Received: Added at production | Revised: Added at production | Accepted: Added at production

DOI: xxx/xxxx

ARTICLE TYPE

Space-efficient data structures for the inference of subsumption and disjointness relations

José Fuentes-Sepúlveda[1,4] | Diego Gatica[1,4] | Gonzalo Navarro[2,4] | M. Andrea Rodríguez[1,4] | Diego Seco[3]

[1]Department of Computer Science, Universidad de Concepción, Chile
[2]Department of Computer Science, University of Chile, Chile
[3]CITIC, Facultade de Informática, Universidade da Coruña, Spain
[4]Millennium Institute for Foundational Research on Data, Chile

Correspondence
Diego Gatica, IMFD and Department of Computer Science, Universidad de Concepción, Edmundo Larenas 219, 4070409, Concepción, Chile. Email: dgatica@udec.cl

Funding Information
This research was supported by the ANID Millennium Science Initiative Program - Code ICN17_002 and PFCHA/Doctorado Nacional/2020-21201986.

Summary

Conventional database systems function as static data repositories, storing vast amounts of facts and offering efficient query processing capabilities. The sheer volume of data these systems store has a direct impact on their scalability, both in terms of storage space and query processing time. Deductive database systems, on the other hand, require far less storage space since they derive new knowledge by applying inference rules. The challenge is how to efficiently obtain the required derivations, compared to having them in explicit form. In this study, we concentrate on a set of predefined inference rules for subsumption and disjointness relations, including their negations. We use compact data structures to store the facts and provide algorithms to support each type of relation, minimizing even further the storage space requirements. Our experimental findings demonstrate the feasibility of this approach, which not only saves space but is often faster than a baseline that uses well-known graph traversal algorithms implemented on top of a traditional adjacency list representation to derive the relations.

KEYWORDS:
deductive database system, inference rule, compact data structure, multigranular data model

1 | INTRODUCTION

Database systems are traditionally seen as passive repositories that offer efficient query processing. A more advanced concept is that of deductive database systems, where new knowledge can be derived from the set of stored *facts*. Deductive databases can be seen as an extension of databases with *rules*[1], which can express recursive queries that form a superset of the relational algebra.

Deductive databases are composed of data (or facts) and rules, which avoid to store unnecessarily large amounts of facts that can be derived via inference rules at query time. There are several other real-world applications that use this combination of facts and inference rules. For instance, in the Semantic Web, OWL (Web Ontology Language) is based on computational logic with rules that represent complex knowledge. It can be used to define taxonomies, i.e., ontologies of things that are organized by containment relations. Popular examples of taxonomies include the Getty Thesaurus of Geographic Names (TGN), with over 1 million places organized hierarchically, and species taxonomies that categorize millions of species. In a related context, a knowledge graph such a Wikidata represents diverse types of binary relations (e.g., instance of, containment, and inside) as triples, where some relations are transitive and reach up millions of triples. Hence, this inference process has been incorporated into flagship implementations of graph databases such as Neo4j and Blazegraph.

A possible drawback of deductive databases is the time needed to derive those omitted facts, as opposed to having them stored directly. The challenge of a deductive database is then to offer competitive query times while using considerably less space than a traditional database. As opposed to derivation strategies for general systems of rules, one may find more efficient solutions to specific systems, which arise in relevant applications. In this work we focus on a set of predefined inference rules concerning subsumption and disjointness relations. The hierarchical subdivision of the geographic space is a typical example to motivate the interest in these particular relations, and it will be used as a running example. The administrative subdivisions have several levels from a coarse level of a country to a finer level of parcels, such as the case of the TIGER dataset composed of more than 11 millions entities and used in our experimental evaluation.

As we mentioned above, subsumption and disjointness relations are important for modeling different types of data. As an example, we use the multigranular data model[2], which is general enough to deal with different domains of application and provides a general axiomatic-based definition of multigranular data. Figure 1 illustrates a granularity structure defined by *coarser/finer than* relations between granularities associated with electoral and administrative sub-divisions in Chile. A granularity, in informal terms, is the level of detail in which data is represented, which is composed of pairwise disjoint granules that map to the domain of representation[3]. In the example, and for the administrative subdivision (right path of the graph in the figure), Chile is composed of granules representing regions, which are the pairwise disjoint union of granules, which are provinces; provinces are a pairwise disjoint union of granules, which are counties. In terms of the electoral subdivisions (left path of the graph), Chile is composed of granules that are senatorial circumscriptions, which are a pairwise disjoint union of electoral districts; electoral districts are composed of pairwise disjoint union of counties; counties are pairwise disjoint union of electoral circumscriptions; and electoral circumscriptions are composed of electoral tables where people finally vote. In this figure, the symbol $\top$ defines a global granularity, which works as an upper limit in the set of granularities. In this example, the granules correspond to subdivisions in the spatial domain[4].

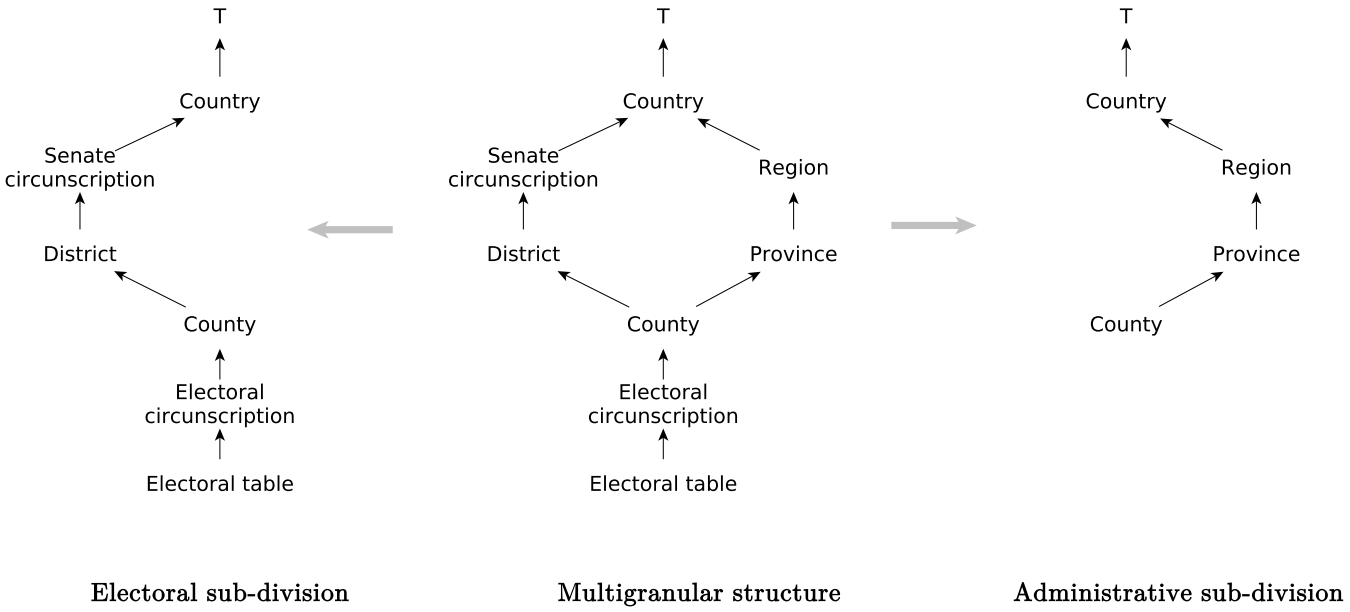


Figure 1 Example of multigranular data: Electoral and administrative sub-divisions in Chile.

Granules are the basic elements of the multigranular data model. They are grouped into granularities, such that granules that belong to a granularity must be disjoint, and every granule of a finer granularity (e.g., a county) must be subsumed by a granule of a coarser granularity (e.g., a region). Granularities are organized into a partial order. The basic relations between granules are then disjointness and subsumption[2], where inference rules can be defined to avoid the explicit representation of relations between every pair of granules.

If we reflect on the workload pattern of the applications mentioned above, we observe two main characteristics: large volumes of data and a primarily static nature (or infrequent modifications). These characteristics are fundamental to justifying the use

of Compact Data Structures (CDS). Note that in applications where some modifications occur, it is feasible to maintain a small dynamic index with those modifications and periodically rebuild the static index. This has been the usual way of working with CDS in other domains such as information retrieval (e.g., web engines).

The work in this paper proposes an efficient implementation, both in terms of space and time, to support such a set of rules. The idea is to explore how new data structures, specifically designed for this domain, can improve the efficiency of deductive reasoning by taking atomic sentences or facts and applying inferences to derive new knowledge. Our strategy to solve rules follows forward chaining, backward chaining, and combinations of them[5]. Forward chaining refers to deriving consequences from given antecedents that are true; backward chaining refers to starting from the relation we want to derive (i.e., the consequent) and searching for the necessary antecedents. A combination of forward and backward strategies arises when there are multiple antecedents that by themselves need to be verified by other rules. In all cases, our proposal uses compact data structures for graphs to represent each type of relation (i.e., subsumption, disjointness, and their negations). These data structures aim to store information in space close to the information theoretic optimum, while supporting a rich set of operations[6]. Our experimental evaluation with large datasets shows the feasibility of this approach: we provide a solution that is not only smaller but usually faster than a baseline that uses well-known graph traversal algorithms implemented on top of a traditional adjacency list representation to derive relations.

The organization of the paper is as follows. Section 2 describes related work on subsumption and disjointness relations, their uses, and inference rules. Section 3 introduces the concepts of granules, basic relations and inferences, as well as the compact data structures that are relevant in the context of this work. Section 4 describes the proposal for an efficient structure, in space and time, to process subsumption and disjointness relations, followed by its experimental evaluation in Section 5. Final conclusions and further research directions are given in Section 6.

2 | RELATED WORK

Subsumption and disjointness are important relations for modeling different types of data. Multigranular data is an example, where granules represent a portion of a domain that must be disjoint and can be organized in terms of a partial order structure by subsumption[2]. Granular data is of particular interest in the spatial domain due to the common organization of spatial objects through inclusion and aggregation, as in the case of a political-administrative subdivision[7]. Beyond the spatial domain, disjointness and subsumption appear when dealing with semantic networks in knowledge representations and conceptual modeling in database theory. In this context, subsumption corresponds to the is-a relation saying that an element of a given type is also an element of another type, and disjointness establishes that two types do not share common elements.

There exist well-known inference rules to derive new knowledge associated with subsumption and disjointness[8], which avoid representing explicitly the relations between every pair of objects. For hierarchical structures, representing subsumption relations explicitly through foreign keys in the relational database context is usual; however, combining subsumption and disjointness requires representing a very large number of relations.

Focusing on the spatial domain, spatial reasoning aims to determine if a set of topological relations defined over a set of spatial objects is consistent, meaning that there are no contradictions among the topological relations, or if the set of topological relations satisfy a consistency network[9,10]. For example, given objects x, y and z, and the relations "x is in y", "y is in z", and "x is disjoint with z", the dataset is topologically inconsistent because the first two relations imply that x is in z. As this example shows, checking topological consistency uses the notion of *composition* of topological relations, which is basically one type of inference rule. Composition allows deriving a relation between x and z from relations between x and y, and between y and z[11].

In addition to the use of inference rules in the spatial domain, Atzeni and Stott Parker Jr.[12,13] provide a set of inference rules for is-a and disjoint relations, and show their soundness and completeness. They use set theory and analyze negative terms as the complement of a given set. Similarly, de Bra and Paredaens[14] propose a set of inference rules for afunctional dependencies (afds) together with functional dependencies (fds), and prove they are sound and complete. In this work we build on the research of Hegner and Rodríguez[8], which, in addition to the rules introduced by Atzeni and Stott Parker Jr.[12,13], incorporates negative relations for a non-empty subset of the domain and proves the soundness and completeness of the set of positive and negative inferences over subsumption and disjointness.

The implementation of rule-based inference systems uses two main strategies: forward chaining and backward chaining[5]. Forward chaining starts with known facts to derive conclusions, whereas backward chaining starts from the conclusions to obtain the facts that support such conclusions. The forward chaining strategy, in particular, has the same form as a relational

database query, which is composed of a set of selection conditions and joins. More precisely, they use match algorithms that take a network structure of selection conditions and joins, and instantiate the network with facts in the database, which can trigger recursive processes. The efficiency of these systems motivates studies addressing their optimization and performance evaluation[15]. Different algorithms create different networks based on storing intermediate results of "and" and "joins"[16,17]; a more recent work[18] compares rule-based systems for the semantic work based on criteria that include not only performance but also functionality. Unlike these previous studies, our work focuses on a set of particular rules that is useful in the context of granular structures organized by subsumption and disjointness, and it proposes forward and backward chaining strategies based on compact data structures to optimize inference processing. We are not aware of strategies that explore the design of particular data structures for networks to improve storage and time processing for a set of inference rules.

3 | PRELIMINARY CONCEPTS

3.1 | Granules and inference rules

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

Following the example given in Section 1, Chile has three granularities that characterize its administrative subdivision: region, province and county. The *granular space* is the set of all regions, provinces and counties, comprising granules from the three different granularities, plus $\bot$ and $\top$ (which represents Chile). In this example, Dom is the geographic space of Chile and GnlToDom is the mapping from the identification of a region to its portion of the geographic space. Granules in each granularity must not overlap, which means that they do not map to sets with common elements in Dom. Intuitively, regions do not overlap because they must be disjoint (even if they are geographically adjacent).

Basic positive rules (or constraints) between granules $g_1, g_2 \in \mathsf{GnlSet}$ are: (i) subsumption rule ($g_1 \sqsubseteq g_2$), meaning that $\text{GnlToDom}(g_1) \subseteq \text{GnlToDom}(g_2)$ and (ii) disjointness rule ($\sqcap\{g_1, g_2\} = \bot$), meaning that $\text{GnlToDom}(g_1) \cap \text{GnlToDom}(g_2) = \emptyset$. Inversely, negative rules are $g_1 \not\sqsubseteq g_2$ and $\sqcap\{g_1, g_2\} \neq \bot$. Note that the application of these rules works under the open world assumption, in contrast with the closed world assumption commonly used in databases; the latter assumes that everything not currently known to be true is false. In the open world assumption, negative rules cannot be derived from the inability to prove their corresponding positive rules, and vice versa.

With these relations, Hegner and Rodríguez[8] propose a set of inference rules. Instead of explicitly storing all possible relations between granules, only a subset of these relations is stored, and the rest are derived through the proposed rules. Two sets of rules are identified: positive and negative; Table 1 shows these two sets of rules.

It was shown[8] that these rule sets are correct and complete. Correctness means that the inference rules cannot derive false relations, while completeness means that (successive applications of) the inference rules derive everything that can be inferred from the basic relations. We say that the rules without premises are "axioms", in our case (c), (d), (e), (f), and (e').

Table 2 shows the inference rules covered in this work and their semantics, excluding axioms because they need no implementation strategy. Those cover all the rules presented in Table 1, except for rule (d'). This is because the strategies we developed are based on the efficient derivation of the subsumption relation, and rule (d') does not have this relation between granules as a premise. Despite this fact, a particular case of this inference rule is covered by rule (6) in Table 2. As a result, we can only ensure completeness and correctness for rules that infer subsumption, disjoint and not-disjoint. For the not-subsumption relation, we offer strategies for its derivation, but we cannot guarantee completeness. The second column in Table 2 shows the mapping between the implemented rule and the original rule it corresponds to. Note that for the case of rules 3, 4 and 5, all of them are based on rule c'. It was implemented this way because cases 3 and 4 correspond to particular cases of the implementation of the rule. In addition, this decomposition allows us a clearer demonstration of correctness of the proposed strategy. A similar reasoning applies to rules 6 and 8, where both are based on rule a', being rule 6 a particular case of this one. In Section 4.2 we explain in more detail how these rules were implemented and their equivalence with the original ones.

As an example of how to use the rules, consider the granules and their explicitly stored relations in Figure 2. Based on these relations, subsumption and disjoint relations can be derived using rules (1) and (2), respectively. For example, the relation $L \sqsubseteq C$

Positive Rules

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

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

Negative Rules

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

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

Table 1 Inferences rules proposed by Hegner and Rodríguez[2].

is derived from the stored relations $L \sqsubseteq G$ and $G \sqsubseteq C$, and $\bigsqcap\{N, I\} = \bot$ can be derived from the relations $N \sqsubseteq H$ and $\bigsqcap\{H, I\} = \bot$. Not-disjoint and not-subsumption relations can be derived using additional rules. More precisely, rules (3), (4) and (5) derive the relation of not-disjoint, and rules (6), (7) and (8) the relation of not-subsumption. For example, rule (3) derives $\bigsqcap\{C, G\} \neq \bot$ since $G \sqsubseteq C$ is a stored relation. Relation $F \not\sqsubseteq J$ can be derived by applying rule (7) since $K \not\sqsubseteq J$ and $K \sqsubseteq F$ are explicitly stored.

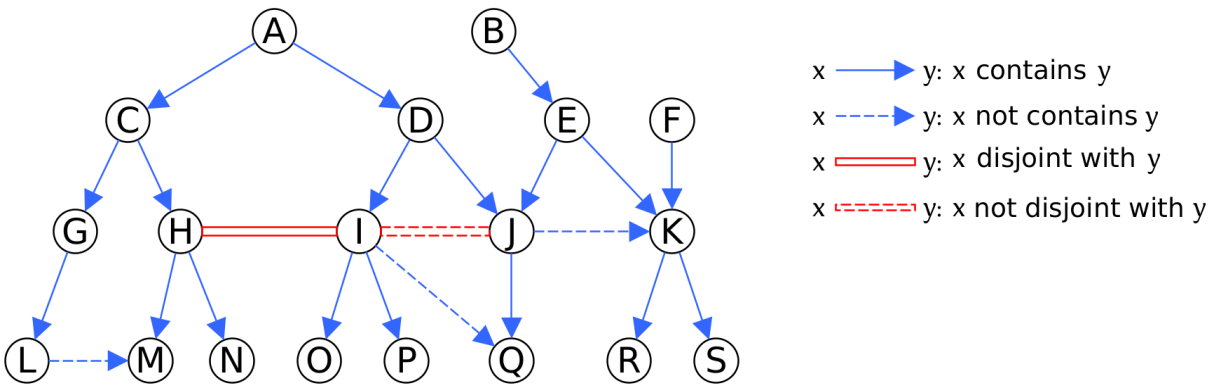


Figure 2 Example of a granule graph. The following are some examples of the stored relations represented in this graph: $C \sqsubseteq A$ (A contains C, which can also be read as C subsumed by A), $J \not\sqsubseteq K$ (J does not contain K), $\bigsqcap\{H, I\} = \bot$ (H disjoint with I), and $\bigsqcap\{I, J\} \neq \bot$ (I not disjoint with J).

3.2 | Succinct data structures

A succinct data structure is an asymptotically *space-efficient* and *query-time-efficient* representation of a data structure[19]. Here, space-efficient means that the data structure uses an amount of space that is close to the information-theoretic lower bound, ideally $z + o(z)$ bits when the information-theoretic lower bound is z. Query-time-efficient means that the data structure achieves an asymptotic query time similar to that (ideally, the same) of structures that do not use a succinct representation.

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

Table 2 Inference rules supported in this work and their equivalence with the original rules proposed by Hegner and Rodríguez[8] (see Table1).

Bitmaps

A Bitmap[20,21,22] $B[1..n]$ is a sequence of $n = |B|$ bits that supports the following queries: i) $rank_x(B, k)$ returns the number of bits equal to x (where $x \in \{0, 1\}$) up to position k in B, ii) $select_x(B, k)$ returns the position of the k-th occurrence of the bit x (where $x \in \{0, 1\}$) in B, and finally iii) the access query, which given a position k returns the bit in that position, $B[k]$. In the case of $select_x(B, k)$, if k is greater than the number of symbols x, the operation returns $n + 1$. As an illustrative example, consider the bitmap B_s in Figure 3, then $rank_1(B_s, |B_s|) = 3$, $select_1(B_s, 1) = 12$, and $B_s[1] = 0$.

The operations rank, select and access can be supported in constant time[21,22] using just sublinear space on top of the bitmap. In addition, subsequent work[23,24] provides compressed versions of the bitmaps by exploiting some properties, such as their sparseness (i.e., an unbalanced number of 0s and 1s).

Wavelet trees

The wavelet tree[25] is a data structure that maintains a sequence of n symbols, $S[1..n] = s_1, s_2, \ldots, s_n$, over an alphabet $\Sigma = [1..\sigma]$, as a balanced binary tree, efficiently supporting the following operations: $access(S, i)$ or $S[i]$, which returns the symbol at position i in S; $rank_c(S, i)$, which counts the times symbol c appears up to position i in S; and $select_c(S, j)$, which returns the position in S of the j-th appearance of symbol c. If we consider the sequence H in Figure 3, which can be represented with a wavelet tree, then $access(H, 1) = H[1] = 10$, $rank_{11}(H, |H|) = 1$, and $select_{14}(H, 1) = 3$. In its simplest form, this structure requires $n\lceil \lg \sigma \rceil + o(n \lg \sigma)$ bits for the data, plus $O(\sigma \lg n)$ bits to store the topology of the tree. The wavelet tree supports more complex queries than the primitives described above. For example, Mäkinen and Navarro[26] studied its connection with a classical two-dimensional range-search data structure and showed how to solve range queries in a wavelet tree. Of particular interest to this work is the operation $range\_check_S(a, a', b, b')$, which returns true if the subsequence $S[a..a']$ contains at least one symbol in the range $[b..b']$, or false, otherwise.

Tree representations

Several compact representations for ordinal trees have been developed. Among the existing representations, two are of particular interest for our work: i) Level-Order Unary Degree Sequence [20] (LOUDS), which represents a tree using a bitmap B. Starting from the root, the tree is traversed level by level. For each new visited node with c children, we append 1^c0 to B. This representation supports a set of queries, which are answered using only rank and select queries over the bitmap B. ii) Balanced parentheses [27] (BP), which represents a tree as a sequence B of balanced parentheses. We traverse the tree in DFS order. Every time we arrive at a node for the first time we append to B an open parenthesis, stored as a 1-bit; when we leave the subtree of a node, we append a close parenthesis, stored as a 0-bit. An example of this structure is B_o in Figure 3, which represents the implementation of the sequence (((( )) (( ) ( ) ( ))) (((( ) ( ) ( ) ( )) (( ))))) obtained from the DFS traversal of the tree in the same figure. The main difference between BP and LOUDS is that in BP any subtree of the original tree is mapped to a contiguous region in B, which facilitates a number of operations that are difficult in LOUDS. Two important primitives of the BP representation are find_open and find_close, which are used to locate the opening and closing parenthesis that match a given closing or opening parenthesis, respectively. The find_open operation takes as input a position in the sequence that corresponds to a closing parenthesis and returns the position of the opening parenthesis that matches it. Similarly, the find_close operation takes as input a position in the sequence that corresponds to an opening parenthesis and returns the position of the closing parenthesis that matches it. For example, considering the aforementioned sequence B_o, find_open of the fourth opening parenthesis returns position 5, whereas find_close of the second closing parenthesis returns position 3. Both representations use $2n + o(n)$ bits to represent a tree of n nodes.

4 | PROPOSED DATA STRUCTURES

We propose a data structure to facilitate the derivation of relations using a subset of the inference rules outlined in Section 3.1, specifically, implementing all the rules presented in Table 2. Our work achieves both correct and complete derivations for $g_1 \sqsubseteq g_2$ (subsumption), $\bigsqcap\{g_1, g_2\} = \bot$ (disjoint), and $\bigsqcap\{g_1, g_2\} \neq \bot$ (not-disjoint). As for $g_1 \not\sqsubseteq g_2$ (not-subsumption), our work can only guarantee correctness but not completeness, as it was explained in Section 3.1.

The derivation of relations follows a two-step strategy: *i)* infer subsumption relations using rule 1, and *ii)* infer new relations using at least one of the rules 2–8. This two-step strategy, which stresses the importance of subsumption relations, owes to the fact that all the rules, except rule 6, have a subsumption relation as a premise. Hence, in Section 4.1 we propose an efficient implementation for subsumption, and in Section 4.2 we describe how to support the other relations.

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

4.1 | Inferring subsumption relations

By representing all the granules and their subsumption relations as a directed graph G^{sub}, we can infer a new subsumption relation between granules g_1 and g_2 ($g_1 \sqsubseteq g_2$) by finding a path that connects them. Classical data structures, such as adjacency lists, adjacency matrices and edge lists can be used for this purpose. As the main operation to support the inference rules involves graph traversals, which are implemented more efficient with adjacency lists, our baseline described in Section 5 is based on those.

In practical cases, such as the granularity structure of Figure 1, it is common to explicitly store only relations between granules of consecutive granularities in the structure. In such cases, the graph $G^{sub}(V, E)$ corresponds to a tree-like graph, that is, a graph in which E is not much larger than V. Because the nodes of the same granularity are disjoint by definition, deviations from a tree can occur only when different overlapping hierarchies exist, with nodes contained in those overlapping areas. The fact that this is uncommon can be exploited in order to define more space-efficient data structures to represent the graphs. We describe next a graph representation that uses $(|E| - |V|) \log_2 |V| + O(|E|)$ bits to represent G^{sub} and, in addition, is optimized to detect the paths that witness subsumption.

Tree-like graphs

Fischer and Peters[28] introduced a succinct data structure to represent directed tree-like graphs called GLOUDS (Graph Level Order Unary Degree Sequence). Tree-like means that only a few edges must be removed from the graph to turn it into a tree. GLOUDS transforms the tree-like graph into a tree T by traversing the graph in BFS order. The starting vertex of the BFS can be any arbitrary vertex with in-degree 0. In our application, we always start with the vertex representing $\top$. When visiting the edge (u, v), if v was already visited, a copy v' of v, called a *shadow node*, is created and the edge (u, v') is added to the tree. We will refer to v' as a copy of v. The tree is then stored using two compact components:

1. A LOUDS-like representation B of T, in which the main difference with the original LOUDS is that the children of a node that are shadow nodes are marked with a symbol 2 instead of a 1 (hence B is no longer a binary sequence).
2. An array H with the identifiers of the shadow nodes in the same order they were written in B, with support for operations $access(H, i)$, $rank_c(H, i)$, and $select_c(H, j)$.

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

In order to improve the performance of checking ancestorship, we modify the representation of the GLOUDS data structure. Specifically, we change the LOUDS representation of T to a balanced parentheses (BP) representation B_o, which allows checking ancestorship in constant time. In BP, nodes are identified by their rank in the DFS traversal of T that produced the representation. Those ranks will be the global identifiers in the implementations of all the rules. We reach constant time because, in the BP representation, a subtree rooted at a vertex v is represented as a contiguous range of parentheses in B_o, which also produces a contiguous range of identifiers. This is a key characteristic of this approach that will also be exploited in Section 4.2, because for the rest of the rules it is important to be able to obtain the set of granules S_g contained by a given granule g in an efficient way. To check if vertex u is an ancestor of another vertex v, we only need to determine if the range representing v is contained in the range representing u. The range representing a vertex can be obtained in constant time by using the primitives of the balanced parenthesis representation explained in Section 3.2. Specifically, if the node identifier is g, we find with $p = select_1(B_o, g)$ the position in B_o of its opening parenthesis, and then with $q = find\_close(B_o, p)$ the position of its closing parenthesis.

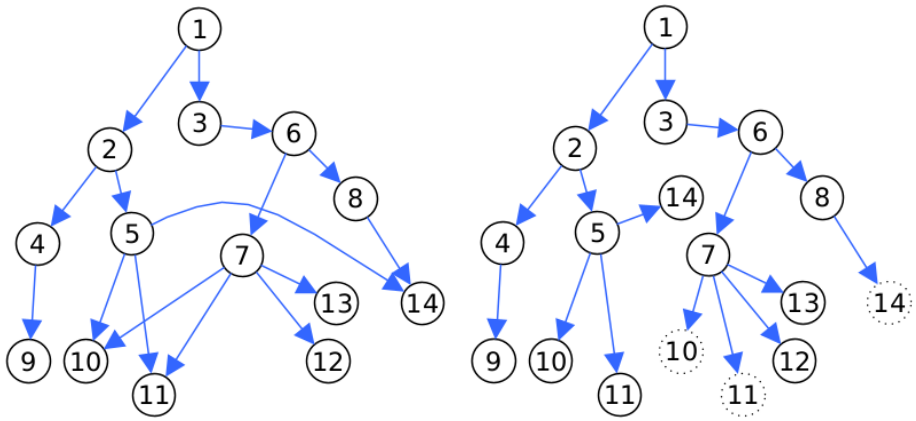
Our adaptation of GLOUDS, referred to as GBP (Graph Balanced Parenthesis), encompasses the following components, as depicted in Figure 3:

1. A bitvector B_o represents T using BP.
2. A second bitvector B_s marks the opening parentheses of B_o that correspond to shadow nodes.
3. The vector H stores the identifiers of those shadow nodes, now in DFS order (i.e., their order in B_s). The vector must support operations $access_H(i)$, $rank_H(c, i)$, and $select_H(c, j)$.

Implementation of inference rule 1

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

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



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

GLOUDS

$$B_{\text{LOUDS}} = 1\,1\,0\,1\,1\,0\,1\,0\,1\,0\,1\,1\,1\,0\,1\,1\,0\,0\;\;0\,0\,0\,2\,2\,0\,2\,0\,0\,0\,0\,0\,0$$
$$H = 10\;11\;14$$

GBP

$$B_o = 1\,1\,1\,1\,0\,0\,1\,1\,0\,1\,0\,1\,0\,0\,0\,0\,1\,1\,1\,1\,0\,1\,0\,1\,0\,1\,0\,0\,1\,1\,0\,0\,0\,0\,0$$
$$B_s = 0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,1\,1\,0\,0\,0\,1$$
$$H = 10\;11\;14$$

(c) GLOUDS and GBP representations. The equivalent parenthesis representation of B_o is ((((())(()()())))((((()()())))(((((()))))))).

Figure 3 Example of a tree-like graph represented with GLOUDS and with GBP.

Algorithm 1: Infers rule 1.

procedure RULE1(g_1, g_2)
  $p_1 \leftarrow select_1(B_o, g_1)$
  $q_1 \leftarrow find\_close(B_o, p_1)$
  $p_2 \leftarrow select_1(B_o, g_2)$
  if $p_1 \leq p_2 \leq q_1$ **then**
    return True ▷ The opening parenthesis of g_2 is contained by the parentheses of g_1
  end if
  $\ell \leftarrow rank_1(B_s, g_1) + 1$
  $r \leftarrow rank_1(B_s, rank_1(B_o, q_1))$
  for $i \leftarrow \ell$ to r **do** ▷ We have to verify the rule for all the shadow nodes in the subtree induced by g_1
    if RULE1($H[i]$,g_2) **then**
      return True
    end if
  end for
  return False
end procedure

Algorithm 2: All granules that reach g.

procedure REACH(g)
  $Q \leftarrow \emptyset$ ▷ Q is a queue
  $S \leftarrow \emptyset$ ▷ S is the result set
  $Q.add(g)$
  while Q is not empty **do** ▷ Compute and add ancestors to Q
    $g' \leftarrow Q.get()$
    $p' \leftarrow parent(g')$
    if p' is valid **then** ▷ The *parent* function returns an invalid node when called with the root node
      $Q.add(p')$
      $S.add(p')$
    end if
    $r \leftarrow rank_{g'}(H, |H|)$
    for $i \leftarrow 1$ to r **do** ▷ Compute and add ancestors of shadow nodes to Q
      $s \leftarrow select_{g'}(H, i)$
      $v \leftarrow select_1(B_s, s)$
      $p \leftarrow parent(v)$
      if p is valid **then**
        $Q.add(p)$
        $S.add(p)$
      end if
    end for
  end while
  return S
end procedure

Algorithm 3: All granules reached by g.

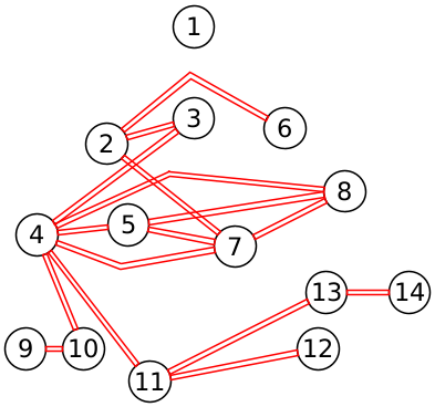
procedure REACHED_BY(g)
  $Q \leftarrow \emptyset$ ▷ Q is a queue
  $S \leftarrow \emptyset$ ▷ S is the result set
  $Q.add(g)$
  while Q is not empty **do**
    $g' \leftarrow Q.get()$
    $p_1 \leftarrow select_1(B, g')$
    $q_1 \leftarrow find\_close(B, p_1)$
    $S.add(\langle p_1, q_1 \rangle)$ ▷ Insert the range representing g' into the result
    $\ell \leftarrow rank_1(B_s, rank_1(B, p_1))+1$
    $r \leftarrow rank_1(B_s, rank_1(B, q_1))$
    for $i \leftarrow \ell$ to r **do**▷ Repeat the same process with each subsumed shadow node
      $Q.add(select_1(B_s, H[i]))$
    end for
  end while
  return S
end procedure

4.2 | Inferring other relations

As we mentioned above, to support the other three relations (disjoint G^{dis}, not-disjoint G^{notdis}, and not-subsumption G^{notsub}) we will use a two-step algorithm. In the first step, we use the GBP-based data structure for G^{sub} defined above to obtain a set of induced subsumption relations S. For the second step, we will have as input the set S and a matrix M^r, where $M^r[i][j]$ indicates that the i-th granule is related with the j-th granule in relation r, where $r = \{dis, notdis, notsub\}$. As an example for the disjoint relation, Figure 4 shows a graph and its equivalent matrix. We note that only the explicit relations, not including those that can be inferred, are stored in these matrices. The identifiers of the nodes in the matrices are those assigned in the GBP-based representation, where the descendants of a node in the tree defined by GBP have identifiers in a consecutive range. Hence, the ranges will be used for querying the matrices. We will represent those binary matrices in a way that (1) uses space proportional to the number of 1s (i.e., of explicit relations to store), and (2) efficiently supports operation $range\_check(a, a', b, b')$, which returns whether there is a marked cell in the region delimited by the rows a and a', and the columns b and b'. Additionally, we will use a simpler operation $check\_cell(a, b) = range\_check(a, a, b, b)$.

Each matrix M^r, which presents the graph $G^r(V, E^r)$, is of dimensions $|V| \times |V|$ and has $|E^r|$ 1s. To store it, we traverse it in row major order, storing the ids of the columns of each marked cell in a sequence I. The resulting sequence is stored in a wavelet tree. To group the values of the same row, we use a bit sequence 10^d, where $d \geq 0$ is the number of cells marked in such row. The bit sequences of all rows are concatenated in top-down order and stored in a plain bitmap C with support for rank/select operations. This representation requires $|E^r| \log |V|$ bits to store the column ids plus $|E^r| + |V| + o(|E^r| + |V|)$ bits to store the bitmap C. Notice that for all the matrices M^r we use the same set V of nodes for all the graphs. The $range\_check(a, a', b, b')$ operation in the matrix is mapped to $range\_check_I(p_1, p_2, b, b')$ in the wavelet tree of I, where $p_1 = rank_0(C, select_1(C, a)) + 1$ and $p_2 = rank_0(C, select_1(C, a' + 1))$. For example, $range\_check(2, 4, 6, 7)$ is mapped to $range\_check_I(p_1, p_2, 6, 7) = range\_check_I(1, 11, 6, 7)$, as $p_1 = rank_0(C, select_1(C, 2)) + 1 = rank_0(C, 2) + 1 = 1$ and $p_2 = rank_0(C, select_1(C, 4 + 1)) = rank_0(C, 16) = 11$.

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



	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	1	0	0	1	1	0	0	0	0	0	0	0
3	0	1	0	1	0	0	0	0	0	0	0	0	0	0
4	0	0	1	0	1	0	1	1	0	1	1	0	0	0
5	0	0	0	1	0	0	1	1	0	0	0	0	0	0
6	0	1	0	0	0	0	0	0	0	0	0	0	0	0
7	0	1	0	1	1	0	0	1	0	0	0	0	0	0
8	0	0	0	1	1	0	1	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	1	0	0	0	0
10	0	0	0	1	0	0	0	0	1	0	0	0	0	0
11	0	0	0	1	0	0	0	0	0	0	0	1	1	0
12	0	0	0	0	0	0	0	0	0	0	1	0	0	0
13	0	0	0	0	0	0	0	0	0	0	1	0	0	1
14	0	0	0	0	0	0	0	0	0	0	0	0	1	0

$C = 1\,1\,0\,0\,0\,1\,0\,0\,1\,0\,0\,0\,0\,0\,0\,1\,0\,0\,0\,1\,0\,1\,0\,0\,0\,0\,1\,0\,0\,0\,1\,0\,1\,0\,0\,1\,0$

$I = 3\,6\,7 \mid 2\,4 \mid 3\,5\,7\,8\,10\,11 \mid 4\,7\,8 \mid 2 \mid 2\,4\,5\,8 \mid 4\,5\,7 \mid 10 \mid 4\,8 \mid 12\,13 \mid 11 \mid 11\,14 \mid 13$

Figure 4 Example of a graph and its matrix representation considering the disjoint relation (top), and components of the wavelet tree representation of such graph (bottom).

Before presenting the implementation of the other rules, we need to introduce two auxiliary algorithms. Algorithm 2 computes all the granules that subsume a granule g, including both induced and explicitly stored relations. Starting with g, the algorithm successively computes the parent relation in the GBP-based representation of G^{sub} (lines 6–11) including shadow nodes (lines 12–21), until no parent relation exists (lines 8 and 17). Finally, Algorithm 3 computes all the granules subsumed by a granule g. Using the property of GBP that subsumed granules tend to have consecutive identifiers, the algorithm returns a set of ranges

instead of a list with all the subsumed granules (line 6–9). The algorithm repeats the same process for each subsumed shadow node (lines 10–14).

4.2.1 | Granules g_1 and g_2 are disjoint

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

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

4.2.2 | Granules g_1 and g_2 are not disjoint

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

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

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

  Theorem. Given granules g_1, g_2, g_3, g_4

$$\frac{\bigsqcap\{g_1,g_2\}\neq\bot \quad g_2\sqsubseteq g_3}{\bigsqcap\{g_1,g_3\}\neq\bot} \equiv \frac{\bigsqcap\{g_1,g_2\}\neq\bot \quad g_2\sqsubseteq g_3 \quad g_1\sqsubseteq g_4}{\bigsqcap\{g_3,g_4\}\neq\bot}$$

  Proof. We have to show implications in both directions.

  1. Let us prove that

$$\frac{\bigsqcap\{g_1,g_2\}\neq\bot \quad g_2\sqsubseteq g_3}{\bigsqcap\{g_1,g_3\}\neq\bot} \Rightarrow \frac{\bigsqcap\{g_1,g_2\}\neq\bot \quad g_2\sqsubseteq g_3 \quad g_1\sqsubseteq g_4}{\bigsqcap\{g_3,g_4\}\neq\bot}$$

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

2. In the other direction

$$\frac{\bigsqcap\{g_1, g_2\} \neq \bot \;\; g_2 \sqsubseteq g_3 \;\; g_1 \sqsubseteq g_4}{\bigsqcap\{g_3, g_4\} \neq \bot} \Rightarrow \frac{\bigsqcap\{g_1, g_2\} \neq \bot \;\; g_2 \sqsubseteq g_3}{\bigsqcap\{g_1, g_3\} \neq \bot}$$

This trivially proved by making again $g_1 = g_4$ and $g_1 \sqsubseteq g_1$.

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

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

4.2.3 | Granule g_1 does not contain granule g_2

Unlike for previous relations, we cannot guarantee completeness for not-subsumption, as explained in Section 3.1. Still, we provide here partial strategies for its derivation. We first check if there is an edge connecting g_1 and g_2 in G^{notsub}. If so, then we conclude that g_1 does not contain g_2. Otherwise, we aim to infer the relation using rules 6, 7 and 8.

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

4.2.4 | Axioms

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

5 | EXPERIMENTAL EVALUATION

5.1 | Implementation details

In this section we provide some practical details about the different implementations. All of them, except the baseline, were implemented using the Succinct Data Structures Library (SDSL).[29]. The source code is available in the following repository: https://github.com/dgaticar/gbp.

Baseline

We compared our data structures with a baseline representation consisting of four adjacency lists, storing direct and inverse relations, one for each relation of subsumption, disjoint, and their respective negations. To compute all the granules reached by a granule g and all the granules that reach g, standard graph traversals algorithms are performed. An alternative baseline would be to store explicitly all the axioms and the induced relations, but this alternative would potentially store up to n^2 relations, which is not feasible. For instance, for the dataset 7 of Section 5.3, the space of this alternative reaches about 16.8GB only for the subsumption relation, while our solution **GBP+Matrix** uses 902.9MB for the four relations.

GLOUDS-based data structures

We implemented two versions of the GLOUDS-based variant. The first version, called **GLOUDS**, implements the original idea of Fischer and Peters[28], using wavelet trees (`wt_int<>` of the SDSL library) to store both the LOUDS representation of the tree-like graph and H, the ids of the shadow nodes. The inference rules were implemented using the ideas presented in Sections 4.1 and 4.2, adapting the Algorithms 2 and 3 to use the LOUDS representation.

For the variant that uses a balanced parenthesis representation instead of LOUDS, called **GBP**, we used the implementation of the compact tree of Navarro and Sadakane[30] to store B_o (`bp_support_sada<>` in SDSL), a plain bitmap to store B_s (`bit_vector` in SDSL), and a wavelet tree to store H (`wt_int<>` in SDSL). We tested three variants of GBP, varying only in how the IDs of each granule are stored in H: *i)* A first variant storing the original IDs directly in a wavelet tree; *ii)* a second variant, where the identifiers stored in H are previously mapped to a contiguous range, using an extra bitmap H_m (`bit_vector` in SDSL) for the mapping between the original IDs and the contiguous identifiers; and *iii)* a third variant, using the same strategy as variant ii) but using a compressed bitmap[23] to represent H_m (`sd_vector` in SDSL). Additionally, we included an extra plain bitmap B_{open} (`bit_vector` in SDSL) to mark the non-shadow nodes of GBP. Technically, this bitmap is redundant (rank and select on this bitmap can be simulated by using rank and select over the other components of the structure), but we decided to include it since we obtained slightly better running times, at the cost of negligible extra space.

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

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

Matrices

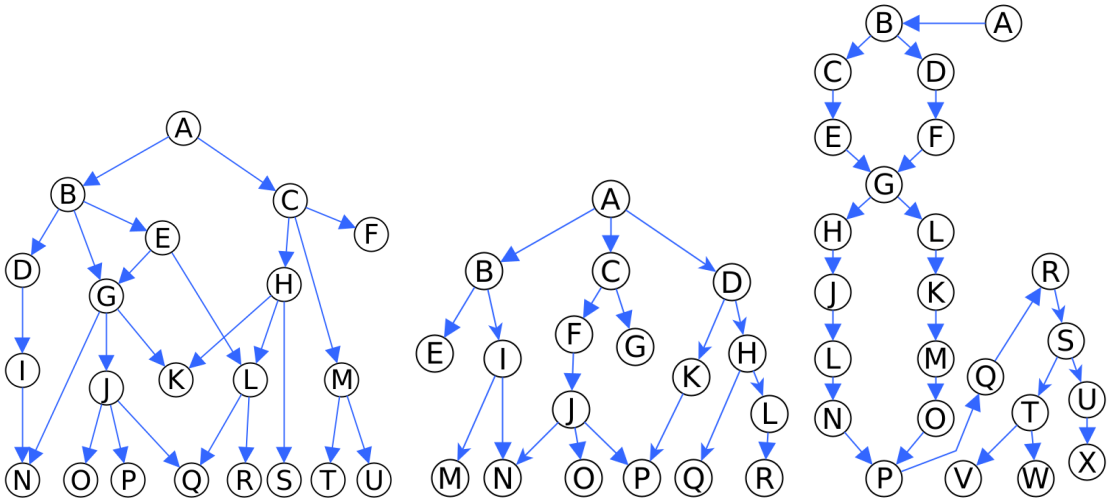
The matrices were implemented verbatim to the description in Section 4.2, using `wt_int<>` and `bit_vector` of the SDSL library to implement wavelet trees and bitmaps, respectively. We refer to the implementation of GBP complemented with the matrices as **GBP+Matrix**. Note that this representation uses a GBP for the relation subsumption and one additional matrix for each of the other three relations. The alternative representation that uses a GBP for each relation is referred to as GBP.

Intersection of sets

For the strategy described in Section 4.2, specifically for inference rule 4, the final step consists in checking out if there is an intersection between two sets S_1 and S_2, obtained in previous steps. Remember that each element of the sets represents a range of values, stored as a pair. The intersection algorithm works in two steps: *i)* Sort both sets S_1 and S_2, based on the first element of each pair, and then *ii)* traverse both sorted sets from smallest to largest values. When visiting the i-th range of S_1 and j-th range of S_2, we check if they intersect or not. After that, the range with the lowest closing value is replaced by the next range in the respective set. Step *ii)* is repeated until the final range in one of the sets is reached. The time complexity is $O(|S_1| \log |S_1| + |S_2| \log |S_2|)$ for the first step, and $O(|S_1| + |S_2|)$ for the second, where $|S|$ represents the cardinality of the set S. No extra space is required.

5.2 | Experimental setup

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



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

Figure 5 Granularity structures used in the generation of the synthetic datasets.

5.3 | Datasets

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

5.3.1 | Synthetic datasets

We developed a multigranular instance generator in which we can control some characteristics of the generated datasets. This allows us to evaluate the performance of the different approaches both under favorable and unfavorable configurations. The generator receives as input a description of the granularity graph, the number of granules in each granularity, and the number of subsumption relations between granularities, with the exception that this description only contains subsumption relations between granularities, in addition to the number of not subsumption, disjoint and not disjoint relations. Based on this information, a multigranular instance is generated, represented in the form of four graphs (one per relation). The generator does not guarantee that the generated instances contain the minimum information, as it may produce relations that can also be derived using inference rules.

Taking the granularity structures from Figure 5 as input, we utilized the instance generator to compute nine synthetic instances – three for each granularity structure. We varied the number of granules per granularity and the number of not subsumption, disjoint, and not disjoint relations. Table 3 summarizes the characteristics of each generated instance, where *Granules* refers to the number of granules in the instance, *Subsumption Edges* to the number of relations in the subsumption graph obtained from the dataset, and *Other Relations* to the amount of not subsumption, disjoint and not disjoint relations (the same amount for the 3 relations).

The *Multigranular Structure I* (Figure 5(a)) represents an instance where the subsumption graphs obtained from it will have a large number of cycles, which is not favorable for the implementations based on compact data structures. The subsumption graphs obtained from *Multigranular Structure II* (Figure 5(b)), instead, will have a low number of cycles, which is favorable for the implementations based on compact data structures. Finally, the *Multigranular Structure III* (Figure 5(c)) represents an instance where the generated subsumption trees will have a larger height than in previous cases, which is unfavorable for the baseline.

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

Table 3 Characteristics of the generated synthetic datasets.

5.3.2 | Real-world dataset

A real-world dataset was obtained to evaluate all the structures in practical scenarios. This dataset is based on the TIGER dataset,[2] provided by the U.S. Census Bureau, which corresponds to geographic and cartographic data of the administrative divisions in the United States. For this study, we computed relations between each possible pair of granules based on their geometry. From these obtained relations, a subset was selected for analysis. The dataset comprises a total of 11,555,150 granules, 11,705,035 subsumption relations and 691,350 other relations. Figure 6 illustrates the granularity structure of this dataset, while Table 4 presents the distribution of granules for each granularity.

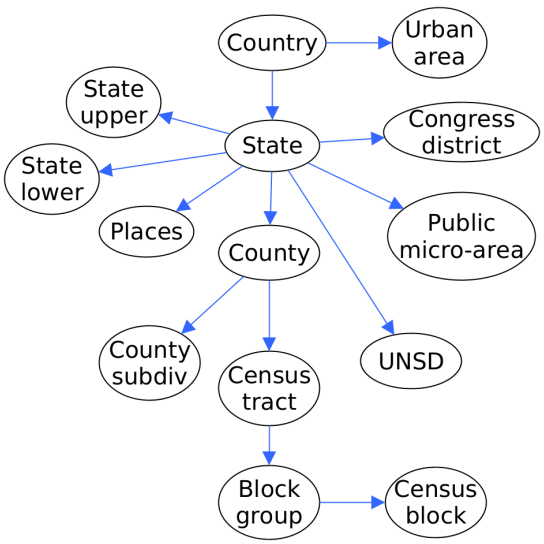


Figure 6 Granularity structure of the real-world dataset.

5.4 | Results and discussion

5.4.1 | Execution time

Based on the implementations described in Section 4, and using the datasets described above, we conducted two experimental evaluations to measure the performance of the implementations on various multigranular instances: *a)* A first experimental

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

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

Table 4 Number of granules for each granularity in the real-world dataset.

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

Table 5 Summary of the best-performing variants for GBP and GBP+Matrix. No mapping represents the variant that does not map the identifiers of H to a contiguous range, plain bitmap represents the variant that maps to a contiguous range using a plain bitmap, while compressed bitmap uses a compressed bitmap for the mapping.

evaluation, in which the assessment is made at the rule level. This analysis offers a more detailed perspective and provides comprehensive information about the performance of different strategies across all the datasets. *b)* A second experimental evaluation, where the assessment is performed at the relation level, as certain relations require checking multiple rules. This experiment, presented only on the real dataset, provides us with an overview of the behavior of the implementations in deriving each relation.

To carry out both experimental evaluations, a total of 10,000 queries were performed. For each executed query, two granules were randomly selected, with the only restriction that both granules could not belong to the same granularity. Each executed operation was repeated 10 times, leaving the first repetition for cache warming and reporting the average of the last nine (a run without cache warming was evaluated and the results were the same, with negligible differences). Regarding the GBP-based implementations, only the variant with the best average performance for each dataset is shown. Algorithms 4, 5, 6 and 7 show the strategy used in the second experimentation to derive the four possible relations. Table 5 shows the best performing variant for GBP and GBP+Matrix.

Algorithm 4: Infer subsumption relation between g_1 and g_2.

```
1: procedure SUBREL(g1, g2)
2:     return RULE1(g1,g2)
3: end procedure
```

Algorithm 5: Infer disjoint relation between g_1 and g_2.

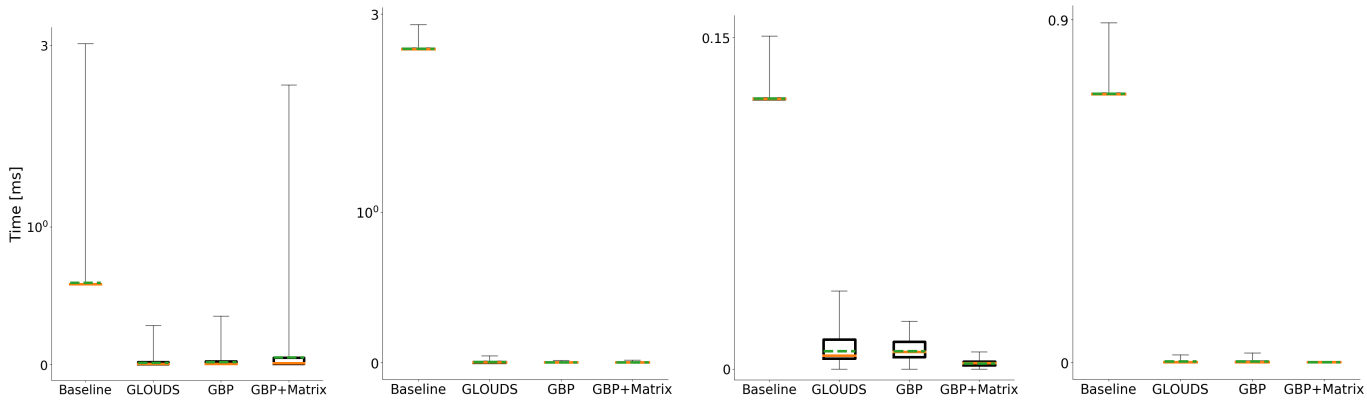
```
1: procedure DISREL(g1, g2)
2:     return RULE2(g1,g2)
3: end procedure
```

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

```
1: procedure NOTDISREL(g1, g2)
2:     if RULE1(g1,g2) or RULE4(g1,g2)
3:       or RULE5(g1,g2) then
4:         return True
5:     end if
6:     return False
7: end procedure
```

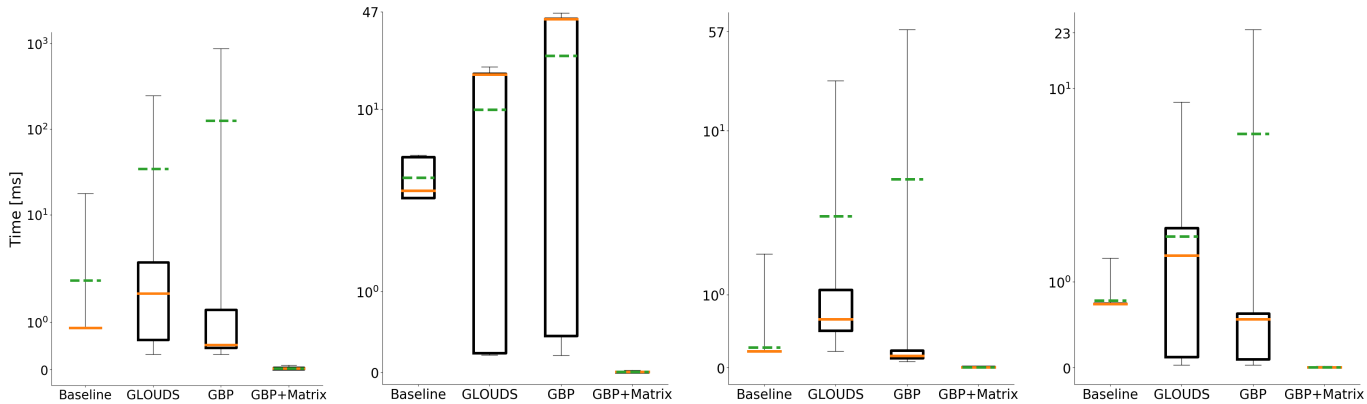
Algorithm 7: Infer not-subsumption relation between g_1 and g_2.

```
1: procedure NOTSUBREL(g1, g2)
2:     if RULE2(g1,g2) or
3:        RULE7_8(g1,g2) then
4:         return True
5:     end if
6:     return False
7: end procedure
```



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

Figure 7 Running time in milliseconds for the inference rule 1.



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

Figure 8 Running time in milliseconds for the inference rule 2.

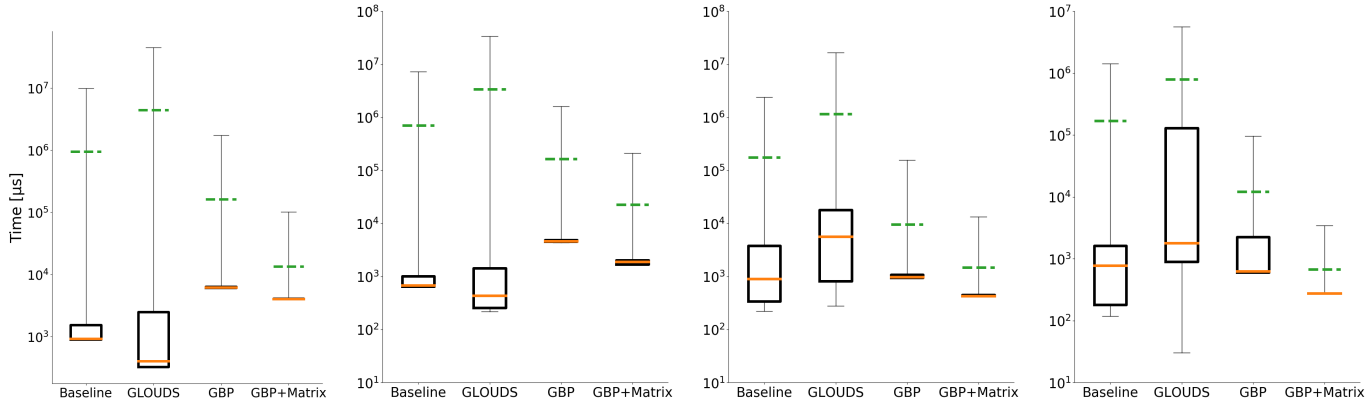
Table 6 shows the average query time of executing the evaluated inference rules. We omit rules 3 and 6 because, as it was explained in Section 4, their implementation is straightforward using either rule 1 or the information explicitly stored. For each multigranular structure, we show only the results of the largest dataset as, in general, there are no significant differences with the other datasets. When necessary, we explicitly mention other datasets to highlight some differences. In general, the GBP+Matrix structure is the one that provides the best query times. Although GLOUDS is highly competitive for rule 1, which forms the foundation for the others, its use in the remaining rules is clearly surpassed by other alternatives. Next, a more detailed analysis with graphs is presented for each of the evaluated rules. In these box and whisker plots, the solid orange lines and the dashed green lines represent the median and the average of the 10,000 queries executed for each operation on each dataset, respectively.

Figure 7 presents the results for inference rule 1. For this rule there is no clear winner, as can be seen in Table 6, the differences in the performance of GLOUDS versus GBP-based structures are not conclusive. This may be partially explained by the fact that, in the datasets, the height of the granularities is negligible compared to the amount of granules per instance. Thus, GLOUDS, whose performance is directly dependent on the height, exhibits a similar behavior to GBP-based alternatives, which do not depend on the height of the instances. In addition, as we will see in the results for the following rules, GBP and GBP+Matrix yield better results when the computation of ranges of consecutive identifiers is combined with the other rules. Notice also that the poor performance of the baseline is mainly due to the large number of cache misses produced at the time of obtaining the ancestors of a granule, by following the reverse references from children to parents (which are explicitly stored). This can be concluded by analyzing the smaller datasets, in which, although the baseline presents worse performance than the other

	Implementation	Rule 1	Rule 2	Rule 4	Rule 5	Rule 7–8
1	Baseline	0.080	0.093	0.123	0.142	0.089
	GLOUDS	**0.004**	0.418	0.502	0.688	0.352
	GBP	0.010	0.843	0.020	0.122	0.019
	GBP+Matrix	0.051	**0.010**	**0.005**	**0.005**	**0.004**
2	Baseline	0.033	0.060	0.091	0.092	0.075
	GLOUDS	**0.001**	0.129	0.321	0.531	0.300
	GBP	0.003	0.180	0.020	0.120	0.080
	GBP+Matrix	0.002	**0.005**	**0.003**	**0.003**	**0.002**
3	Baseline	0.015	0.026	0.029	0.028	0.020
	GLOUDS	0.003	0.065	0.109	0.134	0.098
	GBP	0.006	0.114	0.010	0.29	0.017
	GBP+Matrix	**0.002**	**0.008**	**0.001**	**0.001**	**0.001**
4	Baseline	1.536	1.005	0.992	0.976	0.764
	GLOUDS	0.006	1.211	4.201	7.130	2.805
	GBP	0.013	1.612	0.201	1.203	0.402
	GBP+Matrix	**0.005**	**0.010**	**0.030**	**0.030**	**0.029**
5	Baseline	1.810	0.658	0.904	0.897	0.603
	GLOUDS	**0.001**	0.317	3.092	5.304	2.504
	GBP	0.002	0.371	0.341	1.403	0.251
	GBP+Matrix	0.002	**0.005**	**0.029**	**0.031**	**0.026**
6	Baseline	0.035	0.055	0.070	0.069	0.045
	GLOUDS	0.003	0.088	0.287	0.354	0.201
	GBP	0.006	0.131	0.010	0.093	0.040
	GBP+Matrix	**0.002**	**0.008**	**0.001**	**0.002**	**0.001**
7	Baseline	0.593	1.827	954.293	1,009.283	562.184
	GLOUDS	**0.012**	34.241	4,503.498	8,981.864	2,879.758
	GBP	0.017	125.174	185.857	2,409.587	1,049.544
	GBP+Matrix	0.051	**0.026**	**19.431**	**19.859**	**10.374**
8	Baseline	2.232	3.393	712.394	734.856	419.508
	GLOUDS	0.004	9.994	3,873.431	5,647.110	3,021.004
	GBP	**0.002**	23.453	198.574	1,201.123	198.031
	GBP+Matrix	**0.002**	**0.007**	**23.001**	**405.982**	**13.892**
9	Baseline	0.122	0.278	183.284	188.394	122.495
	GLOUDS	0.008	2.228	1.287.192	1,889.270	582.039
	GBP	0.008	4.265	9.487	183.284	66.398
	GBP+Matrix	**0.002**	**0.009**	**1.653**	**3.982**	**1.302**
Real-world	Baseline	0.706	0.784	145.394	163.304	81.485
	GLOUDS	0.002	1.529	793.948	1,250.290	320.102
	GBP	0.003	5.083	12.998	363.291	29.091
	GBP+Matrix	**0.001**	**0.003**	**0.702**	**1.730**	**0.703**

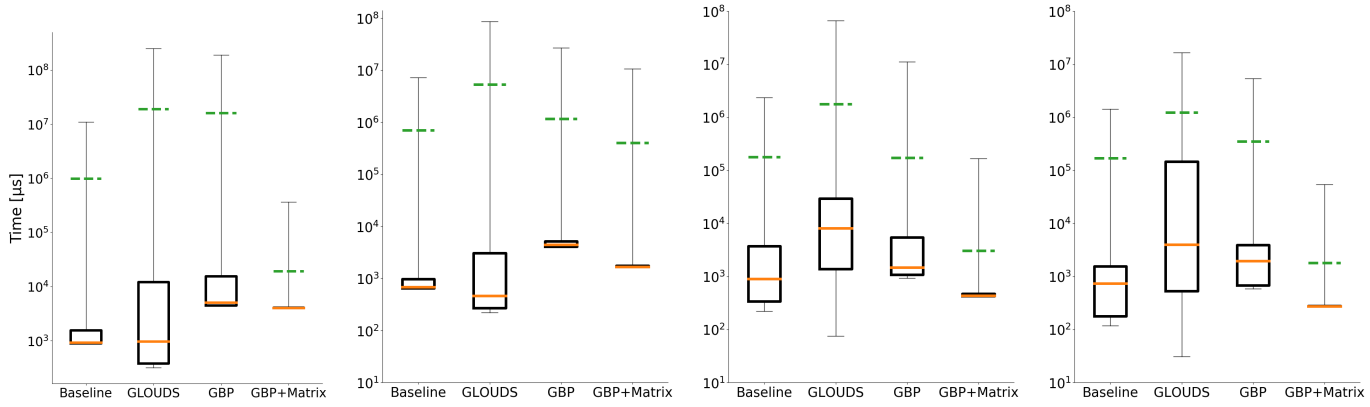
Table 6 Average query times, in ms. The fastest query times for each rule and input dataset are highlighted in bold.

implementations, the differences are smaller. Finally, no significant changes are perceived in the behavior of the structures when confronted with multigranular structures with different complexities.



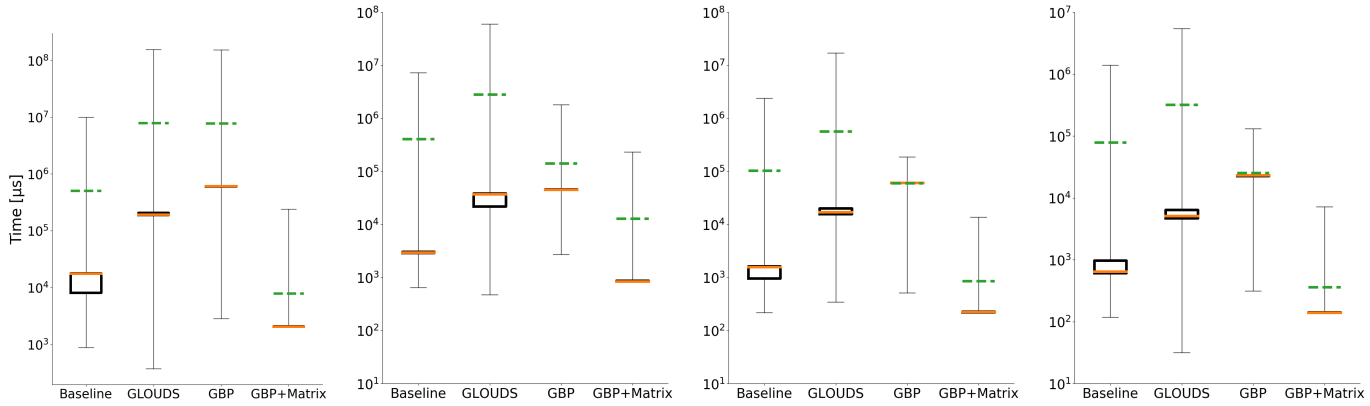
(a) Dataset 7 (b) Dataset 8 (c) Dataset 9 (d) Real-world dataset

Figure 9 Running time in microseconds for the inference rule 4.



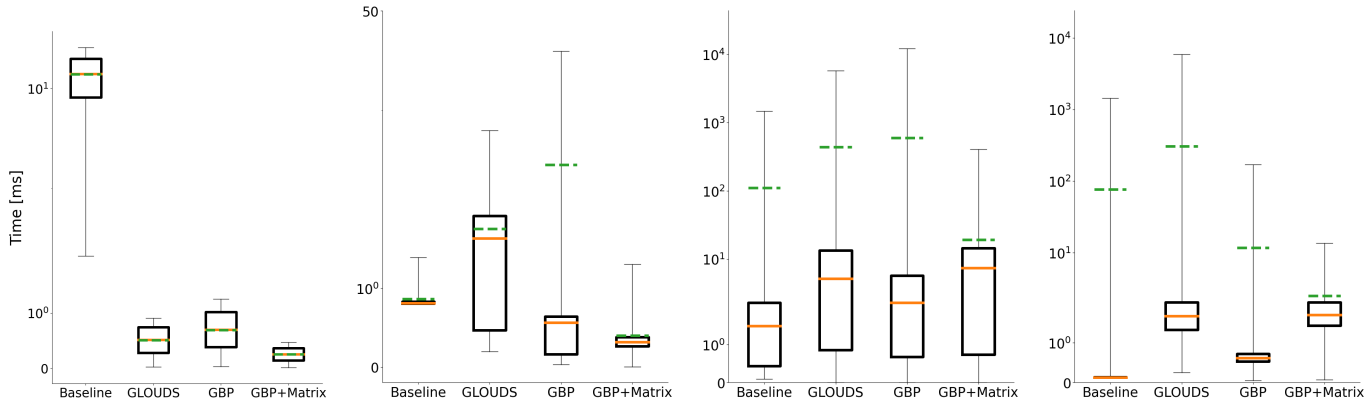
(a) Dataset 7 (b) Dataset 8 (c) Dataset 9 (d) Real-world dataset

Figure 10 Running time in microseconds for the inference rule 5.



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

Figure 11 Running time in microseconds for the inference rule 7-8.



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

Figure 12 Running time in milliseconds for the experimental evaluation at the relation level on the Real-world dataset.

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

Figure 9 displays the results for inference rule 4. In this case, GLOUDS exhibits the highest variance, with the lowest and highest query times observed. This variance contributes to GLOUDS having the worst average performance among the evaluated approaches. On the other hand, GBP+Matrix achieves the best average query time, despite being slower for favorable queries due to the overhead involved to support range operations over the matrix in the cases of nodes without descendants or with only one descendant. However, this approach ensures a low query time for unfavorable cases. GBP performs better than GLOUDS in terms of average query time but has a worse median for datasets 7 and 8. This highlights an advantage of the GBP variants, which may be slightly slower for favorable cases due to the extra mappings between bitmaps but perform better overall, even in the worst-case scenarios. All evaluated implementations exhibit consistent trends regardless of the dataset being evaluated.

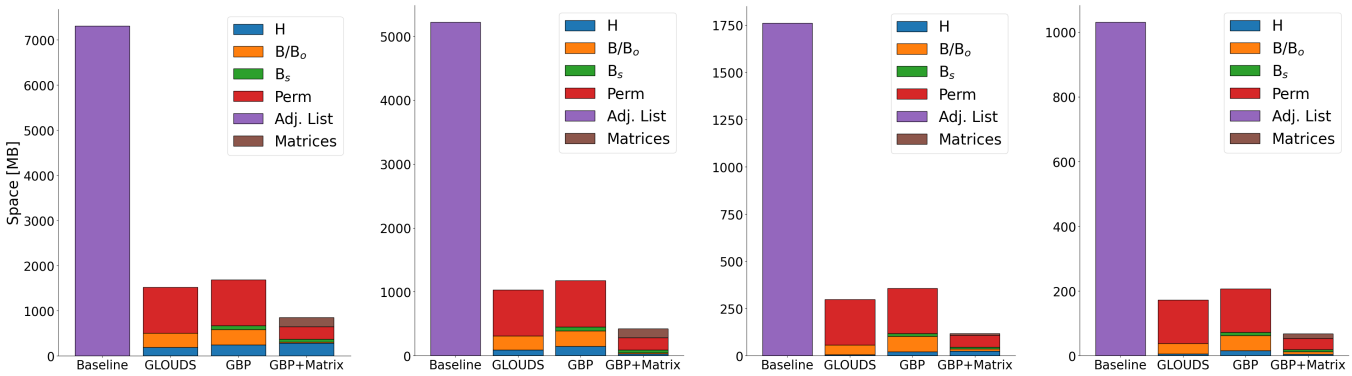
Figure 10 displays the results for inference rule 5. The behavior observed for this rule is similar to that of inference rule 4. It leads, however, to an increase in response time for GLOUDS and GBP in their unfavorable cases, with GBP becoming slower than the baseline in its worst cases. GBP+Matrix also increments its query time in worst-case, but remains the implementation with the best average for all datasets and better median for the dataset 9 and the real-word dataset.

Figure 11 illustrates the outcomes for inference rules 7-8. In this context, akin to the scenario concerning rule 4, it is observed that GLOUDS exhibits the highest variance, with the lowest and highest query times observed. GBP+Matrix, on the other hand, achieves the best average and median query times. Notably, GBP+Matrix performs significantly better on average than the other implementations in datasets based on Multigranular Structure III, consistently delivering the best query times for all evaluated queries (including the most favorable and the most unfavorable). This is because Multigranular Structure III is taller than the others and produces a moderate number of shadow nodes, making it a favorable scenario for GBP+Matrix.

Finally, Figure 12 displays the result for the second experimental evaluation carried out at the relation level on the Real-world dataset. For subsumption and disjoint relations, the same behavior as described above for their individual rules is presented. For the not-disjoint relation it can be seen how GBP+Matrix shows the best performance both on average, median, and for the worst cases. In the case of the not-subsumption relation, it can be seen that GBP+Matrix presents better performance for both the worst case and on average than the rest of the implementations, even so, it presents a worse median than the rest, which shows that, in general, for favorable queries it tends to be slower than other implementations. It is also worth noting that, although GBP has a worse average performance than GBP+Matrix, it exhibits a better median and ranks as the second-best performing implementation concerning unfavorable cases.

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

Table 7 Total space used for each data structure, in MB.



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

Figure 13 Detailed analysis of the space used by each approach.

5.4.2 | Space usage

Table 7 shows the total space used by each implementation for each dataset. In all of them, the baseline occupies considerable more space than the others, while GBP+Matrix is the one that uses the least space. GBP uses, in general, more space than GLOUDS (close to 20% extra in the worst cases), because it needs additional bitmaps to distinguish the shadow nodes.

Figure 13 provides a detailed view of the space occupied by the main components for the largest datasets. As it can be observed, the primary distinction between GBP+Matrix and the GBP and GLOUDS implementations is that it occupies less space to represent permutations. This is because it stores a single permutation to map identifiers from the input dataset to G^{sub}, compared to storing four permutations to map from the input to G^{sub} and from G^{sub} to the other relations.

6 | CONCLUSIONS AND FUTURE WORK

We have focused on developing data structures and strategies capable of deriving information from a set of subsumption and disjointness relations, including their respective negations. To the best of our knowledge, no other works have specifically concentrated on the development of efficient structures for deriving subsumption and disjointness relations. In this article, we propose several strategies for deriving information using a set of inference rules, which had been proven to be both correct and complete[2]. Our implementation is correct and complete for the subsumption, disjoint, and not-disjoint relations. However, we cannot assure completeness for the not-subsumption relation, as our strategy does not support all the rules in the original work of Hegner and Rodríguez[2]. Therefore, developing an efficient implementation that also ensures completeness for the not subsumption relation remains as an open problem.

Our experimental evaluation demonstrates that the proposed methods are not only more space-efficient but also significantly faster when compared to a baseline approach that explicitly stores the information of each relation using graphs and solves operations using well-known traversal algorithms.

In addition to the future research direction mentioned above, an interesting problem is to study the minimum amount of information that must be stored for each relation so that, when applying the proposed rules, all the information can be derived.

In other words, our strategy ensures that, given a dataset, we obtain the maximum amount of information from it, regardless of whether there is more information stored in that dataset than necessary to achieve the same result. Related to knowing the minimum necessary information to store, it would be interesting to develop an algorithm that, given an initial set S of relations, could reduce it to the minimum set S' that allows to derive the maximum number of possible relations. Finally, it could be interesting to study a dynamic variant of the proposed solution so that new facts provided externally or induced through the application of implemented rules can be stored.

ACKNOWLEDGMENTS

This work was funded by: ANID Millennium Science Initiative Program - Code ICN17_002; PAI grant 77190038 and FONDECYT grant 11220545 (1st author); PFCHA/Doctorado Nacional/2020-21201986 (2nd author); FONDECYT Grant 1-230755 (3rd author); GRC: ED431C 2021/53, partially funded by GAIN/Xunta de Galicia; PID2022-141027NB-C21 (EarthDL), TED2021-129245B-C21 (PLAGEMIS), PID2020-114635RB-I00 (EXTRACompact), PDC2021-121239-C31 (FLATCity-POC), and PDC2021-120917-C21 (SIGTRANS): partially funded by MCIN/AEI/10.13039/501100011033 and "NextGenerationEU"/PRTR (5th author). CITIC is funded by the Xunta de Galicia through the collaboration agreement between the Department of Culture, Education, Vocational Training and Universities and the Galician universities for the reinforcement of the research centers of the Galician University System (CIGUS).

References

1. Ramakrishnan R, Ullman JD. A survey of deductive database systems. *J. Log. Program.* 1995; 23(2): 125–149.
2. Hegner SJ, Rodríguez MA. A Model for Multigranular Data and Its Integrity. *Informatica* 2017; 28(1): 45–78.
3. Bettini C, Dyreson CE, Evans WS, Snodgrass RT, Wang XS. A Glossary of Time Granularity Concepts. In: ; 1997: 406-413.
4. Iftikhar N, Pedersen TB. Schema Design Alternatives for Multi-granular Data Warehousing. In: . 6262 of *Lecture Notes in Computer Science*. Springer; 2010: 111–125.
5. Al-Ajlan A. The comparison between forward and backward chaining. *International Journal of Machine Learning and Computing* 2015; 5(2): 106.
6. Navarro G. *Compact Data Structures - A Practical Approach*. Cambridge University Press . 2016.
7. Camossi E, Bertolotto M, Bertino E. A multigranular object-oriented framework supporting spatio-temporal granularity conversions. *International Journal of Geographical Information Science* 2006; 20(5): 511-534.
8. Hegner SJ, Rodríguez MA. Inference Rules for Binary Predicates in a Multigranular Framework. *CoRR* 2023; abs/2303.15138.
9. Mackworth AK. Consistency in Networks of Relations. *Artif. Intell.* 1977; 8(1): 99–118.
10. Li S. On Topological Consistency and Realization. *Constraints An Int. J.* 2006; 11(1): 31–51.
11. Bennett B. Determining consistency of topological relations. *Constraints* 1998; 3(2): 213–225.
12. Atzeni P, Stott Parker Jr. D. Formal Properties of Net-Based Knowledge Representation Schemes. *Data Knowl. Eng.* 1988; 3: 137–147.
13. Atzeni P, Stott Parker Jr. D. Set Containment Inference and Syllogisms. *Theor. Comput. Sci.* 1988; 62(1-2): 39–65.
14. de Bra P, Paredaens J. Horizontal Decompositions for Handling Exceptions to Functional Dependencies. In: Advances in Data Base Theory. Plemum Press; 1982; New York: 123–141.
15. Wang Y, Hanson EN. A Performance Comparison of the Rete and TREAT Algorithms for Testing Database Rule Conditions. In: IEEE Computer Society; 1992: 88–97.

16. Forgy C. Rete: A Fast Algorithm for the Many Patterns/Many Objects Match Problem. *Artif. Intell.* 1982; 19(1): 17–37.

17. Miranker DP. TREAT: A Better Match Algorithm for AI Production System Matching. In: Morgan Kaufmann; 1987: 42–47.

18. Rattanasawad T, Buranarach M, Saikaew KR, Supnithi T. A Comparative Study of Rule-Based Inference Engines for the Semantic Web. *IEICE Trans. Inf. Syst.* 2018; 101-D(1): 82–89.

19. Jacobson GJ. *Succinct static data structures*. Carnegie Mellon University . 1988.

20. Jacobson G. Space-efficient Static Trees and Graphs. In: IEEE Computer Society; 1989: 549–554.

21. Clark D. Compact pat trees. 1997.

22. Munro JI. Tables. In: Springer. Springer Berlin Heidelberg; 1996; Berlin, Heidelberg: 37–42.

23. Okanohara D, Sadakane K. Practical Entropy-Compressed Rank/Select Dictionary. In: SIAM; 2007.

24. Raman R, Raman V, Rao SS. Succinct indexable dictionaries with applications to encoding k-ary trees and multisets. In: ACM/SIAM; 2002: 233–242.

25. Grossi R, Gupta A, Vitter JS. High-order entropy-compressed text indexes. In: ACM/SIAM; 2003: 841–850.

26. Mäkinen V, Navarro G. Rank and select revisited and extended. *Theor. Comput. Sci.* 2007; 387(3): 332–347.

27. Munro JI, Raman V. Succinct Representation of Balanced Parentheses and Static Trees. *SIAM Journal on Computing* 2001; 31(3): 762-776.

28. Fischer J, Peters D. GLOUDS: Representing tree-like graphs. *Journal of Discrete Algorithms* 2016; 36: 39-49. WALCOM 2015.

29. Gog S, Beller T, Moffat A, Petri M. From Theory to Practice: Plug and Play with Succinct Data Structures. In: ; 2014: 326-337.

30. Navarro G, Sadakane K. Fully-Functional Static and Dynamic Succinct Trees. *ACM Transactions on Algorithms* 2014; 10(3): article 16.

How to cite this article: