HAL
open science

A Content-Based Publish/Subscribe Matching Algorithm for 2D Spatial Objects

Athanasios Konstantinidis, Antonio Carzaniga, Alexander L. Wolf

To cite this version:

Athanasios Konstantinidis, Antonio Carzaniga, Alexander L. Wolf. A Content-Based Publish/Subscribe Matching Algorithm for 2D Spatial Objects. 12th International Middleware Conference (MIDDLEWARE), Dec 2011, Lisbon, Portugal. pp.208-227, 10.1007/978-3-642-25821-3_11 . hal-01597775

HAL Id: hal-01597775

https://inria.hal.science/hal-01597775

Submitted on 28 Sep 2017

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons Attribution 4.0 International License

A Content-Based Publish/Subscribe Matching Algorithm for 2D Spatial Objects

Athanasios Konstantinidis,* Antonio Carzaniga,† and Alexander L. Wolf*

*Department of Computing, Imperial College London, UK
†Faculty of Informatics, University of Lugano, CH

Abstract. An important concern in the design of a publish/subscribe system is its expressiveness, which is the ability to represent various types of information in publications and to precisely select information of interest through subscriptions. We present an enhancement to existing content-based publish/subscribe systems with support for a 2D spatial data type and eight associated relational operators, including those to reveal overlap, containment, touching, and disjointedness between regions of irregular shape. We describe an algorithm for evaluating spatial relations that is founded on a new dynamic discretization method and region-intersection model. In order to make the data type practical for large-scale applications, we provide an indexing structure for accessing spatial constraints and develop a simplification method for eliminating redundant constraints. Finally, we present the results of experiments evaluating the effectiveness and scalability of our approach.

1 Introduction

The data models of existing content-based publish/subscribe systems embody only the most basic and primitive types: numbers, strings, and dates. This is true, for example, of the Java Message Service specification (JMS), which is restricted to the primitive data types of Java.[1] In this paper we present an extended data model supporting 2D spatial objects. The representation of 2D objects is essential in many problem domains, but perhaps its primary use is in the representation and processing of geographical information. This type of information has concrete applications in agriculture, transportation, logistics, infrastructure management, and more recently various personal applications.

The simplest approach to incorporating 2D spatial objects into a content-based publish/subscribe system would be to establish some sort of convention for indicating regions in an x/y coordinate space, such as by giving the locations of opposite corners of rectangular regions or by giving the center point of a circular region of some radius. The coordinate space itself might be abstract or it might be associated with a standard reference model such as longitude and

[1] JMS defines an SQL-like selection feature for its content-based subscriptions. However, this selection feature is practically equivalent to a set (i.e., a disjunction) of subscriptions based on name-operator-value constraints.

latitude. Notice that this simple approach has the benefit of requiring only some conventional use of the primitive data types and relational operators already commonly supported by publish/subscribe systems.

The simplest matching problem would be to test whether a point in a space is contained within a given region. This concept has been explored for so-called *location-based services* [1, 9, 10, 16, 29], and used in practice to underpin practical applications such as Facebook Places.[2] It has also been explored for use within the virtual world of games, where game state and actions are bound to a notion of virtual location [22].

Our goal is to support a richer concept of region and region matching. In particular, we seek to support regions having irregular, and therefore more realistic boundary shapes, whereas prior work has focused on simple shapes, such as axis-aligned rectangles [24, 27]. Moreover, we seek to support the matching of regions to other regions, not just points to regions, which implies a much richer set of potential matching relations, such as *overlaps*, where two regions share some interior points but not others, *meets*, where two regions touch but do not overlap, and *equals*, where two regions mutually cover each other.

Consider, for example, an event notification emitted by a severe-weather warning system that reports a storm affecting a geographical area A with maximum expected wind speed w and precipitation level p. A facility management company might be interested in receiving such notifications, but only when A overlaps with one of the company's facilities (e.g., a building), and if the wind speed w and precipitation level p are above certain safety thresholds. In this and similar applications, subscriptions would include 2D geographical regions obtained from, say, a geographical information system (GIS) database, while notification messages would contain 2D observation overlays, such as weather systems, pollutant concentrations, group/herd movement, and the like. In order to capture such regions and their relationships within a content-based publish/subscribe system, a new data type for 2D spatial objects is required.

We derive the new data type from a standard model found in geographical information systems in which a 2D region is defined as a the space enclosed within a boundary consisting of line segments. Given two such boundary-delimited regions, the model induces eight binary relations between the regions. The theoretical basis for the evaluation of the eight relations is the 4-intersection model developed in the seminal work of Egenhofer and Franzosa [14] and used in many spatial database management systems. In this model, each binary relation between two regions A and B can be evaluated by testing the emptiness of four intersections between the boundary, interior, and exterior point sets of A and B.

Starting from this abstract model, we develop a concrete representation of regions based on a *discretization* of their boundaries, and use the eight basic binary relations between regions to form the relational operators of the type. We also derive a small set of concrete conditions that lead to an efficient evaluation of the relations. The boundary discretization, and the corresponding evaluation of the spatial relations, are based on a newly formulated variant of the Egenhofer

[2] `http://www.facebook.com/places/`

and Franzosa 4-intersection model. We were driven to develop this new model because the original requires all interior points of a region to be inspected and conventional discretization methods [17, 18] rely on a global grid structure.

The original model and conventional methods were developed for traditional GIS database applications, where the problem is to compute and store all the spatial relations among large numbers of relatively static regions, and then quickly process a spatial query against those stored relations [26]. In that context it is reasonable to incur the considerable computational costs of a global-grid discretization and a full interior point inspection, since they can be performed in an off-line preprocessing step. Publish/subscribe systems, by contrast, face the problem of having to perform an on-line computation to reveal the spatial relations that exist between previously unseen regions contained in high-rate message traffic and large numbers of stored regions representing spatial constraints. This demands the new approach introduced in this paper in which boundary discretization is dynamic and point inspections can be significantly reduced.

The ideas presented here provide the theoretical basis for accommodating 2D spatial objects in the basic matching/filtering/forwarding function of publish/subscribe systems, as well as the routing function of distributed versions of such systems. In this paper we focus on the matching problem, describing a specific algorithm, indexing structure, and logical simplification method for 2D spatial constraints, integrated and evaluated within a general content-based matching algorithm. The indexing structure, which we call the *CR-tree*, is an extension of the well-known R-tree developed by Guttman [20]. Our extension allows the matching algorithm to efficiently evaluate a large set of spatial constraints, as we demonstrate experimentally. We also demonstrate the effectiveness of the spatial-constraint simplification method.

In summary, we make the following contributions: (1) the use of topological relations between 2D spatial objects within content-based publish/subscribe systems; (2) a discretization of a common 2D model that admits to an efficient representation and use of 2D objects in matching; (3) an indexing structure to process large numbers of 2D spatial constraints during the matching process; and (4) a logical simplification method for 2D spatial constraints. We provide background on general spatial modeling in Section 2, and then define a specific spatial model, its discretization, and a matching algorithm in Section 3. The CR-tree index and spatial simplification method are described in Section 4. We present the results of an experimental evaluation in Section 5.

2 Background

To support 2D objects within a content-based publish/subscribe system, we must define spatial concepts and models that would lead to efficient and robust implementations. In particular, we seek a fast matching algorithm capable of evaluating incoming 2D regions (represented as attribute values in messages) against potentially large numbers of 2D spatial constraints (contained in subscriptions). In this section, we lay out such definitions and models. Furthermore,

since we develop the 2D spatial model as an extension of a concrete content-based matching algorithm, namely the Siena Fast Forwarding algorithm [8], we also review the algorithm and its existing indexing structures.

2.1 Spatial Concepts and Spatial Modeling

Space is regarded as being composed of an infinite number of points forming a continuum, the so-called Euclidean model. We represent a region as a polygon. More specifically:

Definition 1. *A* region *is a simple polygon, that is, the portion of the Euclidean space delimited by a closed finite sequence of line segments such that any two adjacent segments share an end point, and no end point belongs to more than two segments. Furthermore, no point other than an end point is shared by two segments (i.e., line segments do not intersect).*

As a concrete example, consider a message used within an environmental management system to warn about leaks of dangerous chemicals or other polluting agents in lakes or oceans. A hypothetical message of this type is shown in Figure 1. The warning indicates the affected area by specifying a 2D region (attribute "area"). The region is defined by the end points of the line segments that form its boundary, given as a sequence (only partially shown) of point pairs.

string warning = "hazardous leak"
region area = $(23.1, 10.9), (30.3, 27.0), (48.0, 19.0), \ldots$
int concentration = 172
int hazard = 4

Fig. 1. A message describing a 2D object as the region "area".

Using the *region connection calculus* introduced by Cohn et al. [12] we can prove the existence of eight *jointly exhaustive* and *pair-wise disjoint* (JEPD) binary topological relations between 2D regions. This means that any pair of regions must have a topology that is characterized by one and only one of the eight relations. The complete set of relations between two regions A and B is illustrated in Figure 2. The first four relations are symmetric; the remaining four should be read as A *relation* B (e.g., A *inside* B). These qualitative topological relations between regions define the constraints that can be expressed in subscriptions that select 2D spatial objects. Since our objective is to identify all matching constraints for a given region, we must develop an algorithmic evaluation of these relations.

In order to implement such constraints in subscription predicates, we need to model the concept of topological relations between regions. Following the analysis of Egenhofer et al. [13–15], if $A \subseteq \mathbb{R}^2$ is a region, then there exists a set $U \subseteq \mathbb{R}^2$ such that $U = (A^0 \cup \partial A \cup A^-)$, where A^0, A^-, and ∂A are the interior,

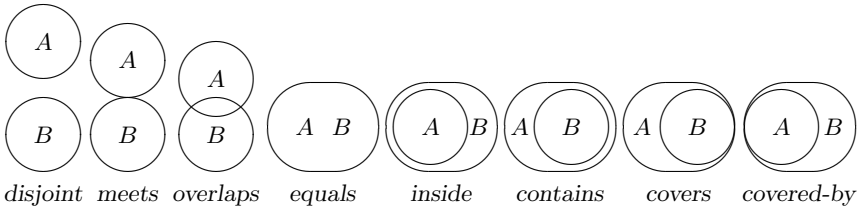


Fig. 2. The eight jointly exhaustive and pair-wise disjoint (JEPD) relations between regions.

exterior, and boundary sets of A, respectively. Notice that they are mutually disjoint subsets of U, that is, A^0, ∂A, and A^- form a partition of U. We refer to U as the *universe*. It is then provable that the topological relation between any two regions $A, B \subseteq U$ can be identified by specifying the nine possible intersections between their interiors, exteriors, and boundaries (the 9-intersection model), each of which can be either empty or non-empty. Egenhofer and Franzosa [14] introduced a model based on only four intersections (the 4-intersection model), obtained by removing redundant entries from the 9-intersection model. Below is the list of intersections in the 4-intersection model of Egenhofer and Franzosa.

Intersection 1. Intersection between the boundary of A and the boundary of B denoted by $\partial A \cap \partial B$.
Intersection 2. Intersection between the boundary of A and the interior of B denoted by $\partial A \cap B^0$.
Intersection 3. Intersection between the interior of A and the boundary of B denoted by $A^0 \cap \partial B$.
Intersection 4. Intersection between the interior of A and the interior of B denoted by $A^0 \cap B^0$.

In Section 3 we introduce a discretization of regions, as well as a variation of this 4-intersection model, that are specifically designed to ensure efficient and robust evaluation of the JEPD spatial constraints.

2.2 The Siena Fast Forwarding Algorithm

Siena is a popular distributed publish/subscribe system that uses name-value pairs in published messages and name-operator-value constraints to define subscriptions [7]. Following the terminology used in Siena, a *filter* is a conjunction of constraints (that defines a subscription) and a *predicate* is a disjunction of filters (representing a set of subscriptions).

The matching algorithm developed within the Siena project, called Siena Fast Forwarding (SFF) [8], follows the approach of Yan and Garcia-Molina [30] by using a counting algorithm for predicate matching. In particular, the algorithm builds a *forwarding table* consisting of a global index of all the unique constraints found in a set of predicates, where each predicate is associated with an interface

of a content-based broker/router. Given this constraint index and an input message, the matching process amounts to finding the set of interfaces through which the message must be forwarded. The algorithm proceeds by evaluating each attribute of the message against the index of constraints, counting the numbers of matching constraints per filter. When the count reaches the total number of constraints in a filter associated with a predicate, the algorithm forwards the message to the interface associated with that predicate. As an example, Figure 3 shows the high-level contents of a forwarding table for a broker/router with two interfaces, I_1 and I_2, each associated with a predicate of two filters, $f_{1.1}$ and $f_{1.2}$, and $f_{2.1}$ and $f_{2.2}$, respectively. Notice that the message shown in Figure 1 would match $f_{2.2}$ and, therefore, would be forwarded through interface I_2.

I_1	$f_{1.1}$	*string* stock ="MTK" *int* price < 100
	$f_{1.2}$	*string* stock = "DYS" *int* price > 200 *bool* bubble = true
I_2	$f_{2.1}$	*region* cloud *overlaps* $(10, 5), (7, 12), \ldots$ *int* pressure < 1000
	$f_{2.2}$	*string* warning = "hazardous leak" *region* area *disjoint* $(3, 2), (15, 40), \ldots$ *int* concentration > 100 *int* hazard > 3

Fig. 3. A forwarding table with 2D regions.

SFF already implements specific constraint indexes for some basic data types, including integer, float, Boolean, and string, along with their operators. It also provides a way to extend the algorithm with more types and operators, and with type- and operator-specific constraint indexes. We use this extension feature to plug in our implementations for representing 2D regions, a spatial constraint index, and the algorithm for evaluating 2D constraints.

3 Spatial Model

The abstract notion of a 2D object is generally defined by a set of points taken from a continuous space. Therefore, in order to realize a concrete implementation of this abstract model, we must somehow map these continuous 2D objects onto a discrete structure amenable to efficient algorithmic processing. The term *discretization* refers generally to both the mapping of a continuous object onto a discrete set and the algorithmic processing of that discrete set. A conventional approach to the discretization of 2D regions is to construct a discretized universe using a global grid structure [17, 18]. However, this method involves the mapping of regions onto grid points, which is a surprisingly complex procedure [19, 21]

that would introduce unacceptable levels of overhead to a publish/subscribe system.

We follow a completely different approach. The key idea behind this new discretization is to use a point-region check as the primary building block for the evaluation algorithm. In particular, let POINTCHECK(p, R) be a decision procedure that returns the topological relation between point p and region R as either *in*, *out*, or *meet*, meaning that p is within the interior, exterior, or boundary of R, respectively. This POINTCHECK procedure takes two discrete structures, a point and a finite set of line segments, and can be implemented efficiently using a ray-shooting algorithm from a computational-geometry library [5, 23]. We use the procedure to efficiently evaluate the eight JEPD spatial relations through a variant of the 4-intersection model of Egenhofer and Franzosa.

3.1 A New 4-Intersection Model

The 4-intersection model of Egenhofer and Franzosa reduces the identification of the topological relations between two regions A and B to the problem of determining whether each of the four intersections is empty or not. Our general approach to discretizing each intersection decision is to find a "witness" point that would indicate that the intersection is not empty. In particular, the first three of the four intersections can be discretized as follows:

Discretization 1. To decide the emptiness of $\partial A \cap \partial B$, find a point p on the boundary of A that can be tested as a boundary point for B. If such a p is found, then consider the intersection *non-empty*; otherwise *empty*.

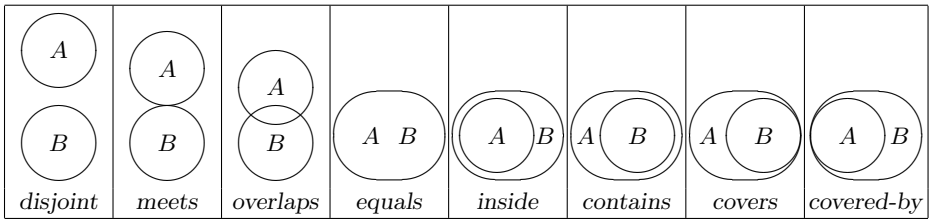
Discretization 2. To decide the emptiness of $\partial A \cap B^0$, find a point p on the boundary of A that can be tested as an interior point for B. If such a p is found, then consider the intersection *non-empty*; otherwise *empty*.

Discretization 3. To decide the emptiness of $A^0 \cap \partial B$, find a point p on the boundary of B that can be tested as an interior point for A. If such a p is found, then consider the intersection *non-empty*; otherwise *empty*.

The fourth intersection in the model of Egenhofer and Franzosa is $A^0 \cap B^0$, which involves two interior sets whose discretization is prohibitively complex for achieving fast content-based matching. To overcome this problem we replace the fourth intersection with the intersection between the boundary set of A and the exterior set of B. Thus, we define a new 4-intersection model with the new fourth intersection being $\partial A \cap B^-$, which admits to an efficient discretization similar in form to the previous three:

Discretization 4. To decide the emptiness of $\partial A \cap B^-$, find a point p on the boundary of A that can be tested as an exterior point for B. If such a p is found, then consider the intersection *non-empty*; otherwise *empty*.

Of course, given this change to the intersection model, we must show that all eight topological relations between regions can still be decided unambiguously.



$(\partial A \cap \partial B, \partial A \cap B^0, A^0 \cap \partial B, \partial A \cap B^-)$

$(\emptyset, \emptyset, \emptyset, 1)$	$(1, \emptyset, \emptyset, 1)$	$(1, 1, 1, 1)$	$(1, \emptyset, \emptyset, \emptyset)$	$(\emptyset, 1, \emptyset, \emptyset)$	$(\emptyset, \emptyset, 1, 1)$	$(1, \emptyset, 1, 1)$	$(1, 1, \emptyset, \emptyset)$

$\emptyset$: empty, 1: non-empty

Table 1. The eight JEPD relations as captured by the new 4-intersection model.

We do this in Table 1, which details the precise correspondence between the four intersections and the topological relations.

We can now proceed to implement the evaluation of the topological relations as a concrete algorithm. We emphasize that this algorithm is based on a crucial property of the new 4-intersection model, namely that all its intersections involve at least one boundary set ∂X, and either another boundary set ∂Y, an interior set Y^0, or an exterior set Y^-. Having a boundary set ∂X to start with, the algorithm can proceed first by discretizing ∂X into a finite set of points $p \in \partial X$ and then by checking the relation of each point p with the other region Y (boundary, interior, or exterior) using the POINTCHECK(p, Y) primitive.

3.2 Algorithm

Our method for identifying the topological relation between two regions A and B, each expressed as a sequence of points, is given as Algorithm 1. The algorithm uses four Boolean variables i_1, i_2, i_3, and i_4 that indicate the presence of "witnesses" for the non-emptiness of each of the four intersections in our 4-intersection model. The variables are initialized as false and become true as soon as the algorithm finds one point in the corresponding intersection.

The algorithm can find witness points for the first, second, and fourth intersections (thereby assigning i_1, i_2, and i_4) through a single iteration over the points of the boundary of the first region ∂A (lines 6–18). In this loop the algorithm considers only some points of the boundary ∂A, effectively discretizing that boundary. The selection of points is performed by the DISCRETIZE procedure sketched on lines 46–49 and discussed in detail in Section 3.3, below.

The loop over the points of ∂A may terminate immediately whenever the algorithm finds witnesses for intersections 2 and 4, since they unambiguously identify the *overlaps* relation. At the end of the loop, the algorithm can identify the *equals*, *inside*, and *covered-by* relations. Lines 19–25 implement this decision based on the values of i_1, i_2, and i_4. Notice that some of the conditions

Algorithm 1 Topological relation between two regions.

```
1: procedure FINDRELATION(A, B)
2:     i1 ← FALSE                                   ▷ witness: i1 ⇒ ∂A ∩ ∂B ≠ ∅
3:     i2 ← FALSE                                   ▷ witness: i2 ⇒ ∂A ∩ B^0 ≠ ∅
4:     i3 ← FALSE                                   ▷ witness: i3 ⇒ A^0 ∩ ∂B ≠ ∅
5:     i4 ← FALSE                                   ▷ witness: i4 ⇒ ∂A ∩ B^− ≠ ∅
6:     for each p ∈ DISCRETIZE(∂A) do
7:         T ← POINTCHECK(p, B);
8:         if T = meet then
9:             i1 ← TRUE
10:        else if T = in then
11:            i2 ← TRUE
12:        else if T = out then
13:            i4 ← TRUE
14:        end if
15:        if i2 ∧ i4 then
16:            return overlaps                      ▷ (?, 1, ?, 1)
17:        end if
18:    end for
19:    if i1 ∧ ¬i2 ∧ ¬i4 then
20:        return equals                            ▷ (1, ∅, ?, ∅)
21:    else if ¬i1 ∧ i2 ∧ ¬i4 then
22:        return inside                            ▷ (∅, 1, ?, ∅)
23:    else if i1 ∧ i2 ∧ ¬i4 then
24:        return covered-by                        ▷ (1, 1, ?, ∅)
25:    end if
26:    for each p ∈ DISCRETIZE(∂B) do
27:        T ← POINTCHECK(p, A);
28:        if T = in then                           ▷ A^0 ∩ ∂B ≠ ∅
29:            i3 ← TRUE
30:            break (goto line 35)
31:        else if T = out then   ▷ no overlap here, so it must be A^0 ∩ ∂B = ∅
32:            break (goto line 35)
33:        end if
34:    end for
35:    if ¬i1 ∧ ¬i3 then                            ▷ no need to check ¬i2 ∧ i4
36:        return disjoint                          ▷ (∅, ∅, ∅, 1)
37:    else if i1 ∧ ¬i3 then
38:        return meets                             ▷ (1, ∅, ∅, 1)
39:    else if ¬i1 ∧ i3 then
40:        return contains                          ▷ (∅, ∅, 1, 1)
41:    else if i1 ∧ i3 then
42:        return covers                            ▷ (1, ∅, 1, 1)
43:    end if
44: end procedure
45:
46: procedure DISCRETIZE(X)          ▷ parameter D is the linear point density
47:     ℓ ← length of ∂X
48:     return ℓD equally spaced points on ∂X
49: end procedure
```

are redundant (e.g., when the algorithm reaches line 19, i_2 and i_4 cannot both be true). However, for clarity and ease of verification, we write each condition explicitly and in the same order as the corresponding relation appears in Table 1.

If none of the immediately identifiable relations hold, the algorithm proceeds by checking whether the third intersection is empty or not. The algorithm iterates through the points of the boundary ∂B, checking each point against A. As soon as a point in ∂B is found to be an interior point of A, the algorithm breaks out of the loop with $i_3 = \mathit{true}$. The algorithm also breaks out of the loop if a point in ∂B is found to be an exterior point of A, since we know that A and B do not overlap. Finding one point of ∂B outside of A implies that no points will be found inside, so the loop terminates with $i_3 = \mathit{false}$.

After the loop, i_1 and i_3 can be used to identify one of the remaining possible relations, namely *disjoint*, *meets*, *contains*, or *covers* (lines 35–43).

3.3 Boundary Discretization

The algorithm tests a finite number of points in a boundary to find a witness for each intersection. These points are chosen uniformly along the boundary by the procedure DISCRETIZE based on a *linear point density* parameter D. More specifically, DISCRETIZE starts with the points that define the boundary (i.e., the end points of the line segments that compose the boundary) and then adds equally spaced points on each segment to reach a total density of D points per unit of length. Let A be a region defined by n line segments of total length ℓ. Assuming $D \geq n/\ell$, DISCRETIZE selects all the n end points of the line segments, plus another $\ell D - n$ equally spaced points along ∂A. Of course, the point density parameter D has a crucial effect on performance and precision. Higher densities mean higher precision but also potentially higher execution times. The setting of this parameter is therefore application specific.

Notice that the algorithm is likely to iterate only over the first region boundary ∂A, with a run-time complexity proportional to the length of ∂A. Therefore, one way to save some time is to evaluate the topological relation between B and A instead of A and B when the length of ∂B is smaller than the length of ∂A. In this case, the algorithm must translate the result of the evaluation between B and A back to the corresponding relation between A and B. This is easily done, since *disjoint*, *meets*, *overlaps*, and *equals* are symmetric relations that do not need translation, and A *contains* B is equivalent to B *inside* A, and A *covered-by* B is equivalent to B *covers* A.

Finally, to increase numerical robustness in the algorithm, we introduce a halo around each point with radius $r = d/2$, where d is the (shortest) distance between the point and its adjacent points along the same line segment. This provides a more robust and qualitatively better result by providing a smoother transition from *disjoint* to *overlaps* via an extended *meets* region.

3.4 Complexity

In the worst case, Algorithm 1 executes two full loops over the discretized boundaries of the input regions. Each loop starts with an invocation of the DISCRETIZE procedure, and each iteration of the loop invokes POINTCHECK. Let n be the total number of line segments in the boundaries of the two input regions, and let ℓ be the total length of those boundaries. Then, the complexity of the POINTCHECK algorithm we use is $O(n)$, and with a point density D, the complexity of DISCRETIZE is $O(\ell D)$. Therefore, since there are a total of ℓD iterations, the overall complexity of Algorithm 1 is $O(n\ell D)$.

4 Indexing and Simplification

The algorithm described in Section 3 simply evaluates one spatial constraint at a time. If we were to incorporate that algorithm as is within a broker/router, the broker/router would have to process each incoming message using a linear scan of all spatial constraints in its forwarding table. In this section we present an indexing structure and simplification method that are designed to avoid linear scans and reduce the sheer number of spatial constraints that must be checked.

4.1 Spatial Index

The indexing structure presented here is an extension of the well-known Guttman R-tree index [20]. Many optimized versions and variants of the original R-tree have been proposed, including the R*-tree [2], TV-tree [25], X-tree [4], and R+-tree [28]. They are mainly used in computer-aided design, GIS database, and computer graphics applications, where fast spatial searching is a necessity.

R-trees have also been employed in publish/subscribe systems, but for a completely different purpose: Rather than representing constraints on 2D spatial objects, they have been used to represent constraints on values drawn from primitive data types, such as integers. For example, Bianchi et al. [6] introduce an R-tree index called the DR-tree to capture the covering relation among subscriptions, such that when a match for a subscription s_1 implies a match for a subscription s_2 they can avoid processing s_2 if a match is found for s_1. Such uses of R-tree-like indexing structures view the constraints as forming axis-aligned, rectangular "value spaces" and simply evaluate them for what we would call here the *covers* or *covered-by* relations. Instead, we develop a variant of the R-tree to represent constraints on true 2D objects of irregular shape, and evaluate them for a broader set of relations meaningful in the realm of physical space.

Guttman's R-tree represents complex spatial objects by covering them with less complex ones. Specifically, the R-tree uses *minimum bounding rectangles* (MBRs) to represent n-dimensional objects. R-trees are structurally similar to other search trees, particularly B-trees. Each node t in an R-tree holds an MBR R_t that contains or covers all spatial objects stored in t's subtree. This property guides the search over the R-tree: The search algorithm starts with a query MBR

R_q and walks the tree by descending into every subtree t whose MBR R_t overlaps with R_q. The search then returns all visited objects that are contained in R_q.

The spatial index we have developed is based on the original R-tree and uses MBRs to represent sets of stored objects. However, unlike an R-tree that stores spatial objects, this index must store *constraints* on spatial objects. We call such a structure a *constraint R-tree*, or *CR-tree*. A CR-tree extends the algorithms of the original R-tree to implement a fast search over a potentially large set of spatial constraints. In particular, the search takes a query object q representing a value in a message, and returns all the spatial constraints stored in the CR-tree that are satisfied by q. A CR-tree stores constraints in its leaves, but is otherwise structurally identical to an R-tree. Each node t in a CR-tree holds the MBR of the union of the regions that define all the constraints stored in t's subtree. In addition, node t holds a list of all the *disjoint* constraints stored in t's subtree. CR-tree insertion and deletion use the corresponding R-tree algorithms, plus a simple linked-list maintenance operation to update the list of *disjoint* constraints.

The CR-tree search algorithm is also based on the R-tree algorithm. It starts by computing the MBR R_q of the query object q and then walks through the CR-tree to find potential matching constraints. To decide whether to visit a node t in the CR-tree, the search algorithm evaluates the topological relation between the query MBR R_q and the node MBR R_t. If t is an internal node, the search proceeds as in an R-tree by visiting t whenever R_q intersects R_t. If R_q does not intersect R_t, then the search algorithm does not visit t, but instead treats all the *disjoint* constraints associated with t as having been immediately matched.

If t is a leaf node, then the search algorithm visits t and evaluates the constraint stored in t if any one of a set S_t of specific relations holds between R_q and R_t. The specific set S_t associated with node t depends on the spatial constraint stored in t. For example, if t stores a *covers* constraint with comparison region X (bounded by t's MBR R_t), then a search with query MBR R_q must visit t and consider that *covers* constraint if R_q *covers* R_t or if R_q *equals* R_t. Notice that these relations are evaluated between rectangles (i.e., MBRs), and can therefore be checked in constant time without resorting to Algorithm 1, which is used only at the point where individual constraints must be evaluated. Table 2 shows the complete mapping between spatial constraints stored in leaf nodes of the CR-tree and the corresponding set of MBR relations [11].

Thus, we can view the CR-tree as a means to reduce the set of constraints that must be thoroughly checked using Algorithm 1. Moreover, the CR-tree does not simply exclude constraints because they do not match, but also allows us to immediately decide that some of those constraints *do* match, again without resorting to Algorithm 1. In this sense, the CR-tree, unlike the R-tree from which it is derived, is more than just a traditional search tree, but also a participant in the evaluation process.

Another important element of an R-tree is the splitting algorithm, which is used to split nodes when they become too large due to the insertion of many objects. In our CR-tree we use the *quadratic-cost* splitting algorithm [20], which provides a good balance between simplicity and performance.

Constraints	MBR relations
disjoint (D)	$D \vee M \vee O \vee I \vee cB \vee Ct \vee Cv$
meets (M)	$M \vee O \vee E \vee I \vee cB \vee Ct \vee Cv$
overlaps (O)	$O \vee E \vee I \vee cB \vee Ct \vee Cv$
inside (I)	I
contains (Ct)	Ct
covered-by (cB)	$cB \vee E$
covers (Cv)	$Cv \vee E$
equal (E)	E

Table 2. MBR relations for each type of constraint.

4.2 Simplification

Simplification is the process of removing redundant constraints from a forwarding table. For example, a filter on integer values $x \neq 0 \wedge x > 10$ can be rewritten simply as $x > 10$ because $x > 10 \Rightarrow x \neq 0$ for all x or, in other words, because $x \neq 0$ "covers" $x > 10$. We develop a similar form of simplification scheme for 2D spatial constraints. As is commonly done in the publish/subscribe literature, we first define "covers" and "conflicts" relations between spatial constraints. Let C_1 and C_2 be two spatial constraints defined by spatial relation rel_1 and region R_1, and relation rel_2 and region R_2, respectively. We then say that C_1 *covers* C_2 iff $X\ rel_2\ R_2 \Rightarrow X\ rel_1\ R_1$ for all regions X. Similarly, we say that C_1 *conflicts* with C_2 iff $\neg(X\ rel_1\ R_1 \wedge X\ rel_2\ R_2)$ for all regions X.

As is the case for basic data types, the covers and conflicts relations between two spatial constraints C_1 and C_2 can be evaluated efficiently on the basis of the topological relation between R_1 and R_2, and can therefore be used to simplify predicates. Let P be a predicate consisting of a disjunction of n filters $f_1, f_2, \ldots, f_n$, where each filter f_i consists of a conjunction of m_i constraints $C_{i,1}, C_{i,2}, \ldots, C_{i,m_i}$. Pairwise redundant constraints in a filter, entire filters, and pairwise redundant filters can now be identified and eliminated through the following simplification rules.

Rule 1. A filter f_i can be removed from predicate P if, for any pair of constraints $C_{i,j}$ and $C_{i,k}$ in f_i, $C_{i,j}$ *conflicts* with $C_{i,k}$. This is because, from the definition of the conflict relation, f_i is always *false*.

Rule 2. If $C_{i,j}$ and $C_{i,k}$ are two constraints in the same filter f_i, then $C_{i,j}$ can be removed if $C_{i,j}$ *covers* $C_{i,k}$. This is because, from the definition of the cover relation, any region that satisfies $C_{i,k}$ also satisfies $C_{i,j}$.

Rule 3. Let f_h and f_i be two filters in predicate P. We can eliminate filter f_i from P if for all constraints $C_{h,k}$ in f_h there exists a constraint $C_{i,j}$ in f_j such that $C_{h,k}$ *covers* $C_{i,j}$. This is because, from the definition of the covers relation, f_i is satisfied by a subset of the messages that satisfy f_h. This rule can also be seen as the definition of a covering relation between filters.

We implement a simplification algorithm based on these rules. The implementation realizes the covers and conflicts relations for all combinations of spatial relations in the intuitive way. For example, (x *covered-by* A) *conflicts with* (x *disjoint* B) if (A *overlaps* B). We discuss the effectiveness of spatial simplifications as part of the experimental evaluation presented Section 5.

5 Evaluation

We now present an experimental evaluation of the matching performance and general scalability of the spatial model and its implementation. In particular, we ask whether the absolute performance is acceptable and, more importantly, how the matching time scales with the number of spatial constraints. For this purpose we use both synthetic workloads and workloads derived from real GIS data. The synthetic workloads are useful in highlighting the scalability of the matching algorithm, especially in worst-case configurations, while the GIS workloads show how the matching algorithm performs in realistic situations.

We also compare the approximate matching of our algorithm against the exact matching of a *polygon-intersection* algorithm. Polygon intersection is a binary decision problem that is unable to distinguish among the specific relations of the region-connection calculus. In particular, a *false* result from polygon intersection indicates either *meets* or *disjoint*, while a *true* result indicates either *overlaps*, *equals*, *inside*, *contains*, *covers*, or *covered-by*. Despite this shortcoming, it is important to consider polygon intersection, since it is a well-studied problem in classical computational geometry having known efficient algorithms [3] and well-engineered implementations.[3] We use it as a performance benchmark.

Synthetic Regions. We generate synthetic regions, serving as spatial constraints or message attributes, within a rectangular universe U of size U_X by U_Y. We initially choose two points within U, uniformly at random, that represent the minimum bounding rectangle R for the region we generate. We then select three or four points each on a separate segment on the perimeter of R. We complete the generation of the region by adding points chosen uniformly at random in R, up to a total of 20 points, and then order the points to form the boundary of the region in such a way that the boundary has no intersecting segments (as per Definition 1 in Section 2.1). Finally, we exclude regions with an area smaller than 10% of the size of the universe U so as to obtain regions of similar size and, therefore, a more diverse combination of topological relations.

Implementation. The experiments use an implementation of the spatial model integrated within the Siena Fast Forwarding (SFF) matching framework[4] and executed with a sequential, single-threaded matching process on an Intel i5 processor with 6GB of DDR3 memory. A crucial parameter in the implementation is the linear point density D. The first set of experiments use a fixed point density of 2000 points per universe half-length, that is $D = 2000/(U_X + U_Y)$. Because

[3] `http://www.cgal.org` (CGAL: Computational Geometry Algorithms Library)
[4] `http://www.inf.usi.ch/carzaniga/cbn/forwarding/`

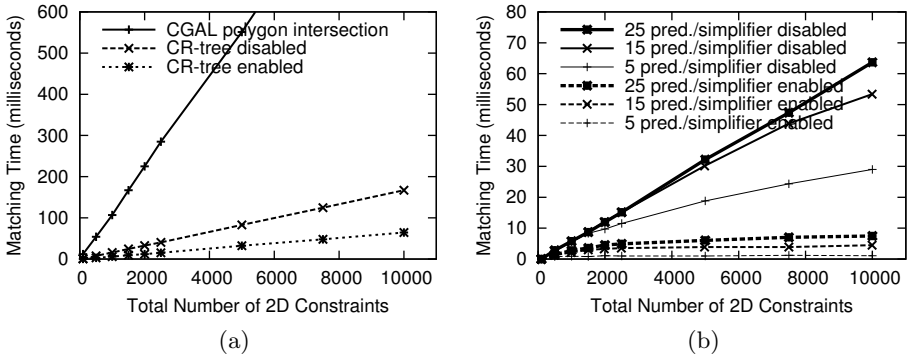


Fig. 4. Performance of the matching algorithm: (a) with and without the CR-tree constraint index enabled and (b) with and without spatial simplification enabled.

we use a square universe ($U_X = U_Y$), the chosen point density intuitively can be seen as a 1000×1000 pixel image. We experimented with other point densities and found that the performance varies more or less linearly with the point density. In the second set of experiments, we study the trade off between accuracy and performance of the matching algorithm under various point densities.

5.1 Worst-Case Performance and Scalability

In our first experiment we measure the time needed to match an input region against synthetic workloads of increasingly larger sets of constraints. This experiment is intended to evaluate the performance of Algorithm 1 and the impact of the CR-tree index. To do so, we generate workloads consisting of N predicates (N up to 10000), each composed of a single filter with a single constraint. All constraints are on the same attribute and all messages contain the same single attribute associated with a 2D region, so each message carries an input region that must in principle be evaluated against all N constraints.

Figure 4a shows the performance of Algorithm 1 in isolation and together with the CR-tree, and in comparison with an exact polygon-intersection algorithm. Each data point represents the average matching time of several input regions over several sets of the same number of constraints. The variability of the matching time is minimal (1–2ms), so we do not show this in the plots. The solid line represents a sequential execution of CGAL's polygon-intersection algorithm over a list of all the constraints. The dashed line represents the same sequential execution of Algorithm 1, while the dotted line represents the use of the CR-tree index to reduce the number of examined constraints.

With the point density used in this experiment, Algorithm 1 incurs an error rate of only 0.089% and runs about six times faster than the exact CGAL polygon-intersection algorithm. Thus, the experiment shows that the approximate algorithm offers a good combination of performance and accuracy. We

repeated this experiment with different point densities and found that the approximate algorithm is still faster than the exact algorithm with a point density of 12000 points per universe half-length and a corresponding error rate of only about 0.015%.

The experiment demonstrates that the absolute matching times are within reasonable bounds. Notice that it represents an extreme worst-case scenario, requiring the evaluation of thousands of constraints on the same attribute and without the possibility of taking shortcuts. The results also show that the CR-tree is effective in reducing the matching time of a linear scan. This, too, is a worst-case scenario for the CR-tree, since we use regions that are large with respect to the universe and, therefore, they do not lend themselves to an effective partition under disjoint MBRs. In the experiment of Section 5.2 we see that when applied to realistic GIS workloads the performance is substantially better.

In a second experiment, shown in Figure 4b, we evaluate the performance of the constraint index (Algorithm 1 within a CR-tree) in a scenario of more articulated predicates. In particular, we construct workloads in which a total of N constraints (N up to 10000) are distributed over 5, 15, and 25 predicates, each consisting of filters of two constraints. Even though this experiment explores a more realistic set of predicates, we still focus only on the performance of the 2D spatial components of the matching algorithm. Therefore, as in the first experiment, we use a single attribute in all constraints and in all messages.

Each line in Figure 4b shows the matching time as a function of the total number of constraints. We can see that the behavior of the SFF matcher extended with support for 2D spatial objects is consistent with that of the original SFF matcher [8], and that the absolute matching times are also reasonable. In particular, the matching times tend to be flat for larger and larger predicates. This effect is not due to the 2D spatial constraint index, but rather to the structure of the SFF algorithm and also to the nature of the matching problem: with fewer and larger predicates, a message is likely to match at least one filter for all predicates, thereby cutting short the full evaluation.

The results in Figure 4b also show the effectiveness of the spatial simplifier. The solid lines show the performance for non-simplified workloads, while the corresponding dashed lines show the performance under simplification. Notice that the simplification method effectively accelerates the flattening of the matching times. This is because as more filters (and constraints) are added to the same predicate, more of those filters (and constraints) are likely to become redundant.

5.2 Accuracy and Performance in Realistic Configurations

To evaluate our algorithm in realistic configurations, we derive workloads from three GIS data sets. The first contains 85550 polygons representing vegetation in southern California, the second 17048 polygons representing natural features (parks, forests, woods, water areas, etc.) in the Rhône-Alpes district of France, and the third 4170 polygons representing buildings in central London.[5]

[5] We obtained the first data set from `http://atlas.ca.gov`, and the second two from `http://www.openstreetmap.org`.

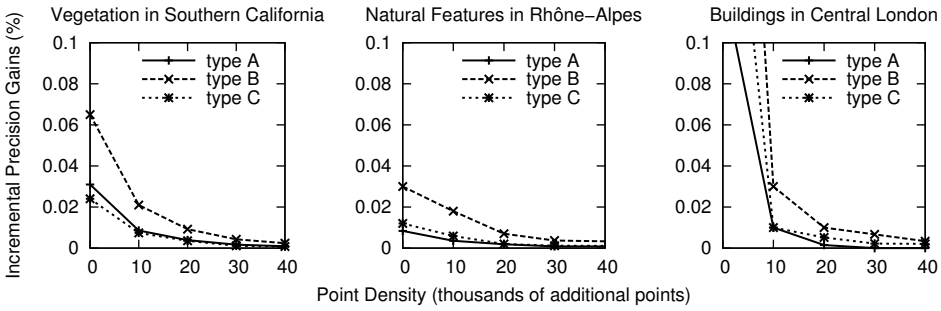


Fig. 5. Incremental precision gain as a function of the point density.

Using the process described at the beginning of this section we generate three different types of input regions for placement into messages, referred to here as A, B, and C. Type A represents regions that are relatively round and uniform shapes, such as clouds and storms, having nearly square MBRs and at most 10 boundary points. Type B represents regions of irregular and more complex shapes, such as the movement of herds and environmental hazards such as forest fires. These regions exhibit correspondingly less-regular MBRs and between 30 and 50 boundary points. Finally, type C represents narrow and long regions, described using up to five boundary points, such as the paths taken by aircraft at high altitudes. The sizes of all three types of regions are chosen randomly to have an MBR area between 1% and 25% of the universe.

We first study how the point density affects the accuracy of the matching algorithm. We run the matching algorithm using 100 of each of the three types of input regions over each of the three GIS-derived sets of spatial constraints. We vary the point density from 0 to 40000 points per universe half-length. Note that these are boundary points considered by the matching algorithm *in addition to* the points that define the boundary (so, a point density of 0 makes sense). For each point density D we compare the results of the matching process for density D with those using density $D + 10000$, and record the difference between the two sets of matched predicates.

We plot the results of this analysis in Figure 5. At a high level we can see that all the curves show very low differences and converge toward an exact match. To appreciate the meaning of each value, consider an incremental precision gain of 0.02% at density $D = 10000$. That 0.02% can be interpreted as the probability of the matching results differing by one predicate (matching in one case and not matching in the other) at densities $D = 10000$ and $D + 10000 = 20000$.

We next evaluate the performance of the matching algorithm. For this experiment we set the point density to $D = 10000$ (intuitively corresponding to a 5000×5000 universe). The experiments are set up according to two scenarios corresponding to the two sets of experiments presented in Section 5.1. For the first scenario, we construct a filter with one constraint from each polygon in the

			Scenario 1 1 filter per predicate 1 constraint per filter		**Scenario 2** 25 predicates 2 constraints per filter	
source data set	**number of constraints**	**region type**	**CR-tree**		**simplifier**	
			disabled	**enabled**	**disabled**	**enabled**
Vegetation in southern California	85550	A	1841.00 ms	*73.00 ms*	67.00 ms	*0.18 ms*
		B	4000.00 ms	*89.00 ms*	78.00 ms	*0.54 ms*
		C	1512.00 ms	*7.00 ms*	4.40 ms	*0.12 ms*
Natural features in Rhône-Alpes	17048	A	951.00 ms	*108.00 ms*	59.00 ms	*1.40 ms*
		B	1323.00 ms	*93.00 ms*	169.00 ms	*6.40 ms*
		C	851.00 ms	*7.00 ms*	72.00 ms	*0.53 ms*
Buildings in central London	4170	A	59.00 ms	*1.40 ms*	0.95 ms	*0.75 ms*
		B	169.00 ms	*6.40 ms*	5.00 ms	*1.24 ms*
		C	72.00 ms	*0.53 ms*	0.32 ms	*0.05 ms*

Table 3. Average matching times for GIS-derived workloads.

GIS data set, choosing 2D operators uniformly at random, and then we assign one filter per predicate. With one filter per predicate, this scenario is intended to stress-test the matching algorithm with and without the CR-tree enabled. In the more realistic second scenario, we construct 25 predicates by building filters consisting of two constraints generated from the regions in the GIS data set, and then simply combine the generated filters into the 25 predicates.

The results are reported in Table 3. The table shows the average matching times of several input regions, for each input region type, and over the three data sets. (Variance in matching times is minimal so not shown.) Notice that the results in both scenarios are consistent with the general qualitative characteristics of the matching algorithm outlined in the synthetic workload experiments of Section 5.1. However, in this more realistic case, the absolute performance of the algorithm is substantially better, with total matching times of fractions of milliseconds. This is because regions within the GIS data sets represent real objects and, therefore, tend to overlap considerably less as well as tend more often to be mutually disjoint than those of the worst-case synthetic workloads.

6 Conclusions

We have presented an enhancement to existing content-based publish/subscribe systems with support for a 2D spatial data type and eight associated relational operators. We described an algorithm for evaluating the spatial relations that is founded on a dynamic discretization method. In order to make the use of this new data type practical we developed an indexing structure for spatial constraints, called the CR-tree, as well as a simplification method for eliminating redundant spatial constraints. Our experimental evaluation demonstrates the effectiveness and scalability of our approach when integrated into a state-of-the-art publish/subscribe matching engine.

In future work we plan to further refine the CR-tree by exploring improved methods for splitting nodes as the number of constraints grows. The method we currently use is a generic one developed for the original R-tree spatial-object index. We suspect that there might be more effective methods, possibly heuristic in nature, tailored to an index of spatial constraints.

References

1. Bauer, M., Rothermel, K.: Towards the observation of spatial events in distributed location-aware systems. In: Proceedings of the International Conference on Distributed Computing Systems Workshops. pp. 581–582 (2002)
2. Beckmann, N., Kriegel, H.P., Schneider, R., Seeger, B.: The R*-tree: An efficient and robust access method for points and rectangles. In: Proceedings of the ACM SIGMOD International Conference on Management of Data. pp. 322–331 (1990)
3. Bentley, J.L., Ottmann, T.A.: Algorithms for reporting and counting geometric intersections. IEEE Transactions on Computers 28, 643–647 (Sep 1979)
4. Berchtold, S., Keim, D.A., Kriegel, H.P.: The X-tree: An index structure for high-dimensional data. In: Proceedings of the 22nd International Conference on Very Large Data Bases. pp. 28–39. San Francisco, California (Sep 1996)
5. de Berg, M., Cheong, O., van Kreveld, M., Overmars, M.: Computational Geometry: Algorithms and Applications. Springer-Verlag (2008)
6. Bianchi, S., Datta, A., Felber, P., Gradinariu, M.: Stabilizing peer-to-peer spatial filters. In: Proceedings of the 27th International Conference on Distributed Computing Systems (Jun 2007)
7. Carzaniga, A., Rosenblum, D.S., Wolf, A.L.: Design and evaluation of a wide-area event notification service. ACM Transactions on Computer Systems 19(3), 332–383 (2001)
8. Carzaniga, A., Wolf, A.L.: Forwarding in a content-based network. In: Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications. pp. 163–174 (2003)
9. Chen, X., Chen, Y., Rao, F.: An efficient spatial publish/subscribe system for intelligent location-based services. In: Proceedings of the 2nd International Workshop on Distributed Event-Based Systems. pp. 1–6. ACM (2003)
10. Chen, Y., Chen, X., Rao, F., Yu, X.L., Li, Y., Liu, D.: LORE: An infrastructure to support location-aware services. IBM Journal of Research and Devevelopment 48, 601–615 (Sep 2004)
11. Clementini, E., Sharma, J., Egenhofer, M.J.: Modelling topological spatial relations: Strategies for query processing. Computers and Graphics 18(6), 815–822 (1994)
12. Cohn, A.G., Renz, J.: Qualitative spatial representation and reasoning. In: Handbook of Knowledge Representation, Foundations of Artificial Intelligence, vol. 3, pp. 551–596. Elsevier (2008)
13. Egenhofer, M.J., Franzosa, R.D.: Point-set topological spatial relations. International Journal of Geographical Information Systems 5(2), 161–174 (1991)
14. Egenhofer, M.J., Franzosa, R.D.: On the equivalence of topological relations. International Journal of Geographical Information Systems 8(6), 133–152 (1994)
15. Egenhofer, M.J., Herring, J.R.: Categorizing binary topological relations between regions, lines, and points in geographic databases. Tech. rep., Department of Surveying Engineering, University of Maine (1990)

16. Eugster, P., Garbinato, B., Holzer, A.: Location-based publish/subscribe. In: Proceedings of the Fourth IEEE International Symposium on Network Computing and Applications. pp. 279–282 (2005)
17. Greene, D.H., Yao, F.F.: Finite-resolution computational geometry. In: Proceedings of the Annual IEEE Symposium on Foundations of Computer Science. pp. 143–152 (1986)
18. Guting, R.H., Schneider, M.: Realms: A foundation for spatial data types in database systems. In: Abel, D., Ooi, B.C. (eds.) Advances in Spatial Databases, pp. 14–35. No. 692 in Lecture Notes in Computer Science, Springer-Verlag (1993)
19. Guting, R.H., Schneider, M.: Realm-based spatial data types: The ROSE algebra. The International Journal on Very Large Data Bases 4(2), 243–286 (Apr 1995)
20. Guttman, A.: R-trees: A dynamic index structure for spatial searching. In: Proceedings of the ACM SIGMOD International Conference on Management of Data. pp. 47–57 (1984)
21. Hoffmann, C.M.: The problems of accuracy and robustness in geometric computation. Computer 22, 31–40 (Mar 1989)
22. Hu, S.Y., Wu, C., Buyukkaya, E., Chien, C.H., Lin, T.H., Abdallah, M., Jiang, J.R., Chen, K.T.: A spatial publish subscribe overlay for massively multiuser virtual environments. In: Proceedings of the International Conference on Electronics and Information Engineering. pp. 314–318 (2010)
23. Laszlo, M.J.: Computational Geometry and Computer Graphics in C++. Prentice-Hall (1995)
24. Leung, H.K.Y., Burcea, I., Jacobsen, H.A.: Modeling location-based services with subject spaces. In: Proceedings of the Conference of the Centre for Advanced Studies on Collaborative Research. pp. 171–181. Toronto, Canada (2003)
25. Lin, K.I., Jagadish, H., Faloutsos, C.: The TV-tree: An index structure for high-dimensional data. The International Journal on Very Large Data Bases 3(4), 517–542 (Oct 1994)
26. Papadias, D., Theodoridis, Y.: Spatial relations, minimum bounding rectangles, and spatial data structures. International Journal of Geographical Information Science 11, 111–138 (1997)
27. Ranganathan, A., Al-Muhtadi, J., Chetan, S., Campbell, R., Mickunas, M.D.: MiddleWhere: A middleware for location awareness in ubiquitous computing applications. In: Proceedings of the 5th ACM/IFIP/USENIX International Conference on Middleware. pp. 397–416. No. 3231 in Lecture Notes in Computer Science, Springer-Verlag (Oct 2004)
28. Sellis, T.K., Roussopoulos, N., Faloutsos, C.: The R-tree: A dynamic index for multi-dimensional objects. In: Proceedings of the 13th International Conference on Very Large Data Bases. pp. 507–518 (1987)
29. Xu, Z., Jacobsen, H.A.: Adaptive location constraint processing. In: Proceedings of the ACM SIGMOD International Conference on Management of Data. pp. 581–592 (Jun 2007)
30. Yan, T.W., Garcia-Molina, H.: Index structures for selective dissemination of information under the Boolean model. ACM Transactions on Database Systems 19(2), 332–364 (Jun 1994)