Compact Geometric Representation of Qualitative Directional Knowledge*

Zhiguo Long[a,b], Hua Meng[c,d,*], Tianrui Li[a,b], Sanjiang Li[e]

[a]*School of Information Science and Technology, Southwest Jiaotong University, China*
[b]*Institute of Artificial Intelligence, Southwest Jiaotong University, China*
[c]*School of Mathematics, Southwest Jiaotong University, China*
[d]*School of Civil Engineering, Southwest Jiaotong University, China*
[e]*Centre for Quantum Software and Information, FEIT, University of Technology Sydney, Australia*

Abstract

To effectively and efficiently deal with large-scale spatial data is critical for applications in the age of information technology. Compact representation of spatial knowledge is one of the emerging research techniques that contribute to this capability. In this article, we consider the problem of compactly representing qualitative directional relations between extended objects, modelled in the Cardinal Direction Calculus (CDC) of Goyal and Egenhofer. For a large dataset of regions, this approach first constructs a simplified geometry for each region, which preserves CDC relations between regions, and then represents each simplified geometry compactly, so that the storage size is small while retrieving CDC relations from the representation is still reasonably fast. More specifically, the method called *necessary cut* is used to construct simple geometries, and the two methods, viz. the *polygon representation* and the *rectangle representation*, are devised to compactly represent the constructed geometries in cubic time w.r.t. the size of the corresponding simple geometry. Theoretical analyses demonstrate that the two representations, especially the rectangle representation, are promising to have small storage size. Moreover, our empirical evaluations on

*Accepted version. Formal version available at: `https://doi.org/10.1016/j.knosys.2020.105616`

*Corresponding author
Email address: `menghua@swjtu.edu.cn` (Hua Meng)

real-world datasets show that, for each dataset the new approach can produce a rectangle representation that has dominant performance against the state of the art techniques in reducing the storage size of the relations, while the average efficiency of retrieving CDC relations based on the rectangle representation is about the same as the fastest method in the literature.
Keywords: qualitative spatial representation, Cardinal Direction Calculus, compact representation

1. Introduction

With the development of information technologies, more and more spatial data are being collected for use in geographical information systems (GISs), location based services, and many other spatial related applications [1, 2]. Qualitative spatial relations, such as *northwest*, is one of the most important information in such applications, where answering queries about these relations is a central function.

Previously, especially in the field of spatial reasoning [3, 4, 5, 6], qualitative spatial relations between objects in a spatial dataset are often represented as a *complete* constraint network with nodes being the objects and labels on the edges being the relations. For several important applications, this complete network representation is inefficient to process qualitative spatial information. Consider for example the task of topological adjustment of geometric data to satisfy a set of predefined spatial relations [7]. The efficiency of the topological adjustment algorithm is strongly related to the number of spatial relations explicitly given, and thus a complete network would not be ideal for representing these relations. Another example is the task for finding a set of objects matching a query specified by a set of spatial relations [8]. Clearly, the smaller the set of specified relations is, the more efficient the matching is. The third example is the task for retrieving a certain kind of spatial relation between two given objects. A complete network of relations that is large in size definitely prohibits the representation being stored in quick-access storage and leads to inefficiency

of query answering [9, 10, 11]. Therefore, it is crucial to find more compact representations than the complete network representation.

In current spatial knowledge systems as well as many other spatial-related applications, it is standard to use the geometries of spatial objects stored in the spatial database directly to calculate these relations, *every time* when a relation is needed. In those systems, the geometry of a region is represented as a complex polygon possibly with holes and multiple components, and the number of geometries for a set of spatial objects usually scales linearly with the number of objects. Nevertheless, the storage of these geometries could still take a large amount of space, as sometimes the number of vertices might be quite large (e.g., we encountered regions with more than $140,000$ vertices in the experiments). This in turn makes calculating qualitative spatial relations less efficient, because the time needed for calculating relations from geometries is proportional to the number of vertices of these geometries. Furthermore, exposing original geometries could lead to security and privacy issues, e.g., attackers might exploit the boundary information and coordinates of some important regions to initiate precise strikes, and people or companies might not want to share precise geometry data to services that only need to know the qualitative relations between spatial objects. To protect privacy and to ensure security, one could provide some kind of "obfuscation" [12] that preserves qualitative spatial relations, rather than the original geometries with accurate coordinates.

Direction relations between extended spatial objects like countries and regions are important spatial knowledge. While many existing direction relation models approximate extended spatial objects as simple objects like points [13] or rectangles [14], the Cardinal Direction Calculus (CDC) [15] represents the direction of a primary object to a reference object by only approximating the reference object by a rectangle (cf. Figure 2). That is, the exact geometry of the primary object is used in the representation. This makes the model very expressive and can distinguish 511 different basic direction relations. Moreover, CDC relations conform closely to human cognition. In the past two decades, the CDC has attracted growing interests of researchers from AI and GIS, see

e.g., [4, 5, 16, 17, 18, 19]. While most of these works focus on how to solve CDC constraint networks, Fogliaroni [9] and Long et al. [10] considered the compact representation of special CDC relations.

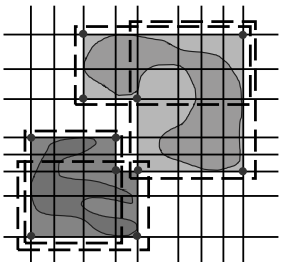


Figure 1: Illustrative example for the proposed methods.

In this paper, we proposed a new approach for compactly representing CDC relations that can work well for *all* basic CDC relations. Precisely, given a set of regions, we proposed a novel geometric representation for the CDC relations between the regions in this set, which associates each region with a geometry that is then compactly represented with a set of simpler geometries. Whenever needed, the CDC relation between any two regions can be easily recovered by calculations over the corresponding simple geometries. We call this the *compact geometric representation* of the CDC relations of the spatial dataset. For example, consider the two regions shown in Figure 1. These two regions are associated with two new geometries consisting of shadowed grid cells, where the CDC relation between the two regions can be obtained by checking the CDC relation between the two new geometries. For each of the two new geometries, we can further represent it with either polygons, given by the black dots in the figure, or rectangles, given by the dashed lines, so that the polygons or rectangles cover the same area of the geometry. It can be seen that the number of black dots or the number of rectangles is much smaller than the number of grid cells. By applying this to all of the regions in a dataset while ensuring the CDC relations between regions can be correctly obtained from the corresponding polygons or rectangles, we obtain a compact geometric representation of the CDC relations of the whole dataset. To this end, the method *necessary cut* is

used to associate each region with a geometry, and two methods, viz. *rectangle representation* and *polygon representation*, are devised to compactly represent those geometries. It turns out that the geometric representation obtained with these methods could have a much smaller storage size than the original geometries and the complete constraint network representation, and the correct cardinal direction relations can also be efficiently extracted from the rectangle representation. Furthermore, the new approach essentially provides an elegant obfuscation and thus could help to protect data privacy. As confirmed by the empirical evaluations on real-world datasets, the rectangle representation approach is significantly superior to the state of the art techniques for compact representation in storage size, and is competitive in the efficiency of answering queries.

In the next section, we introduce some preliminary concepts and briefly review related works. In Section 3, we describe our new compact representation approach in detail. The performance of the new approach is empirically evaluated in Section 4 and Section 5 discusses some implications and limitations of the new approach. Section 6 concludes the article.

2. Preliminaries and Background

The Euclidean plane $\mathbb{R}^2$ (or, the plane, for short) can be considered as a set of points. In this paper, a subset a of the plane is called a *region* if it is nonempty and $a = \overline{a^\circ}$, i.e., it is regular closed. Suppose x-axis and y-axis are the two axes of a Cartesian coordinate system on the plane (see Figure 2).

For a region a, let

$$x^-(a) = \inf\{x : (\exists y)(x, y) \in a\}, x^+(a) = \sup\{x : (\exists y)(x, y) \in a\},$$
$$y^-(a) = \inf\{y : (\exists x)(x, y) \in a\}, y^+(a) = \sup\{y : (\exists x)(x, y) \in a\}.$$

The *minimum bounding rectangle* (MBR) of a is the set $\mathrm{mbr}(a) := [x^-(a), x^+(a)] \times [y^-(a), y^+(a)]$.

A rectangle in the plane is *axis-aligned* if its sides are parallel to the x- and y-axes in the plane. Specifically, $\mathrm{mbr}(a)$ is the smallest axis-aligned rectangle

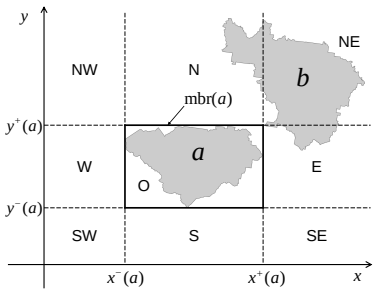


Figure 2: Illustration of MBR and CDC tiles.

that contains a (see Figure 2). In this paper, all the considered rectangles are assumed to be axis-aligned. It is easy to see that an axis-aligned rectangle r can be uniquely determined by the 4-tuple $(x^-(r), x^+(r), y^-(r), y^+(r))$.

By extending the four sides of $\text{mbr}(a)$, we divide the plane into nine tiles, named by $\mathsf{NW}, \mathsf{N}, \mathsf{NE}, \mathsf{W}, \mathsf{O}, \mathsf{E}, \mathsf{SW}, \mathsf{S}, \mathsf{SE}$, respectively (see Figure 2). The Cardinal Direction Calculus (CDC), proposed by Goyal and Egenhofer [15] (1997), is a relation model that encodes the directional relationship between a primary region b and a reference region a by exploiting the nine tiles determined by a. A *basic* CDC relation $\delta(b, a)$ is defined as $\{\chi \mid \chi(a) \cap b^\circ \neq \emptyset\}$, where χ is one of the nine *tile names* $\{\mathsf{NW}, \mathsf{N}, \mathsf{NE}, \mathsf{W}, \mathsf{O}, \mathsf{E}, \mathsf{SW}, \mathsf{S}, \mathsf{SE}\}$, and $\chi(a)$ is one of the nine tiles determined by a. For example, in Figure 2, the CDC relation $\delta(b, a)$ is $\{\mathsf{N}, \mathsf{NE}, \mathsf{E}\}$. Note that for any two regions a and b, $\delta(b, a) = \delta(b, \text{mbr}(a))$.

Given a set of regions in the plane $D = \{a_1, \ldots, a_n\}$, the CDC relations between the regions can be represented in different ways. One naive representation is to record the CDC relation $\delta(a_i, a_j)$ in a complete CDC constraint network

$$\Delta(D) = \{v_i \delta_{ij} v_j : \delta_{ij} = \delta(a_i, a_j), 1 \leq i \neq j \leq n\},$$

where each δ_{ij} is a basic CDC relation. We often call a network like this an *atomic* network. As established in previous works [9, 10, 11], it has several disadvantages:

- when the number n of regions becomes very large, it requires a large storage space (actually $\Theta(n^2)$);

- a large storage space may slow down query answering, as quick-access storage could be infeasible;
- it does not provide any geometric intuition.

Therefore, there is a need for more compact representations.

2.1. Related work

The challenging problem of compactly representing qualitative relations has been addressed by several researchers. Related work on this problem can be divided into two approaches. The first approach is based on qualitative reasoning and aims to simplify the structure of qualitative constraint networks. The second one makes use of geometric information of spatial objects, and focuses on finding a representation of the qualitative relations such that queries about the relations can be efficiently answered.

Representatives of the first approach include [7, 20, 21]. Given a qualitative constraint network Δ of topological RCC8 relations, Egenhofer and Sharma [20] noted that there exists a "smallest" subset of Δ that has the same set of solutions as Δ and contains no *redundant* constraints, where a constraint C is redundant in Δ if $\Delta \setminus \{C\}$ has the same set of solutions as Δ. Later, Rodríguez et al. [8] used this idea to develop a query pre-processing technique, such that the number of topological relations expressed in a query is minimised. However, they only discussed the case of basic topological relations. In a more recent paper, Wallgrün [7] considered the problem of topological adjustment, i.e., how to adapt geometric data to satisfy given topological relations. He found out that removing redundant topological constraints can be beneficial for the optimisation process that is used to solve the problem of topological adjustment. He further proposed two algorithms for removing redundant constraints, but neither can find a "smallest" subset without redundant constraints. Recently, Li et al. [21] showed that, for any constraint network Δ defined over a distributive subalgebra of RCC8, there exists a unique "smallest" subset, called the prime subnetwork of Δ, which contains no redundant constraints and has the same set

of solutions as Δ. They further proposed a cubic time algorithm for constructing the prime subnetwork.

When considering compact representation for answering queries about qualitative relations, removing redundant relations would not be ideal any more, as retrieving removed relations requires qualitative reasoning which is time-consuming. This leads to the second approach of compact representation. Fogliaroni [9] proposed a framework to reduce the number of stored relations while support efficient queries about the qualitative relations. The idea is to divide objects into clusters and make use of *clustering relations*, such as the disjoint topological relation and the *northeast* cardinal direction relation. In this way, only clustering relations between clusters and non-clustering relations between objects need to be stored, and practically the number of stored relations is reduced by a large amount. Later, Long et al. [10] found that some topological relations or CDC relations between objects can be obtained from relations between their MBRs. Based on that observation, they developed an MBR-based representation to compactly represent either the topological relations or the CDC relations, and showed that this representation is more compact than the clustering-based representations while answers queries efficiently.

Long et al. [11] devised an algorithm that generates rectangles for objects, such that the topological relations between them can be efficiently reconstructed from the rectangles. They illustrated that the storage size of the generated rectangles can be much smaller than directly storing all the relations. This algorithm represents a spatial object with a set of rectangles, which is similar to the rectangle representation proposed in this paper. There are two major differences: first, the rectangle representation for CDC relations forms a *cover* of an approximation of the original geometry, while the one for RCC8 does not; second, the CDC relation between two objects is recovered by taking the union of the CDC relations between their corresponding rectangles, while, in [11], the topological relation between two objects is the same of that between the *first* pair of rectangles generated for two objects.

Karlsen and Giese [22] proposed a general framework of using Bintrees to

compactly represent qualitative spatial information for answering queries, and they have shown that it can reduce storage size and query answering time for topological relationships overlap and containment. Recently, Karlsen and Giese [23] developed a more general framework that can encode different qualitative information more compactly with Bintrees, including RCC8 relations. However, the applicability of that framework to CDC relations is not clear.

From the above we can see that the compact representation problem was indeed considered before, but all these works focused on topological or very special CDC relations, and no work considered how to compactly represent *all* kinds of basic CDC relations while retrieving them efficiently.

3. The New Compact Representation Approach

3.1. Geometric representation

Let U be the set of all regions in the plane and $D = \{a_1, \ldots, a_n\} \subseteq U$ be a set of regions that correspond to some spatial objects. An important task related to D is to answer a type of *queries* that are to decide the CDC relation between two given spatial objects. For the collection of the CDC relations between all pairs of D (i.e. the atomic CDC constraint network $\Delta(D)$ induced by D), we can have a *geometric representation*, so that when needed the CDC relation between any two objects can be computed from the geometries.

Definition 3.1. Suppose f is a mapping from $D = \{a_1, \ldots, a_n\}$ to U. Let $f(D) = \{f(a_i) : a_i \in D\}$. If $f(D)$ is a solution of the CDC atomic network $\Delta(D)$, i.e., $\forall a_i, a_j \in D$, $\delta(f(a_i), f(a_j)) = \delta(a_i, a_j)$, then we say $f(D)$ is a *geometric representation* of $\Delta(D)$.

To obtain a geometric representation, it is not necessary to have a fully pre-computed $\Delta(D)$. For example, D itself is always a geometric representation of $\Delta(D)$, even when $\Delta(D)$ is not known yet. From a geometric representation of $\Delta(D)$, to answer queries, we can compute the CDC relations by using the regions $f(a_i)$. However, as has been noticed in [9, 10, 11], computing the

relations by using the original regions in D could take too much time for answering the kind of queries about deciding the CDC relations between regions. The reason is that these regions are represented as complex polygons and may contain many vertices, and that the time to calculate $\delta(a_i, a_j)$ is related to the number of vertices of the polygons. Therefore, for quick calculation of the CDC relation $\delta(f(a_i), f(a_j))$, it will be desirable that each $f(a_i)$ has a simple shape, e.g., having a small number of vertices or all its edges being parallel to the axes. More specifically, to find an f that gives a good geometric representation $\{f(a_i) : a_i \in D\}$ of $\Delta(D)$, we consider the following four aspects:

- faithfulness: $(\forall a_i)(\forall a_j)[\delta(a_i, a_j) = \delta(f(a_i), f(a_j))]$;
- construction efficiency of $f(a_i)$;
- the storage space required for $\{f(a_i) : a_i \in D\}$;
- efficiency of query answering based on $\{f(a_i) : a_i \in D\}$.

Here, queries are restricted to the ones about deciding the CDC relations between regions. In the following, we describe our method for constructing a good geometric representation.

Generally, our idea is to first construct a grid $\mathcal{G}_D$ using the boundary lines of the MBRs of regions in D, and then remove those grid cells from the MBR of each region a_i if they violate the constraint involving a_i.[1] For example, consider the two regions in Figure 2. $\mathrm{mbr}(b)$ intersects the interior of the O-tile of a (i.e. $\mathrm{mbr}(a)$), while b itself does not. Therefore, $\delta(b, a) = \{\mathsf{N}, \mathsf{NE}, \mathsf{E}\} \neq \{\mathsf{N}, \mathsf{NE}, \mathsf{E}, \mathsf{O}\} = \delta(\mathrm{mbr}(b), a)$. To avoid the violation of the constraint $\delta(b, a)$, any grid cells in $\mathrm{mbr}(b)$ that intersects the interior of the O-tile of a should be removed. In this way, for each a_i, we obtain a new region $h(a_i)$ which is a polygon and all its edges are parallel to the x- or y-axis. We call a polygon like this *rectilinear* and call the mapping $h : a_i \mapsto h(a_i)$ the *necessary cut* of D (as it only removes cells that must be cut). We will see that h is a faithful geometric

[1] This technique was first introduced in [4] for solving CDC constraint networks.

representation of D. To increase its construction and query efficiency, we further propose methods for representing the mapped geometries more compactly.

3.2. The induced grid and necessary cut of D

Given a set of distinct x-coordinates $X = \{x_i \in \mathbb{R} : 1 \leq i \leq m\}$ and a set of distinct y-coordinates $Y = \{y_i \in \mathbb{R} : 1 \leq i \leq n\}$, there are m corresponding vertical lines and n corresponding horizontal lines. For convenience, suppose $x_j < x_k$ and $y_j < y_k$ for any $j < k$. Formally, the corresponding lines are defined as

$$L_x = \{(x_i, y) : \forall x_i \in X, \forall y \in \mathbb{R}\},$$
$$L_y = \{(x, y_i) : \forall y_i \in Y, \forall x \in \mathbb{R}\}.$$

The *grid* in the plane generated by $L = L_x \cup L_y$ is the set

$$\mathcal{G}_L = \{[x_i, x_{i+1}] \times [y_j, y_{j+1}] : \forall x_i, x_{i+1} \in X, \forall y_j, y_{j+1} \in Y\}.$$

Here each $c_{ij} = [x_i, x_{i+1}] \times [y_j, y_{j+1}]$ in $\mathcal{G}_L$ is called a *(grid) cell*. Note that the union of any subset of cells in $\mathcal{G}$ is actually a rectilinear polygon. We denote by $U_{\mathcal{G}} = \{\bigcup T : T \subseteq \mathcal{G}\}$ the set of rectilinear polygons formed by subsets of cells in $\mathcal{G}$.

Recall that the CDC relation between two regions is determined by the intersection of the interior of the primary region and the nine CDC tiles of the reference region (see Figure 2). This inspires us to consider the grid induced by the MBRs of all a in D. Let

$$X_D = \{x_i : \exists a \in D, (x^-(a) = x_i \vee x^+(a) = x_i)\},$$
$$Y_D = \{y_j : \exists a \in D, (y^-(a) = y_j \vee y^+(a) = y_j)\},$$

and

$$L_D = \{(x_i, y) : \forall x_i \in X_D, \forall y \in \mathbb{R}\} \cup \{(x, y_i) : \forall y_i \in Y_D, \forall x \in \mathbb{R}\}$$

be the collection of the boundary lines of the MBRs of regions in D. We write the grid generated by L_D as $\mathcal{G}_D$, i.e.

$$\mathcal{G}_D = \{[x_i, x_{i+1}] \times [y_j, y_{j+1}] : \forall x_i, x_{i+1} \in X_D, \forall y_j, y_{j+1} \in Y_D\},$$

where we suppose $x_i < x_{i+1}$ and $y_j < y_{j+1}$, $\forall x_i, x_{i+1} \in X_D$ and $\forall y_j, y_{j+1} \in Y_D$.

Figure 3 illustrates $\mathcal{G}_D$ for the dataset D of four regions. By extending the boundaries of the four MBRs (shaded rectangles), we obtain several horizontal lines and vertical lines which form L_D and thus $\mathcal{G}_D$. The rectangles bounded by the lines in L_D are grid cells.

Figure 3: The grid $\mathcal{G}_D$ generated by the dataset D of four regions.

Based on $\mathcal{G}_D$, we associate each a_i with a rectilinear polygon $h(a_i)$ by removing from $\text{mbr}(a_i)$ only those cells which violate the constraint of a_i to some a_j.

Definition 3.2 (Necessary cut). The *necessary cut* of D is the mapping h from D to $U_{\mathcal{G}_D} = \{\bigcup T : T \subseteq \mathcal{G}_D\}$ such that, for any $c_{kl} \in \mathcal{G}_D$, $c_{kl} \not\subseteq h(a_i)$ if and only if $c_{kl} \not\subseteq \text{mbr}(a_i)$ or there exists $j \neq i$ with $\delta(c_{kl}, a_j) \not\subseteq \delta(a_i, a_j)$.

We note that, for any grid cell $c_{kl} \in \mathcal{G}_D$ and any j, the CDC basic relation $\delta(c_{kl}, a_j)$ is a singleton, i.e., it contains only one of the nine tile names. Moreover, $c_{kl} \subseteq h(a_i)$ if and only if $c_{kl}^{\circ} \cap h(a_i) \neq \emptyset$. Also, we have $a_i \subseteq h(a_i) \subseteq \text{mbr}(a_i)$ for any $a_i \in D$.

Intuitively, the necessary cut removes from each $\text{mbr}(a_i)$ the grid cells that contradicts with the CDC relation $\delta(a_i, a_j)$ for some a_j. Consider the two regions $a, b \in D$ in Figure 4, where $\delta(a, b) = \{\mathsf{N}, \mathsf{NE}, \mathsf{E}\}$. Suppose $\text{mbr}(a)$ is partitioned into several cells by $\mathcal{G}_D$ as shown in the figure. To construct $h(a)$, the cells in $\text{mbr}(a)$ that are not shadowed will be removed when considering $\delta(a, b)$.

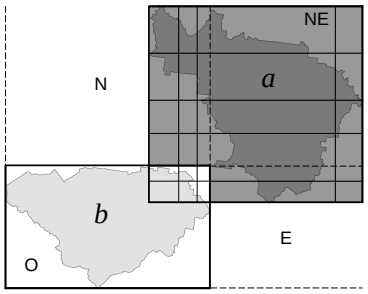


Figure 4: Illustration of necessary cut.

It is easy to see that the necessary cut maintains the MBRs and satisfies the faithfulness requirement.

Proposition 3.3 (By [4]). *For any* $a_i \in D$, $\mathrm{mbr}(a_i) = \mathrm{mbr}(h(a_i))$*; for any* $a_i, a_j \in D$, $\delta(a_i, a_j) = \delta(h(a_i), h(a_j))$.

It is worth noting that there exists some other mapping $f : D \to U_{\mathcal{G}_D}$ s.t. MBRs are maintained and the faithfulness requirement is satisfied, such as *regularisation* [4].

Figure 5 illustrates the necessary cut and regularisation of a region a from the administrative areas of Germany. From the figure, it can be seen that the light grey cells and dark grey cells together form a much simpler shape compared against the light grey cells alone. In fact, the simple shape corresponds to the necessary cut, and the complex one to the regularisation of a.

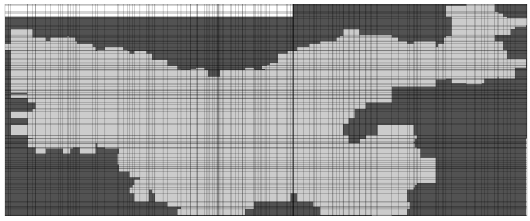


Figure 5: The regularisation of a (light grey coloured cells), and the necessary cut $h(a)$ of a (light grey and dark grey coloured cells), for a region a of the administrative areas of Germany.

As we can see, in the following sense, necessary cut actually gives the "roughest" approximation using grid cells for the regions of D while preserving their

CDC relations.

Proposition 3.4. *For any mapping f from D to $U_{\mathcal{G}_D}$ satisfying the requirements of $\delta(a_i, a_j) = \delta(f(a_i), f(a_j))$ and $\mathrm{mbr}(a_i) = \mathrm{mbr}(f(a_i))$ for any $a_i, a_j \in D$, we have that $f(a_i) \subseteq h(a_i)$ for any $a_i \in D$.*

Proof. See Appendix.

In the following discussions, we will focus on the necessary cut of D.

3.3. Calculation of the necessary cut of D

In order to calculate the necessary cut of D, i.e. $h(a_i)$ for each $a_i \in D$, one can use the $O(n^2)$ ($n = |D|$) time algorithm proposed in [4].

Here, we adopt that algorithm with a further improvement, i.e. by considering only the cells in $\mathrm{mbr}(a_i)$ for the calculation of each $h(a_i)$. To this end, we define some auxiliary matrices in a way similar to that in [4].

Let $n_x^i = |\{x^-(c_{kl}), x^+(c_{kl}) : c_{kl} \in \mathrm{mbr}(a_i)\}| - 1$ and $n_y^i = |\{x^-(c_{kl}), x^+(c_{kl}) : c_{kl} \in \mathrm{mbr}(a_i)\}| - 1$. Suppose c_{kl} is the cell at the k-th row and l-th column of the grid, and $c_{k_0 l_0}$ is the top-left cell in $\mathrm{mbr}(a_i)$. Define $B_i(a)$ to be a matrix of dimension $n_y^i \times n_x^i$, for a set of cell $a \subseteq \mathrm{mbr}(h(a_i))$ as follows[2]:

$$B_i(a)[k,l] = \begin{cases} 1, & \text{if } c_{k_0+k, l_0+l} \in a; \\ 0, & \text{otherwise.} \end{cases} \tag{1}$$

Define

$$Q_i = \sum \{B_i(\mathrm{mbr}(a_i) \cap \chi(a_j)) : a_i^\circ \cap \chi(a_j) = \emptyset \text{ and } j \neq i\}. \tag{2}$$

Note that $Q_i[k,l] > 0$ means c_{k_0+k, l_0+l} is a cell in $\mathrm{mbr}(a_i) \cap \chi(a_j)$ where $a_i^\circ \cap \chi(a_j) = \emptyset$ for some $j \neq i$. As a result, we have the following conclusion.

Proposition 3.5. *The localized matrix $B_i(h(a_i))$ for $h(a_i)$ can be determined as follows*

$$B_i(h(a_i))[k,l] = \begin{cases} 1, & \textit{if } Q_i[k,l] = 0; \\ 0, & \textit{otherwise.} \end{cases} \tag{3}$$

[2]Here, the row and column indices of a matrix starts from 0.

Then by adapting the $O(n^2)$ algorithm in [4], $h(a_i)$ can be calculated in $O(n_i^2 + nt_i)$ time, where $n_i = \max(n_x^i, n_y^i)$ and t_i is the bound of the time needed for deciding if $a_i^\circ \cap \chi(a_j) \neq \emptyset$ for each $j \neq i$ and each $\chi(a_j)$, which is in fact $O(v_i \log v_i)$ (v_i is the number of vertices in a_i) [24]. The adapted algorithm is shown in Algorithm 1.

Algorithm 1: Calculating $h(a_i)$, adapted from [4].

Input: A set $D = \{a_1, \ldots, a_n\}$ of regions and some specific $1 \leq i \leq n$.

Output: $h(a_i)$

```
1  T ← a zero matrix of dimension n_y^i × n_x^i, with c_{k_0 l_0} being the top-left
     cell of it;
2  foreach j and j ≠ i do
3      foreach χ do
4          if a_i° ∩ χ(a_j) ≠ ∅ then
5              {c_{kl} : k_0 + k_1 ≤ k ≤ k_0 + k_2, l_0 + l_1 ≤ l ≤ l_0 + l_2} ←
                 mbr(a_i) ∩ χ(a_j);
6              foreach k ∈ [k_1, k_2] do
7                  T ← T[k, l_1] + 1;
8                  if l_2 + 1 ≤ n_x^i then
9                      T ← T[k, l_2 + 1] − 1;
10 foreach k ∈ [1, n_y^i] do
11     Q_i[k, 1] = T[k, 1];
12 for l = 2 : n_x^i do
13     foreach k ∈ [1, n_y^i] do
14         Q_i[k, l] = T[k, l] + T[k, l − 1];
15 h(a_i) ← mbr(a_i);
16 foreach c_{k_0+k, l_0+l} ∈ h(a_i) do
17     if Q_i[k, l] ≠ 0 then
18         h(a_i) ← h(a_i) \ {c_{k_0+k, l_0+l}};
19 return h(a_i).
```

We shall note that the construction time of $h(a_i)$ is not our major concern here. Instead, we care about the query answering time using the constructed $h(a_i)$. If the construction time is reasonably fast, then with the number of queries increasing continuously, eventually the gain in query answering time will be superior to the construction cost. In fact, the construction time is indeed reasonably fast. For example, for most of the datasets with 400 to $3,500$ regions used in our experiments, constructing all $h(a_i)$ takes less than 2 minutes (the shortest time is about 5 seconds). For larger datasets, such as one with $36,702$ regions, constructing all $h(a_i)$ takes about 1.4 hours, which is still reasonable as the construction of $h(a_i)$ happens offline.

3.4. Compact representation of $h(a_i)$

After obtaining $h(a_i)$, the next problem is how to represent it in small storage size and to answer queries efficiently with the representation. Note that currently $h(a_i)$ is represented as a set of grid cells whose number might be very large, e.g. $O(|D|^2)$. As an example, $h(a)$ in Figure 5 has 13,508 grid cells, while there are only 434 regions in D. It would be inefficient for either storage or answering query.

3.4.1. The polygon representation

As the shape of $h(a_i)$ is actually a polygon, a natural method is to store the vertices of the polygon in a standard way, e.g., the "OpenGIS Implementation Specification for Geographic Information"[3].

Usually, a polygon is represented as two sets of rings of vertices. The first set contains only one ring, viz. the exterior boundary, and the other set contains none or several rings, viz. the interior boundaries or *holes*. Therefore, to identify polygons from $h(a_i)$, we need to identify these rings. Algorithm 2, with the help of Algorithm 3, constructs the set of polygons determined by $h(a_i)$. This set of polygons are used to compactly represent $h(a_i)$.

[3] `http://www.opengeospatial.org/standards/sfa`

Definition 3.6. The set of polygons that are constructed by Algorithm 2 is called the *polygon representation* of $h(a_i)$.

In order to minimize the number of vertices to store, we only use *corner vertices*, i.e. the vertices at which the angle formed by the edges is not 180°. Figure 6 shows the cells of $h(a_i)$ for some a_i and the polygon representation of $h(a_i)$. The corner vertices of the polygon are black points and the edges are bold lines. There are 36 cells in $h(a_i)$ and thus 144 vertices of the cells, while there are only 20 vertices in the polygon representation.

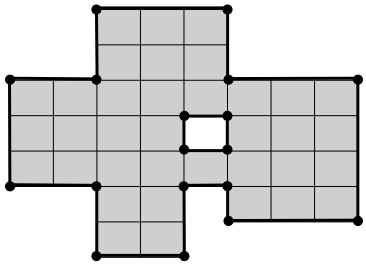


Figure 6: Illustration of a polygon representation.

In particular, Line 1 of Algorithm 2 identifies all the vertices by scanning the cells of $h(a_i)$ vertically and horizontally. This procedure will order the vertices with the same x-coordinate vertically and the vertices with the same y-coordinate horizontally. Such information will be useful for identifying the rings. Each loop from Line 3 to Line 12 finds a polygon formed by the vertices in V. Line 4 exploits Algorithm 3 to identify the exterior boundary. After the exterior boundary is found, all the inner vertices of the polygon formed by the exterior boundary are identified and stored in V'. Then the loop from Line 7 to Line 10 identifies all the rings formed by the vertices in V', which are the holes.

Algorithm 3 identifies a ring of vertices for a given vertex set V. As shown in Line 2, the starting vertex v_0 is the one that has the largest y-coordinate on the left most side of the given set of vertices. If the vertices on the ring is assumed to be in clockwise order, then the next vertex after v_0 must have the same y-coordinate as v_0 and is on the right side of v_0. By this observation,

Algorithm 2: An algorithm for identifying polygons of $h(a_i)$.

Input: $h(a_i)$, a set of grid cells representing the digitalization of a region a_i.

Output: P, the set of polygons determined by $h(a_i)$.

```
1  V ← IdentifyVertices(h(a_i));
2  P ← ∅;
3  while V ≠ ∅ do
4  |  R ← IdentifyRing(V);
5  |  V' ← IdentifyInnerVertices(V, r);
6  |  R ← ∅;
7  |  while V' ≠ ∅ do
8  |  |  R' ← IdentifyRing(V');
9  |  |  R ← R ∪ {R'};
10 |  P ← P ∪ {(R, R)};
11 return P.
```

Line 3 identifies the next vertex of v_0. The loop from Line 6 to Line 16 identifies vertices in clockwise order. Particularly, if the previous vertex v_0 was identified by searching horizontally, then the current searching for v_1 will be done vertically and vice versa, because each vertex is a corner vertex, while whether the search goes up or down, or right or left, depends on the order of v_0 in the list of vertices with the same x-coordinate or the same y-coordinate as v_0. This is done in Lines 9 to 15. All the vertices that are identified will be removed from V in Line 17.

Proposition 3.7. *The time complexity of Algorithm 2 is $O(m^3)$, where m is the number of cells in $h(a_i)$.*

Proof. See Appendix.

In practice, the time needed for Algorithm 2 would be much shorter than what is claimed in the above proposition. The reasons are:

Algorithm 3: IdentifyRing(V)

Input: V, a set of vertices.

Output: R, a list of vertices in V that form a ring.

```
1  R ← ∅;
2  v_0 ← argmax_y argmin_x V;
3  v_1 ← argmin_x {v ∈ V' : x(v) > x(v_0), y(v_1) = y(v_0)};
4  Flag ← true;
5  R ← R ∪ {v_0};
6  while v_1 ≠ v_0 do
7      v_0 ← v_1;
8      R ← R ∪ {v_0};
9      if Flag then
10         v_1 ← SearchVertically(V);
11         Flag ← false;
12     else
13         v_1 ← SearchHorizontally(V);
14         Flag ← true;
15 V ← V \ R;
16 return R.
```

1. $|V|$ would be much smaller than m.
2. The number of iteraions of the outer loop would be much smaller than $|V|$, as each iteration identifies a polygon and removes its vertices from V. The number of polygons will be much smaller than the number of vertices.
3. The execution of Line 5 would take much less time, e.g., $O((|V|-|R|)\cdot|R|)$ because only the vertices not in R need to be checked.

Actually, our experiment on a real-world dataset with $3,145$ regions and average value of $m \approx 14,000$ showed that the average time for each $h(a_i)$ is less than 1 ms on a computer with an Intel® Core™-i5 3.1 GHz CPU.

However, calculating CDC relations between polygons might not be easy, which would result in less efficient performance for answering queries. On the other hand, calculating CDC relations between rectangles would be much easier. Therefore, if we can represent $h(a_i)$ using rectangles, then answering queries would be more efficient than uing polygons. In the following, we consider an alternative method for representing $h(a_i)$ by using rectangles.

3.4.2. The rectangle representation

Note that $h(a_i)$ can be considered as one or more rectilinear polygons. A rectilinear polygon can be *covered* by a set of axis-aligned rectangles. More formally, given a rectilinear polygon P, a *rectangle cover* of P is a set C of axis-aligned rectangles s.t. $\forall r \in C$ we have $r \subseteq P$ and $\bigcup C = P$. In a rectangle cover, rectangles are allowed to overlap. We make use of a rectangle cover to represent $h(a_i)$.

Definition 3.8. A rectangle cover of $h(a_i)$ is called a *rectangle representation* of $h(a_i)$.

Figure 7 shows the cells of $h(a_i)$ for some a_i and a rectangle representation of $h(a_i)$. The edges of the rectangles are bold lines. There are 36 cells in $h(a_i)$ while only 5 rectangles in the rectangle representation. Using rectangle cover has the following advantages:

- A rectangle is determined by four coordinates and it can contain a large number of grid cells in $h(a_i)$. Therefore, it is promising for saving the storage space of $h(a_i)$.
- Calculating CDC relations between rectangles is very simple, and thus it could be efficient to retrieve the original CDC relations using rectangle representation.

An *optimised* rectangle representation of $h(a_i)$ is a rectangle cover of $h(a_i)$ with a minimum number of rectangles. However, finding an optimised rectangle representation is not an easy task. Actually, it is a classic problem to find a

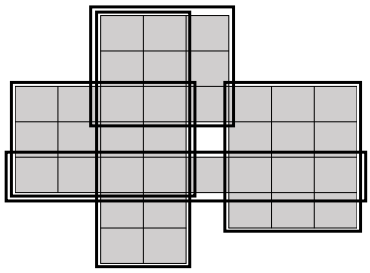


Figure 7: Illustration of a rectangle representation. The rectangles are drawn with some offsets for clearness.

rectangle cover with a minimum number of rectangles for a rectilinear polygon, and this problem is NP-complete in general [25].

Many approximation algorithms and special cases have been considered in the literature. Particularly, when the rectilinear polygon is *x- or y-orthogonally convex*, i.e. the intersection between any line parallel to x- or y-axis and the polygon is an empty set, a point, or a single segment, there exists an exact algorithm [26] for finding a rectangle cover of minimum number; when the rectilinear polygon does not have holes, there is an approximation algorithm with *approximation factor* 2, i.e. it guarantees to find a rectangle cover of number less than twice the minimum number; when the rectilinear polygon is allowed to have holes, Kumar and Ramesh [27] gave an algorithm with approximation factor $O(\sqrt{\log k})$, where k is the number of horizontal grid edges of the polygon. Here, we adopt the algorithm in [27], as it has a good performance guarantee while being easy to implement. The algorithm for identifying a rectangle representation of $h(a_i)$ is shown in Algorithm 4.

The idea of the algorithm is simple. It scans each column of cells in $h(a_i)$ to find *column strips*. A (column) strip is a maximal set of vertically consecutive cells in $h(a_i)$. The method ExpandStrip(s) then expands a strip s horizontally to form a maximal rectangle that is contained in $h(a_i)$. The returned rectangle representation is the set of maximal rectangles found by ExpandStrip(s), with repeated ones removed. Franzblau [28] proposed a similar sweep-line algorithm that expands row strips.

Algorithm 4: Identifying a rectangle representation of $h(a_i)$.

Input: $h(a_i)$

Output: K, a set of axis-aligned rectangles covering $h(a_i)$.

```
K ← ∅;
foreach column c of cells in h(a_i) do
    foreach strip s in c do
        r ← ExpandStrip(s);
        K ← K ∪ {r};
return K.
```

Proposition 3.9. *The time complexity of Algorithm 4 is $O(m^3)$, where m is the number of cells in $h(a_i)$.*

Proof. See Appendix.

In practice, the time needed for Algorithm 4 would be much less than $O(m^3)$, as the number of columns would be much smaller than m, and the number of strips in each column and the number of cells in each strip would also be much smaller than m. Our experiments on a real-world dataset with $3,145$ regions and average value of $m \approx 14,000$ showed that on average it takes less than 2 ms per region on a computer with an Intel® Core™-i5 3.1 GHz CPU.

3.4.3. Storage space analysis

To consider the storage space of the two representations, for the polygon representation, we count the number of coordinates required for the vertices as the size of it; for the rectangle representation, we count the number of coordinates required for the rectangles as the size of it. Let $|V|$ be the number of vertices in the polygon representation, and p^* be the number of rectangles in the rectangle representation. Then the size of the polygon representation is $2|V|$ and the size of the rectangle representation is $4p^*$. For the relation between the sizes of the polygon representation and the rectangle representation, we have the following conjecture.

Conjecture 3.10. *The size of the polygon representation is larger than or equal to the size of the rectangle representation, i.e.* $2|V| \geq 4p^*$.

In the following, we show this conjecture is actually true. Before proving the conjecture, we introduce some auxiliary concepts adapted from [28]. A rectilinear polygon has *convex* vertices and *concave* vertices. A vertex is said to be concave if the inner angle at this vertex is 270°, otherwise it is convex. There are two types of convex vertices. One is *normal convex vertex*, which is the intersection of exactly two edges of the polygon forming a 90° inner angle. The other one is *degenerated convex vertex*, which is the intersection of two pairs of edges, forming two 90° inner angles. In Figure 8, p_1 is a normal convex vertex and p_4 is a degenerated convex vertex. As a result, in the vertex list of a rectilinear polygon, a normal vertex appears exactly once, a degenerated vertex appears exactly twice.

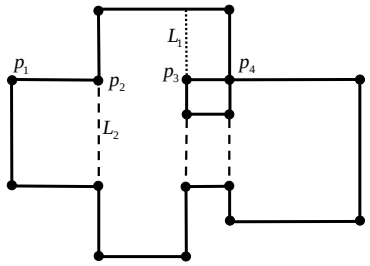


Figure 8: Illustration of different vertices and chords in a polygon.

The concave vertices can be classified with the help of *vertical chords*. A (vertical) chord is a line segment between two points on the boundary of a polygon, on which at least one endpoint is a concave vertex and the other points are in the interior of the polygon. By extending the vertical edges of a polygon to infinite lines, the polygon is *partitioned* into several non-overlapping rectangles and chords are created. For example, Figure 8 shows all the vertical chords of the polygon by dotted lines and broken lines. Generally, in a polygon there are two types of chords:

- type 1 chord: exactly one endpoint is a concave vertex.

- type 2 chord: both endpoints are concave vertices.

In Figure 8, L_1 is a type 1 chord and L_2 is a type 2 chord, and the other type 2 chords are represented by broken lines. We call the concave vertex on a type 1 chord *a type* 1 *vertex*, and a concave vertex on a type 2 chord *a type* 2 *vertex*. In Figure 8, p_3 is a type 1 vertex and p_2 is a type 2 vertex.

For the number of vertices and the rectangles obtained by partitioning a polygon through expanding vertical edges, [28] gave the following lemma.

Lemma 3.11 (By [28]). *Suppose $h(a_i)$ is a rectilinear polygon. Let μ_1 and μ_2 be the number of normal convex vertices and degenerated convex vertices, respectively. Let c_1 and c_2 be the number of type* 1 *and type* 2 *vertical chords, respectively. Let p be the number of rectangles obtained by the partition of $h(a_i)$ into rectangles as discussed before. Then $\mu_1 + 2\mu_2 + 3c_1 + 2c_2 = 4p$.*

By the above lemma, we have the following relation between the number of vertices and the number of rectangles obtained in Algorithm 4.

Proposition 3.12. *Let p^* be the number of rectangles obtained in Algorithm 4. $|V| = \mu_1 + 2\mu_2 + c_1 + 2c_2 \geq 2p \geq 2p^*$.*

Proof. See Appendix.

Our experiments on real-world datasets showed that the number of vertices in the polygons identified by Algorithm 2 for $h(a_i)$ is about three times as the number of rectangles identified in Algorithm 4. Therefore, both in theory and in practice, the rectangle representation is superior to the polygon representation in storage efficiency.

Although theoretically we cannot ensure good performance of the two new representations, we deem that they are promising in practice, especially the rectangle representation. The following observations give some hints about the potential of the rectangle representation in storage efficiency.

A region is *connected* if it cannot be divided into two non-empty disjoint closed sets in the plane.

Proposition 3.13. *Suppose D consists of only connected regions. If* $[\mathrm{mbr}(a_i)]^\circ \cap [\mathrm{mbr}(a_j)]^\circ = \emptyset$, *then* $\forall \chi$, $[\mathrm{mbr}(a_i)]^\circ \cap \chi(a_j) \neq \emptyset$ *iff* $a_i^\circ \cap \chi(a_j) \neq \emptyset$.

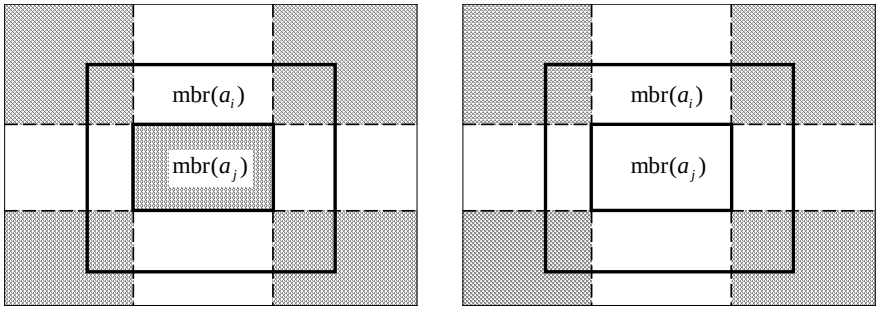
PROOF. See Appendix.

For connected regions, the above proposition means that a cell in $\mathrm{mbr}(a_i)$ will not be removed for constructing $h(a_i)$ because of a_j, given that the MBRs of a_i and a_j are not intersecting interiorly.

We say a region is *interiorly connected* if its interior is a connected point set, i.e. its interior cannot be written as the union of two disjoint open point sets.

Proposition 3.14. *Suppose D consists of only interiorly connected regions. If* $[\mathrm{mbr}(a_i)]^\circ \cap [\mathrm{mbr}(a_i)]^\circ \neq \emptyset$, *then at most four rectangle regions will be removed from* $\mathrm{mbr}(a_i)$ *because of* a_j *in the construction of* $h(a_i)$.

PROOF. See Appendix. An illustration of the situation is shown in Figuer 9.



(a) Removing five rectangle regions. (b) Removing four rectangle regions.

Figure 9: Illustration of removing rectangle regions from $\mathrm{mbr}(a_i)$ because of a_j.

The intuition behind the above two propositions is that the complexity of the shape of $h(a_i)$ is mainly due to intersecting MBRs. Therefore, the average number of MBRs intersecting interiorly with $\mathrm{mbr}(h(a_i))$ would be an important factor affecting the size of the representation, which can be measured by the so-called *average intersection degree* [10].

Definition 3.15 (From [10]). Given a set of regions D, suppose the MBR of a region $a_i \in D$ intersects with the interior of d_i other MBRs of regions in D. Then the average intersection degree (AID) of D is $(\sum_{a_i \in D} d_i)/|D|$.

Intuitively, the average intersection degree of a set of regions is the average number of MBRs whose interior intersect with the MBR of each region in the set. The effect of AID on the storage size of the representations will be illustrated in the section of empirical evaluations. Intuitively, by the above two propositions, the smaller the AID is, the less complex the shape of $h(a_i)$ is.

3.5. Compact geometric representation of CDC relations

So far, we have discussed two methods, viz. the polygon representation and the rectangle representation, for compactly representing the geometric representation of the CDC relations between a set of regions. It is straightforward to see the following conclusion about the relationship between the CDC relations of any two original regions and of the more compact representations.

Proposition 3.16. *Given a set D of regions, suppose $a_i, a_j \in D$. Let P_i and P_j be the polygon representations of $h(a_i)$ and $h(a_j)$, and K_i and K_j be the corresponding rectangle representations. Then*

$$\delta(a_i, a_j) = \bigcup\{\delta(p, \mathrm{mbr}(P_j)) : p \in P_i\},$$

and

$$\delta(a_i, a_j) = \bigcup\{\delta(r, \mathrm{mbr}(K_j)) : r \in K_i\}.$$

With the two methods for compact representation and the method for geometric representation, we actually obtained a new systematic procedure to compactly represent the CDC relations between a set of regions:

1. Given a set D of regions, build the grid $\mathcal{G}_D$;
2. Obtain the necessary cut of D, i.e. $h(D) = \{h(a_i) : a_i \in D\}$;
3. For each $h(a_i)$, apply Algorithm 2 (polygon representation) or Algorithm 4 (rectangle representation), to obtain a more compact representation of $h(a_i)$.

The time complexity for the first step is $O(n)$, where $n = |D|$, as we only need to get the coordinates of the MBR of each region in D. For the second step, it uses Algorithm 1 to obtain $h(a_i)$ which can be done in $O(n_i^2 + nt_i)$time for each $h(a_i)$ (cf. Section 3.3). There are n pieces of such $h(a_i)$, so the time complexity for the second step is $O(n(n_i^2 + nt_i))$. The last step needs $O(m_i^3)$ time for each $h(a_i)$, where m_i is the number of cells in $h(a_i)$. Therefore, the time complexity for the whole procedure is $O(n + n(n_i^2 + nt_i) + nm_i^3) = O(n(n_i^2 + nt_i + m_i^3))$.

In the above procedure, the mapping h is faithful, such that the CDC relation between any two original regions a_i and a_j in D is the same as that between $h(a_i)$ and $h(a_j)$. In other words, the CDC relation between any original regions can be directly calculated later using the necessary cut. Moreover, by Proposition 3.16, the CDC relation between the original regions can also be directly calculated from the more compact representation. Therefore, the resulting compact geometric representation of the CDC relations can then be used to retrieve the original CDC relations.

It is also worth noting that the above procedure is reasonably fast to construct a geometric compact representation of CDC relations. For most of the datasets used in our experiments, the time is from 6 seconds to 3 minutes. For a dataset with $36,702$ regions (AID: 7) it is about 1.4 hours, and for a dataset with $11,613$ regions (AID: 410) it is about 1.7 hours.

In the following, we briefly describe how to accomplish that based on a polygon representation or a rectangle representation. The practical performance for reducing storage space and retrieving relations will be evaluated in Section 4.

3.6. Answering queries with compact geometric representations

There are several types of queries, as summarised in [29]. Here we focus on the queries of retrieving the CDC relation when given two objects, which is very common in applications, e.g., navigation services might be interested in if some place is "to the north of" a certain landmark.

The general approach for answering such queries from a compact geometric representation of the CDC relations is to use geometric calculation to compute

CDC relations between the geometries in the compact representation. In particular, according to Proposition 3.16, for the polygon representation, we check the intersections between the polygons of P_i and the nine tiles formed by $\mathrm{mbr}(P_j)$; for the rectangle representation, we check the intersections between the rectangles of K_i and the nine tiles formed by $\mathrm{mbr}(K_j)$, and both checks intersection of MBRs before checking intersection between polygons.

Spatial indexing techniques could be helpful for further optimizing query answering. Take the rectangle representation as an example. R-trees and similar data structures [30, 31, 32] can be used to index the rectangles in the representation $h(a_i)$, so that it becomes more efficient to check if there is any rectangle intersects the tile $\chi(h(a_j))$. However, such techniques need additional storage space that could be much larger than the geometry-based representation, and such optimization is actually not necessary, as our experiments in the next section will show that query answering is already efficient enough without such indexing techniques. Therefore, here we would not consider them for query answering optimisation.

4. Empirical Evaluation

In this section, we evaluates the two geometry-based representations, viz. the polygon representation and the rectangle representation, in terms of storage size and query answering efficiency.

In the experiments, we employed the real-world datasets used in [10], divided into two classes, viz. Real-1 and Real-2, based on the *average intersection degrees* of these datasets. Real-1 consists of five datasets about the administrative regions from Global Administrative Areas [4], and Real-2 comprises five datasets about the environmental habitats from the European Environment Agency [5]. These datasets contain geometric data for polygon regions represented as coor-

[4] https://gadm.org/

[5] https://www.eea.europa.eu/ds_resolveuid/78a8fdf22fa14fddb8ff218071aeb5d8

dinates of the vertices of the polygons. [6] The information about the datasets is summarised in Table 1 (AID in the table means Average Intersection Degree per region, as defined in Definition 3.15).

Table 1: Datasets Summary.

Dataset	#Regions	AID	Avg. #Vert	Avg. #Comp
Germany	434	6.57	422	1.2
Ukraine	629	6.17	192	1.1
Australia	1395	7.12	2362	4.0
China	2411	6.86	734	1.9
USA	3145	6.42	45	3.5
Real-2.1	600	44.22	200	12.6
Real-2.2	610	105.60	300	21.1
Real-2.3	605	121.37	361	29.0
Real-2.4	611	178.43	198	18.5
Real-2.5	604	204.77	343	34.2

4.1. Storage size

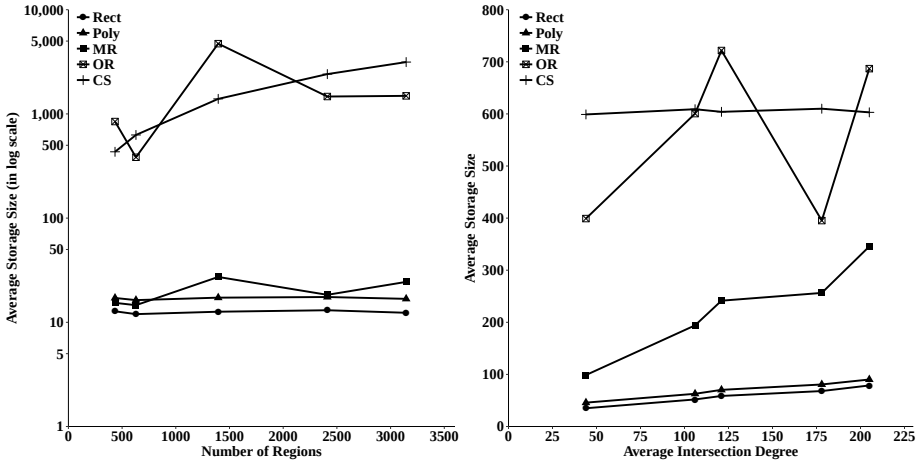
The baseline methods for the comparison of storage size would be the MBR-based representation (MR), the original regions (OR), and the naive complete storage of the CDC relations (CS), as MR is the state of the art which has been shown superior to other previous methods that are applicable, OR is a natural geometric representation, and CS is a purely qualitative representation that is commonly used in applications. Note that the prime subnetwork technique proposed in [21] is not applicable to CDC relations.

The storage size of the polygon representation (Poly) or the rectangle representation (Rect) is measured by the number of coordinates for representing the polygons or the rectangles; for the MBR-based representation, it is measured by the sum of the number of stored CDC relations and the number of coordinates for representing the MBRs of connected components of the regions; for the method using original regions, it is measured by the number of coordinates for

[6] Downloadable from http://zhiguolong.github.io/file/datasets.zip

representing the regions; for the naive complete storage method, it is measured by the number of stored CDC relations. Note that for each dataset, comparing total storage size would be equivalent to comparing *average* storage size obtained by dividing total storage size by the number of regions in the dataset. Therefore, in order to keep the numbers small, we will do the comparisons on average storage size.

Figure 10 shows the average storage size for a region in the datasets. The x-axis of the left figure is the number of regions in datasets and the x-axis of the right figure is AID, which can be regarded as a measure of the complexity of the CDC relations of a dataset. The first figure is used to show the variation of storage sizes when the number of regions changes while the AID is fixed. The second figure intends to show the effect of AID on storage size of the methods. Note that the y-axis of Figure 10(a) is in log scale.



(a) Real-1 (b) Real-2

Figure 10: Storage size comparison on the two classes of real-world datasets.

From the figures, it is clear that there is a large improvement of the other methods over OR and CS. For Real-1 datasets, while the other methods maintain relatively small average storage sizes under 30, the methods OR and CS exhibit rather large average storage sizes of more than hundreds; for Real-2 datasets,

the methods OR and CS have large average storage sizes of more than 400 for each region (note there are about 600 regions in the Real-2 datasets), while those of Rect and Poly are under 100 and MR between 100 and 400.

Moreover, in the figure for Real-1 datasets, when the number of regions increases, the average storage size of Rect and Poly is almost a constant. This indicates that the total storage size of Rect scales linearly to the number of regions, when AID is fixed. On the other hand, although the average storage size of MR stays small, it slightly increases when the number of regions becomes larger. As for CS, its average storage size increases linearly w.r.t the number of regions, which means its total storage size is quadratic to the number of regions. For OR, it is mainly affected by the complexity of the regions. Note that from the figure we can see that the storage size of MR is also affected by the complexity of the regions, because it needs to store the MBRs of connected components of the regions, while for Rect and Poly it is not the case, which is due to that it only cares about the CDC relations rather than the shapes of the regions.

The figure for Real-2 datasets considers the effect of AID on storage size. As AID increases, the average storage size of Rect and Poly increases slightly, while that of MR increases drastically, and the storage size of CS almost does not change due to that the number of regions does not change much. Note that AID reflects the complexity of the CDC relations. This result shows that Rect and Poly can handle complex datasets very well and much better than MR, which is the state of the art. The storage size of OR is again affected by the complexity of the regions, and that also affects the storage size of MR, as shown in the figure that the rate of increment of its storage size (i.e. the slope of line segments) is correlated with the difference of vertex number in datasets.

From the results, it is clear that Rect and Poly dominate the other methods for all the datasets and scales very well w.r.t. either the number of regions or AID, with average storage sizes around 12 (Rect) and 17 (Poly) for Real-1 datasets, and between 30 and 80 (Rect) and between 40 and 90 (Poly) for Real-2 datasets. Also, Rect always has smaller storage size than Poly for all the

datasets, confirming the result in Proposition 3.12. There are two reasons for its good performance:

- when AID is small, i.e. the dataset is not very complex, the CDC relations of the regions usually will not remove many cells from the MBR of a_i and thus the shape of $h(a_i)$ is simple;
- when AID becomes larger, i.e. the dataset becomes more complex, although more cells from the MBR of a_i will be removed when constructing $h(a_i)$, the removed cells are usually not *critical* to make the shape of $h(a_i)$ to be much more complex.

To demonstrate the capability of dealing with larger datasets, we also tested Rect on six large datasets used by [11]. The first four datasets are from European Environment Agency [7], the County dataset is a dataset of county subdivisions of the USA [8], and the School dataset is a dataset of school catchment areas in the USA [9]. The information is summarised in Table 2.

Table 2: Larger Datasets Summary.

Dataset	#Regions	AID	Avg. #Vert	Avg. #Comp
HEU	5322	238.18	155	15.00
HMS	6258	227.89	132	12.91
SEU	10061	413.23	149	14.01
SMS	11613	409.69	129	12.24
County	36702	6.74	782	1.15
School	65192	24.29	782	1.43

The result is summarised in Table 3. As we can see from the result, Rect still performs quite well on larger datasets, even for datasets with more than $60,000$ regions. It outperforms all the other methods. This is especially significant for datasets with larger AID, such as the four datasets HEU, HMS, SEU, and SMS,

[7] https://www.eea.europa.eu/ds_resolveuid/78a8fdf22fa14fddb8ff218071aeb5d8

[8] http://www.census.gov/

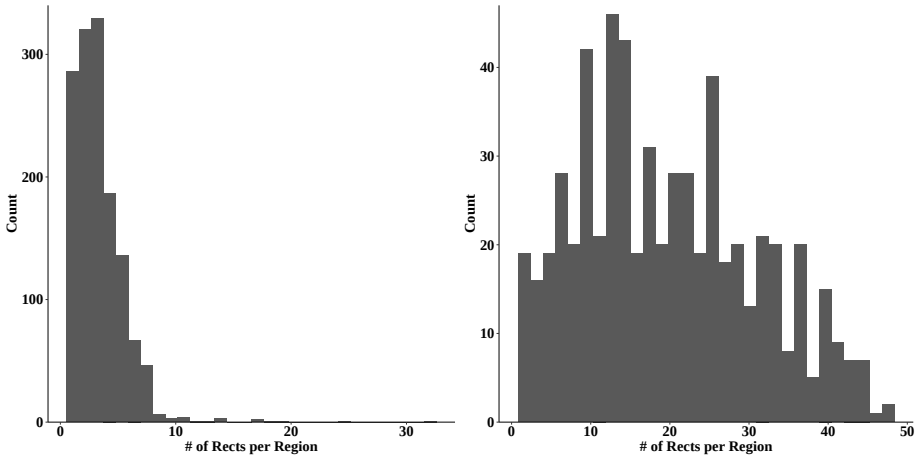
[9] https://nces.ed.gov/programs/edge/SABS

where the average storage sizes (and thus the total sizes) of the other methods are several times (4 to 200) greater than that of Rect.

Table 3: Storage size comparison on larger datasets, averaged over the number of regions.

Method	HEU	HMS	SEU	SMS	County	School
Rect	57.12	53.16	57	52.04	13.08	19.68
MR	302.18	283.53	473.27	462.65	15.34	34.01
OR	310	264	298	258	1564	1564
CS	5321	6257	10060	11612	36701	65191

The effect of different AIDs on Rect can be seen from Figure 11. For a dataset with smaller AID, most regions would have a relatively small number of generated rectangles, while for a dataset with larger AID, the number of regions having larger numbers of generated rectangles will increase, but still regions with smaller numbers of rectangles are the majority.



(a) Australia (b) Real2.5

Figure 11: Histograms showing the distribution of the number of generated rectangles for two datasets "Australia" and "Real2.5", where the x-axis shows different numbers of rectangles and the y-axis shows the numbers of regions that have corresponding numbers of rectangles.

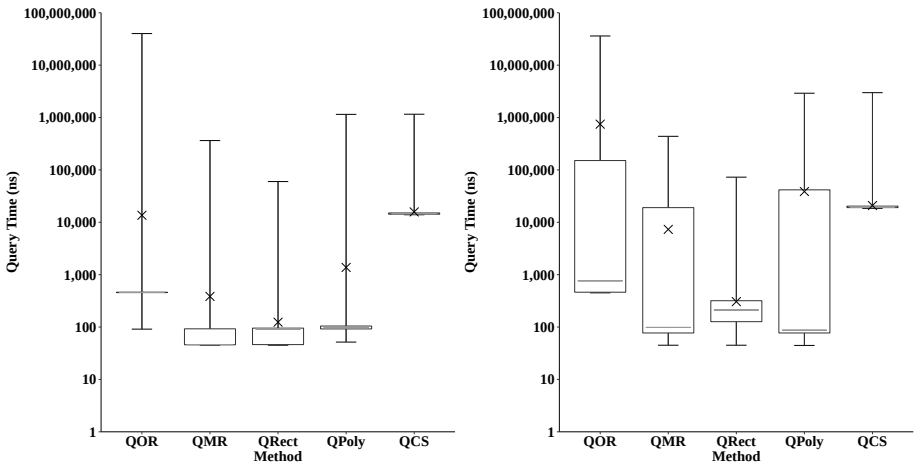
4.2. Answering query

The baseline methods for comparison would be the query answering methods for previously mentioned MR, OR, and CS. These query answering methods are denoted by QMR, QOR, and QCS, respectively. Query answering methods using the rectangle representation and the polygon representation will be denoted by QRect and QPoly, respectively. The queries considered here are to find out the CDC relation between two given objects. As the number of CDC relations that need to be stored by QMR and QCS can be quite large, storing them in a database is more scalable than in a in-memory hash table. Therefore, here we store them in a database hash indexed by using pairs of the regions as keys. The efficiency of answering queries of all these methods will be measured by the time for finding out the CDC relations between pairs of objects. The experiment has been done on a computer running Ubuntu 14.04LTS, with an Intel® Core™-i5 3.1 GHz CPU and 8 GB memory.

We present the results on the dataset "Australia" (with the largest average number of vertices) in Real-1 and the dataset "Real-2.5" (with the largest AID) in Real-2. Figure 12 shows the results of queries for $10,000$ random pairs of objects from each of the two datasets. The $10,000$ pairs are obtained by randomly sampling in the set of all the pairs of objects for each dataset.

In Figure 12, each box plot represents the query times of one method. The first and third quartiles of the query times of a method are represented by the hinges of its corresponding box, and the second quartile (i.e. the median) is the grey horizontal line segment inside the box. The maximum and minimum values are visualised as two short horizontal line segments (called *whiskers*) connected by vertical line segments above and under a box. The mean query time is drawn as "×".

The overall performance of query answering with the two new geometric representations is quite promising, especially the rectangle representation. In particular, for the dataset "Australia" of Real-1, the medians of QRect and QPoly are 91.7 ns and 97.5 ns, respectively, not much higher than the median of QMR, which is 46.2 ns, when compared against QOR and QCS, which are



(a) Real-1 (Australia)

(b) Real-2 (Real-2.5)

Figure 12: Query time comparison on two real-world datasets.

458.0 ns and 14,620.0 ns, respectively. For the dataset "Real-2.5", the median of QPoly is the smallest (87.1 ns), followed by that of QMR (98.3 ns), while that of QRect is not much larger (211.2 ns), as that of QCS is 19,411.0 ns. The median of QOR lies in the middle, which is 762 ns. This indicates that QRect and QPoly can answer half of the queries as efficiently as the state of the art method QMR, and is more efficient than the widely used method by calculating with original geometries. The long query time of QCS is due to that each query needs to be answered by communicating with a database, which is usually not as efficient as reading data directly from the memory. QMR also has this problem for queries that involve objects with intersecting MBRs.

Considering the mean value of each method, which reflects the average time needed for processing a group of queries, it turns out the rectangle representation performed very well again. Specifically, its mean query time is about 123.9 ns for "Australia", which is respectively 91.0%, 67.8%, 99.1%, and 99.2% less than that of QPoly, QMR, QOR, and QCS, and is about 305.6 ns for "Real-2.5", which is respectively 99.2%, 95.8%, 99.9%, and 98.5% less than that of QPoly, QMR, QOR, and QCS. Our hypothesis tests on the data (Wilcoxon signed-rank

tests) confirmed the impression that the average query performance of QRect is significantly more efficient than the other methods (significant at the 1% level). The drastic difference between median and mean values for QOR, QMR, and QPoly is due to that their query times for "harder" cases (i.e. cases whose time is larger than the median time) is usually much larger than the median. For example, for the dataset "Real-2.5", there are more than $3,000$ cases with query time larger than $10,000$ ns for each of the three methods, while their median is all less than $1,000$ ns. On the other hand, QRect has similar mean and median values, indicating that it is also efficient on "harder" cases.

Comparing the two figures, one would note that the performance of QRect is not greatly affected when the number of intersecting MBRs increases, which might be the benefit of decomposing complex shaped polygons into rectangles.

In summary, the results indicate that the rectangle representation can achieve a rather good balance between storage size and query answering efficiency, such that it has the smallest storage size while the query answering method based on it is about as efficient as the best method. Nevertheless, the polygon representation would still be useful in cases where the application or user asks for polygons rather than rectangles in the representation, e.g., in illustrative maps.

5. Implications and Limitations

5.1. Theoretical and managerial implications

In this article, the proposed approach has been demonstrated as promising both in theory and by empirical evaluations. This indicates that the proposed approach can help other algorithms and applications to better process qualitative directional relations, by storing them more compactly and retrieving them quickly. With the help of this approach, other algorithms and applications would be scalable to very large datasets that are common nowadays. The proposed approach also provides obfuscation to protect data privacy, by representing the original geometries in rough approximations.

As an example of how the proposed approach could be useful in practice, consider a location-based service company that needs to display obfuscated geometries for interesting regions on a map, e.g., land use map for a country where accurate boundary information for regions is highly sensitive, and to show in text the directional information of any two regions that are selected by the user. In this example, the proposed approach can be used to provide a compact representation and efficient query answering framework for the directional information, so that the storage space is saved and the scalability of the service is improved, while users can still quickly get directional information from the service. In addition, by using generated polygons and rectangles as approximations to the original geometries, companies or data providers do not need to worry about security issues of the original data.

5.2. Limitations and future work

Although the proposed approach is promising, it still has some limitations: the time complexity of the whole procedure is relatively high, so currently it cannot process datasets with millions of regions; it heavily relies on the properties of the CDC model, so it cannot be easily extended to deal with relations from other models; in the current form, the resulting representation cannot be dynamically updated, so it is hard to process dynamic datasets.

In the future, more efficient improvements of the current approach would be an interesting research topic. Also, similar techniques that apply to more generic qualitative relations can be considered, e.g., the widely used topological relations from the RCC8 model [33] or the nine-intersection model [34]. In addition, to support dynamic information processing is also an important problem for exploration, that is, how to efficiently update the generated geometries when the relations between some objects has been changed.

6. Conclusion

In this article, we addressed the problem of compact representation of qualitative spatial relations such that they can be efficiently retrieved to answer

queries. The novel approach proposed here applies to the directional CDC relations between a large number of regions, and it constructs simple geometries for each region by using necessary cut. The polygon representation and the rectangle representation are considered to represent the constructed simple geometries. Theoretically, the rectangle representation is demonstrated as promising to have small storage size for large datasets, and always has no larger storage size than the polygon representation. Due to their simple shapes, these two methods are fast for retrieving CDC relations. Also, on real-world datasets, the rectangle representation has been shown to be superior to the state of the art techniques in storage size and competitive in query answering efficiency. Nevertheless, there are still limitations of the proposed approach. Improvements and applications will thus be considered in future work.

Acknolwedgements

This work was supported by the National Natural Science Foundation of China (61806170, 61773324, 11671244), the Humanities and Social Sciences Fund of Ministry of Education (18XJC72040001), and the Fundamental Research Funds for the Central Universities (2682018CX25, 2682014ZT28).

Appendix A. Proofs of Propositions

Proof (Proof of Proposition 3.4). We only need to show that $\forall c_{kl} \subseteq \text{mbr}(a_i)$, if $c_{kl} \not\subseteq h(a_i)$ then $c_{kl} \not\subseteq f(a_i)$. Note that if $c_{kl} \not\subseteq h(a_i)$, then $\exists j, \chi$ s.t. $a_i^\circ \cap \chi(a_j) = \emptyset$ and $c_{kl} \subseteq \chi(a_j)$. Then $c_{kl} \not\subseteq f(a_i)$. This is because otherwise $(f(a_i))^\circ \cap \chi(a_j) \neq \emptyset$ while $a_i^\circ \cap \chi(a_j) = \emptyset$, which contradicts that $\delta(a_i, a_j) = \delta(f(a_i), f(a_j))$.

Proof (Proof of Proposition 3.7). The first line of Algorithm 2 scans the cells in $h(a_i)$ twice and the number of vertices in V is at most four times as the number of cells in $h(a_i)$, i.e. $|V| \leq 4m$. Note that each execution of Line 4 (i.e. Algorithm 3) removes at least one vertex, and thus the number of

iterations from Line 3 to Line 12 is at most $|V|$. The steps inside the outer loop take $O(|V|^2)$ time in total. In fact, each execution of Line 4 takes time linear to $|V|$ and each execution of Line 5 takes time of $O(|V| \cdot |R|)$, which is at most $O(|V|^2)$, as we can check for each vertex in $|V|$ if it is in the polygon formed by the exterior boundary R and every check takes $O(|R|)$ [24]. The while loop from Lines 7 to 10 takes time $O(|V'|)$ and at most $O(|V|)$, as it basically visits each vertex in V' once. Therefore, the time complexity of Algorithm 2 is $O(m + |V| * |V|^2) = O(m^3)$.

Proof (Proof of Proposition 3.9). To expand a strip efficiently, we can scan through $h(a_i)$ at the very beginning of the algorithm, and for each cell store the left and right boundary of the horizontal expansion of the cell. That takes $O(m^2)$ time. Then for each strip, we can visit the cells in it only once to find out the left and right boundary of the maximal rectangle containing the strip. As the number of cells in each strip is at most m, ExpandStrip(s) takes $O(m)$ time. On the other hand, there are at most m columns in $h(a_i)$ and for each column there are at most m strips. So the total number of loops is $O(m^2)$. Therefore, the total time complexity is $O(m^2 + m^2 * m) = O(m^3)$.

Proof (Proof of Proposition 3.12). First, it is easy to see that $p^* \leq p$ and $|V| = \mu_1 + 2\mu_2 + c_1 + 2c_2$. Then by Lemma 3.11, we only need to prove $c_1 \leq p$, because if $c_1 \leq p$, then $\mu_1 + 2\mu_2 + c_1 + 2c_2 = 4p - 2c_1 \geq 4p - 2p = 2p$. To this end, we show that for each type 1 vertex v, there is a rectangle in the partition such that v is on the boundary of it and no other type 1 vertex is on the boundary. Note that a type 1 vertex is associated to a type 1 chord. Let d_v be the chord that v is associated to, and m_v be the rectangle containing d_v on the right side of m_v (the existence of such m_v is easy to see). By the definition of type 1 vertex, no other type 1 vertex lies on the right side of m_v, but by the construction, any type 1 vertex lies on the right side of its corresponding rectangle. Therefore, for $v' \neq v$ being another type 1 vertex, we have $m_v \neq m_{v'}$. Thus $c_1 \leq p$.

Proof (Proof of Proposition 3.13). The "if" part is easy to see. For the

"only if" part, suppose $[\text{mbr}(a_i)]^\circ \cap \chi(a_j) \neq \emptyset$. If $\text{mbr}(a_i) \subseteq \chi(a_j)$, then the conclusion is obvious. If $\text{mbr}(a_i) \not\subseteq \chi(a_j)$, then one of the edge of $\chi(a_j)$ has intersection with $\text{mbr}(a_i)$. Note that $[\text{mbr}(a_i)]^\circ \cap [\text{mbr}(a_j)]^\circ = \emptyset$, such an edge must totally pass through $\text{mbr}(a_i)$. Because a_i is connected, this edge has an intersection with a boundary point or an interior point of a_i, say $p = (x_0, y_0)$. By taking an open neighbourhood $B(p, \epsilon)$ of p s.t. $B(p, \epsilon)$ is contained in $\text{mbr}(a_i)$, we know $B(p, \epsilon) \cap a_i^\circ$ is an open set in $a_i^\circ \cap \chi(a_j)$, That is, $a_i^\circ \cap \chi(a_j) \neq \emptyset$.

Proof (Proof of Proposition 3.14). Note that the removed regions must be among $\chi(a_j) \cap \text{mbr}(a_i)$, where χ is a CDC tile. As there are nine CDC tiles for a_j, we only need to consider those nine rectangle regions given by the nine CDC tiles. If two rectangle regions of those are edge-touching with each other, then they can be combined. So we can consider only rectangles regions that are not edge-touching with each other. If five or more rectangle regions are removed, then the O-tile region of a_j will be one of them. This results in only one option, as shown in Figure 9(a). The relation $\delta(a_i, a_j)$ is $\{\mathsf{N}, \mathsf{W}, \mathsf{S}, \mathsf{E}\}$, and thus rectangle regions in $\text{mbr}(a_i)$ formed by the intersection with the NW-, NE-, SW-, SE-, and O-tiles will be removed. Note, however, if this is the case, then a_i cannot be interiorly connected, which is a contradiction.

An example of four rectangle regions being removed is shown in Figure 9. The relation $\delta(a_i, a_j)$ is $\{\mathsf{N}, \mathsf{W}, \mathsf{S}, \mathsf{E}, \mathsf{O}\}$.

References

[1] S. Wang, D. Liu, Knowledge representation and reasoning for qualitative spatial change, Knowledge-Based Systems 30 (2012) 161–171.

[2] Y. Du, F. Liang, Y. Sun, Integrating spatial relations into case-based reasoning to solve geographic problems, Knowledge-Based Systems 33 (2012) 111–123.

[3] B. A. El-Geresy, A. I. Abdelmoty, SPARQS: a qualitative spatial reasoning engine, Knowledge-Based Systems 17 (2004) 89–102.

[4] W. Liu, X. Zhang, S. Li, M. Ying, Reasoning about cardinal directions between extended objects, Artificial Intelligence 174 (2010) 951–983.

[5] A. G. Cohn, S. Li, W. Liu, J. Renz, Reasoning about topological and cardinal direction relations between 2-dimensional spatial objects, Journal of Artificial Intelligence Research 51 (2014) 493–532.

[6] M. Mantle, S. Batsakis, G. Antoniou, Large scale distributed spatio-temporal reasoning using real-world knowledge graphs, Knowledge-Based Systems 163 (2019) 214 – 226.

[7] J. O. Wallgrün, Exploiting qualitative spatial reasoning for topological adjustment of spatial data, in: Proceedings of the 20th International Conference on Advances in Geographic Information Systems, 2012, pp. 229–238.

[8] M. A. Rodríguez, M. J. Egenhofer, A. D. Blaser, Query preprocessing of topological constraints: Comparing a composition-based with neighborhood-based approach, in: Proceedings of the 10th International Symposium on Spatial and Temporal Databases, 2007, pp. 362–379.

[9] P. Fogliaroni, Qualitative spatial configuration queries – Towards next generation access methods for GIS, Ph.D. thesis, University of Bremen, Germany, 2012.

[10] Z. Long, M. Duckham, S. Li, S. Schockaert, Indexing large geographic datasets with compact qualitative representation, International Journal of Geographical Information Science 30 (2016) 1072–1094.

[11] Z. Long, S. Schockaert, S. Li, Encoding large RCC8 scenarios using rectangular pseudo-solutions, in: Proceedings of the 15th International Conference on the Principles of Knowledge Representation and Reasoning, 2016, pp. 463–472.

[12] M. Duckham, L. Kulik, Simulation of obfuscation and negotiation for location privacy, in: Lecture Notes in Computer Science, 2005, pp. 31–48.

[13] G. E. Ligozat, Reasoning about cardinal directions, Journal of Visual Languages and Computing 9 (1998) 23–44.

[14] I. Navarrete, A. Morales, G. Sciavicco, M. A. Cardenas-Viedma, Spatial reasoning with rectangular cardinal relations, Annals of Mathematics and Artificial Intelligence 67 (2013) 31–70.

[15] R. K. Goyal, M. J. Egenhofer, The direction-relation matrix: A representation for directions relations between extended spatial objects, in: Proceedings of the Annual Assembly and the Summer Retreat of University Consortium for Geographic Information Systems Science, 1997, pp. 22–81.

[16] R. K. Goyal, M. J. Egenhofer, Similarity of cardinal directions, in: Proceedings of 7th International Symposium on Spatial and Temporal Databases, 2001, pp. 36–58.

[17] S. Skiadopoulos, M. Koubarakis, Composing cardinal direction relations, Artificial Intelligence 152 (2004) 143–171.

[18] W. Liu, S. Li, Reasoning about cardinal directions between extended objects: The np-hardness result, Artificial Intelligence 175 (2011) 2155–2169.

[19] Y. Izmirlioglu, E. Erdem, Qualitative reasoning about cardinal directions using answer set programming, in: Proceedings of the 32nd AAAI Conference on Artificial Intelligence, 2018.

[20] M. J. Egenhofer, J. Sharma, Assessing the consistency of complete and incomplete topological information, Geographical Systems 1 (1993) 47–68.

[21] S. Li, Z. Long, W. Liu, M. Duckham, A. Both, On redundant topological constraints, Artificial Intelligence 225 (2015) 51–78.

[22] L. H. Karlsen, M. Giese, An efficient representation of general qualitative spatial information using Bintrees, in: Proceedings of the 13th International Conference on Spatial Information Theory, 2017, pp. 4:1–4:15.

[23] L. H. Karlsen, M. Giese, Qualitatively correct bintrees: an efficient representation of qualitative spatial information, GeoInformatica (2019) 1–43.

[24] J. O'Rourke, Computational Geometry in C, Cambridge university press, 1998.

[25] J. C. Culberson, R. A. Reckhow, Covering polygons is hard, Journal of Algorithms 17 (1994) 2–44.

[26] D. S. Franzblau, D. J. Kleitman, An algorithm for constructing regions with rectangles: Independence and minimum generating sets for collections of intervals, in: Proceedings of the 16th Annual ACM Symposium on Theory of Computing, 1984, pp. 167–174.

[27] V. Kumar, H. Ramesh, Covering rectilinear polygons with axis-parallel rectangles, SIAM Journal on Computing 32 (2003) 1509–1541.

[28] D. S. Franzblau, Performance guarantees on a sweep-line heuristic for covering rectilinear polygons with rectangles, SIAM Journal on Discrete Mathematics 2 (1989) 307–321.

[29] P. Fogliaroni, P. Weiser, H. Hobel, Qualitative spatial configuration search, Spatial Cognition & Computation 16 (2016) 272–300.

[30] A. Guttman, R-trees: A dynamic index structure for spatial searching, in: Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data, 1984, pp. 47–57.

[31] N. Beckmann, H. Kriegel, R. Schneider, B. Seeger, The R*-tree: An efficient and robust access method for points and rectangles, in: Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data, 1990, pp. 322–331.

[32] D. Papadias, Y. Theodoridis, T. Sellis, The retrieval of direction relations using R-trees, in: Proceedings of the 5th International Conference on Database and Expert Systems Applications, 1994, pp. 173–182.

[33] D. A. Randell, Z. Cui, A. G. Cohn, A spatial logic based on regions and connection, in: Proceedings of the 3rd International Conference on Principles of Knowledge Representation and Reasoning, 1992, pp. 165–176.

[34] M. J. Egenhofer, R. D. Franzosa, Point-set topological spatial relations, International Journal of Geographical Information System 5 (1991) 161–174.