Universidad de Concepción

Dirección de Postgrado

Facultad de Ingeniería - Programa de Doctorado en Ciencias de la Computación

EFFICIENT QUERY PROCESSING FOR MULTIGRANULAR DATA

Tesis para optar al grado de Doctorado en Ciencias de la Computación

POR

Diego Gatica Romero

CONCEPCIÓN, CHILE

Marzo, 2024

Profesores guía: Andrea Rodríguez y Diego Seco

Departamento de Ingeniería Informática y Ciencias de la Computación

Facultad de Ingeniería

Universidad de Concepción

AGRADECIMIENTOS

Firstly, I extend my heartfelt gratitude to all the professors and mentors who provided invaluable guidance and support throughout the development of this thesis. I am deeply appreciative of my classmates for their unwavering support during my time at the university. I am also indebted to my closest friends and colleagues who stood by me, offering encouragement whenever needed. Lastly, I extend a special thanks to my faithful companion, Kudo, who remained by my side throughout almost every step of this journey.

Abstract

Spatial and temporal attributes are typical examples of data that can be represented at different levels of granularity or resolution. The massive amount of this type of data makes it impractical to store and process data without making use of efficient algorithms and structures. In search of a way to handle multigranular data, several data models have been proposed; however, there is no efficient implementation in terms of space usage and query time for any of the various existing models for handling multigranular data. In this thesis, we study algorithms and data structures to process different queries on multigranular models, specifically, the work done uses succinct data structures and achieves a good trade-off between space usage and query time.

In particular, we start by proposing a succinct data structure and algorithm for the implementation of a multigranular model that is general enough to be used in different domains. This model is based on the relations of subsumption and disjoint between its elements (i.e. granules), and their respective negations, and it proposes the strategy of deriving new relations, in order to reduce the space to be used. The proposed structure used $(|E| - |V|)\log_2 |V| + O(|E|)$ space to store a graph with V vertices and E edges to store a graph that represent the subsumption relation, plus $|E^r| \log |V| + |E^r| + |V| + o(|E^r| + |V|)$ for for each of the other relationships, and improves the derivation of new relations, compared to other implementations. A second succinct data structure is proposed, with a focus on the spatial domain by providing algorithms for processing topological queries like inclusion, disjointness, and adjacency between regions on a multi resolution context. In the case of a set of n regions without a hierarchy, we can manipulate it efficiently using $4n + o(n)$ bit, for the case when we have a hierarchy of height h, our structure proposed requires as little as $O(n \log h)$ bits, while maintaining a similar query time compared to a non-compact implementation.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Many different real applications require storing and handling data at different levels of detail due to the nature of the data, diverse data origins, and resource-constraint specifications. An example of this is working with spatial and temporal data, as they can generally be associated with different temporal scales or different levels of spatial resolution. The multigranular nature of the data presents several problems when handling data at various levels of detail. Some of these problems are how to store information efficiently in space, how to answer queries between objects that are at different levels of detail or how to deal with discrepancies between different levels of detail, where the latter problem is quite common when integrating data from different sources. The following example is used to illustrate the problems mentioned above.

Example 1 Consider a database that stores information about total number of people infected by Covid-19 in Chile at regional level in a specific month, as show in Table 1.1. In addition, Table 1.2 stores the same information but associated at the provincial level in a specific week and differentiated by gender.

For further understanding, recall that the administrative subdivision of Chile defines regions that are composed of provinces. Thus, the space of Chile is organized in granularities (i.e., Region and Province), each of them composed of granules (particular regions and provinces) that make a partition of the space.

Region	Date	Total
Valparaíso	September	5098
Santiago	September	12010
Biobío	September	7478

Table 1.1: Total cases by region

Province	Week	Sex	Total
Quillota	18	Male	53
Arauco	18	Female	33
Concepción	18	Female	207
Biobío	18	Female	196

Table 1.2: Total cases by province

A common query would be to obtain the number of women infected at the regional level. In the example presented above, the region of Biobío in week 18 had a total of 436 women infected. To get this answer, it is required to know how the data at the province level are related with the data at the regional level, which implies storing how each tuple between the two levels are related. This means that storing this information has a quadratic cost in terms of space required in cases where several granularities are present (for only two granularities, foreign keys could be used), which is unfeasible if we have a large amount of data.

Another problem with this query is that we need a way to transform the data between the different levels of detail or granularity. For this example, with only two levels of detail it is trivial, but in the case that we also have the information at the level of county and city, it becomes necessary to have an algorithm that solves this in an efficient way in terms of query time. Finally, the information, in addition of being at different spatial scales, is associated with different temporal levels of detail. This presents another problem, because a month is not formed by an exact number of weeks, so when transforming information between both levels, we may lose accuracy.

The work in this thesis focuses on designing and implementing algorithms and data structures to process different queries on multigranular models. First, we develop algorithms to answer disjoint and subsumption queries based on the model presented in [73]. This proposed new strategy presents a way of dealing with data of a multigranular nature in an efficient manner in terms of query time while maintains low space usage, this is due to an approach of storing partial information, with which it is possible to derive the rest of the information not explicitly stored. As a second line of research, algorithms and data structures were developed based on a generalization of the topological model to a multigranular instance, based on the work presented in [55] with a focus on answering topological queries. The algorithms developed are focused on answering topology-related queries between objects, such as list the neighbors of an object or check for containment between two objects, being that the objects consulted are not necessarily in the same level of detail, keeping the space used low.

This document is organized as follows: Chapter 2 provides the background needed to understand the developed work. Chapter 3 presents related work about handling multigranular data. In Chapter 4 we present the data structure and algorithms developed for the multigranular model presented in [73], while in Chapter 5 we propose several space-efficient data structures to support a topological representation of regions that are organized in a multi-granular or hierarchical structure. In Chapter 6, we discuss open problems and future work. Finally, in Chapter 7 we present our conclusions about this thesis.

1.1 Hypothesis

The thesis work is based on the following hypothesis: *It is possible to design practical implementations of multigranular data models that require small space*. We say an implementation is practical if it can be implemented in commodity architectures, can scale on processing time over the data volume and it uses memory space accordingly with the theoretical results. We complement this hypothesis with the following research questions: i) Are graph-based structures useful for representing hierarchies of granularities? ii) What is the trade-off between representing explicitly versus deriving relationships between granules at different granularities?

1.2 Main Goal

The main goal of this thesis is to design and implement algorithms and data structures that allow the handling of multigranular data in different models; in particular, algorithms for the multigranular model proposed in [73] and algorithms for a restricted multigranular model inspired by the characteristics of the spatial domain.

1.3 Specific Goals

1. To design construction and query algorithms that make use of succinct data structures to answer topological queries in a multigranular instance.
2. To design construction and query algorithms that make use of inference rules to answer subsumption and not disjoint queries in the multigranular model proposed in [73].

3. To evaluate the performance of the developed algorithms against other implementations capable of answering the same queries, using real world data.

1.4 Methodology

- Review of the state of the art in models focused on handling multigranular data; particularly, focusing on general models for handling multigranular data and models focused on geographic partitions.

- The development of algorithms that allowed the use of the inference rules proposed in [75] in a multigranular model, in order to reduce the space used, was undertaken. Using succinct data structures, a new way to represent the stored data was devised. Additionally, algorithms were developed to derive non-stored information using inference rules.

- The development of a succinct data structure focused on answering topological queries about partitions of a spatial region was undertaken, extending the idea of a compact representation of a planar network to a multigranular context. Algorithms were developed to answer topological queries using the proposed data structure. Various alternatives for the development of the structure were explored. The objective of these implementations was to reduce space usage while maintaining competitive query times with respect to other implementations of similar functionality.

- All proposed data structures and algorithms in this thesis were implemented in C++. The SDSL library (available for C++) was used in the implementation because it contains several succinct data structures needed by the designed algorithms.

- An experimental evaluation was conducted to compare the structures and algorithms developed with other implementations of equal functionality. In the evaluation of the algorithms, the time needed for answering the respective queries and the space usage were assessed. All measurements were made on a machine running Linux 3.13.0-86-generic, in 64-bit mode, and the running time was measured using the **clock**$_{g}ettimefunction.Bothrealandsyntheticdatasetswereusedtoevaluatethee$

- Reproducibility of the results obtained in this work is an integral part of this thesis As such, all implementations, corpora and results are available in the following public web repositories: https://github.com/Desidia/gbp (work developed in chapter 4) and https://github.com/Desidia/pemb (work developed in chapter 5) .

1.5 Contributions

- We designed and implemented a data structure for the multigranular model presented in [73]. The proposed structure occupies $(|E|-|V|)\log_2|V|+O(|E|)$ space to store a graph with V vertices and E edges to store a graph that represent the subsumption relation, plus $|E^r|\log|V|+|E^r|+|V|+o(|E^r|+|V|)$ for each of the other relationships space. Based on the proposed structure, several algorithms were developed for the inference of relations between the granular components of the structure. In our experiments, these algorithms demonstrated a significant improvement in terms of their query time with respect to other implementations based on adjacency list and algorithm for graph traversal. The structure and algorithms developed are described in detail in Chapter 4. The developed work was submitted to the *Software, Practice and experience* journal, and is currently under review.
- We designed and implemented several space-efficient data structures to support access to the topological representation of two-dimensional regions that are organized in a multi-granular or hierarchical structure. The proposed structures build upon compact planar graph embeddings and serves to answer queries about inclusion, disjointness, and adjacency between regions. For a set of n regions and a hierarchy of height h, our representation requires as little as $O(n\log h)$ bits, which becomes $O(n)$ if the number of regions increases by a multiplicative constant from each level to the next. Our experimental results show that our propose structure use around as 8 bits per region. Further, with about 16 bits per region, our data structures answer all queries within 10 nanoseconds per retrieved

elements, and in some cases less than half a nanosecond. The structure and algorithms developed are described in detail in Chapter 5. We published our preliminary results in the Data Compression Conference (DCC) [53]. The full version of this work was later extended and published in the journal Information and Computation [54].

- We provide a repository with all our implementations and datasets. The repositories are available on https://github.com/Desidia/gbp (work developed in Chapter 4) and https://github.com/Desidia/pemb (work developed in chapter 5).

Chapter 2

Background

In this section, we present concepts required to understand the work presented in this thesis. We begin by describing existing spatial data models and topological queries. We continue by discussing the multigranular model presented in [73] on which the work presented in Section 5 is based and, finally, we present compact data structures relevant to this work, paying special attention to strategies for compact representations of planar-graph embeddings.

2.1 Spatial data models

Part of the work developed in this thesis focuses on solving topological queries between spatial objects at different granularities. For a better understanding of the work done, the models for representing spatial data [67, 136] are described below.

There are two conceptually different spatial models [116, 64]: the object model and the surface or field model. While object model focus on representing objects that are located on a space, a field or surface model focuses on representing the space to which attributes are associated. Then, these models are materialized in different logical models, being the most relevant the vector and raster models.

2.1.1 Vector model

Vector model uses the approach of representing the world as a collection of discrete objects. Spatial objects have non-spatial and spatial attributes. From a database perspective, spatial objects are typically stored in extended relational databases that provide spatial attributes as a data type with its operators.

Within the vector approach, there are several different ways of storing geometrical information ranging from the simpler, so called Spaghetti model, to more complex Topological model. Both models are common in Geographic Information Systems (GISs) and the difference between them is how the data is structured and organized. The following is a description of both models.

Spaghetti model. In the spaghetti model [23, 68], each point (0-dimension), line (1-dimension), or polygon (2-dimension) feature is represented as a string of (X,Y) coordinate pairs with no inherent structure. The name of the model is due to the fact that one can imagine each line as a strand of spaghetti, in this way, more complex figures (made up of several lines) would be formed by the addition of several strands of spaghetti. A major disadvantage of this model is that each polygon must be made up of its own strands of spaghetti. In other words, each polygon must be defined by its own set of (X,Y) coordinate pairs, that is, even if there are polygons that share the same edge, each one will store in its set of coordinates the representation of the shared edge, which implies storing redundant information and, in consequence, impacting the efficiency of the model.

Despite the location assigned to spatial features (lines), or strand of spaghetti, spatial relationships are not explicitly encoded within the spaghetti model; rather, they are implied by their location. This results in a lack of topological information, which results in complications if the user want to perform measurements or analysis. Nevertheless, the simple structure of the spaghetti data model allows for efficient reproduction of maps and graphics when the topological information is unnecessary, as is the case of applications that focus primarily on making a graphic representation of spatial objects.

Topological model. The topological data model [67, 34, 40] is characterized by the incorporation of topological information directly into the dataset, as the name implies. In essence, it utilizes a set of rules to precisely describe the relationships between adjacent points, lines, and polygons and determines how they share geometries. For example, consider two adjacent polygons. In the spaghetti model, the boundary shared by two polygons must be described individually for each polygon, in a general view, the shared boundary would be described as two identical, but separate lines. By including topological information, the shared boundary would be represented by a single line, storing a reference to distinguish which side of the line belongs to which polygon.

Three fundamental topological principles that are necessary to understand the topological data model are described as follows. First, **connectivity** describes the arc-node topology for the feature dataset. Nodes are more than simple points, in the topological data model, they represent intersection points where two or more arcs converge. **Area definition** involves the concept of polygon-arc topology, where an arc delineating an area serves as the basis for polygon formation. Specifically, in polygon-arc topology, polygons are constructed using arcs, and each arc is stored only once, avoiding redundant. **Contiguity**, the third topological principle, is based on the concept that polygons sharing a boundary are considered to be adjacent. In the context of polygon topology, it is essential for all arcs within a polygon to possess a specific direction, characterized by a from-node and a to-node. This directional attribute facilitates the determination of adjacency information. When polygons share an arc, they are deemed adjacent or contiguous, enabling the definition of the ”left” and ”right” sides of each arc. The explicit storage of this left and right polygon information is an integral part of the attribute data within the topological data model.

2.1.2 Raster model

The raster data model [87, 93, 125] consists of rows and columns of equally sized pixels interconnected to form a planar surface, being these pixels the basic unit of the raster model, which are used for the creation of points, lines, areas, networks, and surfaces. While pixels in GIS data can take various shapes, such as triangles, hexagons, or octagons, square pixels are often preferred for their simplicity. As a result, the majority of raster GIS data utilizes square pixels. When transforming the data model from one projection to another, these square pixels are commonly reshaped into rectangles with different dimensions. The raster data model, recognized as a grid-based system due to its reliance on a consistent series of square pixels, allocates, commonly, a single data value to each grid locale. Within this framework, individual cells contain unique values that characterize spatial phenomena at a location indicated by its row and column. The raster model will average all values within a given pixel to yield a single value.

Therefore, the larger the area covered by each pixel, their associated data values will be less accurate. The area covered by each pixel determines the spatial resolution of the raster model from which it is derived.

Several methods exist for encoding raster data from scratch. Some of these methods include the following:

- Cell-by-cell raster encoding. This encoding method is minimally intensive, capturing raster data by generating records for each cell value organized by both row and column.

- Run-length raster encoding. This method encodes cell values in runs (sequences in which the same data value appears in many consecutive data elements) of pixels with similar values, this may produce highly compressed image files.

- Quad-tree raster encoding. This approach entails dividing a raster into a hierarchical structure of quadrants, each further subdivided based on pixels with similar values. The process of subdividing the raster halts when a quadrant consists entirely of cells with identical values. Such a non-divisible quadrant is referred as *leaf node*.

The use of a raster data model confers many advantages. First, the technology necessary for generating raster graphics is both cheap and widely available. The raster data model possesses a straightforward underlying data structure, where each grid location in the raster image corresponds to a singular value. This inherent simplicity facilitates the interpretation and maintenance of graphics, especially when compared to its vector counterpart.

Despite its advantages, the raster data model comes with several drawbacks. The first disadvantage is the tendency for raster files to be relatively large in size. Particularly in the case of raster images built from the cell-by-cell encoding methodology. A second drawback of the raster model is that the resultant images are often less aesthetically pleasing than their vector counterparts, especially when the raster images undergo enlargement or zooming. Additionally, the geometric transformations inherent in map re-projection efforts pose challenges for raster graphics, constituting a third disadvantage of the raster data model. Altering map projections can lead to changes in the size and shape of the original input layer, often resulting in the loss or addition of pixels.

When compared to the raster data model, vector data models often serve as superior representations of reality, being this the default model implemented in many spatial extensions of database administrators, such as PostGis. This is attributed to the higher accuracy and precision achieved through points, lines, and polygons, as opposed to the regularly spaced grid cells in the raster model. Additionally, vector data allows for greater flexibility in altering the scale of observation and analysis. Another advantage of the vector model, is that topology is inherent. The incorporation of topological information in a vector model yields streamlined spatial analysis, simplifying tasks such as error detection, network analysis, proximity analysis, and spatial transformation.

2.1.3 Topological queries

In general, we can distinguish between three types of queries when working with space objects [35].

Thematic query. Thematic queries consist in the selection and management of thematic information. This process is comparable to a query of a conventional database, whose data have no spatial reference. Thematic queries are usually used to filter spatial objects with respect to certain attributes; an example of this would be *select the regions that have an area* $\leq$ *20* km^2 or *select the rivers of seasonal type*.

Geometric query. Geometric queries consist of obtaining measurable characteristics of space objects. The way to answer geometrical queries will depend on the type of model used for the representation of the spatial objects (vector model or raster model). Common geometric queries are to obtain measurements, such as area, length or diameter, distance between objects, and the proximity analysis (buffering).

Topological query. While geometric queries select objects based on their measurable spatial characteristics and thematic queries identify elements based on their properties, topological selection criteria are rooted in the spatial arrangement of objects in relation to one another. Topological arrangements of objects are accessed through features such as *adjacent*, *part of* or *within*. Topology deals with the spatial and structural characteristics of geometric objects, irrespective of their size, type, or specific geometric shape. Notable topological properties include the object's dimensionality and the relationships arising from the type of intersections between objects. Some common and relevant examples of topological queries are retrieving the spatial objects contained within a certain geometry G, or retrieving the K nearest neighbors of a certain spatial object Q. Importantly, all topological properties remain invariant under any continuous deformation of space.

Several methods have been investigated for the classification of possible topological relationships [32]. An interesting method for the classification of topological relations was proposed in [38, 39]. This work proposes that each space object is composed of two components, its interior and its boundary, and analyzes the intersection of these components between each type of defined space object (point, line and region), known as the 4-intersection model. Later in [41], this model was extended by proposing a new component, the exterior. Due to the large number of relationships obtained using the 9-intersection schema, several of these relationships are grouped together in order to provide a smaller set of topological relationships that are easy to use. Given two regular objects with extension (i.e., regions), A and B, the following topological relationships are defined:

- **Disjoint**: There is no intersection area between object A and object B.
- **Meet**: Object A and object B only meet at a boundary.
- **Overlap**: Object A and object B overlap.
- **Contains**: Object A contains object B.
- **Inside**: Object B lies inside object A.
- **Covers**: Object A covers object B.
- **Covered by**: Object B is covered by object A.
- **Equal**: Object B and object A match

Figure 2.1 shows a graphic example of each of the relationships mentioned above in addition to its 9-intersection matrix, where the first row/column corresponds to the interior of the spatial object, the second row/column corresponds to the boundary and the third second row/column corresponds to the exterior.

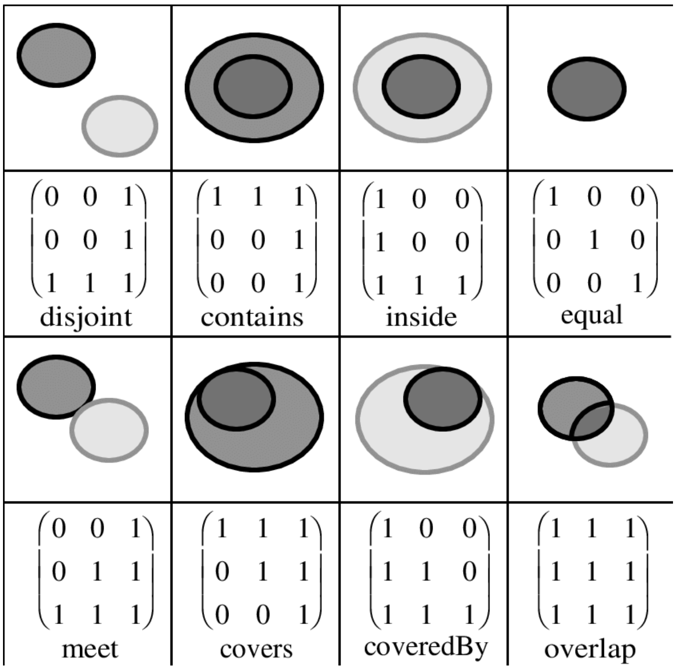

$\begin{pmatrix}0&0&1\\0&0&1\\1&1&1\end{pmatrix}$ disjoint
$\begin{pmatrix}1&1&1\\0&0&1\\0&0&1\end{pmatrix}$ contains
$\begin{pmatrix}1&0&0\\1&0&0\\1&1&1\end{pmatrix}$ inside
$\begin{pmatrix}1&0&0\\0&1&0\\0&0&1\end{pmatrix}$ equal
$\begin{pmatrix}0&0&1\\0&1&1\\1&1&1\end{pmatrix}$ meet
$\begin{pmatrix}1&1&1\\0&1&1\\0&0&1\end{pmatrix}$ covers
$\begin{pmatrix}1&0&0\\1&1&0\\1&1&1\end{pmatrix}$ coveredBy
$\begin{pmatrix}1&1&1\\1&1&1\\1&1&1\end{pmatrix}$ overlap

Figure 2.1: Topological relationships and their 9-intersection matrix . Source: [33]

2.2 Multigranular data model

One of the motivating multigranular data models for this thesis is the model proposed in [73]. This model is general enough to deal with different domains of application and provides a general axiomatic-based definition of multigranular data. Relevant concepts about this model are detailed below.

2.2.1 Basic components of the model

The basic components of the multigranular model are the granularities and the granules that form them. Informally a granularity is a way of dividing a domain into granules that compose it, while a granule is a portion of the domain that does not overlap with any other granule of the same granularity. An example of granularity is the Region granularity in Figure 2.2, while granules are the particular regions (Metropolitana, Maule, etc. for the case of Chile) that compose the granularity .

A partial order relationship can be established between the granularities so that a granularity G is defined as finer than a granularity G' if for every granule in G, there is a granule in G' that contains it. Similarly, the granules also have a partial order structure given by the relationship that a granule may be contained in another granule, what it calls a subsumption relation.

In this multigranular model, a *granular space* is composed of a set of granules $GranuleSet$, the multigranular structure is defined as a pair $\mathfrak{G}$ = ($Grty$($\mathfrak{G}$),$GrAsgn$($\mathfrak{G}$)). $Grty$($\mathfrak{G}$) corresponds to a set of granularities where each one represents a different level of information detail. This set has a transitive and reflexive relationship as well as an upper limit. $GrAsgn$($\mathfrak{G}$) extends the idea of a domain assignment of a relational attribute to the multigranular context, also provides a basic order and assigns a set of granules to each granularity, therefore, $GrAsgn$ works as a mapping function $GrAsgn$: $GranuleSet \rightarrow 2^{Grty(\mathfrak{G})}$.

Figure 2.2 illustrates a granularity structure associated with electoral and administrative sub-divisions in Chile. The arcs in this figure represent relationships between granularities; in this example the granules that make up the province granularity are formed by granules belonging to the county granularity. In this figure the symbol $\top$ define a global granularity, which works as an upper limit in the set of granularities, that is, $GrAsgn$($\top$) = $GranuleSet$. The foregoing highlights that the main notion for working with granules in the space domain is that of space division [78], given that working with spatial granules is mainly a mapping of information instantiated to a specific space.

The multigranular model defines basic rules between granules in order to define the multigranular structure. This set of basic rules *PrBaRules* $< \mathfrak{G} >$ are the following:

- *Subsumption* rule $g_1 \sqsubseteq_{\mathfrak{G}} \bigsqcup_{\mathfrak{G}} S$: It indicates that the domain that represents the granule g_1 is contained in the domain represented by the union of the set S. Note that when $|S| = 1$, that is, it has only one element called g_2, it is equivalent to $g_1 \sqsubseteq_{\mathfrak{G}} g_2$.

- *Disjoint* rule $\bigsqcap_{\mathfrak{G}} \{g_1, g_2\} = \bot_{\mathfrak{G}}$: It indicates that the domain of the granule g_1 does

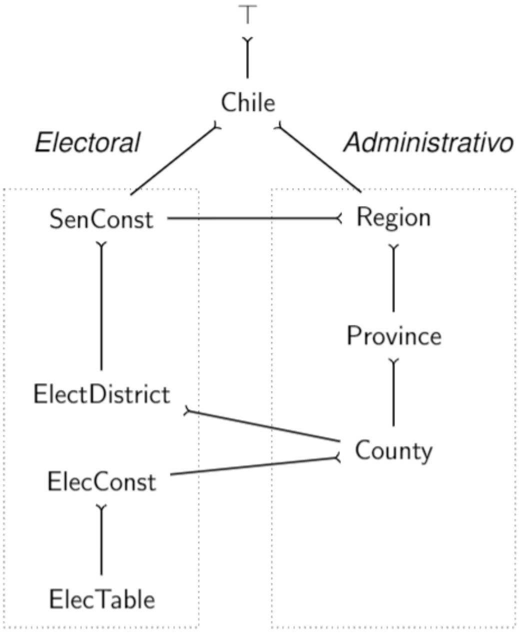


Figure 2.2: Electoral and administrative sub-divisions in Chile

not intersect with the domain of the granule g_2.

The combination of basic rules allows the definition of rules that enable to express conditions that were not treated in previous works that model granularity. An example of a more complex rule would be:

$$Chile = \biguplus Region_R \mid I \leq R \leq XVI$$

This rule expresses through the symbol $\bigsqcup$ that Chile is formed by the *join* of the 16 regions (represented by the number I to XVI) that currently comprise it, while the $\perp$ symbol indicates that the *join* is disjoint, that is, each region that makes up Chile is disjoint from the others. Notice that $\perp$ can be unnecessary to specify if the join is over granules of a same granularity, as it is the case of regions that make Chile. But in a general case, one could join granules of different belonging to different granularities.

2.2.2 Multigranular structure

One of the main components of the multigranular model is the multigranular structure, since it describes how the different levels of granularities are related while describing the relations between granules. In practical terms the multigranular structure is composed of two structures: (i) the **granularities structure**, which describes the type of relations between the granularities and the granules that compose them and (ii) the **granular structure**, which describes the subsumption and disjoint relation between the granules.

The work developed in [74] formalizes the above in [73]. It emphasizes that complex rules are defined on the basis of primitive rule sets. It also defines *join rules* as ($g \star \diamond S$), where $\star \in \{=, \sqsubseteq_{\mathfrak{S}}\}$ and $\diamond \in \{\bigsqcup_{\mathfrak{S}}, \overset{\perp}{\bigsqcup}_{\mathfrak{S}}\}$ being g a granule and S a set of granules.

In [74] the concept of pairs of granularities is introduced. This concept is defined as a pair $< G_1, G_2 > \in Grty(\mathfrak{S}) \times Grty(\mathfrak{S})$, such that $G_1 \neq G_2$, and based on this, a join rule will be called bigranular ($< G_1, G_2 >$) if *Head* $\langle \varphi \rangle \subseteq$ *Granules* $\langle\ \mathfrak{S} | G_1\ \rangle$ and *Body* $\langle \varphi \rangle \subseteq$ *Granules* $\langle\ \mathfrak{S} | G_2\ \rangle$, where $\varphi \in$ *join rules*. These bigranular join rules provide additional information to the already introduced "finer than" relationship between granularities. The work in [74] shows that bigranular join rules imply disjoint in logical terms, and that every bigranular join rule will be resolvable for the disjointness; that is, it is possible to establish in any case whether it is true or false the disjointness between granules. Different types of bigranular join rules are defined in [74] and, for each one of them, the minimum amount of relations between granules that needs to be stored to be able to answer the queries of subsumption and disjoint between granules.

An example of a bigranular join rules is the *Equality join order*: $G_1 \trianglelefteq^{\Phi}_{\mathfrak{S}} G_2$, which indicates that every granule g in G_2 is composed by the union (disjointed) of granules in G_1. *Equality join order* is quite common to find in practice, and indicates that knowing subsumption or not-disjoint is equivalent, so it is sufficient to store only one of them. The definition of bigranular join rules has a practical importance. In real world applications, it is common to deal with this type of rules, and the formalization of these allows the development of a practical implementation.

The structure of granularities provides information on how the granules are related to each other at different granularity levels, even though there are cases where there is no common pattern of relationship at the granularity level. For example, the national parks in Chile are not necessarily contained in a single region such that no bigranular join rule between the Region granularity and the NationalPark granularity can provide information regarding how their granules are related.

To the best of our knowledge, no research has been done on ways to decrease the information stored in the granular structure, that is why in the case that the granularities structure does not provide information regarding the relations between the granules of a bigranular join rule, all the relations between the granules are stored. This is a problem in real situations because of the large amount of data to be handled.

2.2.3 Inference rules and their implementation

To clarify notation, in what follows, $\perp_{\mathfrak{G}}$ represents disjoint, $\not\perp_{\mathfrak{G}}$ represents not disjoint, $\sqsubseteq_{\mathfrak{G}}$ represents subsumption and $\not\sqsubseteq_{\mathfrak{G}}$ represents not subsumption.

In [75], inference rules are defined, which are proven to be correct and complete, to derive relations of subsumption $\sqsubseteq_{\mathfrak{G}}$, not subsumption $\sqsubseteq_{\mathfrak{G}}$, disjoint $\perp_{\mathfrak{G}}$ and not disjoint $\not\perp_{\mathfrak{G}}$. Correctness indicates that the inference rules cannot derive false relations; whereas completeness indicates that these are all possible inference rules over the basic rules. The use of these rules allows a design that does not require storing all the relations between granules. The definition starts by using inference rules that make $\sqsubseteq_{\mathfrak{G}}$ and $\perp_{\mathfrak{G}}$ to be true (positive rules). They are derived from the transitivity of $\sqsubseteq_{\mathfrak{G}}$ and that if it is true that g'_1 and g'_2 are disjoint, then the respective granules $g_1 \sqsubseteq_{\mathfrak{G}} g'_1$ and $g_2 \sqsubseteq_{\mathfrak{G}} g'_2$ are disjoint too. Rules are added to derive unsatisfaction from relations (negative rules) and by exchanging premise predicates for conclusion. Note that the multigranular model contemplates the lack of information in the multigranular structure, so it will not always be possible to derive the information; due to this, the failure to derive a relation through these inference rules does not imply that the denial of that relationship is fulfilled (e.g., if nothing can be derived from rule 1, this does not imply that the not subsumption relation will be given between the granules).

The following are the inference rules defined in [75] to derive the possible relations between two granules. Later, in Chapter 4, the rules of inference are explored in detail.

$$\frac{g_1 \sqsubseteq_{\mathfrak{G}} g_2 \quad g_2 \sqsubseteq_{\mathfrak{G}} g_3}{g_1 \sqsubseteq_{\mathfrak{G}} g_3} \tag{2.1}$$

$$\frac{\bigsqcap_{\mathfrak{S}}\{g_1, g_2\} = \bot_{\mathfrak{S}}}{g_1 \not\sqsubseteq_{\mathfrak{S}} g_2} \tag{2.2}$$

$$\frac{g_1 \not\sqsubseteq_{\mathfrak{S}} g_2 \quad g_3 \sqsubseteq_{\mathfrak{S}} g_2}{g_1 \not\sqsubseteq_{\mathfrak{S}} g_3} \tag{2.3}$$

$$\frac{g_1 \sqsubseteq_{\mathfrak{S}} g_2 \quad g_1 \not\sqsubseteq_{\mathfrak{S}} g_3}{g_2 \not\sqsubseteq_{\mathfrak{S}} g_3} \tag{2.4}$$

$$\frac{\bigsqcap_{\mathfrak{S}}\{g_1, g_3\} \neq \bot_{\mathfrak{S}} \quad g_3 \sqsubseteq_{\mathfrak{S}} g_4 \quad \bigsqcap_{\mathfrak{S}}\{g_2, g_4\} = \bot_{\mathfrak{S}}}{g_1 \not\sqsubseteq_{\mathfrak{S}} g_2} \tag{2.5}$$

$$\frac{g_1 \sqsubseteq_{\mathfrak{S}} g_2 \quad \bigsqcap_{\mathfrak{S}}\{g_2, g_3\} = \bot_{\mathfrak{S}}}{\bigsqcap_{\mathfrak{S}}\{g_1, g_3\} = \bot_{\mathfrak{S}}} \tag{2.6}$$

$$\frac{g_1 \sqsubseteq_{\mathfrak{S}} g_2}{\bigsqcap_{\mathfrak{S}}\{g_1, g_2\} \neq \bot_{\mathfrak{S}}} \tag{2.7}$$

$$\frac{g_1 \sqsubseteq_{\mathfrak{S}} g_2 \quad g_1 \sqsubseteq_{\mathfrak{S}} g_3}{\bigsqcap_{\mathfrak{S}}\{g_2, g_3\} \neq \bot_{\mathfrak{S}}} \tag{2.8}$$

$$\frac{\bigsqcap_{\mathfrak{S}}\{g_1, g_2\} \neq \bot_{\mathfrak{S}} \quad g_2 \sqsubseteq_{\mathfrak{S}} g_3}{\bigsqcap_{\mathfrak{S}}\{g_1, g_3\} \neq \bot_{\mathfrak{S}}} \tag{2.9}$$

2.3 Multi-granular hierarchies

In contrast to what was presented in the previous section, the following details are necessary to understand the work developed in Chapter 5, which is based on the particular case of hierarchies related to spatial data.

The definition of spatial granularity [133] comes from the definition of temporal granularity by [12]. Formally, the spatial granularity is a function that maps non-overlapping portions, referred as granules of the spatial domain, into indexes or identifiers. [24] defined a spatio-temporal granule as a tuple (s, t), meaning that at time index t, the spatial index s is valid. [8] assigns to each spatio-temporal granule a sequence of spatial granules, one per temporal granule.

There exist several relations between granularities. Among them, a granularity P is said to be a partition of a granularity Q, if for each granule $g \in Q$, there exists a set S of granules in P whose geometric union makes up Q [12, 133, 24, 91]. This definition of spatial partition is a natural realization of a granularity, but the notion of granularity is more general because the set of granules that form the granularity may not cover the whole spatial domain.

Partitions have been an important notion to model the spatial domain [42, 96, 37]. Concepts of maps, resolution, spatial objects and topological reasoning build on partitions and their properties. [135] proposed a formalization based on the theory of rough sets [114] to deal with resolution and multi-resolutions in geographic spaces and vague spatial objects. In this work, a resolution is a finite partition of a set S of locations on a plane. Partitions can be organized in terms of a partial order relation; in this sense, the notion of resolution is equivalent to the notion of granularity.

The work in [14] proposes a taxonomy of granular partitions. This taxonomy classifies partitions in terms of: i) degree of structural fit, which refers to the concept of mereological structure; ii) degree of completeness and exhaustiveness of projection, where projection refers to the notion that objects are located at particular cells or granules of a partition; iii) degree of redundancy, in which cells may belong to different partitions.

As seen, multi-granular topological hierarchies, or restricted versions thereof such as spatial partitions, have been studied in the past from different communities, which emphasizes the importance of this model and its implementation.

Given a geographic connected region R, the formalization of a multi-granular spatial hierarchy is as follows. A partition $L = \{r_1, \ldots r_n\}$ is a granularity composed of regions r_i (called granules), such that (i) $\forall r_i, r_j \in L, r_i \mathring{\cap} r_j = \emptyset$ (i.e., regions are disjoint or touch each other, but they do not internally intersect) and (ii) $R = \bigcup_1^n r_i$ (i.e., the geometric union of regions makes the whole R). We will say that regions in L are *neighbors* if they share common boundaries. A partition can be seen as a planar graph, where nodes represent regions and an edge between two nodes indicates that the corresponding regions are *neighbors*.

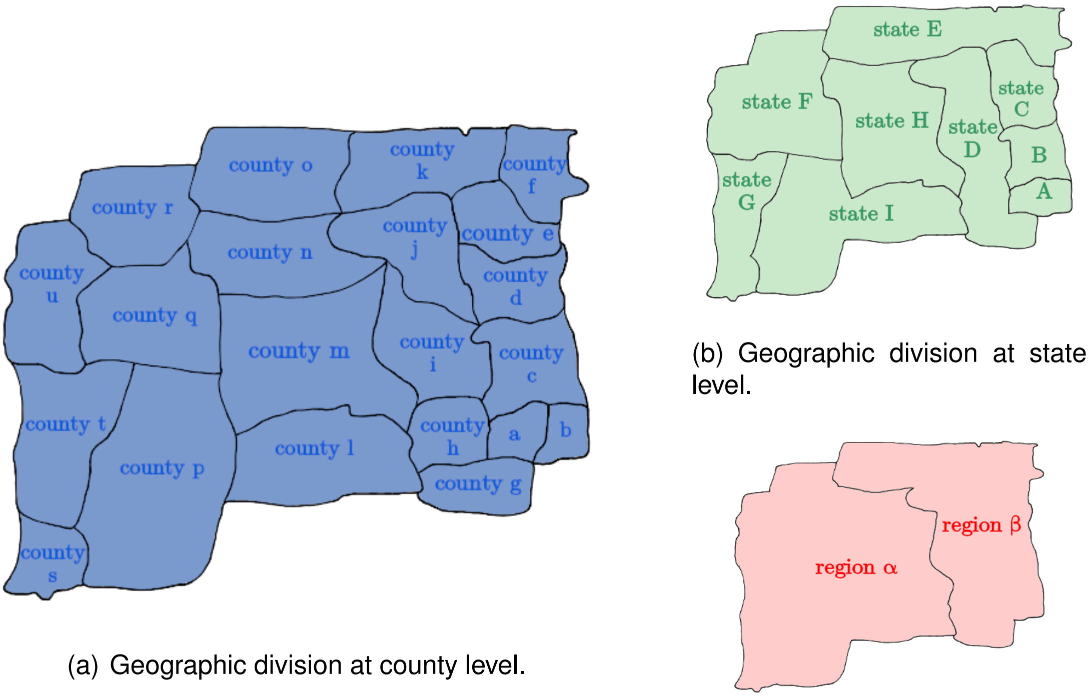
Partitions can be organized into hierarchical structures by inclusion relations. Let $L_1 = \{r_{1,1}, \ldots r_{1,n_1}\}$ and $L_2 = \{r_{2,1}, \ldots r_{2,n_2}\}$ be two partitions, with $n_1 \leq n_2$ being the number of regions per partition. Let $\mathsf{contains}(r, r')$ be a function that returns true if region r contains region r'. Then, L_1 is a coarser level of granularity than L_2, denoted by $L_1 \prec L_2$, if (i) $\forall r_{2,i} \in L_2,\ \exists r_{1,j} \in L_1$ such that $\mathsf{contains}(r_{1,j}, r_{2,i})$ holds (i.e., every region in L_2 is within a region in L_1) and (ii) $\forall r_{1,i} \in L_1,\ \exists S \subseteq L_2\ r_{1,i} = \bigcup_{r_{2,j} \in S} r_{2,j}$ (i.e., each $r_{1,i}$ is made of the union of regions in L_2). We can generalize to several partitions (granularities) $L_1 \prec L_2 \prec \cdots \prec L_h$, with L_1 being the coarsest or lowest level of granularity and L_h the finest or highest level of granularity. Figure 2.3 shows a spatial hierarchy composed of three granularity levels: L_1 is the region level (Figure 2.3(c)), L_2 is the state level (Figure 2.3(b)), and L_3 is the county level (Figure 2.3(a)), so $L_1 \prec L_2 \prec L_3$.

Based on this definition of a partition and of the multi-granular hierarchy, the following properties hold.

- Let $L_i \prec L_j$, with $i < j$, then for each $r' \in L_j$, there is only one $r \in L_i$ such that $\mathsf{contains}(r, r')$. Conversely, for each $r \in L_i$, there must be at least one $r' \in L_j$ such that $\mathsf{contains}(r, r')$.

- Because a partition L_i can be represented as a planar graph, with $n_i = |L_i|$ nodes and m_i edges (the number of pairs of neighboring regions in L_i), it holds $n_i < m_i \leq 3n_i - 6$.

- Let $r_j, r'_j \in L_j$ be regions of a partition, if there is an L_i such that $L_i \prec L_j$, then there exist $r_i, r'_i \in L_i$, not necessarily different, such that $\mathsf{contains}(r_i, r_j)$ and $\mathsf{contains}(r'_i, r'_j)$. Further, if r_j and r'_j are neighbors (i.e., they share a boundary) and $r_i \neq r'_i$, then r_i and r'_i must be neighbors. Further, when r_j and r'_j are neighbors, $\mathsf{contains}(r_i, r_j)$ holds and $\mathsf{contains}(r_i, r'_j)$ does not hold, we say that r_i and r'_j are neighbors as well.



(a) Geographic division at county level.

(b) Geographic division at state level.

(c) Geographic division at region level.

Figure 2.3: Example of a geographic division with aggregation levels Region, State, and County.

2.4 Succinct data structures and compact planar embeddings

The work presented in this thesis focuses on the development of algorithms and structures for the implementation of two multigranular models: i) A first general multigranular model based of work presented in [73] and ii) a second multigranular model focused on hierarchical structures of regions defined by the inclusion relation. The proposed data structures are based on several succinct data structures, since these show a significant space saving, while maintaining good access times to its elements. The concepts related to the succinct structures needed to understand the work done in this thesis are presented below.

2.4.1 Succinct data structures

A succinct data structure is an asymptotically *space-efficient* and *query-time-efficient* representation of a data structure. Space efficiency implies that the space used by the succinct data structure closely approaches the information-theoretic lower bound for that data structure. Specifically, if lwr is the information-theoretic lower bound, then a succinct data structure utilizes $lwr + o(lwr)$ bits. Query-time-efficient implies that the succinct representation attains the optimal query time achieved by other non-succinct data structures. For example, consider the representation of a binary tree with n nodes. In a traditional linked representation, where each node has references to its left-child, right-child and parent, the space complexity is $\Theta(n \lg n)$ bits under the RAM model with $\Theta(\lg n)$ bits per reference. In contrast, in a succinct representation of a binary tree, the consideration of only $C = \binom{2n+1}{n}/(2n+1)$ different binary trees leads to an information-theoretic lower bound of $lwr = \lg C$, which is less than $2n$ bits. Consequently, a succinct representation for a binary tree is expected to utilize $2n + o(n)$ bits while maintaining the ability to support operations in optimal time..

The research on succinct data structures has been broad, including succinct representation for text indexes [58, 107, 94, 63, 66], trees [82, 121, 102, 10, 60, 83, 90, 71, 44], and graphs [82, 28, 27, 6], among others [97, 99, 113, 62]. Those structures also show a good behavior in practice [46, 3].

2.4.2 Bitmaps

A Bitmap B is a sequence of bits that supports the following queries:

- $rank_b(B, i)$ returns the number of bits set to b in B between the positions 0 and i (both included). Consider the bitmap B = 1100110110. Then, $rank_1(B, 4)$ = 3. By default, if no b is specified, we consider that rank operation counts the number of ones.

- $select_b(B, i)$, which is the complementary operation of rank, returns the position in which the i-th bit of kind b is located in B. Following with the example used previously, $select_0(B, 3)$ = 6. Again, if no bit value is specified, the operation select searches for the i-th 1 by default.

- $access(B, i)$ checks the value of a given position i in the bitmap B. Following with the example, $access(B, 6)$ = 0.

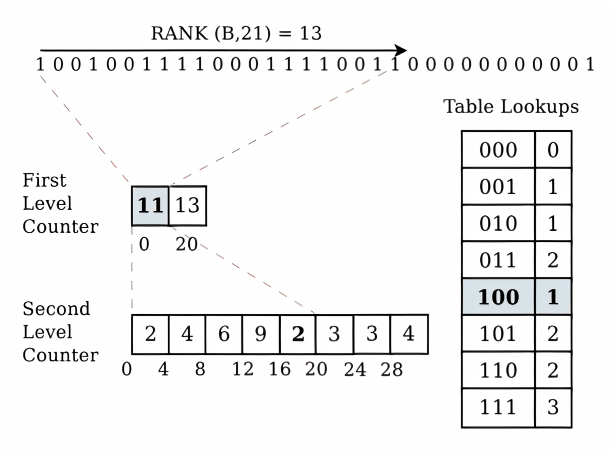


Figure 2.4: Example of two level directory, source [2]

The concepts of rank and select operations were initially introduced in [82]. In this paper, the first bitmap representation supporting the rank operation in constant time was defined. This representation comprises a two-level directory. At the first level, the results of $rank_1(B, i{-}1)$ are stored for each i that is a multiple of $s = \lfloor \log(n) \rfloor \lfloor \log(n)/2 \rfloor$. Figure 2.4 illustrates an example where the first level counter stores $rank_1(B, 19)$ and $rank_1(B, 31)$ with $s = 20$. The second level, associated with each block in the first level, stores various local rank results within its corresponding block. Specifically, it computes $rank_1(S_i, j - 1)$, where S_i is the sequence of bits between two counters of the first level, and j is a multiple of $b = \lfloor \log(n)/2 \rfloor$. Figure 2.4 provides an illustration of this second level directory. For example, the first value in this sequence is 2 because $rank_1(S_1, 3)$ = 2, where S_1 = 100100111...001. Using this directory, the operation $rank_1(B, i)$ can be solved. Firstly, the number of bits from the beginning of the sequence to the position of the nearest multiple of s less than i (referred to as p) is computed using the first level of the directory. Subsequently, the second level is employed to compute the number of ones from position $p + 1$ to the nearest multiple of b less than i (referred to as p'), and this count is added to the previous result. Finally, a lookup table is used to compute the number of ones from position $p' + 1$ to i. These bits work as an index to access a table storing the number of ones for every combination of bits, as depicted on the right of the figure.

The additional space required to facilitate rank operations in this solution encompasses the directory and the lookup table. In the first level of the directory, partial results for $\frac{n}{s}$ superblocks are stored. The cost of storing a counter for each of these superblocks is $\log(n)$ bits, resulting in a total space of O($\frac{n}{\log(n)}$) bits for the first level. In the second level, $\frac{n}{b}$ blocks are stored, with $\log(s)$ bits allocated for each, yielding a total space of O($n\frac{\log\log(n)}{\log(n)}$) for the second level. Lastly, the lookup table stores rank values for every combination of b bits, incurring a cost of O($2^b \cdot b \cdot \log(b)$) = O($\sqrt{n}\log(n)\log\log(n)$) bits. Considering these partial results, the overall space required to support rank and select operations with this approach is o(n).

In [112], a novel approach to storing bit sequences is introduced to reduce space utilization while maintaining comparable response times. The proposal involves dividing the bitmap into blocks of uniform size. Each block is represented by the count of bits with a value of 1 that it contains. A compression scheme is employed, clustering adjacent blocks into intervals of varying lengths. A different technique is presented in [119], where a bitmap compression method is based on a numbering scheme. Similar to the previous approach, the sequence is partitioned into blocks of the same size. Each block is associated with two values: c_i, representing the count of ones within the block, and o_i, serving as an identifier in a vocabulary comprising different combinations of bits. This vocabulary is sorted so that blocks with fewer ones or zeros have shorter identifiers. For a block length of u, the storage cost for each c_i is $\log(u+1)$ bits, and each o_i incurs a cost of $\log\binom{u}{c_i}$ bits. Various implementations exist in the current state of the art. For example, the work in [110] proposed a highly compressed solution specifically designed for managing sparse bitmaps. Another implementation tailored for sparse bitmaps is the gap encoding strategy, which encodes the gaps between sequences of consecutive bits with a value of 1 [122, 94].

2.4.3 Wavelet trees

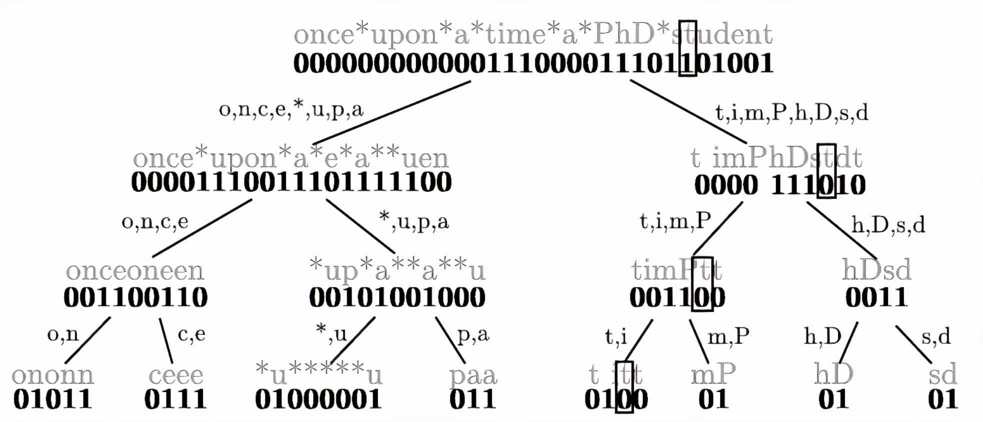
The wavelet tree (*wtree*) was introduced the first time in [65]. Previously, in Computational Geometry, [25] introduced a similar non-succinct data structure. Although the *wtree* was initially designed as a data structure for encoding a reordering of sequence elements [47, 65], its versatility has led to successful applications in various domains. Notably, it has been effectively employed in indexing documents [132], grids [108], and even sets of rectangles [22]. These are just a few examples of its wide-ranging applications; for a more comprehensive overview, we recommend referring to surveys such as [107, 95].

The inception of the wavelet tree (*wtree*) traces back to its introduction in [65], establishing itself as one of the most extensively studied succinct data structures. Prior to this, within the domain of Computational Geometry, a somewhat analogous yet less succinct data structure was pioneered by [25]. This data structure represents a set of points on a two-dimensional grid, detailing a successive reshuffling process where the points begin sorted by one coordinate and conclude sorted by the other. Although initially conceived as a means to encode sequence element reordering [47, 65], its adaptability has led to its successful deployment across various domains. Noteworthy applications include its utilization in document indexing [132], grid management [108], and even the representation of sets of rectangles [22]. These examples represent only a fraction of its versatile applications; for a more exhaustive exploration, we recommend consulting surveys such as [107, 95].

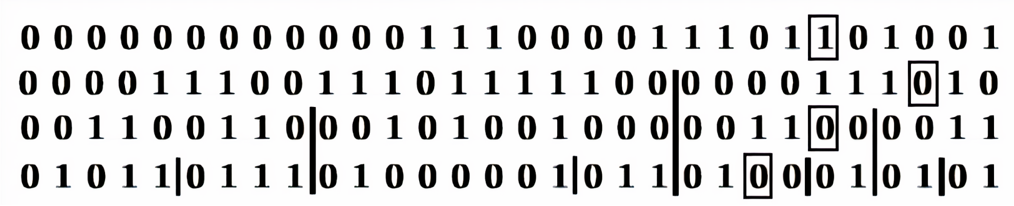
For the purpose of this thesis, a *wtree* is a sophisticated data structure crafted to efficiently manage a sequence of n symbols, denoted as $S = s_1, s_2, \ldots, s_n$, over an alphabet $\Sigma = [1..\sigma]$. It excels in supporting several key operations: $\mathtt{access(S,i)}$, which retrieves the symbol at position i in sequence S; $\mathtt{rank_c(S,i)}$, which tallies the occurrences of symbol c up to position i in sequence S; and $\mathtt{select_c(S,j)}$, which pinpoints the position in sequence S of the j-th appearance of symbol c. *wtree* structures are characterized by their ability to be stored in space bounded by various measures of the entropy of the underlying data, enabling compression.

The *wtree* is conceived as a balanced binary tree where each node represents a range $R \subseteq [1,\sigma]$ of the alphabet Σ. The left child of a node denotes a subset R_l, corresponding to the first half of R, while the right child represents a subset R_r, corresponding to the second half. Every node essentially denotes a subsequence S' of S, comprising symbols with values in R. This subsequence is stored as a bitmap, where a 0 bit indicates that position i belongs to R_l, and a 1 bit indicates that it belongs to R_r.

In this context, we focus on *wtree*s where the symbols of Σ are contiguous in $[1, \sigma]$. If they are not contiguous, a bitmap is utilized to remap the sequence to a contiguous alphabet [30]. Given these constraints, the *wtree* forms a balanced binary tree with $\lceil \lg \sigma \rceil$ levels.



Representation of a *wtree* using $O(1)$ pointers per node and its associated bitmaps. The subsequences of S in the nodes (gray font) and the subsets of Σ in the edges are drawn for illustration purposes.



```
0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 1 1 1 0 1 1 0 1 0 0 1
0 0 0 0 1 1 1 0 0 1 1 1 0 1 1 1 1 1 1 0 0|0 0 0 0 1 1 1 0 1 0
0 0 1 1 0 0 1 1 0|0 0 1 0 1 0 0 1 0 0 0|0 0 1 1 0 0|0 0 1 1
0 1 0 1 1|0 1 1 1|0 1 0 0 0 0 0 1|0 1 1|0 1 0 0|0 1|0 1|0 1
```

Representation of a *wtree* using one pointer per level and its associtaed n-bit bitmap. It can simulate the nagivation on the tree by using the rank operation over the bitmaps.

Figure 2.5: A *wtree* for the sequence $S =$“`once upon a time a PhD student`” and the contiguous alphabet $\Sigma = \{$`o,n,c,e,‘ ’,u,p,a,t,i,m,P,h,D,s,d`$\}$. We draw spaces using stars. Source [124]

In its most basic form, this structure demands $n\lceil \lg \sigma \rceil + o(n \lg \sigma)$ bits for the data and requires an additional $O(\sigma \lg n)$ bits to encode the tree's topology (assuming $O(1)$ pointers per node). It efficiently handles the aforementioned queries in $O(\lg \sigma)$ time by navigating the tree using rank/select operations on bitmaps with $O(1)$-time complexity [120]. An elementary recursive construction algorithm operates in $O(n \lg \sigma)$ time (not accounting for space-efficient construction algorithms [31, 128]).

The space required by the structure can be further reduced. The data can be compressed and stored in space bounded by its entropy through compressed encodings of bitmaps, another strategy to reduce space is to change the shape of the tree, in order to be able to store the tree in a single bitmap B. The time complexities do not change. However, in practice the operations are slowed down due to the extra rank operations needed to navigate. Additionally, the removal of the $O(\sigma \lg n)$ bits of the topology can also be achieved using the wavelet matrix, the idea focuses on organizing all the 0s to the left of all the 1s within a row. This reordering causes the wavelet tree nodes to disperse across the level bitmap while preserving their sequential order. Consequently, the wavelet matrix can emulate any algorithm originally designed for the wavelet tree.

Figure 2.5 shows an example of two *wtree* representations for the sequence $S =$ $``once\ upon\ a\ time\ a\ PhD\ student"$. The first part of Figure 2.5 shows the $O(1)$-pointers-per-node representation, while the seconds part shows the one-pointer-per-level representation.

The following is the traversal performed by the operation $\mathtt{access}(\mathtt{S}, 24)$ in both representation. In the first representation, the query proceeds as follows: Let $curr$ be the root, B_{curr} be the bitmap of the current node, $i = 24$ be the index of interest, R be the range $[0, \sigma - 1] = [0, 15]$, and $rankc(Bcurr, i)$ be the count of c-bits up to position i in B_{curr}. Initially, we inspect the bit $B_{curr}[i]$. Since the bit is 1, we recompute $i = rank1(Bcurr, i) - 1 = 7$, update $curr$ to be the right child of $curr$, and halve $R = [8, 15]$. This process is repeated. Since $B_{curr}[i] = 0$, $i = rank0(Bcurr, i) - 1 = 4$, $curr$ is updated to be the left child of $curr$, and $R = [8, 11]$. Again, with $B_{curr}[i] = 0$, $i = rank0(Bcurr, i) - 1 = 2$, $curr$ is changed to be the left child of $curr$, and $R = [8, 9]$. Finally, in the last level, with $B_{curr}[i] = 0$, the range is $R = [8, 8]$, and the answer for $\mathtt{access}(\mathtt{S}, 24)$ is $\Sigma[8] =`\ t\ '$. Both $\mathtt{rank_c}(\mathtt{S}, \mathtt{i})$ and $\mathtt{select_c}(\mathtt{S}, \mathtt{i})$ involve similar traversals to $\mathtt{access}(\mathtt{S}, \mathtt{i})$. For a more detailed explanation of *wtree* operations, refer to [107].

In the one-pointer-per-level representation, the procedure is akin to the one described earlier, with the exception that the traversal of the tree must be simulated using rank operations over the bitmaps [30].

The *wtree* supports more complex queries beyond the basic primitives described earlier. For instance, research in [94] demonstrated its connection with a classical two-dimensional range-search data structure. More examples of more complex queries are shown in [59], where new algorithms for WT are defined like $range\_report$, $range\_quartile$ or $range\_intersection$. In addition, WT applications to information retrieval are presented.

The utilization of *wtree* has been extensively explored across various contexts, leading to ongoing research aimed at refining *wtree* algorithms and methodologies. In recent studies such as [84, 36, 100], novel strategies have been proposed to enhance the efficiency of *wtree* construction in practical applications. Additionally, in works such as [51, 52, 49], different algorithms and approaches for constructing *wtree* on multicore architectures have been investigated. Furthermore, *wtree* finds applications in diverse domains, as demonstrated by studies such as [86], where a parallel wavelet tree algorithm utilizing Map-Reduce hybridization is proposed for constructing a textual search index,[127], where a new technique employing Word-Based Tagging Coding compression is implemented using Parallel Wavelet Tree to enhance exact matching results in compressed text, and[137], which presents an efficient recursive parallel wavelet tree algorithm for indexing and searching geographical documents. These examples represent just a fraction of the recent research endeavors that leverage and propose novel algorithms based on the *wtree*.

2.4.4 Compact trees

Trees are fundamental data structures in computer science, and they play a crucial role in various applications. One efficient way to represent a tree with n nodes is to use 2n bits instead of O(n) pointers. Generally, two types of trees are distinguished: ordinal trees and cardinal trees. i) **Ordinal Trees**: Ordinal trees allow for an arbitrary number of children per node, but they only distinguish the order of these children; the main focus is on the relative order of children rather than restricting the number of possible children for each node. ii) **Cardinal Trees**: Cardinal trees have a fixed set [1, θ] of types of children, and each node might have one or no child of each type; they are more rigid structures than ordinal trees. In summary, ordinal trees provide flexibility in the number of children per node but emphasize the order, while cardinal trees impose a fixed set of possible children types on each node. The choice between these tree types depends on the specific requirements and constraints of the problem at hand.

For the present work, the compact representation of ordinal trees is of interest(this is because the work developed in Chapter 5 is based on a compact representation). In the following, compact representations for ordinal trees will be described. Table 2.1 shows the operations commonly supported by the various representations of ordinal trees.

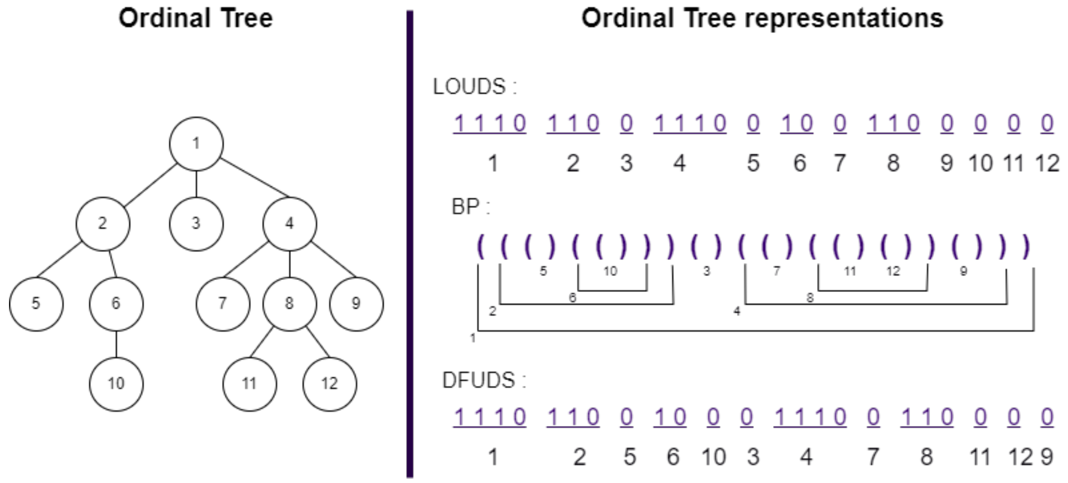


Figure 2.6: Example of ordinal tree and their compact representations

LOUDS representation

The work done for Jacobson in [82] introduced the first succinct tree representation for ordinal trees. This representation relies on the Level-Order Unary Degree Sequence (LOUDS) of a tree. The LOUDS approach involves visiting the nodes in a level-order traversal, encoding their degrees in unary, capturing the number of children each node has. In a level-order traversal, the root is visited first, followed by its children from left to right, and subsequently, all nodes at each successive level. This encoding requires 2n+O(1) bits to represent an ordinal tree with n nodes, additional o(n) bits are used to support rank and select operations on the encoding; an example of this representation can be seen in the Figure 2.6. Jacobson initially analyzed these operations in the bit probe model, demonstrating their support in O($\lg n$) probes. Subsequently, Clark and

Munro [29] extended this capability by showing how to achieve O(1) time complexity for rank and select operations under the word RAM model, considering a word size of $\theta(\lg n)$.

As a result, the LOUDS representation excels in providing efficient O(1) time complexity for parent, child, and child rank operations. Beyond these fundamental operations, it is straightforward to extend support to rank_level, select_level, level_succ, and level_pred_operations, all achievable in constant time.

	Operation	**Description**
1	$parent(x)$	Report the parent of node x
2	$child(x, i)$	Find the ith child of node x
3	$child\_rank(x)$	Report the number of left siblings of node x
4	$degree(x)$	Report the degree of node x
5	$depth(x)$	Report the depth of node x
6	$desc(x)$	Report the number of descendants of node x
7	$level\_ancestor(x, i)$	Find the ancestor of node x that is i levels above node x
8	$subtree\_size(x)$	Report the number of nodes in the subtree rooted at node x
9	$height(x)$	Report the height of the subtree rooted at x
10	$deepest\_node(x)$	Find the deepest node in the subtree rooted at node x
11	$lca(x, y)$	Find the lowest common ancestor of nodes x and y
12	$lmostleaf(x)$ /$rmostleaf(x)$	Find the leftmost/rightmost leaf of the subtree rooted at node x
13	$leafrank(x)$	Report the number of leaves before node x in preorder
14	$leafselect(i)$	Find the ith leaf from left to right
15	$prerank(x)$/$postrank(x)$	Report the number of nodes preceding node x in preorder/postorder
16	$preselect$/$postselect(i)$	Find the ith node in preorder/postorder
17	$levellmost(i)$/$levelrmost(i)$	Find the leftmost/rightmost node among all nodes at depth i
18	$levelsucc(x)$/$levelpred(x)$	Find the node immediately to the left/right of node x among all nodes at depth i

Table 2.1: Commonly supported operations for compact representations of ordinal trees.

Balanced Parenthesis representation

The work done for Munro and Raman in [102] proposed another type of succinct representation of trees based on the isomorphism between balanced parenthesis sequences (BP) and ordinal trees. The BP sequence of a given tree is derived by performing a depth-first traversal. For each visited node, an opening parenthesis is written, followed by a closing parenthesis immediately after all its descendants are visited. This process results in a $2n$-bit encoding of a tree with n nodes, represented as a sequence of balanced parentheses. Identification of a node can be accomplished, for instance, by referencing the position of the opening parenthesis within the corresponding pair; an example of this representation can be seen in the Figure 2.6. The work of Munro and Raman, building on Jacobson's ideas [82], introduced a succinct representation of an ordinal tree with n nodes in $2n$ + o(n) bits based on a balanced parenthesis (BP) sequence. This representation supports various basic operations in constant time using an auxiliary structure of size o(n) bits. By showing how to translate operations performed on the tree into basic operations on the parenthesis sequence that represents it, Munro and Raman [102] presented a succinct representation of an ordinal tree with n nodes in $2n$ + o(n) bits based on BP, which supports parent, desc, depth, rankpre/post and selectpre/post in constant time, and child(x, i) in O(i) time. Munro et al. [103] enhanced the previous work by introducing constant-time support for leaf rank, leaf select, left leaf, right leaf, and leaf size on the BP representation. This improvement involved the use of o(n) additional bits, which were also utilized in designing space-efficient suffix trees. Chiang et al. [27] demonstrated constant-time support for the degree operation. Munro and Rao [104] extended support to level ancestor, level_succ, and level_pred in constant time, with applications in the succinct representations of functions. Lu and Yeh [90] presented constant-time support for child, child rank, height, and LCA (lowest common ancestor) operations.

Depth-First Unary Degree Sequence representation

In the work presented by Benoit et al. [10], a novel tree representation called Depth-First Unary Degree Sequence (DFUDS) was introduced. This representation, derived by traversing nodes in depth-first order (preorder) and encoding their degrees in unary, differs from the commonly used LOUDS encoding. In the DFUDS sequence, a node with a degree of d is represented by d opening parentheses followed by a closing parenthesis, listing nodes in preorder. The representation lists all nodes in preorder, starting with an additional opening parenthesis added at the beginning of the sequence; an example of this representation can be seen in the Figure 2.6. The DFUDS number of a node is determined by the rank of the opening parenthesis in its parent's description corresponding to the node. The work in [10] presented a succinct tree representation based on DFUDS using 2n + o(n) bits, fundamental operations like child, parent, degree, and desc are supported in constant time. Each node is identified by referencing the position of the first parenthesis in its representation. Jansson et al. [83] extended the DFUDS representation by incorporating o(n) additional bits. This extension enabled constant-time support for various operations, including child rank, depth, level ancestor, LCA (lowest common ancestor), left leaf, right leaf, leaf rank, leaf select, leaf size, rankPRE, and selectPRE. Barbay et al. [7] demonstrated how to support rankDFUDS and selectDFUDS operations, facilitating constant-time conversions between the preorder number and DFUDS number for the same node. This constant-time conversions provided the basis for strategies supporting queries in labeled trees, as proposed in [7].

Range min-max tree

The most practical compact representation for ordinal trees is the *range min-max tree* (RMMT) [109]. The RMMT is a complete binary tree that stores some statistics about the number of opening and closing parentheses of the balanced parenthesis sequence B. The compact tree is built upon a basic operation called excess, defined as $\text{excess}(i) = \text{excess}(i-1) + 1$ if $B[i] =$ '(', or $\text{excess}(i) = \text{excess}(i-1) - 1$ if $B[i] =$ ')'. The sequence B is virtually divided into blocks of length l, where each block is represented by a leaf of the RMMT. For the leaf v associated with a block $B[s..e]$, the whole excess, defined as $v.e = \text{excess}(e) - \text{excess}(s-1)$, and the minimum excess value of the leaf, defined as $v.min = \text{min}_{i \in [s,e]}\{\text{excess}(i) - \text{excess}(s-1)\}$, are stored. For an internal node u with left child u_l and right child u_r, the whole and minimum excess values are also stored, defined as $u.e = u_l.e + u_r.e$ and $u.min = \min\{u_l.min, u_l.e + u_r.min\}$, respectively. Once those values are stored in the RMMT, the operations find_open, find_close and `enclose` are reduced to the primitive operations $\texttt{fwd\_search}(B, i, d)$ (the leftmost position $j > i$ in B, such that $\text{excess}(i) + d = \text{excess}(j)$, with $d < 0$) and $bwdsearch(B, i, d)$ (the rigthmost position $j < i$ in B, such that $\text{excess}(i) + d = \text{excess}(j)$, with $d < 0$). Figure 2.7 present an example of a range min-max tree, using the ordinal tree presented in figure 2.6, in this example each block is of size 6, where e is the whole excess and min is the minimum of each block. Both primitive operations scan sequentially a constant number of blocks of B and move up and down in the RMMT looking for the answer, spending $O(l + \lg \frac{n}{l})$ time. We assume $l = \Theta(\log n)$ in this paper, so the time is $O(\log n)$ and the space is $O(n)$ bits. These complexities can be reduced to $O(1)$ and $o(n)$ by means of more complex data structures [109].

Within the same time and space complexities, the RMMT can also support $rank_{(}(B, i)$ and $select_{(}(B, i)$ by storing a new field n' on each node of the RMMT. For each leaf v of the RMMT, $v.n'$ stores the number of opening parentheses in the block $B[s..e]$ associated with v, and for each internal node u with left and right children u_l and u_r, we store $u.n' = u_l.n' + u_r.n'$.

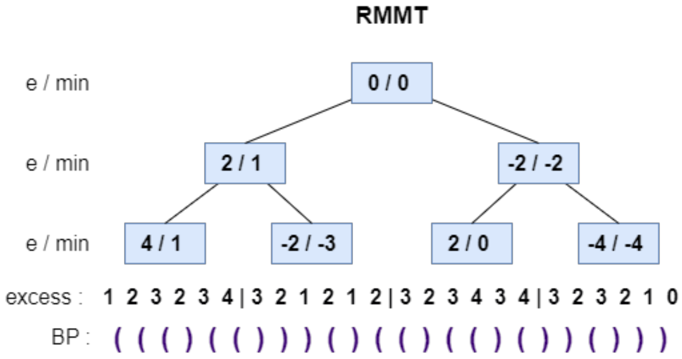
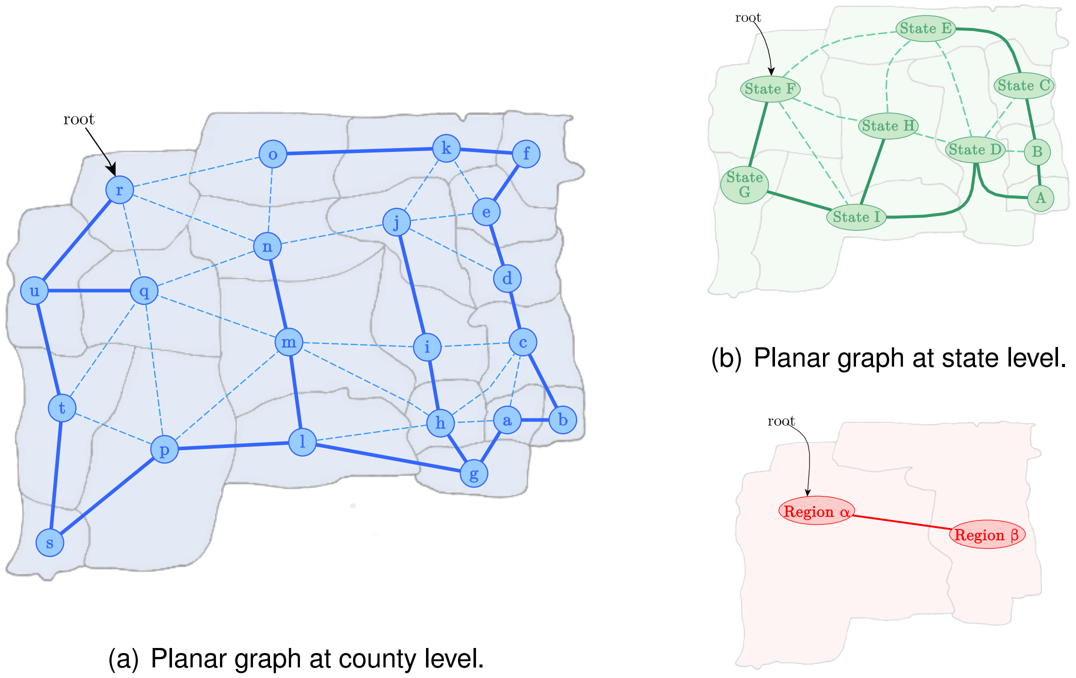

RMMT
e / min
0 / 0
e / min
2 / 1
-2 / -2
e / min
4 / 1
-2 / -3
2 / 0
-4 / -4
excess : 1 2 3 2 3 4 | 3 2 1 2 1 2 | 3 2 3 4 3 4 | 3 2 3 2 1 0
BP : ((( ) (( ))) ( ) (( ) (( ) ( )) ( )))

Figure 2.7: Example of Range min-max tree, using Ordinal tree presented in image 2.6

All previous operations can be applied to a sequence $S[1..n]$ composed by two intertwined balanced parenthesis sequences, B and B^*. For convenience, B is represented with parentheses, B^* with brackets (for the remainder of the paper, parentheses refers



(a) Planar graph at county level.

(b) Planar graph at state level.

(c) Planar graph at region level.

Figure 2.8: Planar graph representations of the aggregation levels of Figure 2.3. Spanning trees are represented with thick edges.

to round brackets $()$, while brackets refers to square brackets $[]$), and the intertwine is represented with a bitmap $A[1..n]$, such that $A[i] = 1$ iff $S[i] =$ '(' or $S[i] =$ ')', and $A[i] = 0$, otherwise. Thus, the operation $rank_{()}(S, i)$ (the number of opening or closing parentheses in S up to position i) is supported in constant time by $rank_1(A, i)$. Similarly, $select_{()}(S, i)$ (the position of the i-th opening or closing parenthesis in S) is supported in constant time by $select_1(A, i)$. In the same way, for $S[i] =$ ')', $\mathsf{find\_open}(S, i)$ is mapped to $select_1(A, \mathsf{find\_open}(B, rank_1(A, i)))$. Operations find_close and `enclose` are supported similarly.

2.4.5 Compact planar embeddings

Compact representations of planar-graph embeddings have been studied [98, 105], an example of planar-graph embeddings can be seen in Figure 2.8, which shows the induced planar embedding of the geographic area of Figure 2.3.

Tutte [131] demonstrated that representing a specific embedding of a connected planar multigraph with n vertices and m edges requires approximately $m \log 12$ or 3.58m bits in the worst case. Turán [130] provided a simple representation using 4m bits. Jacobson [82] proposed a compact representation using O(m) bits, designed for efficient navigation, and based on book embeddings [139]. Munro and Raman [102] proposed a representation that 2m + 8n + o(m) bits instead of the estimated 64n bit used by the Jacobson's representation, while still maintaining fast navigation, and based on the same book embeddings. Keeler and Westbrook [85] emphasized that the constant factor in [82] space bound is relatively large and proposed a representation using $m \log 12$ + O(1) bits for planar graphs (not embeddings). This applies to planar embeddings containing either no self-loops or no vertices with degree 1; however, they sacrificed fast navigation in this approach. Chiang, Lin, and Lu [27], building upon prior work by Chuang et al. [28], presented a representation (without allowing self-loops) that utilizes 2m+3n+o(m) bits with fast navigation, based on so-called orderly spanning trees. It's worth noting that, while all planar graphs can be represented with orderly spanning trees, some planar embeddings cannot. For simple planar embeddings (with no self-loops or multiple edges, hence $m \leq 3n$), the representation proposed by Chiang, Lin, and Lu [27] achieves a space of 2n + 2m + o(m) $\leq$ 4m + o(m) on connected graphs. Barbay et al.[6] proposed a data structure that employs O(m) bits to represent simple planar graphs while ensuring fast navigation. However, it is important to note that their constant factor is relatively large, specifically $18n$ + o(m).

In a significant breakthrough, Blelloch and Farzan [16], building upon the work of Blandford et al.[15], achieved a remarkable result by matching Tutte's lower bound on general planar embeddings. Their structure utilizes $m \log 12$ + o(m) bits and supports efficient navigation, relying on small vertex separators [89]. Notably, this representation can handle any planar graph within its lower-bound space plus a sublinear redundancy, even in cases where the exact lower bound is unknown for general planar graphs [17]. Although Blelloch and Farzan successfully addressed the problem theoretically, their representation is intricate and has not been implemented. Other researchers, such as He et al.[72], Aleardi [1], Fusy et al.[57], and Yamanaka [138], have explored specific types of planar graphs, focusing on tri-connected planar graphs and triangulations.

Turán's representation [130] is noteworthy for its simplicity. The representation is constructed in two stages. In the first stage, an arbitrary spanning tree T of the planar embedding is computed. In the second stage, a depth-first search (DFS) traversal is performed, generating a sequence S of length $2m$, where m is the number of edges in the planar embedding. Throughout the traversal, an edge in T is denoted by either '(' or ')', depending on whether it is the first or second time the edge is visited. Similarly, an edge not in T is represented by '[' or ']'. Utilizing two bits per symbol in S, the representation requires $4m$ bits of space.

Although Turan's representation does not provide primitives to navigate the graph, [48] augmented it using compact representations of trees and bitmaps that add up $o(m)$ extra bit, and enable navigation operations. The work is based on the idea that if we perform a depth-first traversal of a spanning tree T of G, starting from any vertex on the outer face of G and always process the edges incident to the vertex v we are visiting in counter-clockwise order, then each edge not in T corresponds to the next edge we cross in a depth-first traversal of the complementary spanning tree T^*. The extended representation lists the incident edges of a vertex and the edges bounding a face both in $O(1)$ time per edge, computes the vertex degree in any time in $\omega(1)$, and checks whether two vertices are neighbors in any time in $\omega(\log m)$. Later, [55, 56] improved the time bounds and extended the representation to support the topological model (without multi-granularity) using $4m + o(m)$ bits and offering relevant time guarantees; recall Table 2.2. Hereinafter, we refer to their work as PEMB.

In Chapter 5, we present a generalization of PEMB in order to support multi-granular hierarchies of spatial objects. A map with several levels of granularity can be seen as a set of planar embeddings, one per level, plus the information about containment relationships among levels. The embedding of level i has n_i nodes and m_i edges. A straightforward representation consists of using PEMB to represent each planar embedding of the collection (using $4m_i + o(m_i)$ bits per level i), plus $h - 1$ integer vectors to store the region of the preceding level that contains each region. This arrangement, for example, supports the query contains in $O(h)$ time. Its main drawback is the space consumption of the vectors, $\lceil n_i \log n_{i-1} \rceil$ bits at each level $i > 1$.

The design of compact data structures to manage spatial and spatio-temporal data

Operation	Complexity
Do regions r_1 and r_2 share a boundary?	any in $\omega(1)$
Is boundary e on the border of region r_1?	$O(1)$
Regions separated by boundary edge e	$O(1)$
Boundary edges of region r_1	$O(1)$ per boundary edge
Regions adjacent to region r_1	$O(1)$ per region
Number of regions adjacent to region r_1	any in $\omega(1)$

Table 2.2: Topological operations considered in [55, 56]. Let r_1 and r_2 be two regions and e be a boundary edge.

goes beyond the planar-graphs applications described above, and indeed it has been an active area of research in the last few years. There are some outstanding results both for the raster model [19] and for the vector model [22]. Also, there exist compact representations for trajectories of moving objects in free space [21] and when the movement is restricted to a network [20]. In [126], algorithms for solving spatial joins between raster and vector datasets on top of compact representations are presented. The underlying motivation of all those works is to provide data structures for different spatio-temporal data types with fast query time in small space.

Chapter 3

Related work

This section reviews related work concerning the implementation of multigranular models. Specifically, it describes extensions to the datawarehouse model focused on managing multigranular data, multigranular data models for spatio-temporal data, and mechanisms for dealing with uncertainty when operating with data at different levels of granularity.

3.1 Extension of Datawarehouse models

In the world of databases, a well-known model focused on answering queries at different levels of aggregation is the datawarehouse model [80], which is characterized by storing complete information at the most detailed level and then pre-compute aggregations for the defined information dimensions. In [78], extensions of the datawarehouse schema are proposed by adding the granularity attribute and modifying the fact table. This allows us to exploit the multigranular nature of the data beyond what schemata such as the star schema or the snowflake schema are capable of, because these schemata require a rigid structure of the levels of detail. The proposed multigranular schema stores information at different levels of granularity, however and unlike the model in [74, 75], it does not specify relationships through join or disjoint join rules, but only based on subsumption, and it focuses on data that do not change or suffer updates.

In the context of multigranular data aggregation in datawarehouse, a practical approach is provided by [79]. The approach is mainly focused on a single dimension based on data aggregation in relational systems. This work focuses on proposing multigranular solutions for the time dimension (table commonly used to store time stamps). They propose to add the *granularity* attribute to the table, which indicates for each tuple the level of granularity it holds, and provide different algorithms to perform the aggregation of data according to this new proposed attribute and design options for the fact table to handle data at different levels of granularity.

The work in [77] presents a multi-dimensional and multi-granular schema (MMDW) for data warehousing. MMDW is an extension of the standard star schema and it is based on ROLAP. MMDW has the ability to store the multi-granular data and to aggregate data on-the-fly or to pre-compute and store the aggregated data at any desired combination of dimensions and granularities for analysis and reporting purposes. The advantages of MMDW are that it is a general model so it can be used in various real applications. It also offers better query times and uses less space than other models used in data warehouses. Despite of this, the model still occupies excessive space due to the fact that it pre-computes information among all the possible combinations of granularities and, although it allows calculating information at the moment of being consulted, it does not provide algorithms that simplify this computation. Finally, it is based on performing the corresponding aggregation operation through the required granularities.

3.2 Multigranular Spatio-Temporal models

The notion of granularity appears in the spatio-temporal information domain. A highly referenced paper on temporal granularity is [12]. This work introduces the concept of spatial granularity, which refers to a mapping function from a domain of indexes to a subset of the temporal domain. It also introduces the concept of granule, which is the element that composes a granularity and refers to a portion of the temporal domain to be represented. The work in [12] also defines a set of granularity relationships, which are relationships between the granules composing two granularities that have the same time domain. In a similar way, the work in [133] defines spatial granularity as a mapping from a domain of indexes to portions of a space, called spatial granules. It defines methods to navigate between granularities and highlights the problem of uncertainty when moving between granularities from greater to less detail.

The paper in [13] describes formalisms and the way to work with temporal granularities, which is described as the qualification of sentences to different temporal granularities and the definition of the relationships between the different temporal granularities. Basic components are proposed to model temporal granularity, emphasizing the concept of language, which refers to the approach used to represent information at different granularities. Two approaches to languages can be distinguished: i) The quantitative approach, which consists of a model capable of positioning temporary entities within a metric framework. It can be based on two approaches, a logical approach or one based on set theory, and ii) The qualitative approach, which consists of a model capable of characterizing the position of temporary entities with respect to others, this characterization can be both topological and vectorial.

In [43] an extension of the relational model capable of handling temporal granularity is proposed. Such model is summarized as a database in which each tuple is a timestamp under some defined granularity. In [134] a method is developed to answer questions about this model. The answers are computed based on the hypothesis associated with the database instance, handling the missing values, considering them constant or interpolating them. The authors propose an arc-consistency algorithm to check when granularities are periodic with respect to a common finer granularity, while proposing an approximate algorithm to handle the constraints of each granularity and the transformation of information from one granularity to another.

In [45] a conceptual model for handling multigranular spatial data is proposed. The authors propose to work with two types of granularity dimensions, the spatial granularity that focuses on the possible variations of the geometry of an object with respect to different scales, and the semantic granularity that focuses on the possible variations of the set of domain objects with respect to the levels of detail requested by different users/applications.

In the context of instances with spatial/temporal data, the work in [24] defined a spatio-temporal granule as a tuple (s,t), meaning that at time index t, the spatial index s is valid. In a similar way, the work in [8] assigns to each spatio-temporal granule a sequence of spatial granules, one for each granule in the temporal granularity. In [11] a representation of multigranular spatio-temporal data is proposed, this work proposes the use of a meta-scheme in conjunction with the relational model and defines a new type of data with which they are able to represent the space-time granularity. It also provides tools to operate on objects with different granularities in order to address their integration and inter-operability.

The study presented in [26] introduces a formal framework designed to extend the granularity conversion and granular comparison of spatio-temporal data across heterogeneous granularity systems. This framework starts by generalizing coexisting granularity systems to accommodate their heterogeneity. It defines a graph model to represent the structures of these systems, capturing the semantics and uncertainty associated with granularity conversions. The framework introduces two key constraints for composing inter-system granularity conversions: semantic preservation and semantic consistency. The study establishes that granularity systems can be combined, only if they adhere to either semantic preservation or semantic consistency, in addition to globally sharing a common refined granularity.

In the context of granularity, prior research has introduced the concept of relations between granularities, enabling the characterization of structures that organize the domain. One such notion is spatial partition, where if granularity P is a partition of granularity Q, then for each granule $g \in Q$ there exists a subset S of granules in P such that g is the union of non-overlapping elements of S [12, 133, 24, 91]. Note that this definition of spatial partition is a natural realization of a granularity, but the notion of granularity is more general in the sense that the set of granules that form the granularity may not cover the whole domain. Even more, partition has been an important notion for the spatial domain [42, 96, 37]. Concepts of maps, resolution, spatial objects and topological reasoning use partitions and their properties. The work in [135] proposes a formalization based on the theory of rough sets [114] to deal with resolution and multiresolutions in geographic spaces and vague spatial object. In this work, a resolution is a finite partition of a set S of locations on a plane. Partitions can be organized in terms of a partial order relation, and in this sense, the notion of resolution is equivalent to the notion of granularity. A taxonomy of granular partitions is found in [14]. This taxonomy classified partitions in terms of: i) Degree of structural fit, which refers to the concept of mereological structure. ii) Degree of completeness and exhaustiveness of projection, where projection refers to the notion that objects locate at a particular cells or granules of a partition. iii) Degree of redundancy in which cells may belong to different partitions.

3.3 Inference mechanisms

Database systems are traditionally seen as passive repositories with capability for efficient query processing. However, there also exist deductive database systems where new knowledge can be derived. Deductive databases can be seen as an extension of databases with rules [118]. The multigranular model presented in Section 2.2 have as basic information the subsumption and disjoint relations between its elements, and proposes the idea of not storing all the relations explicitly, but a subset and derive the rest through the application of several inference rules. In general, subsumption and disjointness are important relations for modeling multigranular data, where granules represent a portion of a domain that must be disjoint and can be organized in terms of a partial order structure by subsumption [73]. Granular data is of particular interest in the spatial domain due to the common organization of spatial objects through inclusion and aggregation, such as it is the case of a political-administrative subdivision [24]. Beyond the spatial domain, disjointness and subsumption appear when dealing with semantic networks in knowledge representations and conceptual modeling in database theory. In this context, subsumption corresponds to the is-a relation saying that an element of a given type is also an element of another type, and disjointness establishes that two types do not share common elements.

Associated with subsumption and disjointness, there are inference mechanisms to derive new knowledge [76], which also avoid representing explicitly all possible relations at once. For hierarchical structures, representing explicitly subsumption relations through foreign keys in the relational database context is usual; however, combining subsumption and disjointness scale up to represent a very large number of relations.

Focusing on the spatial domain, spatial reasoning tries to determine if a set of topological relations defined over a set of spatial objects is said to be consistent, meaning that there are no contradictions among the topological relations, or if the set of topological relations satisfy a consistency network [92, 88]. For example, given three objects x, y and z, and the relations x is in y, y is in z, and x is disjoint to z, the dataset is topologically inconsistent since the first two relations imply that x is in z holds. As this example shows, checking topological consistency uses the notion of *composition* of topological relations, which is basically one type of inference rule. Composition is such that given the relations between objects x and y, and between y and z, one can derive the relation between x and z [9].

In addition to the use of inference rules in the spatial domain, the work in [4, 5] provides a set of inference rules for is-a and disjoint relations and shows their soundness and completeness. It uses set-theory and analyzes negative terms as complement of a given set. Similarly, the work in [18] proposes a set of inference rules for afunctional dependencies (afds) together with functional dependencies (fds) and proves that they are sound and complete.

3.4 Handling uncertainty in multigranular data

In the realm of managing information with uncertainty, one widely used model is rough sets. A rough set is denoted by a pair of crisp sets known as *lower and upper approximation*. The lower approximation represents the set of objects that unequivocally belong to the target set, while the upper approximation encompasses objects that may potentially belong to the target set. An insightful characterization of rough sets, introducing the concept of types (originally referred to as kinds), is presented in [115]. Two distinct approaches are employed for characterizing rough sets: the accuracy coefficient, which measures how closely the rough set approximates the target set, and the topological characterization introduced through the notion of types. Practical applications of rough sets often involve combining both types of information about the borderline region, incorporating accuracy measures and topological classifications of the set under consideration [129].

Building upon this foundation, the work in [129] explored the types of rough sets by investigating the types associated with union and intersection operations of rough sets of different types. The extension of these concepts to a multigranular instance is introduced in [117], where notations and definitions are provided. The study delves into the topological properties of multigranular rough sets concerning set-theoretic operations such as union, intersection, and complement. The findings offer insights into obtaining answers for these operations in a multigranular instance, facilitating the approximation of classifications and rule induction.

Chapter 4

A data structure for subsumption and disjoint relations in a multigranular model

The work in this chapter describes an efficient data structure, both in space and time, to process subsumption and disjoint relations between granules as part of the multigranular model in [73], and the inference rules presented in [76], which are used to derive relations between granules.

The organization of this chapter is as follows. It begins by presenting the rules of inference to be considered for the developed data structure in Section 4.1. Then, the structure is presented and the developed algorithms are detailed in Section 4.2. Finally, the experimental evaluation performed is presented in Section 4.3.1.

4.1 Inference rules

In this work we adopt the multigranular data model [73], where the notion of granularity enables the classification of the underlying granules that form a granularity. In this model, a *granular space* is composed of a set of granules GnlSet, which includes the top $\top$ (virtual granule that subsumes all granules) and bottom $\perp$ granules (virtual granule that is subsumed by all granules). A *granule structure* is then a pair $(\mathsf{Dom}, \mathrm{GnlToDom})$ composed of a non-empty domain Dom and a mapping function $\mathrm{GnlToDom} : \mathsf{GnlSet} \rightarrow 2^{\mathsf{Dom}}$ from granules to a subset of the domain, such that $\mathrm{GnlToDom}(\top) = \mathsf{Dom}$, $\mathrm{GnlToDom}(\perp) = \emptyset$, and for every $g \neq \perp$, $\mathrm{GnlToDom}(g) \neq \emptyset$. Intuitively, GnlSet can be seen as the set of labels that map through $\mathrm{GnlToDom}$ to a portion of Dom, like the name of a county maps to its portion of the geographic space (i.e., a subset of Dom).

It was shown [76] that these rule sets are correct and complete. Correctness means that the inference rules cannot derive false relations, while completeness means that

Positive Rules

$$\frac{g_1 \sqsubseteq g_2 \quad g_2 \sqsubseteq g_3}{g_1 \sqsubseteq g_3}\,(a) \qquad \frac{g_1 \sqsubseteq g_2 \quad \bigsqcap\{g_2, g_3\} = \bot}{\bigsqcap\{g_1, g_3\} = \bot}\,(b) \qquad \frac{}{g \sqsubseteq g}\,(c)$$

$$\frac{}{\bigsqcap\{\bot, g\} = \bot}\,(d) \qquad \frac{}{\bot \sqsubseteq g}\,(e) \qquad \frac{}{g \sqsubseteq \top}\,(f)$$

Negative Rules

$$\frac{g_1 \sqsubseteq g_2 \quad g_1 \not\sqsubseteq g_3}{g_2 \not\sqsubseteq g_3}\,(a') \qquad \frac{g_1 \not\sqsubseteq g_3 \quad g_2 \sqsubseteq g_3}{g_1 \not\sqsubseteq g_2}\,(b') \qquad \frac{g_2 \sqsubseteq g_3 \quad \bigsqcap\{g_1, g_2\} \neq \bot}{\bigsqcap\{g_1, g_3\} \neq \bot}\,(c')$$

$$\frac{\bigsqcap\{g_1, g_2\} \neq \bot \quad \bigsqcap\{g_3, g_2\} = \bot}{g_1 \not\sqsubseteq g_3}\,(d') \qquad \frac{}{\bigsqcap\{g, g\} \neq \bot}\,(e')$$

Table 4.1: Inferences rules proposed by Hegner and Rodríguez [73].

(successive applications of) the inference rules derive everything that can be inferred from the basic relations. We say that the rules without premises are “axioms”, in our case (c), (d), (e), (f), and (e’) in Table 4.1.

Table 4.2 shows the inference rules covered in this work and their semantics, excluding axioms because they need no implementation strategy. Those cover all the rules presented in Table 4.1, except for rule (d'). This is because the strategies we developed are based on the efficient derivation of the subsumption relation, and rule (d') does not have this relation between granules as a premise. Despite this fact, a particular case of this inference rule is covered by rule (6) in Table 4.2. As a result, we can only ensure completeness and correctness for rules that infer subsumption, disjoint and not-disjoint. For the not-subsumption relation, we offer strategies for its derivation, but we cannot guarantee completeness. The second column in Table 4.2 shows the mapping between the implemented rule and the original rule it corresponds

	Rule	Orig. Rule	Meaning	Relation
1	$\frac{g_1 \sqsubseteq g_2 \quad g_2 \sqsubseteq g_3}{g_1 \sqsubseteq g_3}$	a	If $\mathrm{GnlToDom}(g_1) \subseteq \mathrm{GnlToDom}(g_2)$ and $\mathrm{GnlToDom}(g_2) \subseteq \mathrm{GnlToDom}(g_3)$, then $\mathrm{GnlToDom}(g_1) \subseteq \mathrm{GnlToDom}(g_3)$	subsumption
2	$\frac{g_1 \sqsubseteq g_2 \quad \bigsqcap\{g_2,g_3\}=\bot}{\bigsqcap\{g_1,g_3\}=\bot}$	b	If $\mathrm{GnlToDom}(g_1) \subseteq \mathrm{GnlToDom}(g_2)$ and $\mathrm{GnlToDom}(g_2) \cap \mathrm{GnlToDom}(g_3) = \emptyset$, then $\mathrm{GnlToDom}(g_1) \cap \mathrm{GnlToDom}(g_3) = \emptyset$	disjoint
3	$\frac{g_1 \sqsubseteq g_2}{\bigsqcap\{g_1,g_2\}\neq\bot}$	c'	If $\mathrm{GnlToDom}(g_1) \subseteq \mathrm{GnlToDom}(g_2)$, then $\mathrm{GnlToDom}(g_1) \cap \mathrm{GnlToDom}(g_2) \neq \emptyset$	not disjoint
4	$\frac{g_1 \sqsubseteq g_2 \quad g_1 \sqsubseteq g_3}{\bigsqcap\{g_2,g_3\}\neq\bot}$	c'	If $\mathrm{GnlToDom}(g_1) \subseteq \mathrm{GnlToDom}(g_2)$ and $\mathrm{GnlToDom}(g_1) \subseteq \mathrm{GnlToDom}(g_3)$, then $\mathrm{GnlToDom}(g_2) \cap \mathrm{GnlToDom}(g_3) \neq \emptyset$	
5	$\frac{g_2 \sqsubseteq g_3 \quad \bigsqcap\{g_1,g_2\}\neq\bot}{\bigsqcap\{g_1,g_3\}\neq\bot}$	c'	If $\mathrm{GnlToDom}(g_2) \subseteq \mathrm{GnlToDom}(g_3)$ and $\mathrm{GnlToDom}(g_1) \cap \mathrm{GnlToDom}(g_2) \neq \emptyset$, then $\mathrm{GnlToDom}(g_1) \cap \mathrm{GnlToDom}(g_3) \neq \emptyset$	
6	$\frac{\bigsqcap\{g_1,g_2\}=\bot}{g_1 \not\sqsubseteq g_2}$	a'	If $\mathrm{GnlToDom}(g_1) \cap \mathrm{GnlToDom}(g_2) = \emptyset$, then $\mathrm{GnlToDom}(g_1) \not\subseteq \mathrm{GnlToDom}(g_2)$	not subsumption
7	$\frac{g_1 \not\sqsubseteq g_2 \quad g_3 \sqsubseteq g_2}{g_1 \not\sqsubseteq g_3}$	b'	If $\mathrm{GnlToDom}(g_1) \not\subseteq \mathrm{GnlToDom}(g_2)$ and $\mathrm{GnlToDom}(g_3) \subseteq \mathrm{GnlToDom}(g_2)$, then $\mathrm{GnlToDom}(g_1) \not\subseteq \mathrm{GnlToDom}(g_3)$	
8	$\frac{g_1 \sqsubseteq g_2 \quad g_1 \not\sqsubseteq g_3}{g_2 \not\sqsubseteq g_3}$	a'	If $\mathrm{GnlToDom}(g_1) \subseteq \mathrm{GnlToDom}(g_2)$ and $\mathrm{GnlToDom}(g_1) \not\subseteq \mathrm{GnlToDom}(g_3)$, then $\mathrm{GnlToDom}(g_2) \not\subseteq \mathrm{GnlToDom}(g_3)$	

Table 4.2: Inference rules supported in this work and their equivalence with the original rules proposed by Hegner and Rodríguez [76] (see Table4.1).

to. Note that for the case of rules 3, 4 and 5, all of them are based on rule c'. It was implemented this way because cases 3 and 4 correspond to particular cases of the implementation of the rule. In addition, this decomposition allows us a clearer demonstration of correctness of the proposed strategy. A similar reasoning applies to rules 6 and 8, where both are based on rule a', being rule 6 a particular case of this one. In Section 4.2.2 we explain in more detail how these rules were implemented and their equivalence with the original ones.

As an example of how to use the rules, consider the granules and their explicitly stored relations in Figure 4.1. Based on these relations, subsumption and disjoint relations can be derived using rules (1) and (2), respectively. For example, the relation $L \sqsubseteq C$ is derived from the stored relations $L \sqsubseteq G$ and $G \sqsubseteq C$, and $\sqcap\{N, I\} = \bot$ can be derived from the relations $N \sqsubseteq H$ and $\sqcap\{H, I\} = \bot$. Not-disjoint and not-subsumption relations can be derived using additional rules. More precisely, rules (3), (4) and (5) derive the relation of not-disjoint, and rules (6), (7) and (8) the relation of not-subsumption. For example, rule (3) derives $\sqcap\{C, G\} \neq \bot$ since $G \sqsubseteq C$ is a stored relation. Relation $F \not\sqsubseteq J$ can be derived by applying rule (7) since $K \not\sqsubseteq J$ and $K \sqsubseteq F$ are explicitly stored.

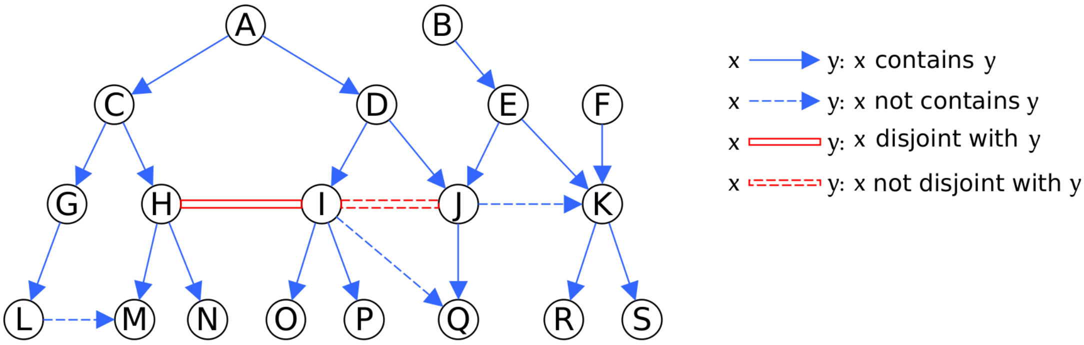


Figure 4.1: Example of a granule graph.

4.2 Proposed data structures

We propose a data structure to facilitate the derivation of relations using a subset of the inference rules outlined in Section 4.1, specifically, implementing all the rules presented in Table 4.2. Our work achieves complete derivations for $g_1 \sqsubseteq g_2$ (subsumption), $\sqcap\{g_1, g_2\} = \bot$ (disjoint), and $\sqcap\{g_1, g_2\} \neq \bot$ (not-disjoint), while a partial derivation is obtained for $g_1 \not\sqsubseteq g_2$ (not-subsumption).

The derivation of relations follows a two-step strategy: *i)* infer subsumption relations using rule 1, and *ii)* infer new relations using at least one of the rules 2–8. This two-step strategy, which stresses the importance of subsumption relations, owes to the fact that all the rules, except rule 6, have a subsumption relation as a premise. Hence, in Section 4.2.1 we propose an efficient implementation for subsumption, and in Section 4.2.2 we describe how to support the other relations.

For all the algorithms that support the rules we use compact data structures to represent the different relation graphs (i.e., the graphs that store explicit knowledge about each relation). To avoid confusion, we denote the graphs representing the relations subsumption, disjoint, not-disjoint, and not-subsumption as G^{sub}, G^{dis}, G^{notdis}, and G^{notsub}, respectively. For technical reasons that will be explained later, the identifiers of the granules in G^{sub} will be considered as the global identifiers for all the rules. When alternative representations need to assign new identifiers to the granules, a mapping from granules in rule 1 and the new identifiers will be explicitly stored and its space reported as part of the data structure.

4.2.1 Inferring subsumption relations

By representing all the granules and their subsumption relations as a directed graph G^{sub}, we can infer a new subsumption relation between granules g_1 and g_2 ($g_1 \sqsubseteq g_2$) by finding a path that connects them. Classical data structures, such as adjacency lists, adjacency matrices and edge lists can be used for this purpose. As the main operation to support the inference rules involves graph traversals, which are implemented more efficient with adjacency lists, our baseline described in Section 4.3 is based on those.

In practical cases, such as the granularity structure of Figure 2.2, it is common to explicitly store only relations between granules of consecutive granularities in the structure. In such cases, the graph $G^{sub}(V, E)$ corresponds to a tree-like graph, that is, a graph in which E is not much larger than V. Because the nodes of the same granularity are disjoint by definition, deviations from a tree can occur only when different overlapping hierarchies exist, with nodes contained in those overlapping areas. The fact that this is uncommon can be exploited in order to define more space-efficient data structures to represent the graphs. We describe next a graph representation that uses $(|E| - |V|) \log_2 |V| + O(|E|)$ bits to represent G^{sub} and, in addition, is optimized to detect the paths that witness subsumption.

Tree-like graphs Fischer and Peters [50] introduced a succinct data structure to represent directed tree-like graphs called GLOUDS (Graph Level Order Unary Degree Sequence). Tree-like means that only a few edges must be removed from the graph to turn it into a tree. GLOUDS transforms the tree-like graph into a tree T by traversing the graph in BFS order. When visiting the edge (u, v), if v was already visited, a copy v' of v, called a *shadow node*, is created and the edge (u, v') is added to the tree. We will refer to v' as a copy of v. The tree is then stored using two compact components:

1. A LOUDS-like representation B of T, in which the main difference with the original LOUDS is that the children of a node that are shadow nodes are marked with a symbol 2 instead of a 1 (hence B is no longer a binary sequence).

2. An array H with the identifiers of the shadow nodes in the same order they were written in B.

This representation supports simple navigational queries, such as computing in-degree and neighbors in $O(1)$ time per returned vertex, and out-degree in $O(1)$ time in total. We could use GLOUDS to store G^{sub}, which would allow us to infer subsumption relations between granules g_1 and g_2 by checking ancestorship: $g_2 \sqsubseteq g_1$ iff g_1 is an ancestor of g_2, or of a copy of some g such that $g_2 \sqsubseteq g$. Note that the ancestorship can be checked in time proportional to the length of the path between g_1 and g_2 (or g_1 and g), while the final condition involves recursively verifying subsumption from nodes g.

In order to improve the performance of checking ancestorship, we modify the representation of the GLOUDS data structure. Specifically, we change the LOUDS representation of T to a balanced parentheses (BP) representation B_o, which allows checking ancestorship in constant time. In BP, nodes are identified by their rank in the DFS traversal of T that produced the representation. Those ranks will be the global identifiers in the implementations of all the rules. We reach constant time because, in the BP representation, a subtree rooted at a vertex v is represented as a contiguous range of parentheses in B_o, which also produces a contiguous range of identifiers. This is a key characteristic of this approach that will also be exploited in Section 4.2.2, because for the rest of the rules it is important to be able to obtain the set of granules S_g contained by a given granule g in an efficient way. To check if vertex u is an ancestor of another vertex v, we only need to determine if the range representing v is contained in the range representing u. The range representing a vertex can be obtained in constant time by using the primitives of the balanced parenthesis representation explained in

Section 2.4.4. Specifically, if the node identifier is g, we find with $p = select_1(B_o, g)$ the position in B_o of its opening parenthesis, and then with $q = find\_close(B_o, p)$ the position of its closing parenthesis.

Our adaptation of GLOUDS, referred to as GBP (Graph Balanced Parenthesis), encompasses the following components, as depicted in Figure 4.2:

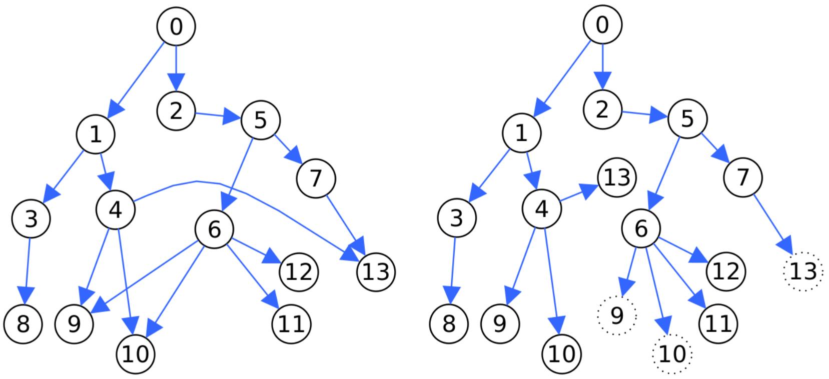
1. A bitvector B_o represents T using BP.

2. A second bitvector B_s marks the opening parentheses of B_o that correspond to shadow nodes.

3. The vector H stores the identifiers of those shadow nodes, now in DFS order (i.e., their order in B_s).

Implementation of inference rule 1 Algorithm 1 checks if granule g_2 subsumes granule g_1. First, if g_2 is an ancestor of g_1 in T, then g_2 subsumes g_1 (line 6). If not, we check if one of the shadow nodes that descend from g_2 is an ancestor of g_1 (lines 8–14)[1]. Otherwise, the rule returns false (line 15). This process is a forward chaining reasoning based on the transitivity property of subsumption. All the operations used in the algorithm take constant time. Hence, the complexity is dominated by the number of recursive calls. This depends linearly on the number of shadow nodes, which is usually small in practice.

[1] By definition a node is an ancestor of itself.



(a) Original graph (b) Tree with shadow nodes

GLOUDS

$$B_{LOUDS} = 1\ 1\ 0\ 1\ 1\ 0\ 1\ 0\ 1\ 0\ 1\ 1\ 1\ 0\ 1\ 1\ 0\ 0\quad 0\ 0\ 0\ 2\ 2\ 0\ 2\ 0\ 0\ 0\ 0\ 0\ 0$$
$$H = 9\ 10\ 13$$

GBP

$$B_o = 1\ 1\ 1\ 1\ 0\ 0\ 1\ 1\ 0\ 1\ 0\ 1\ 0\ 0\ 0\ 1\ 1\ 1\ 1\ 0\ 1\ 0\ 1\ 0\ 1\ 0\ 0\ 1\ 1\ 0\ 0\ 0\ 0\ 0$$
$$B_s = 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 1\ 0\ 0\ 0\ 1$$
$$H = 9\ 10\ 13$$

(c) GLOUDS and GBP representations

Figure 4.2: Example of a tree-like graph represented with GLOUDS and with GBP.

```
Procedure RULE1($g_1$,$g_2$)
    $p_1 \leftarrow select_1(B_o, g_1)$
    $q_1 \leftarrow find\_close(B_o, p_1)$
    $p_2 \leftarrow select_1(B_o, g_2)$
    if $p_1 \leq p_2 \leq q_1$ then
        return $True$
    $l \leftarrow rank_1(B_s, g_1 - 1)$
    $t \leftarrow rank_1(B_o, q_1)$
    $r \leftarrow rank_1(B_s, t)$
    foreach $i \leftarrow l + 1$ to $r$ do
        if rule1($H[i]$,$g_2$) then return $True$
```

Algorithm 1: Algorithm to infer relation 1.

```
Procedure REACH($g$)
    $Q \leftarrow \emptyset$
    $S \leftarrow \emptyset$
    $Q.add(g)$
    while $Q$ is not empty do
        $g' \leftarrow Q.get()$
        $p' \leftarrow parent(g')$
        if $p'$ is valid then
            $S.add(p')$
        $r \leftarrow rank_{g'}(H, |H|)$
        for $i \leftarrow 1$ to $r$ do
            $s \leftarrow select_{g'}(H, i)$
            $v \leftarrow select_1(B_s, s)$
            $p \leftarrow parent(v)$
            if $p$ is valid then
                $Q.add(p)$
                $S.add(p)$
    return S
```

Algorithm 2: Algorithm to compute all granules that

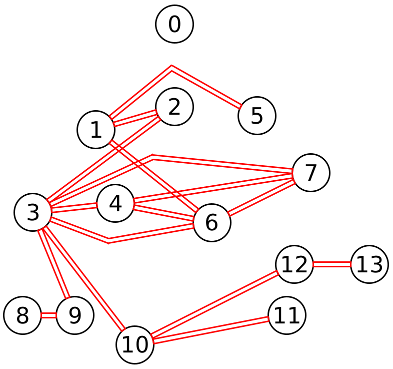
```
Procedure REACHED_BY($g$)
    $Q \leftarrow \emptyset$
    $S \leftarrow \emptyset$
    $Q.add(g)$
    while $Q$ is not empty do
        $g' \leftarrow Q.get()$
        $p_1 \leftarrow select_1(B, g')$
        $q_1 \leftarrow find\_close(B, p_1)$
        $S.add(\langle p_1, q_1 \rangle)$
        $lt \leftarrow rank_1(B, p_1 - 1)$
        $l \leftarrow rank_1(B_s, lt) + 1$
        $rt \leftarrow rank_1(B, q_1)$
        $r \leftarrow rank_1(B_s, rt)$
        for $i \leftarrow l$ to $r$ do
            $Q.add(select_1(B_s, H[i]))$
            $Q.add(select_1(B_s, H[i]))$
```

4.2.2 Inferring other relations

As we mentioned above, to support the other three relations (disjoint G^{dis}, not-disjoint G^{notdis}, and not-subsumption G^{notsub}) we will use a two-step algorithm. In the first step, we use the GBP-based data structure for G^{sub} defined above to obtain a set of induced subsumption relations S (obtained using Algorithm 2). For the second step, we will have as input the set S and a matrix M^r, where $M^r[i][j]$ indicates that the i-th granule is related with the j-th granule in relation r, where $r = \{dis, notdis, notsub\}$. As an example for the disjoint relation, Figure 4.3 shows a graph and its equivalent matrix. We note that only the explicit relations, not including those that can be inferred, are stored in these matrices. The identifiers of the nodes in the matrices are those assigned in the GBP-based representation, where the descendants of a node in the tree defined by GBP have identifiers in a consecutive range. Hence, the ranges will be used for querying the matrices. We will represent those binary matrices in a way that (1) uses space proportional to the number of 1s (i.e., of explicit relations to store), and (2) efficiently supports operation range_check(a,a',b,b'), which returns whether there is a marked cell in the region delimited by the rows a and a', and the columns b and b'. Additionally, we will use a simpler operation check_cell(a,b) = range_check(a,a,b,b).

Each matrix M^r, which presents the graph $G^r(V, E^r)$, is of dimensions $|V| \times |V|$ and has $|E^r|$ 1s. To store it, we traverse it in row major order, storing the ids of the columns of each marked cell in a sequence I. The resulting sequence is stored in a wavelet tree. To group the values of the same row, we use a bit sequence 10^d, where $d \geq 0$ is the number of cells marked in such row. The bit sequences of all rows are concatenated in top-down order and stored in a plain bitmap C with support for rank/select operations. This representation requires $|E^r| \log |V|$ bits to store the column ids plus $|E^r|+|V|+o(|E^r|+|V|)$ bits to store the bitmap C. Notice that for all the matrices M^r we use the same set V of nodes for all the graphs. The range_check(a,a',b,b') operation in the matrix is mapped to range_check$_I$(p_1,p_2,b,b') in the wavelet tree of I, where $p_1 = rank_0(C, select_1(C, a)) + 1$ and $p_2 = rank_0(C, select_1(C, a' + 1)) - 1$.

An alternative implementation of the matrices is to use the GBP representation for the edges present in G^{dis}, G^{notdis} and G^{notsub}. The space for G^r would then be $(|E^r| - |V|) \log |V| + O(|E^r|)$ bits, as described for G^{sub}, plus $|V| \log |V|$ bits to map the identifiers from those of G^{sub} to those of G^{r}, for a total of $|E^{r}| \log |V| + O(|E^{r}|)$ bits. We can then check the presence of an edge (i.e., a matrix cell) in constant time, but matrix ranges must be checked element by element.



	1	2	3	4	5	6	7	8	9	10	11	12	13
1	0	1	0	0	1	1	0	0	0	0	0	0	0
2	0	0	1	0	0	0	0	0	0	0	0	0	0
3	0	1	0	1	0	1	1	0	1	1	0	0	0
4	0	0	1	0	0	1	1	0	0	0	0	0	0
5	1	0	0	0	0	0	0	0	0	0	0	0	0
6	1	0	1	1	0	0	1	0	0	0	0	0	0
7	0	0	1	1	0	1	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	1	0	0	0	0
9	0	0	1	0	0	0	0	1	0	0	0	0	0
10	0	0	1	0	0	0	0	0	0	0	1	1	0
11	0	0	0	0	0	0	0	0	0	1	0	0	0
12	0	0	0	0	0	0	0	0	0	1	0	0	1
13	0	0	0	0	0	0	0	0	0	0	0	1	0

Figure 4.3: Example of the graph and its matrix representation considering the disjoint relation.

Granules g_1 and g_2 are disjoint

We first check if there is an edge connecting g_1 and g_2 in G^{dis}, which is stored as M^{dis}. If so, then g_1 and g_2 are disjoint. If not, we try to infer the relation using Rule 2.

- *Rule 2.* This rule states that g_1 and g_2 are disjoint if there exist granules g'_1 and g'_2, such that $g_1 \sqsubseteq g'_1$ and $g_2 \sqsubseteq g'_2$, and g'_1 and g'_2 are disjoint. Thus, we compute in G^{sub} the sets S_1 and S_2 of granules that contain g_1 and g_2, respectively (see Algorithm 2). Then, for every pair of granules $g'_1 \in S_1$ and $g'_2 \in S_2$, we compute `check_cell(`g'_1`,`g'_2`)` in M^{dis}. If at least one call to `check_cell()` returns true, then g_1 and g_2 are disjoint. Note that this strategy directly captures repeated applications of Rule 2, because it obtains all possible ancestors of both granules and there is no other way to derive a possible relation between them.

Granules g_1 and g_2 are not disjoint

We first check if there is an edge connecting g_1 and g_2 in G^{notdis}. If such an edge exists, we have that g_1 and g_2 are not disjoint. Otherwise, we aim to infer it using rules 3, 4 and 5.

- *Rule 3.* We check if g_1 is contained by g_2, or vice versa, in G^{sub} using Rule 1. If so, then g_1 is not disjoint with g_2.
- *Rule 4.* We compute two sets, denoted S_1 and S_2, comprising all the granules in G^{sub} that are subsumed by g_1 and g_2, respectively. An essential property to consider here is that, given the representation of G^{sub} using GBP, the granules returned are organized into contiguous ranges. The amount of these ranges corresponds to the number of shadow nodes reachable during a traversal of G^{sub} initiated from either g_1 or g_2, plus one. Notice that this traversal is not actually performed. Instead, we directly compute the ranges using Algorithm 3. If the intersection of the sets S_1 and S_2 is nonempty, then g_1 and g_2 are not disjoint.
- *Rule 5.* This rule is implemented in a similar way as Rule 4, but instead of checking the intersection of S_1 and S_2, we apply the operation `range_check()` in M^{notdis} for each range contained in S_1 with each range contained in S_2. If the operation returns true at least once, then g_1 and g_2 are not disjoint. Note that this verification accommodates the potential application of Rule 5 multiple times.

  It is evident that the strategy proposed for rule 5 entails the adoption of a new representation of this rule: $\frac{\bigsqcap_{\mathfrak{G}}\{g_1,g_2\}\neq\bot_{\mathfrak{G}} \quad g_2\sqsubseteq_{\mathfrak{G}} g_3 \quad g_1\sqsubseteq_{\mathfrak{G}} g_4}{\bigsqcap_{\mathfrak{G}}\{g_3,g_4\}\neq\bot_{\mathfrak{G}}}$. This method of representing rule 5 proves advantageous in capturing the concept that both g_1 and g_2 can represent a range of granules. The following demonstration establishes the equivalence of this proposal to the original rule 5.

  Theorem. Given granules g_1, g_2, g_3, g_4

$$\frac{\bigsqcap_{\mathfrak{G}}\{g_1,g_2\}\neq\bot_{\mathfrak{G}} \quad g_2\sqsubseteq_{\mathfrak{G}} g_3}{\bigsqcap_{\mathfrak{G}}\{g_1,g_3\}\neq\bot_{\mathfrak{G}}} \equiv \frac{\bigsqcap_{\mathfrak{G}}\{g_1,g_2\}\neq\bot_{\mathfrak{G}} \quad g_2\sqsubseteq_{\mathfrak{G}} g_3 \quad g_1\sqsubseteq_{\mathfrak{G}} g_4}{\bigsqcap_{\mathfrak{G}}\{g_3,g_4\}\neq\bot_{\mathfrak{G}}}$$

  Proof. We have to show implications in both directions.

1. Let us proof that

$$\frac{\bigsqcap_{\mathfrak{S}}\{g_1, g_2\} \neq \bot_{\mathfrak{S}} \quad g_2 \sqsubseteq_{\mathfrak{S}} g_3}{\bigsqcap_{\mathfrak{S}}\{g_1, g_3\} \neq \bot_{\mathfrak{S}}} \Rightarrow \frac{\bigsqcap_{\mathfrak{S}}\{g_1, g_2\} \neq \bot_{\mathfrak{S}} \quad g_2 \sqsubseteq_{\mathfrak{S}} g_3 \quad g_1 \sqsubseteq_{\mathfrak{S}} g_4}{\bigsqcap_{\mathfrak{S}}\{g_3, g_4\} \neq \bot_{\mathfrak{S}}}$$

   Assume that $\frac{\bigsqcap_{\mathfrak{S}}\{g_1,g_2\} \neq \bot_{\mathfrak{S}} \quad g_2 \sqsubseteq_{\mathfrak{S}} g_3}{\bigsqcap_{\mathfrak{S}}\{g_1,g_3\} \neq \bot_{\mathfrak{S}}}$ is true but $\frac{\bigsqcap_{\mathfrak{S}}\{g_1,g_2\} \neq \bot_{\mathfrak{S}} \quad g_2 \sqsubseteq_{\mathfrak{S}} g_3 \quad g_1 \sqsubseteq_{\mathfrak{S}} g_4}{\bigsqcap_{\mathfrak{S}}\{g_3,g_4\} \neq \bot_{\mathfrak{S}}}$ is false. To have that $\frac{\bigsqcap_{\mathfrak{S}}\{g_1,g_2\} \neq \bot_{\mathfrak{S}} \quad g_2 \sqsubseteq_{\mathfrak{S}} g_3 \quad g_1 \sqsubseteq_{\mathfrak{S}} g_4}{\bigsqcap_{\mathfrak{S}}\{g_3,g_4\} \neq \bot_{\mathfrak{S}}}$ is false, then $\bigsqcap_{\mathfrak{S}}\{g_1, g_2\} \neq \bot_{\mathfrak{S}} \quad g_2 \sqsubseteq_{\mathfrak{S}} g_3 \quad g_1 \sqsubseteq_{\mathfrak{S}} g_4$ should be true but $\bigsqcap_{\mathfrak{S}}\{g_3, g_4\} \neq \bot_{\mathfrak{S}}$ false. Make $g_1 = g_4$, then we have $\bigsqcap_{\mathfrak{S}}\{g_1, g_2\} \neq \bot_{\mathfrak{S}} \quad g_2 \sqsubseteq_{\mathfrak{S}} g_3 \quad g_1 \sqsubseteq_{\mathfrak{S}} g_1$ and we should have that $\bigsqcap_{\mathfrak{S}}\{g_3, g_1\} \neq \bot_{\mathfrak{S}}$ is false. Since $\bigsqcap_{\mathfrak{S}}\{g_3, g_1\} \neq \bot_{\mathfrak{S}}$ is true because we start with $\frac{\bigsqcap_{\mathfrak{S}}\{g_1,g_2\} \neq \bot_{\mathfrak{S}} \quad g_2 \sqsubseteq_{\mathfrak{S}} g_3}{\bigsqcap_{\mathfrak{S}}\{g_1,g_3\} \neq \bot_{\mathfrak{S}}}$ being true, we reach a contradiction.

2. In the other direction

$$\frac{\bigsqcap_{\mathfrak{S}}\{g_1, g_2\} \neq \bot_{\mathfrak{S}} \quad g_2 \sqsubseteq_{\mathfrak{S}} g_3 \quad g_1 \sqsubseteq_{\mathfrak{S}} g_4}{\bigsqcap_{\mathfrak{S}}\{g_3, g_4\} \neq \bot_{\mathfrak{S}}} \Rightarrow \frac{\bigsqcap_{\mathfrak{S}}\{g_1, g_2\} \neq \bot_{\mathfrak{S}} \quad g_2 \sqsubseteq_{\mathfrak{S}} g_3}{\bigsqcap_{\mathfrak{S}}\{g_1, g_3\} \neq \bot_{\mathfrak{S}}}$$

   This trivially proved by making again $g_1 = g_4$ and $g_1 \sqsubseteq_{\mathfrak{S}} g_1$.

With the above, we can conclude that both representations of the rule are equivalent. Note that the proposed strategy presents us with an iterative way of deriving the rule, without incurring in recurrence or the need to combine with other rules to derive $\bigsqcap_{\mathfrak{S}}\{g_1, g_2\} \neq \bot_{\mathfrak{S}}$. This can be viewed simply as follows: If $\bigsqcap_{\mathfrak{S}}\{g_1, g_2\} \neq \bot_{\mathfrak{S}}$ could be derived through another relation, the only alternative is through rules 3 or 4, and therefore:

- If derived through rule 3: this means that $g_1 \sqsubseteq_{\mathfrak{S}} g_2$, therefore, by using rule 3, we would also derive directly $\bigsqcap_{\mathfrak{S}}\{g_3, g_4\} \neq \bot_{\mathfrak{S}}$.
- If derived through rule 4: this means that there exists a granule g_x such that $g_x \sqsubseteq_{\mathfrak{S}} g_1$ and $g_x \sqsubseteq_{\mathfrak{S}} g_2$, therefore, by using rule 4, we would also derive directly $\bigsqcap_{\mathfrak{S}}\{g_3, g_4\} \neq \bot_{\mathfrak{S}}$.

Granule g_1 does not contain granule g_2

Unlike for previous relations, we cannot guarantee completeness for not-subsumption, as explained in Section 4.1. Still, we provide here partial strategies for its derivation.

We first check if there is an edge connecting g_1 and g_2 in G^{notsub}. If so, then we conclude that g_1 does not contain g_2. Otherwise, we aim to infer the relation using rules 6, 7 and 8.

- *Rule 6.* The verification of this rule is straightforward: If there is an edge connecting g_1 and g_2 in G^{dis}, or if by applying the rules for deriving disjoint between g_1 and g_2 we conclude that $\bigsqcap\{g_1, g_2\} = \bot$, then g_1 does not contain g_2, and vice versa.
- *Rules 7 and 8.* For rule 7, we compute the set S_1 of all the granules in G^{sub} that contain g_1 (see Algorithm 2), and check if there is an edge (g_2, g_1') in G^{notsub} for every granule $g_1' \in S_1$, if this is true, we derive the relation $g_2 \not\sqsubseteq g_1$. For rule 8, we compute the set S_2 of all the granules in G^{sub} reachable from g_2 (see Algorithm 3), and check if there is an edge (g_2', g_1) in G^{notsub} for some granule $g_2' \in S_2$, if this is true, we derive the relation $g_2 \not\sqsubseteq g_1$. We implement the verification of these two rules in the same procedure: we first compute S_1 and S_2 as previously described. Then, we check if there is an edge (g_2', g_1') in G^{notsub} for some pair of granules $g_1' \in S_1$ and $g_2' \in S_2$. To implement this more efficiently, we use the operation `range_check()` in G^{notsub} for each granule in S_1 with each range in S_2.

Axioms

Given the characteristics of the axioms, no strategy is necessary for their implementation; their verification is straightforward.

4.3 Experimental evaluation

4.3.1 Implementation details

In this section we provide some practical details about the different implementations. All of them, except the baseline, were implemented using the *Succinct Data Structures Library (SDSL)* [61].

Baseline. We compared our data structures with a baseline representation consisting of four adjacency lists, storing direct and inverse relations, one for each relation of subsumption, disjoint, and their respective negations. To compute all the granules reached by a granule g and all the granules that reach g, standard graph traversals algorithms are performed. An alternative baseline would be to store explicitly all the axioms and the induced relations, but this alternative would potentially store up to n^2 relations, which is not feasible. For instance, for the dataset 7 of Section 4.3.3, the space of this alternative reaches about 16.8GB only for the subsumption relation, while our solution GBP$+Matrix$ uses 902.9MB for the four relations.

GLOUDS-based data structures. We implemented two versions of the GLOUDS-based variant. The first version, called **GLOUDS**, implements the original idea of Fischer and Peters [50], using wavelet trees to store both the LOUDS representation of the tree-like graph and H, the ids of the shadow nodes. The inference rules were implemented using the ideas presented in Sections 4.2.1 and 4.2.2, adapting the Algorithms 2 and 3 to use the LOUDS representation.

For the variant that uses a balanced parenthesis representation instead of LOUDS, called **GBP**, we used the implementation of the compact tree of Navarro and Sadakane [106] to store B_o, a plain bitmap to store B_s, and a wavelet tree to store H. We tested three variants of GBP, varying only in how the IDs of each granule are stored in H: *i)* A first variant storing the original IDs directly in a wavelet tree; *ii)* a second variant, where the identifiers stored in H are previously mapped to a contiguous range, using an extra bitmap H_m for the mapping between the original IDs and the contiguous identifiers; and *iii)* a third variant, using the same strategy as variant ii) but using a compressed bitmap [110] to represent H_m. Additionally, we included an extra plain bitmap B_{open} to mark the non-shadow nodes of GBP. Technically, this bitmap is redundant (rank and select on this bitmap can be simulated by using rank and select over the other components of the structure), but we decided to include it since we obtained slightly better running times, at the cost of negligible extra space.

A granule can have different identifiers in the graphs representing the different relations and in the datasets. In particular, structures to support the mapping from identifiers in a dataset to its corresponding G^{sub}, supporting direct and inverse mapping, and from graph G^{sub} to the others, supporting direct mapping, are needed. In our implementation, the direct and inverse mapping were implemented using compact permutations of the SDSL library.

In addition, a change from the strategy presented in the previous section to verify inference rule 1 was introduced. The strategy outlined above was based on obtaining a set S of ranges of granules contained by a given granule g, and then checking if the queried granule g' was contained in the set S trough the `range_check()` operation. This strategy is not affected by the height of potentially high hierarchies. However, in practice, it is faster to follow a strategy based on obtaining the ancestors of a granule, since the tested datasets have relatively low height. We adapted Algorithm 2 accordingly.

Matrices. The matrices were implemented verbatim to the description in Section 4.2.2. We refer to the implementation of GBP complemented with the matrices as **GBP+Matrix**. Note that this representation uses a GBP for the relation subsumption and one additional matrix for each of the other three relations. The alternative representation that uses a GBP for each relation is referred to as GBP.

Intersection of sets. For the strategy described in Section 4.2.2, specifically for inference rule 4, the final step consists in checking out if there is an intersection between two sets S_1 and S_2, obtained in previous steps. Remember that each element of the sets represents a range of values, stored as a pair. The intersection algorithm works in two steps: *i)* Sort both sets S_1 and S_2, based on the first element of each pair, and then *ii)* traverse both sorted sets from smallest to largest values. When visiting the i-th range of S_1 and j-th range of S_2, we check if they intersect or not. After that, the range with the lowest closing value is replaced by the next range in the respective set. Step *ii)* is repeated until the final range in one of the sets is reached. The temporal complexity is $O(|S_1| \log |S_1| + |S_2| \log |S_2|)$ for the first step, and $O(|S_1| + |S_2|)$ for the second, where $|S|$ represents the cardinality of the set S. No extra space is required.

4.3.2 Experimental setup

We run all the experiments on a computer with an Intel Xeon Gold (5320T) processor, clocked at 2.3 GHz; 252 GB DDR4 RAM memory, with speed 3,200 MT/s; 40 physical cores each one with L1i and L1d caches (32 KB and 48 KB, respectively), and L2 cache (1,280 KB); and a shared L3 cache of size 30 MB. The operating system is Linux 5.10.0-13-amd64 (Debian 10.2.1-6), in 64-bit mode. All the evaluated algorithms were implemented in C++ and compiled with GCC version 10.2.1 and -O3 optimization flag.

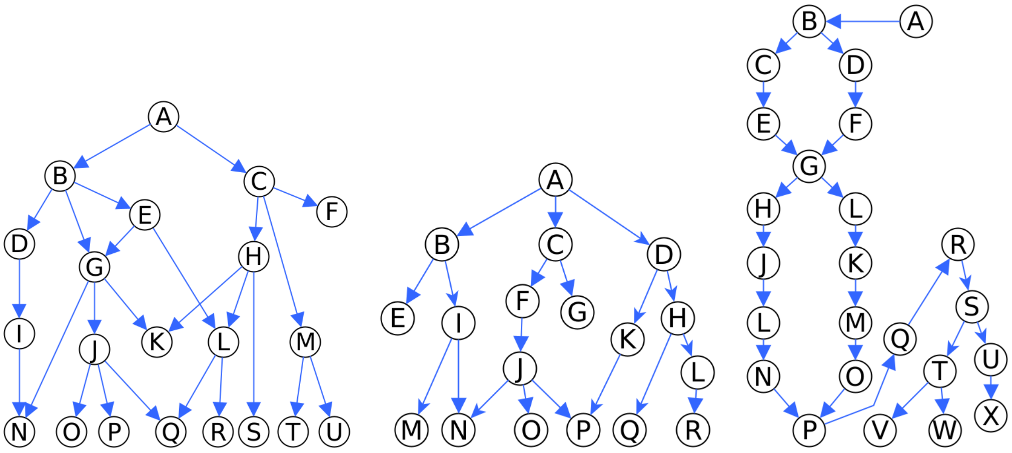
4.3.3 Datasets

The datasets used for the evaluation of the implemented algorithms can be classified into two categories: i) Synthetic datasets, generated in order to evaluate the behavior of the implementations in scenarios with different characteristics, and ii) Real datasets, used to evaluate the algorithms in a real world instance.

Synthetic datasets

We have developed a multigranular instance generator that allows us to manipulate various characteristics of the generated datasets. This enables us to assess the performance of different approaches across a spectrum of scenarios, ranging from favorable to unfavorable configurations. The generator takes input parameters including a description of the granularity graph, the number of granules in each granularity, and the count of subsumption relations between granularities. Notably, the input description includes subsumption relations between granularities as well as counts of non-subsumption, disjoint, and non-disjoint relations. Utilizing this information, the generator constructs a multigranular instance, represented in the form of four graphs (one for each relation). It is important to note that the generator does not guarantee minimal information in the generated instances, as it may produce relations that could also be inferred using inference rules.

Taking the granularity structures from Figure 4.4 as input, we utilized the instance generator to compute nine synthetic instances – three for each granularity structure.



(a) Multigranular structure I (b) Multigranular structure II (c) Multigranular structure III

Figure 4.4: Granularity structures used in the generation of the synthetic datasets.

We varied the number of granules per granularity and the number of not subsumption, disjoint, and not disjoint relations. Table 4.3 summarizes the characteristics of each generated instance, where *Granules* refers to the number of granules in the instance, *Subsumption Edges* to the number of relations in the subsumption graph obtained from the dataset, and *Other Relations* to the amount of not subsumption, disjoint and not disjoint relations (the same amount for the 3 relations).

The *Multigranular Structure I* (Figure 4.4(a)) represents an instance where the subsumption graphs obtained from it will have a large number of cycles, which is not favorable for the implementations based on compact data structures. The subsumption graphs obtained from *Multigranular Structure II* (Figure 4.4(b)), instead, will have a low number of cycles, which is favorable for the implementations based on compact data structures. Finally, the *Multigranular Structure III* (Figure 4.4(c)) represents an instance where the generated subsumption trees will have a larger height than in previous cases, which is unfavorable for the baseline.

Structure	Dataset	Granules	Subsumption Edges	Other Relations
Multigranular Structure I	1	7,742,101	10,962,100	300,000
	4	77,421,001	109,621,000	300,000
	7	77,421,001	109,621,000	30,000,000
Multigranular Structure II	2	5,723,001	7,723,000	30,000
	5	57,230,001	77,230,000	30,000
	8	57,230,001	77,230,000	3,000,000
Multigranular Structure III	3	2,968,721	2,989,840	30,000
	6	5,937,441	5,979,680	30,000
	9	19,739,201	19,823,680	3,000,000

Table 4.3: Characteristics of the generated synthetic datasets.

Real-world dataset

A real-world dataset was obtained to evaluate all the structures in practical scenarios. This dataset is based on the TIGER dataset,[2] provided by the U.S. Census Bureau, which corresponds to geographic and cartographic data of the administrative divisions in the United States. For this study, we computed relations between each possible pair of granules based on their geometry. From these obtained relations, a subset was selected for analysis. The dataset comprises a total of $11{,}555{,}150$ granules, $11{,}705{,}035$ subsumption relations and $691{,}350$ other relations. Figure 4.5 illustrates the granularity structure of this dataset, while Table 4.4 presents the distribution of granules for each granularity.

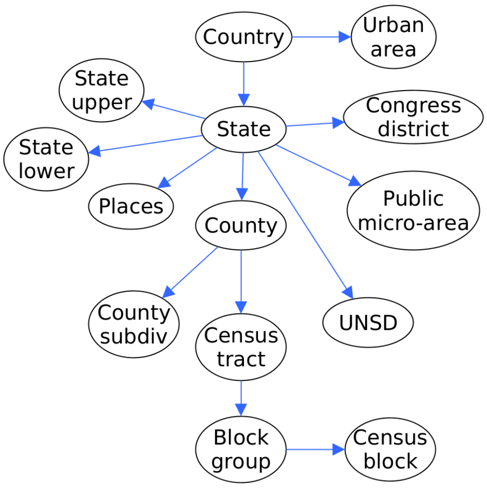


Figure 4.5: Granularity structure of the real-world dataset.

[2] TIGER dataset, version 2019. https://www2.census.gov/geo/tiger/TIGER2019/

Granularity	Granules	Granularity	Granules
Census blocks	11,166,336	County subdivisions	36,693
Block groups	220,740	Places	29,853
Census tracts	74,133	Unified school districts	10,887
Counties	3,233	Public Use microdata area	2,380
States	56	State legislative districts lower	4,833
Congressional districts	444	State legislative district upper	1,961
Urban areas	3,601	Country	1

Table 4.4: Number of granules for each granularity in the real-world dataset.

4.3.4 Results and discussion

Execution time

Based on the implementations described in Section 4.2, and using the datasets described above, we conducted two experimental evaluations to measure the performance of the implementations on various multigranular instances: *a)* A first experimental evaluation, in which the assessment is made at the rule level. This analysis offers a more detailed perspective and provides comprehensive information about the performance of different strategies across all the datasets. *b)* A second experimental evaluation, where the assessment is performed at the relation level, as certain relations require checking multiple rules. This experiment, presented only on the real dataset, provides us with an overview of the behavior of the implementations in deriving each relation.

To carry out both experimental evaluations, a total of 10,000 queries were performed. For each executed query, two granules were randomly selected, with the only restriction that both granules could not belong to the same granularity. Each executed operation was repeated 10 times, leaving the first repetition for cache warming and reporting the average of the last nine. Regarding the GBP-based implementations, only the variant with the best average performance for each dataset is shown. Algorithms 4, 5, 6 and 7 show the strategy used in the second experimentation to derive the four possible relations. Table 4.5 shows the best performing variant for GBP and GBP$+Matrix$.

	Datasets									
	1	2	3	4	5	6	7	8	9	Real-world
GBP	no mapping	no mapping	no mapping	no mapping	compressed bitmap	no mapping	no mapping	no mapping	compressed bitmap	plain bitmap
GBP+$Matrix$	no mapping	plain bitmap	plain bitmap	compressed bitmap	plain bitmap	plain bitmap	no mapping	compressed bitmap	compressed bitmap	plain bitmap

Table 4.5: Summary of the best-performing variants for GBP and GBP+$Matrix$. *No mapping* represents the variant that does not map the identifiers of H to a contiguous range, *plain bitmap* represents the variant that maps to a contiguous range using a plain bitmap, while *compressed bitmap* uses a compressed bitmap for the mapping.

```
Procedure SUBREL($g_1$,$g_2$)
    return RULE1($g_1$,$g_2$)
```

Algorithm 4: Infer subsumption relation between g_1 and g_2.

```
Procedure DISREL($g_1$,$g_2$)
    return RULE2($g_1$,$g_2$))
```

Algorithm 5: Infer Disjoint relation between g_1 and g_2.

```
Procedure NOTDISREL($g$)
    if RULE1($g_1$,$g_2$) then
        return TRUE
    if RULE4($g_1$,$g_2$) then
        return TRUE
    if RULE5($g_1$,$g_2$) then
        return TRUE
    return FALSE
```

Algorithm 6: Infer Not-disjoint relation between g_1 and g_2.

```
Procedure NOTSUBREL($g$)
    if RULE2($g_1$,$g_2$) then
        return TRUE
    if RULE7_8($g_1$,$g_2$) then
        return TRUE
    return FALSE
```

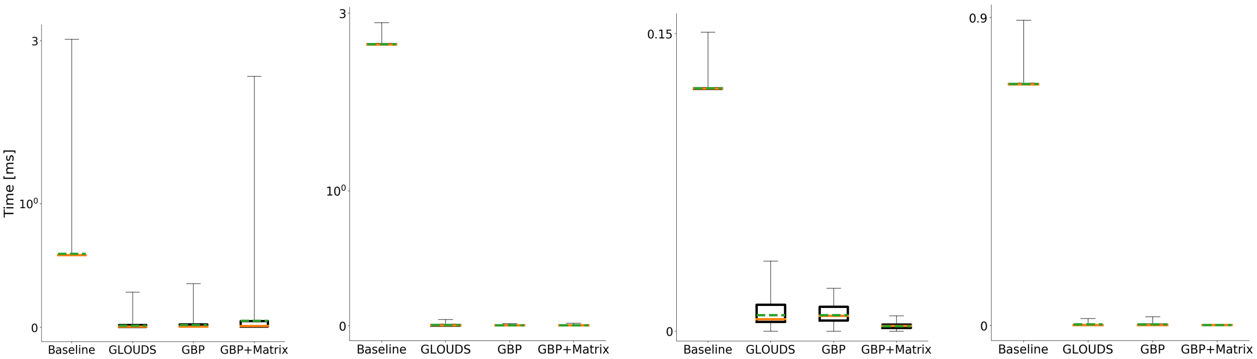
Algorithm 7: Infer Not-subsumption relation between g_1 and g_2.

Table 4.6 shows the average query time of executing the evaluated inference rules. We omit rules 3 and 6 because, as it was explained in Section 4.2, their implementation is straightforward using either rule 1 or the information explicitly stored. For each multi-granular structure, we show only the results of the largest dataset as, in general, there are no significant differences with the other datasets in the behavior of the algorithms. When necessary, we explicitly mention other datasets to highlight some differences. In general, the GBP+$Matrix$ structure is the one that provides the best query times. Although GLOUDS is highly competitive for rule 1, which forms the foundation for the others, its use in the remaining rules is clearly surpassed by other alternatives. Next, a more detailed analysis with graphs is presented for each of the evaluated rules. In these box and whisker plots, the solid orange lines and the dashed green lines represent the median and the average of the 10,000 queries executed for each operation on each dataset, respectively.

	Implementation	**Rule 1**	**Rule 2**	**Rule 4**	**Rule 5**	**Rule 7–8**
1	Baseline	0.080	0.093	109.447	110.237	64.327
	GLOUDS	0.004	0.418	498.780	677.693	269.359
	GBP	0.010	0.843	19.347	132.677	14.818
	GBP+$Matrix$	0.051	0.010	1.544	3.019	0.868
2	Baseline	0.033	0.060	88.517	89.201	66.301
	GLOUDS	0.001	0.129	382.118	565.223	268.735
	GBP	0.003	0.180	19.980	111.255	16.996
	GBP+$Matrix$	0.002	0.005	2.581	2.756	1.871
3	Baseline	0.015	0.026	28.745	29.143	15.910
	GLOUDS	0.003	0.065	108.523	146.766	55.975
	GBP	0.006	0.114	1.473	28.426	1.340
	GBP+$Matrix$	0.002	0.008	0.301	0.322	0.191
4	Baseline	1.536	1.005	924.500	935.604	558.991
	GLOUDS	0.006	1.211	4,440.620	7,203.446	2,402.746
	GBP	0.013	1.612	183.484	1,425.052	129.811
	GBP+$Matrix$	0.005	0.010	28.732	29.891	16.320
5	Baseline	1.810	0.658	881.351	876.708	455.990
	GLOUDS	0.001	0.317	3,834.380	5,803.235	1,932.735
	GBP	0.002	0.371	287.755	1,546.167	157.213
	GBP+$Matrix$	0.002	0.005	27.553	29.035	13.832
6	Baseline	0.035	0.055	61.340	61.538	36.181
	GLOUDS	0.003	0.088	238.832	365.719	134.817
	GBP	0.006	0.131	3.367	84.028	3.020
	GBP+$Matrix$	0.002	0.008	0.622	0.881	0.381
7	Baseline	0.593	1.827	946,395.831	985,031.864	504,671.148
	GLOUDS	0.012	34.241	4,492,048.734	8,734,626.647	2,469,116.228
	GBP	0.017	125.174	185,807.011	2,650,916.996	919,867.408
	GBP+$Matrix$	0.051	0.026	13,505.470	19,021.349	7,620.357
8	Baseline	2.232	3.393	699,895.857	706,570.169	410,272.566
	GLOUDS	0.004	9.994	3,414,470.570	5,312,987.940	2,818,667.450
	GBP	0.002	23.453	164,676.120	1,262,852.952	178,314.425
	GBP+$Matrix$	0.002	0.007	22,801.088	400,027.292	12,800.239
9	Baseline	0.122	0.278	177,104.284	178,217.012	104,075.019
	GLOUDS	0.008	2.228	1.157,154.270	1,773,641.919	568,361.203
	GBP	0.008	4.265	9,555.768	173,426.880	59,694.149
	GBP+$Matrix$	0.002	0.009	1,471.840	3,072.575	827.238
Real-world	Baseline	0.706	0.784	168,944.355	169,866.044	78,652.403
	GLOUDS	0.002	1.529	789,767.128	1,237,133.055	319,982.573
	GBP	0.003	5.083	12,133.495	349,513.661	25,323.073
	GBP+$Matrix$	0.001	0.003	670.821	1,623.978	330.210

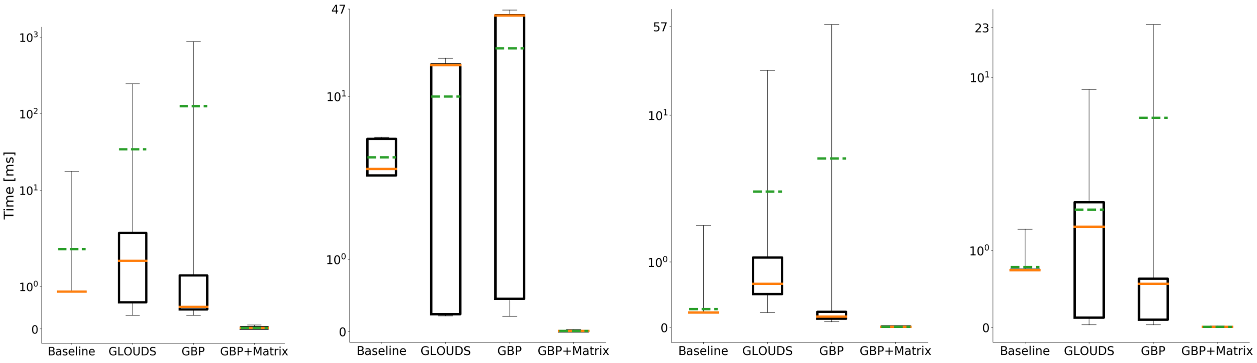
Table 4.6: Average query times, in ms.

Figure 4.6 presents the results for inference rule 1. For this rule there is no clear winner, as can be seen in Table 4.6, the differences in the performance of GLOUDS versus GBP-based structures are not conclusive. This may be partially explained by



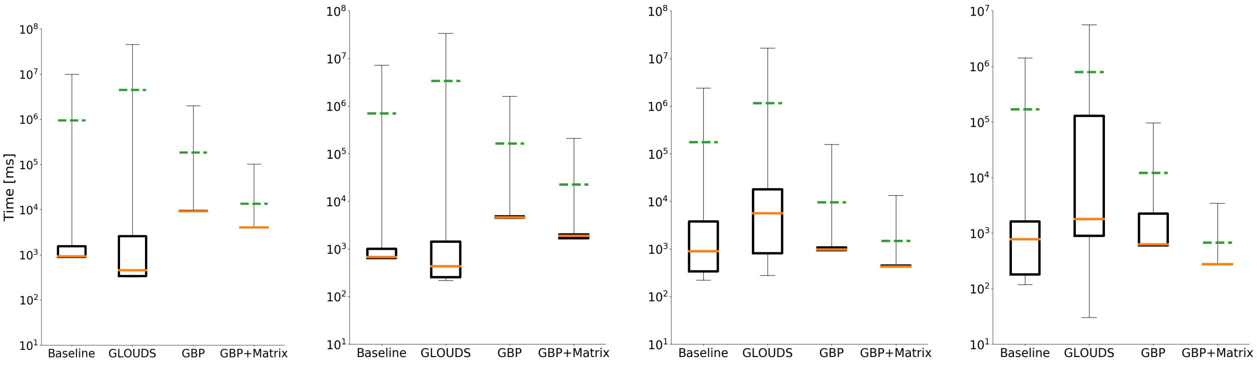
(a) Dataset 7 (b) Dataset 8 (c) Dataset 9 (d) Real-world dataset

Figure 4.6: Running time in milliseconds for the inference rule 1.



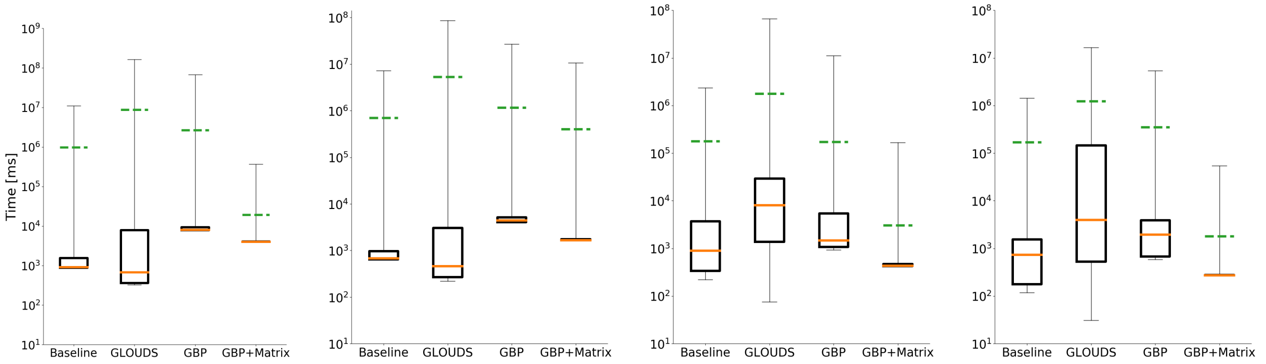
(a) Dataset 7 (b) Dataset 8 (c) Dataset 9 (d) Real-world dataset

Figure 4.7: Running time in milliseconds for the inference rule 2.



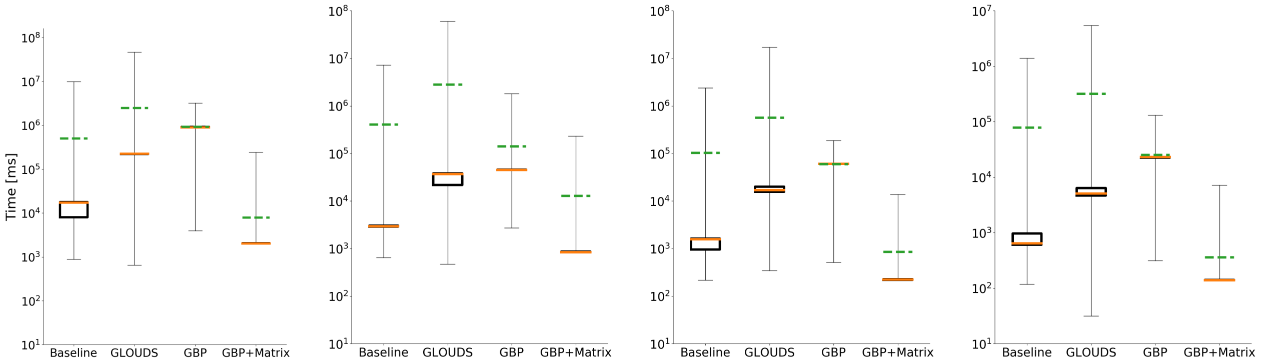
(a) Dataset 7 (b) Dataset 8 (c) Dataset 9 (d) Real-world dataset

Figure 4.8: Running time in milliseconds for the inference rule 4.



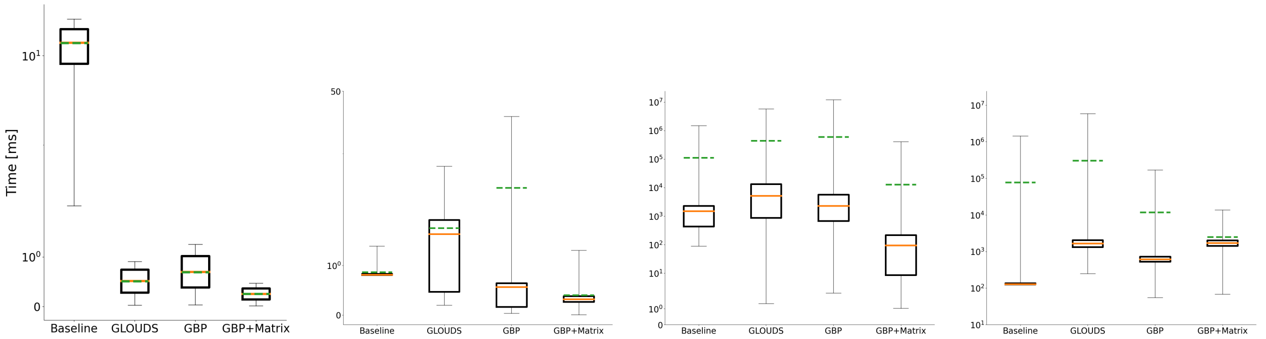
(a) Dataset 7 (b) Dataset 8 (c) Dataset 9 (d) Real-world dataset

Figure 4.9: Running time in milliseconds for the inference rule 5.



(a) Dataset 7 (b) Dataset 8 (c) Dataset 9 (d) Real-world dataset

Figure 4.10: Running time in milliseconds for the inference rule 7-8.



(a) Subsumption relation (b) Disjoint relation (c) Not-disjoint relation (d) Not-subsumption relation

Figure 4.11: Running time in milliseconds for the experimental evaluation at the relation level on the Real-world dataset.

the fact that, in the datasets, the height of the granularities is negligible compared to the amount of granules per instance. Thus, GLOUDS, whose performance is directly dependent on the height, exhibits a similar behavior to GBP-based alternatives, which do not depend on the height of the instances.

In addition, as we will see in the results for the following rules, GBP and GBP$+Matrix$ yield better results when the computation of ranges of consecutive identifiers is combined with the other rules. Notice also that the poor performance of the baseline is mainly due to the large number of cache misses produced at the time of obtaining the ancestors of a granule, by following the reverse references from children to parents (which are explicitly stored). This can be concluded by analyzing the smaller datasets, in which, although the baseline presents worse performance than the other implementations, the differences are smaller. Finally, no significant changes are perceived in the behavior of the structures when confronted with multigranular structures with different complexities.

Figure 4.7 presents the results for inference rule 2. In general, GBP$+Matrix$ presents better running times than the rest of the implementations. This is because, in comparison with GLOUDS and GBP, the use of matrices accelerates the verification of the existence of a relation other than subsumption. Remember that in GBP$+Matrix$, the identifiers used in the matrices are the same used in the GBP representation of G^{sub}, therefore, no mapping is needed, while for GLOUDS and GBP a mapping is mandatory. When considering the mean of the reported query times, GLOUDS outperforms GBP for all the evaluated instances. This is primarily due to the main difference between both implementations for this inference rule, which lies in the way of obtaining the ancestors of a node. This operation is faster in GLOUDS. It is worth noting that, despite this, GBP has a better median than GLOUDS in most of the datasets, except for dataset 8, which suggests that, in general, it has a higher number of favorable cases.

Figure 4.8 displays the results for inference rule 4. In this case, GLOUDS exhibits the highest variance, with the lowest and highest query times observed. This variance contributes to GLOUDS having the worst average performance among the evaluated approaches. On the other hand, GBP$+Matrix$ achieves the best average query time, despite being slower for favorable queries due to the overhead involved to support range operations over the matrix in the cases of nodes without descendants or with only one descendant. However, this approach ensures a low query time for unfavorable cases. GBP performs better than GLOUDS in terms of average query time but has a worse median for datasets 7 and 8. This highlights an advantage of the GBP variants, which may be slightly slower for favorable cases due to the extra mappings between bitmaps (B_o and B_s) but perform better overall, even in the worst-case scenarios. All evaluated implementations exhibit consistent trends regardless of the dataset being evaluated.

Figure 4.9 displays the results for inference rule 5. The behavior observed for this rule is similar to that of inference rule 4. It leads, however, to an increase in response time for GLOUDS and GBP in their unfavorable cases, with GBP becoming slower than the baseline in its worst cases. GBP$+Matrix$ also increments its query time in worst-case, but remains the implementation with the best average for all datasets and better median for the dataset 9 and the real-word dataset.

Figure 4.10 illustrates the outcomes for inference rules 7-8. In this context, akin to the scenario concerning rule 4, it is observed that GLOUDS exhibits the highest variance, with the lowest and highest query times observed. GBP$+Matrix$, on the other hand, achieves the best average and median query times. Notably, GBP$+Matrix$ performs significantly better on average than the other implementations in datasets based on Multigranular Structure III, consistently delivering the best query times for all evaluated queries (including the most favorable and the most unfavorable). This is because Multigranular Structure III is taller than the others and produces a moderate number of shadow nodes, making it a favorable scenario for GBP$+Matrix$.

Finally, Figure 4.11 displays the result for the second experimental evaluation carried out at the relation level on the Real-world dataset. For subsumption and disjoint relations, the same behavior as described above for their individual rules is presented. For the not-disjoint relation it can be seen how GBP$+Matrix$ shows the best performance both on average, median, and for the worst cases. In the case of the not-subsumption relation, it can be seen that GBP$+Matrix$ presents better performance for both the worst case and on average than the rest of the implementations, even

	Datasets									
Implementations	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**Real-world**
Baseline	709.4	519.8	261.6	7,073.0	5,196.5	523.0	7,308.2	5,220.1	1,761.4	1,031.3
GLOUDS	132.0	92.2	40.5	1,504.7	986.2	83.6	1,541.7	1,048.1	304.1	176.6
GBP	152.6	108.2	49.9	1,690.3	1,148.1	102.5	1,712.9	1,194.0	363.7	215.9
GBP + Matrix	57.6	40.5	15.0	630.0	431.9	30.3	902.9	456.8	126.4	72.8

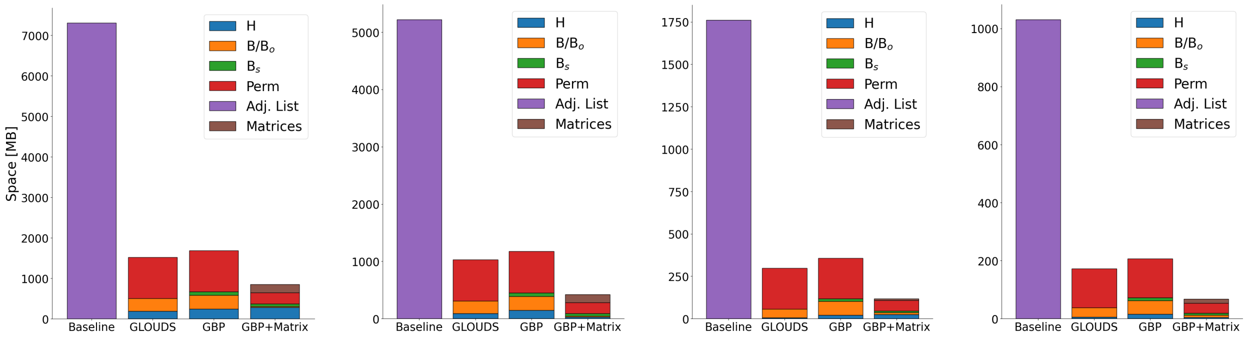
Table 4.7: Total space used for each data structure, in MB.

so, it presents a worse median than the rest, which shows that, in general, for favorable queries it tends to be slower than other implementations. It is also worth noting that, although GBP has a worse average performance than GBP$+Matrix$, it exhibits a better median and ranks as the second-best performing implementation concerning unfavorable cases.

Space usage

Table 4.7 shows the total space used by each implementation for each dataset. In all of them, the baseline occupies considerable more space than the others, while GBP$+Matrix$ is the one that uses the least space. GBP uses, in general, more space than GLOUDS (close to 20% extra in the worst cases), because it needs additional bitmaps to distinguish the shadow nodes.

Figure 4.12 provides a detailed view of the space occupied by the main components for the largest datasets. As it can be observed, the primary distinction between GBP$+Matrix$ and the GBP and GLOUDS implementations is that it occupies less space to represent permutations. This is because it stores a single permutation to map identifiers from the input dataset to G^{sub}, compared to storing four permutations to map from the input to G^{sub} and from G^{sub} to the other relations.



(a) Dataset 7 (b) Dataset 8 (c) Dataset 9 (d) Real-world dataset

Figure 4.12: Detailed analysis of the space used by each approach.

Chapter 5

Compact Representations of Spatial Hierarchical Structures

The work in this chapter proposes new data structures to implement a topological model in spatial database systems that represents and accesses data organized as a hierarchical structure of regions defined by the inclusion relation (spatial version of the subsumption relationship presented in the previous chapter) . Our approach differs from classical indexing structures that optimize spatial queries [69, 123, 70], which are specially designed to answer spatial range queries that return objects that are inside of a specific region given as the input of the query.

The work in [55] showed how to implement the topological spatial data model using a planar-graph compact structure, which is based on Turán's representation [130] (structure a.k.a. PEMB). We extended their work by adding extensions to answer the topological relations of disjointness, inclusion, and adjacency at different levels of detail, which are useful in the context of spatial partitions such as administrative subdivision of the space. Note that this approach restricts our results to the two-dimensional space. Table 5.1 shows notation to be taken into account for the remainder of the chapter ,while Table 5.2 shows as a summary the execution time and space used for each approach.

The organization of this chapter is as follows: It begins by presenting the the 3 approaches developed in Section 5.1 to support the hierarchical information and the algorithm develop for each approach, after that, a general strategy (valid in any of the 3 described approaches) is presented in Section 5.4 for the support of disconnected regions. Finally, the experimental evaluation performed is presented in Section 5.5.

R	Partition into top-level regions
h	Number of levels in the hierarchy, 1 to h
L_i	Set of regions of level i, L_1 corresponds to R
n_i	Number of regions at level i, $n_i = \|L_i\|$
m_i	Number of pairs of neighboring regions of level i, $m_i = \Theta(n_i)$
d_r	Number of neighbors of region r at its same level
n	Total regions of all levels, $n = \sum_{i=1}^{h} n_i$
m	Total neighboring pairs at all levels, $m = \sum_{i=1}^{h} m_i = \Theta(n)$
S_i	Representation of the planar embedding of level i

Table 5.1: Notations.

Operation		Complexities	
	Approach 1	Approach 2	Approach 3
contains(r_1, r_2)	$O(1)$	$O(\log n \log h)$	$O(1)$
touches(r_1, r_2)	$O(min(d_{r_1}, d_{r_2}))$	$O(min(d_{r_1}, d_{r_2}) \log n \log h)$	$O(min(d_{r_1}, d_{r_2}))$
contained(L_j, r_1)	$O(1)$	$O(\log n \log h)$	$O(1)$
Space in bits	$O(n \lg h) + o(hn_h)$	$O(n \lg h)$	$O(n \log n)$
Space w/exponential growing	$O(n)$	$O(n)$	$O(n \log n)$

Table 5.2: Multi-granular operations considered in this article, where r_1 and r_2 are regions at levels $i \leq j$, respectively, and d_{r_1} and d_{r_2} are their respective number of neighboring regions. The complexity of the contained query is per returned element. We present three solutions (one per column) with different complexities.

5.1 Approaches developed

5.1.1 Approach 1: Mapping via bitmaps

Instead of building PEMB independently for each planar embedding of the collection, we proposed an approach to synchronize the construction of the compact representation of the embeddings, which allows to implicitly encode the mapping among consecutive granularity levels in less space. The synchronization is made by the spanning trees of the different aggregation levels. In Section 5.1.3 we present an algorithm to compute a spanning tree at level h, from which we can induce valid spanning trees for the other aggregation levels as follows (we prove later than this construction is correct).

Definition 1. *Given a spanning tree T of the planar embedding of L_h, we induce spanning trees for the planar embeddings of $L_1, L_2, \ldots, L_{h-1}$ using the following rules:*

- *Let (u, v) be an edge of T, and let u^i and v^i be the regions containing regions u and v at level L_i, respectively. Then, the edge (u^i, v^i) belongs to the spanning tree of level L_i.*
- *Multiple edges and self-loops are deleted.*

Figure 2.8 shows an example of spanning trees following Definition 1. From the spanning tree of Figure 2.8(a) we can induce the spanning trees of Figures 2.8(b) and 2.8(c).

5.1.2 Structure

Each granularity level L_i is stored in two components (see Figure 5.1): *1)* The planar graph embedding is stored using PEMB, generating a sequence S_i of parentheses and brackets, where parentheses represent the spanning tree of the planar embedding and brackets represent the edges not in the spanning tree. In Section 5.1.3, we show how to construct the sequence S_i. The space consumption for the h granularity levels is $4\sum_{i=1}^{h} m_i + o(\sum_{i=1}^{h} m_i) = 4m + o(m)$ bits; *2)* The mapping among granularity levels is stored as a bitmap B_i of length $2n_h$ with support for $rank$ and $select$ operations. Following Definition 1, the edges of the spanning tree of L_i, induced from the spanning tree of L_h, are marked in the bitmap B_i. Precisely, let e be an edge of the spanning tree of L_i, then we set $B_i[p] = 1$ and $B_i[q] = 1$, where p and q are the positions in S_h of the opening and closing parentheses of the edge in L_h that induced e. Notice that for level h we do not need to store a bitmap B_h. Since B_i has only $2n_i$ 1s, its compressed representation requires $2n_i \log \frac{n_h}{n_i} + O(n_i) + o(n_h)$ bits.

Overall, the space of this representation is $4m + o(m) + 2\sum_i n_i \log \frac{n_h}{n_i} + O(n) + o(hn_h) = 2\sum_i n_i \log \frac{n_h}{n_i} + O(n) + o(hn_h)$ bits. Since $\lg \frac{n_h}{n_i} \leq \lg \frac{n}{n_i}$ and, by Jensen's inequality, $\sum_i n_i \lg \frac{n}{n_i} \leq n \lg h$, the total space is in $O(n \lg h) + o(hn_h)$ bits. Further, the space is $O(n)$ under the exponential growing assumption: since $n_i \leq n_h/c^{h-i}$, $\sum_i n_i \lg \frac{n_h}{n_i} \leq \sum_i n_h(h-i)/c^{h-i} \cdot \log c = O(n_h)$. In turn, the $o(hn_h)$ term can be $O(hn_h/\lg n_h) = O(n_h)$ [113].

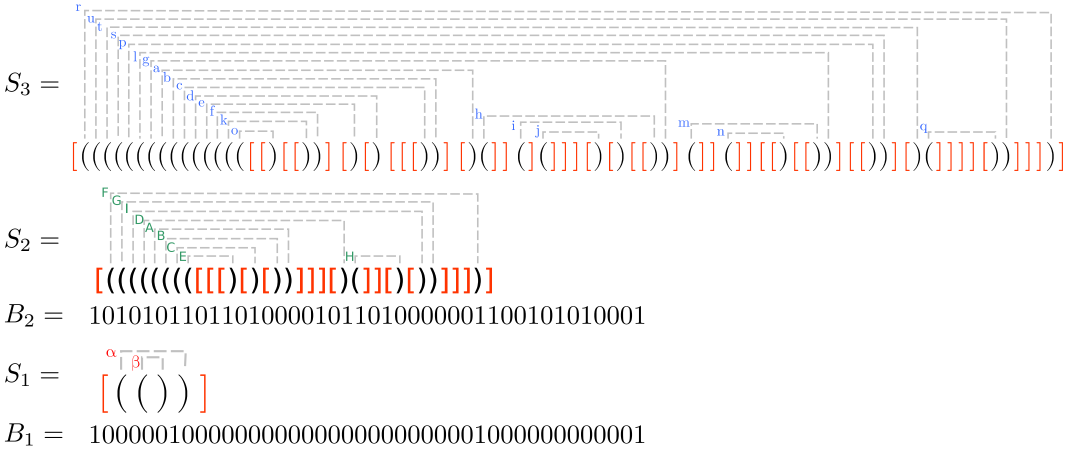


Figure 5.1: Compact representation of the geographic division of Figure 2.8.

5.1.3 Construction

The compact representation is built by performing a DFS traversal on the planar graph of the highest granularity level, h. During the traversal, when we mark a vertex as visited, we also mark as visited the $h-1$ regions that contain it in coarser granularity levels. Thus, an edge (u, v) is traversed when the target vertex v has not been visited before and one of the following conditions holds: *a)* u and v are contained by the same region at level $h-1$; *b)* at least one of the regions containing vertex v has not been visited before. Algorithm 8 shows the strategy previously described.

In the traversal, each edge of the planar embedding of L_h is processed twice[1] and only the edges of the spanning tree are traversed. Let us focus on the generation of S_i and B_i, where, by default, all values of B_i are 0s. Assume that we are processing the j-th edge $e = (r_1, r_2)$ of L_h, where regions $r'_1 \neq r'_2$ contain regions r_1 and r_2 at level i, respectively. The following conditions are checked (function CHECKEDGE(e) in algorithm 8):

1. If it is the first time that e is processed and the edge (r'_1, r'_2) belongs to the spanning tree of level i, then $B_i[j] = 1$ and a symbol '(' is appended to S_i.

2. If it is the first time that e is processed and the edge (r'_1, r'_2) does not belong to the spanning tree of level i, then a symbol '[' is appended to S_i.

[1] We assume that the input graph is undirected, and hence each edge is processed twice.

Input : Planar graph embedding G at the highest level L_h, with n vertices and m edges, the hierarchy relations among regions, and the starting vertex s

Output: Parentheses sequence S_h and the bitmaps $B_{i\in[1..h-1]}$.

```
Algorithm traversalDFS(G, s)
    $m' \leftarrow$ number of regions across all granularity levels
    Stack $O$, Stack $E$
    $visited_{vertex}[1..n]$ // default value no_visited
    $visited_{edge}[1..m]$ // default value no_traversed
    $visited_{reg}[1..m']$ // default value no_visited
    $entry\_edge_{h-1}$ // Map for each level different to h, default value $0$
    foreach incident edge $e$ of $s$, in cw order do $O.push(e)$
    while Stack $O$ is not empty do
        $e \leftarrow O.top()$; $O.pop()$
        if checkEdge($e$) = 1 then
            Append '(' to $S_h$, $E.push(e)$
            $visited_{edge}[e] \leftarrow$ traversed$_{()}$
            $visited_{vertex}[e.src] \leftarrow$ visited
            foreach incident edge $e'$ of $e.tgt$ in cw order do
                $O.push(e')$
            foreach granularity level $l$ do
                // $l$ From 0 to $h-1$
                $p \leftarrow$ getRegion(l, e.src)
                $q \leftarrow$ getRegion(l, e.tgt)
                if $visited_{reg}[q] =$ no_visited AND $p \neq q$ then
                    Append '(' to $S_l$
                    firstVisit($e, l$)
                    $B_l[i] \leftarrow 1$
            $i \leftarrow i + 1$// Initially, $i \leftarrow 0$
        else if checkEdge($e$) = 2 then
            Append '[' to $S_h$
            $visited_{edge}[e] \leftarrow$ traversed$_{[]}$
            foreach granularity level $l$ do
                $p \leftarrow$ getRegion(l, e.src)
                $q \leftarrow$ getRegion(l, e.tgt)
                if $entry\_edge_l[p, q] = 0$ AND $p \neq q$ then
                    Append '[' to $S_l$
                    firstVisit($e, l$)
        else if checkEdge($e$) = 3 then
            Append ')' to $S_h$
            foreach granularity level $l$ do
                $p' \leftarrow$ getRegion(l+1, e.src)
                $q' \leftarrow$ getRegion(l+1, e.tgt)
                if $entry\_edge_l[q, p] = 1$ then
                    Append ')' to $S_l$
                    $B_l[i] \leftarrow 1$
            $i \leftarrow i + 1$
        else if checkEdge($e$) = 4 then
            Append ']' to $S_h$
            foreach granularity level $l$ do
                $p' \leftarrow$ getRegion(l+1, e.src)
                $q' \leftarrow$ getRegion(l+1, e.tgt)
                if $entry\_edge_l[q, p] = 1$ then
                    Append ']' to $S_l$
    return $S$
```

```
Procedure checkEdge(e)
    if $visited_{edge}[e] = \text{traversed}_{()}$ then return 3
    else if $visited_{edge}[e] = \text{traversed}_{[]}$ then return 4
    else if $visited_{vertex}[e.src] = \text{visited}$ then return 2
    else if $visited_{vertex}[e.src] = \text{no\_visited}$ AND checkLevels($e$) = False then return 2
    else return 1
Procedure checkLevels(e)
    foreach Granularity level $l$ from $h$ to 1 do
        $r_1 \leftarrow$ getRegion(l, e.src)
        $r_2 \leftarrow$ getRegion(l, e.tgt)
        if $r_1 = r_2$ then return True
        else if $visited_{reg}[r_2] = \text{visited}$ then return False
    return True
Procedure firstVisit(e, l)
    $q \leftarrow$ getRegion(l, e.tgt)
    $p' \leftarrow$ getRegion(l+1, e.src)
    $q' \leftarrow$ getRegion(l+1, e.tgt)
    $visited_{reg}[q] \leftarrow \text{visited}$
    $entry\_edge_l[p', q'] \leftarrow 1$
```

Algorithm 8: Construction algorithm. For an edge e, its endpoints are referred as $e.src$ and $e.tgt$. The function getRegion(l, v) returns the vertex containing v at level l.

3. If it is the second time that e is processed and the edge (r'_1, r'_2) belongs to the spanning tree of level i, then $B_i[j] = 1$ and a symbol ')' is appended to S_i.

4. Finally, if it is the second time that e is processed and the edge (r'_1, r'_2) does not belong to the spanning tree of level i, then a symbol ']' is appended to S_i.

Observe that $B_i[j] = 1$ indicates that we are entering to or exiting from a region at granularity level i, depending on whether it is the first or second time that such edge has been processed. In particular, exiting from a region means that all its regions contained at finer granularity levels have been processed.

By using an auxiliary bitmap to mark the processed edges at each $i < h$, all sequences S_i and bitmaps B_i can be computed at the same time during the traversal, obtaining a final time complexity of $O(n_h + hm_h) \subseteq O(hn)$, dominated by the at most h comparisons per edge. We now prove the correctness of the construction algorithm.

Lemma 1. *The algorithm described above computes a valid spanning tree.*

Proof. We show by contradiction that there are no cycles and that all regions of L_h belong to the produced subgraph. On the one hand, a cycle means that during the construction an edge (u, v), where both u and v and/or their containing regions are marked as visited, was added to the set. However, that contradicts the rule that only edges leading to a non-visited target regions are added at any level i. Therefore, the produced subgraph is acyclic.

Note that, when we leave a region by an edge to another, we do not reenter the region, to avoid cycles. We resume the traversal of the region only once we return from the outgoing edge. This makes the traversal of a region reach all of its nodes, exactly as if the outgoing edges were ignored. This is the key to show that all the regions are reached by the tree, which makes it a spanning tree. Assume the opposite, and let r be a region that is not reached and that touches a reached region r'. Such a region must exist because there is a path of regions between every non-reached region and the region where we start the traversal, which is reached by definition. When the algorithm traversed r', it reached all of its nodes, in particular the one with an edge towards r, which exists because r and r' are neighbors. At that point, r was not reached and the traversal should have entered it; a contradiction.

□

5.1.4 Operations

In order to support the operations of Table 5.2, we provide the following primitive operations to navigate the compact representation.

Basic primitives. Hereinafter, we consider that each region, represented by a vertex in the planar embedding of L_i, is identified by its pre-order rank in the traversal of the spanning tree of level i.

- go_up_L$_\mathsf{h}$$(x, i)$: This operation allows us to map the x-th region of granularity level i to a region at level h. To do that, we must find the position of the x-th region in S_i with $z = select_((S_i, x)$, to then map such position into the bitmap B_i, with $y = select_1(B_i, rank_{()}(S_i, z))$. Finally, the position of the output region corresponds to the position of the y-th open parenthesis in S_h, which can be obtained with $select_{()}(S_h, y)$. The time complexity is $O(1)$, since it depends on constant time operations $rank$ and $select$.

- $\mathsf{go\_down\_L_h}(x, d)$: This operation is complementary to $\mathsf{go\_up\_L_h}$, mapping the x-th region of L_h into a region at level $h - d$. We start as for $\mathsf{go\_up\_L_h}$, finding the position of the x-th region in S_h with $z = select_{(}(S_h, x)$, to then map it to the bitmap B_{h-d} with $p = rank_{()}(S_h, z)$. The final answer corresponds to the position in S_{h-d} of the nearest ancestor y of x in the spanning tree of level L_{h-d} that is marked in B_{h-d}. To do that, we compute $q = select_{()}(S_{h-d}, rank_1(B_{h-d}, p))$. If $S_{h-d}[q] =$ '(', then q is the answer, otherwise it is $q' = \texttt{enclose}(S_{h-d}, \mathsf{find\_open}(S_{h-d}, q))$. This operation takes constant time.

- $\mathsf{region\_id}(S_i, x)$: This operation returns the id of the region represented by the open parenthesis $S_i[x] =$ '('. It can be solved in constant time with $rank_{(}(S_i, x)$.

- $\mathsf{go\_level}(x, i, j)$: This operation is a generalization of operations $\mathsf{go\_up\_L_h}$ and $\mathsf{go\_down\_L_h}$, mapping the x-th region of L_i into a region at level j. It can be solved in $O(1)$ time by mapping the x-th region of L_i into a region of L_h, to then map such region of L_h into a region of L_j, as $\mathsf{go\_down\_L_h}(\mathsf{go\_up\_L_h}(x, i), h - j)$. Notice that when $j < i$, we are going down in the hierarchy, whereas when $j > i$ we are going up.

Main operations. We now focus on the operations of Table 5.2. Let $r_1 \in L_i$ and $r_2 \in L_j$ be two regions such that $i \leq j$:

- $\mathsf{contains}(r_1, r_2)$: *Does region r_1 contain region r_2?*. First, if r_1 and r_2 belong to the same level (i.e., $i = j$), we just return whether $r_1 = r_2$. Otherwise, we compute the region $r_2' \in L_i$ that contains r_2, $r_2' = \mathsf{region\_id}(S_i, \mathsf{go\_level}(r_2, j, i))$, and return whether $r_1 = r_2'$. The time complexity of this query is $O(1)$.

- $\mathsf{touches}(r_1, r_2)$: *Does region r_1 share a boundary with region r_2?* We distinguish two cases: *1)* If r_2 is not contained in r_1 ($\mathsf{contains}(r_1, r_2)$ = false), we must find a neighbor of r_2 that *is* contained in region r_1; and *2)* if r_2 is contained in r_1 ($\mathsf{contains}(r_1, r_2)$ = true), then we must find a neighbor of r_2 that *is not* contained in r_1. For each neighbor w of r_2, we compute its containing region at level i as $z = \mathsf{region\_id}(S_i, \mathsf{go\_level}(w, j, i))$. For the first case, if we cannot find a neighbor of r_2 such that $r_1 = z$, then we return false; otherwise we return true. Similarly, for the second case, if we cannot find a neighbor of r_2 such that $r_1 \neq z$, then we return false; otherwise we return true. The time complexity is $O(d_{r_2})$, depending directly on the number of neighbors of r_2.

- contained(L_j, r_1): *List all regions at level j contained in region r_1.* To support this operation, we report all regions in the range $S_j[a..b]$ that are contained by the region r_1, where $a = \text{go\_level}(r_1, i, j)$ and $b = \text{find\_close}(S_j, a)$. To report the regions, we traverse the range left-to-right reporting every region region_id(S_j, a'), where initially $a' = a$ and then it is redefined as the position of the next open parenthesis, $a' = \text{leftmost}_((S_j, a')$, until $a' > b$. It is possible, however, that each such position a' is marked as the beginning of a new region, in which case we have to skip the subtree with $a' = \text{leftmost}_((S_j, \text{find\_close}(S_j, a'))$. An opening parenthesis at position p is marked if $B_i[c] = 1$, where $c = select_1(B_j, rank_{()}(S_j, p))$. Thus, this operation can be answered in $O(n_j)$ time. Despite its high worst-case complexity, we implement this solution with competitive practical results, see Section 4.3. We can, however, improve the theoretical result so as to spend $O(1)$ time per output region, by limiting the number of skipped subtrees between consecutive output regions. This can be done by adding dummy vertices that work as the root of consecutive subtrees that must be skipped. By marking the dummy vertices in the bitmap B, we can skip them during the left-to-right traversal. Thus, skipping a dummy vertex is equivalent to skip its descendant subtrees. The dummy vertices skipped then amortize to the number of the vertices that belong to the output, because there is at least one useful node between every two dummy nodes. The extra space is $O(n_j)$ bits in the level j, since we can add up to one dummy vertex per edge of the planar embedding. Additionally, to distinguish the dummy vertices, we can mark them in a bitmap of $O(n_j)$ bits.

Theorem 2 summarizes the results of this first approach:

Theorem 2. *A geographic connected region organized as a multi-granular hierarchy with n regions in total and h granularity levels can be represented in $O(n \lg h) + o(hn_h)$ bits, where n_h is the number of regions at granularity level L_h. The same representation* *supports operations* contains(r_1, r_2) *in constant time,* touches(r_1, r_2) *in* $O(min(d_{r_1}, d_{r_2}))$ *time, and* contained(L_j, r_1) *in constant time per returned element, where* r_1 *and* r_2 *represent a region at granularity levels* L_i *and* L_j*, respectively, and* d_{r_1} *and* d_{r_2} *are their respective number of neighboring regions. Under the exponential growing assumption, the space consumption is* $O(n)$ *bits.*

Subsections 5.2 and 5.3 introduce two new approaches that provide trade-offs for the work done in this approach. In particular, the approach of Section 5.2 improves the space consumption, both in practice (as we show in Section 4.3) and in theory by a sublinear term, at the cost of increasing the running time by a factor of $O(\log n \log h)$, meanwhile the approach of Section 5.3 reduces in practice the running time of the operations at the cost of increasing space consumption.

5.2 Approach 2: Mapping sequence

The data structures of the first approach have two sources of redundancy:

- If $B_i[k] = 1$, then $B_j[k] = 1$ for all $j \geq i$, that is, the mapping bitmaps are contained in the next ones.
- Following Definition 1, from S_h we can derive the sequences S_i, $i < h$. In particular, the k-th parenthesis of sequence S_i corresponds to $S_h[select_{()}(S_h, select_1(B_i, k))]$.

Our second approach removes both sources of redundancy, in exchange for higher time complexities.

Instead of storing the h compact planar embeddings, we only store the sequence S_h. By construction, the sequence S_i is implicitly contained in the sequence S_{i+1}. Therefore, S_h implicitly represents all the sequences $S_1, \ldots, S_{h-1}$. We need a way to check if a vertex in S_h, represented by a pair ( ), is present or not in an (implicit) arbitrary sequence $S_{i \in \{1 \ldots h-1\}}$. Note that a parenthesis that is present in a sequence S_i is also present in all sequences $S_{j \in \{i+1 \ldots h\}}$, therefore, the first source of redundancy implies that the mapping bitmaps B_i can be replaced by a single sequence that tells the lowest level j a position of S_h belongs to. In turn, the bitmap B_i defines the sequence S_i, so in

$S_3^{()}$= ((((((((((((((( )))))))) ((( )))) (( )))))) ( )))

$S_3^{[]}$= [[[[[ ] [[[[[ ] [ ]]]]]] [[[[ ]]]]] [[[[ ] [[ ] [ ]]]] [ ]]]]

$B_{()}$= 1 3 2 3 2 3 1 2 3 2 2 3 2 3 3 3 3 2 3 2 2 3 2 3 3 3 3 3 3 1 2 3 3 2 3 2 3 2 3 3 3 1

$B_{[]}$= 1 2 2 2 3 3 2 3 2 3 3 3 3 3 3 2 3 2 2 2 3 3 3 3 3 3 2 2 2 3 3 3 3 2 3 3 3 3 2 3 3 3 3 2 2 1

Figure 5.2: Sequences $S_3^{()}$, $S_3^{[]}$, $B_{()}$ and $B_{[]}$ for the sequence S_3 of Figure 5.1.

principle storing the sequence of lowest levels plus S_h should be sufficient. We need, however, to navigate the sequence of parentheses and brackets of S_i. Although we do not represent S_i explicitly, we will represent the needed RMMTs.

5.2.1 Structure

Let $S_h^{()}$ be the sequence composed of only the parentheses of S_h. We define the sequence $B_{()}[1..2n_h]$, which stores the lowest level each parenthesis of $S_h^{()}$ belongs to. Formally, $B_{()}[k] = j$ iff the k-th parenthesis of $S_h^{()}$ is present at the sequence S_j but not at S_{j-1}. Thus, the i-th parenthesis of $S_h^{()}$ is present at S_j iff $B_{()}[i] \leq j$, and the position in $S_h^{()}$ of the i-th parenthesis of S_j can be computed as $select_{\leq j}(B_{()}, i)$. Analogously, we define the sequences $S_h^{[]}$ and $B_{[]}[1..2(m_h - n_h + 2)]$, associated with the dual graph of the planar embedding of level h. Figure 5.2 shows the sequences $S_3^{()}$, $S_3^{[]}$, $B_{()}$, and $B_{[]}$ corresponding to the sequence S_3 of Figure 5.1.

The representation is then composed by:

- The planar graph embedding of L_h, represented with PEMB. It uses $4m_h + o(m_h)$ bits.

- The RMMTs of the balanced parenthesis sequences $S_1^{()}, S_2^{()}, \ldots, S_h^{()}$, and $S_1^{[]}, S_2^{[]}, \ldots, S_h^{[]}$. Summing up the $2h$ RMMTs, the space usage is $O(m)$ bits. See Figure 5.3 for an example of the RMMTs of Figure 5.1.

- The sequences $B_{()}$ and $B_{[]}$ with support for $rank_{\leq}$ and $select_{\leq}$ operations, using $2(m_h + 2)\lceil \lg h \rceil + o(m_h \lg h) = 2m_h \lg h + o(n \lg h)$ bits.

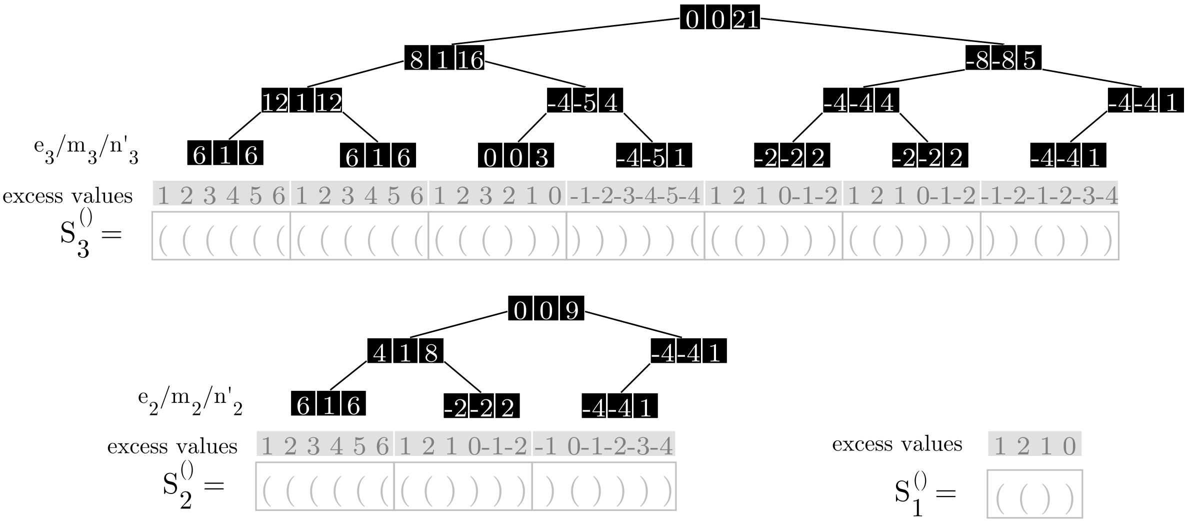


Figure 5.3: RMMTs of balanced parenthesis sequences $S_1^{()}$, $S_2^{()}$ and $S_3^{()}$. A node of the RMMT covering the block $S_i^{()}[i..j]$ stores: the last excess value of the block (e_i), the minimum excess value of block (m_i), and the number of opening parentheses in the block(n_i'). The values and parentheses in gray are not explicitly stored.

Since $m_h = O(m)$ and $m = \Theta(n)$, the total space is $O(n \lg h)$ bits. In fact, the sequences $B_{()}$ and $B_{[]}$ can be represented to within their zero-order entropy. The sequence $B_{()}$ has $n_i - n_{i-1} - \ldots - n_1 \leq n_i$ occurrences of the symbol i, and therefore its entropy $H_0(B_{()})$ is at most $\sum_{i \in [1..h]} \frac{n_i}{n} \log \frac{n}{n_i}$. Similarly, the entropy $H_0(B_{[]})$ of $B_{[]}$ is at most $\sum_{i \in [1..h]} \frac{m_i}{m} \log \frac{m}{m_i}$. Under the exponential growing assumption, $n_i \leq n_h / c^{h-i}$ and, since $m_i = \Theta(n_i)$, there exists a constant d such that $m_i \leq m_h / d^{h-i}$. As shown in the end of Section 5.1.2, both entropies are $O(1)$. Both $B_{()}$ and $B_{[]}$ can then be stored in space $O(m(H_0(B_{()}) + H_0(B_{[]}))) + o(n_h \log h) + O(h \log n) = O(n) + o(n \log h)$ bits [120], and the space $o(n \log h)$ can be $O(n \log h / \log n) = O(n)$ [113]. Thus, the total space is $O(n)$ bits.

We can traverse the implicit sequences $S_1^{()}, \ldots, S_{h-1}^{()}$ by performing $select_{\leq j}$ and $rank_{\leq j}$ operations over the sequence $B_{()}$. The i-th parenthesis of S_j is obtained in $O(\log n_h \log h)$ time as $select_{\leq j}(B_{()}, i)$, and the number of parentheses of S_j in the range $S_h^{()}[1..i]$ is obtained in $O(\log h)$ time as $rank_{\leq j}(B_{()}, i)$. Similarly, operations $\mathsf{find\_open}(S_j^{()}, i)$, $\mathsf{find\_close}(S_j^{()}, i)$ and $\mathtt{enclose}(S_j^{()}, i)$ are supported in $O(\log n_h \log h + \log n_h) = O(\log n_h \log h)$ time, where the term $\log n_h \log h$ corresponds to the traversal of a block in the RMMT of

$S_j^{()}$, performing an operation $\mathsf{leftmost}_{\leq j}(S_h^{()}, i)$ for each parenthesis of the block, and the term $\log n_h$ comes from the up/down traversal of the RMMT.

5.2.2 Operations

As before, we introduce the implementation of basic primitives upon which the main operations are constructed. The time complexities of all the operations become $O(\log n_h \log h)$.

- $\mathsf{go\_up\_L_h}(x, i)$: To support this operation we use the RMMT of the sequence $S_i^{()}$ to find the x-th open parenthesis, $z = select_((S_i^{()}, x)$. Then, we map the position of the parenthesis to the sequence $B_{()}$ by computing $y = select_{\leq i}(B_{()}, z)$. Finally, the position of the sought region in S_h is $select_{()}(S_h, y)$.

- $\mathsf{go\_down\_L_h}(x, d)$: The answer is the position q in S_h so that $(q, \mathsf{find\_close}(S_h^{()}, q))$ most tightly encloses the x-th parenthesis of S_h and $B_{()}[q] \leq h - d$. We find the position of the opening parenthesis representing the x-th region of L_h with $p = select_((S_h^{()}, x)$. Then, $q = rank_{\leq h-d}(B_{()}, p)$ is the number of parentheses in $S_h^{()}[1..p]$ that belong to $S_{h-d}^{()}$. If the q-th parenthesis is opening (i.e., $S_h^{()}[select_{\leq h-d}(B_{()}, q)] =$ '('), the answer is q. Otherwise, the answer is its closest ancestor, at position $q = \mathtt{enclose}(S_{h-d}^{()}, \mathsf{find\_open}(S_{h-d}^{()}, q))$.

- $\mathsf{region\_id}(S_i, x)$: We map the position of x to $S_h^{()}$ with $p = select_{\leq i}(B_{()}, x)$, and then count the number of opening parentheses up to position p that belong to S_i using its RMMT, $rank_((S_i^{()}, p)$.

The operation go_level is implemented just as in Section 5.1.1, $\mathsf{go\_level}(x, i, j) = \mathsf{go\_down\_L_h}(\mathsf{go\_up\_L_h}(x, i), h - j)$, with time complexity $O(\log n_h \log h)$.

The implementation of the main operations contains, touches, and contained follows the same steps of their counterparts in Section 5.1.1, reaching complexities $O(\log n_h \log h)$, $O(d_{r_2} \log n_h \log h)$, and $O(\log n_h \log h)$ per element, respectively. In particular, for the operation touches, the traversal of the neighbors of a region is performed using the RMMT primitives `fwd_search` and `bwd_search`.

The following theorem summarizes the results of this approach:

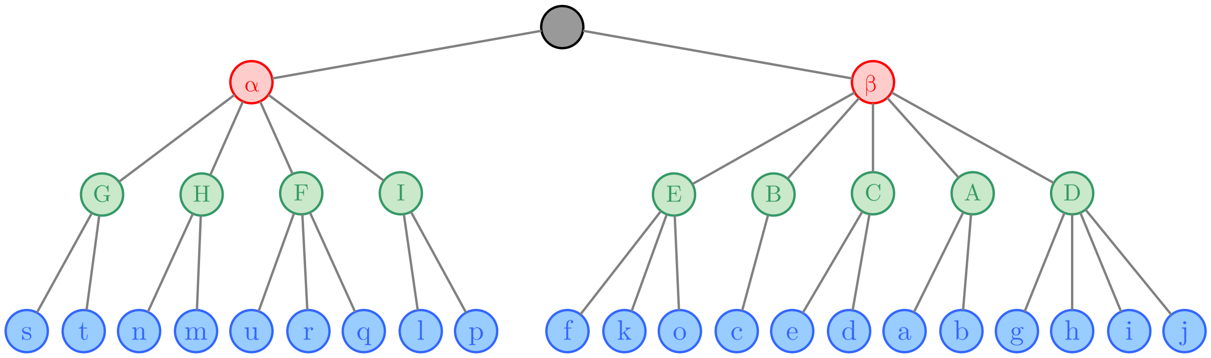
Theorem 3. *A geographic connected region organized as a multi-granular hierarchy with n regions in total and h granularity levels can be represented in $O(n \lg h)$ bits. The* *same representation supports operations* $\textsf{contains}(r_1, r_2)$ *in* $O(\lg n \lg h)$ *time,* $\textsf{touches}(r_1, r_2)$ *in* $O(min(d_{r_1}, d_{r_2}) \log n \log h)$ *time, and* $\textsf{contained}(L_j, r_1)$ *in* $O(\lg n \lg h)$ *time per returned element, where* r_1 *and* r_2 *represent a region at granularity levels* L_i *and* L_j*, respectively, and* d_{r_1} *and* d_{r_2} *are their respective number of neighboring regions. Under the exponential growing assumption, the space consumption is* $O(n)$ *bits.*

Note that our asymptotic space complexity does not change if we represent the sequences S_i in explicit form. In this case we can operate them directly and, although the complexities do not change, we expect them to be much faster in practice (the structure, in turn, becomes larger in practice). This approach is much more direct, as we only have to change the operations on bitmaps B_i by operations on the sequences $B_{()}$ and $B_{[]}$.

5.3 Approach 3: Hierarchy tree

Our third approach aims to offer better running times in practice, though using more space, compared to the representation of Section 5.1.1. As in our first representation, the planar embeddings representing the topology of each aggregation level are stored independently using PEMB. However, the topological hierarchy is stored in a different manner. Instead of using the h bitmaps B_i, we represent a tree H associated with the relation contains, called the *hierarchy tree*. For every pair of regions r_1 and r_2 such that $r_1 \in L_i$ and $r_2 \in L_{i+1}$, and $\textsf{contains}(r_1, r_2)$ is true, region r_2 is added to the tree H as a child of region r_1. Additionally, a dummy root is added connecting the nodes that represent regions of L_1. Thus, all nodes at depth i in H represent regions at aggregation level i. Figure 5.4(a) shows the tree H for the topological hierarchy of Figure 2.8.

Once the tree H is computed, we store its topology as a balanced parenthesis sequence T^H. During the traversal, we additionally store in a permutation M the preorder rank in T^H of the opening parenthesis representing each node of H. The values stored in M are laid level by level (1 to h), in the order the PEMB representation of each L_i represents the corresponding nodes. Notice that such an indexing allows us to map the regions between the topological hierarchy and the planar embeddings, and



(a) Hierarchy tree H representing the topological hierarchy of Figure 2.8. The root of the tree is a dummy node.

$T^H =$ (((()())(()())(()()())(()()))((()()())(())(()())(()())(()()()())))

αGs t Hn m Fu r q Il p βEf k o Bc Ce d Aa b Dg h i j

	1	2	3
O=	2	4	13

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33
M=	1	2	16	9	3	13	29	26	21	23	17	6	11	10	5	4	15	14	30	27	28	22	25	24	18	19	20	31	32	33	8	7	12
	root	α	β	F	G	I	D	A	B	C	E	H	r	u	t	s	p	l	g	a	b	c	d	e	f	k	o	h	i	j	m	n	q

(b) Balanced parenthesis sequence T^H, sequence M and offsets O of the tree H of Figure 5.4(a).

Figure 5.4: Components to store the topological hierarchy in the third representation.

vice-versa. Further, the position of the leftmost value of each level i in M is stored in an array of offsets $O[1..h]$. For instance, if the region $r \in L_i$ is the j-th visited region of that level during DFS traversal of L_i, and is also the k-th region visited in the traversal of T^H, then, $M[O[i]+j-1]=k$. Figure 5.4(b) shows an example of T^H, M and O.

This representation uses $4m+o(m)$ bits for the h planar embeddings. The balanced parenthesis sequence T^H, supporting navigational operations, uses $2n+o(n)$ bits. The permutation M uses $(1+\epsilon)n\log n+O(n)$ bits, with a representation that also computes $M^{-1}(j)$, that is, where in M is the value j, in time $O(1/\epsilon)$ [101]. The total space is then $O(n\log n)$ bits.

5.3.1 Operations

We now describe how the operations are computed with this representation.

- contains(r_1, r_2): We map both regions to T^H and check if $r_1 \in L_i$ is an ancestor of

$r_2 \in L_j$. Let $r_1' = M[O[i] + r_1 - 1]$ and $r_2' = M[O[j] + r_2 - 1]$ be the mappings in T^H of r_1 and r_2. Then the answer is true iff $r_1' \leq r_2' \leq \mathsf{find\_close}(r_1')$. The operation contains then takes $O(1)$ time.

- $\mathsf{touches}(r_1, r_2)$: This is built on top of operation contains as in Section 5.1.1. The time complexity is then $O(d_{r_2})$.

- $\mathsf{contained}(L_j, r_1)$: The regions to report correspond to all the descendants of $r_1 \in L_i$ at depth $j > i$ in T^H. The node representing r_1 in T^H is $r_1' = M[O[i] + r_1 - 1]$. Let $p = select_((T^H, r_1')$ and $q = \mathsf{find\_close}(T^H, p)$ be the positions of the opening and closing parentheses of r_1' in T^H. We then report the regions of all the opening parentheses at depth $j - i$ from p up to q. To do that, we go down in $O(1)$ time from r_1' up to its leftmost descendant u at depth $j - i$, reporting the position $p' = \mathtt{fwd\_search}(T^H, p, j - i)$. Then, we keep reporting the region to the right of p' with its same depth, up to $p' > q$, by computing $p' = \mathsf{level\_next}(p') = \mathtt{fwd\_search}(T^H, \mathsf{find\_close}(T^H, p'), 1)$. For every position p' to report, we return its region id with $M^{-1}(rank_((T^H, p')) - O[j]$. The time complexity of operation contained is then $O(1/\epsilon)$ (i.e., any desired constant) per element reported.

The following theorem summarizes the results of this section:

Theorem 4. *A geographic connected region organized as a multi-granular hierarchy with n regions in total and h granularity levels can be represented in $O(n \lg n)$ bits. The same representation supports operations* $\mathsf{contains}(r_1, r_2)$ *in $O(1)$ time,* $\mathsf{touches}(r_1, r_2)$ *in $O(min(d_{r_1}, d_{r_2})$ time, and* $\mathsf{contained}(L_j, r_1)$ *in $O(1)$ time per returned element, where r_1 and r_2 represent a region at granularity levels L_i and L_j, respectively, and d_{r_1} and d_{r_2} are their respective number of neighboring regions.*

5.4 Storing multiple connected components

The approaches proposed above support only hierarchies and maps that form a single connected component. However, in some scenarios, maps can be composed by more than one connected component. An example of this would be partitions that include islands. In this section, we present a strategy to support multiple connected components. The strategy is independent of the approaches proposed in previous sections and can be implemented as an extension of any of them.

The main idea is to extend the use of a planar graph embedding to a multigranular instance, therefore, a requirement for the construction of the planar embeddings is that the entry and exit vertices of a region must be the same. For this reason, the approaches proposed above do not support disconnected sub-regions, because depending on the way the traversal is made at the time of building the planar embeddings, the entry and exit vertices may not be the same. To solve this problem, it is proposed to split the regions containing disconnected sub-regions in such way that the new regions generated contain only connected regions.

Given a region r at level L_i, composed by $c > 1$ connected components, we treat the connected components as independent regions r_1, r_2, ..., r_c, increasing the total number of regions at level L_i. To store the information that regions $r_1, r_2, \ldots, r_c$ actually conform only one region r, we store two bitmaps, D_i and MD_i, and an integer array C_i. The entry $D_i[r] = 1$ indicates that region r is conformed by multiple connected components; otherwise, $D_i[r] = 0$. The bitmap MD_i stores in unary the number of connected components of region r and the array C_i stores the regions r_1, r_2, ..., r_c, when $c > 1$. The construction of the representation is performed as follows:

1. We perform a traversal of the planar embedding of level L_i detecting the set R of regions composed by multiple connected components with respect to the level L_{i+1}.

2. For each region $r \in R$, r is partitioned into its $c > 1$ connected components r_1, r_2, ..., r_c. The entry $D_i[r]$ is set to 1, the sequence $0^{c-2}1$ is appended to the bitmap MD_i, and the regions r_1, r_2, ..., r_c are appended to the array C_i.

3. The embedding of level L_i is updated with the new regions $r_1, r_2, \ldots, r_c$.

4. We repeat steps 1-3 for level L_{i-1}.

Figure 5.5 shows an example of how regions are partitioned.

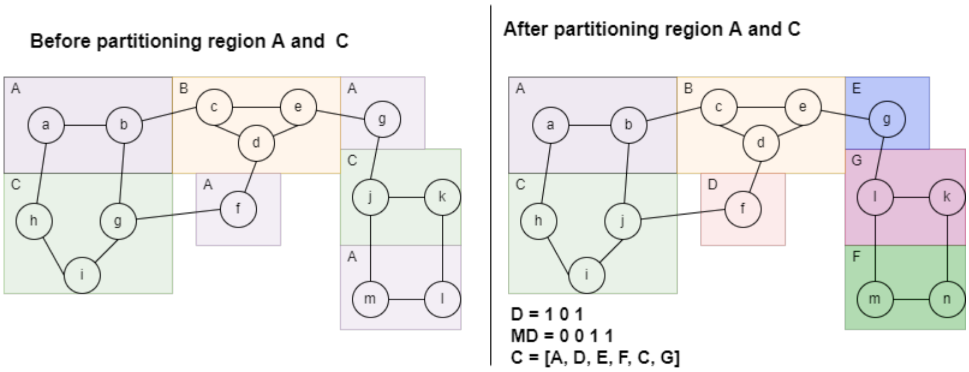

Before partitioning region A and C
After partitioning region A and C
A
B
C
D
E
F
G
a
b
c
d
e
f
g
h
i
j
k
l
m
n
D = 1 0 1
MD = 0 0 1 1
C = [A, D, E, F, C, G]

Figure 5.5: Example partitioning a region

Under this setting, operation contains(r', r), with $r' \in L_i$, $r \in L_j$ and $L_i \prec L_j$, refers to whether every connected component of r is contained in some connected component of r'. To support it, we first recover all the connected components of r' and mark them in a bitmap B. Then we check if $D_j[r] = 1$, to determine if the region r is partitioned. If needed, we obtain its connected components by traversing the range $C_j[p, q]$, where $p = select_1(MD_j, rank_1(D_j, r) - 1) + 1$ and $q = select_1(MD_j, rank_1(D_j, r))$. We map each connected component r_k of r to its containing region r'_k at level i, an check if r'_k is marked in B. We return whether every component r'_k was marked in B. The time complexity is $O(w_{ij}c + c')$, where c' and c are the number of connected components of r' and r, respectively, and w_{ij} is the cost of mapping regions from level j to level i, which can be done with any of the solutions discussed in Sections 5.1.1–5.3.

Similarly, operation touches(r', r) checks whether some connected component of r' shares a boundary with some connected component of r. To solve it, for each connected component of r we map its neighbors, at level j, to their containing regions at level i, marking them in a bitmap B. Finally, we compare the connected components of r' with the marked regions in B, and return whether a coincidence is found. The time complexity is $O(\hat{d}_r w_{ij} + c')$, where $\hat{d}_r$ is the number of neighbors of r at level j, computed as the sum of the neighbors of each connected component that conforms r.

Operation contained(L_j, r), with $r \in L_i$ and $L_i \prec L_j$, lists all the regions at granularity level L_j that are contained in some connected component of r. To implement it, we recover all c connected components at level i of r, and map each of them to its descendants at level j. Notice that the resulting regions at level j may be grouped into $c' \geq 1$

connected components, which must be recovered as for the basic case. Thus, the time complexity is $O(w_{ij}c + t_{c'}c')$, where $t_{c'}$ is the cost of traversing the regions contained in the c' connected components at level j.

The additional space consumption for arrays C_i, MD_i, and D_i, is $O(n + c \log c)$ bits, where c is the number of components across all levels. Note that the connected regions involve only 1 bit of extra space, used in D_i to indicate they have only one connected component. In practical datasets, c is much smaller than n (see the next section, for example).

5.5 Experimental evaluation

5.5.1 Experimental setup

All the experiments were carried out on a computer equipped with an Intel Core i7 (3820) processor, clocked at 3.6 GHz; 32 GB DDR3 RAM memory, clocked at 1,334 MHz; 4 physical cores each one with L1i, L1d and L2 caches of size 32 KB, 32 KB and 256 KB, respectively; and a shared L3 cache of size 10 MB. The computer runs Linux 3.13.0-86-generic, in 64-bit mode. All our algorithms and the baseline were implemented in C++, using the library SDSL [61], and compiled with GCC version 4.8.4 and -O3 optimization flag. For the compact planar embeddings, we directly use the code of [48]. We measured running times using the `clock_gettime` function.

Datasets

The datasets used to evaluate our approaches are based on the TIGER dataset,[2] provided by the U.S. Census Bureau, which corresponds to geographic and cartographic data of the administrative divisions in the United States. The dataset is organized as a hierarchy of granularities with levels L_1 to L_6 being State, County, Census tract, Census block group, Census block, and Face, respectively (see Table 5.3). With this base information, we generated four datasets, `tiger_8s`, `tiger_usa`, `whole_usa`, and `tiger_usa`$^+$.

[2] TIGER dataset, version 2019. https://www2.census.gov/geo/tiger/TIGER2019/

Dataset	Level	Vertices (n)	Edges (m)	Dataset	Level	Vertices (n)	Edges (m)
	L_1	9	20		L_1	50	140
	L_2	595	1,730		L_2	3,110	9,095
tiger_8s	L_3	11,626	31,412	tiger_usa	L_3	72,512	201,631
(1 comp.)	L_4	33,804	91,891	(1 comp.)	L_4	216,243	597,784
	L_5	2,233,031	5,429,483		L_5	11,004,160	26,732,935
	L_6	4,761,354	10,326,904		L_6	19,735,874	43,837,150
	L_1	57	140		L_1	3,852,017	6,392,483
	L_2	3,235	9,102		L_2	4,518,394	8,364,881
whole_usa	L_3	74,135	201,824	tiger_usa$^+$	L_3	5,686,152	11,767,903
(98 comp.)	L_4	220,743	598,245	(1 comp.)	L_4	7,821,874	17,711,491
	L_5	11,166,337	26,746,322		L_5	11,846,172	27,868,766
	L_6	20,037,199	44,503,624		L_6	19,735,874	43,837,150

Table 5.3: Datasets used in our experiments. Each level includes one node representing the external face of the embedding.

The first dataset, tiger_8s, contains the information of eight neighboring states (Nevada, Utah, Arizona, Colorado, New Mexico, Kansas, Oklahoma and Texas), while tiger_usa includes the information of the whole continental part of the country. During the construction of both datasets, we found cases where a region was composed of disconnected subregions (e.g., Santa Catalina Island is a disconnected region of the State of California). In such cases, we only considered the largest subregion. Additionally, both datasets are conformed by one connected component.

On the other hand, the dataset whole_usa corresponds to the tiger_usa dataset, but including the disconnected subregions, and Alaska, Hawaii and overseas U.S. islands, being conformed by 98 connected components. Finally, we generated the synthetic dataset tiger_usa$^+$, which corresponds to the dataset tiger_usa with a different (fictitious) grouping of regions. By choosing random starting regions at level L_6, a BFS traversal was performed to group from 1 up to 10 contiguous regions into one. The BFS traversals were performed until all regions of level L_6 were grouped. The procedure was repeated for all levels L_5 up to L_2. We use this dataset to evaluate situations where the ratio of grouping is smaller than in the original dataset.

5.5.2 Evaluated implementations

Based on the approaches described in Sections 5.1.1 to 5.3 for the representation of the multi-granular maps, we developed the following implementations:

Approach 1 (T**).** Implementation based on the approach described in Section 5.1.1, which uses compact planar embeddings to represent each level of granularity, as well as $h-1$ bitmaps, where we use a plain bitmap for level h and bitmaps of type T for the rest of levels, to store hierarchy-related information, where T can be: *i)* PLAIN (a plain bitvector), *ii)* SD (the sparse bitmap SD-array [111]), *iii)* RRR (an H_0-compressed bitvector [120]).

Approach 2 (RMMT**).** Implementation based on the approach described in Section 5.2, which uses a compact planar embedding for the highest level of detail, and range min-max trees (RMMT) for the sequences $B_{[]}$ and $B_{()}$, to represent the mapping among aggregation levels.

Approach 2 (PLAIN-S**).** A variant of the previous one that uses compact planar embeddings to represent each level of granularity, and range min-max trees over integer vectors (stored as a plain vector) to store the hierarchical information represented in the sequence $B_{()}$. Although storing the hierarchical information implies an increase in the space usage compared to what was proposed in Section 5.2, it drastically improves the query time of the proposed operations. For the scanning of the RMMT blocks, two strategies S are evaluated: linear search (L) and binary search (BS), where binary search can be done by computing $rank_{\leq i}$ on each comparison.

Approach 2 (WT-S**).** Another variant of the approach described in Section 5.2, similar to APPROACH 2 (RMMT). This implementation uses compact planar embeddings to represent each level of granularity and range min-max trees, with the difference that it uses a wavelet tree to store the hierarchical information represented in the sequence $B_{()}$. This represents a saving in terms of space usage when compared with Approach 2 (PLAIN-S), at the cost of a slower access time to the elements in $B_{()}$. Again, for the scanning of the RMMT blocks, two strategies S are evaluated: linear search (L) and binary search (BS).

Approach 3. Implementation based on the approach described in Section 5.3, which uses compact planar embeddings to represent each level of granularity in combination with a balanced parenthesis sequence representing the hierarchy tree and a compact permutation data structure representing M, for the mapping between planar embeddings and the hierarchy tree.

Baseline. As a baseline, we developed a data structure that also uses the compact planar embeddings of [48] to represent each level, but the hierarchy is stored in non-compact form. Specifically, each level $i \in \{0..h-1\}$ of the hierarchy is stored in a vector in which position j, representing a region r', stores the index of the region r at level $i-1$ that contains r'. In addition, for a region r at level i, the data structure stores pointers to all the regions at level $i+1$ contained in r. In this data structure, the operation go_level(x, i, j) is supported in $O(h)$ time, because all the levels of the hierarchy are traversed in the worst case. All the main operations were implemented in a similar fashion to our approaches, hence providing running times of $O(h)$, $O(min(d_{r_1}, d_{r_2})h)$ and $O(\sum_{k=i}^{j} n_k)$, for contains(r_1, r_2), touches(r_1, r_2), and contained(L_j, r_1), respectively, where r_1 is a region at level L_i and r_2 a region at level L_j.

5.5.3 Performance on connected regions

We first consider the basic case of connected regions. The performance of APPROACH 2 variants is mainly dependent on the use of the RMMT, and this in turn depends on the length l of the RMMT blocks. We considered values $l \in [2^4 \,..\, 2^{15}]$.

Regarding the evaluation of operations contains and touches, we executed 200 random operations for each pair of aggregation levels.[3] As for operation contained, we executed the queries between all possible pairs of aggregation levels. For contains and contained, there are 15 valid pairs $((L_i, L_j), i \in [1, 5], j \in [i+1, 6])$, whereas for touches there are 21 valid pairs $((L_i, L_j), i \in [1, 6], j \in [i, 6])$. This gives a total of 3,000

[3] The outer face is omitted from the pool of candidates because of its very large number of neighbors, which may impact the results.

operations of the first type, 4,200 operations of the second type, and 11,666,872 operations of third type. In the results, for each experiment we show the average time of 30 repetitions.

Figure 5.6 shows the space-time tradeoffs obtained on the datasets tiger_usa and tiger_usa$^+$ (we omit dataset tiger_8s because it performed similarly to tiger_usa), with the three operations.

The first observation is that, as expected, APPROACH 2 (RMMT) uses by far the least amount of space, using as little as 8–12 bits per region. In exchange, however, it is one and even two orders of magnitude slower than other approaches, because of the need to navigate over simulated parenthesis sequences.

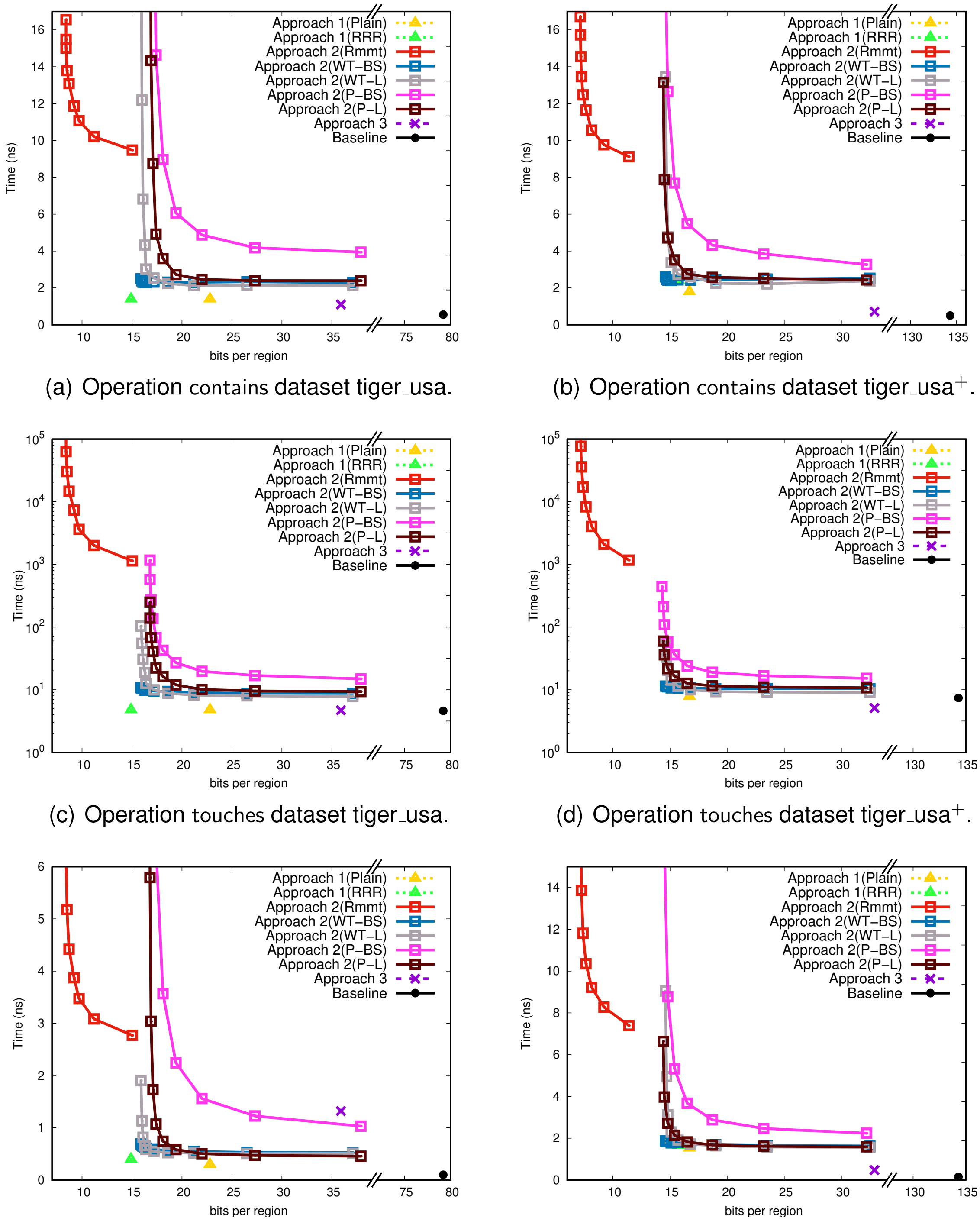
The second observation is that APPROACH 1 (SD) essentially dominates all the other approaches in the space-time tradeoff map of tiger_usa, using 15–16 bits per region and taking 0.4–10 nanoseconds per operation. The only exception is the baseline, which sometimes outperforms APPROACH 1 (SD) in time, yet at the cost of using 80–135 bits per region, that is, about 5–8 times more space.

On the synthetic dataset tiger_usa$^+$, we use APPROACH 1 (RRR) instead of APPROACH 1 (SD), because it saves more space. In this dataset, the least-space variant of APPROACH 2 (WT-BS) is equally fast and uses slightly less space (indeed, the sweet points of several other variants are pretty close). In this dataset, APPROACH 3 offers considerably better times using about twice the space, around 34 bits per region.

The only considerably worse variant is APPROACH 2 (PLAIN-BS), followed by APPROACH 2 (PLAIN-L) in the dataset tiger_usa.

Figures 5.7 and 5.8 show the results grouped by distance level, where all valid pairs $(L_i, L_j),\ i \in [1, 6-c],\ j = i+c$ are grouped into the distance level c. For APPROACH 2 we only maintain the variants APPROACH 2 (RMMT) with block length $l = 2^9$ and APPROACH 2 (WT-BS) with $l = 2^{15}$. Hereinafter, in our analysis of the second approach, we only show those two implementations and their best configurations. For the contained operation, the running time was normalized by the number of regions returned.

In general, the distance c does not significantly affect the time performance of the operations, except for the operation contained, where times tend to improve with larger distances. This is because more regions are reported as the distance grows, and this



(a) Operation contains dataset tiger_usa.

(b) Operation contains dataset tiger_usa$^+$.

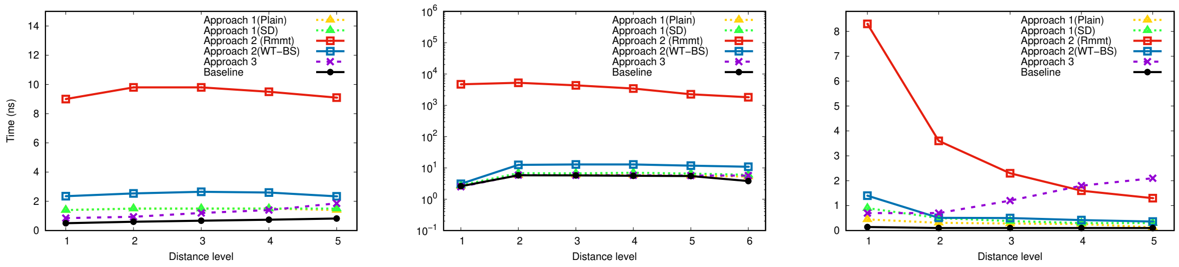
(c) Operation touches dataset tiger_usa.

(d) Operation touches dataset tiger_usa$^+$.

(e) Operation contained dataset tiger_usa.

(f) Operation contained dataset tiger_usa$^+$.

Figure 5.6: Running time in nanoseconds using the datasets `tiger_usa` and `tiger_usa`$^+$.

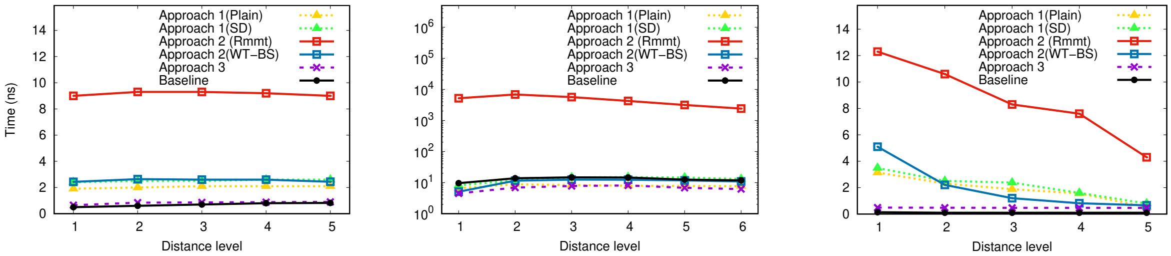


(a) Operation contains.

(b) Operation touches.

(c) Operation contained.

Figure 5.7: Running time in nanoseconds using the dataset tiger_usa, where distance level corresponds to the distance between the levels of the queried granules.



(a) Operation contains.

(b) Operation touches.

(c) Operation contained.

Figure 5.8: Running time in nanoseconds using the dataset tiger_usa^{+}.

decreases the time per reported region due to cache effects. In general, the baseline is the fastest implementation on all the operations. It is, however, closely followed in almost all cases by some variant of APPROACH 1, which uses many times less space.

Similar results can be observed for the dataset tiger_usa^{+}, except that APPROACH 3 becomes the second fastest on the operation contained. This owes to the way this dataset was constructed: its hierarchy tree is wider and has more nodes than the hierarchy tree of tiger_usa. APPROACH 3 is more cache-friendly when reporting many nearby regions.

5.5.4 Space Usage

Table 5.4 shows the space consumption of the baseline and the three different approaches. For the datasets tiger_8s and tiger_usa, APPROACH 1 (PLAIN) uses about 30% the space consumed by the baseline while APPROACH 1 (SD) uses about 19%.

	AP. 1 (PLAIN)	AP. 1 (COMPRESS)	AP. 2 (RMMT)	AP. 2 (WT)	AP. 3	Baseline
tiger_8s	23.6	13.1	8.7	14.0	32.9	66.9
tiger_usa	88.4	57.9	38.0	62.1	137.3	310.6
tiger_usa$^+$	111.6	104.9	54.4	97.6	196.9	897.4

Table 5.4: Space usage in MB.

For the dataset tiger_usa$^+$, both implementations use about 12% of the space consumed by the baseline. It is important to notice that the compressed bitmaps, either SD or RRR, do not save much extra space for the dataset tiger_usa$^+$, due to the high amount of 1-bits because of the smaller grouping ratio between contiguous levels of the dataset.

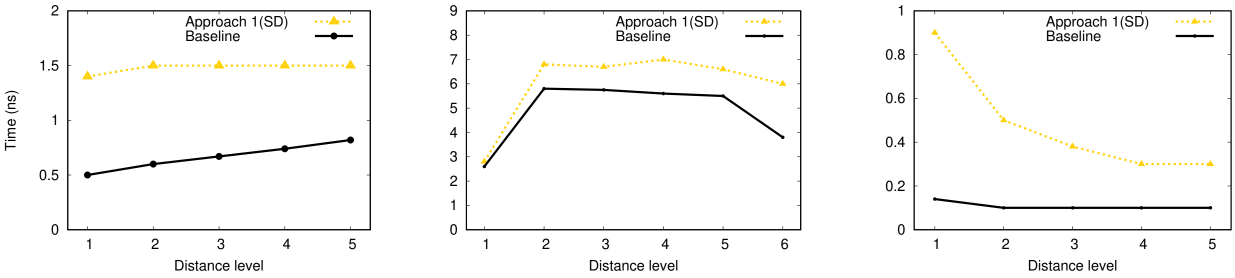
APPROACH 2 (RMMT) is the most space-efficient implementation, using only 12% of the space consumed by the baseline for datasets tiger_8s and tiger_usa), and 6% for the dataset tiger_usa$^+$. APPROACH 2 (WT) uses about 21% for the first two datasets and 11% for tiger_usa$^+$. Finally, APPROACH 3 uses about 46% for the first two datasets and 20% for the last one.

In general, the change in the topological hierarchy of the dataset tiger_usa$^+$ compared to the dataset tiger_usa affects the baseline in a greater extent than the compact representations.

5.5.5 Performance with non-connected components

Because of the low number of related components present in the datasets, the strategy presented did not incur a significant overhead of space or a significant increase in execution time, so replicating the experimentation for all implementations would not reflect changes with respect to the graphs presented in Figure 5.9, therefore, only the implementation that showed the best trade-off was considered.

A final experimental evaluation was performed using the dataset whole_usa in order to measure the impact of the proposed strategy for dealing with more than one connected component. Since the number of connected components is low regarding to the total number of regions (98 connected components and around 20 million regions at level L_6), the expected overhead of the proposed strategy is very limited. Thus, to



(a) Operation contains.

(b) Operation touches.

(c) Operation contained.

Figure 5.9: Running time in nanoseconds using the dataset whole_usa.

represent non-connected components, we extended the approach with the most interesting time-space trade-off, APPROACH 1 (SD).

Regarding space consumption, the baseline uses 325.2 MB, while APPROACH 1 (SD) uses 60.1 MB, where 0.5 MB correspond to space consumption of bitmaps D, MD and C. The proposed strategy of Section 5.4 adds 43,009 new regions obtained from the partition of regions with more than one connected component, which induces less than 1% of extra space.

Figure 5.9 shows the average running time for the three operations. From the figure, we can conclude that the proposed strategy to deal with multiple connected components impacts the execution time in a negligible way, maintaining running times similar to those of Figure 5.7.

Chapter 6

Discussion and Future Work

In this chapter, a more general discussion about the results obtained in the experimental evaluations of the two main contributions of this thesis (Chapters 4 and 5) is presented, as well as some open problems and potential new lines of research arising from the work presented in this thesis.

6.1 Discussion

Recall that the main contribution of this thesis is to provide strategies for the implementation of two models focused on the management of multigranular information.

First, we present algorithms for a space-efficient and response-time efficient implementation of a general multigranular model presented in [73] (Chapter 4). Based on the experimental evaluation carried out, we can conclude that the strategies presented allow an efficient derivation of the relations covered by the model, being the $GBP + Matrix$ implementation the one that showed the best performance on average for all the covered inference rules. Although the $Baseline$ implementation shows better response time for certain queries, it should be noted that $GBP + Matrix$ shows a notoriously better response time than the other implementations evaluated for queries that required a longer time. It is important to highlight the following conclusions obtained through experimental evaluation: i) Despite the theoretical complexities of the developed algorithms, in practice there were unexpected results, which are mainly due to the impact of cache failures at the time of carrying out the queries; an example of this can be seen in the queries associated with the inference rule 1, where the baseline implementation showed a worse performance than the other alternative structures. This is mainly because it is the largest data structure and is based on an adjacency list, which is an implementation prone to have more cache failures. ii) $GBP + Matrix$ is the implementation that occupies less space, as mentioned in chapter 4, which is because it occupies space proportional to the amount of relations other than subsumption and only requires storing a single mapping vector. This implies that in certain datasets that are not very favorable for the structure, where there is a larger number of relationships other than subsumption in comparison to the total number of granules, the space used by the dataset will be similar to the one used by GBP or $GLOUDS$. iii) Inference rule 4 shows that the GBP-based implementations presents a notoriously better performance on average than the other implementations for the 4 datasets evaluated. This holds particular significance since a substantial portion of response time is dedicated to retrieving the descendants of the queried granules. The efficiency demonstrated in obtaining descendants, a common query when dealing with hierarchies, underscores the potential for new research avenues aimed at harnessing and further optimizing this advantage.

Second, in Chapter 5 new data structures to implement a topological model that represents and accesses data organized as a hierarchical structure of regions defined by the inclusion relation are proposed. Three different strategies were developed for the management of hierarchical information, each one presenting its own particularities: i) The first proposed approach is the one that presents the best trade-off between response time and space used, serving as a general use structure when not seeking to improve space used or response time for a particular query. ii) A second approach, which focuses on minimizing the redundant information in the strategy presented in the first approach, uses less space (6% of the space used by the baseline), at the expense of being the one that presents the worst response time for the 3 queries analyzed, and therefore, being the variant to use in cases where it is necessary to minimize space. iii) a third approach, focused on improving the response times presented by the two previous approaches, changing the way of representing the hierarchy presented in the other approaches by a new data structure called hierarchical tree. Due to the introduced new structure, it occupies more space, in exchange for improving the response times to obtain the descendants of a node (which is used to detect the nodes contained at a certain specific level in the hierarchy). It should be noted that this implementation presents better performance in datasets of higher complexity in terms of number of granules and partitions per level.

6.2 Open lines

The following are several lines of research stemming from the work presented in this thesis, beginning by commenting on the lines of research focused on the work presented in Chapter 4, continuing with the lines of research focused on the work presented in Chapter 5, and ending with some topics of common interest for all the work presented in this thesis.

6.2.1 Future works related to the data structure for multigranular model

Chapter 4 presents strategies and algorithms for an efficient implementation both, in terms of space used for the representation of the relationships between granules and query time, for the multigranular model presented in [73], using the inference rules proposed in [75]. The work developed focused on how to implement a subset of rules efficiently at a particular level. Based on what has been presented, the following works are proposed to be developed in order to provide a data structure capable of being used for an implementation of the proposed multigranular model in [73].

Algorithm for the not supported inference rule

Strategies for inference rules related to subsumption, disjoint and not-disjoint relations were proposed, but for the case of the not-subusmption relation, only a subset of the proposed rules in [75] were implemented, specifically the following rule was not implemented:

$$\frac{\bigsqcap_{\mathfrak{G}}\{g_1, g_3\} \neq \bot_{\mathfrak{G}} \quad g_3 \sqsubseteq_{\mathfrak{G}} g_4 \quad \bigsqcap_{\mathfrak{G}}\{g_2, g_4\} = \bot_{\mathfrak{G}}}{g_1 \not\sqsubseteq_{\mathfrak{G}} g_2}$$

The main difficulty in developing a strategy for this rule is due to the presence of two relations other than containment as the premise of the rule, so it was not possible to develop a strategy that would ensure a complete derivation of the rule while maintaining a fast response time.

Minimum set of information to be stored

The strategies proposed for the inference rules covered in this paper are capable of deriving relations from a set of explicitly stored relations, however, this set may have redundancies in the sense of storing relations that can be derived through the same rules. This implies an extra cost in the space used, in addition to negatively influence the performance of the queries in practice, since the cost of obtaining redundant granules is greater than the possible savings by having more granules that can answer the query. Having a characterization of the minimum set to be stored facilitates the development of an algorithm for the uncovered inference rule, by being able to characterize the relationships involved in the rule, in the same way, it facilitates the improvement of the algorithms and the proposed structure. Because of this, it is relevant to develop an algorithm capable of reducing a set S of explicit relationships to be stored to a subset S' that has no redundant relationships.

Algorithms for deriving relations

As previously mentioned, the primary focus of this work was on developing efficient algorithms for testing individual inference rules. However, a comprehensive query-time strategy for collectively testing all the inference rules for each relation was not explicitly addressed. The significance of this aspect becomes evident in the second experimental evaluation conducted on the real dataset. In this evaluation, the performance in deriving negative relations deteriorates compared to the specific evaluation of each rule, mainly due to the potential re-processing of previously computed information and an increase in cache missed. We propose the integration of the rules into a unified method capable of verifying each relation, akin to the approach employed for rules 7 and 8.

Other research lines

In addition to the lines of research described above, there are still several challenges to be solved in order to achieve a practical implementation for the multigranular model. Some of the relevant challenges are listed below:

1. **Granularity structure**: The work developed in the implementation of the inference rules only concerns the granule structure, without considering the information provided by the granularity structure. In [74] there are several bigranular relationships that define the behavior at granularity level between the granules that compose them, and we propose to integrate the information provided by the various possible bigranular relationships to the algorithms already developed in this thesis, in order to speed up the queries and reduce the amount of information to store.

2. **Integration mechanisms**: One of the particularities of this multigranular model is the facility it has to add new multigranular data sources; because of this, it would not be strange the emergence of discrepancy between the information at the time of adding new data sources. The development of verification and integration mechanisms is proposed, with the objective of maintaining the correctness of the previously stored information.

3. **Complex queries and new types of relations**: The multigranular model focuses on the containment and disjoint relationships (and their respective negations), arguing that on the basis of these it is possible to elaborate more complex rules. The present work focuses only on the derivation of these relations. It is proposed the development of a set of common queries in multigranular instances and algorithms for these, such as, for example, obtaining all granules that intersect but neither subsume nor are subsumed by a g granule, taking advantage of the strengths of the developed structure, such as its effectiveness in obtaining the descendants of a granule.

6.2.2 Future work related to compact representations of spatial hierarchical structures

Chapter 5 presents 3 different strategies for the implementation of a topological model with the ability to answer queries within a multigranular context, where the hierarchical structure is extracted from the containment relationship of a region and the different partitions that can be generated from it. The proposed implementations present different trade-offs between the space used and the response time of the proposed queries, varying from a data structure that does not present redundant information, at the cost of a longer query time, to an implementation that stores extra information, in order to improve response times in practice. The following is a description of some of the lines of research of interest arising from this work.

Optimize distances between parentheses

We can divide the proposed structures into two components, i) the representation of each level of the hierarchy, and ii) the representation of the hierarchical division. One of the main structures for the representation of both components is the sequence of balanced parentheses (a.k.a. BP). One of the main operations to use in the BP is the $find\_close$ operation, used to navigate between the different nodes represented by the BP, having a complexity of O($log\frac{b-a}{N}$), where N is the length of the parenthesis sequence, a is the position of the open parenthesis, and b is the position of the closed parenthesis associated with a. Because of this, one way to speed up queries in practice is to decrease the distance between the parentheses. This is valid for the presented work since there are different ways to build the $pemb$ (planar embedding) given an initial partition S (this is because any spanning tree is valid for the construction). The work presented in [81] explores various strategies for $pemb$ generation, demonstrating an improvement in query times by reducing the distance between parentheses. In the case of the present work, a new variable to consider is added, since the way in which the initial $pemb$ is generated influences the generation of the BP used for the hierarchy representation. Because of this, we propose the development of new strategies and heuristics for the generation of the $pemb$, evaluating the performance of the queries and analyzing in particular the effect of these strategies both for the generation of the initial $pemb$ and the generation of the hierarchical representation in an invidual way.

Inclusion of relevant queries

The work developed is focused on answering 3 queries: Contains, Touches and Contanied, these are not the only queries of interest to answer in a hierarchical model, given that the model is based on extracting the hierarchy from a partition of a spatial region, answering the eight named spatial relationship predicates defined in the standard OGC SFS 2.1, and three non-standard relationship predicates. The 8 spatial relationship predicates are of direct verification using the implemented queries, an example of this is to verify $overlap$, where, given that we work with partitions of a region, it is not possible that there is a pair of regions that meet the condition, or the case of checking $intersect$, where it is enough to answer the touches query. However, to answer $ContainsProperly$, one must perform a combination of contains and touches queries. This opens the possibility of developing algorithms that seek to simultaneously verify both conditions, instead of verifying each one individually.

Finally, in the case of query $Touches$, there is still a opportunity for improvement, given that it would be expected to achieve an implementation that requires time proportional to the number of neighbors of one of the regions. This could be addressed either by using some auxiliary structure to speed up this query while maintaining the proposed strategies, or by developing a new data structure focused on improving this query.

6.2.3 Common open problems in both multigranular models

Finally, two problems of common interest for both multigranular models covered in this thesis are mentioned below.

- **Dynamic structures**: The structures devised in this thesis are static, meaning they do not support the modification of information once the structure is constructed. While there exist dynamic proposals for the foundational components used in developing structures for each model, a direct application is insufficient. For instance, in the case of GBP, a proper mapping between granules and their identifiers obtained during construction is required. Similarly, for $pembs$, maintaining coherence in the order in which regions are traversed during construction is essential. The development of strategies that allow the correct integration and use of dynamic variants of the components for the multigranular models covered in this thesis is proposed.

- **Evaluation of multigranular models in particular instances**: For both multigranular models we mainly used synthetic data or real datasets related to geographic divisions, for example, an instance of spatial partitioning, or hierarchical partitioning based on web pages, this type of data source is widely known, it would be interesting to use other multigranular instances different from the typical geographic or temporal divisions, in order to evaluate the behavior in instances with different characteristics and queries to be answered, .

Chapter 7

Conclusions

At present, the amount of information that needs to be stored and processed is greater than ever, every day the available data sources increase, in the same way the available information evolves in terms of its form, it becomes more and more frequent the need to manage information that is available at different levels of detail. Because of this, it becomes a necessity to have tools that are capable of both storing and processing this information efficiently. The work developed in this thesis presents strategies for the implementation of two models focused on the management of multigranular information. The developed algorithms are mainly based on the use of succinct data structures, with the objective of reducing the use of space, while maintaining response times similar to common algorithms or data structures.

First, data structures and strategies were presented to allow the development of an efficient implementation of the multigranular model presented in Chapter 4. For this multigranular model, several inference rules were proposed in [76], with the objective of not storing all the information explicitly, but only a subset of it. At that time, there were no other works focused on the development of efficient structures for deriving subsumption and disjointness relations. This work proposed several strategies for deriving information based on a set of inference rules, which had been proven to be both correct and complete. The implementation was correct and complete for the subsumption, disjoint, and not-disjoint relations. The experimental evaluation demonstrated that the proposed methods were not only more compact but also significantly faster when compared to a baseline approach that explicitly stored the information of each relation using graphs and solved operations using well-known traversal algorithms.

Secondly, data structures and algorithms were presented that are capable of compactly representing a hierarchical partitioning of the space, so that basic queries regarding containment and adjacency of regions of arbitrary levels can be computed efficiently. It is known that the topology of a planar graph with n regions without a hierarchy can be efficiently manipulated within $4n + o(n)$ bits. On a hierarchy of height h, our representation requires as little as $O(n \log h)$ bits, which becomes $O(n)$ if the number of regions increases by a multiplicative constant from each level to the next. Within this asymptotically optimal space, we designed various representations that efficiently determine (1) whether a region contains another, (2) whether a region touches another, and (3) all the regions of some level contained by a given region. Our experimental results showed that we could represent the partitioning and hierarchical information within as little as 8 bits per region in practice, which is about twice the space required to represent a partition without hierarchies. Further, with about 16 bits per region (i.e., roughly 4 times the space without hierarchies), our data structures answered all queries within 10 nanoseconds per retrieved element, and in some cases, less than half a nanosecond.

In this thesis, two compact structures were proposed, and different algorithms were developed for them, with a focus on the multigranular context. In the first part, strategies were developed for a multigranular model based on subsumption and disjoint relations, while in the second part, the focus was on an implementation geared towards answering queries about a spatial structure generated from the division of a region S and the containment relationship between the various sub-regions generated. It is expected that this thesis will help to strengthen the research related to multigranular models, as well as the development of new compact structures focused on particular instances.

Bibliography

[1] Luca Castelli Aleardi, Olivier Devillers, and Gilles Schaeffer. Succinct representation of triangulations with a boundary. In *Workshop on Algorithms and Data Structures*, pages 134–145. Springer, 2005.

[2] Sandra Álvarez García. Compact and efficient representations of graphs. 2014.

[3] Diego Arroyuelo, Rodrigo Cánovas, Gonzalo Navarro, and Kunihiko Sadakane. Succinct trees in practice. In Guy E. Blelloch and Dan Halperin, editors, *Proceedings of the Twelfth Workshop on Algorithm Engineering and Experiments, ALENEX 2010, Austin, Texas, USA, January 16, 2010*, pages 84–97. SIAM, 2010.

[4] Paolo Atzeni and Douglas Stott Parker Jr. Formal properties of net-based knowledge representation schemes. *Data Knowl. Eng.*, 3:137–147, 1988.

[5] Paolo Atzeni and Douglas Stott Parker Jr. Set containment inference and syllogisms. *Theor. Comput. Sci.*, 62(1-2):39–65, 1988.

[6] Jérémy Barbay, Luca Castelli Aleardi, Meng He, and J. Ian Munro. Succinct representation of labeled graphs. *Algorithmica*, 62(1-2):224–257, 2012.

[7] Jérémy Barbay, Meng He, J Ian Munro, and Srinivasa Rao Satti. Succinct indexes for strings, binary relations and multilabeled trees. *ACM Transactions on Algorithms (TALG)*, 7(4):1–27, 2011.

[8] Alberto Belussi, Carlo Combi, and Gabriele Pozzani. Formal and conceptual modeling of spatio-temporal granularities. In *International Database Engineering and Applications Symposium IDEAS*, pages 275–283. ACM, 2009.

[9] Brandon Bennett. Determining consistency of topological relations. *Constraints*, 3(2):213–225, 1998.

[10] David Benoit, Erik D. Demaine, J. Ian Munro, Rajeev Raman, Venkatesh Raman, and S. Srinivasa Rao. Representing trees of higher degree. *Algorithmica*, 43(4):275–292, 2005.

[11] Elisa Bertino, Dolores Cuadra, and Paloma Martínez. An object-relational approach to the representation of multi-granular spatio-temporal data. In Oscar Pastor and João Falcão e Cunha, editors, *Advanced Information Systems Engineering, 17th International Conference, CAiSE 2005, Porto, Portugal, June 13-17, 2005, Proceedings*, volume 3520 of *Lecture Notes in Computer Science*, pages 119–134. Springer, 2005.

[12] Claudio Bettini, Curtis E. Dyreson, William S. Evans, Richard T. Snodgrass, and Xiaoyang Sean Wang. A glossary of time granularity concepts. In *Temporal Databases, Dagstuhl*, pages 406–413, 1997.

[13] Claudio Bettini, Sushil Jajodia, and Xiaoyang Sean Wang. *Time Granularities in Databases, Data Mining, and Temporal Reasoning*. Springer, 2000.

[14] Thomas Bittner and Barry Smith. A taxonomy of granular partitions. In *Spatial Information Theory: Foundations of Geographic Information Science, International Conference, COSIT 2001, Morro Bay, CA, USA, September 19-23, 2001, Proceedings*, volume 2205 of *Lecture Notes in Computer Science*, pages 28–43. Springer, 2001.

[15] Daniel K Blandford, Guy E Blelloch, and Ian A Kash. Compact representations of separable graphs. 2003.

[16] Guy E Blelloch and Arash Farzan. Succinct representations of separable graphs. In *Combinatorial Pattern Matching: 21st Annual Symposium, CPM 2010, New York, NY, USA, June 21-23, 2010. Proceedings 21*, pages 138–150. Springer, 2010.

[17] Nicolas Bonichon, Cyril Gavoille, Nicolas Hanusse, Dominique Poulalhon, and Gilles Schaeffer. Planar graphs, via well-orderly maps and trees. *Graphs and Combinatorics*, 22:185–202, 2006.

[18] Paul De Bra and Jan Paredaens. Horizontal decompositions for handling exceptions to functional dependencies. In Hervé Gallaire, Jean-Marie Nicolas, and Jack Minker, editors, *Advances in Data Base Theory, Vol. 2, Based on the Proceedings of the Workshop on Logical Data Bases, December 14-17, 1982, Centre d'études et de recherches de Toulouse, France*, Advances in Data Base Theory, pages 123–141, New York, 1982. Plemum Press.

[19] Nieves R. Brisaboa, Ana Cerdeira-Pena, Guillermo de Bernardo, Gonzalo Navarro, and Oscar Pedreira. Extending general compact querieable representations to GIS applications. *Inf. Sci.*, 506:196–216, 2020.

[20] Nieves R. Brisaboa, Antonio Fariña, Daniil Galaktionov, and M. Andrea Rodríguez. A compact representation for trips over networks built on self-indexes. *Inf. Syst.*, 78:1–22, 2018.

[21] Nieves R. Brisaboa, Adrián Gómez-Brandón, Gonzalo Navarro, and José R. Paramá. Gract: A grammar-based compressed index for trajectory data. *Inf. Sci.*, 483:106–135, 2019.

[22] Nieves R. Brisaboa, Miguel Rodríguez Luaces, Gonzalo Navarro, and Diego Seco. Space-efficient representations of rectangle datasets supporting orthogonal range querying. *Inf. Syst.*, 38(5):635–655, 2013.

[23] Warren Burton. Representation of many-sided polygons and polygonal lines for rapid processing. *Communications of the ACM*, 20(3):166–171, 1977.

[24] Elena Camossi, Michela Bertolotto, and Elisa Bertino. A multigranular object-oriented framework supporting spatio-temporal granularity conversions. *International Journal of Geographical Information Science*, 20(5):511–534, 2006.

[25] Bernard Chazelle. A functional approach to data structures and its use in multidimensional searching. *SIAM J. Comput.*, 17(3):427–462, 1988.

[26] Muhao Chen, Shi Gao, and X. Sean Wang. Converting spatiotemporal data among heterogeneous granularity systems. In *2016 IEEE International Conference on Fuzzy Systems, FUZZ-IEEE 2016, Vancouver, BC, Canada, July 24-29, 2016*, pages 984–992. IEEE, 2016.

[27] Yi-Ting Chiang, Ching-Chi Lin, and Hsueh-I Lu. Orderly spanning trees with applications. *SIAM J. Comput.*, 34(4):924–945, 2005.

[28] Richie Chih-Nan Chuang, Ashim Garg, Xin He, Ming-Yang Kao, and Hsueh-I Lu. Compact encodings of planar graphs via canonical orderings and multiple parentheses. *CoRR*, cs.DS/0102005, 2001.

[29] David R Clark and J Ian Munro. Efficient suffix trees on secondary storage. In *Proceedings of the seventh annual ACM-SIAM symposium on Discrete algorithms*, pages 383–391, 1996.

[30] Francisco Claude and Gonzalo Navarro. Practical rank/select queries over arbitrary sequences. In Amihood Amir, Andrew Turpin, and Alistair Moffat, editors, *String Processing and Information Retrieval, 15th International Symposium, SPIRE 2008, Melbourne, Australia, November 10-12, 2008. Proceedings*, volume 5280 of *Lecture Notes in Computer Science*, pages 176–187. Springer, 2008.

[31] Francisco Claude, Patrick K. Nicholson, and Diego Seco. Space efficient wavelet tree construction. In Roberto Grossi, Fabrizio Sebastiani, and Fabrizio Silvestri, editors, *String Processing and Information Retrieval, 18th International Symposium, SPIRE 2011, Pisa, Italy, October 17-21, 2011. Proceedings*, volume 7024 of *Lecture Notes in Computer Science*, pages 185–196. Springer, 2011.

[32] E Clementini. Topological relations in spatial databases. *Intelligent Systems: Technology and Applications*, 4:47–71.

[33] Eliseo Clementini and Paolino Di Felice. An algebraic model for spatial objects with indeterminate boundaries. *Geographic objects with indeterminate boundaries*, 2:155–169, 1996.

[34] Eliseo Clementini and Paolino Di Felice. A model for representing topological relationships between complex geometric features in spatial databases. *Information sciences*, 90(1-4):121–136, 1996.

[35] Leila De Floriani, Paola Marzano, and Enrico Puppo. Spatial queries and data models. In *Spatial Information Theory A Theoretical Basis for GIS: European Conference, COSIT'93 Marciana Marina, Elba Island, Italy September 19–22, 1993 Proceedings 1*, pages 113–138. Springer, 1993.

[36] Patrick Dinklage, Jonas Ellert, Johannes Fischer, Florian Kurpicz, and Marvin Löbel. Practical wavelet tree construction. *Journal of Experimental Algorithmics (JEA)*, 26:1–67, 2021.

[37] Matthew P. Dube and Max J. Egenhofer. Partitions to improve spatial reasoning. In Ugur Demiryurek and Mohamed Sarwat, editors, *Proceedings of the 1st ACM SIGSPATIAL PhD Workshop, SIGSPATIAL PhD Workshop 2014, Dallas/Fort Worth, Texas, USA, November 4-7, 2014*, pages 1:1–1:5. ACM, 2014.

[38] Max Egenhofer. A mathematical framework for the definition of topological relations. In *Proc. the fourth international symposium on spatial data handing*, pages 803–813, 1990.

[39] Max J Egenhofer. A formal definition of binary topological relationships. In *International conference on foundations of data organization and algorithms*, pages 457–472. Springer, 1989.

[40] Max J Egenhofer, Andrew U Frank, and Jeffrey P Jackson. A topological data model for spatial databases. In *Symposium on Large Spatial Databases*, pages 271–286. Springer, 1989.

[41] Max J Egenhofer and John Herring. Categorizing binary topological relations between regions, lines and points in geographic databases, the 9-intersection: Formalism and its use for naturallanguage spatial predicates. *Santa Barbara CA National Center for Geographic Information and Analysis Technical Report*, 1(1):94–1, 1994.

[42] Martin Erwig and Markus Schneider. Partition and conquer. In *Spatial Information Theory: A Theoretical Basis for GIS, International Conference COSIT '97, Laurel Highlands, Pennsylvania, USA, October 15-18, 1997, Proceedings*, volume 1329 of *Lecture Notes in Computer Science*, pages 389–407. Springer, 1997.

[43] Jérôme Euzenat and Angelo Montanari. Time granularity. In Michael Fisher, Dov M. Gabbay, and Lluís Vila, editors, *Handbook of Temporal Reasoning in Artificial Intelligence*, volume 1 of *Foundations of Artificial Intelligence*, pages 59–118. Elsevier, 2005.

[44] Arash Farzan and J. Ian Munro. A uniform paradigm to succinctly encode various families of trees. *Algorithmica*, 68(1):16–40, 2014.

[45] Iginia De Fent, Donatella Gubiani, and Angelo Montanari. Granular geograph: a multi-granular conceptual model for spatial data. In Andrea Calì, Diego Calvanese, Enrico Franconi, Maurizio Lenzerini, and Letizia Tanca, editors, *Proceedings of the Thirteenth Italian Symposium on Advanced Database Systems, SEBD 2005, Brixen-Bressanone (near Bozen-Bolzano), Italy, June 19-22, 2005*, pages 368–379, 2005.

[46] Paolo Ferragina, Rodrigo González, Gonzalo Navarro, and Rossano Venturini. Compressed text indexes: From theory to practice. *ACM J. Exp. Algorithmics*, 13, 2008.

[47] Paolo Ferragina, Giovanni Manzini, Veli Mäkinen, and Gonzalo Navarro. Compressed representations of sequences and full-text indexes. *ACM Trans. Algorithms*, 3(2):20, 2007.

[48] Leo Ferres, Jose Fuentes-Sepúlveda, Travis Gagie, Meng. He, and Gonzalo. Navarro. Fast and compact planar embeddings. *Comput. Geom.*, 89:101630, 2020.

[49] Johannes Fischer, Florian Kurpicz, and Marvin Löbel. Simple, fast and lightweight parallel wavelet tree construction. In *2018 Proceedings of the Twentieth Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 9–20. SIAM, 2018.

[50] Johannes Fischer and Daniel Peters. Glouds: Representing tree-like graphs. *Journal of Discrete Algorithms*, 36:39–49, 2016. WALCOM 2015.

[51] José Fuentes-Sepúlveda, Erick Elejalde, Leo Ferres, and Diego Seco. Efficient wavelet tree construction and querying for multicore architectures. In *International Symposium on Experimental Algorithms*, pages 150–161. Springer, 2014.

[52] José Fuentes-Sepúlveda, Erick Elejalde, Leo Ferres, and Diego Seco. Parallel construction of wavelet trees on multicore architectures. *Knowledge and Information Systems*, 51:1043–1066, 2017.

[53] José Fuentes-Sepúlveda, Diego Gatica, Gonzalo Navarro, M Andrea Rodrígucz, and Diego Seco. Compact representation of spatial hierarchies and topological relationships. In *2021 Data Compression Conference (DCC)*, pages 113–122. IEEE, 2021.

[54] José Fuentes-Sepúlveda, Diego Gatica, Gonzalo Navarro, M Andrea Rodríguez, and Diego Seco. Compact representations of spatial hierarchical structures with support for topological queries. *Information and Computation*, 292:105029, 2023.

[55] Jose Fuentes-Sepúlveda, Gonzalo Navarro, and Diego Seco. Implementing the topological model succinctly. In *SPIRE*, pages 499–512, 2019.

[56] Jose Fuentes-Sepúlveda, Gonzalo Navarro, and Diego Seco. Implementing the topological model in near-optimal space and time. *CoRR*, abs/1911.09498, 2021.

[57] Éric Fusy, Gilles Schaeffer, and Dominique Poulalhon. Dissections, orientations, and trees with applications to optimal mesh encoding and random sampling. *ACM Transactions on Algorithms (TALG)*, 4(2):1–48, 2008.

[58] Travis Gagie, Gonzalo Navarro, and Simon J. Puglisi. New algorithms on wavelet trees and applications to information retrieval. *Theor. Comput. Sci.*, 426:25–41, 2012.

[59] Travis Gagie, Gonzalo Navarro, and Simon J Puglisi. New algorithms on wavelet trees and applications to information retrieval. *Theoretical Computer Science*, 426:25–41, 2012.

[60] Richard F. Geary, Rajeev Raman, and Venkatesh Raman. Succinct ordinal trees with level-ancestor queries. *ACM Trans. Algorithms*, 2(4):510–534, 2006.

[61] Simon Gog, Timo Beller, Alistair Moffat, and Matthias Petri. From theory to practice: Plug and play with succinct data structures. In *Symposium on Experimenal Algorithms (SEA)*, pages 326–337. Springer, 2014.

[62] Simon Gog and Matthias Petri. Optimized succinct data structures for massive data. *Softw. Pract. Exp.*, 44(11):1287–1314, 2014.

[63] Rodrigo González and Gonzalo Navarro. Rank/select on dynamic compressed sequences and applications. *Theor. Comput. Sci.*, 410(43):4414–4422, 2009.

[64] Michael F Goodchild. Geographical data modeling. *Computers & Geosciences*, 18(4):401–408, 1992.

[65] Roberto Grossi, Ankur Gupta, and Jeffrey Scott Vitter. High-order entropy-compressed text indexes. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms, January 12-14, 2003, Baltimore, Maryland, USA*, pages 841–850. ACM/SIAM, 2003.

[66] Roberto Grossi and Giuseppe Ottaviano. The wavelet trie: maintaining an indexed sequence of strings in compressed space. In Michael Benedikt, Markus Krötzsch, and Maurizio Lenzerini, editors, *Proceedings of the 31st ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2012, Scottsdale, AZ, USA, May 20-24, 2012*, pages 203–214. ACM, 2012.

[67] Ralf Hartmut Güting. An introduction to spatial database systems. *VLDB J.*, 3(4):357–399, 1994.

[68] Ralf Hartmut Güting et al. Gral: An extensible relational database system for geometric applications. In *VLDB*, volume 89, pages 33–44. Citeseer, 1989.

[69] Antonin Guttman. R-trees: A dynamic index structure for spatial searching. *SIGMOD Rec.*, 14(2):47–57, June 1984.

[70] Marios Hadjieleftheriou, Erik Hoel, and Vassilis J. Tsotras. Sail: A spatial index library for efficient application integration. *Geoinformatica*, 9(4):367–389, December 2005.

[71] Meng He, J. Ian Munro, and Srinivasa Rao Satti. Succinct ordinal trees based on tree covering. *ACM Trans. Algorithms*, 8(4):42:1–42:32, 2012.

[72] Xin He, Ming-Yang Kao, and Hsueh-I Lu. A fast general methodology for information-theoretically optimal encodings of graphs. *SIAM Journal on Computing*, 30(3):838–846, 2000.

[73] Stephen J. Hegner and M. Andrea Rodríguez. A model for multigranular data and its integrity. *Informatica, Lith. Acad. Sci.*, 28(1):45–78, 2017.

[74] Stephen J. Hegner and M. Andrea Rodríguez. Implicit representation of bigranular rules for multigranular data. In *DEXA (1)*, volume 11029 of *Lecture Notes in Computer Science*, pages 372–389. Springer, 2018.

[75] Stephen J. Hegner and M. Andrea Rodríguez. Inferences rules for binary predicates in a multigranular database. Technical report, 2018.

[76] Stephen J. Hegner and M. Andrea Rodriguez. Inference rules for binary predicates in a multigranular framework, 2023.

[77] Nadeem Iftikhar. MMDW: A multi-dimensional and multi-granular schema for data warehousing. In Manuel Graña, Carlos Toro, Jorge Posada, Robert J. Howlett, and Lakhmi C. Jain, editors, *Advances in Knowledge-Based and Intelligent Information and Engineering Systems - 16th Annual KES Conference, San Sebastian, Spain, 10-12 September 2012*, volume 243 of *Frontiers in Artificial Intelligence and Applications*, pages 1211–1220. IOS Press, 2012.

[78] Nadeem Iftikhar and Torben Bach Pedersen. Schema design alternatives for multi-granular data warehousing. In *DEXA (2)*, volume 6262 of *Lecture Notes in Computer Science*, pages 111–125. Springer, 2010.

[79] Nadeem Iftikhar and Torben Bach Pedersen. Using a time granularity table for gradual granular data aggregation. In Barbara Catania, Mirjana Ivanovic, and Bernhard Thalheim, editors, *Advances in Databases and Information Systems*

- 14th East European Conference, ADBIS 2010, Novi Sad, Serbia, September 20-24, 2010. Proceedings, volume 6295 of *Lecture Notes in Computer Science*, pages 219–233. Springer, 2010.

[80] W. H. Inmon. *Building the data warehouse*. John Wiley & sons, 2005.

[81] Alexander Irribarra-Cortés, José Fuentes-Sepúlveda, Diego Seco, and Roberto Asín. Speeding up compact planar graphs by using shallower trees. In *2022 Data Compression Conference (DCC)*, pages 282–291. IEEE, 2022.

[82] Guy Jacobson. Space-efficient static trees and graphs. In *30th Annual Symposium on Foundations of Computer Science, Research Triangle Park, North Carolina, USA, 30 October - 1 November 1989*, pages 549–554. IEEE Computer Society, 1989.

[83] Jesper Jansson, Kunihiko Sadakane, and Wing-Kin Sung. Ultra-succinct representation of ordered trees with applications. *J. Comput. Syst. Sci.*, 78(2):619–631, 2012.

[84] Yusaku Kaneta. Fast wavelet tree construction in practice. In *International Symposium on String Processing and Information Retrieval*, pages 218–232. Springer, 2018.

[85] Kenneth Keeler and Jeffery Westbrook. Short encodings of planar graphs and maps. *Discrete Applied Mathematics*, 58(3):239–252, 1995.

[86] Dharmendra Kumar, Divakar Yadav, and Diwakar Singh Yadav. Spatial indexing and searching using parallel wavelet tree. *International Journal of Information Technology*, 15(7):3583–3591, 2023.

[87] YC Lee, ZL Li, and YL Li. Taxonomy of space tessellation. *ISPRS Journal of Photogrammetry and Remote Sensing*, 55(3):139–149, 2000.

[88] Sanjiang Li. On topological consistency and realization. *Constraints An Int. J.*, 11(1):31–51, 2006.

[89] Richard J Lipton and Robert Endre Tarjan. A separator theorem for planar graphs. *SIAM Journal on Applied Mathematics*, 36(2):177–189, 1979.

[90] Hsueh-I Lu and Chia-Chi Yeh. Balanced parentheses strike back. *ACM Trans. Algorithms*, 4(3):28:1–28:13, 2008.

[91] Maria Antonina Mach and Mieczyslaw L. Owoc. Knowledge granularity and representation of knowledge: Towards knowledge grid. In *Intelligent Information Processing V - 6th IFIP TC 12 International Conference, IIP 2010, Manchester, UK, October 13-16, 2010. Proceedings*, volume 340 of *IFIP Advances in Information and Communication Technology*, pages 251–258. Springer, 2010.

[92] Alan K. Mackworth. Consistency in networks of relations. *Artif. Intell.*, 8(1):99–118, 1977.

[93] David J Maguire. The raster gis design model—a profile of erdas. *Computers & Geosciences*, 18(4):463–470, 1992.

[94] Veli Mäkinen and Gonzalo Navarro. Rank and select revisited and extended. *Theor. Comput. Sci.*, 387(3):332–347, 2007.

[95] Christos Makris. Wavelet trees: A survey. *Comput. Sci. Inf. Syst.*, 9(2):585–625, 2012.

[96] Mark McKenney and Markus Schneider. Spatial partition graphs: A graph theoretic model of maps. In Dimitris Papadias, Donghui Zhang, and George Kollios, editors, *Advances in Spatial and Temporal Databases, 10th International Symposium, SSTD 2007, Boston, MA, USA, July 16-18, 2007, Proceedings*, volume 4605 of *Lecture Notes in Computer Science*, pages 167–184. Springer, 2007.

[97] Dinesh P. Mehta and Sartaj Sahni, editors. *Handbook of Data Structures and Applications*. Chapman and Hall/CRC, 2004.

[98] J. I. Munro and P. K. Nicholson. Compressed representations of graphs. In *Encyclopedia of Algorithms*, pages 382–386. Springer, 2016.

[99] J. Ian Munro. Tables. In Vijay Chandru and V. Vinay, editors, *Foundations of Software Technology and Theoretical Computer Science, 16th Conference, Hyderabad, India, December 18-20, 1996, Proceedings*, volume 1180 of *Lecture Notes in Computer Science*, pages 37–42. Springer, 1996.

[100] J Ian Munro, Yakov Nekrich, and Jeffrey S Vitter. Fast construction of wavelet trees. *Theoretical Computer Science*, 638:91–97, 2016.

[101] J. Ian Munro, Rajeev Raman, Venkatesh Raman, and Srinivasa Rao S. Succinct representations of permutations and functions. *Theoret. Comput. Sci.*, 438:74–88, 2012.

[102] J. Ian Munro and Venkatesh Raman. Succinct representation of balanced parentheses, static trees and planar graphs. In *38th Annual Symposium on Foundations of Computer Science, FOCS '97, Miami Beach, Florida, USA, October 19-22, 1997*, pages 118–126. IEEE Computer Society, 1997.

[103] J. Ian Munro, Venkatesh Raman, and S. Srinivasa Rao. Space efficient suffix trees. *J. Algorithms*, 39(2):205–222, 2001.

[104] J. Ian Munro and S. Srinivasa Rao. Succinct representations of functions. In Josep Díaz, Juhani Karhumäki, Arto Lepistö, and Donald Sannella, editors, *Automata, Languages and Programming: 31st International Colloquium, ICALP*

2004, Turku, Finland, July 12-16, 2004. Proceedings, volume 3142 of *Lecture Notes in Computer Science*, pages 1006–1015. Springer, 2004.

[105] G. Navarro. *Compact Data Structures – A Practical Approach*. Cambridge University Press, 2016.

[106] G. Navarro and K. Sadakane. Fully-functional static and dynamic succinct trees. *ACM Transactions on Algorithms*, 10(3):article 16, 2014.

[107] Gonzalo Navarro. Wavelet trees for all. In Juha Kärkkäinen and Jens Stoye, editors, *Combinatorial Pattern Matching - 23rd Annual Symposium, CPM 2012, Helsinki, Finland, July 3-5, 2012. Proceedings*, volume 7354 of *Lecture Notes in Computer Science*, pages 2–26. Springer, 2012.

[108] Gonzalo Navarro, Yakov Nekrich, and Luís M. S. Russo. Space-efficient data-analysis queries on grids. *Theor. Comput. Sci.*, 482:60–72, 2013.

[109] Gonzalo Navarro and Kunihiko Sadakane. Fully functional static and dynamic succinct trees. *ACM Trans. Algorithms*, 10(3):16:1–16:39, 2014.

[110] Daisuke Okanohara and Kunihiko Sadakane. Practical entropy-compressed rank/select dictionary. In *Proceedings of the Nine Workshop on Algorithm Engineering and Experiments, ALENEX 2007, New Orleans, Louisiana, USA, January 6, 2007*. SIAM, 2007.

[111] Daisuke Okanohara and Kunihiko Sadakane. Practical entropy-compressed rank/select dictionary. In *Proceedings of the Meeting on Algorithm Engineering & Expermiments*, page 60–70, USA, 2007. Society for Industrial and Applied Mathematics.

[112] Rasmus Pagh. Low redundancy in static dictionaries with O(1) worst case lookup time. In Jirí Wiedermann, Peter van Emde Boas, and Mogens Nielsen, editors, *Automata, Languages and Programming, 26th International Colloquium, ICALP'99, Prague, Czech Republic, July 11-15, 1999, Proceedings*, volume 1644 of *Lecture Notes in Computer Science*, pages 595–604. Springer, 1999.

[113] Mihai Patrascu. Succincter. In *49th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2008, October 25-28, 2008, Philadelphia, PA, USA*, pages 305–313. IEEE Computer Society, 2008.

[114] Zdzislaw Pawlak. Rough sets. *International Journal of Parallel Programming*, 11(5):341–356, 1982.

[115] Zdzislaw Pawlak. *Rough sets - theoretical aspects of reasoning about data*, volume 9 of *Theory and decision library : series D*. Kluwer, 1991.

[116] Donna J Peuquet. A conceptual framework and comparison of spatial data models. *Cartographica: The International Journal for Geographic Information and Geovisualization*, 21(4):66–113, 1984.

[117] R. & Tripathy B. K. Raghavan. On some topological properties of multigranular rough sets. 2(3):536–543, 2011.

[118] Raghu Ramakrishnan and Jeffrey D. Ullman. A survey of deductive database systems. *J. Log. Program.*, 23(2):125–149, 1995.

[119] Rajeev Raman, Venkatesh Raman, and S. Srinivasa Rao. Succinct indexable dictionaries with applications to encoding k-ary trees and multisets. In David Eppstein, editor, *Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms, January 6-8, 2002, San Francisco, CA, USA*, pages 233–242. ACM/SIAM, 2002.

[120] Rajeev Raman, Venkatesh Raman, and Srinivasa Rao Satti. Succinct indexable dictionaries with applications to encoding ¡i¿k¡/i¿-ary trees, prefix sums and multisets. *ACM Trans. Algorithms*, 3(4):43–es, November 2007.

[121] Rajeev Raman and S. Srinivasa Rao. Succinct representations of ordinal trees. In Andrej Brodnik, Alejandro López-Ortiz, Venkatesh Raman, and Alfredo Viola, editors, *Space-Efficient Data Structures, Streams, and Algorithms - Papers in Honor of J. Ian Munro on the Occasion of His 66th Birthday*, volume 8066 of *Lecture Notes in Computer Science*, pages 319–332. Springer, 2013.

[122] Kunihiko Sadakane. New text indexing functionalities of the compressed suffix arrays. *J. Algorithms*, 48(2):294–313, 2003.

[123] Hanan Samet. Bibliography on quadtrees and related hierarchical data structures. In Laurens R. A. Kessener, Frans J. Peters, and Marloes L. P. van Lierop, editors, *Data Structures for Raster Graphics*, pages 181–201, Berlin, Heidelberg, 1986. Springer Berlin Heidelberg.

[124] José Fuentes Sepúlveda. *Parallel Construction of Succinct Data Structures*. PhD thesis, Universidad de concepción Concepción, Chile, 2016.

[125] Shashi Shekhar, Mark Coyle, Brajesh Goyal, Duen-Ren Liu, and Shyamsundar Sarkar. Data models in geographic information systems. *Communications of the ACM*, 40(4):103–111, 1997.

[126] Fernando Silva-Coira, José R. Paramá, Susana Ladra, Juan R. López, and Gilberto Gutiérrez. Efficient processing of raster and vector data. *PLOS ONE*, 15(1):1–35, 01 2020.

[127] Shashank Srivastav, Pradeep Kumar Singh, and Divakar Yadav. A method to improve exact matching results in compressed text using parallel wavelet tree. *Scalable Computing: Practice and Experience*, 22(4):387–400, 2021.

[128] German Tischler. On wavelet tree construction. In Raffaele Giancarlo and Giovanni Manzini, editors, *Combinatorial Pattern Matching - 22nd Annual Symposium, CPM 2011, Palermo, Italy, June 27-29, 2011. Proceedings*, volume 6661 of *Lecture Notes in Computer Science*, pages 208–218. Springer, 2011.

[129] B. K. Tripathy and Anirban Mitra. Some topological properties of rough sets and their applications. *Int. J. Granul. Comput. Rough Sets Intell. Syst.*, 1(4):355–369, 2010.

[130] G. Turán. On the succinct representation of graphs. *Discrete Applied Mathematics*, 8(3):289 – 294, 1984.

[131] William Thomas Tutte. A census of planar maps. *Canadian Journal of Mathematics*, 15:249–271, 1963.

[132] Niko Välimäki and Veli Mäkinen. Space-efficient algorithms for document retrieval. In Bin Ma and Kaizhong Zhang, editors, *Combinatorial Pattern Matching, 18th Annual Symposium, CPM 2007, London, Canada, July 9-11, 2007, Proceedings*, volume 4580 of *Lecture Notes in Computer Science*, pages 205–215. Springer, 2007.

[133] Sheng Wang and Dayou Liu. Spatio-temporal database with multi-granularities. In *Advances in Web-Age Information Management WAIM*, volume 3129 of *Lecture Notes in Computer Science*, pages 137–146. Springer, 2004.

[134] Xiaoyang Sean Wang, Sushil Jajodia, and V. S. Subrahmanian. Temporal modules: An approach toward federated temporal databases. *Inf. Sci.*, 82(1-2):103–128, 1995.

[135] Michael F. Worboys. Imprecision in finite resolution spatial data. *GeoInformatica*, 2(3):257–279, 1998.

[136] Michael F Worboys and Matt Duckham. *GIS: a computing perspective*. CRC press, 2004.

[137] Arun Kumar Yadav, Divakar Yadav, Akhilesh Verma, Mohd Akbar, and Kartikey Tewari. Scalable thread based index construction using wavelet tree. *Multimedia Tools and Applications*, 82(9):14037–14053, 2023.

[138] Katsuhisa Yamanaka and Shin-ichi Nakano. A compact encoding of plane triangulations with efficient query supports. *Information Processing Letters*, 110(18-19):803–809, 2010.

[139] Mihalis Yannakakis. Embedding planar graphs in four pages. *Journal of Computer and System Sciences*, 38(1):36–67, 1989.