Compact Representations of Spatial Hierarchical Structures with Support for Topological Queries*

José Fuentes-Sepúlveda[a,d], Diego Gatica[a,d], Gonzalo Navarro[b,d], M. Andrea Rodríguez[a,d], Diego Seco[c,d]

[a]*Department of Computer Science, Universidad de Concepción, Chile.*
[b]*Department of Computer Science, University of Chile, Chile.*
[c]*CITIC, Facultade de Informática, Universidade da Coruña, Spain.*
[d]*Millennium Institute for Foundational Research on Data, Chile.*

Abstract

Among different spatial data models, the topological model for spatial regions explicitly represents common boundaries. This model pursues the efficiency of topology-related queries and the elimination of data redundancy. This paper proposes 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, such as the political and administrative partition of a country. In the context of these hierarchies, we focus on queries that search for inclusion, disjointness, and adjacency between regions. The proposed structures build upon compact planar graph embeddings, which show to have a good trade-off between space and time.

Keywords: Multi-granular hierarchy, Topological model, Spatial partition, Compact data structures

2020 MSC: 68P05, 68P30

*This work was funded by: ANID Millennium Science Initiative Program - Code ICN17_002; PAI grant 77190038 (1st author); PFCHA/Doctorado Nacional/2020-21201986 (2nd author); FONDECYT Grant 1-200038 (3rd author); CYTED grant 519RT0579 (4th and 5th authors); GRC: ED431C 2021/53, partially funded by GAIN/Xunta de Galicia (5th author); TED2021-129245B-C21 (PLAGEMIS), PDC2021-120917-C21 (SIGTRANS), PID2020-114635RB-I00 (EXTRACompact) and PID2019-105221RB-C41 (MAGIST), partially funded by MCIN/AEI/10.13039/501100011033 and "NextGenerationEU"/PRTR (5th author). An early partial version of this paper appeared in *Proc. DCC 2021* [1].

1. Introduction

Spatial modeling usually distinguishes between *field-* and *entity-based* views of the space [2, 3]. While the field-based view of the space associates attributes with areas in the space, the entity-based view represents the space as spatial objects with explicit identities. The latter is the most common approach to represent objects in spatial databases, and thus there exist several spatial data models implementing it. Within these spatial data models, the topological model [4, 5] explicitly represents common boundaries between entities. Basic elements of this model are points, nodes, arcs, and regions. An arc is composed of its extreme nodes (or points of intersection), a sequence of points between those nodes, and the two regions that share the arc. By using the concept of arcs, the representation of a common boundary is unique, eliminating duplication when representing two adjacent entities. Such a model is useful to answer topological queries that search for objects that are adjacent or disjoint. Adjacency and disjointness are the most useful topological relations when the space is *partitioned* into regions, that is, it is divided into disjoint regions whose geometric union make up the whole space.

Partitions are a central notion for the spatial domain [6]. They can be also formalized as a form of granularity composed of granules that cannot partially overlap. The notion of granularity refers to the level of detail used in the representation of a domain composed of identifiable and disjoint units [7, 8]. For example, the spatial granularity of 'county' contains all counties that are spatial regions that do not overlap. The definition of granularity we use in this work generalizes spatial partitions, but we analyze the relation between these two concepts in Section 2. Partitions can be organized in terms of a partial order relation by the topological relation of inclusion (inside or part-of) creating spatial hierarchical structures such as the administrative subdivisions of a country, which are useful to associate spatial references with statistical or non spatial data in traditional databases, and also to define dimensions in data warehousing systems. Such a structure captures different levels of detail or granularity in the representation of space.

The work in this paper proposes 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. Our approach differs from classical indexing structures that optimize spatial queries [9, 10, 11], 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 new structures are based on Compact Data Structures (CDSs) [12], which have proven to successfully represent different data types in small space while supporting rich sets of operations. CDSs have obtained remarkable results in domains where there is a need to handle data in devices whose capacity is surpassed by the data volume. The sheer volume of data is known to be one of the main characteristics of the spatial domain, and hence, CDSs have been also successfully applied in this domain [13, 14, 15, 16]. With this approach we expect not only to handle large volume of spatial data, but also to support spatial algorithms on small devices such as sensors, wearables or smartphones.

In this regard, the work by [17] showed how to implement the topological spatial data model using a planar-graph compact structure, which is based on Turán's representation [18]. 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. Recently, the work by [1] introduced a compact data structure to support the same relations. Our work includes the following contributions: i) we propose two new approaches that offer good space-time trade-offs, ii) we show how to generalize our approaches to the case where the maps are composed by more than one connected component, and iii) we perform a comprehensive experimental evaluation of the three approaches.

Our three approaches use planar graphs to represent partitions of the space, which allows us to use well-studied properties and methods from graph theory [19, 20]. The main difference between our approaches is in how they represent inclusion relationships between regions at different granularities. Both the representation of these relationships and the planar graphs use compact data structures, which allows performing most of the work in main memory, resorting to secondary memory only to solve operations about the actual geometries. Our strategies then complement classical spatial indexing methods.

The organization of the paper is as follows. Section 2 describes related work and preliminaries including syntax notation used along the paper. Sections 3, 4 and 5 introduce the three proposed data structures. Then, Section 6 generalizes those data structures to domains composed by more than one connected component. Section 7 experimentally evaluates our structures. Final conclusions and research directions are in Section 8.

2. Related work and preliminaries

2.1. Multi-granular hierarchies

The definition of spatial granularity [8] comes from the definition of temporal granularity by [21]. Formally, the spatial granularity is a function that maps non-overlapping portions, referred as granules of the spatial domain, into indexes or identifiers. [22] defined a spatio-temporal granule as a tuple (s, t), meaning that at time index t, the spatial index s is valid. [23] 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 [21, 8, 22, 24]. 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 [6, 25, 26]. Concepts of maps, resolution, spatial objects and topological reasoning build on partitions and their properties. [27] proposed a formalization based on the theory of rough sets [28] 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.

[29] propose 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.

2.2. Multi-granular spatial hierarchy

Given a geographic connected region R,[1] the formalization of a multi-granular spatial hierarchy is as follows. A partition $L = \{r_1, \dots r_n\}$ is a granularity com-

[1]In Section 6, we extend our proposal for regions that are not necessarily connected (e.g., islands).

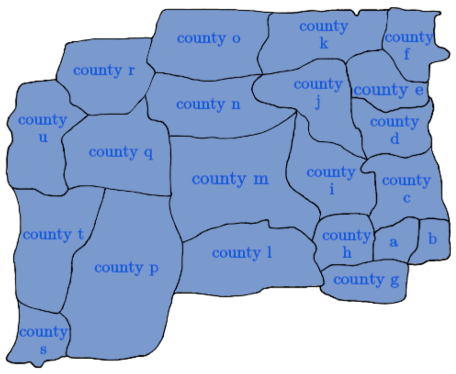
posed of regions r_i (called granules), such that (i) $\forall r_i, r_j \in L, r_i \cap r_j = \emptyset$ (i.e., regions are disjoint or touch each other, but they do not internally intersect) and (ii) $R = \bigcup_1^n r_i$ (i.e., the geometric union of regions makes the whole R). We will say that regions 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 $L_1 = \{r_{1,1}, \dots r_{1,n_1}\}$ and $L_2 = \{r_{2,1}, \dots r_{2,n_2}\}$ 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 granularity and L_h the finest or highest level of granularity. Figure 1 shows a spatial hierarchy composed of three granularity levels: L_1 is the region level (Figure 1(c)), L_2 is the state level (Figure 1(b)), and L_3 is the county level (Figure 1(a)), so $L_1 \prec L_2 \prec L_3$.

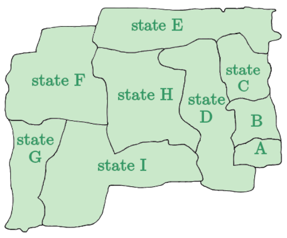
Based on this definition of a partition and of the multi-granular hierarchy, the following properties hold.

- 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 $\mathsf{contains}(r, r')$.

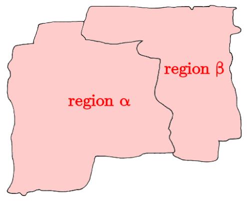
- Because a partition L_i can be represented as a planar graph, with $n_i = |L_i|$ nodes and m_i edges (the number of pairs of neighboring regions in L_i), it holds $n_i < m_i \leq 3n_i - 6$.



(a) Geographic division at county level.



(b) Geographic division at state level.



(c) Geographic division at region level.

Figure 1: Example of a geographic division with aggregation levels Region, State, and County.

- 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.

2.3. Topological queries

As in the work of [17], we restrict our scope to *pure topological* queries and to the static version of the model. We then leave out the queries that use geometric information such as, given the coordinates of a point, find the granule it belongs to. A simple information-theoretic argument shows that basic queries that refer

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 1: Topological operations considered in [17, 30]. Let r_1 and r_2 be two regions and e be a boundary edge.

to geometry cannot be answered without using a linear number of *integers* per granule. Several well-known spatial data structures fitting in this space, like R-trees [9] and Quadtrees [10], can efficiently answer geometry queries like spatial range, spatial join, and nearest neighbor queries. These indexes were especially designed to answer spatial point or range queries, but they are inefficient to solve pure topological queries such as overlapping, touching, and inclusion. If we restrict the queries to the topological domain, instead, it is possible to represent the data within a linear number of *bits*, and furthermore, efficiently solve many queries; see Table 1. Those queries, however, do not consider multi-granular models, just a single space partition.

In this work we explore the space and time complexities that can be achieved on pure topological queries over multi-granular models. The set of queries we consider is based on the international standard ISO/IEC 13249-3:2016 [31]. Most of those operations are also implemented in flagship spatial databases, such as PostgreSQL.[2]. In addition to those already considered in Table 1, we handle operations that relate regions (i.e., granules) of different level of granularity. In particular, contains(r_1, r_2) (i.e., do region r_1 contains region r_2?), touches(r_1, r_2) (i.e., do regions r_1 and r_2 share a common boundary?),

[2]`http://postgis.net/docs/Topology.html`

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, $n_i = \lvert L_i \rvert$
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 2: Notations.

and $\mathsf{contained}(L_j, r)$ (i.e., list all regions at granularity level L_j that are contained in region r). As example, consider again Figure 1. Then, relation $\mathsf{contains}(\mathit{state}\ H, \mathit{county}\ n)$ is true, whereas relation $\mathsf{contains}(\mathit{state}\ C, \mathit{county}\ o)$ is false. Further, $\mathsf{contained}(L_3, \mathit{region}\ \alpha)$ returns the counties $\{m, n, r, q, u, t, s, p, l\}$.

Using the notation summarized in Table 2, Table 3 lists the operations we consider and the complexities we obtain. We analyze the space in general and also under the realistic *exponential growing* assumption, that there is a constant $c > 1$ such that $n_i \geq c \cdot n_{i-1}$ for all i. This assumption implies that the number of regions grows exponentially with the level and thus $h = O(\lg n_h)$. The assumption holds, for example, if every region is split into at least two regions in the next level (thus $c = 2$).

2.4. Compact data structures

With the main purpose of manipulating huge amounts of data, compact data structures [12] aim to represent data in space close to its information-theoretic lower bound. Unlike compression techniques, where decompression is needed to support operations, compact data structures allow us to implement operations directly over the compact representation. Through this work we use compact data structures for sequences and ordinal trees.

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

Table 3: 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.

Bitmaps. A bitmap $B[1..n]$ is an array of bits supporting three operations: access(B, i) (the bit in B at position i), rank$_b(B, i)$ (the number of occurrences of bit $b \in \{0, 1\}$ in B up to position i), and select$_b(B, i)$ (the position of the i-th appearance of bit b in B). One can support all those operations in $O(1)$ time using $n + o(n)$ bits [32]. When B has $m \ll n$ 1s, it can be represented within $m \lg \frac{n}{m} + O(m + n/\lg^c n)$ bits, for any constant c, still solving the operations $O(c)$ time [33].

Compact sequences. Compact sequences are well-known compact data structures with a myriad of applications, ranging from text indexing to planar maps. Given a sequence $A[1..n]$, where $A[i] \in \Sigma$, interesting operations are the same access(A, i), rank$_a(A, i)$, and select$_a(A, i)$, which extend those of bitmaps for any $a \in \Sigma$. The sequence can be represented in $nH_0 + o(n \lg \sigma \lg \lg n/\lg n)$ bits, supporting the operations in time $O(\lg \sigma/\lg \lg n)$ [34]. Here, H_0 is the zero-order empirical entropy of A, $H_0 = \sum_{a \in \Sigma} \frac{n_a}{n} \lg \frac{n}{n_a}$, where a appears n_a times in A. Note the time is constant if $\sigma \in O(\text{polylog } n)$. Using the basic operations, more complex ones can be implemented, such as rightmost$_a(A, i) =$ select$_a(A, \text{rank}_a(A, i))$, the position of the rightmost symbol a up to position

i, and $\mathsf{leftmost}_a(A, i) = \mathsf{select}_a(A, \mathsf{rank}_a(A, i) + 1)$, the position of the leftmost symbol a after position i.

Generalizations of rank and select, $\mathsf{rank}_{\leq a}(A, i)$ (the number of occurrences of symbols less than or equal to a in A up to position i), $\mathsf{select}_{\leq a}(A, i)$ (the position of the i-th symbol less than or equal to a in A), and $\mathsf{leftmost}_{\leq a}(A, i)$ (the position of the leftmost symbol less than or equal to a after position i), can be supported in time $O(\lg \sigma)$, $O(\lg n \lg \sigma)$ and $O(\lg \sigma)$, respectively, using wavelet trees [35].

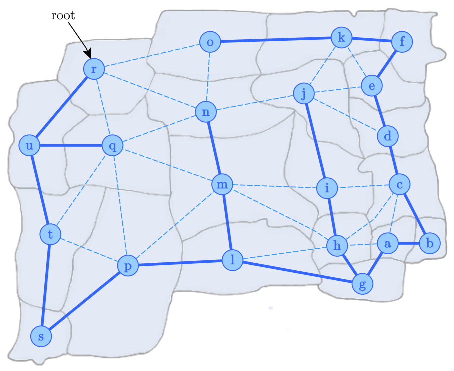
Compact trees. The topology of a tree with n nodes can be represented by a balanced parenthesis sequence of length $2n$. The sequence is obtained by performing a DFS traversal of the tree, writing an open parenthesis every time an edge is visited for the first time, and a close parenthesis when an edge is visited for the second time. Given a balanced parentheses sequence $B[1..2n]$, the operation $\mathsf{find\_close}(B, i)$ returns the position of the matching closing parenthesis of the opening parenthesis $B[i]$, and $\mathsf{find\_open}(B, i)$ returns the position of the matching opening parenthesis of $B[i]$. Operation $\mathsf{enclose}(B, i)$ returns the position of the opening parenthesis that, together with its matching closing parenthesis, most tightly contains i. Those operations are supported in constant time using $2n + o(n)$ bits [36].

The most practical compact representation for ordinal trees is the *range min-max tree* (RMMT) [36]. 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 $\mathsf{excess}(i) = \mathsf{excess}(i - 1) + 1$ if $B[i] =$ '(', or $\mathsf{excess}(i) = \mathsf{excess}(i - 1) - 1$ 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

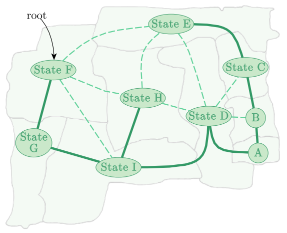
$v.e = \mathsf{excess}(e) - \mathsf{excess}(s-1)$, and the minimum excess value of the leaf, defined as $v.min = \mathsf{min}_{i \in [s,e]}\{\mathsf{excess}(i) - \mathsf{excess}(s-1)\}$, are stored. For an internal node u with left child u_l and right child u_r, the whole and minimum excess values are also stored, defined as $u.e = u_l.e + u_r.e$ and $u.min = \mathsf{min}\{u_l.min, u_l.e + u_r.min\}$, respectively. Once those values are stored in the RMMT, the operations $\mathsf{find\_open}$, $\mathsf{find\_close}$ and $\mathsf{enclose}$ are reduced to the primitive operations $\mathsf{fwd\_search}(B, i, d)$ (the leftmost position $j > i$ in B, such that $\mathsf{excess}(i) + d = \mathsf{excess}(j)$, with $d < 0$) and $\mathsf{bwd\_search}(B, i, d)$ (the rigthmost position $j < i$ in B, such that $\mathsf{excess}(i) + d = \mathsf{excess}(j)$, with $d < 0$). 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 $l = \Theta(\lg n)$ in this paper, so the time is $O(\lg n)$ 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 [36].

Within the same time and space complexities, the RMMT can also support $\mathsf{rank}_{(}(B, i)$ and $\mathsf{select}_{(}(B, i)$ 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 $u.n' = u_l.n' + u_r.n'$.

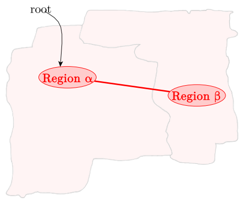
All previous operations can be applied to a sequence $S[1..n]$ composed by two intertwined balanced parenthesis sequences, B and B^*. For convenience, B is represented with parentheses, B^* with brackets (for the remainder of the paper, parentheses refers to round brackets (), while brackets refers to 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 $\mathsf{rank}_{()}(S, i)$ (the number of opening or closing parentheses in S up to position i) is supported in constant time by $\mathsf{rank}_1(A, i)$. Similarly, $\mathsf{select}_{()}(S, i)$ (the position of the i-th opening or closing parenthesis in S) is supported in constant time by $\mathsf{select}_1(A, i)$. In the same way, for $S[i] =$ ')', $\mathsf{find\_open}(S, i)$ is mapped to $\mathsf{select}_1(A, \mathsf{find\_open}(B, \mathsf{rank}_1(A, i)))$. Operations $\mathsf{find\_close}$ and $\mathsf{enclose}$ are sup-



(a) Planar graph at county level.



(b) Planar graph at state level.



(c) Planar graph at region level.

Figure 2: Planar graph representations of the aggregation levels of Figure 1. Spanning trees are represented with thick edges.

ported similarly.

2.5. *Compact representation of topology data*

We focus on the planar graph embedding representation of a geographic area divided into regions whose interiors do not overlap. The embedding is composed by nodes representing the geographic regions and edges connecting two regions that share a geographic boundary. Figure 2 shows the induced planar embedding of the geographic area of Figure 1.

Although there exist various compact representations of planar graph embeddings [37, 12], the representation of [18] is one of the simplest. It consists of a sequence S of length $2m$, where m is the number of edges of the planar embedding. For its construction, Turán's representation performs a depth-first search (DFS) traversal over an arbitrary spanning of the planar embedding, appending to S a '(' or a ')' depending on whether it is the first or second time that an edge of the spanning tree is visited. For edges that do not belong to the spanning tree, a '[' or a ']' is appended following the same conditions. By using two bits per symbol of S, the representation uses $4m$ bits of space. For example, the embedding of Figure 2(a) can be represented by sequence S_3 in Figure 3.

Although Turan's representation does not provide primitives to navigate the graph, [38] augmented it using compact representations of trees and bitmaps that add up $o(m)$ extra bits, and enable navigation operations. 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 $\omega(\lg m)$. Later, [17, 30] 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 1. Hereinafter, we refer to their work as PEMB.

In this work, we generalize 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, $\lceil n_i \lg n_{i-1} \rceil$ bits at each level $i > 1$. In what follows, we introduce three approaches to represent a multi-granular map in less space while efficiently supporting the queries.

3. Approach 1: Mapping via bitmaps

This section summarizes the results of [1], which we complement with a new proof of correctness of the construction method, and a tighter time/space complexity analysis for the construction algorithm and for the contained operation (see Table 3). Instead of building PEMB independently for each planar embedding of the collection, they 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 3.2 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 shows an example of spanning trees following Definition 1. From the spanning tree of Figure 2(a) we can induce the spanning trees of Figures 2(b) and 2(c).

3.1. Structure

Each granularity level L_i is stored in two components (see Figure 3): *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

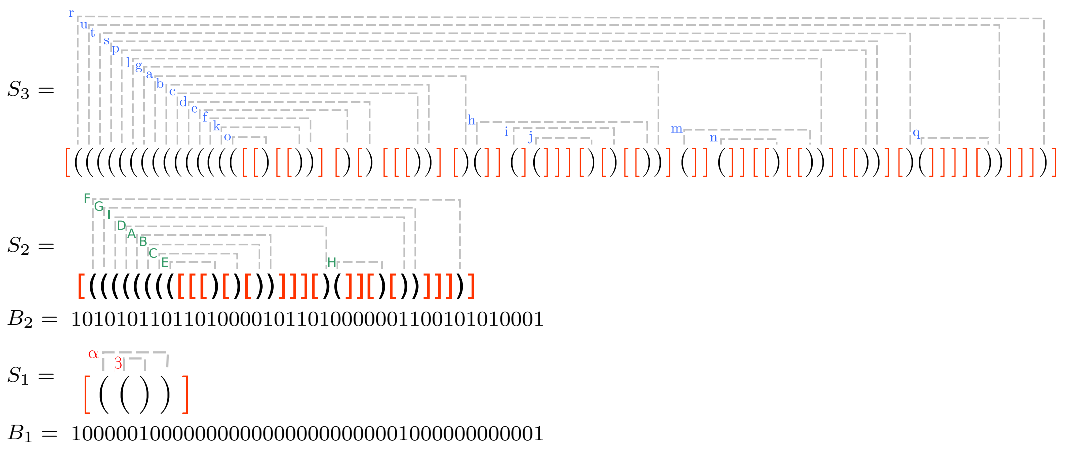

$S_3 =$
r
u
t
s
p
l
g
a
b
c
d
e
f
k
o
h
i
j
m
n
q
$S_2 =$
F
G
I
D
A
B
C
E
H
[(((((((((([[[)[)[))]]][)(]][)[))]]]))]
$B_2 =$ 101010110110100001011010000001100101010001
$S_1 =$
α
β
[(())]
$B_1 =$ 100000100000000000000000000000100000000001

Figure 3: Compact representation of the geographic division of Figure 2.

Section 3.2, 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 $2n_i \lg \frac{n_h}{n_i} + O(n_i) + o(n_h)$ bits; recall Section 2.4.

Overall, the space of this representation is $4m + o(m) + 2\sum_i n_i \lg \frac{n_h}{n_i} + O(n) + o(hn_h) = 2\sum_i n_i \lg \frac{n_h}{n_i} + O(n) + o(hn_h)$ bits. Since $\lg \frac{n_h}{n_i} \le \lg \frac{n}{n_i}$ and, by Jensen's inequality, $\sum_i n_i \lg \frac{n}{n_i} \le n \lg h$, 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 $n_i \le n_h/c^{h-i}$, $\sum_i n_i \lg \frac{n_h}{n_i} \le \sum_i n_h(h-i)/c^{h-i} \cdot \lg c = O(n_h)$. In turn, the $o(hn_h)$ term can be $O(hn_h/\lg n_h) = O(n_h)$ [33].

3.2. 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.

In the traversal, each edge of the planar embedding of L_h is processed twice[3] 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:

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.

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)

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

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.

3.3. Operations

In order to support the operations of Table 3, we provide the following primitive operations to navigate the compact representation, based on the operations described in Section 2.

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.

- $\mathsf{go\_up\_L_h}(x, i)$: 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 $z = \mathsf{select}_{(}(S_i, x)$, to then map such position into the bitmap B_i, with $y = \mathsf{select}_1(B_i, \mathsf{rank}_{()}(S_i, z))$. 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 $\mathsf{select}_{()}(S_h, y)$. 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 $z = \mathsf{select}_{(}(S_h, x)$, to then map it to the bitmap B_{h-d} with $p = \mathsf{rank}_{()}(S_h, z)$. 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 $q = \mathsf{select}_{()}(S_{h-d}, rank_1(B_{h-d}, p))$. If $S_{h-d}[q]$ = '(', then q

is the answer, otherwise it is $q' = \mathsf{enclose}(S_{h-d}, \mathsf{find\_open}(S_{h-d}, q))$. 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 $\mathsf{rank}_{(}(S_i, x)$.

- $\mathsf{go\_level}(x, i, j)$: 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 3. 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.

- $\mathsf{contained}(L_j, r_1)$: *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 $a = \mathsf{go\_level}(r_1, i, j)$ and $b = \mathsf{find\_close}(S_j, a)$. To report the regions, we traverse the range left-to-right reporting every region $\mathsf{region\_id}(S_j, a')$, where initially $a' = a$ and then it is redefined as the position of the next open parenthesis, $a' = \mathsf{leftmost}_{(}(S_j, a')$, 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 $a' = \mathsf{leftmost}_{(}(S_j, \mathsf{find\_close}(S_j, a'))$. An opening parenthesis at position p is marked if $B_i[c] = 1$, where $c = \mathsf{select}_1(B_j, \mathsf{rank}_{()}(S_j, p))$. 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 7. 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* $\mathsf{contains}(r_1, r_2)$ *in constant 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 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.*

Sections 4 and 5 introduce two new approaches that provide trade-offs for the work of [1]. In particular, the approach of Section 4 improves the space consumption, both in practice (as we show in Section 7) and in theory by a sub-linear term, at the cost of increasing the running time by a factor of $O(\lg n \lg h)$, meanwhile the approach of Section 5 reduces in practice the running time of the operations at the cost of increasing space consumption.

4. 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.

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

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

$B_{()}$= 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 4: Sequences $S_3^{()}$, $S_3^{[]}$, $B_{()}$ and $B_{[]}$ for the sequence S_3 of Figure 3.

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 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.

4.1. 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 $\mathsf{select}_{\leq j}(B_{()}, i)$. 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 4 shows the sequences $S_3^{()}$, $S_3^{[]}$, $B_{()}$, and $B_{[]}$ corresponding to the sequence S_3 of Figure 3.

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

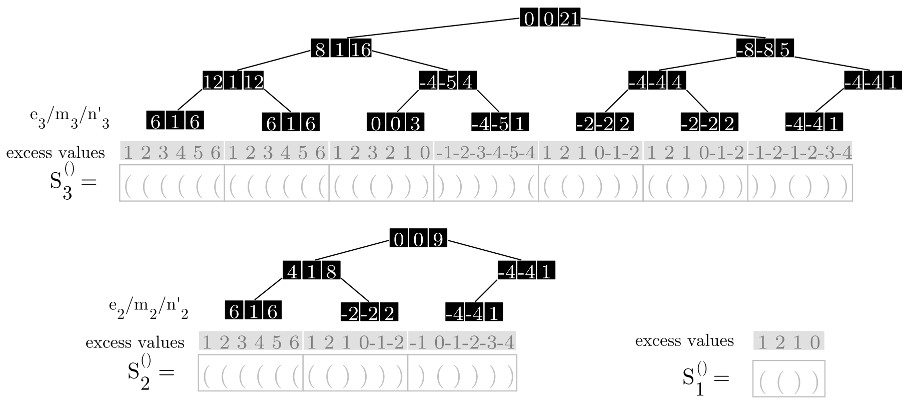


Figure 5: RMMTs of balanced parenthesis sequences $S_1^{()}$, $S_2^{()}$ and $S_3^{()}$. A node of the RMMT covering the block $S_i^{()}[i..j]$ stores: the last excess value of the block (e_i), the minimum excess value of block (m_i), and the number of opening parentheses in the block(n'_i). The values and parentheses in gray are not explicitly stored.

$S_1^{[]}, S_2^{[]}, \ldots, S_h^{[]}$. Summing up the $2h$ RMMTs, the space usage is $O(m)$ bits. See Figure 5 for an example of the RMMTs of Figure 3.

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

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 $\sum_{i\in[1..h]} \frac{n_i}{n} \lg \frac{n}{n_i}$. Similarly, the entropy $H_0(B_{[]})$ of $B_{[]}$ is at most $\sum_{i\in[1..h]} \frac{m_i}{m} \lg \frac{m}{m_i}$. 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 3.1, both entropies are $O(1)$. Both $B_{()}$ and $B_{[]}$ can then be stored in space $O(m(H_0(B_{()}) + H_0(B_{[]}))) + o(n_h \lg h) + O(h \lg n) = O(n) + o(n \lg h)$ bits [39], and the space $o(n \lg h)$ can be $O(n \lg h / \lg n) = O(n)$ [33]. Thus, the total space is $O(n)$ bits.

We can traverse the implicit sequences $S_1^{()}, \dots, S_{h-1}^{()}$ by performing $\mathsf{select}_{\leq j}$ and $\mathsf{rank}_{\leq j}$ operations over the sequence $B_{()}$. The i-th parenthesis of S_j is obtained in $O(\lg n_h \lg h)$ time as $\mathsf{select}_{\leq j}(B_{()}, i)$, and the number of parentheses of S_j in the range $S_h^{()}[1..i]$ is obtained in $O(\lg h)$ time as $\mathsf{rank}_{\leq j}(B_{()}, i)$. Similarly, operations $\mathsf{find\_open}(S_j^{()}, i)$, $\mathsf{find\_close}(S_j^{()}, i)$ and $\mathsf{enclose}(S_j^{()}, i)$ are supported in $O(\lg n_h \lg h + \lg n_h) = O(\lg n_h \lg h)$ time, where the term $\lg n_h \lg h$ 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 $\lg n_h$ comes from the up/down traversal of the RMMT.

4.2. 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 $O(\lg n_h \lg h)$.

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

- $\mathsf{go\_down\_L_h}(x, d)$: The answer is the parenthesis 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 $p = \mathsf{select}_{(}(S_h^{()}, x)$. Then, $q = \mathsf{rank}_{\leq h-d}(B_{()}, p)$ 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., $S_h^{()}[\mathsf{select}_{\leq h-d}(B_{()}, q)] =$ '('), the answer is q. Otherwise, the answer is its closest ancestor, at position $q = \mathsf{enclose}(S_{h-d}^{()}, \mathsf{find\_open}(S_{h-d}^{()}, q))$.

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

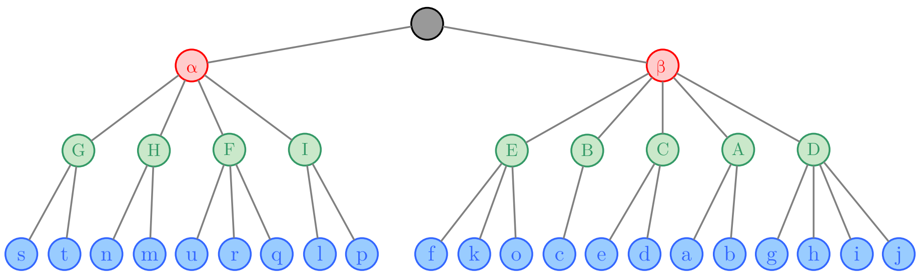
The operation $\mathsf{go\_level}$ is implemented just as in Section 3, $\mathsf{go\_level}(x, i, j) = \mathsf{go\_down\_L_h}(\mathsf{go\_up\_L_h}(x, i), h - j)$, with time complexity $O(\lg n_h \lg h)$.

The implementation of the main operations contains, touches, and contained follows the same steps of their counterparts in Section 3, reaching complexities $O(\lg n_h \lg h)$, $O(d_{r_2} \lg n_h \lg h)$, and $O(\lg n_h \lg h)$ 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.

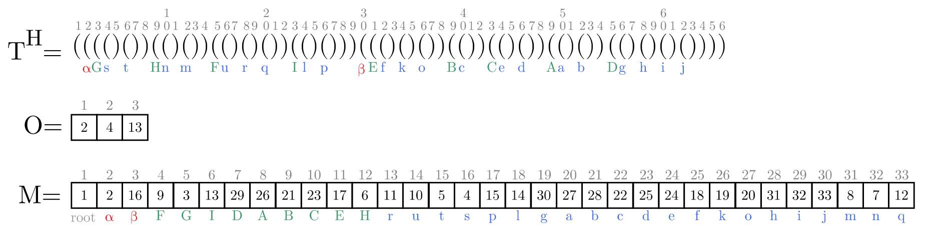
The following theorem summarizes the results of this approach:

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* $\mathsf{contains}(r_1, r_2)$ *in $O(\lg n \lg h)$ time,* $\mathsf{touches}(r_1, r_2)$ *in $O(min(d_{r_1}, d_{r_2}) \lg n \lg h)$ time, and* $\mathsf{contained}(L_j, r_1)$ *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 $B_{[]}$.



(a) Hierarchy tree H representing the topological hierarchy of Figure 2. The root of the tree is a dummy node.



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

O=

1	2	3
2	4	13

M=

	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 6(a).

Figure 6: Components to store the topological hierarchy in the third representation.

5. 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 3. 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 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 6(a) shows the tree H for the topological hierarchy of Figure 2.

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 pre-order 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 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 6(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 $(1 + \epsilon)n \lg n + O(n)$ bits, with a representation that also computes $M^{-1}(j)$, that is, where in M is the value j, in time $O(1/\epsilon)$ [40]. The total space is then $O(n \lg n)$ bits.

5.1. 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 $r'_1 \leq r'_2 \leq$ find_close(r'_1). The operation contains then takes $O(1)$ time.

- touches(r_1, r_2): This is built on top of operation contains as in Section 3. The time complexity is then $O(d_{r_2})$.

- 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 $p = \mathsf{select}_((T^H, r'_1)$ 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 $p' = \mathsf{fwd\_search}(T^H, p, j-i)$. Then, we keep reporting the region to the right of p' with its same depth, up to $p' > q$, by computing $p' = \mathsf{level\_next}(p') = \mathsf{fwd\_search}(T^H, close(p'), 1)$ [12, p. 270]. For every position p' to report, we return its region id with $M^{-1}(\mathsf{rank}_((T^H, p')) - O[j]$. 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 $O(n \lg h)$ bits. The same representation supports operations* contains(r_1, r_2) *in $O(1)$ time,* touches(r_1, r_2) *in $O(min(d_{r_1}, d_{r_2})$ time, and* 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.*

6. 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 above and can be implemented as an extension of any of them.

Given a region r at level L_i, composed by $c > 1$ connected components, we treat the connected components as independent regions $r_1, r_2, \ldots, r_c$, increasing

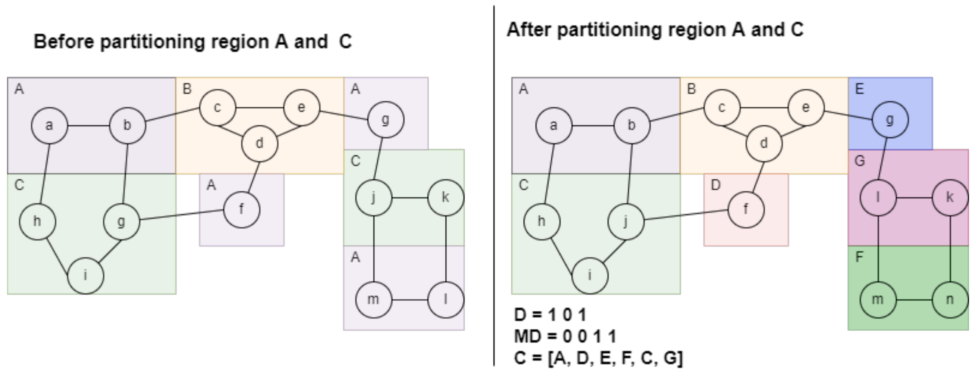

Before partitioning region A and C
After partitioning region A and C
A
B
C
D
E
F
G
a
b
c
d
e
f
g
h
i
j
k
l
m
n
D = 1 0 1
MD = 0 0 1 1
C = [A, D, E, F, C, G]

Figure 7: Example partitioning a region

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 $r_1, r_2, \ldots, r_c$, 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 $r_1, r_2, \ldots, r_c$. 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 $r_1, r_2, \ldots, r_c$ 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 7 shows an example of how regions are partitioned.

Under this setting, operation $\mathsf{contains}(r', r)$, 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 $p = \mathsf{select}_1(MD_j, \mathsf{rank}_1(D_j, r) - 1) + 1$ and $q = \mathsf{select}_1(MD_j, \mathsf{rank}_1(D_j, r))$. 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 3–5.

Similarly, operation $\mathsf{touches}(r', r)$ 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 $\mathsf{contained}(L_j, r)$, 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 $O(n+c \lg c)$ 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).

7. Experimental evaluation

7.1. 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 [41], and compiled with GCC version 4.8.4 and -O3 optimization flag. For the compact planar embeddings, we directly use the code of [38]. We measured running times using the `clock_gettime` function.

7.1.1. Datasets

The datasets used to evaluate our approaches are based on the TIGER dataset,[4] 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 4). With this base information, we generated four datasets, `tiger_8s`, `tiger_usa`, `whole_usa`, and `tiger_usa`$^+$.

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

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.

7.2. Evaluated implementations

Based on the approaches described in Sections 3 to 5 for the representation of the multi-granular maps, we developed the following implementations:

Approach 1 (T)**.** Implementation based on the approach described in Section 3, 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 [42]), *iii)* RRR (an H_0-compressed bitvector [39]).

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

Approach 2 (RMMT). Implementation based on the approach described in Section 4, 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 4, 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 $\mathsf{rank}_{\leq i}$ on each comparison.

Approach 2 (WT-S). Another variant of the approach described in Section 4, 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, 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 [38] 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 $\mathsf{go\_level}(x, i, j)$ 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 $\mathsf{contains}(r_1, r_2)$, $\mathsf{touches}(r_1, r_2)$, and $\mathsf{contained}(L_j, r_1)$, respectively, where r_1 is a region at level L_i and r_2 a region at level L_j.

7.3. 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.[5] 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 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 8 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,

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

yet at the cost of using 80–135 bits per region, that is, about 5–8 times more space.

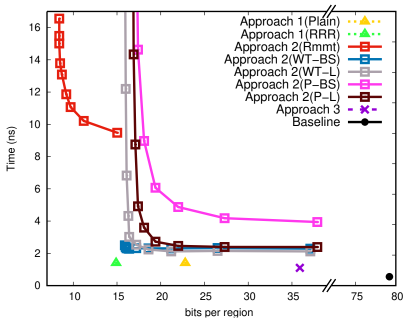
On the synthetic dataset `tiger_usa`$^+$, we use Approach 1 (RRR) instead of Approach 1 (SD), because it saves more space. In this dataset, the least-space variant of Approach 2 (WT-BS) is equally fast and uses slightly less space (indeed, the sweet points of several other variants are pretty close). 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`.

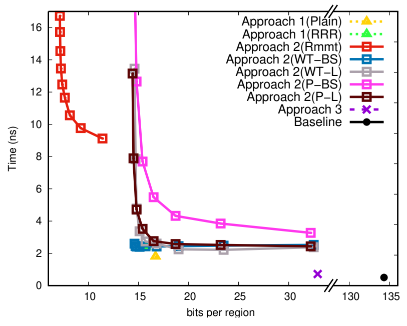
Figures 9 and 10 show the results grouped by distance level, where all valid pairs $(L_i, L_j),\ i \in [1, 6-c],\ j = i + c$ are grouped into the distance level c. For Approach 2 we only maintain the variants Approach 2 (rmmt) with block length $l = 2^9$ and Approach 2 (WT-BS) with $l = 2^{15}$. For 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 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.

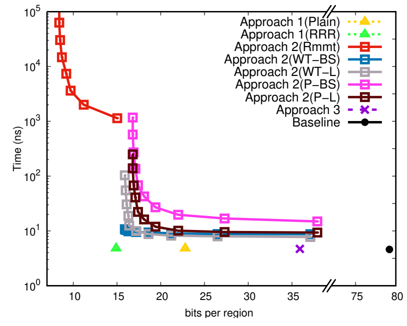
Similar results can be observed for the dataset `tiger_usa`$^+$, except that Approach 3 becomes the second fastest on the operation contained. This owes to the way this dataset was constructed: its hierarchy tree is wider and has more nodes than the hierarchy tree of `tiger_usa`. Approach 3 is more cache-friendly



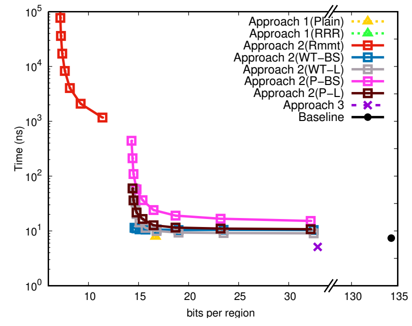
(a) Operation **contains** dataset tiger_usa.



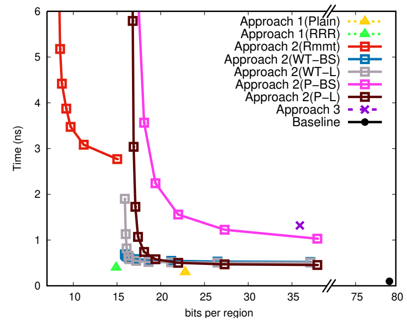
(b) Operation **contains** dataset tiger_usa$^+$.



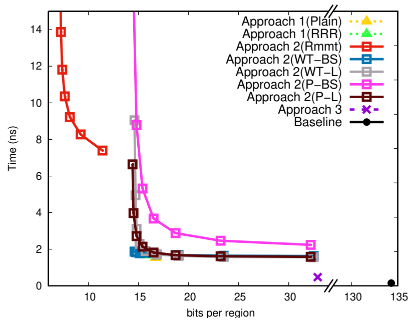
(c) Operation **touches** dataset tiger_usa.



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



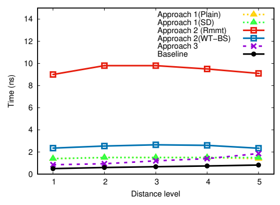
(e) Operation **contained** dataset tiger_usa.



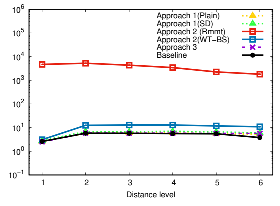
(f) Operation **contained** dataset tiger_usa$^+$.

Figure 8: Running time in nanoseconds using the datasets `tiger_usa` and `tiger_usa`$^+$.

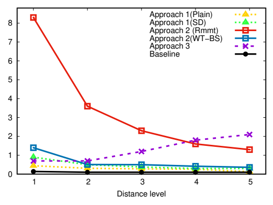
when reporting many nearby regions.



(a) Operation contains.

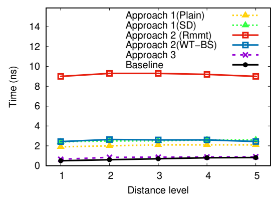


(b) Operation touches.

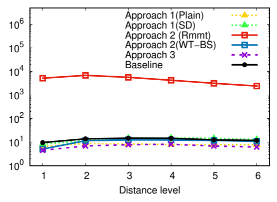


(c) Operation contained.

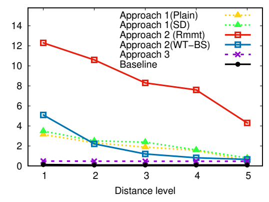
Figure 9: Running time in nanoseconds using the dataset `tiger_usa`, where distance level corresponds to the distance between the levels of the queried granules.



(a) Operation contains.



(b) Operation touches.



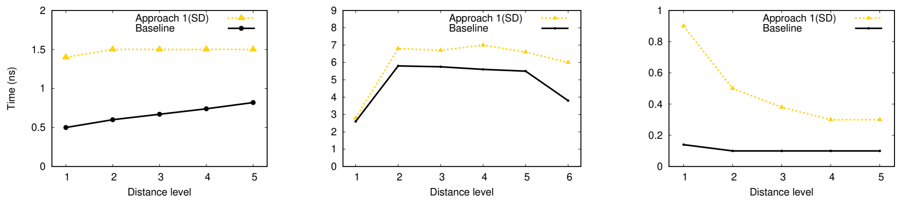
(c) Operation contained.

Figure 10: Running time in nanoseconds using the dataset `tiger_usa`$^+$.

7.4. Performance with non-connected components

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 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 6 adds 43,009 new regions obtained from the partition of regions with more than one connected



(a) Operation contains. (b) Operation touches. (c) Operation contained.

Figure 11: Running time in nanoseconds using the dataset `whole_usa`.

component, which induces less than 1% of extra space.

Figure 11 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 9.

8. Conclusions and Future Work

We have focused on the problem 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 a set of 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 $O(n \lg h)$ 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 design 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 show that we can 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 answer all queries within 10 nanoseconds per retrieved elements, and in some cases less than half a nanosecond.

A challenge for future work is to obtain better theoretical complexities for the operations. Despite the good times obtained in practice, operation (2) requires time proportional to the number of neighbors of one of the regions, for example. Another line of future work is to expand the set of operations. Postgresql, for example, implements eight named spatial relationship predicates defined in the standard OGC SFS, and three non-standard relationship predicates. Some of them do not apply in our domain given the restrictions of a spatial partition, in which the interior of the granules cannot intersect (see Section 2.2). This is the case of *overlaps*, for example. Some others, such as *equals* and *disjoint* can be easily implemented with the operations provided in our work. In some domains, the *contains* predicate has a variant named *containsProperly* or *includes*, which returns true when a region contains another and there is no intersection in the boundary of such regions.

Data and codes availability statement

The data and codes that support the findings of this study are available at `https://figshare.com/s/2d0d3f0825666c9c595c`.

References

[1] J. Fuentes-Sepúlveda, D. Gatica, G. Navarro, M. A. Rodríguez, D. Seco, Compact representation of spatial hierarchies and topological relationships, in: 31st Data Compression Conference, DCC 2021, IEEE, 2021, pp. 113–122.

[2] H. Couclelis, People manipulate objects (but cultivate fields): Beyond the raster-vecter debate in gis, in: Theories and Methods of Spatio-Temporal

Reasoning in Geographic Space. LNCS vol. 639, Springer-Verlag, 1992, pp. 65–77.

[3] S. Shekhar, M. Coyle, B. Goyal, D.-R. Liu, S. Sarkar, Data models in geographic information systems, Comm. ACM 40 (4) (1997) 103–111.

[4] B. Kuijpers, J. Paredaens, J. V. den Bussche, Lossless representation of topological spatial data, in: SSD, 1995, pp. 1–13.

[5] Y. Deng, P. Z. Revesz, Spatial and topological data models, in: Information Modeling in the New Millennium, Idea Group, 2001, pp. 345–359.

[6] M. Erwig, M. Schneider, Partition and conquer, in: COSIT, Vol. 1329, Springer, 1997, pp. 389–407.

[7] C. Bettini, X. Wang, S. Jajodia, A general framework for time granularity and its application to temporal reasoning, Ann. Math. Art. Intell. 22 (1998) 29–58.

[8] S. Wang, D. Liu, Spatio-temporal database with multi-granularities, in: WAIM, Vol. 3129, Springer, 2004, pp. 137–146.

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

[10] H. Samet, Bibliography on quadtrees and related hierarchical data structures, in: Data Structures for Raster Graphics, 1986, pp. 181–201.

[11] M. Hadjieleftheriou, E. Hoel, V. J. Tsotras, Sail: A spatial index library for efficient application integration, Geoinformatica 9 (4) (2005) 367–389.

[12] G. Navarro, Compact Data Structures – A Practical Approach, Camb. U. Press, 2016.

[13] N. R. Brisaboa, A. Cerdeira-Pena, G. de Bernardo, G. Navarro, O. Pedreira, Extending general compact querieable representations to GIS applications, Inf. Sci. 506 (2020) 196–216.

[14] N. R. Brisaboa, M. R. Luaces, G. Navarro, D. Seco, Space-efficient representations of rectangle datasets supporting orthogonal range querying, Inf. Syst. 38 (5) (2013) 635–655.

[15] N. R. Brisaboa, A. Gómez-Brandón, G. Navarro, J. R. Paramá, Gract: A grammar-based compressed index for trajectory data, Inf. Sci. 483 (2019) 106–135.

[16] N. R. Brisaboa, A. Fariña, D. Galaktionov, M. A. Rodríguez, A compact representation for trips over networks built on self-indexes, Inf. Syst. 78 (2018) 1–22.

[17] J. Fuentes-Sepúlveda, G. Navarro, D. Seco, Implementing the topological model succinctly, in: SPIRE, 2019, pp. 499–512.

[18] G. Turán, On the succinct representation of graphs, Discr. Appl. Math. 8 (3) (1984) 289 – 294.

[19] Z. Chen, M. Grigni, C. H. Papadimitriou, Planar map graphs, in: STOC, 1998, pp. 514–523.

[20] M. J. Alam, M. Kaufmann, S. G. Kobourov, T. Mchedlidze, Fitting planar graphs on planar maps, J. Graph Algorithms Appl. 19 (1) (2015) 413–440.

[21] C. Bettini, C. E. Dyreson, W. S. Evans, R. T. Snodgrass, X. S. Wang, A glossary of time granularity concepts, in: Temporal Databases, Dagstuhl, 1997, pp. 406–413.

[22] E. Camossi, M. Bertolotto, E. Bertino, A multigranular object-oriented framework supporting spatio-temporal granularity conversions, Int. J. Geo. Inf. Sci. 20 (5) (2006) 511–534.

[23] A. Belussi, C. Combi, G. Pozzani, Formal and conceptual modeling of spatio-temporal granularities, in: IDEAS, 2009, pp. 275–283.

[24] M. A. Mach, M. L. Owoc, Knowledge granularity and representation of knowledge: Towards knowledge grid, in: IIP, Vol. 340, 2010, pp. 251–258.

[25] M. McKenney, M. Schneider, Spatial partition graphs: A graph theoretic model of maps, in: SSTD, Vol. 4605, Springer, 2007, pp. 167–184.

[26] M. P. Dube, M. J. Egenhofer, Partitions to improve spatial reasoning, in: SIGSPATIAL PhD, ACM, 2014, pp. 1:1–1:5.

[27] M. F. Worboys, Imprecision in finite resolution spatial data, GeoInformatica 2 (3) (1998) 257–279.

[28] Z. Pawlak, Rough sets, J. Parallel Program. 11 (5) (1982) 341–356.

[29] T. Bittner, B. Smith, A taxonomy of granular partitions, in: COSIT, Vol. 2205, Springer, 2001, pp. 28–43.

[30] J. Fuentes-Sepúlveda, G. Navarro, D. Seco, Implementing the topological model in near-optimal space and time, CoRR abs/1911.09498.

[31] ISO/IEC 13249-3:2016. Information technology – Database languages – SQL multimedia and application packages – Part 3: Spatial, Tech. rep. (2016).

[32] J. I. Munro, Tables, in: Proc. 16th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS), LNCS 1180, 1996, pp. 37–42.

[33] M. Pătraşcu, Succincter, in: Proc. 49th Annual IEEE Symposium on Foundations of Computer Science (FOCS), 2008, pp. 305–313.

[34] P. Ferragina, G. Manzini, V. Mäkinen, G. Navarro, Compressed representations of sequences and full-text indexes, ACM Transactions on Algorithms 3 (2) (2007) article 20.

[35] G. Navarro, Wavelet trees for all, Journal of Discrete Algorithms 25 (2014) 2–20.

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

[37] J. I. Munro, P. K. Nicholson, Compressed representations of graphs, in: Encyclopedia of Algorithms, Springer, 2016, pp. 382–386.

[38] L. Ferres, J. Fuentes-Sepúlveda, T. Gagie, M. He, G. Navarro, Fast and compact planar embeddings, Comput. Geom. 89 (2020) 101630.

[39] R. Raman, V. Raman, S. R. Satti, Succinct indexable dictionaries with applications to encoding k-ary trees, prefix sums and multisets, ACM Trans. Algorithms 3 (4) (2007) 43–es.

[40] J. I. Munro, R. Raman, V. Raman, S. R. S., Succinct representations of permutations and functions, Theoret. Comput. Sci. 438 (2012) 74–88.

[41] S. Gog, T. Beller, A. Moffat, M. Petri, From theory to practice: Plug and play with succinct data structures, in: SEA, 2014, pp. 326–337.

[42] D. Okanohara, K. Sadakane, Practical entropy-compressed rank/select dictionary, in: ALENEX, SIAM, 2007.