# A foundation for spatio-textual-temporal cube analytics

Iqbal, Mohsin; Lissandrini, Matteo; Pedersen, Torben Bach

*Published in:*
Information Systems

*DOI (link to publication from Publisher):*
10.1016/j.is.2022.102009



*Publication date:*
2022

*Document Version*
Publisher's PDF, also known as Version of record

Information Systems 108 (2022) 102009

Contents lists available at ScienceDirect

Information Systems

journal homepage: www.elsevier.com/locate/is

# A foundation for spatio-textual-temporal cube analytics

Mohsin Iqbal *, Matteo Lissandrini, Torben Bach Pedersen

*Aalborg University, Denmark*

ARTICLE INFO

*Article history:*
Received 11 July 2021
Received in revised form 21 November 2021
Accepted 14 February 2022
Available online 16 February 2022
Recommended by Gottfried Vossen

*Keywords:*
Data cube
OLAP
Spatial analytics
Textual analytics
Spatio-textual-temporal data
Spatial-textual-temporal measures

ABSTRACT

Large amounts of *spatial, textual, and temporal (STT) data* are being produced daily. This is data containing an *unstructured* component (text), a *spatial* component (geographic position), and a *time* component (timestamp). Therefore, there is a need for a powerful and *general* way of analyzing *STT data together*. In this paper, we define and formalize the *Spatio-Textual-Temporal Cube (STTCube)* structure to enable *combined* effective and efficient analytical queries over *STT data*. Our novel data model over *STT objects* enables novel *joint and integrated* STT insights that are hard to obtain using existing methods. Furthermore, our proposed STTCube Incremental Maintenance ($IM_{stt}$) method maintains the already constructed STTCube efficiently when new data arrives. Moreover, we introduce the new concept of *STT measures* with associated novel STT-OLAP operators. To allow for efficient large-scale analytics, we present a pre-aggregation framework for exact and approximate computation of *STT measures*. Our comprehensive experimental evaluation on a real-world Twitter dataset confirms that our proposed methods reduce query response time by 1–5 orders of magnitude compared to the *No Materialization* baseline and decrease storage cost between 97% and 99.9% compared to the *Full Materialization* baseline while adding only a negligible overhead in the STTCube construction time. Moreover, *approximate computation* achieves an accuracy between 90% and 100% while reducing query response time by 3–5 orders of magnitude compared to *No Materialization* and $IM_{stt}$ achieves an order of magnitude improvement in maintenance time compared to the baseline maintenance method.



## 1. Introduction

Due to the increased usage of mobile devices and advancements in accurate geo-tagging, more and more geo-tagged data is being produced [1]. In particular, social media platforms like Twitter and Facebook are some of the primary sources of geo-tagged data, usually in the form of posts, comments, and reviews. This type of data contains spatial, textual, and temporal (STT) information. As a result, *STT data* analysis is becoming increasingly important [2] since it allows to extract new insights regarding customer satisfaction, user-generated content shared online, and brand reputation [3].

*STT data* contains information regarding topics discussed w.r.t. time and location, hence presenting an invaluable link between user opinions and the real world. For example, STT data can help us analyze an advertisement campaign to identify the best locations for ad placements. Traditionally, this information is accessed through spatial keyword-queries [4], e.g., to retrieve topics within a specific location or identify in which locations some topic is discussed. However, keywords or topics search are *point-wise* search tasks. Instead, there is a significant need to *provide more extensive analytics analogous to traditional OLAP-style analytics*.

Consider the case of an analyst analyzing social media posts (e.g., Fig. 1). The analyst would collect a large number of such posts and focus on the three major pieces of information, namely: time, location, and the important keywords present in the text (as exemplified in Table 1). A typical STT analysis will then compare the number of posts that have been posted within a specific region and time window w.r.t. some keywords of interest. For instance, an example STT query is *"find the top-k trending hashtags aggregated by topic within the user-defined polygon around Paris this month"*. Such a query would allow the analyst to identify which topics users are currently talking about and use this for a marketing campaign, e.g., to identify what users mention in their plans for New Year's Eve.

The traditional data cube model is one of the most widely used tools to analyze structured data. Since their introduction, data cubes have been extended to analyze different types of data, like sales [5], locations [6], time-series [7], and text [8], but *separately*. In particular, some works propose OLAP operators to analyze either textual data [9,10] or spatial data [5,6].

However, no previous work proposed a unified model and set of operators enabling *integrated* and *joint* analysis of *STT data*. Moreover, as we propose to *jointly* analyze STT dimensions together with other dimensions, we are also able to define novel

* Corresponding author.
*E-mail addresses:* mohsin@cs.aau.dk (M. Iqbal), matteo@cs.aau.dk (M. Lissandrini), tbp@cs.aau.dk (T.B. Pedersen).

https://doi.org/10.1016/j.is.2022.102009

**Table 1**
Spatio-Textual-Temporal sample dataset.

| | Time | Location | Terms |
|---|---|---|---|
| 1 | 11:12:13 20-10-2019 | 57.016254, 09.991203 | Apple, fruit, love |
| 2 | 11:18:23 24-10-2019 | 56.187421, 10.171410 | Potato, #NewYear |
| 3 | 11:35:56 20-10-2019 | 56.151078, 10.204762 | Banana, Season |
| 4 | 16:12:14 24-10-2019 | 57.016254, 09.991203 | Potato, Salad, #Fresh |
| ... | ... | ... | ... |


**Fig. 1.** Geo-tagged tweet: An example of a STT object.

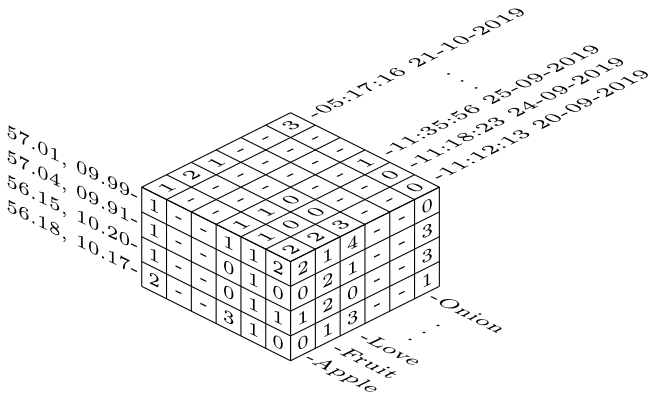



**Fig. 2.** STTCube example.

families of measures that have not been studied before, namely *STT measures*. As we show later, these measures allow to produce more advanced analytics instead of, e.g., simple keyword frequency.

*The STTCube.* In this paper, we introduce the Spatio-Textual-Temporal Cube (STTCube) to analyze *STT data*. *STTCube is* a *logical* multidimensional data model rather than a *conceptual* one. As such, it is focused on powerful and efficient OLAP querying of STTCube rather than high-level conceptual modeling. The core idea is to aggregate and analyze STT objects over the dimensions capturing the time, location, and textual aspects of the posts. In the *STTCube*, the cells contain the number of posts for a given time period/instant, location/region, and keyword (see Fig. 2 for a simplified example). Adding spatial, textual, and temporal support to a traditional data cube is not straight-forward due to the presence of *n-n* relationships in textual hierarchies (e.g., a post mentioning both "banana" and "carrot" will map to both the "Fruits" and "Vegetables" categories in the textual hierarchy'). and because existing families of measures cannot support *joint* and *integrated* analysis involving spatial, textual, and temporal dimensions, e.g., finding the trending keywords grouped by regions, defined by geometry shapes, over a time interval. Hence, we introduce new families of measures (STT Measures) and OLAP operators that extract *combined insights* from STT dimensions and measures. STTCube provides specialized *spatio-textual* and *spatio-textual-temporal measures* such as *Top-k Dense Keywords within an area* and *Top-k Volatile Keywords within an area* that deliver the integrated aggregates over *STT data*. Moreover, a set of analytical operators, namely STT slice, dice, roll-up, and drill-down, are proposed. This results in a data model able to support *spatio-textual-temporal OLAP* (STTOLAP) operators. An incremental maintenance method is also proposed to efficiently incorporate new data into the *already* constructed STTCube. Furthermore, we propose *Partial Exact Materialization (PEM)* and *Partial Approximate Materialization (PAM)* methods for efficient exact and approximate computations of *STT measures*, respectively. Among other things, we also provide a systematic set of solutions to handle $n-n$ relationships in textual hierarchies.

*Contributions.* In this work, we present the following contributions: (I) We extend the standard cube model to add support for *spatial, textual, and temporal* dimensions and hierarchies and *spatio-textual* and *spatio-textual-temporal* measures (Sections 3.1–3.3). (II) We propose a set of analytical operators (STTOLAP) over *spatio-textual-temporal data* (Section 4). (III) We introduce *keyword density* and *keywords volatility* as prototypical *spatio-textual* and *spatio-textual-temporal measures* (Section 3.3). (IV) We propose a pre-aggregation framework (STTCube materialization) for efficient, exact (*PEM*) and approximate (*PAM*), computation of the proposed *STT measures* (Section 5). (V) We propose techniques for processing *spatio-textual-temporal objects* and the construction of the *STT-Cube* (Section 6.1). (VI) We propose a novel incremental maintenance method for efficiently maintaining an already constructed STTCube (Section 6.2). (VII) We evaluate the pre-aggregation framework's (*PEM* and *PAM*) query response time, storage cost, and accuracy by comparing it with the *No STT Cube*, *Full Materialization*, and *No Materialization* baselines. Our pre-aggregation framework provides 1–5 orders of magnitude improvement in query response time and a 97% to 99.9% reduction in storage cost with an accuracy between 90% and 100% (Section 7). Furthermore, our proposed incremental maintenance method achieves an order of magnitude improvement over the baseline method.

## 2. Related work

OLAP and the *Data Cube* [21] are used heavily in business intelligence to obtain insights over the historical, current, and future state of business. With the emergence of the web and social media, an immense amount of unstructured data is being produced, which must be included in the analytical process.

Table 2 summarizes state of the art on spatial, textual, and temporal analytics by listing the properties and gaps in the current methods.

The *Text-Cube* [8,22] allows OLAP-like queries on text data by providing dimensions and hierarchies for terms. Moreover, it supports the computation of two information retrieval (IR) measures: *inverted index* and *term frequency*. *EXODuS* [11] processes semi-structured document stores (i.e., JSON) using a schema-on-read approach to allow exploratory OLAP on text. Text OLAP [12] extends traditional OLAP to support textual dimensions and keyword-based top-*k* search [13]. *Yet, all these approaches lack support for spatial and temporal data and the advanced measures and operators required for spatio-textual-temporal analytics.*

For spatial data, GeoMiner [6] proposes a cube structure for mining characteristics, comparisons, and association rules from geo-spatial data. The coupling of GIS and OLAP is known as Spatial OLAP (SOLAP) [23] and Spatial cube [5] allows to perform SOLAP on the semantic web. *Yet, these solutions focus on spatial data only and lack support for textual and temporal data.*

There are solutions that combine more than one component of data, e.g., spatio-temporal [24], into the same model but do not provide combined *STT* analytics. Among those, the contextualized warehouse [20] combines traditional OLAP with a textual

**Table 2**
Presence (✔) or absence (✘) of support for spatial and textual data, dimensions, hierarchies, and measures in existing methods.

| Method | Textual Support | | | | Spatial Support | | | | ST | STT |
|---|---|---|---|---|---|---|---|---|---|---|
| | Data | Dimension | Hierarchy | Measure | Data | Dimension | Hierarchy | Measure | Measure | Measure |
| EXODuS [11] | ✔(JSON) | ✘ | ✘ | ✘ | ✘ | ✘ | ✘ | ✘ | ✘ | ✘ |
| TextCube [8] | ✔ | ✔ | ✔ | ✔ | ✘ | ✘ | ✘ | ✘ | ✘ | ✘ |
| Text OLAP [12] | ✔ | ✔ | ✔ | ✘ | ✘ | ✘ | ✘ | ✘ | ✘ | ✘ |
| TextCubeTopKCells [13] | ✔ | ✔ | ✔ | ✔ | ✘ | ✘ | ✘ | ✘ | ✘ | ✘ |
| Geo Miner [6] | ✘ | ✘ | ✘ | ✘ | ✔ | ✔ | ✔ | ✔ | ✘ | ✘ |
| SpatialCube [5] | ✘ | ✘ | ✘ | ✘ | ✔ | ✔ | ✔ | ✔ | ✘ | ✘ |
| StreamCube [14] | ✔ | ✘ | ✘ | ✔ | ✔ | ✔ | ✔ | ✘ | ✘ | ✘ |
| TwitterSand [15] | ✔ | ✔ | ✔ | ✘ | ✔ | ✘ | ✘ | ✔ | ✘ | ✘ |
| TextStreams [16] | ✔ | ✘ | ✘ | ✔ | ✔ | ✘ | ✘ | ✘ | ✘ | ✘ |
| TopicExploration [17] | ✔ | ✘ | ✘ | ✔ | ✔ | ✔ | ✔ | ✘ | ✘ | ✘ |
| SocialCube [18] | ✔ | ✘ | ✘ | ✔ | ✔ | ✔ | ✔ | ✘ | ✘ | ✘ |
| TopicCube [19] | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✘ | ✘ | ✘ |
| ContextualizedWarehouse [20] | ✔ | ✘ | ✘ | ✔ | ✔ | ✔ | ✔ | ✘ | ✘ | ✘ |
| STTCube | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ |

warehouse. This allows the user to provide some keywords, select a market (country or region), retrieve documents matching the keywords as context, and then analyze the facts related to those keywords and documents. Similarly, *Topic Cube* [19] extends the functionality of a traditional cube and combines probabilistic topic modeling with OLAP by introducing the *topic hierarchy*. TwitterSand [15] and StreamCube [14] exploit textual and spatial information to gain insights by clustering twitter hashtags and tweets in a region, respectively. *STT data* is also analyzed to extract events and topics information in TextStreams [16] and TopicExploration [17]. Finally, SocialCube [18] tries to capture human, social, and cultural behavior by performing the linguistic analysis (sentiment analysis) over tweets. All these approaches focus on the unstructured nature of text along with spatial and temporal data. However, they *do not provide Integrated STT analytics*, for example, they do not provide the ability to *compute aggregate spatial, textual, temporal, and spatio-textual-temporal measures over spatial, textual, and temporal dimensions and hierarchies*.

When dealing with Data Cube design and implementation in general, it arises the issue on how to translate the conceptual model into a logical model. In our case, this issue arises especially for the case of textual dimensions, which require handling hierarchies with many-to-many relationships across members of different levels, e.g., a term belonging to different topics. *Multi-DimER* [25] models facts with different kinds of hierarchies and measures in a *conceptual* rather than logical multidimensional model. The gap between conceptual and logical models is also studied in a survey of summarizability [26] exploring the differences between a multidimensional conceptual model and its alternative logical representation. Furthermore, there exist studies in the literature that address the challenges of handling many-to-many relationships between fact and dimension tables [27]. These studies are not specific for the STT use case, i.e., they do not focus on spatial and textual hierarchies. Nonetheless, some considerations still applies. In particular, in our work we apply the snowflake schema with bridge tables for its flexibility, because it limits redundant information, and because it is a common and well established method [27]. Moreover, different from the general case studied in previous work, here we are the first to study specifically the effect of replication-based and majority-based methods when applied to the textual hierarchy on STT OLAP computations.

Spatial top-$k$ keyword-queries [2,28,29] *answer only point-wise* queries and do not support aggregation functions or hierarchies. Thus, they do not support more complex OLAP-style analytical tasks, which we do. There are methods that solve a very specific task for a specific type of data [30–32]. These methods are fundamentally different from STTCube because *STTCube provides a generic framework for a wide range of STT analytics over different kinds of STT data sources*, including, but not limited to, geo-tagged tweets. Also, STTCube can take advantage of the improvements suggested over other cubes, e.g., Nanocubes [33] and DICE [34], making it a powerful tool for OLAP-style *STT analytics*.

Our summary of related work in Table 2 shows that no existing method provides integrated support for *STT data*, unlike STTCube. To the best of our knowledge, a proper formalization of a data cube model for *STT data* able to support complex analytics for *STT objects* at scale is missing. In particular, no previous method studies dimensions, hierarchies, and measures that allow processing STT data *jointly*. Furthermore, the main novel challenge for *STT-OLAP* is handling $n - n$ relationships inside the *STT* dimensions effectively since $n - n$ relationships do not allow traditional pre-aggregation techniques to be used. Moreover, arbitrary temporal ranges with multiple levels of granularity add complexity to *STT* measures computations. As a remedy, we propose *STTCube* which enables the *joint* and *integrated* analysis of *STT objects* by introducing new sets of STT measures to gain in-depth insights using STTOLAP operators.

## 3. Spatio-textual-temporal cubes

Here, we define the *STTCube*, an extension of the traditional data cube to allow storage and analysis of *STT objects*. Data cubes are used to model and analyze multi-dimensional data. The basic building block of a data cube is the cell that contains *facts*. Each fact is the observational object for analysis, with one or more numerical *measures* associated with it, e.g., in a typical domain a fact can be a *Sales* transaction for a *Product* for which we store the *Quantity* and the *Total Sales Price* as measures. Facts in a cube are characterized by different *dimensions* that provide the context for analysis, e.g., *Product Category* and *Transaction Date*. Dimensions contain one or more *hierarchies* divided into *levels* to compute measure aggregations and perform analysis at various levels of detail. Each level contains multiple members, e.g., in the *Product Category* level, the members are the distinct product categories. The lowest level of each hierarchy is the fact dimension value (e.g., the specific *Transaction Date* or *Product Category*). In contrast, the highest level is a unique level *All* with just one value *all* to which all members belong as a single group. For instance, the *Time* dimension is always present in a data cube and usually has a hierarchy that groups transaction dates at the *Month*, *Quarter*, *Year*, and finally *All* transactions irrespective of their date levels. When moving up in a hierarchy, *aggregation functions* allow aggregating the measure values of lower-level cells into a single measure value in the upper cell. Lastly, each level has some associated *attributes*, which describe their members, e.g., the number of days in a month. By specifying a combination of dimensions,

hierarchies, and levels, we can identify one or more cells in the data cube and then analyze any of the measures for the facts contained in such cells.

**Definition 3.1** (*Data Cube*)**.** An *n-dimensional* data cube $\mathcal{CS}_{dc}$ is a tuple $\mathcal{CS}_{dc} = (D, M, F)$, with a set of dimensions $D = \{d_1, d_2, \ldots, d_n\}$, a set of measures $M = \{m_1, m_2, \cdots, m_k\}$, and a set of facts $F$. A dimension $d_i \in D$ has a set of hierarchies $H_{d_i}$. Each hierarchy $h \in H_{d_i}$ has a set of hierarchy steps (discussed in Section 3.1) and is organized into a set of levels $L_h$. Each level $l \in L_h$ contains a set of members and has a set of attributes $A_l$. Each attribute $a \in A_l$ is defined over a domain. Each measure $m \in M$ is a function defined over a domain that can return either a single value or a complex object. The domain of a dimension $d_i$ is denoted by $\delta(d_i)$

*Spatio-Textual-Temporal (STT) objects.* In this work, we consider data cubes to analyze Spatio-Textual-Temporal (STT) Objects. An *STT object* records place (geo-coordinates or location where it was created), text (a review or a user comment), and time (when it was created). Social networks with geo-tagged micro-blog posts are typical *STT data* sources (e.g., the geo-tagged tweet in Fig. 1).

**Definition 3.2** (*STT Object*)**.** A spatio-textual-temporal object is a tuple $obj_{st} = (\lambda, \varphi, \tau)$ where $\lambda$, $\varphi$, and $\tau$ are base level members of the location, text, and time dimensions, respectively.

*Location* is represented as the latitude and longitude pair $\lambda \in (\mathbb{R} \times \mathbb{R})$. *Text* is a bag-of-words $\varphi = \{w_1, w_2, w_3, \ldots, w_n\}$ where $w_i \in \mathcal{W}$ is a string and is called a *Term*. We use the common *bag-of-words*[35] model instead of vectors because the terms order in $\varphi$ does not matter. Among all Terms, keywords are a user-defined subset of important Terms $W_k \subseteq \mathcal{W}$. For instance, the user can decide that hashtags (terms starting with #) have special meanings and are a special type of keyword. *Time* specifies a precise instant (a timestamp) to some resolution (e.g., seconds). Table 1 contains examples of *STT objects* with their location, a set of keywords extracted from the text, and timestamp.

### 3.1. The STTCube schema

For analytical processing of *STT objects* we propose to model them as an *STTCube*. An STTCube $\mathcal{CS}_{stt} = (D, M, F)$ is a data cube (Definition 1) with three special dimensions, namely *Location*, *Text*, and *Time*, along with zero or more traditional dimensions, i.e., $D = \{d_{Location}, d_{Text}, d_{Time}, d_4, \ldots, d_n\}$. That is, the STTCube adds a set of new dimensions, which still adhere to the traditional definition of dimensions in the standard cube [5,21], but need special extensions. In the following, we will first provide the core definitions and then focus specifically on how the STT dimensions differ from traditional dimensions.

**Definition 3.3** (*Dimension*)**.** A dimension $d_i$ is a tuple $d_i = (L_{d_i}, \preceq, l_\uparrow, l_\downarrow)$, with a set of dimension levels $L_{d_i}$ having a special member *all*, a partial order $\preceq$ on $L_{d_i}$, and $l_\downarrow \in L_{d_i}$ and $l_\uparrow \in L_{d_i}$ being the bottom and top elements of the partial order, respectively.

*Dimensions.* Each dimension $d_i$ in the set of dimensions $D = \{d_1, d_2, \ldots, d_n\}$ has a set of hierarchies $H_{d_i}$. The domain of a dimension $d_i$ is denoted by $\delta(d_i)$. An *STTCube* stores *STT objects* as facts modeling their spatial, textual, and temporal features in the corresponding dimensions. Fig. 2 shows a 3-dimensional *STTCube* built on the sample dataset in Table 1 where each row represents one fact (i.e., the members of $F$) with dimensions $D = \{d_{Location}, d_{Text}, d_{Time}\}$. Domains for the respective dimensions are

$$\delta(d_{Location}) = \{(57.016, 09.991), (56.187, 10.171), \ldots\}$$
$$\delta(d_{Text}) = \{Apple, Fruit, Potato, \ldots\}$$
$$\delta(d_{Time}) = \{11:12:1320-10-2019, \ldots\}$$

Hence w.r.t. Definition 2, the dimensions capturing $\lambda$, $\varphi$, and $\tau$ are the spatial, textual, and temporal dimensions, respectively. STTCube supports one spatial, textual, and temporal dimension with the possibility of having multiple hierarchies for each.

*Dimension hierarchy.* A hierarchy is spatial, textual, or temporal if it contains spatial, textual, or temporal levels, respectively. In Fig. 2, the *Location* dimension is a spatial dimension with a spatial hierarchy going from $\lambda$ to *City, Region*, and *Country* and the *Text* dimension is a textual dimension aggregating $\varphi$ into the *Term, Theme, Topic*, and *Concept* levels. Similarly, *Time* is a temporal dimension. Hierarchy steps $HS_h = \{hs_1, hs_2, hs_3, \ldots, hs_n\}$ define the mechanism of moving from a lower (child) level to an upper (parent) level and vice versa. A hierarchy step $hs_i = (l_\downarrow, l_\uparrow, cardinality) \in HS_h$ entails that members of a child level $l_\downarrow$ can be aggregated together if they correspond to the same member at the parent level $l_\uparrow$ and that this correspondence between children to parent members has the given *cardinality* $\in \{1-1, 1-n, n-1, n-n\}$. For instance, the step from *Date* to *Month* has an $n-1$ cardinality, while *Term* to *Topic* has an $n-n$ cardinality (e.g., the Carrot *Term* correspond both to the Gardening and Food *Topics*, while the Food *Topic* has as child members not only Carrot but also Apple).

*Level attributes.* As mentioned earlier, a level $l$ is associated with a set of attributes $A_l = \{a_1, a_2, \ldots, a_n\}$ and has a set of members $l = \{l_1, l_2, \ldots, l_3\}$. Attribute values describe the different characteristics of each member from that level. Spatial, textual, and temporal levels are then usually characterized by spatial, textual, and temporal attributes. For instance, at the *City* level, all members have the *Boundary* attribute whose value is the polygon defining the boundary of the respective city. An example of a textual attribute is *Sentiment* which captures the polarity of the associated textual member. Similarly, an *integer value* representing the number of days in a specific month is a temporal attribute. Note that both measures and filtering conditions in STT OLAP operators (described later) can perform complex operations that compute formulas that use a combination of level attribute values and measure values.

### 3.2. Managing STT hierarchies

We now describe the STTCube's dimensions and hierarchies. For now, *STTCube* supports balanced hierarchies only; thus, imbalanced hierarchies are out of the scope of this work and left for future work.

*Spatial dimensions.* Spatial information can be analyzed at different levels and granularities. It is important to note that facts in an STTCube are composed only by geographical *points* (i.e., each tweet or user post is associated with a coordinate, not with shapes or polygons). Points can be aggregated either within a predefined spatial grid or based on semantic information.

*Grid-Based Hierarchy.* Here, the geographic area being analyzed is divided into small equal size cells with a predefined resolution, e.g., $1 \times 1$ km$^2$. At the lowest level, each latitude and longitude point is assigned to the cell they fall in. To analyze data at a coarser granularity, neighboring cells are combined into a larger cell at the parent level (e.g., $3 \times 3$ km$^2$). This hierarchy's computation is application-specific and is computed as per the requirements. This hierarchy can be built automatically, without the need for any meta-data.

*Semantic-Based Hierarchy.* Here, data is analyzed in a predefined taxonomy, e.g., an administrative division. Therefore, we move within the taxonomy, e.g., from the *Location* to the *City* level, from the *City* level to the *Region* level, and so on up to

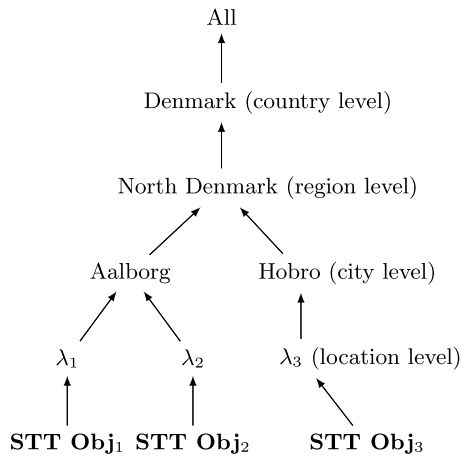



**Fig. 3.** Example of semantic-based spatial dimension hierarchy.

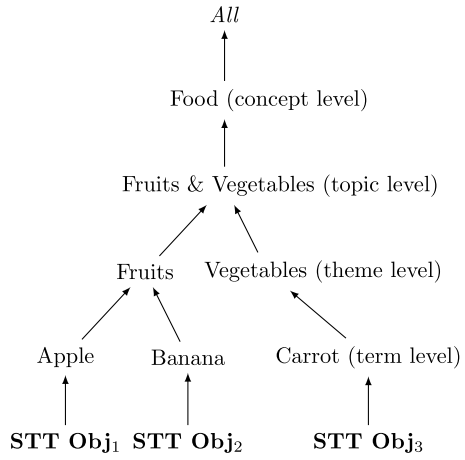



**Fig. 4.** Example of replication-based textual dimension hierarchy.

the *All* level. This hierarchy requires each object coordinate to be associated with a member in the lowest level in the hierarchy (usually in a pre-processing step) and requires the taxonomy information to build the entire hierarchy.

Fig. 3 shows the semantic-based spatial hierarchy constructed for three *Location points* $\lambda_1$, $\lambda_2$ and $\lambda_3$ belonging to the respective STT Objects.

*Textual dimensions.* Hierarchies in the textual dimension move from specific concepts to general ones. This follows a generic taxonomic structure connecting more specific terms to more general ones (i.e., hypernyms) [36]. Textual hierarchies are implemented using WordNet [37] which is discussed in Section 7. In particular, *Terms* are the base level which are grouped into *Themes*, *Themes* into larger categories called *Topics*, and *Topics* in turn grouped into *Concepts*. *Differently from most hierarchies, the members in the levels of a textual hierarchy are typically in an n−n relationship.* Hence, when moving between textual levels, we need to decide how measure values get aggregated. Below we propose a set of aggregation techniques to address this issue.

*Replication-Based Hierarchy.* This is a common approach where each member of a child level is aggregated into all the parent members. Hence, its value is effectively replicated. This approach leads to a *counting problem* when parent levels are further aggregated. For example, the first data instance in Table 1 will be part of two *Themes*: 1) Fruits because it contains *Term* {apple and fruit} and 2) Emotion because of *Term* {love}.

*Majority-Based Hierarchy.* If a fact can be mapped to more than one parent member, then that fact will be part of the parent member with the most representation (e.g., in terms of frequency). This scheme avoids double counting of facts in parent members. In case of ties, some tie-breaking heuristic or a user-defined criterion can be employed instead, e.g., the first fact in Table 1 will be part of only the Fruits *Theme* because it has the two representative *Term* {apple, fruit}, as compared to Emotion having only one *Term* {love}.

*Custom Hierarchy.* In general, other user-specified criteria and rules can be defined to establish how child–parent level steps will be aggregated in case of ambiguities. For instance, a domain-specific *importance score* can be assigned to the hierarchy members during the *STTCube* construction. In this way, facts will be part of only the parent member with the highest importance.

Fig. 4 shows the textual hierarchy constructed for an STT object containing the terms apple, banana, and carrot.

*Temporal dimensions.* Similarly, temporal dimension allows to analyze *STT objects* at different levels of granularity w.r.t. time and has the following two temporal hierarchies: $\tau \rightarrow$ *Day* $\rightarrow$ *Month* $\rightarrow$ *Quarter* $\rightarrow$ *Year* $\rightarrow$ *all* and $\tau \rightarrow$ *Second* $\rightarrow$ *Minute* $\rightarrow$ *Hour* $\rightarrow$ *all*. Here, the first contains a hierarchy of *Date* aggregated by the temporal levels *Day, Month, Quarter*, and *Year* (total 5 levels including *All*), whereas the second is a hierarchy for *TimeOfDay* having 4 levels in total.

### 3.3. Spatial, textual, and temporal measures

As defined earlier, an *n-dimensional STTCube* has a set of measures $M=\{m_1, m_2, m_3, \ldots, m_k\}$, which permit to analyze *STT objects* by computing values at different levels of granularity. For instance, the *STTCube* in Fig. 2 models *Location*, *Text*, and *Time* with *Fact Count* as a measure (i.e., *Fact Count*$\in M$). In practice, it maintains the count of *STT objects* at given spatial, textual, and temporal aggregation levels. We have tweets at the base level of the *STTCube*. Measure values are linked to the base levels only, e.g., to the *Location* level. No facts are linked directly to higher levels, e.g., to the *City* or *Country* levels. Measure values at different levels in the hierarchies are obtained by applying an aggregation function over the *STT objects*. Examples of aggregation functions are *SUM*, *COUNT*, *MIN*, *MAX*, and *AVG*. The *STTCube* in Fig. 2 uses *COUNT* as an aggregation function. For example, it reports that on *September, 20th* at *AAU Bus Terminal* the *Term apple* was mentioned in 2 facts.

A measure is spatial if it is defined over a spatial domain. A spatial measure is then computed over a collection of spatial values (e.g., geographical points or geometry shapes like polygons). A spatial measure can be a simple value, e.g., the (numeric) area of the convex hull of multiple shapes, or a complex spatial object, e.g., the polygon representing the convex hull itself. A measure is textual if it is defined over a textual domain and can be either a simple numeric value or a complex textual object. Analogously, a measure is temporal if it is defined over a temporal domain, A measure is spatio-textual if it is defined over a spatial and textual domain and is a combination of spatial and textual measures. Finally, a measure is spatio-textual-temporal if it is defined over a spatial, textual, and temporal domain and is a combination of spatial, textual, and temporal measures. Note that, to compute some STT measures, we are effectively computing formulae that use both measure values, e.g., the number of facts, as well as attributes values of STT dimension members (e.g., the area of a polygon for a region). Below, we propose a list of *spatio-textual and spatio-textual-temporal measures* to be used as part of *STTCube* to analyze *STT objects* effectively.

*Keyword locations.* is a spatio-textual measure which returns a list of $(\xi_i, w_j)$ tuples pairing a keyword $w_j$ (i.e., a textual object) to the geographical locations $\xi_i$ (at the current level in the spatial hierarchy) where it appears. For instance, computing keyword locations at the *City* level describes in which *City* each keyword is discussed.

*Top-k keywords within an area.* is a spatio-textual measure which returns a list of tuples $(\xi, \overrightarrow{kw})$ consisting of a geometry shape $\xi$ representing a geographical area and the list of top-$k$ most frequent keywords $\overrightarrow{kw} = [w_1, w_2, \ldots, w_k]$ in that area. Analogous to previous measures, it can also be computed at different levels of aggregation so that it can return the top-k keywords for each *City* or each *Region*.

*Keyword density.* is a spatio-textual measure which returns a list of tuples $(\xi_i, w_j, \rho_{ij})$ consisting of a geometry shape $\xi_i$ representing a geographical area, a keyword $w_j$, and its density $\rho_{ij}$ in the area $\xi_i$. The density $\rho_{ij}$ of a keyword $w_j$ over an area $\xi_i$ is computed as $\rho_{ij} = \frac{freq(\xi_i, w_j)}{SurfaceArea(\xi_i)}$, in which $freq(\xi_i, w_j)$ is the frequency of the keyword $w_j$ in the area $\xi_i$ (i.e., the number of objects located within $\xi_i$ in which $w_j$ appears) and *SurfaceArea* is the surface area of $\xi_i$. For example, if we have two *Regions* $r_1$, $r_2$ with $SurfaceArea(r_1) = 10\ m^2$, $SurfaceArea(r_2) = 100\ m^2$, and the term *Apple* with frequency 5 and 30 in $r_1$ and $r_2$, respectively (see Fig. 5), then, keyword densities are $\rho_1$=0.5, $\rho_2$=0.3 for $r_1$ and $r_2$, respectively.

*Top-k dense keywords within an area.* is a spatio-textual measure which returns a list of tuples $(\xi_i, \overrightarrow{kw})$ computing the keyword density as described in the measure above, but in this case, it returns the top-k keywords $\overrightarrow{kw} = [w_1, w_2, \ldots, w_k]$ with the highest density.

*Keyword volatility.* is a spatio-textual-temporal measure (becomes textual-temporal If no region is specified) which returns a list of tuples $(\xi_i, w_j, T_k, \Delta\rho_{ijk})$ consisting of a geometry shape $\xi_i$ representing a geographical area, a keyword $w_j$, a time interval $T_k$, and its change in density $\Delta\rho_{ijk}$ in the area $\xi_i$ over the time interval $T_k$ (divided into $k$ equal intervals). The change in density $\Delta\rho_{ijk}$ of a keyword $w_j$ in an area $\xi_i$ over a time interval $T_k$ is computed as $\Delta\rho_{ijk} = \frac{\sum_{z=1}^{k} |\rho_{ij_z} - \rho_{ij_{z-1}}|}{k}$, where $\rho_{ij_z}$ represents the density of the keyword $w_j$ in the area $\xi_i$ at a specific time instance $T_{k_z}$. Furthermore, the change in the density computation formula can be updated depending on the analysis requirements, e.g., it can be changed to weighted density (assign different weights to each interval in $T_k$) or to rate of change computation using linear regression [38].

*Top-k volatile keywords within an area.* is a spatio-textual-temporal measure which returns a list of tuples $(\xi_i, \overrightarrow{kw})$ computing the keyword volatility as described above, but in this case, it returns the top-k volatile keywords $\overrightarrow{kw} = [w_1, w_2, \ldots, w_k]$ with the highest change in density.

*Distributive, algebraic, and holistic measures.* There are three types (also known as additivity) of measures: distributive, algebraic, and holistic, depending on whether it is possible to compute the value of a measure at a parent level directly from the values at the child level [39]. For distributive and algebraic measures, this is possible. For instance, the *Fact Count* at the *State* level can be computed by summing up the *Fact Counts* at the *City*. *Keyword Density* is instead an algebraic measure. We can compute the higher-level aggregate values of this measure if we store for each child level both the frequency of each keyword and the *SurfaceArea*. The *Top-k Keywords*, the *Top-k Dense Keywords*, and *Top-k Volatile keywords within an area* measures, instead, are holistic, since the value at a parent level cannot be computed directly from the values at the child level, but it is necessary to recompute them directly from the base facts every time.

Consider the computation of *Top-3 Dense Keywords within an Area* in Fig. 5 given the two *Regions* $r_1$ and $r_2$ with *SurfaceArea* 10 $m^2$ and 100 $m^2$, respectively, and the computation at the parent level $r_3$=$r_1 \cup r_2$ (grayed-out rows are not part of the computed measure value). The values in the top-3 for the members $r_1$ and $r_2$ at the child level are not sufficient to compute the correct densities for region $r_3$. Both, some of the computed density (in column $\rho_{Top-3}$, while the correct values are reported in $\rho_{all}$) and consequently the final ranking, would be wrong. For instance, the keyword *Strawberry* would not have been returned (if computed algebraically) because it is neither in the top-3 for $r_1$ nor $r_2$. To compute the correct response, either we have to store all the aggregate values for each possible cell or we have to reprocess all the facts covered by the query. When dealing with large datasets these approaches are not feasible. Hence, in Section 5 we provide a framework for the computation of an exact and approximate solution with accuracy guarantees.

## 4. STTOLAP operators

A data cube allows different **O**n**L**ine **A**nalytical **P**roce-ssing (OLAP) operators to group, filter, and analyze cells and subsets of cells at different levels of granularity and under different perspectives. Those operators are known as *Slice*, *Dice*, *Roll-Up*, and *Drill-Down* [21], and they take as input a cube and produce as output another cube. In the following, we extend the basic OLAP operators [40] to *STTOLAP operators*, i.e., for spatial, textual, and temporal dimensions, hierarchies, and measures. In general, an OLAP (and STTOLAP) operator $OP(C', params)$=$C''$ accepts as input a cube $C'$=$(D', M', F')$, some parameters *params*, and outputs a new cube $C''$=$(D'', M'', F'')$. In this way, a new OLAP operator can be applied to $C''$. Among all cubes, we distinguish the initial or *base cube* $C$ as the cube containing all the original information at the base level. In the following, we generally assume every OLAP operator *OP* to have access to $C$ (hence, we will not explicitly show it in the signature of the operators) since some operators need access to the base cube $C$, and not only to $C'$, to produce the desired result.

### 4.1. STT-slice

Slice operates over the current data cube $C'$ and given a dimension member $v_i$ for dimension $d_i$, it keeps only cells in $C'$ corresponding to $v_i$ while removing the dimension $d_i$.

**Definition 4.1** (*STT-Slice*)**.** The STT-Slice operator is defined as $STTSlice(C', v_i) = C''$. It takes an $n$-dimensional *STTCube* $C'$ and a member $v_i$ of the level $l_i$ of a dimension $d_i \in \{d_{Location}, d_{Text}, d_{Time}\}$, i.e., either the spatial, textual, or temporal dimension, and produces a resulting cube $C''$ with $n-1$ dimensions, i.e., the resulting cube is $C''$=$(D'', M'', F'')$ where, $M'' = M'$, $D'' = D' \setminus \{d_i\}$, and the facts in $F''$ are new STT objects obtained from $F'$ by selecting only those that are associated to $v_i$,. Hence, it returns measure values, aggregated at the current level $l_i$ for dimension member $v_i$ only and removes the dimension of $v_i$.

The member $v_i$ could be an object in the taxonomy of a semantic-based hierarchy or could be a grid cell at some granularity level. An example of the slice operator is "slice location dimension on user-defined polygon representing Aalborg", where "Aalborg" is a dimension member in $l_{City}$. Similarly, $v_i$ could be a specific *Theme* or *Topic* for the textual dimension and a specific *day* or *month* for the temporal dimension.

Region $r_3 = r_1 \cup r_2$

| Keyword | $\Sigma_{all}$ | $\Sigma_{Top-3}$ | Area | $\rho_{all}$ | $\rho_{Top-3}$ |
|---|---|---|---|---|---|
| Carrot | 42 | 40 | 110 $m$ | 0.38 | 0.36 |
| Apple | 35 | 35 | | 0.32 | 0.32 |
| Strawberry | 22 | 00 | | 0.20 | 0.00 |
| Banana | 20 | 20 | | 0.18 | 0.18 |
| Orange | 16 | 05 | | 0.15 | 0.05 |
| Potato | 04 | 04 | | 0.04 | 0.04 |

Region $r_1$

| Keyword | Count | Area | Density |
|---|---|---|---|
| Apple | 5 | 10 $m$ | 0.50 |
| Orange | 5 | | 0.50 |
| Potato | 4 | | 0.40 |
| Strawberry | 3 | | 0.30 |
| Carrot | 2 | | 0.20 |

Region $r_2$

| Keyword | Count | Area | Density |
|---|---|---|---|
| Carrot | 40 | 100 $m$ | 0.40 |
| Apple | 30 | | 0.30 |
| Banana | 20 | | 0.20 |
| Strawberry | 19 | | 0.19 |
| Orange | 11 | | 0.10 |

**Fig. 5.** Example: Merging of holistic measure.

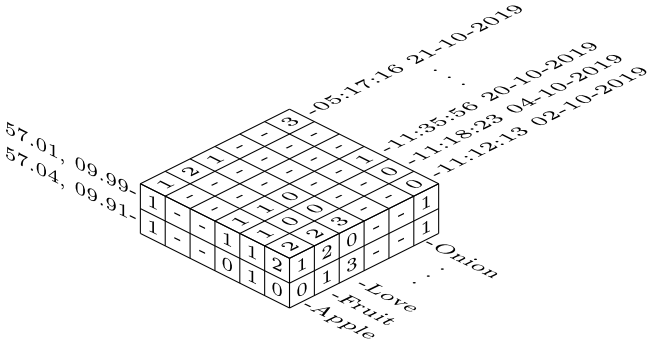



**Fig. 6.** Dice on *Latitude* ≥ 57.00 **and** *date* ≥01-10-2019.

### 4.2. STT-dice

While the slice operator selects and removes a single dimension, the dice operator produces a new cube whose cell contents have been filtered based on a set of conditions (e.g., complex predicates or queries covering several cells) but without removing any dimension. That is, it produces a resulting cube with the same number of dimensions but considering only the facts that satisfy the provided set of conditions. Such conditions can also use a combination of spatial, textual, temporal, and general-purpose functions. These functions can perform different computations, e.g., filter objects based on specific conditions, or compare two objects and return a Boolean value, or they can produce a numeric value based on some computation that is then used for filtering.

**Definition 4.2** (*STT-Dice*)**.** The *STT-Dice* operator is defined as $STTDice(C', COND_i) = C''$, where $COND_i$ is a set of atomic or compound spatial, textual, or temporal logical conditions. Then, we have that $C''=(D'', M'', F'')$ where, $M'' = M'$, $D'' = D'$, but we have that $F''\subseteq F'$ contains only the STT objects satisfying $COND_i$. Thus, STT-Dice selects only cell(s) from $C'$ that satisfy the provided spatial, textual, or temporal logical condition $COND_i$ and produces a resulting cube $C''$ with all the dimensions already present in $C'$ but restricted to only a subset of the facts in $C'$.

Two important characteristics of the STT-Dice operator are that (1) the filtering condition $COND_i$ often is complex, and (2) the filtering condition can exploit both attributes of dimension members (e.g., the polygon of a region in the spatial dimension) and aggregate values of measures (e.g., the number of tweets) when filtering. For instance, an STT-Dice operator can select the cell(s) that intersect with a user-provided polygon describing a custom region of interest and containing at least $n$ observations, or alternatively, the cells with at least 10 terms and relevance score for the *Food* topic above 0.7. Fig. 6 shows the resultant STTCube when we dice with *COND latitude* ≥ (57.00) **and** *date* ≥*(01-10-2019)* on the STTCube shown in Fig. 2.

### 4.3. STT-roll-up

The Roll-Up operator aggregates measure values along a hierarchy by moving from a child level to a parent level. This allows to move the analysis to a coarser granularity. The *STT-Roll-Up* operator groups facts by aggregating the measure values of all child members that belong to the same parent member in a spatial, textual, or temporal hierarchy.

**Definition 4.3** (*STT-Roll-Up*)**.** The *STT-Roll-Up* operator is defined as $STTRollUp(C', l_{i\downarrow}, l_{i\uparrow}) = C''$, where, given a STTCube $C'$, a child level $l_{i\downarrow}$ and a target parent level $l_{i\uparrow}$ (identifying a hierarchy step function $hs_i$ in the spatial, textual, or temporal dimension $d_i \in \{d_{Location}, d_{Text}, d_{Time}\}$), the output cube $C''=(D'', M'', F'')$ is a new STTCube that has the same dimensions of $C'$, i.e., $|D'| = |D''|$, but where the new set of facts $F''$ is obtained from $F'$ by grouping them based on the hierarchy step function $hs_i$, and by applying, for each measure $m \in M$, the aggregation function associated with $m$ to create aggregated measure values for a new set of STT Objects for each grouping of $F'$ at the higher parent level $l_{i\uparrow}$.

For instance, when we *Roll-Up* from *City* level to the *Region* level, *Fact Count* values get summed to compute the new total. Similarly, "Roll-Up to Topic level from Theme level" groups facts by aggregating the measure values of all *Themes* that belong to the same *Topic* and "Roll-Up to Quarter level from Month level" groups facts by aggregating the measure values of all *Months* that belong to the same *Quarter*. Fig. 7 shows the resultant STTCube when we perform roll-up to City and Theme levels on the STTCube shown in Fig. 2.

### 4.4. STT-drill-down

The inverse of the Roll-Up is the Drill-Down operator, which shows data at finer granularity by dis-aggregating measure values along a hierarchy. The base cube is required for this operation as we cannot uniquely dis-aggregate measure values knowing only the values at the parent level.

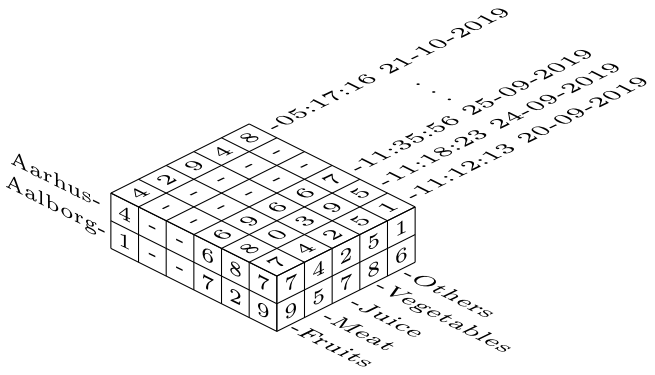



**Fig. 7.** Roll-Up to City & Theme levels.

**Definition 4.4** (*STT-Drill-Down*)**.** The *STT-Drill-Down* operator is defined as $STTDrillDown(C', l_{i\uparrow}, l_{i\downarrow}) = C''$, where, given an STTCube $C'$ at a level $l_{i\uparrow}$ for the spatial, textual, or temporal dimension $d_i \in \{d_{Location}, d_{Text}, d_{Time}\}$, and a target child-level $l_{i\downarrow}$ (identifying a hierarchy step function $hs_i$), produces a new STTCube $C'' = (D'', M'', F'')$ at finer granularity, with the same dimensions as $C'$, i.e., $|D'| = |D''|$, and where the new set of facts $F''$ is obtained from $F'$ by grouping the facts of the base cube $F$ at the selected level $l_{i\downarrow}$ and computing for every measure $m \in M$ the measure values aggregated along the selected spatial, textual, or temporal hierarchy, respectively.

For instance, *STT-Drill-Down* can move from the *Region* level to the *City* level dis-aggregating the *Fact Count* of each *Region* in the counts for each *City* in that *Region*. Similarly, "Drill-Down to Term from Theme", following the inverse textual hierarchy step *Term* ←*Theme*, and "Drill-Down to Day from Year" following the inverse temporal hierarchy step *Day*←*Year* are examples of STT-Drill-down operators for textual and temporal dimension, respectively. Note that, in this case, while the new values are dis-aggregated from the parent level to the lower level, they are necessarily computed by aggregating from the base cube since aggregations are non-reversible.

### 4.5. OLAP vs STTOLAP

The above operator definitions show that the main novel challenge for *STTOLAP* is handling $n-n$ relationships inside the dimensions effectively since $n-n$ relationships do not allow traditional pre-aggregation techniques to be used. Based on the type of hierarchy, we use hierarchy-specific computation, as explained in Section 3.2. Furthermore, arbitrary temporal range with multiple levels of granularity adds complexity to *STT* measures computations. As a remedy, we propose multiple strategies to manage these $n-n$ relationships (Section 3.2), handle arbitrary time intervals (Section 5.4), and show their efficiency and effectiveness in Section 7.

## 5. Cube materialization

Here, we describe the cube materialization methods, the cost model for STTCube materialization and incremental maintenance, and pre-aggregation framework for pre-computing the STT measure values.

### 5.1. Cube materialization methods

Cube materialization is the process of pre-aggregating measure values at different levels of granularity in the cube to compute query responses from pre-aggregated results instead of the raw data, and hence improve query response time for *STTOLAP operators* [41]. In a data cube, a *cuboid* is a collection of *level members* and associated *measure values* for a unique combination of dimension hierarchy levels. Each unique combination is represented by a separate cuboid (Fig. 8). Any cuboid in the path going down the hierarchy towards the base level (DLT in Fig. 8) from a specific cuboid is called an *ancestor* cuboid of that specific cuboid. Conversely, any cuboid in the path going up the hierarchy towards the *all* level (⋆ in Fig. 8) from a specific cuboid is called a descendant cuboid of that specific cuboid. For instance, if we request the *Fact Count* for the *State* of Denmark and have stored *Fact Count* at the *Region* level, we can avoid accessing the raw data and compute the aggregation from much fewer rows. This is an example of *partial materialization*, i.e., the actual cuboid at the *State* level, containing the answer to the query, was not materialized, but the system was still able to exploit the cuboid for *Region*.

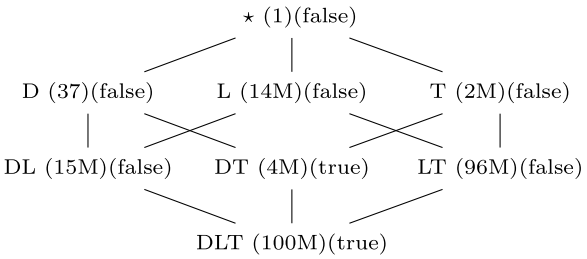



**Fig. 8.** Spatio-Textual-Temporal lattice.

What to materialize and how much to materialize depends on the trade-off between query response time and storage cost. *Full Materialization (FM)* is obtained by pre-computing measure values for *all* combinations of levels in *all* hierarchies. This approach requires massive storage but achieves the best query response time since every operation can just look up already pre-computed results. At the other extreme, *No Materialization (NM)* only materializes the base cuboid and does not require any extra space but will require aggregated measure values to be recomputed from the base cuboid every time, hence incurring much slower response times. A middle-ground solution is to partially materialize the cube, i.e., to materialize only some of the possible cuboids. In this strategy, some queries will be able to exploit pre-aggregated values at the current level, while other queries can exploit pre-aggregated values at lower levels for distributive or algebraic measures.

**Example.** Consider a simple lattice of *cuboid* for a cube with 3 dimensions (Fig. 8), each with a single 2-level hierarchy, namely with base levels Location (L) with 14M rows for the spatial dimension, Term (T) with 2M rows for the textual dimension, and Date (D) with 37 rows for the temporal dimension, each of which can be then rolled up to the *all* level (⋆) with only one row. Each node in the lattice is associated with two values; first, the *number of rows* in the cuboid, and second a *flag* (true/false) to mark if the current cuboid is materialized. At the bottom of the lattice, we have the base cuboid (which is always materialized) with Date, Location, and Term (DLT) containing in this example all rows (100M). If we Roll-Up the spatial dimension from Location to All, we would obtain a new cuboid (DT) with 4M rows. The cuboid DLT is referred to as the *ancestor* of the cuboid DT. If the cube is partially materialized, i.e., not all cuboids are materialized, and the cuboid DT is materialized, then to obtain the *Fact Count* for every Date and Term, the cuboid DT with 4M rows would contain the answer already pre-computed without the need to compute such an answer from the base cube DLT with 100M rows. Moreover, when the cuboid $T$ is not materialized, we can still compute the *Fact Count* for every Term from the cuboid DT by accessing only 4M values instead of the 100M in DLT.

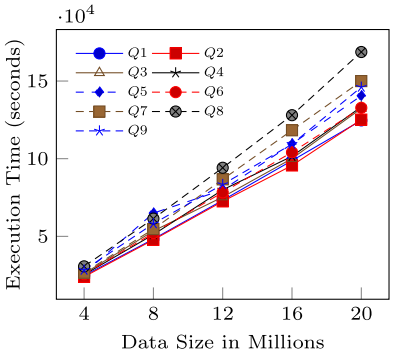



**Fig. 9.** QRT Vs data size.

**Table 3**
Spatio-Textual-Temporal queries.

| Query | Description |
|---|---|
| Q1 | Top-*k* Dense/Volatile *Terms* in a *City [time span]* |
| Q2 | Top-*k* Dense/Volatile *Topics* in a *City [time span]* |
| Q3 | Top-*k* Dense/Volatile *Concepts* in a *Country [time span]* |
| Q4 | Top-*k* Dense/Volatile *Terms* in a *Region [time span]* |
| Q5 | Top-*k* Dense/Volatile *Concepts* in a *Region [time span]* |
| Q6 | Top-*k* Dense/Volatile *Themes* in a *Region [time span]* |
| Q7 | Top-*k* Dense/Volatile *Terms* in a *Country [time span]* |
| Q8 | Top-*k* Dense/Volatile *Terms* in a *Country* Group by *Region [time span]* |
| Q9 | Top-*ALL* Dense/Volatile *Topics* in a *Country* Group by *Region [time span]* |

### 5.2. Cost model

The core of the proposed *partial materialization* approach depends on the trade-off between the storage cost of materializing any particular cuboid and the actual *benefit* that the materialization of the cuboid provides. To evaluate this benefit, we have to estimate the (run time) cost of a query. To devise a cost model for this estimation, we performed a micro-benchmark: we selected a set of representative queries (Q1–Q9, details in Table 3) for the aforementioned *spatio-textual-temporal measures* and measured the run time of these queries on increasing data sizes. The micro-benchmark (Fig. 9) confirmed that the running time is directly proportional to the data size (the number of rows), i.e., it confirms that we can use the Linear Cost Model [41] (most widely used cost model for cube materialization) and the associated benefit calculation. In Section 6.2 we will discuss STTCube incremental maintenance in detail and present a maintenance algorithm that allows us to minimize the STTCube maintenance cost, such that we can avoid considering such cost during the selection of the views to be materialized. Then, to model the dependency relationships among all the possible cuboids, we use the lattice framework [41] (Fig. 8). Hence, to compute the benefit of materializing a particular cuboid $c$, we need to compare the cost of answering queries at all levels of granularity (i.e., for the current cuboid $c$ and all its descendants in the lattice) with the current set of materialized cuboids against the cost when $c$ is also materialized.

$$Benefit(c) = \sum_{c' \in descendants(c) \cup \{c\}} cost(c') - size(c)$$

For instance, assume the lattice in Fig. 8 and that only the base cuboid (DLT) with 100M rows is materialized. If we consider DT with 4M rows, we have that, if materialized, queries against the cuboids D, T, $\star$, and DT itself can be answered through it (with a cost of 4M). In contrast, without materializing DT, we will need to compute the answer against DLT (with a cost of 100M). Hence, materializing DT will achieve a benefit of $(100 - 4) * 4 = 384$M. Whereas, materializing LT with 96M rows does not achieve a significant benefit ($(100 - 96) * 4 = 16$M).

We adopt this chosen linear cost model and extend the Greedy algorithm approach [41] to our task (Algorithm 1). *Additionally, and different from [41], Algorithm 1 accepts an input parameter K and materializes only the top-K measures values in each cuboid.* We discuss the top-$K$ value estimation in Section 7. For instance, for $K = 10$, it will materialize the top-10 keywords in each cuboid. Then, any top-k query, with $k \leq K$, for a materialized cuboid will return the pre-computed answer. In case of *Majority-based* hierarchies, $n - n$ relations in the hierarchies have no impact on the Greedy algorithm, whereas, for *Replication-based* hierarchies, the Greedy algorithm needs to be specialized to correctly compute the values when elements are replicated.

Algorithm 1, given a size budget $B$ (measured in rows, cuboids, or GB), proceeds until the size of the current cube is as large as possible within the budget (Line 6). At each step, it selects among all the non-materialized cuboids (Line 3) the one with the highest benefit (Line 4) and materializes it (Line 5). The difference between the exact (PEM) and approximate (PAM) materialization using Algorithm 1 is the value of $K$. When $K{=}\infty$ the complete sorted list of measure values will be stored so that all top-$k$ queries can be answered for that cuboid. We set $K{=}\infty$ and $K{=}n$ (to materialize only top $n$ measure values) for *PEM* and *PAM*, respectively.

### 5.3. Partial exact materialization

We propose a partial exact materialization technique for precomputing the *spatio-textual* and *spatio-textual-temporal measure* values. To answer an *STT query* for these measures we materialize two other distributive measures, namely *Keyword Frequency* $f$ and *SurfaceArea* $a$. Then, since *Keyword Density* $\rho$ and *Keyword Volatility* $\Delta\rho$ are algebraic measures, they can be computed from the values of *Keyword Frequency* $f$ and *SurfaceArea* $a$. Finally, *Top-k Dense Keywords* and *Top-k Volatile keywords* are holistic but for an exact solution we materialize *Top-ALL* and hence, compute it from the materialized measure values (Fig. 5).

**Algorithm 1:** Greedy Materialization

```
1 GreedyMaterialization (B, 𝕆, K)
    Input: Budget B, STTCube 𝕆, desired top-k K
    Output: Partially Materialized STTCube 𝕆'
2   do
3     Candidates ← {V∈𝕆 | ¬V.isMaterialized};
4     V̄ ← max_{V∈Candidates} Benefit(V);
5     𝕆.materialize(V̄, K);
6   while size(𝕆)≤B;
7   return 𝕆';
```

**Query rewriting.** Finally, as in [41], after *STTCube* materialization, queries are still formulated in terms of the base cuboid but rewritten by the system to be evaluated over the smallest cuboid.

**Algorithm 2:** Top-K Dense Keywords within an Area

1 **TopKDense** ($\Phi = \{(\xi_1, \overrightarrow{kw_1}), (\xi_2, \overrightarrow{kw_2}), \ldots, (\xi_n, \overrightarrow{kw_n})\}$, $k$)
  **Input:** Set of Top-k+1 Dense Keywords lists $\Phi$, Integer k
  **Output:** $(\xi, \overrightarrow{kw})$ top-k keywords $\overrightarrow{kw}$ in the merged area $\xi$, $\delta$ number of guaranteed top positions
2 $\xi \leftarrow \bigcup_{i\in[1,n]} \xi_i$ ; // Merge areas
3 $A \leftarrow$*SurfaceArea*($\xi$);
4 $\overline{kw} \leftarrow \{\}$ ; // Empty dictionary
5 **foreach** $(\xi_i, \overrightarrow{kw_i}) \in \Phi$ **do**
6  **foreach** $j \in [1, \ldots, k{+}1]$ **do**
7  $w \leftarrow \overrightarrow{kw_i}$.*get*($j$); // keyword at $j$
8  $f \leftarrow \overrightarrow{kw_i}$.*freq*($j$); // frequency at $j$
9  **if** $w \notin \overrightarrow{kw}$ **then**
10  $\overline{kw}[w] \leftarrow f$;
11  **else**
12  $\overline{kw}[w] \leftarrow \overline{kw}[w] + f$;
13  $\epsilon$ += $\overrightarrow{kw_i}$.*freq*($k+1$);
14 $\overrightarrow{kw} \leftarrow$*topK*($\overline{kw}, A$) ; // top-k dense keywords
15 $\delta \leftarrow \max_{j\in[1,\ldots,k]} \overrightarrow{kw}$.*freq*($j$) $\geq \epsilon$ ;
16 **return** $\overrightarrow{kw}$, $\delta$

### 5.4. Partial approximate materialization

As a result of the materialization performed by Algorithm 1, when querying a non-materialized cuboid, we can directly exploit values in the cuboid's materialized ancestors when computing all distributive and algebraic measures. On the other hand, for holistic measures, we have to perform some additional computation. For instance, as mentioned earlier, to compute the value for the *Top-k Dense Keywords in an area*, we can exploit the pre-computed *Keyword Density* values. However, then we need to perform the top-$k$ selection. That is, if the top-$k$ for the current view is not materialized, we cannot exploit the materialized top-$k$ of the ancestor views without incurring the risk of returning the wrong result.

Yet, it is possible to exploit the top-$k$ computation in some materialized cuboid to retrieve an *approximate* top-$k$ and estimate the result's accuracy [42]. In practice, for the *Top-k Dense Keywords within an area*, given a target $k$ for the top-$k$ computation, when materializing a cuboid, we materialize the top-$k$+1 most dense keywords for that cuboid (i.e., set $K = k{+}1$ in Algorithm 1). Then, to compute the top-$k$ dense keywords for a descendant cuboid by exploiting a materialized ancestor cuboid, we determine which members of the list are guaranteed to be correct.

Algorithm 2 implements this computation for *Top-k Dense Keywords within an area*. It receives as input the set $\Phi = \{(\xi_1, \overrightarrow{kw_1}), (\xi_2, \overrightarrow{kw_2}), \ldots, (\xi_n, \overrightarrow{kw_n})\}$ of lists of top-K (i.e., $k$+1) dense keywords, and the value for $k$. The output is the ranked list of top-$k$ dense keywords in the area $\xi$ that is composed by the merging of the areas $\xi_1, \xi_2, \ldots, \xi_n$. It computes the *SurfaceArea* of the merged area $\xi$ (lines 2–3). Then it merges all the aggregated keyword frequencies in a single dictionary $\overline{kw}$ (lines 4–12) by getting each keyword in each list (line 7) and the corresponding frequency (line 8). Moreover, it keeps track of the upper-bound $\epsilon$ frequency for keywords outside the current materialized ranking for possible error reporting (line 13). The upper-bound $\epsilon$ helps to report which keywords are guaranteed to be correct if the result was computed using *all* the keywords instead of only top-$k$+1 keywords, i.e., any keyword with frequency $\leq \epsilon$ is guaranteed to be correct. In contrast, any keyword with frequency $> \epsilon$ may be incorrectly included in the top-$k$ list due to the unavailability of *all* the keywords (as shown in Fig. 5). Once all frequencies are merged, we can compute the top-$k$ dense keywords using the aggregated frequencies and the current surface area (line 14). Finally, by comparing the value of $\epsilon$ with the frequencies of keywords in the aggregated top-$k$, we report how many positions in the current ranking are guaranteed to be exact (line 15). In the best case, the frequency of the keyword at position $k$ will be at least $\epsilon$, and thus the computed top-$k$ is guaranteed to be correct. On the other hand, if a keyword at position $k$ has a frequency below $\epsilon$, we cannot guarantee the correctness of its position or that any other keyword not reported has the same or higher frequency.

Algorithm 3 implements this computation for *Top-k Volatile Keywords within an area*. It receives as input the set $\Phi = \{(\xi_1, \overrightarrow{kw_1}, T_1), (\xi_2, \overrightarrow{kw_2}, T_2), \ldots, (\xi_n, \overrightarrow{kw_n}, T_n)\}$ of lists of top-K (i.e., $k$+1) dense keywords in a specific area with respective time stamps, time interval $T_x$ divided into $x$ equal-sized interval (e.g., day or month), and the value for $k$. The output is the ranked list of top-$k$ volatile keywords in the area $\xi$ that is composed by the merging of the areas $\xi_1, \xi_2, \ldots, \xi_n$. It computes the *SurfaceArea* of the merged area $\xi$ (lines 2). Then it merges all the aggregated keyword frequencies (line 10) and changes in keyword frequencies (line 11) for each time instance in $T_x$(line 7) in respective dictionaries $\overline{kw}$ and $\Delta f$ (lines 4–13) by getting each keyword in each list (line 8) and the corresponding frequencies (line 9). If a keyword is not found in the $\overline{kw}$, $\Delta f$, or $prev_f$ dictionaries then its value is considered to be *zero*.

Moreover, it keeps track of the upper-bound $\epsilon$ frequency for keywords outside the current materialized ranking for possible error reporting (line 13). The upper-bound $\epsilon$ helps to decide which keywords are guaranteed to be correct if the result was computed using *all* the keywords instead of only top-$k$+1 keywords, i.e., any keyword with frequency $\leq \epsilon$ is guaranteed to be correct whereas any keyword with frequency $> \epsilon$ may be incorrectly included in the top-$k$ list due to the unavailability of *all* the keywords (as shown in Fig. 5). Once all frequencies and changes in frequencies are merged, we compute the top-$k$ volatile keywords using the aggregated values (line 14). Finally, by comparing the value of $\epsilon$ with the frequencies of keywords in the aggregated top-$k$, we report how many positions in the current ranking are guaranteed to be exact (line 15). In the best case, the frequency of the keyword at position $k$ will be at least $\epsilon$, and thus the computed top-$k$ is guaranteed to be correct.

**Algorithm 3:** Top-K Volatile Keywords within an Area

1 **TopKVolatile** ($\Phi = \{(\xi_1, \overrightarrow{kw_1}, T_1), \ldots, (\xi_n, \overrightarrow{kw_n}, T_n)\}$,$T_x$,$k$)
  **Input:** Set of Top-k+1 Volatile Keywords lists $\Phi$, Set of $x$ Timestamps $T_x$, Integer k
  **Output:** $(\xi, \overrightarrow{kw}, T_n)$ top-k keywords $\overrightarrow{kw}$ in the merged area $\xi$ over time interval $T_x$, $\delta$ number of guaranteed top positions
2 $\xi \leftarrow \bigcup_{i\in[1,n]} \xi_i$, $A \leftarrow$*SurfaceArea*($\xi$) ; // Merge areas
3 $\overline{kw} \leftarrow \{\}$, $\Delta f \leftarrow \{\}$, $prev_f \leftarrow \{\}$ ; // Empty dictionaries
4 **foreach** $t \in T_x$ **do**
5  **foreach** $(\xi_i, \overrightarrow{kw_i}, T_i) \in \Phi$ **do**
6  **foreach** $j \in [1, \ldots, k{+}1]$ **do**
7  **if** $t \in T_i$ **then**
8  $w \leftarrow \overrightarrow{kw_i}$.*get*($j$); // keyword at $j$
9  $f \leftarrow \overrightarrow{kw_i}$.*freq*($j$); // frequency at $j$
10  $\overline{kw}[w] \leftarrow \overline{kw}[w] + f$;
11  $\Delta f[w] \leftarrow \Delta f[w] + |prev_f[w] - f|$;
12  $prev_f[w] \leftarrow f$;
13  $\epsilon$ += $\overrightarrow{kw_i}$.*freq*($k+1$);
14 $\overrightarrow{kw} \leftarrow$*topK*($\overline{kw}, A, \Delta f$) ; // top-k volatile keywords
15 $\delta \leftarrow \max_{j\in[1,\ldots,k]} \overrightarrow{kw}$.*freq*($j$) $\geq \epsilon$ ;
16 **return** $(\xi, \overrightarrow{kw}, T_n)$, $\delta$

**Algorithm 4:** STTCubeConstruction

```
1  ConstructSTTCube (𝒳, 𝒯, 𝒢, B, K)
       Input: Collection of Spatio-Textual-Temporal Objects 𝒳, Textual
              Taxonomy 𝒯, Geographical Taxonomy 𝒢, Materialization Budget
              B, desired top-k K
       Output: Spatio-Textual-Temporal Cube 🗇
2      🗇 ← load empty cube;
3      🗇.d_Time ← initialize temporal dimension;
4      🗇.d_Location ← initialize spatial dimension;
5      🗇.d_Text ← initialize textual dimension;
6      🗇.F ← initialize empty Fact Table;
7      foreach x ∈ 𝒳 do
8          UpdateTemporalHierarchies(x.τ, 🗇.d_Time);
9          λ′ ←ProcessLocation(x.λ);
10         UpdateSpatialHierarchies(λ′, 𝒢, 🗇.d_Location);
11         φ′ ←ProcessText(x.φ);
12         UpdateTextualHierarchies(φ′, 𝒯, 🗇.d_Text);
13         add fact(x.τ, λ′, φ′) to the fact batch insert pool;
14     GreedyMaterialization(B, 🗇, K);
15     return 🗇;
```

## 6. STTCube construction and maintenance

Here, we describe the proposed approaches for constructing and maintaining an *STTCube* from a dataset of *STT objects*. First, we explain the construction of spatial, textual, and temporal hierarchies, the population of the fact table, and computation of STT measures from the dataset of *STT objects*. Next, we discuss the proposed method for *incrementally maintaining* the already constructed STTCube by analyzing the associated costs and their impact on the selection of cuboids for materialization. Lastly, we discuss the proposed algorithm for STTCube incremental maintenance.

### 6.1. STTCube construction

Algorithm 4 describes the pre-processing techniques and construction of the STTCube in detail. Algorithm 4 takes a collection $\mathcal{X}$ of *STT objects* to be analyzed, a textual taxonomy $\mathcal{T}$ with semantic information about the terms, themes, topics, and concepts, and a geographical taxonomy $\mathcal{G}$ for cities, regions, and countries. Standard *date functions* are used for the temporal dimension processing. The proposed STTCube design is realized using the classic *snowflake* schema (Section 7). Moreover, it also receives as input the parameters $B$ and $K$ as the budget and number of top-$K$ keywords for the partial materialization.

Algorithm 4 constructs the STTCube in an incremental way, it initializes an empty cube (line 2), and then the corresponding spatial $d_{Location}$, textual $d_{Text}$, and temporal $d_{Time}$ dimensions (lines 3–5) as well as the *Fact Table* $F$(line 6). In particular, the $d_{Location}$ has the grid-based hierarchy and the semantic-based hierarchy with the base level at each object's *Location* $\lambda$ (i.e., the geographical point), and then the levels *City*, *Region*, *Country*, and *All* (5 levels in total). $d_{Text}$, instead, has the hierarchy built from the base level *Term* $\varphi$ and then *Theme*, *Topic*, *Concept*, and *All* (5 levels in total). Finally, $d_{Time}$ contains the *Date* and *TimeOfDay* hierarchies mentioned in Section 3.

Once the basic structure is prepared, Algorithm 4 loops through each STT object in $\mathcal{X}$ (lines 8–13). In this loop, it extracts and initializes from each *STT object* the base-level members $\lambda, \varphi$, and $\tau$ for each dimension. Then, once the base level data has been extracted, it proceeds with building the various dimension hierarchies starting from the existing base-level members and exploiting the provided spatial $\mathcal{G}$ and textual $\mathcal{T}$ taxonomies (lines 8–12). Once the dimension hierarchies are built, the *STT object* itself is then inserted in the fact table $F$ of the *STTCube* (line 13) so that each fact is linked to the lowest (base) level members $\lambda, \varphi$, and $\tau$ in the respective dimensions. Algorithm 4 uses batch pooling to perform updates and inserts in batches. In this step (line 13), the fact measure values are also computed (e.g., the keyword count). As the last step (line 14), Algorithm 4 executes the (partial) materialization procedure.

**Spatial Hierarchies Construction.** In our proposed *STTCube* the base level for the spatial hierarchies is the *Location* $\lambda$ present in the raw data, i.e., the longitude and latitude points. Hence, we use Military Grid Reference System (MGRS) for grid-based hierarchy, and when building the semantic-based hierarchy, individual points are linked to the respective cities using the information in the available geographical taxonomy $\mathcal{G}$, or to a special member for points that link to unknown locations. Therefore, this corresponds to the step function from $\lambda$ to *City*. The spatial taxonomy $\mathcal{G}$ is also used to generate the spatial hierarchy step functions for the higher levels.

**Textual Hierarchies Construction.** The unstructured nature of the text makes it a challenging task to convert it into a dimension of a cube. In Algorithms 4 and 5, the `ProcessText` (Algorithm 6) function (line 11) implements the following standard text processing [43] steps: (1) splits the text into individual words (line 2), (2) removes *stop words* and *invalid* words (line 6), and (3) converts the remaining words to their base form, e.g., "works" and "working" have the same base form "work", (line 10). The final processed text is used to populate the *Term* base-level $\varphi$ in the textual dimension. This implements the base step function and links every fact to one or more *Terms*, hence it has an $n-n$ cardinality. Moreover, while constructing the higher levels, using the semantic taxonomy $\mathcal{T}$ (e.g., WordNet), each *STT object* is linked to one or more *Themes*, and similarly for *Topics*, and *Concepts*.

**STT Base Cube Update.** If the cube is already constructed, i.e., the cube is being maintained instead of constructed for the first time, then Algorithm 5 maintains the base STTCube by incrementally incorporating the new data. Algorithm 5 takes an already constructed STTCube 🗇, a collection $\mathcal{X}$ of *new STT objects* to

**Algorithm 5:** UpdateSTTBaseCube

```
1  UpdateSTTBaseCube (🗇, 𝒳, 𝒯, 𝒢)
       Input: STTCube 🗇, Collection of Spatio-Textual-Temporal Objects 𝒳,
              Textual Taxonomy 𝒯, Geographical Taxonomy 𝒢
       Output: Updated Spatio-Textual-Temporal Base Cube 🗇′
2      foreach x ∈ 𝒳 do
3          UpdateTemporalHierarchies(x.τ, 🗇.d_Time);
4          λ′ ←ProcessLocation(x.λ);
5          UpdateSpatialHierarchies(λ′, 𝒢, 🗇.d_Location);
6          φ′ ←ProcessText(x.φ);
7          UpdateTextualHierarchies(φ′, 𝒯, 🗇.d_Text);
8          add fact(x.τ, λ′, φ′) to the fact batch insert pool;
9      return 🗇′;
```

**Algorithm 6:** ProcessText

```
1  ProcessText (φ)
       Input: Text φ
       Output: Processed Text φ′
2      words ← φ.split();
3      φ′ ← {} ;                                  // Empty bag-of-words
4      foreach word ∈ words do
5          if word ∈ StopWords or word is Invalid then
6              word.excluded ← true;
7              continue;
8          else
9              word.excluded ← false;
10             word.lemma ← Lemmatize(word);
11         φ′.add(word);
12     return φ′;
```

be analyzed, a textual taxonomy $\mathcal{T}$ with semantic information about the terms, themes, topics, and concepts, and a geographical taxonomy $\mathcal{G}$ for cities, regions, and countries.

Differently from Algorithm 4, Algorithm 5 receives an already constructed STTCube as input. Hence, it does not have to load and initialize an empty cube with respective dimensions and fact table. Algorithm 5 (similar to Algorithm 4) loops through each STT object in $\mathcal{X}$ (lines 2–8), incrementally maintain various dimension hierarchies, adds the *STT object* to the fact batch insert pool, and computes the measure values for the (new) facts. As the last step (line 9) Algorithm 5 returns the updated STT base Cube.

### 6.2. STTCube maintenance

After the STTCube is initially constructed, it is required that we can efficiently perform incremental maintenance, i.e., incorporate new data, primarily new fact data (e.g., new tweets), as it becomes available. To address this, we need to consider how the cost of cube maintenance, i.e., the cost to update the contents of each materialized view, affects the selection of the views to be materialized and how to subsequently perform the maintenance most efficiently. In accordance with the literature [44], our main observation is that the volume of the new data to be incorporated in the already constructed STTCube is much smaller than the data already stored in it. For example, in the case of the tweets dataset (Section 7), we collect on average around 3 million tweets per day, so that after six weeks, adding a new day's data corresponds to approximately only 1/50 of the already collected data. Moreover, the size difference between the newly available and already incorporated data in the STTCube keeps growing over time.

The problem of view selection in the presence of updates is a well-known issue [44]. In particular, in previous work, the set of available views in a Data Warehouse is modeled through a so-called OR-graph, which corresponds to our STT lattice in Fig. 8. As analyzed in such works [44], for the case of Data Cubes, the frequency of updates is (much) smaller than the frequency of queries. Therefore, since the general volume of updates across all views is negligible when compared to the volume of data already in each view, we are able to apply the *same* view selection algorithm, with the *same* performance guarantees, *without* the need to *consider* the *update cost* [44]. Furthermore, our proposed STTCube incremental maintenance method only updates the already materialized views as new data becomes available, i.e., the set of materialized views will not change. In the following, we present a view maintenance procedure that allows us to do so.

Given a materialized view $mv$ and a base fact delta $\Delta_f$, the set of data that needs to be updated or inserted (there are no deletes, explained later) to the data cube, the following equation states the cost of incremental maintenance ($IM$) and its associated components.

$$IM(mv, \Delta_f) = Read(mv, \Delta_f) + Upsert(mv, \Delta_t)$$

$Read(mv, \Delta_f)$ represents the cost of reading the base fact delta $\Delta_f$ to generate a target delta $\Delta_t$ ($\Delta_f \rightarrow \Delta_t$) by aggregating rows in $\Delta_f$ in such a way that each row of $\Delta_t$ represents either an update or insert over the view $mv$. In some cases, it is required to read some of $mv$ as well in order to produce a *merged row* combining an existing row in $mv$ and a new row in $\Delta_f$. However, the number of rows needed to be read from $mv$ is proportional to the size of $\Delta_f$. Hence, the cost of $Read(mv, \Delta_f)$ is represented by the following equation.

$$Read(mv, \Delta_f) = size(\Delta_f)$$

$Upsert(mv, \Delta_t)$ represents the cost of upserting rows, i.e., inserting new ones and updating existing ones, in the view $mv$. Each *row* in $\Delta_t$ is either an insert or update on $mv$. *NoOfUpdates* and *NoOfInserts* are update and insert counters, respectively. Each update or insert operation increments the respective counter by one and is represented by the following equation.

$$NoOfUpdates = |\Delta_t \cap mv|$$

$$NoOfInserts = |\Delta_t \setminus mv|$$

Moreover, the cost of an insert is (much) smaller than of an update. They are represented by *CostInsert* and *CostUpdate*, respectively. Thus, the cost of upsert is represented by the following equation.

$$Upsert(mv, \Delta_t) = CostInsert \times NoOfInserts + CostUpdate \times NoOfUpdates$$

Therefore, the cost of upsert is proportional to the size of $\Delta_t$.

$$Upsert(mv, \Delta_t) \sim size(\Delta_t)$$

Lastly, there are no delete operations in the process of STTCube incremental maintenance because every new STT object to be incorporated is a new object, e.g., in the case of social media platform like Twitter, tweets are not modified or deleted once created.

**Example.** Fig. 10 shows the proposed STTCube incremental maintenance ($IM_{stt}$) mechanism. It shows the flow and aggregation of deltas, i.e., the left column shows how $\Delta_o$ (delta containing the new STT objects) is aggregated for a specific materialized view ($mv$) (or base STTCube), and each row represents how a $mv$ (or base STTCube) is updated using the respective $\Delta_t$ (target delta computed for the specific $mv$ (or base STTCube)). The incremental maintenance starts with the availability of new STT objects $\Delta_o$ (top-left gray box). At first, $IM_{stt}$ groups similar STT objects, e.g., objects having the same terms, location, and timestamps, by computing $\Delta_f$ (base fact delta), and uses $\Delta_f$ to update the base STTCube, i.e., updating the dimensions and fact table. Next, for each $mv$, $IM_{stt}$ uses the *best available* (smallest that can be aggregated to the granularity of $mv$) delta, we call it $\Delta_s$ (source delta), to produce a new $\Delta_t$ (target delta) by aggregating available data in $\Delta_s$ at the same level of aggregation as the considered $mv$. Each row of $\Delta_t$ corresponds to an update or insert on the respective $mv$. Once the $\Delta_t$ is ready, $IM_{stt}$ uses it to maintain the respective $mv$. Also, $IM_{stt}$ processes the materialized views in a specific order, i.e., the largest ones first. In Fig. 10, $IM_{stt}$ uses $\Delta_f$ as source delta to compute $\Delta_{mv1}$ (a target delta) for view $mv1$ and maintains $mv1$ using $\Delta_{mv1}$. Similarly, for $mv2$, $IM_{stt}$ uses $\Delta_{mv1}$ as source delta to compute $\Delta_{mv2}$ for $mv2$ and uses $\Delta_{mv2}$ to maintain $mv2$. For $mv1$, the read, update, and insert numbers are 4 (number of rows in the respective $\Delta_s$), 2 rows updated, and 2 rows inserted, respectively. Similarly, for $mv2$, the read, update and insert numbers are 2, 2, and 0, respectively.

Algorithm 7 implements the proposed STTCube incremental maintenance mechanism. It receives as input the STTCube to be updated, a collection $\Delta_o$ of *new STT objects* to be incorporated in the STTCube, a textual taxonomy $\mathcal{T}$ with semantic information about the terms, themes, topics, and concepts, a geographical taxonomy $\mathcal{G}$ for cities, regions, and countries, and STTCube lattice $L$. The output is the updated STTCube $'$. It starts by invoking the `UpdateSTTBaseCube` function (described in Algorithm 5) to update the base cube, i.e., update respective hierarchies and the fact table with new data (line 2). It computes the base fact delta $\Delta_f$ containing all hierarchies' lowest-level members (line 3). At this step, it aggregates multiple facts in $\Delta_o$ representing the same aggregated data. Next, it constructs an empty list $\Delta S$ to store all the computed deltas (some of which are computed later in the process) and adds the currently computed $\Delta_f$ to $\Delta S$ (lines 4–5). It sorts all the available materialized views in for optimal

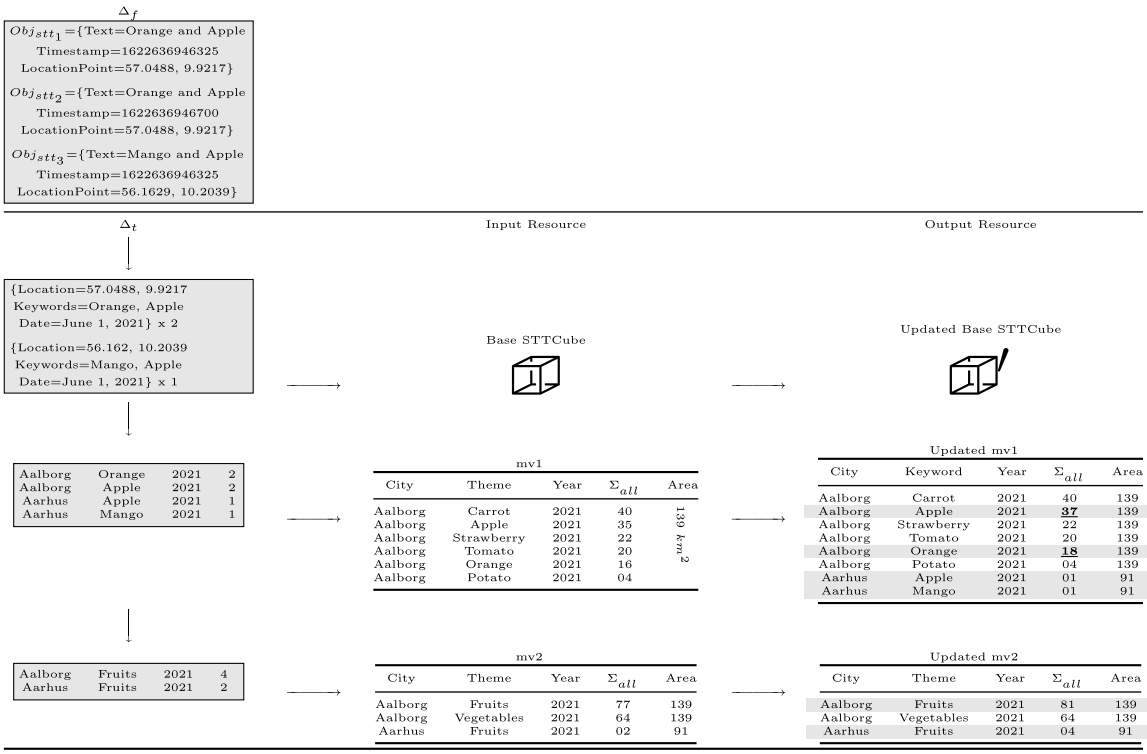



**Fig. 10.** Example: STTCube's maintenance cost.

**Algorithm 7:** STTCube Maintenance

```
1  STTCubeMaintenance (𝔇, Δo, 𝒯, 𝒢, L)
       Input: STTCube 𝔇, Collection of new STT Objects Δo, Textual
              Taxonomy 𝒯, Geographical Taxonomy 𝒢, STTCube Lattice L
       Output: Incrementally Maintained STTCube 𝔇′
2      UpdateSTTBaseCube(𝔇, Δo, 𝒯, 𝒢);   // Update base STTCube 𝔇
         using Algorithm 5
3      Δf ← compute base fact delta;
4      ΔS ← [];                          // Empty list of computed deltas
5      ΔS.add(Δmv);
6      MV ←Sort(𝔇.MaterializedViews, L) ;   // Order views, larger
         views first, for optimal update using lattice
7      foreach mv ∈ MV do
8          Δs ← find the smallest source delta for mv using ΔS and L;
9          Δmv ← compute target delta for mv using Δs;
10         foreach row ∈ Δmv do
11             if row ∉ mv then
12                 add row to the batch insert pool;
13             else
14                 add update for row to the batch update pool;
15         ΔS.add(Δmv);      // Add currently computed Δmv to the
             list of available deltas ΔS
16     return 𝔇′
```

maintenance, which means that it maintains views in a top-down fashion, i.e., maintain larger views first (see Fig. 8) and computes respective deltas early on, which will serve as better candidates for maintaining the later views (line 6). For example, in Fig. 10 it maintains $mv1$ before $mv2$ and uses $\Delta_{mv1}$ to maintain $mv2$. Then it loops through all the sorted views (lines 7–14) and finds the smallest source delta $\Delta_s$ that can be used to maintain the current view $mv$ from $\Delta S$ (line 8). Then it computes target delta $\Delta_{mv}$ by aggregating rows in $\Delta_s$ such that each row in the computed target delta is either an update or insert to the current materialized view $mv$ (line 9). Once the $\Delta_{mv}$ is computed, Algorithm 5 iterates through all the rows in $\Delta_{mv}$ and performs either an update or insert on $mv$ (lines 10–14). Here, it uses batch pooling to perform updates and inserts in batches since most DBMS have utilities to apply such batches much faster than using individual SQL statements. Finally, it adds the newly computed $\Delta_{mv}$ to the list of available deltas $\Delta S$ (line 15). Once the base fact table, dimensions tables, and all the materialized views in the 𝔇 have been updated, Algorithm 7 returns the incrementally maintained STTCube 𝔇′ (line 16).

## 7. Experimental evaluation

Now, we report on the performance of *STTCube* analysis. In particular, we compare the different materialization strategies for *STTCube* and *No STTCube (NC)* implementations in terms of query response time (QRT) and storage cost. *NC* answers the queries by computing the query response from base data without constructing the *STTCube*. Also, we compare QRT and hierarchy construction time for different combinations of hierarchy schemes. Moreover, we also report on the accuracy of *PAM* and demonstrate the advantage in performance when compared to *PEM*. Furthermore, we compare our proposed $IM_{stt}$ method with a baseline maintenance method in terms of maintenance time. Lastly, we compare QRTs for different spatial and textual hierarchy schemes, showing that combinations of *Grid-based spatial and Majority-based textual (GM)* hierarchy scheme achieves the fastest QRTs among all hierarchy combinations.

**Experimental Setup.** We evaluate the *STTCube* on a real-world Twitter dataset containing 125 million tweets collected over *six* weeks. We perform experiments on five different sizes of datasets to show the impact of data size on query response time). Each tweet contains the tweet location, text, and time. We implemented the *STTCube* in a leading commercial RDBMS, called *RDBMS-X* as we cannot disclose the name. The proposed design is realized using a *snowflake schema* to avoid redundancy in the dimension data. Using a snowflake schema requires more joins to be executed; hence, it can take much longer to produce results depending on the dimensions, but it provides less redundancy and more flexibility in handling $n - n$ hierarchies. Moreover, for

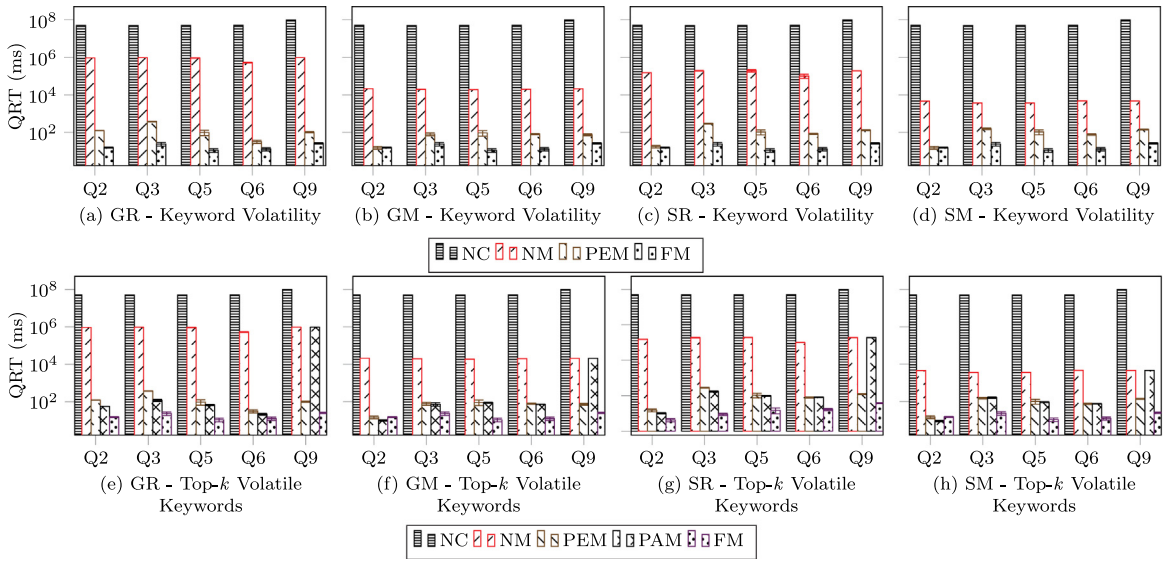



**Fig. 11.** QRTs for STT measures for different combinations of hierarchy schemes over 125 Million Tweets (log scale).

$n-n$, hierarchies are implemented using a bridge table. Furthermore, QRTs also heavily relies on the internal implementation of the join algorithms in the used DBMS, but these are outside the scope of our analysis.

We implemented the *Pre-Processing (PP)* component, where the whole raw dataset is parsed and the relational tables are populated, in Java (v11). Most of the tests are run on a Windows Server machine with 2 Intel Xeon 2.50 GHz CPUs from 2017, fast solid-state drives, and 16 GB RAM. We perform the QRTs scalability (Fig. 12) and STT-Cube incremental maintenance (Fig. 14) experiments on a Ubuntu server machine with 32 AMD 3.0 GHz CPUs, fast solid-state drives, and 251 GB RAM, so we can handle the very large data volumes in those experiments.

We implemented the *semantic-based* and *grid-based* hierarchy schemes for the spatial dimension, replication-based and majority-based hierarchy schemes for the textual dimension (Section 3.2), and Date hierarchy for the temporal dimension.

In particular, we extracted the taxonomy for the spatial dimension from GeoNames [45]. For the *City* level, we considered all the cities having *population* $>$ 1000 and for the *Region* level, we use administrative divisions information available in the GeoNames dataset. We use the reverse geocoding process to find the city name for the *Location* coordinates. Moreover, the grid-based hierarchy has been implemented with the Military Grid Reference System (MGRS) using squares with a side of size 1 meter, 10 m, 100 m, 1000 m, and 10000 m.

For the textual dimension, as a taxonomy for *Terms*, *Themes*, *Topics*, and *Concepts*, we use the widely used WordNet [37]. We use WordNet as an *imperfect* text collection of relevant terms; hence, all terms that are missing in there, such as hashtags or terms formed by words glued together without spaces, are not present in its hierarchy. We use the direct *HYPERNYM* link of WordNet to decide the parent member for a *Term*, *Theme*, and *Topic*. If a term is present in WordNet and has a super-class (*HYPERNYM*), then the super-class becomes the parent of the term. Otherwise, it becomes its own parent (this avoids unbalanced hierarchies and UNKNOWN values in the hierarchy). For text preprocessing – tokenization, lemmatization, and stop word removal – we use the Stanford Core NLP library [46]. We implemented the temporal dimension using the standard *Date* and *Time* functions supported in *RDBMS-X*.

**Spatial, Textual, and Temporal Levels Members.** In the constructed STTCube, the base levels contain 40.1 million unique *Location Points* and 9.8 million unique *Terms* (both valid and invalid). The GeoNames taxonomy contains 132*K* cities, divided into 4*K* administrative divisions (regions) for 247 countries. Among those, we have tweets for 104 K cities, 3.8 K regions, and 246 distinct countries. In the textual hierarchy, terms are grouped into 23.8 K *Themes*, 19.4 K *Topics*, and 17.6 K *Concepts*. Furthermore, the temporal dimension spans over 37 days. Finally, for *PAM* we materialize $K$=31 densest keywords (selection of K is discussed later in this section, Fig. 15(c)).

We compare *PEM* and *PAM* strategies with the following three baselines. **No STTCube (NC):** is the traditional RDBMS setup with all textual, spatial, and temporal functions implemented as built-in or user-defined functions. Specifically, *NC* uses user-defined functions for text (for retrieving individual terms) and location processing (e.g., identification of the city a particular longitude, latitude point belongs to) and built-in functions for timestamp. Further, *NC* filters on location and timestamp for the queried *area* and time and performs a series of joins, e.g., 4 joins for *Concept* level, to retrieve information for the requested textual level. Finally, it groups results on the textual and temporal columns, computes the *STT measure values*, and performs the top-*k* selection. *NC* is the *traditional* solution one would go for without the *STTCube*. **No Materialization (NM):** constructs the *STTCube* and minimizes the storage cost by only materializing the base cuboid and computing all query responses from it. **Full Materialization (FM):** minimizes the QRT by materializing every cuboid in the *STTCube*. With this approach, queries are answered through a lookup in the pre-computed cuboid. *These three baselines are at the extreme ends of the space–time trade-off and are usually infeasible for large datasets.*

**Queries.** We perform experiments using nine different *STT queries*. Each *STT query*, described in Table 3, targets different levels of spatial, textual, and temporal granularities. Each query requests either dense or volatile keywords. The *[time span]* constraint is only required for dense keywords queries; hence, queries without time constraint are dense keywords queries. We execute each query ten times with randomly generated parameters for each method and report mean and standard deviation.

**Query Response Time.** For *Top-k Dense and Top-k Volatile Keywords within an area* measures, we compare the QRT of *PEM* and *PAM* with the *NC*, *NM*, and *FM* baselines. For *Keyword Density* and *Keyword Volatility*, no approximate solution is possible so we

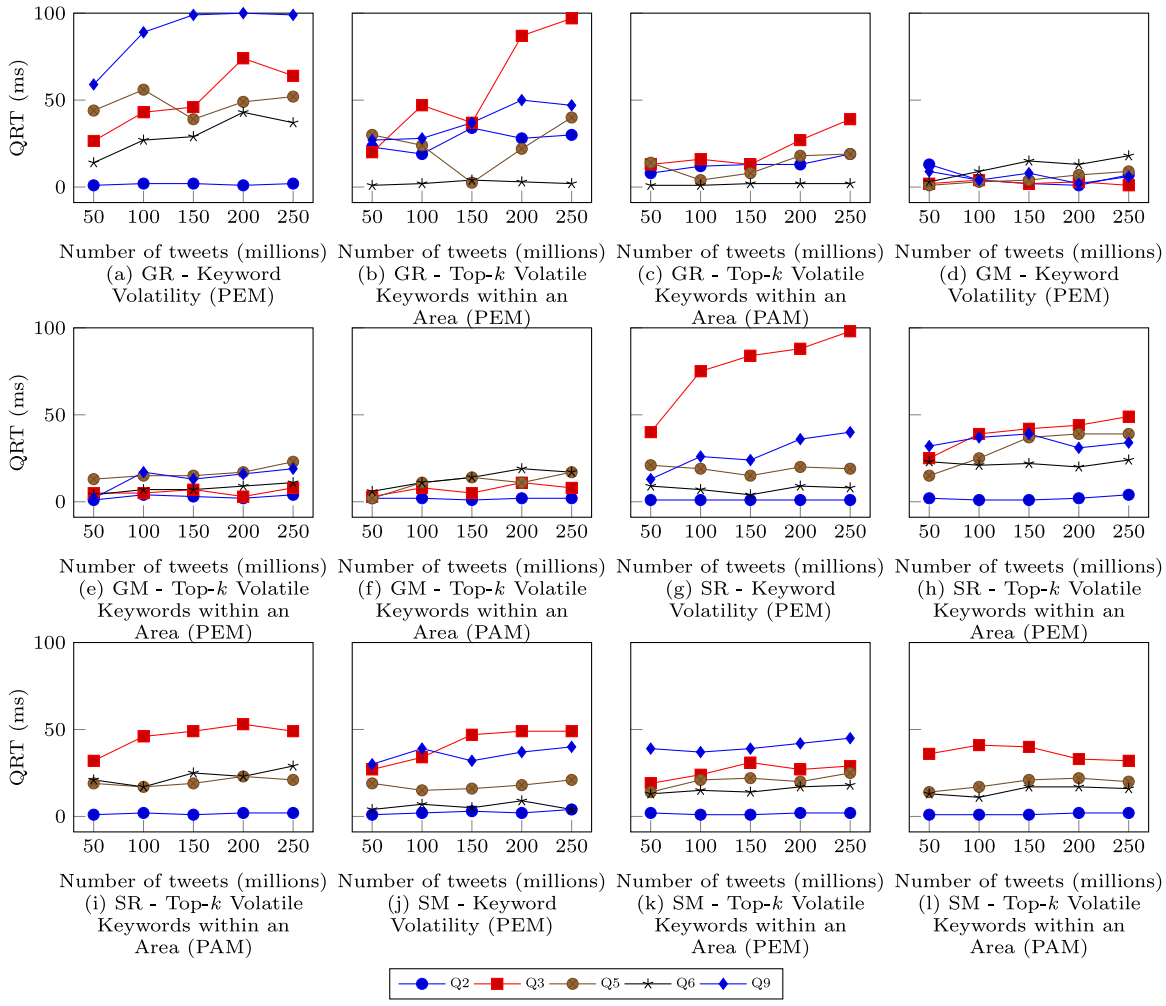



**Fig. 12.** QRT Vs data size.

only compare *PEM* with *NC*, *NM*, and *FM*. As the *Majority-based* textual hierarchy scheme links facts to *Theme* instead of *Terms* (Section 3.2), we only evaluate *five* out of *nine* queries requesting *Theme*, *Topic*, and *Concept* for it (Figs. 11b, 11d, 11f, and 11h). Furthermore, we cannot evaluate *PAM* for Q9 as no approximate solution is possible for it.

We plot results in Figs. 11a–11h for 100% (125M) of data and *five* out of *nine* queries, as the results are similar for smaller data sizes and omitted queries.[1] Specifically, Figs. 11a and 11e show the QRTs for the Grid-based spatial and Replication-based textual (**GR**) hierarchy combination for all *measures*. Similarly, Figs. 11b and 11f, Figs. 11c and 11g, and Figs. 11d and 11h show QRTs for Grid-based spatial and Majority-based textual (**GM**), Semantic-based spatial and Replication-based textual (**SR**), and Semantic-based spatial and Majority-based textual (**SM**) combinations, respectively.

Fig. 11 has queries on the *x*-axis and QRTs in msec on the *y*-axis (note: log scale). Fig. 11 confirms that

- *NC* is 1−5 orders of magnitude slower than *NM*. Specifically, regardless of the spatial hierarchy scheme, it is 1−2 and 3−5 orders of magnitude slower than *NM* for *Replication-based* and *Majority-based* textual hierarchy, respectively. The *Majority-based* textual hierarchy scheme achieves faster QRTs because it does not process individual *Terms* but directly links *Theme* to the *fact*, hence, drastically reducing the number of rows to process (from millions to thousands).
- *NM* is 1−4 and 3−5 orders of magnitude slower than *PEM* and *PAM*, respectively, for all measures (both algebraic and holistic) and combinations of hierarchy schemes.
- *PEM* is on average six times slower than *FM* which achieves its fast QRTs at the expense of a highly increased storage cost (Fig. 13(a)).
- *PAM* achieves *near-optimal* QRTs because it materializes only *the K densest keywords* in the cuboid, hence it has much fewer rows to process.
- QRTs for *Q9* for *Top*-k *Volatile Keyword within an area* and *Top*-k *Dense Keywords within an area* holistic measures for all combinations of hierarchy schemes are the worst for *PAM* (same as *NM*) because it requests *ALL* keywords' densities instead of *top-k* which cannot be computed from the approximate pre-aggregated information. To generate a response for *Q9*, we have to process all detailed data directly from the base facts. In comparison, *PEM* and *PAM* materialize a subset of views (also a subset of rows for *PAM*) and use the pre-aggregated measure values in those views to efficiently generate a response for a query instead of processing base facts, thus improving the overall QRT.
- *NC* is the slowest of all (1 − 5 orders of magnitude slower than the slowest *STTCube NM*) because it has to process the complete dataset for computing each query response

[1] We have omitted the figures for *Keywords Density* and *Top*-k *Volatile Keyword within an area* measures (can be found in Appendix Fig. A.1) as we observed similar results.

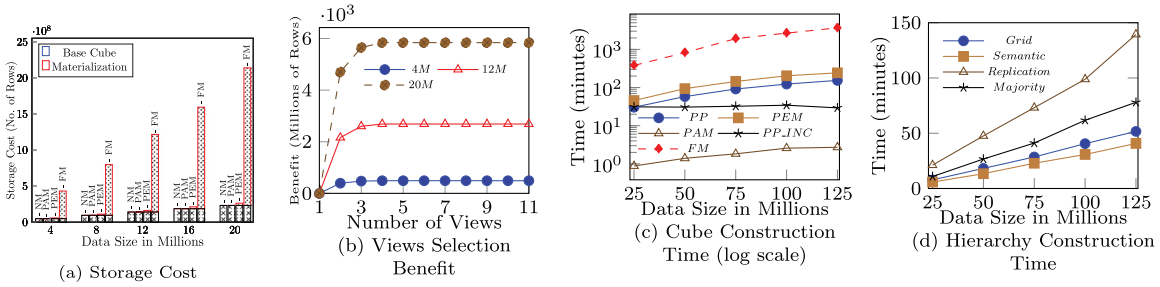



**Fig. 13.** Benefit and storage cost for views materialization and construction time for cube and hierarchies.

and cannot take advantage of the *STTCube* optimizations for algebraic and holistic *STT measures*.

- Among all the hierarchy scheme combinations (explained in Section 3.2), *GM* has the fastest QRTs mainly because of *Majority-based* which drastically reduces the row count by linking the *Theme* directly to each *Fact* instead of individual *Terms*, whereas, *GR* has the slowest QRTs due to *Replication-based* having far more rows to process than *Majority-based* textual hierarchy.
- Furthermore, *Grid* and *Semantic-based* spatial hierarchies have similar QRTs, i.e., GM and GR have the same performance as SM and SR, respectively.

Fig. 12 shows the scalability of *PEM* and *PAM* over growing data sizes for different combinations of hierarchy schemes and confirms that the QRTs are almost constant as the data grows.[2] This is because the sizes of the materialized views do not increase a lot as the data grows. Only new dimension members, e.g., new cities or topics, increase the size of materialized views, but only by a small fraction. Figs. 12d–12f confirm that the *GM* hierarchy combination results in the fastest QRTs, i.e., all QRTs $< 50$ msec. On the contrary, Figs. 12a–12c show that *GR* yields the slowest QRTs, with QRT as high as 100 msec. Fig. 12 confirms that *PAM* consistently achieves the fastest QRTs, i.e., all QRTs $< 50$ msec, regardless of the hierarchy schemes. Fig. 12 shows that *PEM* and *PAM* scale linearly w.r.t. data size.

**Storage Cost.** We now compare the storage cost for *FM*, *PEM*, *PAM*, and *NM*. We do not compare *NC*'s storage cost because it does not construct STTCube and hence does not materialize anything. We only show the storage cost for up to 20 million because *FM* takes an unfeasible amount of time (shown in Fig. 13(c)), while for the other methods and over the larger datasets, we observe the same trend. We use the number of rows in a view as its storage cost. The base cube's storage cost is always needed. Besides that, every additional materialized view adds to the storage cost, as displayed in Fig. 13(a), that shows the storage cost of *NM*, *PAM*, *PEM*, and *FM* over growing data sizes. The materialization of the STTCube using *PEM* and *PAM* only adds 13% and 0.1% to the storage cost of the base cube, respectively. Whereas using *FM* increases the storage cost by more than an order of magnitude. *PEM* reduces the storage cost by only materializing a subset of views (four views) and still achieves 2-5 orders of magnitude improvement in QRT (Figs. 11). *PAM* further reduces the storage cost by only materializing a subset of rows in a view (top-$k$) and gains an additional order of magnitude improvement in QRT. On the other hand, *FM* materializes all views in a cube, i.e., 500 ($5 \times 5 \times 5 \times 4$) views in our case, which makes the view materialization storage cost much higher (one order of magnitude) than the base cube itself, as shown in Fig. 13(a). Fig. 13(a) confirms that our proposed methods *PEM* and *PAM* reduce the storage cost between 97% and 99.9% compared to *FM*.

**Views Selection for Materialization.** Our proposed methods *PEM* and *PAM* are partial materialization methods that materialize only a subset of the cuboids. Hence, an important trade-off to be understood is between the number of cuboids to materialize, the corresponding storage cost, and the gain in query response time achieved. We empirically evaluate the benefit gained (improvements in QRT for all dependent cells, which can be answered using this view) against the cost of materializing the view (Algorithm 1). We consider the base cube as a necessary view to be materialized and consider its benefit as zero. Fig. 13(b) shows that materializing three cuboids (*(Day, City, Term)*, *(Day, Location, Theme)*, and *(Day, Region, Term)*) on top of the base cube gain the most benefit after which we do not get a significant advantage of materializing further cuboids. The reason is that the materialized cuboids are already small enough, so the benefit of materializing any descendant cuboid is small. Hence, materializing 4 cuboids represents the best trade-off between QRT and storage cost.

**Pre-Processing and Cube Construction.** Here, we report the time for the construction of *STTCube*. Construction of an STTCube is divided into two steps: 1) *Pre-Processing (PP)* of base facts (*STT objects*) and population of the relational tables and 2) materialization of views. Further, the materialization of views can be done either using *FM*, *PEM*, or *PAM*. In Fig. 13(c), we have data sizes on the $x$-axis and time in minutes on the $y$-axis (note: log scale). *FM* is the most time-consuming among all and adds significant overhead on top of *PP* time and does not scale. On the contrary, *PEM* and *PAM* time is negligible compared to the *FM* time. Hence, with *PEM* and *PAM* *STTCube* construction time scales linearly. To evaluate *STTCube*'s ability to handle updates (maintenance wall-clock time), we performed several updates of 25M tweets each (*PP_INC* in Fig. 13(c)). Experiments confirm that STTCube's update time grows linearly with the amount of new *STT objects* because it only processes the new STT objects and updates respective fact and dimensions tables.

Furthermore, we compare the different hierarchy schemes w.r.t. their construction time. Fig. 13(d) shows the hierarchies' construction time for different hierarchy schemes. It is evident from Fig. 13(d) that, among all, the *Replication-based* textual hierarchy scheme takes the longest to construct because, for every single *spatio-textual-temporal* object, it has to process each individual *Term* and construct hierarchy for it. Whereas, for all other schemes, for each spatio-textual-temporal object, only one hierarchy instance is processed. Fig. 13(d) confirms that all of the hierarchy schemes are constructed in linear time w.r.t. data size, allowing STTCube to support multiple hierarchy schemes.

**STTCube Incremental Maintenance.** Next, we performed experiments to demonstrate the advantages of our proposed STTCube incremental maintenance method $IM_{stt}$ over a baseline maintenance solution, $Recompute_{all}$, of recomputing the views from scratch *after* each update. We constructed an STTCube over 30

[2] We have omitted the figures for *Keywords Density* and *Top-k Volatile Keyword within an area* measures as we observed similar results.

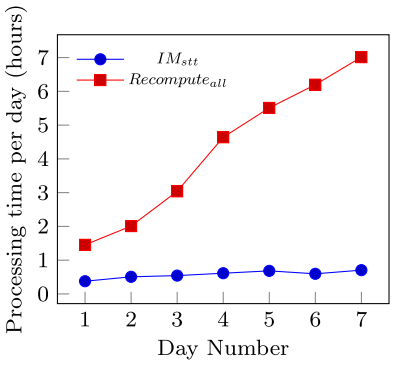



**Fig. 14.** STTCube incremental maintenance.

million tweets and partially materialize it (see Fig. 13(b)). Then, we performed several maintenance operations to incorporate one day's data (30 million for each day, i.e, $\Delta$=30$M$) in the already constructed STTCube and recorded the execution time for the $IM_{stt}$ and $Recompute_{all}$. This corresponds to including data for one more week into the existing STTCube by performing seven maintenance operations of one day each. Fig. 14 shows the time taken by $IM_{stt}$ and $Recompute_{all}$ methods for incorporating new data into the already constructed STTCube. The $x$-axis shows the day number for each maintenance operation, while the $y$-axis shows the processing time for each day (in hours). It is evident that $IM_{stt}$ is on average an order of magnitude faster than $Recompute_{all}$ in this setting. $IM_{stt}$ execution time increases very slowly and linearly with the size of the existing STTCube. On the contrary, $Recompute_{all}$'s execution time is growing on average 9 times faster with the increase in the size of the existing STTCube. Fig. 14 confirms that time taken by $IM_{stt}$ is negligible compared to $Recompute_{all}$ even when it is constructed for only one week of data. In a real setting, an STTCube usually contains data for many weeks and months (even years); in this case, the difference between $IM_{stt}$ and $Recompute_{all}$ execution time will be much, up to several orders of magnitude, higher. It is evident from these experiments that the proposed incremental maintenance method $IM_{stt}$ maintains STTCube efficiently and the size of the existing STTCube has a limited impact. Thus, we have confirmed our hypothesis that the volume/ cost of updates on a view is negligible compared to the volume of data in that view. We can apply the same view selection algorithm, with the same performance guarantees, without the need to consider the maintenance cost.

**Accuracy.** Given that *PAM* efficiently computes the approximate measure values, it becomes necessary to evaluate its accuracy [42]. To evaluate the accuracy of *PAM*, we use *NM*'s results as ground truth. Our evaluation result in Fig. 15(a) confirms that it achieves high accuracy. Specifically, it is 100% for 6 out of 8 queries and 90%–97% for 2. Queries with 90%–97% accuracy request as many keywords as are materialized, and the risk of having wrong results near the border (bottom of the top-$k$ list) is higher.

**QRT of STTOLAP Operators.** Our proposed materialization strategies (*PEM and PAM*) improves the QRTs for *STTOLAP operators*. To demonstrate this, we perform a series of STTOLAP operations and measure their QRT for different materialization strategies. Fig. 15(b) shows the QRTs for multiple *STTOLAP operations* for different materialization strategies. We have STTOLAP operators on the $x$-axis (RU, D, S, and DD represents STT Roll Up, Dice, Slice, and Drill Down operators, respectively) on QRT in msec on the $y$-axis. It is evident that *NM* is on average 3–5 orders of magnitude slower than *PEM* which is one order of magnitude slower than *PAM*. Furthermore, *PAM* achieves near-optimal QRTs, just a fraction higher than *FM*. These experiments confirm that *STTCube's* materialization methods (*PEM and PAM*) improves *STTOLAP operators'* QRTs by materializing only a subset of cuboids.

**Top-K Value Estimation.** Here, we study the relationship between QRT and the value of materialized $K$. We create *seven* different *STTCube* materialization versions using 10, 20, 50, 100, 200, 500, and 1000 as the value of $K$. Next, we use the Gamma distribution to generate 100 random numbers, to be used as top-$k$ values, in the range of 1 and 1000. We chose the Gamma distribution because it resembles a common long-tail distribution for top-$k$ values. We execute each query for all the 100 generated top-$k$ values over all *seven* materialization versions. Fig. 15(c) shows the QRT for all queries over different materialization versions. For $K$=10 and 20, the median value is the same as the box top, hence not visible in the plot. It is evident from Fig. 15(c) that a larger value of materialized $K$ achieves faster QRTs (lower median value) because almost all the queries are answered using the pre-computed measure values. But, in the case of smaller $K$, all the queries requesting $k$>$K$ need to be answered using the non-pre-computed measure values from the base cuboid. Hence, resulting in slower QRTs (higher median value). A larger value of $K$ such as 1000 is not recommended because 1) there will be very few queries requesting a larger top-$k$ and 2) it will require more storage cost (Fig. 13(a)). Specifically, between $K$=50 and 100 and $K$=100 and 200 QRT decrease by 35% and 0% but storage increase 250% and 200%, respectively. Hence, these experiments confirm that choosing a value between 20–50 for $K$ in our current experimental settings is a near-optimal choice. Furthermore, selecting the $K$ value to materialize is use-case dependent. In practice, one can target the most frequent request value for $k$ in a normal business analysis scenario (e.g., the 90th percentile from historic log of queries and workloads) and use a value for $K$ that is slightly above that value (e.g., the 91th percentile).

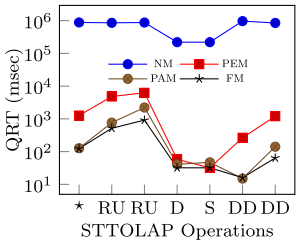
**Summary of Findings.** Our empirical evaluation confirms that among the *spatio-textual-temporal cube* materialization strategies, *NM* uses the least amount of storage but has the worst QRT, while *FM* requires far too much storage to achieve optimal QRT. Among all the methods, *NC* has by far the worst QRT. In comparison, our proposed methods *PEM* and *PAM* improve QRT with 1–5 orders of magnitude compared to *NM*, reduce storage cost between 97% to 99.9% compared to *FM* and add only a minor overhead in the *spatio-textual-temporal cube* construction time. Furthermore, our proposed incremental maintenance method $IM_{stt}$ improves maintenance time by an order of magnitude compared to $Recompute_{all}$ in our experimental setting. Thus, *PEM*, *PAM*, and $IM_{stt}$ are the best-suited techniques for enabling efficient spatio-textual-temporal cube analytics.

## 8. Conclusion and future work

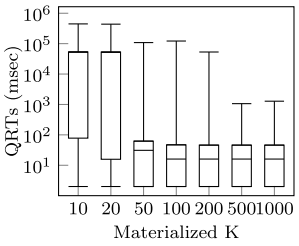
The widespread adoption of mobile devices, in conjunction with social media platforms, is generating an enormous amount of *STT data*. This research work was motivated by the need for enabling efficient *combined* analytical processes over *STT data*. In this paper, we defined and formalized the *STTCube* structure to effectively perform *STTCube analytics*. *We introduced STT hierarchies, STT measures, and STTOLAP operators to analyze STT data together.* Our proposed STTOLAP operators handle n–n relationships inside the *STT* dimensions effectively which allows us to pre-aggregate STT measure values. We also proposed an incremental maintenance method to efficiently maintain the already constructed STTCube by incorporating the new data as it becomes available. For efficient, exact and approximate, computation of *STT measures*, we proposed a pre-aggregation framework able to provide faster response times by requiring a controlled amount of extra

| Query | Data Size in Millions | | | | |
|---|---|---|---|---|---|
| | 25 | 50 | 75 | 100 | 125 |
| Q1 | 100.0 | 100.0 | 100.0 | 100.0 | 100.0 |
| Q2 | 100.0 | 100.0 | 100.0 | 100.0 | 100.0 |
| Q3 | 100.0 | 100.0 | 90.0 | 95.0 | 90.0 |
| Q4 | 92.3 | 100.0 | 100.0 | 100.0 | 100.0 |
| Q5 | 100.0 | 100.0 | 100.0 | 100.0 | 100.0 |
| Q6 | 100.0 | 100.0 | 100.0 | 100.0 | 100.0 |
| Q7 | 93.3 | 96.7 | 90.0 | 93.3 | 93.3 |
| Q8 | 100.0 | 100.0 | 100.0 | 100.0 | 100.0 |

(a) PAM's Accuracy




(b) STTOLAP Operations' QRTs




(c) Materialized-K Vs QRT

**Fig. 15.** PAM's Accuracy and QRT's for STTOLAP Operators and Materialized-K.

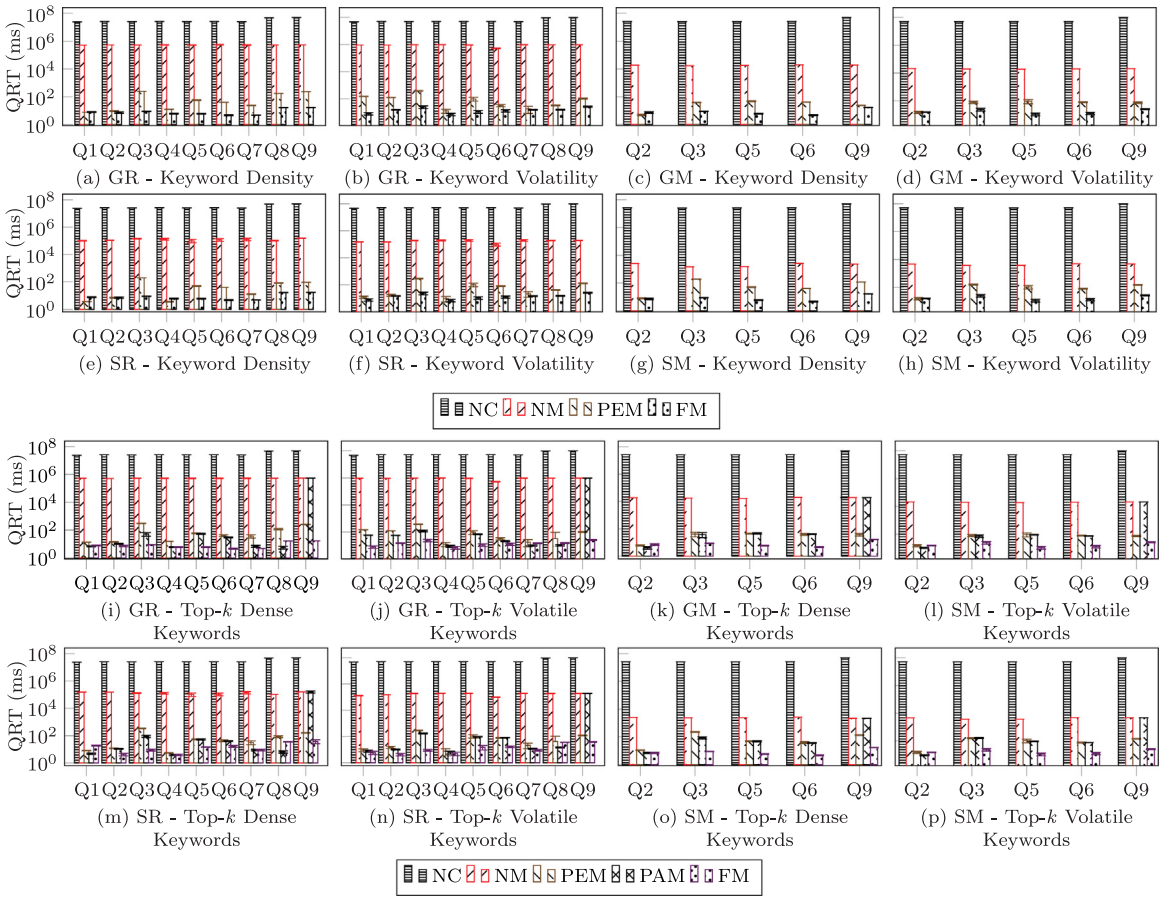



**Fig. A.1.** QRTs for STT measures for different combinations of hierarchy schemes over 125 Million of Data.

storage to store pre-computed measure values. We performed experiments on real-world twitter dataset, compared *PEM* and *PAM*'s QRTs with *NM*, *FM*, and *NC*, and evaluated the space–time trade-off among different materialization methods, both exact and approximate. We observed how partial materialization provides 1 to 5 orders of magnitude reduction in query response time, with between 97% and 99.9% reduced storage cost compared to full materialization techniques. Moreover, the approximate materialization provides accuracy between 90% and 100% while requiring considerably less space compared to no materialization techniques. Furthermore, our proposed STTCube incremental maintenance method maintains STTCube efficiently by reducing the STTCube maintenance time by an order of magnitude. In future work, we plan to enhance STTCube with additional *STT measures* and distributed implementation.

## Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

## Acknowledgements

This research is partially funded by the European Commission through the Erasmus Mundus Joint Doctorate Information

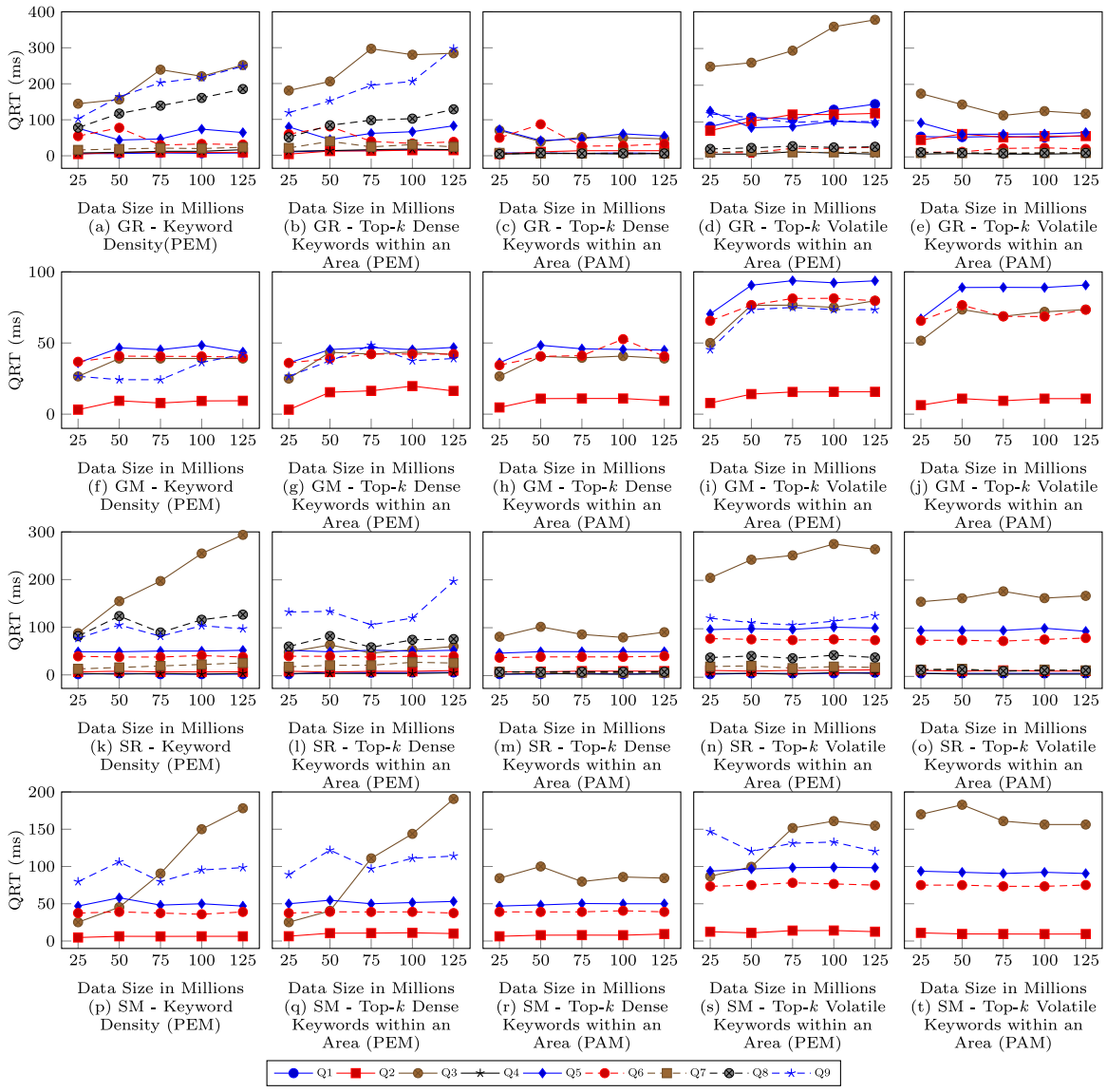



**Fig. A.2.** QRT Vs Data Size.

Technologies for Business Intelligence (EM IT4BI-DC), the Danish Council for Independent Research (DFF) under grant agreement no. DFF-8048-00051B, Aalborg University's Talent Programme, and the Poul Due Jensen Foundation. Furthermore, Matteo Lissandrini is supported by the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 838216.

## Appendix. Complete graphs

See Figs. A.1 and A.2.

## References

[1] G. Cong, K. Feng, K. Zhao, Querying and mining geo-textual data for exploration: Challenges and opportunities, in: ICDEW, 2016.
[2] G. Cong, C.S. Jensen, Spatial keyword queries and beyond, in: SIGMOD, 2016.
[3] R. Othman, R. Belkaroui, R. Faiz, Customer opinion summarization based on Twitter conversations, in: WIMS, 2016.
[4] X. Cao, L. Chen, G. Cong, C.S. Jensen, Q. Qu, A. Skovsgaard, D. Wu, M.L. Yiu, Spatial keyword querying, in: ER, 2012.
[5] N. Gür, T.B. Pedersen, E. Zimanyi, K. Hose, A foundation for spatial data warehouses on the semantic web, Semant. Web (2017).
[6] J. Han, K. Koperski, N. Stefanovic, GeoMiner: A system prototype for spatial data mining, 1997.
[7] Y. Chen, G. Dong, J. Han, B.W. Wah, J. Wang, Multi-dimensional regression analysis of time-series data streams, in: VLDB, 2002.
[8] C.X. Lin, B. Ding, J. Han, F. Zhu, B. Zhao, Text cube: Computing IR measures for multidimensional text database analysis, in: ICDM, 2008.
[9] M. Azabou, K. Khrouf, J. Feki, C. Soulé-Dupuy, N. Vallès, Analyzing textual documents with new OLAP operators, in: AICCSA, 2016.
[10] C. Zhang, J. Han, Multidimensional mining of massive text data, in: DMKD, 2019.
[11] M.L. Chouder, S. Rizzi, R. Chalal, Exploratory OLAP over doc stores, Inf. Syst. (2019).
[12] D. Yu, D. Xu, D. Wang, Z. Ni, Hierarchical topic modeling of Twitter data for online analytical processing, IEEE Access (2019).
[13] B. Ding, B. Zhao, C.X. Lin, J. Han, C. Zhai, A. Srivastava, N.C. Oza, Efficient keyword-based search for top-K cells in text cube, IEEE Trans. Knowl. Data Eng. (2011).
[14] W. Feng, C. Zhang, W. Zhang, J. Han, J. Wang, C. Aggarwal, J. Huang, STREAMCUBE: Hierarchical spatio-temporal hashtag clustering for event exploration over the Twitter stream, in: ICDE, 2015.
[15] J. Sankaranarayanan, H. Samet, B.E. Teitler, M.D. Lieberman, J. Sperling, TwitterStand: News in tweets, in: SIGSPATIAL, 2009.
[16] M. Walther, M. Kaisser, Geo-spatial event detection in Twitter stream, in: ECIR, 2013.
[17] K. Zhao, L. Chen, G. Cong, Topic exploration in spatio-temporal document collections, in: SIGMOD, 2016.

[18] X. Liu, K. Tang, J. Hancock, J. Han, M. Song, R. Xu, B. Pokorny, A text cube approach to human social and cultural behavior in the Twitter stream, in: SBP, 2013.
[19] D. Zhang, C.X. Zhai, J. Han, A. Srivastava, N. Oza, Topic modeling for OLAP on multidimensional text databases, Stat. Anal. Data Min. (2009).
[20] J.M. Pérez-Martínez, R. Berlanga-Llavori, M.J. Aramburu-Cabo, T.B. Pedersen, Contextualizing data warehouses with documents, Decis. Support Syst. (2008).
[21] C.S. Jensen, T.B. Pedersen, C. Thomsen, Multidimensional Databases and Data Warehousing, Morgan & Claypool Publishers, 2010.
[22] F. Ravat, O. Teste, R. Tournier, G. Zurfluh, Top_keyword: An aggregation function for textual document OLAP, in: DaWaK, 2008.
[23] S. Rivest, Y. Bédard, P. Marchand, Toward better support for spatial decision making: defining the characteristics of spatial on-line analytical processing (SOLAP), Geomatica (2001).
[24] S. Wang, J. Cao, P. Yu, Deep learning for spatio-temporal data mining: A survey, IEEE Trans. Knowl. Data Eng. (2020).
[25] E. Malinowski, E. Zimányi, Hierarchies in a multidimensional model: From conceptual modeling to logical representation, Data Knowl. Eng. (2006).
[26] J.-N. Mazón, J. Lechtenbörger, J. Trujillo, A survey on summarizability issues in multidimensional modeling, Data Knowl. Eng. (2009).
[27] W. Rowen, I.-Y. Song, C. Medsker, E. Ewen, An analysis of many-to-many relationships between fact and dimension tables in dimensional modeling, Data Min. Knowl. Discov. (2001).
[28] L. Chen, G. Cong, C.S. Jensen, D. Wu, Spatial keyword query processing: An experimental evaluation, PVLDB (2013).
[29] A. Magdy, L. Abdelhafeez, Y. Kang, E. Ong, M. Mokbel, Microblogs data management: a survey, VLDB J. (2020).
[30] A. Almaslukh, A. Magdy, A.M. Aly, M.F. Mokbel, S. Elnikety, Y. He, S. Nath, W.G. Aref, Local trend discovery on real-time microblogs with uncertain locations in tight memory environments, GeoInformatica (2019).
[31] M.D. Lieberman, H. Samet, J. Sankaranarayanan, J. Sperling, STEWARD: Architecture of a spatio-textual search engine, in: GIS, 2007.
[32] B. Pat, Y. Kanza, Where's waldo?: Geosocial search over myriad geotagged posts, in: SIGSPATIAL, 2017.
[33] L. Lins, J.T. Klosowski, C.E. Scheidegger, Nanocubes for real-time exploration of spatiotemporal datasets, IEEE Trans. Vis. Comput. Graph. (2013).
[34] P. Jayachandran, K. Tunga, N. Kamat, A. Nandi, Combining user interaction, speculative query execution and sampling in the DICE system, in: ICDE, 2014.
[35] T. Joachims, Text categorization with support vector machines: Learning with many relevant features, in: ECML, 1998.
[36] J.D. Knijff, F. Frasincar, F. Hogenboom, Domain taxonomy learning from text: The subsumption method versus hierarchical clustering, Data Knowl. Eng. (2013).
[37] C. Fellbaum, WordNet: An Electronic Lexical Database, MIT Press, 1998.
[38] J.F. Kenney, E.S. Keeping, Mathematics of statistics, part 1, chapter 15, 1962, Van Nostrand.
[39] J. Gray, A. Bosworth, A. Lyaman, H. Pirahesh, Data cube: a relational aggregation operator generalizing GROUP-BY, CROSS-TAB, and SUB-TOTALS, in: ICDE, 1996.
[40] T.B. Pedersen, C.S. Jensen, Multidimensional data modeling for complex data, in: M. Kitsuregawa, M.P. Papazoglou, C. Pu (Eds.), Proceedings of the 15th International Conference on Data Engineering, Sydney, Australia, March 23-26, 1999, IEEE Computer Society, 1999, pp. 336–345.
[41] V. Harinarayan, A. Rajaraman, J.D. Ullman, Implementing data cubes efficiently, in: SIGMOD, 1996.
[42] A. Skovsgaard, D. Sidlauskas, C.S. Jensen, Scalable top-k spatio-temporal term querying, in: ICDE, 2014.
[43] R. Feldman, M. Fresko, Y. Kinar, Y. Lindell, O. Liphstat, M. Rajman, Y. Schler, O. Zamir, Text mining at the term level, in: PKDD, 1998.
[44] H. Gupta, Selection of views to materialize in a data warehouse, in: ICDT, 1997.
[45] GeoNames, 2020, http://download.geonames.org/. (Accessed 09 September 2020).
[46] C. Manning, M. Surdeanu, J. Bauer, J. Finkel, S. Bethard, D. McClosky, The stanford CoreNLP natural language processing toolkit, in: ACL, 2014.