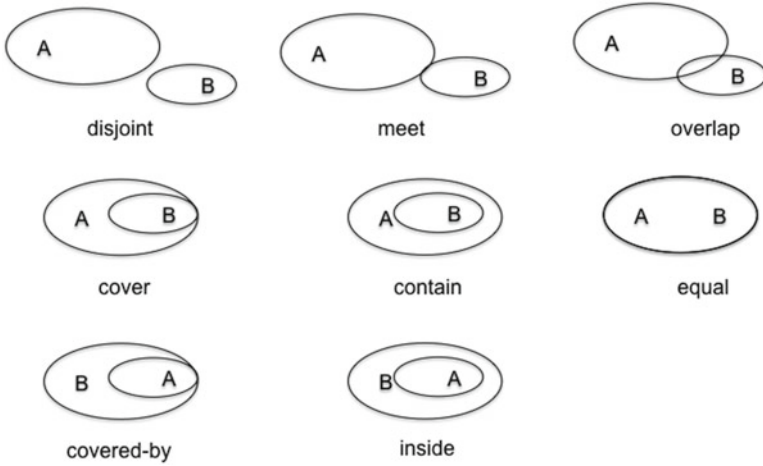


Fig. 2.3 Formalization of topological relationships between two regional objects

Formally, these relationships are defined over the possible combinations of nine set intersections among objects' components, assuming that an object O is defined as a triple $\langle O_{interior}, O_{boundary}, O_{exterior} \rangle$ of sub-spaces partitioning the entire space. As an example, *meet* is defined as the combination:

$$\begin{aligned}
 &A_{interior} \cap B_{interior} = \emptyset, A_{interior} \cap B_{boundary} = \emptyset, A_{interior} \cap B_{exterior} \neq \emptyset \\
 &A_{boundary} \cap B_{interior} = \emptyset, A_{boundary} \cap B_{boundary} \neq \emptyset, A_{boundary} \cap B_{exterior} \neq \emptyset \\
 &A_{exterior} \cap B_{interior} \neq \emptyset, A_{exterior} \cap B_{boundary} \neq \emptyset, A_{exterior} \cap B_{exterior} \neq \emptyset
 \end{aligned}$$

and so on, for the rest of the relationships.

The above relationships assume regional objects, e.g. polygons. In a similar way, a (more narrow) set of relationships can be defined between linear objects (e.g. *intersect*), between one regional and one linear object (e.g. *cross*), between one regional and one point object (e.g. *inside*), and so on. In our running example, Canada and Brazil are *disjoint* whereas US and Mexico *meet*, Buenos Aires is *inside* Argentina, etc.

Queries over spatial databases obviously focus on the geometry of entities. Typical spatial queries are the following:

- *Point query* (input: spatial dataset D , point p): find objects in D that contain p .
- *Range query* (input: spatial dataset D , region r): find objects in D that overlap r .
- *Nearest-neighbor (NN) query* (input: spatial dataset D , point p , number k): find the top- k objects in D that lie nearest to p .
- *Spatial Join query* (input: spatial datasets D_1 and D_2): find pairs (o_1, o_2) of objects in datasets D_1 , D_2 , respectively, that satisfy a spatial condition (usually, overlap).

In our running example, pointing a location on the map we may ask to find the country it belongs to or the city that is located nearest to it; drawing a range on the map, we may ask to find the cities that are located inside it; joining countries and cities we may ask to find the cities that are located within a buffer of 10 km width along countries’ borderlines; etc.

2.2 Spatial Database Management

Modern DBMS have been extended to support spatial databases like *Countries-and-Cities*. Their extensions are based on the Abstract Data Type (ADT) functionality, provided by most DBMS vendors nowadays, which is supported by appropriate operators (methods over data types) and facilitated during query processing with appropriate indexing mechanisms.

2.2.1 Abstract Data Types

In modern DBMS, spatial objects are modeled through the ADT paradigm. For instance, PostgreSQL supports a number of geometric data types—point, line segment, box, path, polygon, circle, etc.—built upon simpler data types (Fig. 2.4).

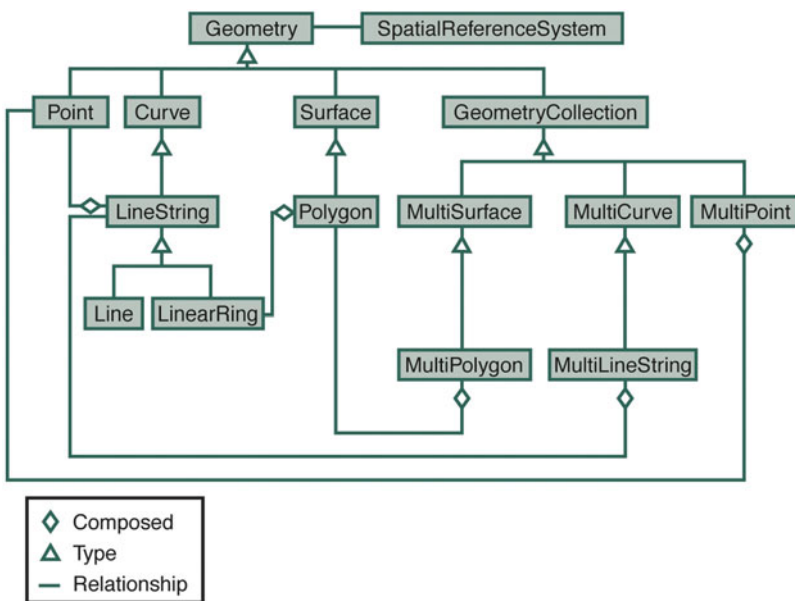


Fig. 2.4 PostgreSQL geometric data types

For instance, *line segment* is defined as a pair of points, *polygon* is defined as a sequence of points, *circle* is defined as a pair of a point and a number (radius), and so on. Moreover, the PostgreSQL ADT mechanism provides a number of methods (to find the number of points in a path, the length of a path, the center of a circle, the distance between two points, whether two boxes overlap or not, whether two line segments are parallel or not, whether a box is left of another box or not, etc.).

Having these data types in our repertoire, we may create PostgreSQL tables for *Countries-and-Cities* database as follows...

```
CREATE TABLE Countries (name varchar(40), population integer, continent
                           varchar(20), geometry polygon);
```

```
CREATE TABLE Cities (name varchar(20), population integer, country varchar(40),
                        capital boolean, geometry point);
```

... and search for entries with respect to alphanumeric (on the conventional attributes) or even spatial criteria (on the geometry attributes). In the following examples, we illustrate the usage of two PostgreSQL geometric operators, namely *contained in* (denoted by symbol '<@') and *distance between* (denoted by symbol '<->'):

```
# Search for cities that lie within a given rectangle
SELECT name
FROM Cities
WHERE (geometry <@ box '(476271, 420400, 479243, 420750)') = TRUE;

# Search for capital cities that lie at a distance less than 50 km from the territory of a
  foreign country
SELECT Cities.name
FROM Cities, Countries
WHERE (circle(Cities.geometry,50000) <-> Countries.geometry) = 0
AND Cities.country <> Countries.name;
```

A natural question that arises from the above discussion is how to organize spatial information in order to efficiently support queries like the above or, in a more general discussion, the ones mentioned earlier (point, range, NN, spatial join). In other words, what kind of indexing and query processing is required when we consider spatial, rather than conventional (alphanumeric) data types?

2.2.2 Indexing and Query Processing Issues

If we consider the geometries of the countries stored in *Countries-and-Cities* database, in fact they are polygons formed by hundreds or thousands of points, depending on the desired resolution. It is also obvious that countries (the same holds for cities) cannot be ordered upon their geometry. Formally, we are not able to define the '<'

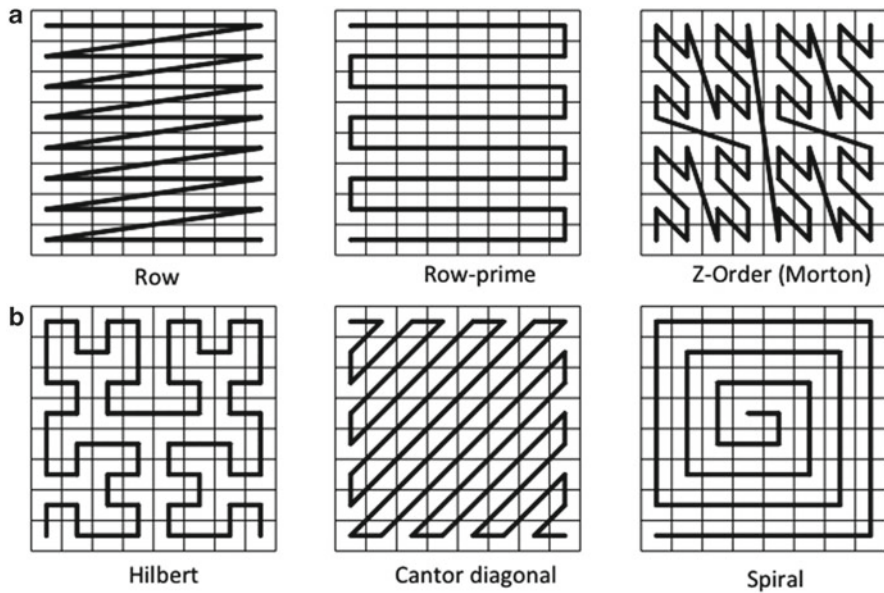


Fig. 2.5 Popular space-filling curves for mapping 2-dimensional objects in 1-dimensional space

operator so that any two entities that lie close in reality also lie close in the ordering provided by this operator, and vice-versa. If it was the case, then we could store the ordering number of each polygon and re-use the (ubiquitous) B⁺-tree for indexing. Actually, this was one of the earliest roadmaps followed for handling spatial data and, in order to achieve it, space-filling curves, such as Hilbert ordering, were used (popular space-filling curves are illustrated in Fig. 2.5). Nevertheless, it appeared quickly that this was not the road to success and native spatial indexes had to be invented instead.

Actually, the key differences between alphanumeric and spatial data is that the latter (a) are of high complexity and (b) lack total ordering. Nevertheless, we can do something on both issues: regarding (a), we may use spatial data approximations of low complexity; regarding (b), we may adopt multi-dimensional search techniques.

Among several spatial data approximations that have been proposed, the *Minimum Bounding Rectangle* (MBR) turned out to be the most popular and widely used. As a definition, $MBR(o)$ is the minimum (hyper-)rectangle, which has its sides parallel to the axes of the adopted coordinate system and entirely covers object o . Figure 2.6 illustrates a few examples of MBR approximations of spatial objects.

Unfortunately, though obviously, as depicted from Fig. 2.6, approximations are not identical to the original shapes they origin from. Of course, this is unavoidable in the vast majority of cases, and raises the need for a *filter-refinement* procedure to support typical spatial queries, such as the ones that were presented earlier.

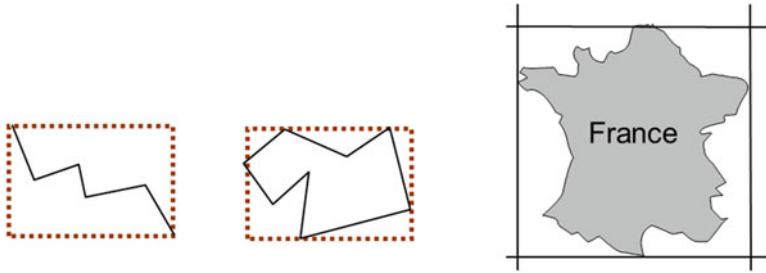


Fig. 2.6 The MBR approximation of three spatial objects: a polyline (*left*), a polygon (*center*), and a country in *Countries-and-Cities* database (*right*)

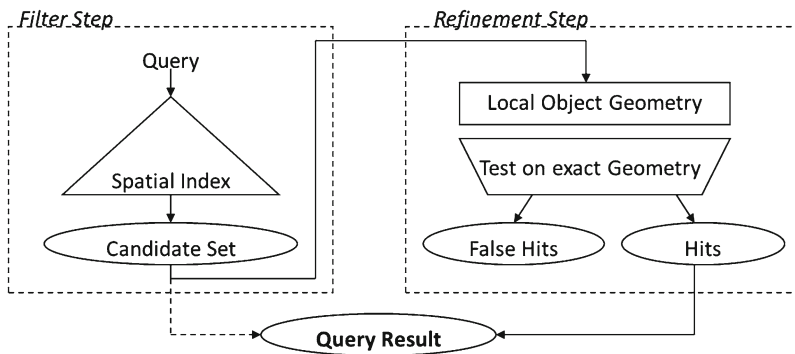


Fig. 2.7 The Filter-Refinement two-step methodology for processing spatial queries

For instance, the MBR of USA *overlaps* with the MBR of Canada but, in fact, the two countries only *meet*, with respect to the terminology of Fig. 2.3.

The Filter-Refinement methodology is a two-step process for processing a spatial query Q over a dataset D :

- *Filter step*: find a set $S' \subset D$ that contains the answer set for query Q using MBR approximations of spatial objects in D (appropriately indexed in a spatial indexing method);
- *Refinement step*: among the candidate answers, find the exact answer set $S \subset S'$ for query Q using the actual geometries of spatial objects in S' .

It is clear that, since the processing of actual geometries (usually based on computational geometry techniques) is much more expensive than the processing of the simple MBR geometry, we prefer to run the expensive part over a subset S' instead of the entire set D of spatial objects. Technically, finding the candidate answer set S' exploits on a spatial index that organizes the MBRs of spatial objects. The flow of Filter-Refinement process is illustrated schematically in Fig. 2.7.

Figure 2.7 infers that in some cases of spatial queries, it is not necessary to run the refinement step and examine the actual geometries of some of the candidate answers (see the dotted line that connects the ‘Candidate Set’ directly with the ‘Query Result’ component). In other words, an object in candidate answer set S' could directly be forwarded to answer set S without the need to examine its actual geometry. Take, for instance, the following situation: in a range query over *Countries-and-Cities* database we ask to find which countries overlap a window we draw on the map; if the MBR of a specific country fully lies inside the window then there is no need to examine the actual geometry; it is for sure that the actual geometry also lies inside the window!

According to the above filter-refinement process, query processing is facilitated by an appropriate indexing method that organizes MBR approximations of spatial objects. As already mentioned, unfortunately, we cannot adopt “total ordering” in multi-dimensional space; if this could have been done, then we could have adopted e.g. B⁺-trees that efficiently organize ordered information (numbers, strings, date values, etc.). Alternative solutions examined in the past to tackle this problem, researched two main directions:

1. adopt a partial ordering mechanism to transform spatial objects into numbers or ranges of numbers and then index them using B⁺-trees, or
2. invent novel spatial indexing techniques, mostly based on the B⁺-tree concept.

As already mentioned, techniques falling under the first category exploited on popular space traversal methods, such as Peano and Hilbert (see Fig. 2.5). However, it was the second category that finally was the winner in this debate, with R-tree being its most popular representative. It is not exaggeration to argue that, nowadays, R-tree is “ubiquitous” in extensible DBMS as B⁺-tree was (and still is) for conventional DBMS applications.

Actually, R-tree extends B⁺-tree ideas to multi-dimensional space. Like B⁺-tree, R-tree is a height-balanced tree with the index records in its leaf nodes containing pointers to the actual data objects and guarantees that the space utilization is at least 50 %. Leaf node entries are in the form (id, MBR) , where id is an identifier that points to the actual object and MBR is the MBR approximation of the actual object. Non-leaf node entries are of the form (ptr, MBR) , where ptr is a pointer to a child node, and MBR is the minimum bounding rectangle covering all child nodes. In a disk-based implementation, a node in the tree corresponds to a disk page and contains between m and M entries. The maximum number M of entries in each node (otherwise, *fanout*) is predefined according to the size of the node (disk page) and the size of each entry. An example of the R-tree indexing method in 2-dimensional space appears in Fig. 2.8:

In the example of Fig. 2.8, 10 data MBRs (r_1, \dots, r_{10}) are organized in 4 leaf nodes (assuming fanout $M=3$), which are, in turn, organized in 2 nodes containing the respective MBRs (R_3, \dots, R_6), which are, in turn, organized in a root node containing the respective MBRs (R_1, R_2), thus forming a tree of height 3.

Propagating down the tree in an appropriate way, we can answer point, range, NN, and spatial join queries very efficiently: in logarithmic cost instead of the linear

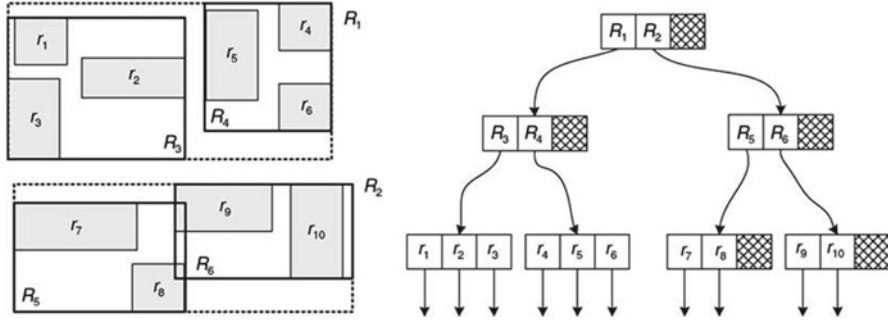


Fig. 2.8 An example of R-tree indexing technique

cost that would be the case if no index existed. For sake of completeness, we sketch the typical R-tree-based algorithms for processing range (point included as a special case), NN, and spatial join queries in spatial databases. This is because these algorithms are used as starting points for plenty of algorithms and techniques on mobility data that will be discussed in the chapters that follow.

R-tree nodes contain entries consisting of a pair (*MBR*, *ptr*) with the MBR approximation of the enclosed geometry and a pointer to the child node (or to the actual geometry, if the entry is at the leaf level), respectively. In range and spatial join query algorithms sketched below, starting from the root node we propagate down the R-tree(s) considering whether the node entries overlap with the geometry of interest (the query window or an entry in the second R-tree, respectively). As such, the cost is order of height of the R-tree, hence logarithmic with the number of entries. On the other hand, unlike B⁺-tree, it is not unusual the case where more than one paths should be propagated in order to answer spatial queries. For example, if we perform a point query on the R-tree of Fig. 2.8 with the query point *p* be located close to the upper-right corner of *r*₈, then we need to visit both nodes under *R*₅ and *R*₆, although visiting the latter will turn out to be useless.

Algorithm R-tree-based-RangeSearch

Input: R-tree node *N*, query window *Q*

Output: answer set *S*

1. IF *N* is a non-leaf node
2. FOR each entry *e* in *N*
3. IF *e*.*MBR* overlaps *Q*
4. CALL R-tree-based-RangeSearch (*e*.*ptr*, *Q*)
5. ELSE // *N* is a leaf node
6. FOR each entry *e* in *N*
7. IF *e*.*MBR* overlaps *Q*
8. add *e* to *S*

Algorithm R-tree-based-SpatialJoin

Input: R-tree node N1, R-tree node N2 // assumption: R-trees of equal height

Output: answer set S

1. IF N1, N2 are non-leaf nodes
2. FOR each pair of entries (e1, e2) in N1xN2
3. IF e1.MBR overlaps e2.MBR
4. CALL R-tree-based-SpatialJoin (e1.ptr, e2.ptr)
5. ELSE // N1, N2 are leaf nodes
6. FOR each pair of entries (e1, e2) in N1xN2
7. IF e1.MBR overlaps e2.MBR
8. add (e1, e2) to S

In a similar fashion, NN queries are addressed as follows (p being the query point): starting from the root node, all entries are sorted according to the lower bound of their potential distance from p (called, MINDIST), and the entry with the smallest distance is visited first. The process is repeated recursively until we reach the leaf level; there, a candidate answer (NN) is found. During backtracking and in order to improve the answer, we visit only those entries whose MINDIST is smaller than the distance of the already found NN from p .

Algorithm R-tree-based-NNSearch

Input: R-tree node N, query point P

Output: object S

1. nearest = INFINITY
2. IF N is a non-leaf node
3. FOR each entry e in N
4. Add e in BranchList in ascending order according to MINDIST
5. Prune BranchList with respect to MINDIST
6. FOR each entry e in BranchList
7. Prune BranchList with respect to MINDIST
8. CALL R-tree-based-Range-Search (e.ptr, P)
9. ELSE // N is a leaf node
10. FOR each entry e in N
11. IF distance (e.MBR, P) < nearest
12. nearest = distance (e.MBR, P)
13. S = e

For example, if we perform a NN query on the R-tree of Fig. 2.8 with the query point p be located close to the lower-right corner of R_3 , we retrieve R_1 , then R_3 , then r_2 , but it is not enough; we then have to retrieve (at least) R_6 (since its MINDIST is smaller than the distance between p and r_2), and, surprisingly, the correct answer is found there (it is r_9)! Also, please take into consideration that if we are searching for the actual geometry (instead of the MBR) that lies nearest to the query point, the algorithm should be aware of it and perhaps provide more than one candidate answers to be passed to the refinement step.

2.3 Spatial Data Warehousing

Exploring spatial data includes spatial data warehousing for OLAP analysis purposes and spatial data mining for extracting hidden knowledge. Data Warehousing (DW) has been widely investigated for conventional, non-spatial data. According to a popular definition, *a Data Warehouse is a subject-oriented, integrated, time-variable, non-volatile information system aiming at decision-making*. To rephrase the above definition, DW plays the role of a database that integrates historical information from other sources (operational databases, etc.) according to a subject, and upon the information stored in this database, multi-dimensional analysis is performed aiming at decision-making (the term OLAP, for *On Line Analytical Processing*, is widely used for this kind of analysis). As such, it is rather static database and, only periodically, it is refreshed by looking into updated sources through an ETL (for *Extract—Transform—Load*) process.

A typical DW architecture is illustrated in Fig. 2.9a: data sources (mainly, operational databases) feed the DW using an ETL (and, periodically, *Refresh*) process; the DW is also accompanied by metadata information; using an OLAP engine, different (subject-oriented) data cubes are built upon the data stored in the DW; operations upon the data cubes include multi-dimensional OLAP analysis, cross-tab querying and reporting, and, more advanced, data mining processing.

Technically speaking, a data cube is a view over the database stored in DW. Its typical structure (in a relational model terminology) consists of a centrally located *fact table* with measures and foreign keys to *dimension tables*, the so-called *star schema*. Figure 2.9b illustrates such an example for a conventional database (e.g. retail): the data cube is three-dimensional, where Time, Product and Location are the three dimensions, whereas quantity and turnover are the two measures to be analyzed. Other, more complex data cube designs include the so-called *snowflake* and *constellation* schemas; in the former, each dimension may be represented by joins over more than one tables, while in the latter, more than one fact tables are present, sharing common dimension tables.

Having such information in their hands, business analysts may perform multi-dimensional analysis over the measures and dimensions defined in the data cube. The most popular OLAP operations include *roll-up* (moving from a finer to a coarser analysis), *drill-down* (the opposite of roll-up), *slice* (performing a filtering according to a condition), and *cross-over* (looking back to the detailed information that supports the OLAP result). An example series of analyses/operations on the data cube that appears in Fig. 2.9b is the following:

- *Find the total turnover per month and per city*, a roll-up operation;
- *Especially in March, find the turnover per city*, a slice operation;
- *Especially in March, find the turnover on weekdays vs. weekends*, a drill-down operation.
- *Retrieve the database records that support the above result*, a cross-over operation.

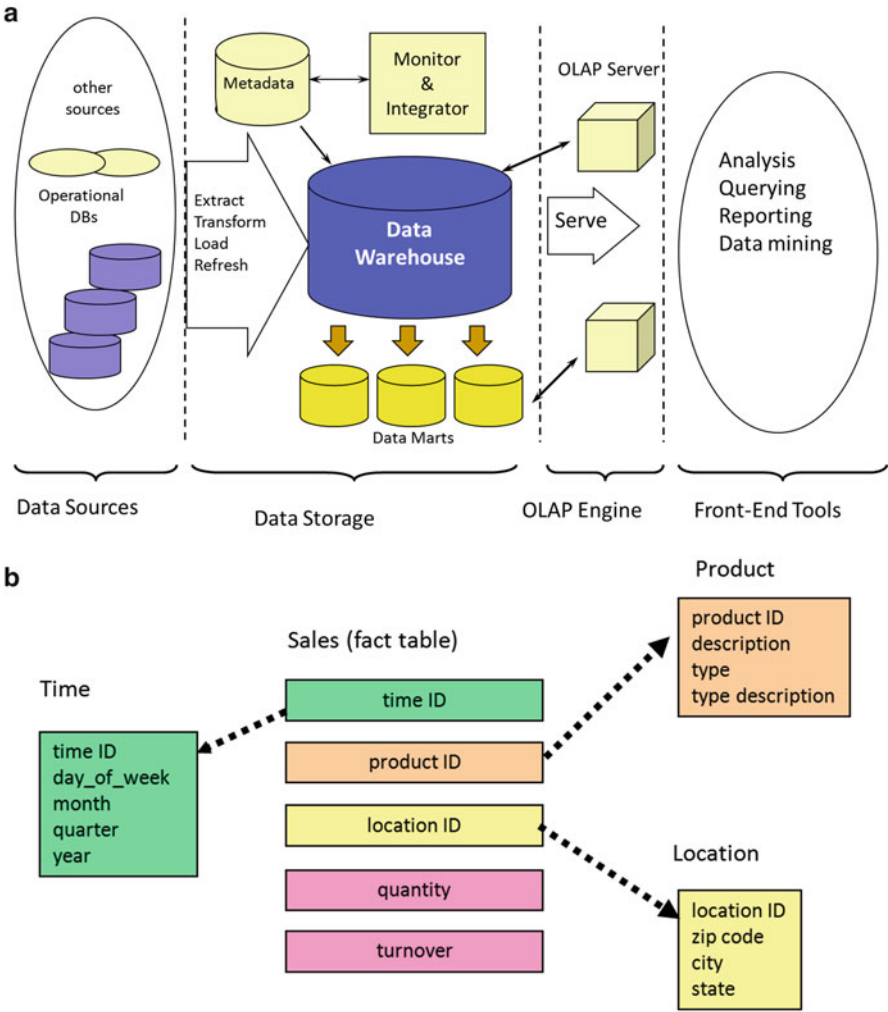


Fig. 2.9 Conventional data warehousing: (a) typical DW architecture; (b) typical data cube following star schema

Adding spatial dimensions and measures in the above paradigm results in the so-called *Spatial DW*. In correspondence to the traditional paradigm, Spatial DW include:

- At the data cube design level: spatial measures (in the fact table), spatial dimensions and spatial hierarchies (in the dimension tables);
- At the OLAP analysis level: spatial OLAP operations, i.e. operations like roll-up and slice extended to spatial predicates;
- At the data cube implementation level: appropriate indexes and efficient query processing techniques.

As expected, a critical issue in data warehousing and OLAP analysis concerns *aggregation*, which refers to summarizing of the properties of data over particular dimensions of interest. In our case, where we focus on the geographical dimension, aggregation is about e.g. computing the total area of a union of areas. As for relational data, aggregate functions for spatial data are grouped into three categories: distributive, algebraic, and holistic. An aggregate function is called *distributive* if it can be computed in a distributive manner, i.e. the final result can be derived by the partial results, without required to retrieve the actual data that contribute to the partial results; examples of distributive functions in relational data cubes include `sum()`, `count()`, `min()`, and `max()`. An aggregate function is called *algebraic* if it involves a constant number of distributive functions (`avg()` is such an example). On the other hand, an aggregate function is called *holistic* if there does not exist an algebraic function with a constant number of arguments that characterizes the computation, therefore the actual data that contribute to the partial results should be retrieved in order for the overall result to be calculated; typical examples of holistic functions include `median()` and `rank()`.

In the spatial domain, examples of spatial operators according to the above classification can be also defined. For instance, calculating the geometric union and the MBR approximation are distributive, calculating the centroid is algebraic, while calculating the minimum distance is holistic function.

2.4 Spatial Data Mining

In a more advanced analysis than multi-dimensional OLAP over data aggregations, data mining includes tasks that are quite expensive to be processed but, on the other hand, uncover the knowledge hidden in the database (hence, Knowledge Discovery in Data—KDD). In general, data mining tasks can be classified as:

- *Clustering*: determining a finite set of implicit classes that describe the data.
- *Classification*: finding rules to assign data items to pre-existing classes.
- *Dependency analysis*: finding rules to predict the value of an attribute on the basis of the values of other attributes.
- *Deviation and outlier analysis*: searching for data items that exhibit unexpected deviations or differences from some norm.
- *Trend detection*: finding lines and curves to data to summarize the database.
- *Generalization and characterization*: obtaining a compact description of the database, for example, as a relatively small set of logical statements that condense the information in the database.

All the above types are also applicable in *spatial data mining*. In the remainder of the section we will focus on cluster analysis (also with a note on sampling, which is a very related problem) and dependency analysis in the form of co-location pattern mining.

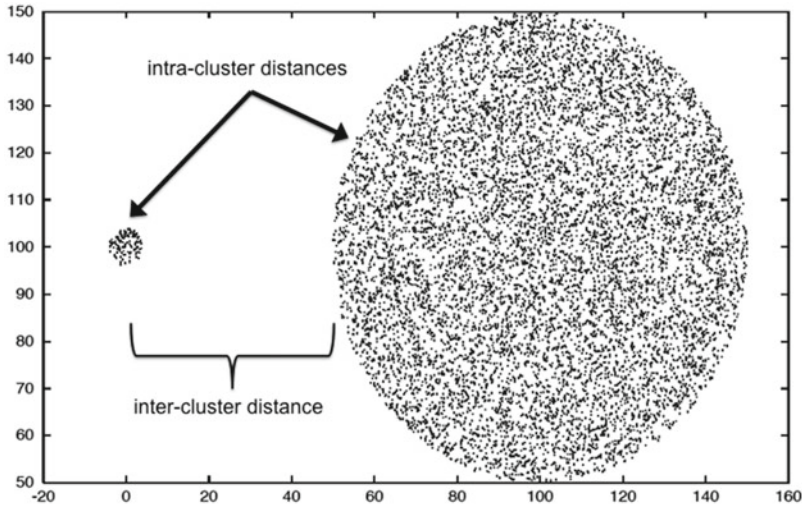


Fig. 2.10 The concepts of intra-cluster and inter-cluster distances

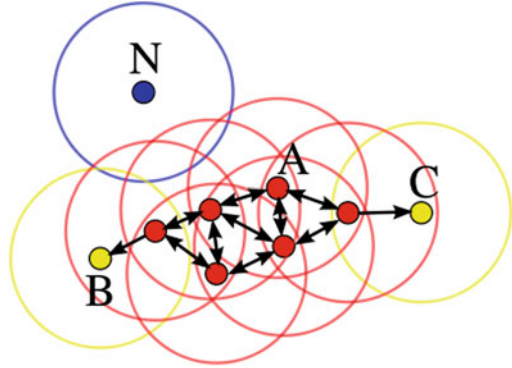
2.4.1 Cluster Analysis

In cluster analysis, the objective is to map a dataset of N spatial objects, $D = \{o_1, \dots, o_N\}$, usually points, into a clustering consisting of K groups (clusters), $C = \{C_1, \dots, C_K\}$, so as similar objects are grouped together in the same cluster and dissimilar objects are grouped in different clusters; similarity between a pair of objects is defined according to a similarity function $Sim(o_i, o_j)$. In other words, the ultimate goal is that *intra-cluster* distances are minimized and *inter-cluster* distances are maximized, whatever is the shape and density of the resulting clusters (Fig. 2.10).

To address this problem, there exist dozens of algorithms; others work in a *hierarchical* fashion, finding all possible groupings from 1 to N clusters while others result in a specific clustering of K clusters; others assume the number K of clusters to be input while others not; others are able to find and isolate outliers while others not. There is also an important classification of clustering algorithms as *partitioning* (where the target is to partition the space in K partitions and each partition corresponds to a cluster), and *density-based* (where clusters are formed according to the spatial density of objects and may take irregular shapes).

As a typical example of conventional data mining technique, K-means, perhaps the most popular clustering algorithm, is a partitioning algorithm aiming to separate the space in K distinct partitions (thus, favoring spherical clusters). It does not distinguish between clustered points and noise, so noise removal should be performed in an earlier data preparation phase. Although intuitive and generic, partitioning algorithms like K-means have some shortcomings. In detail, they are not capable of detecting clusters of arbitrary shape and they are weak in detecting meaningful

Fig. 2.11 Clustering point data with DBSCAN: core (in red; dark grey in b/w) vs. border (in yellow; light grey in b/w) vs. noise points (in blue; black in b/w)



clusters in datasets where the density has big variations. Finally, these techniques have the intrinsic limitation that the user should tune the K parameter, which corresponds to the number of clusters, as well as the fact that they are sensitive to noise (actually they cannot handle it at all).

To overcome these shortcomings, the concept of density-based clustering emerged to propose nice solutions for them. The core idea of density-based clustering algorithms is that clusters are formed in areas of higher density than the rest of the dataset, while points in sparse areas (called noise or border points) separate the clusters among them. This category of algorithms is robust to noise and outliers, while they can discover clusters of various shapes, making them suitable for many real world applications, as those working with mobility data.

The DBSCAN algorithm is one of the most well-known density-based clustering algorithms in the literature. According to DBSCAN, a cluster is a maximal subset of points in the database that are *density-connected*. In order to clarify the notion of density-connected points, we need first to define the concept of *density-reachability*. Given a distance threshold ϵ and an integer $minPts$, a point p_{i+1} is *directly density-reachable* from a point p_i , if p_{i+1} belongs to the ϵ -radius neighborhood of p_i , inside which there are more than $minPts$ points (i.e. as such it is considered as a dense (core) point). A point p_n is said to be *density-reachable* from a point p_1 w.r.t. ϵ and $minPts$, if there exists a chain of points p_1, \dots, p_n such that p_{i+1} is directly density-reachable from p_i , $1 \leq i < n$. The density-reachable relation is not symmetric (i.e. p_1 might not be *density-reachable* from p_n , even if the reverse is valid) as p_n might lie on the edge of a cluster, having in its ϵ -radius neighborhood less than $minPts$ points. This asymmetry is repaired by the concept of *density-connectivity*. More specifically, two points p_1 and p_n are *density-connected*, if there is a point p such that both p_1 and p_n are *density-reachable* from p . Now, the density-connected relation is symmetric. In the example presented in Fig. 2.11, points at A are core points, points B and C are density-reachable from A (though not core, hence border points), whereas point N is neither core nor density-reachable from a core, hence noise.

DBSCAN starts with an arbitrary point that has not been visited. If the ϵ -radius neighborhood of the selected point contains at least *minPts* points, a cluster is initiated, otherwise the point is labeled as noise. If a point is a dense part of a cluster, its ϵ -radius neighborhood is also merged to that cluster. This process continues until no further density-connected points can be found. In such a case, a new unvisited point is selected, which either initiates another cluster or it is considered noise.

Algorithm DBSCAN

Input: dataset D, neighborhood distance eps, population threshold MinPts

Output: clusters C and noise N

```

1.  C = 0
2.  FOR each unvisited point P in dataset D
3.      mark P as visited
4.      NeighborPts = regionQuery(P, eps)
5.      IF sizeof(NeighborPts) < MinPts
6.          add P to noise N
7.      ELSE
8.          C = next cluster
9.          expandCluster(P, NeighborPts, C, eps, MinPts)

expandCluster(P, NeighborPts, C, eps, MinPts)

1.  add P to cluster C
2.  FOR each point P' in NeighborPts
3.      IF P' is not visited
4.          mark P' as visited
5.          NeighborPts' = regionQuery(P', eps)
6.          IF sizeof(NeighborPts') >= MinPts
7.              NeighborPts = NeighborPts joined with NeighborPts'
8.          IF P' is not yet member of any cluster
9.              add P' to cluster C

regionQuery(P, eps)

10. RETURN all points within P's eps-neighborhood (including P)

```

A weak point of DBSCAN is that it is unable to discover clusters of varying density. Its successor, OPTICS solved this shortcoming, by also making one of the two parameters, i.e. the ϵ distance threshold, not necessary, as the user may easily set it by a maximum value implying the *maximum search radius*. According to OPTICS, objects are linearly ordered based on their spatial closeness. Additionally, for each point, a distance measure (called *reachability distance*) that represents the required density of this point to be accepted to belong to the same cluster with its neighbor is stored. The *reachability distance* of point p_1 with respect to point p_2 is actually the smallest distance, such that p_1 is directly density-reachable from p_2 , provided that p_2 is a core point, otherwise it is undefined. The final outcome of the OPTICS algorithm is a so-called *reachability plot* in which the ordering of the points forms the

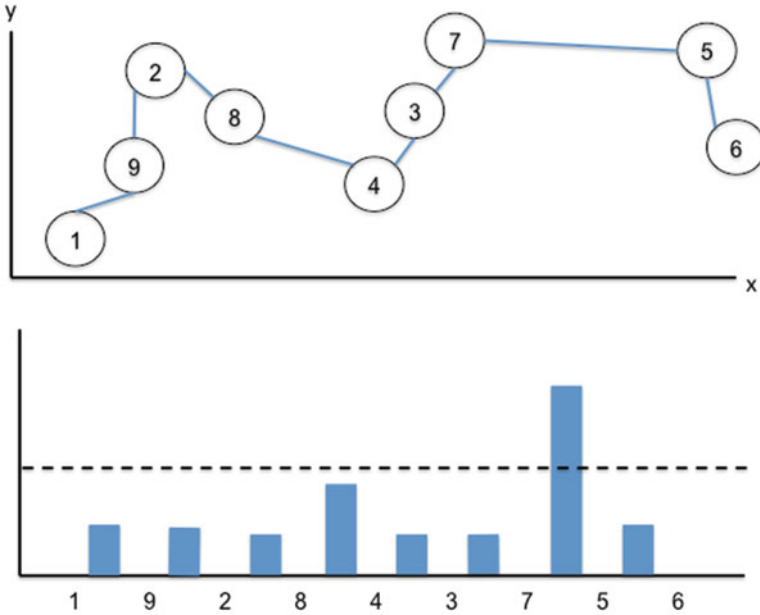


Fig. 2.12 Clustering point data with OPTICS: a set of spatial objects traversed by their pair-wise proximity (*top*) and the resulting reachability plot (*bottom*)

x-axis and the y-axis is the reachability distance of the corresponding point. This plot may be considered as a special kind of *dendrogram*, which enables the derivation of the formed clusters. Intuitively, objects within the same cluster have small values of *reachability distance*, therefore correspond to valleys in the reachability plot. Obviously, the deeper a valley is, the denser the corresponding cluster.

Figure 2.12 illustrates an example of OPTICS clustering. Starting from an arbitrary object, each one finds its nearest-neighbor among the non-visited ones (top of Fig. 2.12) and this traversal results in a reachability plot (bottom of Fig. 2.12). The dashed bar on the plot decides the grouping of objects in clusters, according the valleys produced (two in our example).

Very related to clustering is the *sampling* problem, where the goal is to select a representative subset from a large population so as the ‘properties’ (distribution in space, classes, patterns, etc.) of the original set are maintained in the sample. Effective sampling in spatial data should definitely take the density information into consideration. Consider, for instance, the two clusters illustrated in Fig. 2.10 above. The rightmost consists of a much higher number of points than the leftmost. On the other hand, the density (i.e. the number of points in each cluster with respect to its area) of the small cluster is larger than that of the large one. Uniform sampling cannot maintain this property, and the large cluster will be over-represented in the sample. Moreover, by uniform sampling a very small cluster is in dangerous not to be represented at all. To prevent these problems, density-biased algorithms favor dense clusters in order for them to participate with an adequate number of points in the sample.

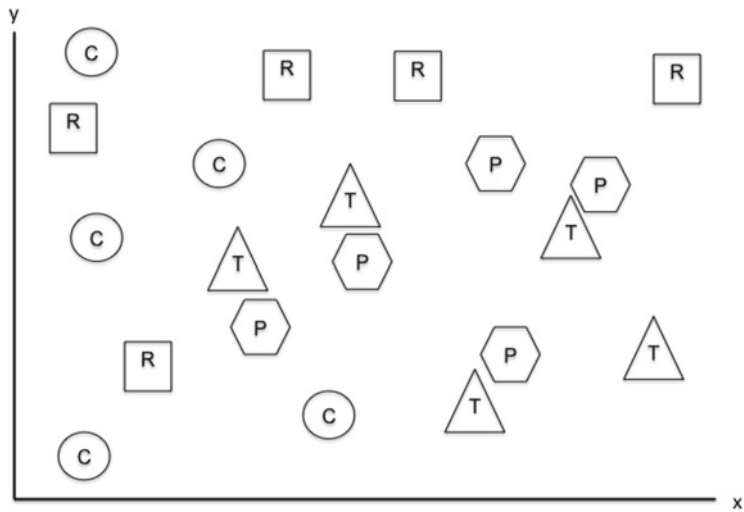


Fig. 2.13 Example of co-location pattern mining: co-location rules “ $T \rightarrow P$ ” and “ $P \rightarrow T$ ” appear frequently

2.4.2 Co-Location Pattern Mining

In spatial co-location pattern mining, the objective is to find co-location rules in a spatial database. For example, in an animal ecology database that stores locations of wild animals in the jungle, a co-location rule “lion \rightarrow tiger” indicates that wherever a lion appears, a tiger also appears in lion’s neighborhood with high probability. Formally, given a spatial dataset consisting of tuples $\langle id, ft, loc \rangle$, where id is the tuple identifier, ft is the feature type, and loc is the location, and a neighbor relation R over locations, the goal is to find co-location rules “ $X \rightarrow Y$ ” that are found in the dataset with high (above a threshold) probability.

Figure 2.13 illustrates an example of co-location pattern mining. Assume a population of circles (C), rectangles (R), triangles (T), and pentagons (P): it is clear that triangles and pentagons are co-located with high probability, which results in co-location rules “ $T \rightarrow P$ ” and “ $P \rightarrow T$ ”.

2.5 Data Privacy Aspects

The respect and protection of human dignity, secrecy and free growth of personality, constitute fundamental pursuits of every democratic society. Among human rights, *privacy* is perhaps the most difficult to define. Privacy can be interpreted as *the personal right of individual to choose freely how she will manage her personal information*.

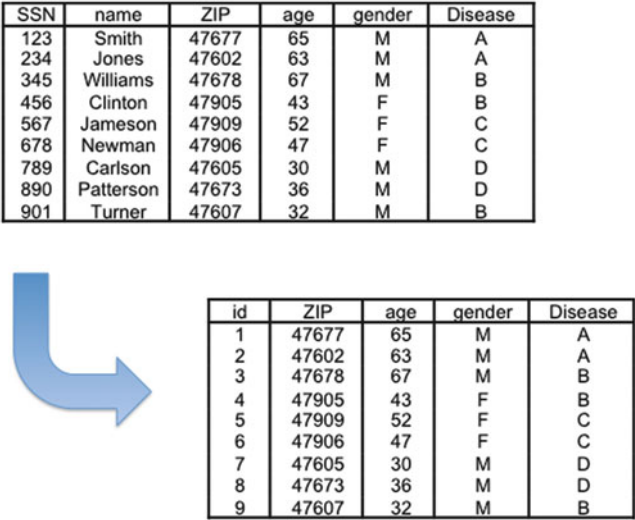


Fig. 2.14 A hospital’s dataset: original in-house (*top*) vs. sanitized published version (*bottom*)

With the rapid growth of technology, privacy aspects came in the light. Nowadays, our everyday actions leave traces (shopping transactions with credit and loyalty cards, electronic administrative transactions, health records, etc.). Several public and private bodies are by law obliged to publish available datasets for analysis purposes. A naïve approach of maintaining individuals’ privacy suggests that by only removing the obvious identifiers such as identity, privacy is preserved.

However, linking data from various sources might lead to draw inferences, which are not possible to conclude from a single source. Consequently, when data holders remove identifiers only, they cannot actually guarantee the anonymity of entities whose data undergoing public release. The set of attributes that each one is not specific enough to identify individuals but through their combination re-identification of an individual can be accomplished, are called *quasi-identifiers*. Quasi-identifiers contain data, such as ZIP code, birthdate, gender, etc. Whereas, attributes that contain information that individuals want to keep secret, are called *sensitive attributes*.

Consider the following example, in which a hospital collects medical and demographic data about its patients. Should for research purposes this dataset be analyzed, the hospital is obliged to provide a “sanitized version” of it (by removing identifiers, such as name and SSN). In this way, no one is (or should be) able to identify that a patient suffers from a specific kind of illness. Figure 2.14 illustrates the dataset that the hospital finally releases. In this table, {ZIP, age, gender} are the quasi-identifiers while Disease is the sensitive attribute.

The “privacy breach” is clear in the above example. Assume that an adversary knows that Mr. Smith, aged 65 and living in ZIP 47677, is included in the dataset, then she may infer with 100 % confidence that Mr. Smith suffers from disease “A” (since there is only one record with these quasi-identifier values).

In order to protect individuals’ privacy from attacks of this kind, the *K-anonymity* principle requires that, in the released dataset, the individuals should be indistinguishable with respect to the set of quasi-identifiers. Formally, a dataset is considered to be *K-anonymous* when for any given set of quasi-identifiers, a record is indistinguishable from $K-1$ other records. The set of records satisfying *K-anonymity* requirement, is called *equivalence class*. *K-anonymity* principle guarantees that an adversary cannot associate a particular record with a specific individual with probability greater than $1/K$, even she: (a) has gained access to the anonymized dataset, (b) is informed that a given individual is included in the dataset, and (c) is aware of the corresponding set of quasi-identifiers.

K-anonymization may be achieved through generalization, suppression or perturbation of quasi-identifier values. More specifically, *generalization* replaces original with higher-level values, taking conceptual hierarchies into consideration (where lower levels contain more information than higher levels). Note that as soon as a value is decided to be generalized, all values of this attribute are generalized, accordingly. Alternatively, *suppression* is performed by hiding the values of the attributes that do not appear in the dataset at least K times. Finally, *perturbation* replaces the actual value with a random value from the standard distribution of values for that attribute. The overall distribution of values is not affected but the individual data values are not correct. The objective of all these techniques is to provide a *K-anonymized* version of the dataset with the minimum deviation from the original.

Figure 2.15 is the *K-anonymized* version ($K=3, 4$) of the dataset illustrated in Fig. 2.14, where the generalization method has been adopted. For instance, gender values are generalized to ‘undefined’ while ZIP and age values are generalized to ranges. Note that in the *K-anonymous* version, every tuple belongs to an equivalence class together with at least $K-1$ other tuples. As such, continuing the previous example, the adversary now cannot infer Mr. Smith’s disease, since at least K tuples exist with the same quasi-identifier values as of Mr. Smith. Moreover, it is obvious that the larger the K , the higher the deviation from the original information.

Although intuitive, *K-anonymity* may not be able to protect sensitive values when their diversity is low. More particular, it suffers from two types of attacks: *homogeneity* and *background knowledge* attack. The former corresponds to cases where a sensitive value occurs repeatedly in a set of tuples with the same quasi-identifier values after the anonymization of the dataset; the latter involves extra information available to an attacker, which may allow to an attacker to increase probability of being able to determine sensitive information about an individual. In the example illustrated in Fig. 2.14 (top), if the disease of id=3 was “A” instead of “B” then an adversary would infer that Mr. Smith’s disease is “A” (homogeneity attack). On the other hand, if an adversary knows that Mr. Smith has very low

id	ZIP	age	gender	Disease
1	476**	60-79	M	A
2	476**	60-79	M	A
3	476**	60-79	M	B
4	479**	40-59	F	B
5	479**	40-59	F	C
6	479**	40-59	F	C
7	476**	20-39	M	D
8	476**	20-39	M	D
9	476**	20-39	M	B

($K = 3$)

id	ZIP	age	gender	Disease
1	47*	50-99	*	A
2	47*	50-99	*	A
3	47*	50-99	*	B
5	47*	50-99	*	C
4	47*	00-49	*	B
6	47*	00-49	*	C
7	47*	00-49	*	D
8	47*	00-49	*	D
9	47*	00-49	*	B

($K = 4$)

Fig. 2.15 The K -anonymous version of the hospital's dataset: $K=3$ (top); $K=4$ (bottom)

probability of having disease “B” then she could infer that most probably Mr. Smith’s disease is “A” (background knowledge attack).

Finally, in the area of *Privacy Preserving Data Mining* (PPDM), there have appeared methods for modification of data before publishing, in order for sensitive information to be protected. Such techniques include *perturbation* (altering an attribute value by a new value or adding noise), *blocking* (replacing an existing value by e.g. a “?”), *aggregation* or *merging* (replacing a value by the respective of a coarser category with respect to a hierarchy), *swapping* (interchanging values of individual records), and *sampling* (releasing data for only a sample of a population).

2.6 Summary

Spatial information has been studied for years in SDBMS. In this chapter, we provided a brief overview of the topic, from modeling to management and exploration, which by nature is huge and cannot fit in a book chapter. Nevertheless, the presented material is the necessary background knowledge for the discussion that will follow in the next chapters regarding the mobility data domain.

2.7 Exercises

- Ex. 2.1. Develop your own Countries-and-Cities database sketched in Example 2.1. To populate it with real information, you may look at various online resources (Wikipedia.org, Wikimapia.org, OpenStreetMap.org, etc.)
- Ex. 2.2. From Fig. 2.6, it is clear that MBR approximations of spatial objects are not identical to the original shapes they origin from. however, this is not always true; there do exist objects with the property that their MBR is equal to their actual geometry. Find at least three 2-dimensional shapes with this property.
- Ex. 2.3. According to the discussion that followed Fig. 2.7, there exist spatial queries that make the refinement step unnecessary for some of the candidate answers. Could you think of cases of spatial queries that make the refinement step completely unnecessary, i.e. the candidate answer set S' is by definition equal to the actual answer set S ?
- Ex. 2.4. Using the example R-tree of Fig. 2.8, perform (1) a point query where the reference point p is located in the center of R_1 , (2) a range query where the query window r is equal to the intersection $R_5 \cap R_6$, and (3) a 1-NN query where the reference point p is located at the lower-right corner of R_2 . What are the lessons learned?
- Ex. 2.5. Consider the example data cube illustrated in Fig. 2.9b: (a) extend it to spatial data cube by adding a ‘geometry’ attribute in the spatial dimension; (b) over the ‘geometry’ attribute, define the aggregate functions “minimum bounding box”, “region area”, “region perimeter”, “center of gravity” and argue for each of them whether it is distributive, algebraic, or holistic.
- Ex. 2.6. According to the discussion about OPTICS, the reachability plot can be used to produce a dendrogram that provides clustering alternative, from a single cluster including all entities to a cluster per entity. Display the dendrogram that corresponds to the reachability plot illustrated in Fig. 2.12 (bottom).
- Ex. 2.7. In the dataset of cities you populated for the purposes of Ex. 2.1, find groups and outliers using DBSCAN and OPTICS. What are the lessons learned? For instance, how did you manage to set appropriate values in the parameters of the algorithms?
- Ex. 2.8. From the Countries-and-Cities database developed in Ex. 2.1, extract two point datasets: locations of capital cities (CI) and locations of countries’ centroids (CO). Does a co-location rule “CI \rightarrow CO” (and vice versa) make sense? In other words, are capital cities usually located centrally in their countries or not?
- Ex. 2.9. Discuss the shortcomings of K -anonymity principle, using Figs. 2.14 and 2.15 as the running example. If you were not aware of the quasi-identifier attributes, which tasks would you perform in order to recognize them? How would you proceed in order to identify a “good” value for K ?

2.8 Bibliographical Notes

Shekhar and Chawla (2003) is an excellent textbook on spatial databases. The topological relationships discussed in Sect. 2.1 were proposed by Egenhofer and his colleagues back in late '80s and early '90s (Egenhofer 1989; Egenhofer and Herring 1991). The so-called 4- and 9-intersection models are based on the set intersections between objects' interior, boundary (and exterior only for the latter model). Nowadays, they have been adopted by most commercial DBMS in their spatial extensions.

Regarding industrial support to spatial databases, well-known vendors provide it through data types and functions according to the OGC standards. More resources can be found at vendors' web sites, like e.g. in Microsoft (2013), MySQL (2013), Oracle (2013), PostGIS (2013).

The Filter-Refinement methodology for spatial query processing, presented in Sect. 2.2.2, was proposed in Orenstein (1986). R-tree, discussed in Sect. 2.2.2, is the most popular indexing method in spatial databases; the original structure, together with efficient algorithms for index maintenance (insertions, deletions) and querying (point and range queries), was proposed in Guttman (1984). Among dozens of R-tree variations, the most popular include the R⁺-tree (Sellis et al. 1987) and the R*-tree (Beckmann et al. 1990) while Hilbert R-tree (Kamel and Faloutsos 1994) imposes linear ordering on spatial data (actually, their MBRs) using the Hilbert space-filling curve. Manolopoulos et al. (2005) lists a number of 70 access methods belonging to the R-tree family within a 20 years period (1984–2004). The “ubiquity” of the R-tree is also argued there, with the applications that it has been adopted covering from spatial, image, multimedia, and time-series databases to multi-dimensional (OLAP) analysis. R-tree family seems never ending: indicatively, (Sharifzadeh and Shahabi 2010) recently proposed the VoR-tree, an R-tree variant using Voronoi diagrams for the efficient processing of NN queries.

Speaking about NN queries, the depth-first NN query processing algorithm sketched in Sect. 2.2.2 was originally proposed in Roussopoulos et al. (1995) and then updated by Cheung and Fu (1998), while a best-first alternative was proposed in Hjaltason and Samet (1999). Those works were followed by an extensive list of NN query variations, including *closest-pairs* (Corral et al. 2000), *constrained NN* (Ferhatosmanoglou et al 2001), NN queries under network constraints (Jensen et al. 2003), *all-NN* (Zhang et al. 2004), and *nearest surrounder* queries (Lee et al. 2010). Efficient spatial join query processing using R-trees was proposed in Brinkhoff et al. (1993). Extensions to multi-way joins were studied in Papadias et al. (1999, 2001a), Mamoulis and Papadias (2001). Incremental distance join, where the results are reported one by one ordered by distance, was studied in Hjaltason and Samet (1998). Methods for efficient processing of topological and direction relations in R-tree supported spatial databases were proposed in Papadias et al. (1995), Papadias and Theodoridis (1997).

The performance of R-trees under various types of spatial queries is evaluated, among other works, in Papadopoulos and Manolopoulos (1997), Theodoridis et al.

(2000), Papadias et al. (2001a), Corral et al. (2006). In Manolopoulos et al. (2005), the interested reader may find plenty of R-tree based algorithms for simple (range, NN, etc.) as well as more complex queries (reverse NN, multi-way spatial join, closest-pair, all-NN, etc.)

The definition about data warehouses that appears in Sect. 2.3 is due to Inmon (1992). Data cubes and the classification of aggregation functions in distributive, algebraic, and holistic were proposed in Gray et al. (1997). Spatial data cubes were proposed in Han et al. (1998). The transition from conventional to spatial data warehouses is discussed in Malinowski and Zimányi (2008). R-trees have been used for OLAP purposes as well: the aggregate R-tree (aR-tree), proposed in Papadias et al. (2001b), processes aggregate queries in spatial data warehouses.

The classification of data mining tasks that appears in Sect. 2.4 is due to Fayyad et al. (1996), also discussed in Andrienko et al. (2008). Tan et al. (2005) is an excellent introductory textbook to data mining, whereas Shekhar et al. (2010) is a recent survey on spatial data mining. Regarding the clustering algorithms mentioned in Sect. 2.4.1, K-means was originally proposed in MacQueen (1967) and finds implementations in almost every data analysis software under different names. DBSCAN and its extension, OPTICS, were proposed in Ester et al. (1996) and Ankerst et al. (1999), respectively. The pseudocode of the DBSCAN algorithm presented in Sect. 2.4.1 and Fig. 2.11 are taken from the respective lemma in Wikipedia.org. Other efficient clustering techniques include BIRCH (Zhang et al. 1996), which is based on a pre-clustering scheme to find a first set of clusters and then continues in a hierarchical fashion, the grid-based algorithm STING (Wang et al. 1997), CURE (Guha et al. 1998), which is based on sampling and partitioning, and C²P (Nanopoulos et al. 2001), which exploits on closest-pairs search in R-trees. Clustering spatial data in the presence of obstacles is discussed in Tung et al. (2001). Sampling in large spatial datasets has been studied in Kollios et al. (2002), Nanopoulos et al. (2006). Spatial co-location pattern mining has been discussed in Shekhar and Huang (2001).

Regarding relational data anonymity, the K -anonymity principle, presented in Sect. 2.5, was introduced in the seminal paper by Sweeney (2002). To overcome K -anonymity shortcomings, several variations have been proposed, including l -diversity (Machanavajjhala et al. 2006) and t -closeness (Li et al. 2007). In particular, a dataset is said to satisfy l -diversity if each equivalence class has at least l ‘well-represented’ values for the sensitive attribute. Parameter l corresponds to the number of distinct values of a sensitive attribute and defines the degree of protection of the dataset. As the number of l increases, the adversary needs more information in order to extract possible values of sensitive attributes. On the other hand, a dataset is said to satisfy t -closeness if for each equivalence class, the distance between the distribution of a sensitive attribute in the class and the distribution of the attribute in the whole dataset is no more than a threshold t . For further insights into this research area, the reader is referred to the recent survey paper by Fung et al. (2010). Linking data mining with data privacy, PPDM was first discussed in the seminal work by Agrawal and Srikant (2000). The classification of PPDM techniques that is mentioned in Sect. 2.5 is due to Verykios et al. (2004).

References

- Agrawal R, Srikant R (2000) Privacy-preserving data mining. In: Proceedings of SIGMOD
- Andrienko N, Andrienko G, Pelekis N, Spaccapietra S (2008) Basic concepts of movement data. In: Giannotti F, Pedreschi D (eds) *Mobility, data mining and privacy—geographic knowledge discovery*. Springer, New York, pp 15–38
- Ankerst M, Breunig MM, Kriegel HP, Sander J (1999) OPTICS: ordering points to identify the clustering structure. In: Proceedings of SIGMOD
- Beckmann N, Kriegel HP, Schneider R, Seeger B (1990) The R*-tree: an efficient and robust access method for points and rectangles. In: Proceedings of SIGMOD
- Brinkhoff T, Kriegel HP, Seeger B (1993) Efficient processing of spatial joins using R-trees. In: Proceedings of SIGMOD
- Cheung KL, Fu A (1998) Enhanced nearest neighbor search on the R-tree. *SIGMOD Rec* 27(3):16–21
- Corral A, Manolopoulos Y, Theodoridis Y, Vassilakopoulos M (2000) Closest pair queries in spatial databases. In: Proceedings of SIGMOD
- Corral A, Manolopoulos Y, Theodoridis Y, Vassilakopoulos M (2006) Cost models for distance joins queries using R-trees. *Data Knowl Eng* 57(1):1–36
- Egenhofer MJ (1989) A formal definition of binary topological relationships. In: Proceedings of FODO
- Egenhofer MJ, Herring J (1991) Categorizing binary topological relations between regions, lines and points in geographic databases. Technical Report, University of Maine
- Ester M, Kriegel HP, Sander J, Xu X (1996) A density-based algorithm for discovering clusters in large spatial databases with noise. In: Proceedings of KDD
- Fayyad UM, Piatetsky-Shapiro G, Smyth P (1996) From data mining to knowledge discovery: an overview. In: Fayyad UM, Piatetsky-Shapiro G, Smyth P, Uthurusamy R (eds) *Advances in knowledge discovery and data mining*. MIT Press, Cambridge, MA, pp 1–34
- Ferhatosmanoglou H, Stanoi I, Agrawal D, Abbadi A (2001) Constrained nearest neighbor queries. In: Proceedings of SSTO
- Fung B, Wang K, Chen R, Yu P (2010) Privacy-preserving data publishing: a survey of recent developments. *ACM Comput Surv* 42(4):1–55
- Gray J, Chaudhuri S, Bosworth A, Layman A, Reichart D, Venkatrao M, Pellow F, Pirahesh H (1997) Data cube: a relational aggregation operator generalizing group-by, cross-tab, and sub-totals. *Data Min Knowl Disc* 1(1):29–53
- Guha S, Rastogi R, Shim K (1998) CURE: an efficient clustering algorithm for large databases. In: Proceedings of SIGMOD
- Guttman A (1984) R-trees: a dynamic index structure for spatial searching. In: Proceedings of SIGMOD
- Han J, Stefanovic N, Koperski K (1998) Selective materialization: an efficient method for spatial data cube construction. In: Proceedings of PAKDD
- Hjaltason G, Samet H (1998) Incremental distance join algorithms for spatial databases. In: Proceedings of SIGMOD
- Hjaltason G, Samet H (1999) Distance browsing in spatial databases. *ACM Trans Database Syst* 24(2):265–318
- Inmon WH (1992) *Building the data warehouse*. Wiley, New York
- Jensen CS, Kolarv J, Pedersen TB, Timko I (2003) Nearest neighbor queries in road networks. In: Proceedings of GIS
- Kamel I, Faloutsos C (1994) Hilbert R-tree: an improved R-tree using fractals. In: Proceedings of VLDB
- Kollios G, Gunopulos D, Koudas N, Berchtold S (2002) Efficient biased sampling for approximate clustering and outlier detection in large datasets. *IEEE Trans Knowl Data Eng* 15(5):1170–1187
- Lee KCK, Lee WC, Leong HV (2010) Nearest surround queries. *IEEE Trans Knowl Data Eng* 22(10):1444–1458
- Li N, Li T, Venkatasubramanian S (2007) t-closeness: privacy beyond k-anonymity and l-diversity. In: Proceedings of ICDE
- Machanavajjhala A, Gehrke J, Kifer D, Venkatasubramanian M (2006) l-diversity: privacy beyond k-anonymity. In: Proceedings of ICDE

- MacQueen JB (1967) Some methods for classification and analysis of multivariate observations. In: Proceedings of BSMSP
- Malinowski E, Zimányi E (2008) Advanced data warehouse design: from conventional to spatial and temporal applications. Springer, New York
- Mamoulis N, Papadias D (2001) Multiway spatial joins. *ACM Trans Database Syst* 26(4):424–475
- Manolopoulos Y, Nanopoulos A, Papadopoulos AN, Theodoridis Y (2005) R-trees: theory and applications. Springer, New York
- Microsoft (2013) Spatial data (SQL Server). <http://msdn.microsoft.com/en-us/library/bb933790.aspx>
- MySQL (2013) MySQL 5.7 reference manual: 12.18 spatial extensions. <http://dev.mysql.com/doc/refman/5.7/en/spatial-extensions.html>
- Nanopoulos A, Theodoridis Y, Manolopoulos Y (2001) C²P: clustering based on closest pairs. In: Proceedings of VLDB
- Nanopoulos A, Theodoridis Y, Manolopoulos Y (2006) Index-based density biased sampling for clustering applications. *Data Knowl Eng* 57(1):37–63
- Oracle (2013) Oracle spatial developer's guide. <http://www.oracle.com/pls/db112/>
- Orenstein JA (1986) Spatial query processing in an object-oriented database system. In: Proceedings of SIGMOD
- Papadias D, Theodoridis Y (1997) Spatial relations, minimum bounding rectangles, and spatial data structures. *Int J Geogr Inf Sci* 11(2):111–138
- Papadias D, Theodoridis Y, Sellis TK, Egenhofer MJ (1995) Topological relations in the world of minimum bounding rectangles: a study with R-trees. In: Proceedings of SIGMOD
- Papadias D, Mamoulis N, Theodoridis Y (1999) Processing and optimization of multiway spatial joins using R-trees. In: Proceedings of PODS
- Papadias D, Mamoulis N, Theodoridis Y (2001a) Constrained-based processing of multiway spatial joins. *Algorithmica* 30(2):188–215
- Papadias D, Kalnis P, Zhang J, Tao Y (2001b) Efficient OLAP operations in spatial data warehouses. In: Proceedings of SSTD
- Papadopoulos A, Manolopoulos Y (1997) Performance of nearest neighbor queries in R-trees. In: Proceedings of ICDT
- PostGIS (2013) PostGIS 2.0 Manual. <http://postgis.net/docs/manual-2.0/>
- Roussopoulos N, Kelley S, Vincent F (1995) Nearest neighbor queries. In: Proceedings of SIGMOD
- Sellis TK, Roussopoulos N, Faloutsos C (1987) The R+-tree: a dynamic index for multi-dimensional objects. In: Proceedings of VLDB
- Sharifzadeh M, Shahabi C (2010) VoR-tree: R-trees with Voronoi diagrams for efficient processing of spatial nearest neighbor queries. In: Proceedings of VLDB
- Shekhar S, Chawla S (2003) Spatial databases: a tour. Prentice Hall, Upper Saddle River, NJ
- Shekhar S, Huang Y (2001) Discovering spatial co-location patterns. In: Proceedings of SSTD
- Shekhar S, Zhang P, Huang Y (2010) Spatial data mining. In: Maimon O, Rokach L (eds) Data mining and knowledge discovery handbook, 2/e. Springer, New York
- Sweeney L (2002) K-anonymity: a model for protecting privacy. *Int J Uncertain Fuzziness Knowl Based Syst* 10(5):557–570
- Tan PN, Steinbach M, Kumar V (2005) Introduction to data mining. Addison-Wesley, Boston
- Theodoridis Y, Stefanakis E, Sellis TK (2000) Efficient cost models for spatial queries using R-trees. *IEEE Trans Knowl Data Eng* 12(1):19–32
- Tung AKH, Hou J, Han J (2001) Spatial clustering in the presence of obstacles. In: Proceedings of ICDE
- Verykios VS, Bertino E, Fovino IN, Parasiliti Provenza L, Saygin Y, Theodoridis Y (2004) State-of-the-art in privacy preserving data mining. *SIGMOD Rec* 33(1):50–57
- Wang W, Yang J, Muntz R (1997) STING: a statistical information grid approach to spatial data mining. In: Proceedings of VLDB
- Zhang T, Ramakrishnan R, Linvy M (1996) BIRCH: an efficient data clustering method for very large databases. In: Proceedings of SIGMOD
- Zhang J, Mamoulis N, Papadias D, Tao Y (2004) All-nearest-neighbors queries in spatial databases. In: Proceedings of SSDBM

Mobility Data Management and Exploration

Pelekis, N.; Theodoridis, Y.

2014, XV, 300 p. 157 illus., Hardcover

ISBN: 978-1-4939-0391-7