Probabilistic Granule-Based Inside and Nearest Neighbor Queries

Sergio Ilarri[1], Antonio Corral[2], Carlos Bobed[1], and Eduardo Mena[1]

[1] Dept. of Computer Science and Systems Engineering, University of Zaragoza, 50018 Zaragoza, Spain.
{silarri,cbobed,emena}@unizar.es
[2] Dept. of Languages and Computing, University of Almeria, 04120 Almeria, Spain.
acorral@ual.es

Abstract. The development of location-based services and advances in the field of mobile computing have motivated an intensive research effort devoted to the efficient processing of location-dependent queries. In this context, it is usually assumed that location data are expressed at a fine geographic precision. Adding support for location granules means that the user is able to use his/her own terminology for locations (e.g., GPS, cities, states, provinces, etc.), which may have an impact in the semantics of the query, the way the results are presented, and the performance of the query processing.
Along with its advantages, the management of the so-called location granules introduces new challenges for query processing. In this paper, we analyze two popular location-dependent constraints, inside and nearest neighbors, and enhance them with the possibility to specify location granules. In this context, we study the problem that arises when the locations of the objects are subject to some imprecision.

1 Introduction

Nowadays, there is a great interest in mobile computing, motivated by an ever-increasing use of mobile devices, that aims at providing data access anywhere and at anytime. In the mobile computing field, there has been an intensive research effort in *location-based services* (LBS). These services provide value-added by considering the locations of the mobile users in order to offer more customized information.

How to efficiently process continuous location-dependent queries (e.g., tracking the available taxi cabs near a moving user) is one of the greatest challenges in location-based services. Thus, these queries require a continuous monitoring of the locations of relevant moving objects in order to keep the answer up-to-date efficiently. Moreover, even if the set of objects satisfying the query condition does not change, their locations and distances to the user do change continuously, and therefore the answer to the query must be updated with the new location data (e.g., to update the locations of the objects on a map).

Existing works on location-dependent query processing implicitly assume GPS locations for the objects in a scenario (e.g., [1, 2]). However, precise locations may be unavailable or even be inconvenient for the user. Thus, for example,

providing the user the latitude/longitude of the objects in an answer is probably of little use unless this information is combined with some kind of map. For instance, a train tracking application could just need to consider in which city the train currently is, not its exact coordinates. For such applications, it is useful to define the concept of *location granule* (similar to the concept of *place* in [3]) as a set of physical locations. In the previous example, every city would correspond with a location granule. Other examples of location granules could be: freeways, buildings, offices in a building, etc. Notice that managing location granules instead of precise geographic locations could also be interesting for privacy reasons.

The use of location granules to enhance the expressivity of location-dependent queries was first proposed in [4], that also considered the basic aspects of inside constraints with location granules. The idea is that the user should be able to express queries and retrieve results according to the concept of "location" that he/she requires, whether he/she needs to talk in terms of GPS locations (the finest type of location granule possible) or locations at a different *resolution*. As described in that work, the use of location granules can have an impact on: 1) the presentation of results (location granules can be represented by using graphics, text, sounds, etc., depending on the requirements of the user), 2) the semantics of the queries (the user expresses the queries according to his/her own location terminology, and therefore the answers to those queries will depend on the interpretation of location granules), and 3) the performance of the query processing (the location tracking overload is alleviated when coarse location granules, instead of precise GPS locations, are used).

However, the use of location granules calls for the definition of new query processing approaches. The fact that most locations are inherently uncertain increases the difficulty of this task. In this paper, we focus on query processing issues and study in detail how *inside* constraints[3] and *nearest neighbor* constraints can be processed by taking into account the possible uncertainty of the locations managed for the objects involved. Dealing with uncertainty leads us to consider *probabilistic granule-based inside queries* and *probabilistic granule-based nearest neighbor queries* that, as far as we know, have not been considered in the literature. As an example of the first type of query, imagine that we want to monitor police units (e.g., police cars and policemen) that are with a probability of at least 80% within a radius r of the building where a certain suspect is currently located; alternatively, we may want to monitor all the police units that *may* be within that radius and obtain the probability that they are actually within the radius. As an example of the second type of query, let us suppose a group of tourists arriving at an airport and that need five taxi cabs to reach their hotels: they could query about the five taxi cabs that are (most probably) the nearest to their terminal. As opposed to a classical nearest neighbor query without locations granules, this query will return taxis outside the boundaries of

[3] By *inside constraints* we mean constraints that are satisfied by objects located within a certain circular range around a given moving object.

the terminal only if there are less than five taxis available in the terminal (e.g., maybe calling a taxi from a different terminal is more expensive).

The structure of the rest of the paper is as follows. In Section 2, we briefly describe the datatypes and the basic architecture that we consider. In Section 3, we explain the mechanism proposed to process *inside* constraints with location granules. In Section 4, we focus on *nearest* constraints. In Section 5, we describe our approach to deal with uncertainty. In Section 6, we present some related works. Finally, some conclusions and plans for future work appear in Section 7.

2 Datatypes and Basic Architecture

A location granule is composed of one or more geographic areas which identify a set of GPS locations under a common name. For example, *Madrid* is a location granule of type *city*, such that it can be said that a certain car is in Madrid or in the location (x,y), depending on the location granularity required (*city* or *GPS* granularity, respectively).

The datatypes managed in the proposed system are summarized in Table 1. Objects are characterized by an identifier, a location (*loc.x* and *loc.y*), a class, and probably other attributes specific to their class. A location granule has an identifier and is represented by a set of figures (Fs). It provides three main operators: *inGr* (short for *inGranule*) returns a boolean indicating whether a certain GPS location is within the granule, *distGr* (*distanceGranule*) computes the *limits-distance* between the GPS location provided and the granule (defined as the minimum distance to the boundaries of the areas composing the granule), and *distBtwGrs* (*distanceBetweenGranules*) computes the distance between two granules. A location granule mapping has an identifier and is composed by a set of granules. The following main operators are defined for granule mappings: *getGrs* (*getGranules*) returns the subset of granules that contain the given GPS location, *getNGr* (*getNearestGranule*) obtains the nearest granule to the specified GPS location, and finally *getGrsObj* (*getGranulesObject*) and *getNGrObj* (*getNearestGranuleObject*) are similar to the previous two operators but considering an object instead of a GPS location. For operators that need to return a single answer (e.g., *getNGrObj*), if there are several results satisfying the operator (e.g., two granules at the same distance from the object) then one is returned randomly. For brevity, we will use *gr* instead of *getNGr* in the rest of the paper.

It should be noted that several disconnected areas can define a location granule; for example, Spain could be seen as a location granule that consists of its peninsular provinces and its islands. Different granule mappings could be defined over the same geographic area, and the user can choose the most appropriate for his/her context. Moreover, a granule can belong to several mappings at the same time. In this way, granule definitions can be re-used to compose different granule mappings. For instance, if we have a granule mapping $M1$ composed of granules corresponding to the different provinces of Spain and another mapping $M2$ with granules defining regions in Spain, we can build a new granule mapping

Datatype	*Tuple format*	*Operators*
Object (O)	<id, loc, class, otherAttr>	representObject
Location Granule (G)	<id, Fs>	inGr: G x GPS → Boolean distGr: GPS x G → Real distBtwGrs: G x G → Real representGranule
Granule Mapping (M)	<id, granules>	getGrs: M x GPS → $\wp(G)$ getNGr: M x GPS → G getGrsObj: M x O → $\wp(G)$ getNGrObj (gr): M x O → G

Table 1. Datatypes and main operators

where some regions in $M2$ are replaced by the corresponding province granules in $M1$ (because we could want a finer location granularity within those regions).

We focus on the query processing of *location-dependent queries with location granules*; for example, "retrieve the cars that are within 100 miles of the city where car38 is, showing their locations with city granularity" (i.e., indicating the city where each retrieved car is). To process these types of queries, two main architectural elements are considered:

- The *Location Server* is a module of a *Server* computer which handles location data about moving objects and is able to answer *standard SQL-like queries* about them. No assumption is made about the way that this location information is managed (e.g., stored in databases, estimated using predefined trajectories, or pulled on demand from the moving objects themselves).
- The *Query Processor* is a module of the Server able to process *location-dependent queries with location granules* by interacting with the Location Server. This module will consider the granule mappings specified by the user in the query constraints. These mappings may be defined by the user himself/herself (*User Mappings*) or be predefined (*Server Mappings*).

For illustrative purposes, we use an SQL-like syntax to express the queries and constraints, which allows us to emphasize the use of location granules and state the queries concisely. The structure of location-dependent queries is:

SELECT *projections*
FROM *sets-of-objects*
WHERE *boolean-conditions*

where *sets-of-objects* is a list of object classes that identify the kind of objects interesting for the query, *boolean-conditions* is a boolean expression that selects objects from those included in *sets-of-objects* by restricting their attribute values and/or demanding the satisfaction of certain *location-dependent constraints*, and *projections* is the list of attributes or location granules that must be retrieved from the selected objects.

Specifications of granule mappings can appear in the SELECT and/or in the WHERE clause of a query, depending on whether those location granules must be used for the visualization of results or for the processing of constraints, respectively. If no location granule mappings are specified, GPS locations are assumed. In this paper, we only focus on query processing issues[4]. For clarity, we will consider that granule mappings are specified by using the *gr* operator, but using *getGrsObj* is also possible.

3 Processing Inside Constraints with Location Granules

In this section, we explain how *inside* constraints are processed. The general syntax of an *inside* constraint is *inside(r, obj, target)*, which retrieves the objects of a certain class *target* (such objects are called *target objects* and their class the *target class*) within a specific distance r (which is called the *relevant radius*) of a certain moving object *obj* (that is called the *reference object* of the constraint). In this general syntax, the locations of the reference object and the target objects are considered as GPS locations. However, the second and/or the third argument of the inside constraint can specify that the location has to be interpreted according to a certain granule mapping instead, as we will explain in the following. Thus, *obj* can be replaced by *gr(map1, obj)* and *target* can be replaced by *gr(map2, target)*, where *map1* and *map2* are granule mapping identifiers (not necessarily the same one).

As an example, *"SELECT Car.id FROM Car WHERE inside(130 miles, car38, Car)"* is a query that retrieves the cars within 130 miles of *car38*. In this example, the reference object is *car38*, the target class is *Car*, and the relevant radius is *130 miles*. If granules are associated to the *inside* constraint of that query, three cases (with different semantics) can be distinguished, as we will explain in the rest of this section[5].

3.1 Inside constraint with a granule for the reference object

In this case, the corresponding *inside* constraint is interpreted as follows:

$$\textit{inside(r, gr(map, obj), target)} = \{oi \mid (oi \in \textit{target}) \wedge \ (\exists\, p \in GPS \mid \\ \textit{inGr}(\textit{gr}(\textit{map}, \textit{obj}), p) \wedge \textit{distance}(p, (oi.\textit{loc}.x, oi.\textit{loc}.y)) \leq r)\}$$

where *distance* represents the Euclidean distance between two geographic locations. This constraint retrieves the target objects (instances of the class *target*) whose distance from the granule of the reference object *obj* (according to the granule mapping *map*) is not greater than the relevant radius r. As an example, *inside(130 miles, gr(province, car38), Car)* is satisfied by the cars within 130 miles of the province of *car38* (the reference object of the *inside* constraint).

[4] We refer readers interested in the presentation aspect to [4] and the interactive applet at `http://sid.cps.unizar.es/ANTARCTICA/LDQP/granulesRepresentation.html`.

[5] An interactive demonstration showing these different cases is available as a Java applet at `http://sid.cps.unizar.es/ANTARCTICA/LDQP/granules.html`.

To obtain the objects that satisfy an *inside* constraint with a granule for the reference object, the following operations are performed: 1) the granule of the reference object is obtained; 2) the area/s corresponding to such a granule is/are enlarged by the relevant radius in order to obtain the *relevant area/s*; and 3) the target objects within that area are retrieved. The operation corresponding to the second step, which implies computing the Minkowski sum [5] of the area/s composing the granule and a disk with radius the relevant radius, is called *buffering* in the context of Geographic Information Systems [6]:

$$\textit{buffer(r, granule)} = \textit{granule'} \mid (|\textit{granule.Fs}| = |\textit{granule'.Fs}|) \wedge$$
$$\wedge\ (\forall\ \textit{Fi'} \in \textit{granule'.Fs}: \exists\ \textit{Fi} \in \textit{granule.Fs} \mid \textit{Fi'} = \textit{buffer(r, Fi)}), \text{ where:}$$

–$\textit{buffer(r, A)} = \textit{A'} \mid (\forall\ p \in \textit{GPS}: \textit{contains(A, p)} \Longrightarrow \textit{containsF(A', circle(p, r))}$
–$\textit{containsF(area2, area1)} \Longleftrightarrow (\forall\ p \in \textit{GPS}: \textit{contains(area1, p)} \Longrightarrow \textit{contains(area2, p)})$

3.2 Inside constraint with a granule for the target class

An *inside* constraint can include a location granule for the target class, in which case the constraint is interpreted as follows:

$$\textit{inside(r, obj, gr(map, target))} = \{oi \mid (oi \in \textit{target}) \wedge\ (\exists\ p \in GPS \mid$$
$$\textit{inGr}(gr(map, oi), p) \wedge \textit{distance}(p, (\textit{obj.loc.x}, \textit{obj.loc.y})) \leq r)\}$$

That is, the constraint is satisfied by the target objects (instances of the class *target*) located in location granules (defined by the granule mapping *map*) whose boundaries intersect a circle of the relevant radius r centered on the reference object *obj*. As an example, the constraint *inside(130 miles, car38, gr(province, Car))* is satisfied by the cars located in provinces whose boundaries are (totally or partially) within 130 miles of *car38*.

To obtain the objects that satisfy an *inside* constraint with a location granule for the target class, the following operations are performed: 1) a circular area of radius the relevant radius centered on the current GPS location of the reference object is computed; 2) the granules intersected by such an area are determined; and 3) the target objects within any of those granules are obtained.

3.3 Inside constraint with granules for the reference and target

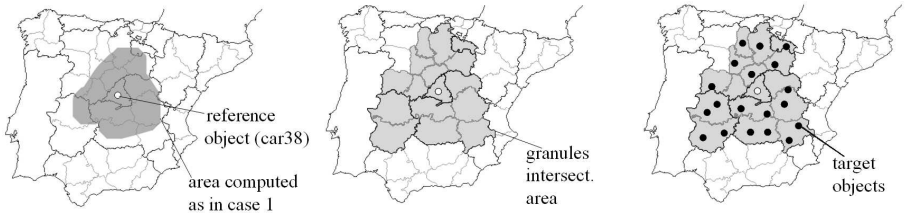
This final situation is a mixture of the two previous cases. The corresponding *inside* constraint is interpreted as follows:

$$\textit{inside(r, gr(map1, obj), gr(map2, target))} =$$
$$\{oi \mid (oi \in \textit{target}) \wedge\ (\exists\ p1 \in GPS, \exists\ p2 \in GPS \mid$$
$$(\textit{distance}(p1, p2) \leq r)\ \wedge\ \textit{inGr}(gr(\textit{map}2, oi), p1)\ \wedge\ \textit{inGr}(gr(\textit{map}1, \textit{obj}), p2)\}$$

That is, this constraint is satisfied by the target objects (instances of the class *target*) located in location granules (defined by the granule mapping *map2*) whose boundaries are within the relevant radius r from the granule of the reference object *obj* (determined according to the granule mapping *map1*). As an

example, the constraint *inside(130 miles, gr(province, car38), gr(province, Car))* is satisfied by the cars located in provinces whose borders are (total or partially) within 130 miles of the province where *car38* is. In this example, the same type of granule *province* is specified for both the reference object and the target class.

In this case, to obtain an answer, an area is computed first by enlarging the granule of the reference object by the relevant radius (see Figure 1.a), exactly as it is done in the steps *1* and *2* in the case explained in Section 3.1 (*case 1*). Then, the set of granules intersected by such an area are determined (see Figure 1.b), similarly to step *2* in the case considered in Section 3.2 (*case 2*). Finally, the objects within those granules are retrieved (see Figure 1.c).



(a) Step 1 (b) Step 2 (c) Step 3

Fig. 1. Inside with granules for the reference object and the target class: steps

4 Processing NN Constraints with Location Granules

In this section, we explain how *nearest neighbor* constraints are processed. The general syntax of a *nearest neighbor* constraint is *nearest(N, obj, target)*, that is satisfied by the N objects of the class *target* that are the nearest ones to the reference object *obj*. The argument N is optional (if not provided, N is assumed to be one). Unless specified otherwise, the locations of the reference object and the target objects are considered as GPS locations. As an example, the query *"SELECT Car.* FROM Car WHERE nearest(car38, Car)"* retrieves the attributes of the nearest car to car38 in terms of geographic distance between GPS coordinates. However, the second and/or the third argument of the *nearest* constraint can specify that the location has to be interpreted according to a certain granule mapping instead. If granule mappings are specified in a query, three cases can be distinguished, as we will discuss in the rest of this section.

4.1 Nearest constraint with a granule for the reference object

In this case, the corresponding *nearest* constraint is interpreted as follows:

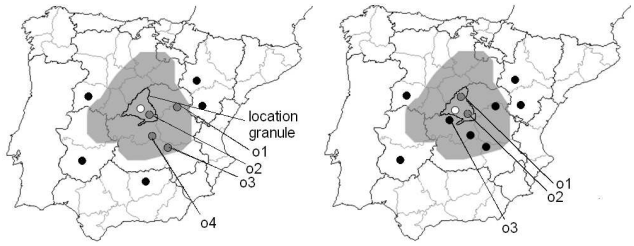
$$\textit{nearest(N, gr(map, obj), target)} = S \mid (|S| = N) \wedge (gr(map, obj) = g) \wedge$$
$$\wedge\ S = \{oi \mid (oi \in \textit{target}) \wedge (\nexists oj \notin S \mid distGr(oj.loc, g) < distGr(oi.loc, g))\}$$

That is, the constraint is satisfied by the N objects of the class *target* that are the nearest to the granule of *obj* (according to the granule mapping *map*). As an example, the constraint *nearest(5, gr(province, car38), Car)* is satisfied by the five nearest cars to the province of *car38*.

To obtain the objects that satisfy a *nearest* constraint with a granule for the reference object, the following operations are performed. First, the granule of the reference object is obtained. Then, the objects of the class *target* within that granule are retrieved; let us assume that the number of objects retrieved is M. All those objects are closer to the granule than any other object outside the granule. If $M \geq N$, then any N of those M objects can be retrieved as satisfying the *nearest* constraint, as all of them have the same distance to the granule (that is, zero). If $M < N$, then we must retrieve $N' = N - M$ additional objects. For this, we inspire by [7], that retrieves the objects within a query sphere which increases iteratively until all the nearest objects required are obtained. In our case, to retrieve additional objects, we apply a buffering operation $buffer(r', gr(map, obj))$ on the granule of the reference object *obj*, as explained in Section 3.1, and retrieve the objects within that enlarged granule. The best choice for the *expansion radius* r' would be the smallest value that allows us to retrieve N' additional objects. Unfortunately, we can only try to guess an appropriate value for r'. If less than N' objects are retrieved, we need to expand r' and repeat the operation until we have retrieved a number of objects $M' \geq N'$. Then, we sort these objects according to their distances to $gr(map, obj)$ and consider the first N' objects to complete the set of objects satisfying the constraint (the order between two objects at the same distance is arbitrary). In practice, the maximum radius r' to consider will be limited, as the user will not be interested in objects further than a certain distance from the reference object [8]. In that case, the number of objects returned may be smaller than N.

We show a couple of examples in Figure 2. In the scenario on the left, we consider a 4-NN query, which would retrieve the four objects $o1$, $o2$, $o3$, and $o4$. One of these objects ($o2$) is within the granule (and therefore at distance zero from the reference object) and the other three are obtained by using the expansion mechanism described above. In the scenario on the right, we consider a 2-NN query which retrieves the objects $o1$ and $o2$. It should be noted that one of the objects retrieved ($o1$) is clearly further in terms of Euclidean distance than one of the objects that is not retrieved ($o3$). For simplicity, we consider the same expansion radius in both scenarios, although it could obviously be reduced for the scenario on the right.

We would like to emphasize that if there are additional standard (i.e., not location-dependent) SQL constraints in the query that affect the target class *target*, those constraints must be verified before selecting the N objects. Otherwise, some of those objects could be discarded later if they do not satisfy such constraints, and the processing would end up with less that the N objects required. For simplicity, and without loss of generality, we will assume in the rest of the paper that no such constraints exist.



(a) (b)

Fig. 2. Nearest neighbor queries in different scenarios: (a) 4-NN and (b) 2-NN

4.2 Nearest constraint with a granule for the target class

In this case, the corresponding *nearest* constraint is interpreted as follows:

$$\begin{gathered} \textit{nearest(N, obj, gr(map, target))} = S \mid (|S| = N) \wedge \\ \wedge\ S = \{oi \mid (oi \in target) \wedge\ (gr(map, oi) = gi) \wedge \\ \wedge\ (\nexists oj \notin S \mid (gr(map, oj) = gj) \wedge\ distGr(obj.loc, gj) < distGr(obj.loc, gi))\} \end{gathered}$$

That is, the constraint is satisfied by the N objects of the class *target* that are in granules that are the closest ones to the geographic location of the reference object *obj*. As an example, the constraint *nearest(5, car38, gr(province, Car))* is satisfied by the five cars that are in the provinces whose boundaries are the closest ones to the current location of *car38*.

To obtain the objects that satisfy a *nearest* constraint with a granule for the target class, we follow an approach similar to the one described for the case where the location granule affects the reference object. However, in the iterative step, instead of a buffering operation with increasing radius, we need to consider a query sphere (centered on the location of *obj*) with increasing radius, as in [7]. The granules intersecting the sphere, and the objects of the class *target* within those granules, are retrieved, similarly to what was explained in Section 3.2 for *inside* constraints. Once we have performed enough iterations to collect at least N objects, we sort them according to the distances of their granules to the location of *obj* and retrieve the N objects with the smallest distances.

4.3 Nearest constraint with granules for the reference and target

In this case, the corresponding *nearest* constraint is interpreted as follows:

$$\begin{gathered} \textit{nearest(N, gr(map1, obj), gr(map2, target))} = S \mid (|S| = N) \wedge\ (gr(map1, obj) = g) \wedge \\ \wedge\ S = \{oi \mid (oi \in target) \wedge\ (gr(map, oi) = gi) \wedge \\ \wedge\ (\nexists\ oj \notin S \mid (gr(map, oj) = gj) \wedge\ distBtwGrs(g, gj) < distBtwGrs(g, gi))\} \end{gathered}$$

That is, the constraint is satisfied by the N objects of the class *target* that are in granules (of the mapping *map*2) that at the closest ones to the granule of the reference object *obj* (according to the mapping *map*1). As an example, the constraint *nearest(5, gr(province, car38), gr(province, Car))* is satisfied by the five cars that are in the provinces whose boundaries are the closest ones to the current province of *car38*. In this example, the same mapping is used for both the reference object and the target class, but this is not required.

To obtain the objects that satisfy a *nearest* constraint with a granule for the reference object and the target class, we apply iteratively buffering operations on the granule of the reference object (according to the mapping *map*1) as described in Section 4.1. In each iteration, we compute the granules of the mapping *map*2 that intersect the obtained buffer and retrieve the objects of the class *target* within. Once we have collected N objects, we stop iterating. Then, we just need to sort the objects according to the distances between their *map*2-based granules and the *map*1-based granule of the reference object, and return the N objects with the smallest distances. The distance between two granules can be computed as the minimum of the distances between the figures composing them[6].

Notice that some optimizations are possible if $map1 = map2$. Thus, for example, we could start by retrieving the objects within the granule of the reference object as explained in Section 4.1. If the mappings to consider for the reference object and the target class are different, then obtaining such objects is of no use.

5 Dealing with Imprecise Locations

In the previous description, there is the implicit assumption that it is possible to obtain the GPS location of an object by querying the corresponding Location Server. However, sometimes this fine-grain location information is not available. This could be due to the imprecision inherent to the location mechanism used to obtain the locations of the moving objects (e.g., if the cell-ID positioning mechanism is used [9], a location can be as imprecise as the size of the cell where it is contained) or due to performance reasons (the tracking cost incurred to keep a GPS location up-to-date is higher than when a coarser location resolution is used). These coarse-grain locations can be considered as special kinds of location granules, which are called *uncertainty-based location granules* in this paper. Two types of uncertainty-based location granules can be identified:

- *Relative uncertainty-based location granules.* This occurs when the geographic location of the object is available but there is a certain degree of uncertainty. For example, if the Location Server knows that the precision of the GPS location of an object is within five meters, then the granule is defined as a circle of a five-meter radius centered on that GPS location. Three sources of uncertainty can be considered by the Location Server to compute this imprecision: the inherent imprecision of the positioning technology (e.g., GPS), the

[6] To compute the distance between two polygons, the algorithm proposed in `http://cgm.cs.mcgill.ca/~orm/mind2p.html` can be used.

latency in communicating the location to the Server, and the threshold used by the location update policy to minimize the tracking cost (e.g., see [10]).
- *Absolute uncertainty-based location granules.* This means that the actual location is within a certain geographic area. For example, let us consider a Bluetooth-based[7] positioning mechanism that allows to determine the room where a person is located within a building. In this example, not only the precision of the positioning mechanism advises the use of such a granularity level, but a more precise location would probably be useless too.

For clarity, in the following we will explain separately the case of uncertainty-based location granules that affect a reference object and the case of granules that affect the target objects (although, obviously, both situations can co-exist).

5.1 Uncertainty-Based Location Granule for a Reference Object

If a location granule mapping has been specified in the query for the reference object and the retrieved location of the reference object is an uncertainty-based location granule, then obtaining a granule according to the specified mapping can return several granules. This is because the uncertainty-based location granule could intersect several granules in the required granule mapping. The unnamed granule whose area is the union of the areas of all such granules must be used as the granule indicating the location of the reference object. Such a granule will be called a *union-based location granule* because it is obtained as the union of several granules defined in the granule mapping specified by the user. At this point, as other works do (see Section 6), we assume that the Location Server provides, along with the location of the reference object, a *probability density function* (*pdf*) that represents the distribution of the probabilities of the possible locations of the object. In this case, we could use that *pdf* to assign to each of the intersecting granules the probability that the object is actually within such a granule[8], by evaluating an integral. Then, three options are possible:

- *If the user is just interested in the most probable answer*, we could just consider the most probable granule for the reference object –the granule with the highest probability p computed– and continue processing the constraint as usual. The answer obtained will be the correct one with probability p.
- *If the user is interested in all the possible answers, tagged with their probability, or in a may/must interpretation of the query*, we could process the constraint several times, one for each granule composing the union-based location granule. In this case, each constraint obtains a set of objects that satisfy the constraint with a certain probability (the probability that the reference object is within the corresponding granule). The union of the answers for each of those constraints would lead to the query interpreted with *may* semantics, whereas the intersection implies a *must* semantics [11].

[7] `http://www.bluetooth.com`

[8] If no *pdf* is available, we could just assume (for example) a uniform distribution and compute the probabilities by considering the percentage of overlapping area.

– *If the user is interested in a may interpretation of the query*, besides the previous approach (less efficient), we can also simply consider the union-based location granule and return all the objects that may be in the answer, independently of the probability that they are part of the answer.

Let us consider now the *case where no granule mapping has been specified for the reference object*. This implies that the uncertainty-based location granule of the reference object has no semantic meaning (i.e., it is not composed of underlying location granules that are in a granule mapping specified by the user). Instead, the user has expressed a constraint relative to the fine-grained geographic location of the reference object and the use of the uncertainty-based location granule is simply forced by the lack of precision in the location available. For the case of an *inside constraint*, we proceed as explained above. However, if we have a *nearest neighbor constraint*, considering such a granule as if it were determined based on a mapping specified by the user would be misleading. As an example, in Figure 3 object $o2$ (instead of $o1$) is more likely to be the closest object to o, both in the left and in the right scenarios. As the uncertainty-based granule of o (or any of its underlying granules) has no meaning for the user, we must weigh the probability that the reference object be at any point within that granule when computing the distance to another object for nearest neighbor processing. Intuitively, if no granule is specified either for the target objects and the uncertainty-based granule of the reference object o is g, then we can statistically estimate the average distance to a target object o_t as follows:

$$distGr(o_t.loc, g) = \int_{\forall p | inGr(g,p)} distance(p, o_t.loc) * Prob(p)$$

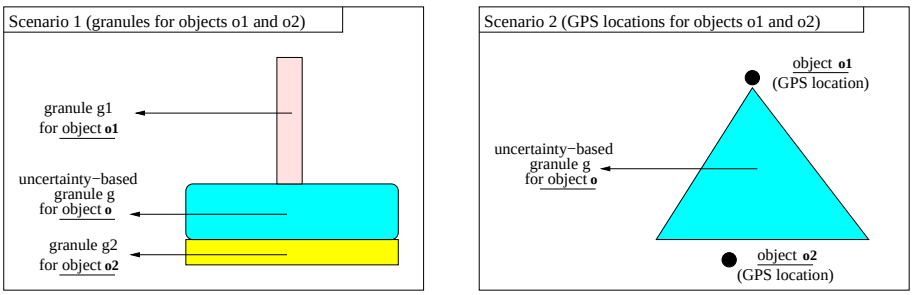


Fig. 3. Uncertainty-based location granules: two sample scenarios

where $Prob(p)$ represents the probability that the reference object is at location p (according to the corresponding probability density function). The previous expression simply tells us that we can compute the expected distance as the average of the possible distances weighted by their probabilities. Similarly, if there is a granule mapping specified for the target class and the target object o_t is in granule g', we could estimate the average distance as:

$$distBtwGrs(g, g') = \int_{\forall p | inGr(g,p)} distGr(p, g') * Prob(p)$$

There are several methods available to compute the previous integrals, as explained in [12]. We have proposed here the computation of an expected distance, as advocated in works such as [12, 13]. An alternative would be to compute the probability that an object o_t is the k_{th} nearest object to the reference object o, by considering the uncertain locations of the other objects as well (e.g., see [14]).

5.2 Uncertainty-Based Location Granule for a Target Object

In this case, for *inside constraints* the mapping should be performed at the level of the Location Servers. Thus, if a Location Server is asked about the moving objects within a certain area and the location of a certain object is an uncertainty-based location granule, then the Location Server will need to compute the intersection of such a location granule and the given area and return that object if the intersection is not empty. In case a granule mapping has been specified for the target class in that constraint, the underlying granules are also returned and can be processed similarly to what was explained in the previous section. If not, as in the previous case, the object can be assigned a probability of being actually within the area indicated.

For *nearest neighbor constraints*, the uncertainty-based location granule for a target object can be managed similarly. First, the Location Server returns the target objects within the area requested (as explained above). Then, if the user was interested in the GPS locations of the target objects (i.e., he/she specified no location granule for the target class), then we can compute distances by weighting the probability that the object is at different locations within the uncertainty-based location granule (as explained in Section 5.1 for the reference object). If, on the contrary, the user specified a granule mapping for the target class, we can use the probabilities assigned to the different underlying granules to compute the possible answers.

6 Related Work

Both inside queries [1, 2] and nearest neighbor queries [15, 16] have been studied in the literature of spatio-temporal and moving object databases (see [17] for a survey of different works in the field). However, existing works on location-dependent query processing implicitly assume GPS locations for the objects in a scenario. Although some works acknowledge the importance of considering different location resolutions (e.g., [3]), the processing of classical constraints such as *inside* or *nearest* is not considered in that context.

The importance of dealing with the uncertainty of location information is emphasized in different works. Probabilistic queries, even though not in the context of moving objects, were introduced in [18]. For moving objects, probabilistic queries are usually computed by estimating the locations of the objects through a probability density function that models the uncertainty, such that the probability that an object is within a certain region can be computed by integration.

As solving these integrals is frequently expensive (numerical methods are usually required), a filter step is introduced to prune the search space. Different relevant proposals exist in the literature. For example, probabilistic range queries are the focus of [19] and probabilistic nearest neighbor queries are studied in works such as [20, 14]. As it is difficult to provide a good overview of contributions in this area in a short space, we refer the interested reader to [17]. No existing proposal has considered probabilistic queries with location granules.

7 Conclusions and Future Work

The expressivity of location-dependent queries can be enhanced by allowing the user to specify the use of location granules. This brings the query to the user's level and may impact not only the query semantics but also the performance and the way the results are presented to the user. In this paper, we have focused on the semantics aspects by studying how *inside* and *nearest neighbor* constraints can be processed when location granules are involved. Moreover, we have proposed solutions to deal with uncertainty in the locations available. Thus, besides the processing of constraints with location granules, an important novelty of this work lies in the combination of location granules with probabilistic approaches.

We are currently carrying out an exhaustive performance evaluation with different real and synthetic granule mappings and locations within the context of the distributed location-dependent query processing system LOQOMOTION [8]. Our preliminary results are promising. As future work, we plan to study other popular location-dependent constraints (such as *closest-pairs* and *similarity joins* [21]) from the perspective of location granules. We will also consider the integration of other approaches to compute the probabilistic distances for nearest neighbor queries (e.g., by considering the proposal in [14]).

Acknowledgements

Work supported by the projects TIN2007-68091-C02-02 and TIN2008-03063.

References

1. Cai, Y., Hua, K.A., Cao, G., Xu, T.: Real-time processing of range-monitoring queries in heterogeneous mobile databases. IEEE Transactions on Mobile Computing **5** (2006) 931–942
2. Gedik, B., Liu, L.: MobiEyes: A distributed location monitoring service using moving location queries. IEEE Transactions on Mobile Computing **5** (2006) 1384–1402
3. Hoareau, C., Satoh, I.: A model checking-based approach for location query processing in pervasive computing environments. In: OTM 2007 Workshops, Vilamoura, Algarve, Portugal. Volume 4806 of LNCS. (2007) 866–875
4. Ilarri, S., Mena, E., Bobed, C.: Processing location-dependent queries with location granules. In: OTM 2007 Workshops, Vilamoura, Algarve, Portugal. Volume 4806 of LNCS. (2007) 856–866

5. Skiena, S.S.: The Algorithm Design Manual. Springer, New York, USA (2008)
6. van Kreveld, M. In: Computational geometry: Its objectives and relation to GIS. Nederlandse Commissie voor Geodesie (NCG) (2006) 1–8
7. Jagadish, H., Ooi, B., Tan, K., Yu, C., Zhang, R.: iDistance: An adaptive B+-tree based indexing method for nearest neighbor search. ACM Transactions on Database Systems **30** (2005) 364–397
8. Ilarri, S., Mena, E., Illarramendi, A.: Location-dependent queries in mobile contexts: Distributed processing using mobile agents. IEEE Transactions on Mobile Computing **5** (2006) 1029–1043
9. Trevisani, E., Vitaletti, A.: Cell-ID location technique, limits and benefits: An experimental study. In: 6th IEEE Workshop on Mobile Computing Systems and Applications (WMCSA'04), English Lake District, UK. (2004) 51–60
10. Wolfson, O., Jiang, L., Sistla, A.P., Chamberlain, S., Rishe, N., Deng, M.: Databases for tracking mobile units in real time. In: 7th International Conf. on Database Theory (ICDT'99), Jerusalem, Israel. (1999) 169–186
11. Sistla, A., Wolfson, O., Chamberlain, S., Dao, S.: Querying the uncertain position of moving objects. In: Temporal Databases: Research and Practice. Volume 1399 of Lecture Notes in Computer Science. (1998) 310–337
12. Xiao, L., Hung, E.: An efficient distance calculation method for uncertain objects. In: Computational Intelligence and Data Mining (CIDM'07), Honolulu, Hawaii, USA. (2007) 10–17
13. Ngai, W.K., Kao, B., Chui, C.K., Cheng, R., Chau, M., Yip, K.Y.: Efficient clustering of uncertain data. In: 6th International Conference on Data Mining (ICDM'06), Hong Kong. (2006) 436–445
14. Beskales, G., Soliman, M.A., IIyas, I.F.: Efficient search for the top-k probable nearest neighbors in uncertain databases. Proceedings of the VLDB Endowment **1** (2008) 326–339
15. Mouratidis, K., Papadias, D., Bakiras, S., Tao, Y.: A threshold-based algorithm for continuous monitoring of k nearest neighbors. IEEE Transactions on Knowledge and Data Engineering **17** (2005) 1451–1464
16. Zheng, B., Xu, J., Lee, W.C., Lee, L.: Grid-partition index: A hybrid method for nearest-neighbor queries in wireless location-based services. The VLDB Journal **15** (2006) 21–39
17. Ilarri, S., Mena, E., Illarramendi, A.: Location-dependent query processing: Where we are and where we are heading. (ACM Computing Surveys) (to appear).
18. Cheng, R., Kalashnikov, D.V., Prabhakar, S.: Evaluating probabilistic queries over imprecise data. In: ACM SIGMOD International Conf. on Management of Data (SIGMOD'03), San Diego, California, USA. (2003) 551–562
19. Tao, Y., Xiao, X., Cheng, R.: Range search on multidimensional uncertain data. ACM Transactions on Database Systems **32** (2007) 15
20. Kriegel, H.P., Kunath, P., Renz, M.: Probabilistic nearest-neighbor query on uncertain objects. In: 12th International Conf. on Database Systems for Advanced Applications (DASFAA'07), Bangkok, Thailand. (2007) 337–348
21. Corral, A., Manolopoulos, Y., Theodoridis, Y., Vassilakopoulos, M.: Closest pair queries in spatial databases. In: ACM SIGMOD International Conference on Management of Data (SIGMOD'00), Dallas, Texas, USA. (2000) 189–200