Compact Representation of Spatial Hierarchies and Topological Relationships

José Fuentes-Sepúlveda[1], Diego Gatica[1,3], Gonzalo Navarro[2,3], M. Andrea Rodríguez[1,3] and Diego Seco[1,3]

[1]Department of Computer Science, Universidad de Concepción, Chile
[2]Department of Computer Science, University of Chile, Chile
[3]Millennium Institute for Foundational Research on Data, Chile

Abstract

The topological model for spatial objects identifies common boundaries between regions, explicitly storing adjacency relations, which not only improves the efficiency of topology-related queries, but also provides advantages such as avoiding data duplication and facilitating data consistency. Recently, a compact representation of the topological model based on planar graph embeddings was proposed. In this article, we provide an elegant generalization of such a representation to support hierarchies of vector objects, which better fits the multi-granular nature of spatial data, such as the political and administrative partition of a country. This representation adds a small space on top of the succinct base representation of each granularity, while efficiently answering new topology-related queries between objects not necessarily at the same level of granularity.

Introduction

An object-oriented model for spatial objects is the topological model [1, 2], where common boundaries between regions are identified to avoid duplication and to facilitate data consistency, and where the queries of interest are topological in nature such as "What provinces are adjacent to or inside of a particular region?". Topological databases consist of a finite set of labeled points, curves, and areas, and where points are typically associated with coordinates in the Euclidean plane. Although much research has focused on indexing structures to optimize spatial queries [3–5], to the best of our knowledge not much work addresses the optimization of queries in topological data models. Given the sheer volume of data in the spatial domain, we approach this problem with Compact Data Structures (CDSs) [6].

A recent work [7] presents a planar-graph compact data structure to support a topological data model. We built upon this structure by extending it to account for answering inclusion, disjoint, and adjacency topological queries in a multi-granular context of spatial hierarchical structures, such as the political and administrative partition of a country. Granularity defines units that quantitatively measure data with respect to the dimensions of the domain they represent [8, 9]. Multiple granularities can be organized in a partial order structure, such as the political subdivisions of a country, which are useful to associate spatial references with non spatial data in traditional databases, and also to define dimensions in data warehousing systems.

This work was funded by ANID: Millennium Science Initiative Program - Code ICN17_002; also by PAI grant 77190038 (1st author), PFCHA/Doctorado Nacional/2020-21201986 (2nd author), FONDECYT Grant 1-200038 (3rd author), and CYTED grant 519RT0579 (4th and 5th authors).

Our structure starts with the geometric representation of regions as a partition of the space and extracts common boundaries, which become edges of a planar graph. Using planar graphs for spatial information [10, 11] allows us to use properties and strategies well-studied in graph theory. In addition, our structure represents each granularity and the inclusion relationships between regions at different levels of granularity using CDSs. This approach is complementary with the storage and indexing of geometries in larger memories (usually secondary memory). However, with our approach most of the work is done in main memory using the compact topological index, resorting to the geometric indexes stored in secondary memory only when necessary. Our solution is limited to the case of regions composed by contiguous sub-regions.

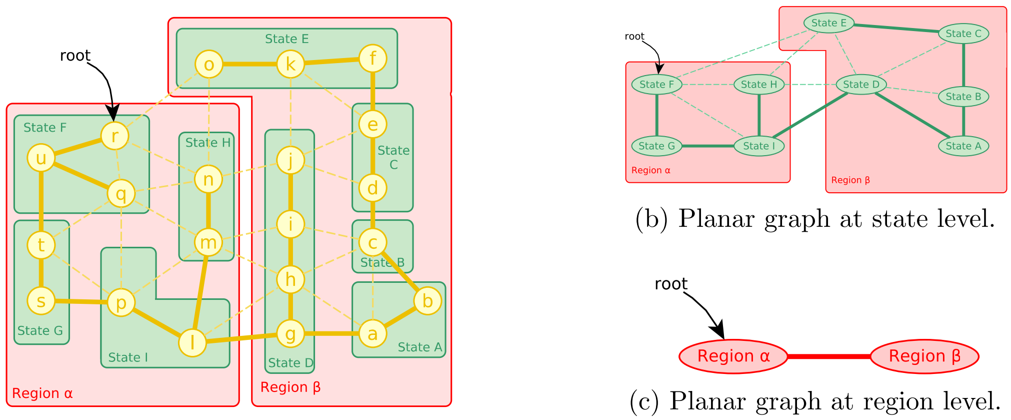
Notation. Let R be a geographic area that can be divided into regions $r_1, r_2, \ldots$, such that all regions are disjoint and their union is R. The area R can be recursively divided h times. All the regions obtained after i recursive divisions are part of the *i-th granularity level,* L_i. The highest granularity level, L_h, is composed of indivisible regions and the lowest granularity level, L_0, corresponds to R. We represent by n_i the number of regions at granularity level L_i, and by d_r the number of regions sharing a boundary with region r. Relation $\mathsf{contains}(x, y)$ is true if the geographic boundaries of region y are completely contained inside the boundaries of region x, with $x \in L_i$, $y \in L_j$, and $i \leq j$. Thus, given a region $y \in L_j$, there always exists a *unique* region $x \in L_i$ for which $\mathsf{contains}(x, y)$ is true, for any $i \in [0..j]$. For example, Figure 1a shows an area divided into three granularity levels: region level (L_1), state level (L_2) and county level (L_3). From the figures, relation $\mathsf{contains}(\textit{state } H, \textit{county } n)$ is true, whereas relation $\mathsf{contains}(\textit{state } C, \textit{county } o)$ is false.

Preliminaries. Our solution rests on well-studied compact data structures [6]. Through this work we use bitmaps, $A[1..n]$, where interesting operations are $\mathsf{rank}_c(A, i)$ (the number of occurrences of symbol c in A up to position i) and $\mathsf{select}_c(A, i)$ (the position of the i-th appearance of symbol c in A), both supported in constant time using $n + o(n)$ bits. Using them, more complex operations can be implemented, such as $\mathsf{rightmost}_c(A, i) = \mathsf{select}_c(A, \mathsf{rank}_c(A, i))$, the position of the rightmost symbol c before position i, and $\mathsf{leftmost}_c(A, i) = \mathsf{select}_c(A, \mathsf{rank}_c(A, i) + 1)$, the position of the leftmost symbol c after position i.

Compact trees can be represented as a sequence of balanced parentheses, performing a DFS traversal over the tree, writing an open parenthesis for each forward edge, and a close parenthesis for each backward edge. Given a balanced parentheses sequence $B[1..2n']$ representing the topology of a tree with n' nodes, 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 $2n' + o(n')$ extra bits.

Related work

The notion of granularity in the spatio-temporal domain defines the units that quantitatively measure data with respect to the dimensions of the domain they represent.



(a) Planar graph at county level.

(b) Planar graph at state level.

(c) Planar graph at region level.

Figure 1: Geographic division with aggregation levels: Region, State and County, and their planar graph representation. Spanning trees are represented with thick edges.

Temporal granularity was defined by [12], and later [9] defined spatial granularity as a mapping function from a domain of indexes to portions of a space, called spatial granules. One relevant property of the granules of a granularity is that they do not overlap with each other. Associated with the concept of granularity, previous works also define relations between granularities, which allows us to characterize structures that organize the domain. One of them is the notion of spatial partition, meaning that 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 elements, which do not overlap, of S [9, 12–14].

Regarding efficient processing, there exists a large number of proposed data structures for spatial data, which typically address spatial range, spatial join, and nearest neighbor queries. Among them, two classical structures are the R-tree [3] and the Quadtree [4], which when applied to spatial objects assume classical vector (or Spaghetti) spatial representation of objects. These indexes can also solve topological queries such as overlap, inclusion, and disjoint relationships, but such approach is computationally expensive. Unlike the classical vector representation, a topological spatial representation is tailored for topological queries.

In this work, we focus on the planar graph embedding representation of a geographic area divided into regions whose interiors do not overlap, i.e., a granularity. The planar graph embedding induced by a geographic area is composed by nodes representing geographic regions and edges representing two regions sharing a geographic boundary. See examples of induced planar embeddings in Figures 1a-1c. Among all compact representations of planar-graph embeddings [6, 15], Turán's representation [16] stands out for its simplicity. The representation is built in two stages. First, an arbitrary spanning tree T of the planar embedding is computed. Second, a DFS traversal is performed, computing a sequence S of length $2m$, where m is the number of edges of the planar embedding. During the traversal, an edge in T is represented either by '(' or ')', depending on whether it is the first or second time that the edge is visited. Similarly, an edge not in T is represented by '[' or ']'. Using two bits per symbol of S, the representation uses $4m$ bits of space.

The main drawback of this representation is that it does not provide primitives to navigate the graph. To overcome such limitation, Ferres *et al.* [17] augmented Turán's representation with $o(m)$ extra bits, using compact representations of trees and bitmaps. The new representation lists the incident edges of a vertex in $O(1)$ time per edge and the edges bounding a face in $O(1)$ time per edge, finds the degree of a vertex in $O(f(m))$ time for any $f(m) \in \omega(1)$, and determines whether two vertices are neighbors in $O(f(m))$ time for any $f(m) \in \omega(\log m)$. Later, Fuentes-Sepúlveda *et al.* [7] improved the bounds of the operations and showed how to extend the representation to provide the first theoretical compact representation of the topological model. In this work, we show how to generalize such a representation of the topological model to support multi-granular hierarchies of spatial objects.

Data structure

A map with several levels of granularity can be seen as a collection of planar embeddings, one for each level, and the relations among levels. To obtain a compact representation, the data structure in [7] can be used for the collection of planar embeddings. The main remaining challenge is to store a mapping among the regions at different levels in order to support queries based on the relation $\mathsf{contains}(\cdot,\cdot)$. For that, we propose a new approach to compute the spanning tree needed for the compact representation in [7], instead of the *arbitrary* spanning tree that they use. In the next section we show how to compute a more suitable spanning tree, whose topology implicitly encodes the mapping among consecutive granularity levels. Our representation encodes each granularity level using the following components (see Figure 2):

1. The planar graph embedding of each level is represented by the compact data structure of Fuentes-Sepúlveda *et al.* [7]. Across the h granularity levels, the space consumption of this component is $4\sum_{i=1}^{h} m_i + o(\sum_{i=1}^{h} m_i)$ bits, where m_i is the number of edges in the planar graph embedding of level L_i.

2. A bitmap $B_i[1..2n_h]$ with $\mathsf{rank}/\mathsf{select}$ support, where n_h is the number of regions in the highest granularity L_h. The bitmap B_i is used to mark some vertices of the balanced-parentheses representation S_h of the spanning tree at granularity level L_h. Such a spanning tree is already stored in the compact planar graph embedding of level L_h. By default, all entries of B_i are 0-bits. During the DFS traversal of the spanning tree, if the k-th visited vertex is the first vertex for which relation $\mathsf{contains}(x, k{-}th\ vertex\ of\ L_h)$ is true, where x is a region at granularity L_i, then $B_i[p] = 1$ and $B_i[q] = 1$, where p and q are the positions of the opening and closing parentheses representing vertex k in S_h. For each region x at granularity L_i, only one region y of L_h fulfills the condition. The total space is $2n_h(h-1)$ bits for the bitmaps B_i, and $o(hn_h)$ bits for the $\mathsf{rank}/\mathsf{select}$ data structures over them, with $1 \leq i \leq h-1$ (no bitmap is needed at level h).

Construction. To construct our representation, we adapt the classical DFS traversal algorithm based on a stack. In the traditional algorithm, an edge (u, v) is traversed when the target vertex v has not been visited before. In the adapted version, the DFS

Figure 2: Compact representation of the geographic division of Figures 1a-1c.

traversal is performed on the planar graph of the highest granularity level, L_h. Now, an edge (u, v) is traversed when the target vertex v and the regions containing it at lower levels L_i, $i < h$, have not been visited before, or when the target vertex v has not been visited and both vertices, u and v, belong to the same region at granularity level L_i, $i < h$. We say that a region r at level L_i, $i < h$, has not been visited when none of the regions contained in r have been visited.

During the DFS traversal of the planar embedding at level L_h, when traversing the j-th edge e with target region r for the first time,[1] we check the region r' containing r at level L_i, with $i < h$. If r' has not been visited yet, we mark it as visited and set $B_i[j] = 1$. Setting the j-th bit of B_i indicates that the target region of the j-th edge e of the traversal is the entering point to the region r' at granularity level L_i. Notice that the same edge e will be traversed again as the k-th edge of the DFS traversal. We set $B_i[k] = 1$ to indicate that the source region of the k-th traversed edge is the exiting point of the region r', i.e., we complete the traversal of the whole region.

Each time the algorithm processes an edge e, two conditions must be checked: 1) if the edge e meets the condition to be traversed, and 2) if it is the first time that the edge e is processed. Depending on such conditions, the algorithm appends (,), [or] to the output sequence S_h. After this, the algorithm checks for each level L_i, with $i < h$, if the edge e at level i meets the conditions to append the same symbol previously appended to S_h to the sequence S_i. Given an edge $e = (p, q)$ at level L_h, and the regions p' and q' containing regions p and q at level L_i, respectively, the conditions to check for each symbol are the following:

1. If region q' has not been visited yet and it is different from region p', then a (is appended to S_i.

2. If region q' has been already visited and it is the first time that the pair of regions (p', q') is processed, then a [is appended to S_i.

3. If it is the second time that the pair (p', q') is processed, then a) or] is appended to S_i, depending on if its matching symbol is a (or [, respectively.

---

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

The complexity of computing the parenthesis sequence and the bitmap B_i is dominated by the, at most, h comparisons per edge. Since each comparison takes logarithmic time, the complexity of the construction algorithm is $O(n + mh \lg m')$, where m' is the number of edges across all levels, the term $\lg m'$ comes from a dictionary data structure used to store edges processed at level L_i, with $i < h$.

Primitive operations. Before explaining how to support the main operations, we describe the basic primitives that are needed to navigate the hierarchy using the two components described above. These primitives are based on the operations described in the Preliminaries. Henceforth, each vertex in the planar embedding of level L_i is identified by its pre-order rank in the traversal of the spanning tree of the level.

- $\mathsf{go\_up\_L_h}(x, i)$: Find the position in S_h of the first region at level L_h that is contained in the x-th region of level L_i, with $i \leq h$. To support this operation, we use the bitmap B_i, which allows us to map regions of level L_i into regions of level L_h. First, we look for the position of the x-th opening parenthesis in S_i by computing $z = \mathsf{select}_{(}(S_i, x)$. Then, we find the position of the 1-bit that represents the x-th region in B_i by computing $y = \mathsf{select}_1(B_i, \mathsf{rank}_{()}(S_i, z))$. Finally, the position of the output region in S_h is computed by $\mathsf{select}_{()}(S_h, y)$. Since $\mathsf{go\_up\_L_h}(x, i)$ depends on rank/select operations, it takes constant time.
- $\mathsf{go\_down\_L_h}(x, d)$: For the x-th region of level L_h, find the position of the region at level L_{h-d} that contains x. To support this operation, we first compute the position p in B_{h-d} of the x-th region in level L_h, which can be done in constant time as $p = \mathsf{rank}_{()}(S_h, \mathsf{select}_{(}(S_h, x))$. The answer is the position in S_{h-d} of the nearest ancestor y of x that is marked in B_{h-d}. For 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 and the nearest ancestor is $\mathsf{rank}_{(}(S_{h-d}, q)$. Otherwise, the answer is $q' = \mathsf{enclose}(S_{h-d}, \mathsf{find\_open}(S_{h-d}, q))$. Since all operations take constant time, $\mathsf{go\_down\_L_h}(x, d)$ also takes constant time.
- $\mathsf{region\_id}(S_i, x)$: Given the x-th opening parenthesis of S_i, it returns the position or ID of the region represented by such parenthesis. This query can be implemented in $O(1)$ time as $\mathsf{rank}_{(}(S_i, x)$.
- $\mathsf{go\_level}(x, i, j)$: Maps the x-th region of level L_i into the level L_j. This query can be implemented as a composition of the previous queries: First, map the x-th region of level L_i towards level L_h, and then map from level L_h towards level L_j. Thus, this operation can be implemented in $O(1)$ time as $\mathsf{go\_down\_L_h}(\mathsf{go\_up\_L_h}(x, i), h - j)$. Note that if $j < i$, we are going down in the hierarchy, and if $j > i$, we are going up.

Main operations. Given a region r_1 at level L_i, and a region r_2 at level L_j, with $i < j$, we study the following operations:

- $\mathsf{contains}$: *Does region r_1 contain region r_2?* To solve it, we first compute the region z that contains region r_2 at level L_i, which can be done as $z = \mathsf{region\_id}(S_i, \mathsf{go\_level}(r_2, j, i))$. If $r_1 = z$, we return true; otherwise, we return false. If r_1 and r_2 belong to the same level of aggregation, and $r_1 = r_2$, we return true: otherwise, we return false. The time complexity of this query is $O(1)$.

• touches: *Does region r_1 share a boundary with region r_2?* To answer this operation, we check all neighbors of r_2 at level L_j. For each neighbor w of r_2, we compute the region $z = \mathsf{region\_id}(S_i, \mathsf{go\_level}(w, j, i))$ that contains w at level L_i. We distinguish two cases: a) If region r_2 is not contained into region r_1 ($\mathsf{contains}(x, y) =$ false), we must find one neighbor of r_2 that is contained into region r_1. Thus, if $r_1 = z$, we return true. If after checking all neighbors of r_2 we cannot establish that $r_1 = z$, we return false; b) the second case is symmetric. If region r_1 contains region r_2 ($\mathsf{contains}(x, y)$=true), then we must find one neighbor of r_2 that is not contained into r_1. Therefore, if $r_1 \neq z$, we return true. If after checking all neighbors of r_2 we cannot establish that $r_1 \neq z$, we return false. The time complexity of this query is $O(d_{r_2})$, where d_{r_2} is the number of neighbors of region r_2.

• contained: *List all regions at level L_j contained into region r_1.* We compute the range $S_j[a..b]$ that contains all the regions at level j that are contained by the region r_1, where $a = \mathsf{go\_level}(r_1, i, j)$ and $b = \mathsf{find\_close}(S_j, a)$. Then, we traverse the range left-to-right reporting the regions that are contained into r_1. The traversal is performed as follows: 1) The first reported region is $\mathsf{region\_id}(S_j, a)$. Then, we set the position $p = \mathsf{leftmost}_{(}(S_j, a)$, which returns the position of the leftmost open parenthesis after position a. 2) If $p \geq b$, we are done. If not, we check if the opening parenthesis at position p is marked as the beginning of a new region. If it is marked, we set $p = \mathsf{leftmost}_{(}(S_j, \mathsf{find\_close}(S_j, p))$ and repeat point 1. If not, we report $\mathsf{region\_id}(S_j, p)$, set $p = \mathsf{leftmost}_{(}(S_j, p)$ and repeat point 1. To check if the opening parenthesis at position p is marked, we need to compute $c = \mathsf{rank}_{()}(S_j, p)$. If $B_i[c] = 1$, then the parenthesis at position p is marked. The time complexity of this query is $O(n_j)$, where n_j is the number of regions at level L_j that are contained in region r_1.

Experimental evaluation

Experimental setup. The experiments were carried out on a machine with an Intel Core i7 (3820) processor clocked at 3.6 GHz. The machine runs Linux 3.13.0-86-generic, in 64-bit mode. Each core has L1i, L1d and L2 caches of size 32 KB, 32 KB and 256 KB, respectively. The shared L3 cache is of 10 MB. The machine has a 32 GB DDR3 RAM memory, clocked at 1334 MHz. The algorithms were implemented in C++, using the library SDSL [18], and compiled with GCC version 4.8.4 and -O3 optimization flag.[2] We measured the running time using the `clock_gettime` function. For the bitmaps B_i, we develop two implementations, `ours` and `ours_sd`. The first one uses plain bitmaps in all levels, and the second uses a plain bitmap in the L_h level and SD-arrays in the rest. The compact planar embeddings of each granularity level were implemented using the code of Ferres *et al.* [17].

Datasets. To test our data structure, we used the dataset TIGER[3], provided by the U.S. Census Bureau. The TIGER dataset provides geographic and cartographic information of the administrative divisions of the territory in USA. The information is hierarchically organized in granularity levels. For the current work, we generated

[2]Our implementation is available at `https://github.com/Desidia/pemb`

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

two datasets, `tiger_usa` and `tiger_8s`, with the following hierarchy (from lowest, L_1, to highest, L_6, granularity level): State, County, Census tract, Census block group, Census block and Face (see Table 1[4]). The dataset `tiger_8s` corresponds to the eight neighboring states of Nevada, Utah, Arizona, Colorado, New Mexico, Kansas, Oklahoma and Texas, and `tiger_usa` corresponds to the whole continental part of USA.

Dataset	Level	Vertices (n)	Edges (m)
`tiger_usa`	L_1	50	140
	L_2	3,110	9,095
	L_3	72,512	201,631
	L_4	216,243	597,784
	L_5	11,004,160	26,732,935
	L_6	19,735,874	43,837,150
`tiger_8s`	L_1	9	20
	L_2	595	1,730
	L_3	11,626	31,412
	L_4	33,804	91,891
	L_5	2,233,031	5,429,483
	L_6	4,761,354	10,326,904

Table 1: Datasets used in our experiments. Each level includes one node representing the external face of the embedding.

Dataset	Structure	Embedding	Hierarchy
`tiger_usa`	Baseline	50.94	259.68
	`ours`	50.94	37.54
	`ours_sd`	50.94	6.96
`tiger_8s`	Baseline	11.45	55.51
	`ours`	11.45	12.21
	`ours_sd`	11.45	1.63

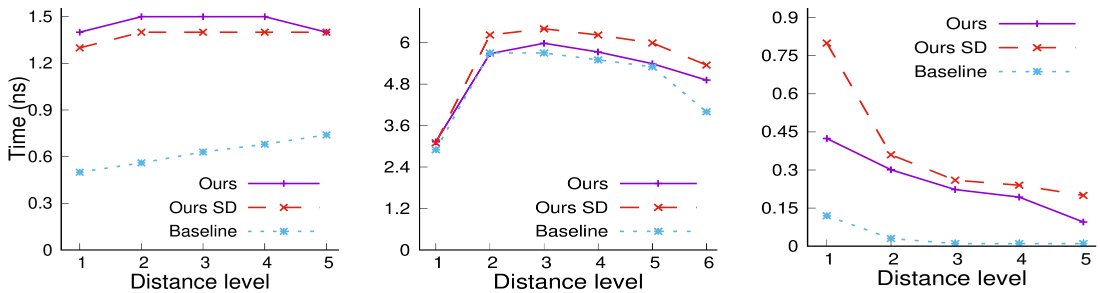
Table 2: Space consumed in MB.

Baseline. We compare our structure against a pointer-based baseline that also implements each level using the compact planar embeddings in [17]. Additionally, the baseline stores a vector for each level $i \in \{0..h-1\}$, in which each position j, representing a region r', stores the index of the region r at level $i+1$ that contains r'. For a region r at level L_i, it also stores pointers to the regions at level L_{i+1} that are contained into r. In this representation, the operation $\mathsf{go\_level}(x, i, j)$ is supported in $O(h)$ time, since in the worst case we must traverse all levels. The main operations were implemented using a logic similar to the one of our structure. Thus, contains, touches, and contained are supported in $O(h)$, $O(d_y h)$ and $O(n_j)$ time, respectively.

Space Usage. Table 2 shows the space consumption of the three approaches. For both datasets, our data structure uses about 29% the space consumed by the baseline, and 19% when using SD-arrays. In terms of bits per region, `ours` and `ours_sd` use 23.9 and 15.7 bits per region resp., for the dataset `tiger_usa`, considering the regions of all the levels, while the baseline uses 84 bits. Similar values were observed for the other datasets. Considering only the space of the hierarchy, `ours` and `ours_sd` use 15% and 3% of the space needed by the baseline, respectively.

Running time. To evaluate the running time of contains and touches, 200 random operations were executed for each pair of aggregation levels (we omit the outer face from the pool of candidates since it has as a large number of neighbors, which may disturb the results). For contained, all possible queries between each pair of aggregation levels were executed. For contains and contained, there are 15 valid pairs $((L_i, L_j), i \in [1,5], j \in [i+1, 6])$, and for touches there are 21 valid pairs $((L_i, L_j), i \in [1,6], j \in [i, 6])$. In total, we executed 3,000 operations of the first

[4] A more detailed description of the datasets is available at `http://www.inf.udec.cl/~jfuentess/datasets/graphs.php`



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

Figure 3: Running time in nanoseconds using the dataset `tiger_usa`.

Granularity level	ours					ours_sd					Baseline				
	L_5	L_4	L_3	L_2	L_1	L_5	L_4	L_3	L_2	L_1	L_5	L_4	L_3	L_2	L_1
L_6	0.21	0.20	0.26	0.20	0.10	0.34	0.25	0.26	0.25	0.24	0.03	0.01	0.01	0.01	0.01
L_5	–	0.59	0.49	0.20	0.18	–	0.85	0.52	0.25	0.24	–	0.10	0.05	0.01	0.02
L_4	–	–	0.53	0.23	0.22	–	–	1.18	0.31	0.29	–	–	0.16	0.02	0.03
L_3	–	–	–	0.36	0.28	–	–	–	0.40	0.36	–	–	–	0.03	0.04
L_2	–	–	–	–	0.43	–	–	–	–	1.60	–	–	–	–	0.29

Table 3: Running time in nanoseconds for dataset `tiger_usa` and operation contained

type, 4,200 operations of the second, and 11,666,872 operations of third type. Each operation was repeated 30 times and the average of those repetitions is reported.

Figure 3 shows the average running time for the three operations using the largest dataset `tiger_usa` (similar results were observed for the other dataset). In the figure, the results were 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. In contained, the running time was divided by the number of regions returned. As expected, our approach is slower than the baseline, but still in the order of nanoseconds. Another important observation is that the use of the SD-array does not significantly increase running times.

Table 3 shows in detail the results of executing contained between all valid pairs for the dataset `tiger_usa`. As in Figure 3c, we report query time per returned region. As this query lists all the regions at a specific level that are contained into the queried region, the number of results per query is drastically affected by the targeted region and level. Each point in the plots of Figure 3c aggregates the values of one diagonal of these tables. As it can be seen in the rows of the tables, as the target level is farther away from the original level, all the structures obtain faster times, which can be explained by the amortization over the number of returned regions.

Conclusions and Future Work

We introduced a compact data structure for multi-granular topological hierarchies, which is based on previous results on compact planar graph embeddings [7]. For a hierarchy of h levels, the proposed structure uses only $4\sum_{i=1}^{h} m_i + 2n_h(h-1) + o(hn_h)$ bits, where n_i and m_i correspond to the number of vertices and edges of the i-th granularity level, respectively. In practice, our representation of the hierarchies use about

20% of the space needed by a baseline implementation, and less than 5% when using compressed bitmaps. In combination with the compact planar graphs, this produces a solution that uses 23.9 bits per region (or 15.7 when using SD-arrays), whereas the baseline uses 84 bits. Regarding running time, as expected, our proposal is in general slower than the baseline, but competitive, specially for operation touches. Another important conclusion is that the use of compressed bitmaps does not drastically increase running time, providing even faster times for operation contains.

Our model assumes that all sub-regions composing a region at higher granularity level are contiguous, which is the most common case (more than 99.7% of the faces are contiguous in our dataset). In the full version, we will show how to adapt our representation to support non-contiguous sub-regions.

References

[1] B. Kuijpers, J. Paredaens, and J. Van den Bussche, "Lossless representation of topological spatial data," in *SSD*, 1995, pp. 1–13.

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

[3] A. Guttman, "R-trees: A dynamic index structure for spatial searching," *SIGMOD Rec.*, vol. 14, no. 2, pp. 47–57, 1984.

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

[5] M. Hadjieleftheriou, E. Hoel, and V. J. Tsotras, "Sail: A spatial index library for efficient application integration," *Geoinformatica*, vol. 9, no. 4, pp. 367–389, 2005.

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

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

[8] C. Bettini, X. Wang, and S. Jajodia, "A general framework for time granularity and its application to temporal reasoning," *Ann. Math. Art. Intell.*, vol. 22, pp. 29–58, 1998.

[9] Sh. Wang and D. Liu, "Spatio-temporal database with multi-granularities," in *WAIM*. 2004, vol. 3129, pp. 137–146, Springer.

[10] Z. Chen, M. Grigni, and Ch. H. Papadimitriou, "Planar map graphs," in *STOC*, 1998, pp. 514–523.

[11] Md. J. Alam, M. Kaufmann, S. G. Kobourov, and T. Mchedlidze, "Fitting planar graphs on planar maps," *J. Graph Algorithms Appl.*, vol. 19, no. 1, pp. 413–440, 2015.

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

[13] E. Camossi, M. Bertolotto, and E. Bertino, "A multigranular object-oriented framework supporting spatio-temporal granularity conversions," *Int. J. Geo. Inf. Sci.*, vol. 20, no. 5, pp. 511–534, 2006.

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

[15] J. I. Munro and P. K. Nicholson, "Compressed representations of graphs," in *Encyclopedia of Algorithms*, pp. 382–386. 2016.

[16] G. Turán, "On the succinct representation of graphs," *Discr. Appl. Math.*, vol. 8, no. 3, pp. 289 – 294, 1984.

[17] L. Ferres, J. Fuentes-Sepúlveda, T. Gagie, M. He, and G. Navarro, "Fast and compact planar embeddings," *Comput. Geom.*, vol. 89, pp. 101630, 2020.

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