Hierarchical Visual Exploration and Analysis on the Web of Data

Nikos Bikakis [a,b], George Papastefanatos [b], Melina Skourla [a] and Timos Sellis [c]
[a] *National Technical University of Athens, Greece*
[b] *IMIS, ATHENA Research Center, Greece*
[c] *RMIT University, Australia*

Abstract. The purpose of data visualization is to offer intuitive ways for information perception and manipulation, especially for non-expert users. Most traditional visualization tools and methods operate on an offline way requiring from users to access data locally. They also restrict themselves on dealing with small dataset sizes, which can be easily visually analysed with conventional visualization techniques. However, the *Web of Data* has realized the availability of a great amount and variety of big interlinked datasets that are dynamic in nature; most of them offer SPARQL or API endpoints for online access and analysis. Modern visualization techniques must address the challenge for ad-hoc visualizations of large dynamic sets of data offering efficient data organization and exploration techniques. Moreover, they must take into account user-defined exploration scenarios and visualization preferences. In this work, we present a model for building, visualizing, and interacting with hierarchically organized numeric and temporal *Linked Data* (LD). Our model is built on top of a lightweight tree-based structure which can be efficiently constructed on-the-fly for a given set of data. This tree structure organizes input objects into a hierarchical multi-level model based on the objects' values. Additionally, we define two versions of this structure, which adopts different data organization approaches, well-suited to visual exploration and analysis context. Furthermore, statistical computations can be efficiently performed on-the-fly in the proposed structure. Considering different exploration scenarios over large datasets, the proposed model enables efficient multi-level exploration, offering incremental construction via user interaction, and dynamic adaptation of the hierarchies based on user's preferences. The proposed model is realized in a Web-based prototype tool, called *rdf:SynopsViz* that offers multi-level visual exploration and analysis over LD datasets. Finally, we provide a performance evaluation and a empirical user study employing real datasets.

Keywords: Linked Data, Semantic Web, Visual analytics, Hierarchical exploration, RDF Data.

1. Introduction

The purpose of data visualization is to offer intuitive ways for information perception and manipulation that essentially amplify, especially for non-expert users, the overall cognitive performance of information processing. This is of great importance in the *Web of Data*, where the volume and heterogeneity of available information make it difficult for humans to manually explore and analyse large datasets. Moreover, information visualization enables users to infer correlations and causalities and supports sense-making activities [22] over data that are not always possible with traditional data mining techniques.

Following the abundance of *Linked Data* (LD) on the Web, several recent efforts have offered tools and techniques for information visualization and visual exploration of LD in many different domains, such as statistical, biological, spatiotemporal [23,51]. They offer different ways to consumers of LD to interact with the data and perform visual operations, such as navigation and browsing over resources, visual representation and analysis of resources and the relationships between them, as well as visual querying and filtering. These are further specialized to the requirements of the specific domain; e.g., graph-based visualizations are usually used for link-based analysis, maps are employed for spatial-enriched data, charts and timelines focus on numerical and temporal data analysis, etc.

One of the major challenges in the LD visual exploration is related to the size that characterizes many popular LD datasets. Considering the visual information seeking mantra: "*overview first, zoom and filter, then details on demand*" [61], gaining overview is a crucial task in the visual exploration scenario. However, offering an overview of a large dataset, is an extremely challenged task. Additionally, information overloading is a common issue in large dataset visualization; a basic requirement for the proposed approaches is to offer mechanisms for information abstraction and summarization.

The above challenges can be overcome by adopting hierarchical approaches [27]. Hierarchical approaches (a.k.a. aggregation-based techniques) allow the visual exploration of very large datasets, offering a dataset overview, as well as an intuitive and usable way for finding specific parts within a dataset. Particularly, in hierarchical approaches, the user first obtains an overview of the dataset (both structure and a summary of its content) before proceeding to data exploration operations, such as roll-up and drill-down, filtering out a specific part of it and finally retrieving details about the data. Therefore, hierarchical approaches directly support the visual information seeking mantra. Also, hierarchical approaches can effectively address the problem of information overloading as it provides information abstraction and summarization.

A second challenge is related to the availability of SPARQL endpoits for online data access and retrieval, possessing the challenge of visualizing data that is dynamically retrieved by a remote site. In this respect, visualization techniques must offer scalability and efficient processing for on-the-fly analysis and visualization of dynamic datasets. Finally, the requirement for online visualization must be coupled with the diversity of preferences and requirements posed by different users and tasks. Therefore, the proposed approaches should provide the user with the ability to customize the exploration experience, allowing users to organize data into different ways according to the type of information or the level of details she wishes to visualize.

Considering the general problem of visualizing big datasets, most approaches aim at providing appropriate summaries and abstractions over the enormous number of available data objects. Beyond hierarchical aggregations, a variety of them adopt data reduction methods based on filtering and sampling [11,49]. Recently, incremental sampling and filtering is used to approximate answers computed over progressively larger samples of the data [39,29,2]. In contrast, however, with the hierarchical techniques, the aforementioned approaches do not offer multi-level abstractions and exploration functionality.

Addressing the aforementioned challenges, in this work, we introduce a generic model that combines user-customized hierarchical exploration with online analysis of numeric and temporal data. At the core lies a lightweight hierarchical model, constructed on-the-fly for a given set of data. The proposed model is a tree-based structure that aggregates data objects into multiple levels of hierarchically related groups based on objects' numeric and temporal values. Our model also enriches groups with statistical information regarding their content, offering richer overviews and insights into the detailed data. An additional feature is that it allows users to organize data exploration in different ways, by parametrizing the number of groups, the range and cardinality of their contents, the number of hierarchy levels, etc. On top of this model, we propose three multi-level user exploration scenarios and present two methods for efficient exploration: the first one achieves the incremental construction of the model based on the user interaction, whereas the second one enables the online adaptation of the model to the user's preferences. Finally, the proposed model is realized in a Web-based tool, called *rdf:SynopsViz* that offers a variety of visualization techniques, such as charts, timelines, and treemaps for hierarchical visual exploration and analysis over LD datasets.

Contributions. The main contributions of this paper are summarized as follows.

- We introduce a model for building, visualizing, and interacting with hierarchically organized numeric and temporal LD.
- We implement our model as a lightweight, main memory tree-based structure, which can be efficiently constructed on-the-fly.
- We propose two tree structure versions, which adopt different approaches for the data organization.
- We describe a simple method to estimate the tree construction parameters, when no user preferences are available.
- We define different exploration scenarios assuming various user exploration preferences.
- We introduce a method that incrementally constructs the hierarchy tree via user interaction.
- We propose a method that dynamically adapts an existing hierarchy to a new set of user-defined parameters.

– We develop a prototype system which implements the presented model, offering multi-level visual exploration and analysis over LD.
– We conduct a thorough performance evaluation and an empirical user study, using the DBpedia 2014 dataset.

Outline. The remainder of this paper is organized as follows. Section 2 presents the proposed hierarchical model, and Section 3 provides the exploration scenarios and methods for efficient hierarchical exploration. Then, Section 4 presents the rdf:SynopsViz system and demonstrate the basic functionality. The evaluation of our system is presented in Section 5. Section 6 reviews related work, while Section 7 concludes this paper.

2. The HETree Model

In this section we present HETree (**H**ierarchical **E**xploration **Tree**), our model for building, visualizing, and interacting with hierarchically organized LD. HETree is defined in the context of hierarchical visual exploration and analysis over one-dimensional numeric and temporal (i.e., datetime values) LD. The proposed model can be adopted by various existing visualization techniques (e.g., charts, scatterplots, timeline, etc.), offering scalable and multi-level visual representations over non-hierarchical data.

In what follows, we present some basic aspects of our working scenario (i.e., visual exploration and analysis scenario) and highlight the main assumptions and requirements employed in the construction of our model. First, the input data in our scenario can be retrieved directly from a triple store, but also produced dynamically; i.e., either from a SPARQL query or from data filtering (e.g., faceted browsing). Thus, we consider that data visualization is performed online; i.e., we do not assume an offline preprocessing phase in the construction of the visualization model. Second, users can specify different requirements or preferences with respect to the data organization. For example, a user prefers to organize the data as a deep hierarchy for a specific task, while for another task a flat hierarchical organization is more appropriate. Therefore, even if the data is not dynamically produced, the data organization is dynamically adapted to the user preferences. The same also holds for any additional information (e.g., statistical information) that is computed for each group of objects. This information must be recomputed when the groups of objects (i.e., data organization) are modified.

From the above, a basic requirement is that the model must be constructed on-the-fly for any given data and users preferences. Therefore, we implement our model as a lightweight, main memory tree structure, which can be efficiently constructed on-the-fly. We define two versions of this tree structure, following data organization approaches well-suited to visual exploration and analysis context: the first version considers fixed-range groups of data objects, whereas the second considers fixed-size groups. Finally, our structure allows efficient on-the-fly statistical computations, which are extremely valuable for the exploration and analysis scenario.

The basic idea of our model is to hierarchically group data objects based on values of one of their properties. Input data objects (i.e., RDF triples) are stored at the leaves, while internal nodes aggregate their child nodes. The root of the tree represents (i.e., aggregates) the whole dataset. The basic concepts of our model can be considered similar to a simplified version of a static 1D R-Tree [32].

Regarding the visual representation of the model and data exploration, we consider that both data objects sets (leaf nodes contents) and entities representing groups of objects (leaf or internal nodes) are visually represented enabling the user to explore the data in a hierarchical manner. Note that our tree structure organizes data in a hierarchical model, without setting any constraints on the way the user interacts with these hierarchies. As such, it is possible that different strategies can be adopted, regarding the traversal policy, as well as the nodes of the tree that are rendered in each visualization stage.

In the rest of this section, preliminaries are presented in Section 2.1. In Section 2.2, we introduce the proposed tree structure. Sections 2.3 and 2.4 present the two versions of the structure. Finally, Section 2.5 discusses the specification of the parameters required for the tree construction, and Section 2.6 presents how statistics computations can be performed over the tree.

2.1. Preliminaries

We consider an *RDF dataset* R consisting of a set of *RDF triples*. As *input data*, we assume a set of RDF triples D, where $D \subseteq R$ and triples in D have as objects either numeric (e.g., xsd:int, xsd:decimal) or temporal values (e.g., xsd:dateTime, xsd:date). In other words, we consider triples that contain as predicates datatype properties with either numeric or temporal ranges. Let tr be an RDF triple, $tr.s$, $tr.p$ and

$tr.o$ represent, respectively, the *subject*, *predicate* and *object* of the RDF triple tr.

Given input data D, S is an *ordered set* of RDF triples, produced from D, where triples are sorted based on objects' values, in ascending order. Assume that $S[i]$ denotes the i-th triple, with $S[1]$ the first triple. Then, for each $i < j$, we have that $S[i].o \leq S[j].o$. Also, $D = S$, i.e., for each $tr, tr \in D$ iff $tr \in S$.

Figure 1 presents a set of 10 RDF triples, representing persons and their ages. In Figure 1, we assume that the subjects $p0$-$p9$ are instances of a class *Person* and the predicate *age* is a datatype property with *xsd:int* range.

$p0$ *age* 35	$p5$ *age* 35
$p1$ *age* 100	$p6$ *age* 45
$p2$ *age* 55	$p7$ *age* 80
$p3$ *age* 37	$p8$ *age* 20
$p4$ *age* 30	$p9$ *age* 50

Fig. 1. Running example input data (RDF triples)

Example 1. In Figure 1, given the RDF triple $tr = p0$ *age* 35, we have that $tr.s = p0$, $tr.p = age$ and $tr.o = 35$. Also, given that all triples comprise the input data D and S is the ordered set of D based on the object values, in ascending order; we have that $S[1] = p8$ *age* 20 and $S[10] = p1$ *age* 100. □

Assume an *interval* $I = [a, b]$, where $a, b \in \mathbb{R}$; then, $I = \{k \in \mathbb{R} \mid a \leq k \leq b\}$. Similarly, for $I = [a, b)$, we have that $I = \{k \in \mathbb{R} \mid a \leq k < b\}$. Let I^- and I^+ denote the lower and upper bound of the interval I, respectively. That is, given $I = [a, b]$, then $I^- = a$ and $I^+ = b$. The *length* of an interval I is defined as $|I^+ - I^-|$.

In this work we assume rooted trees. The number of the children of a node is its *degree*. Nodes with degree 0 are called *leaf nodes*. Moreover, any non-leaf node is called *internal node*. *Sibling nodes* are the nodes that have the same parent. The *level of a node* is defined by letting the root node be at level zero. Additionally, the *height of a node* is the length of the longest path from the node to a leaf. A leaf node has a height of 0.

The *height of a tree* is the maximum level of any node in the tree. The *degree of a tree* is the maximum degree of a node in the tree. An *ordered tree* is a tree where the children of each node are ordered. A tree is called an *m-ary tree* if every internal node has no more than m children. A *full m-ary tree* is a tree where every internal node has exactly m children. A *perfect m-ary tree* is a full m-ary tree in which all leaves are at the same level.

2.2. The HETree Structure

In this section, we present in more detail the *HETree* structure. HETree hierarchically organizes numeric and temporal[1] data into groups; intervals are used to represents these groups. HETree is defined by the tree degree and the number of leaf nodes[2]. Essentially, the number of leaf nodes corresponds to the number of groups where input data objects are organized. The tree degree corresponds to the (maximum) number of groups where a group is split in the lower level.

Given a set of RDF triples D, a positive integer ℓ denoting the number of leaf nodes; and a positive integer d denoting the tree degree; an *HETree* (D, ℓ, d) is an *ordered d-ary tree*, with the following basic properties.

- The tree has exactly ℓ number of leaf nodes.
- All leaf nodes appear in the same level.
- Each leaf node contains a set of RDF triples, sorted in ascending order based on their objects' values. Given a leaf node$n$, $n.data$ denote the data objects contained in n.
- Each internal node has at most *d* children nodes. Let n be an internal node, $n.c_i$ denotes the i-th child for the node n, with $n.c_1$ be the leftmost child.
- Each node corresponds to an interval. Given a node n, $n.I$ denotes the interval for the node n.
- At each level, all nodes are sorted based on the lower bounds of their intervals. That is, let n be an internal node, for any $i < j$, we have that $n.c_i.I^- \leq n.c_j.I^-$.
- For a leaf node, its interval is bounded by the objects' values of the triples included in this leaf node. Let n be the leftmost leaf node; assume that n contains x triples from D. Then, we have that $n.I^- = S[1].o$ and $n.I^+ = S[x].o$, where S is the ordered RDF set resulted from D.
- For an internal node, its interval is bounded by the union of the intervals of its children. That is, let n be an internal node, having k child nodes; then, we have $n.I^- = n.c_1.I^-$ and $n.I^+ = n.c_k.I^+$.

[1]Note that our structure handles numeric and temporal data in a similar manner. Also, other types of one-dimensional data may be supported, with the requirement that a total order can be defined over the data.

[2]Note that following a similar approach, the HETree can also be defined by specifying the tree height instead of degree or number of leaves.

We, furthermore, present two different approaches for organizing the data in the HETree. Assume the scenario in which a user wishes to (visually) explore and analyse the historic events from DBpedia, per decade. In this case, user orders historic events by their dates and organizes them into groups of equal ranges (i.e., decade). In a second scenario, assume that a user wishes to analyse in the Eurostat dataset the gross domestic product (GDP) organized into fixed groups of countries. In this case, the user is interested in finding information like: the range and the variance of the GDP values over the top-10 countries with the highest GDP factor. In this scenario, the user orders countries by their GDP and organizes them into groups of equal sizes (i.e., 10 countries per group).

In the first approach, we organize RDF triples into groups, where the object values of each group covers equal range of values. In the second approach, we organize RDF triples into groups, where each group contains the same number of triples. In the following sections, we present in detail the two approaches for organizing the data in the HETree.

2.3. A Content-based HETree (HETree-C)

In this section we introduce a version of the HETree, named HETree-C (Content-based HETree). This HETree version organizes data into equally sized groups. The basic property of the HETree-C is that each leaf node contains approximately the same number of RDF triples and the content (i.e., triples) of a leaf node specifies its interval. For the tree construction, the triples are first assigned to the leaves and then the intervals are defined.

An *HETree-C* (D, ℓ, d) is an HETree, with the following extra property. Each leaf node contains λ or $\lambda - 1$ triples, where[3] $\lambda = \left\lceil \frac{|D|}{\ell} \right\rceil$. Particularly, the $\ell - (\lambda \cdot \ell - |D|)$ leftmost leaves contain λ triples, while the rest leaves contain $\lambda - 1$. We can equivalently define the HETree-C by providing the number of triples per leaf λ, instead of the number of leaves ℓ.

Example 2. Figure 2 presents an HETree-C constructed by considering the set of RDF triples D from Figure 1, $\ell = 5$ and $d = 3$. As we can observe, all the leaf nodes contain equal number of triples. Particularly, we have that $\lambda = \left\lceil \frac{10}{5} \right\rceil = 2$. Considering the leftmost leaf node d, we can see that it contains two triples in the following order $p8$ *age* 20, $p4$ *age* 30. As a result, the lower bound for the its interval, is equal to the value of the first triple object, i.e., $d.I^{-} = 20$; the upper bound is equal to the value of the last triple object, i.e., $d.I^{+} = 30$. □

[3]We assume that, the number of triples is at least as the number of leaves; i.e., $|D| \geq \ell$.

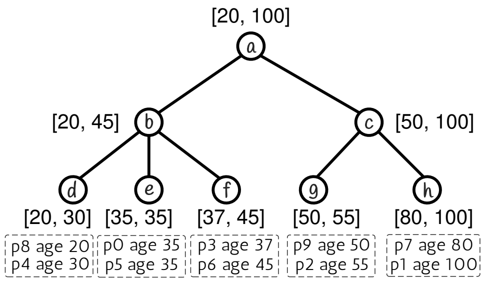


Fig. 2. A Content-based HETree (HETree-C)

2.3.1. The HETree-C Construction

We construct the HETree-C in a bottom-up way. Algorithm 1 describes the HETree-C construction. The algorithm takes as input: (1) a set of RDF triples D; (2) the number of leaf nodes ℓ; and (3) the tree degree d. Initially, the algorithm sort the RDF triples set D in ascending order, based on triples objects values (*line 1*). Then, the algorithm uses two main procedures to construct the tree. The first procedure, named constrLeaves-C creates the leaf nodes of the tree (*line 2*). The second procedure, named constrtInterNodes, creates the internal nodes of the tree (*line 3*). Finally, the root node of the constructed tree is returned (*line 4*).

Algorithm 1. createHETree-C/R (D, ℓ, d)

```
Input: $D$: set of triples; $\ell$: number of leaf nodes; $d$: tree degree
Output: $r$: root node of the HETree tree

1 $S \leftarrow$ sort $D$ based on objects values
2 $L \leftarrow$ constrLeaves-C/R($S$, $\ell$)
3 $r \leftarrow$ constrtInterlNodes($L$, $d$)
4 return $r$
```

Procedure 1 presents the pseudocode for the constrLeaves-C procedure. First, the procedure uses an ordered set of RDF triples S and creates ℓ leaf nodes containing the S triples. Then, the procedure computes the number of triples per leaf λ (*line 1*), as well as the number of leaves that contain λ triples (*line 2*). Then, ℓ leaf nodes are constructed (*lines 4–16*). For the first k leaves, λ triples are inserted, while for the rest leaves, $\lambda - 1$ triples are inserted (*lines 6–9*). The interval of each leaf is specified by the objects values of the first and last triple inserted in this leaf (*lines 13–14*). Finally, the set of created leaf nodes is returned (*line 17*).

Procedure 1: constrLeaves-C(S, ℓ)

```
Input: $S$: ordered set of triples; $\ell$: number of leaf nodes
Output: $L$: ordered set of leaf nodes
1  $\lambda \leftarrow \left\lceil \frac{|S|}{\ell} \right\rceil$
2  $k \leftarrow \ell - (\lambda \cdot \ell - |S|)$
3  $beg \leftarrow 1$
4  for $i \leftarrow 1$ to $\ell$ do
5      create an empty leaf node $n$
6      if $i \leq k$ then
7          $num \leftarrow \lambda$
8      else
9          $num \leftarrow \lambda - 1$
10     $end \leftarrow beg + num$
11     for $t \leftarrow beg$ to $end$ do
12         $n.data \leftarrow S[t]$
13     $n.I^{-} \leftarrow S[beg].o$
14     $n.I^{+} \leftarrow S[end].o$
15     $L[i] \leftarrow n$
16     $beg \leftarrow end + 1$
17 return $L$
```

Procedure 2 describes the constrtInterNodes procedure. This procedure builds the internal nodes in a recursive manner. Particularly, the procedure takes as input a set of nodes H, as well as the tree degree d. The basic idea of this procedure is the following. For the nodes H, their parents nodes P are created (*lines 4-16*); then, the procedure calls itself using as input the parent nodes P (*line 21*). The recursion terminates when the number of created parent nodes is equal to one (*line 17*); i.e., the root of the tree is created.

Computational Analysis. The computational cost for the HETree-C construction (Algorithm 1) is the sum of three parts. The first is sorting the input data, which can be done in the worst case in $O(|D|log|D|)$, employing a linearithmic sorting algorithm; e.g., mergesort. The second part is the constrLeaves-C procedure, which requires $O(|D|)$ for scanning all data object. The third part is the constrtInterNodes procedure, which requires $d \cdot (\lceil \frac{\ell}{d} \rceil + \lceil \frac{\ell}{d^2} \rceil + \lceil \frac{\ell}{d^3} \rceil + \ldots + 1)$, with the sum being the number of internal nodes in the tree. Note that the maximum number of internal nodes in a d-ary tree corresponds to the number of internal nodes in a perfect d-ary tree of the same height. Also, note the number of internal nodes of a perfect d-ary tree of height h is $\frac{d^h - 1}{d-1}$. In our case, the height of our tree is $h = \lceil log_d \ell \rceil$. Hence, the maximum number of internal nodes is $\frac{d^{\lceil log_d \ell \rceil} - 1}{d-1} \leq \frac{d \cdot \ell - 1}{d-1}$. Therefore, the constrtInterNodes procedure, in worst case requires $O(\frac{d^2 \cdot \ell - d}{d-1})$. Therefore, the overall computational cost for the HETree-C construction in the worst case is $O(|D|log|D| + |D| + \frac{d^2 \cdot \ell - d}{d-1}) = O(|D|log|D| + \frac{d^2 \cdot \ell - d}{d-1})$.

Procedure 2: constrtInterNodes(H, d)

```
Input: $H$: ordered set of nodes; $d$: tree degree
Output: $r$: root node for $H$
Variables: $P$: ordered set of $H$'s parent nodes
1  $p_{num} \leftarrow \left\lceil \frac{|H|}{d} \right\rceil$          //number of parents nodes
2  $t \leftarrow d - (p_n \cdot d - |H|)$          //last parent's number of children
3  $c_{beg} \leftarrow 1$          //first child node
4  for $p \leftarrow 1$ to $p_{num}$ do
5      create an empty internal node $n$
6      if $p = p_{num}$ then
7          $c_{num} \leftarrow t$          //number of children
8      else
9          $c_{num} \leftarrow d$
10     $c_{end} \leftarrow c_{beg} + c_{num}$          //last child node
11     for $j \leftarrow c_{beg}$ to $c_{end}$ do
12         $n.c[j] \leftarrow H[j]$
13     $n.I^{-} \leftarrow H[c_{beg}].I^{-}$
14     $n.I^{+} \leftarrow H[c_{end}].I^{+}$
15     $P[p] \leftarrow n$
16     $c_{beg} \leftarrow c_{end} + 1$
17 if $p_{num} = 1$ then
18     $r \leftarrow P$
19     return $r$
20 else
21     return constrtInterINodes($P$, $d$)
```

2.4. A Range-based HETree (HETree-R)

The second version of the HETree is called HETree-R (Range-based HETree). HETree-R organizes data into equally ranged groups. The basic property of the HETree-R is that each leaf node covers an equal range of values. Therefore, in HETree-R, the data space defined by the objects values is equally divided over the leaves. In opposition to HETree-C, in HETree-R the interval of a leaf specifies its content. Therefore, for the HETree-R construction, the intervals of all leaves are first defined and then triples are inserted.

An *HETree-R* (D, ℓ, d) is an HETree, with the following extra property. The interval of each leaf node has the same length; i.e., covers equal range of values. Formally, let S be the sorted RDF set resulted from D, for each leaf node its interval has length ρ, where[4] $\rho = \frac{|S[1].o - S[|S|].o|}{\ell}$. Therefore, for a leaf node n, we have that $|n.I^{-} - n.I^{+}| = \rho$. For example, for the leftmost leaf, its interval is $[S[1].o, S[1].o + \rho)$. The HETree-R is equivalently defined by providing the interval length ρ, instead of the number of leaves ℓ.

Example 3. Figure 3 presents an HETree-R tree constructed by considering the set of triples D (Figrue 1), $\ell = 5$ and $d = 3$. As we can observe from Figure 3, each leaf node covers equal range of val-

[4]We assume here that, there is at least one triple in D with different object value than the rest triples.

ues. Particularly, we have that the interval of each leaf must have length $\rho = \frac{|20-100|}{5} = 16$. □

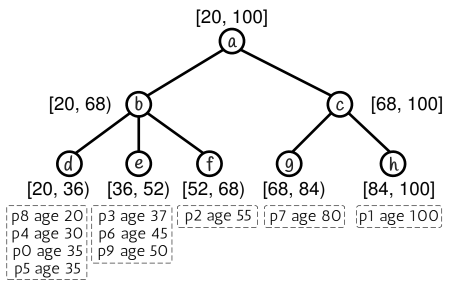


Fig. 3. A Range-based HETree (HETree-R)

2.4.1. *The HETree-R Construction*

This section studies the construction of the HETree-R structure. The HETree-R is also constructed in a bottom-up fashion.

Similarly with the HETree-C version, Algorithm 1 is used for the HETree-R construction. The only difference is the constrLeaves-R procedure (*line 2*), which creates the leaf nodes of the HETree-R and is presented in Procedure 3.

The procedure takes as input an ordered set of RDF triples S, as well as the number of leaves nodes ℓ. First, it computes the range ρ of the leaves (*line 1*). Then, it constructs ℓ leaf nodes (*lines 2–9*) and assigns same intervals to all of them (*lines 4–8*), it traverses all RDF triples in S (*lines 10–12*) and places them to the appropriate leaf node (*line 12*). Finally, returns the set of created leaves (*line 13*).

Procedure 3: constrLeaves-R(S, ℓ)

```
Input: $S$: ordered set of triples; $\ell$: number of leaf nodes
Output: $L$: ordered set of leaf nodes
1  $\rho \leftarrow \frac{|S[1].o - S[|S|].o|}{\ell}$
2  for $i \leftarrow 1$ to $\ell$ do
3      create an empty leaf node $n$
4      if $i = 1$ then
5          $n.I^- \leftarrow S[1].o$
6      else
7          $n.I^- \leftarrow L[i-1].I^+$
8      $n.I^+ \leftarrow n.I^- + \rho$
9      $L[i] \leftarrow n$
10 for $t \leftarrow 1$ to $|S|$ do
11     $j \leftarrow \left\lfloor \frac{S[t].o - S[1].o}{\rho} \right\rfloor + 1$
12     $L[j].data \leftarrow S[t]$
13 return $L$
```

Computational Analysis. The computational cost for the HETree-R construction (Algorithm 1) for sorting the input data (*line 1*) and creating the internal nodes (*line 3*) is the same as in the HETree-C case. The constrLeaves-R procedure (*line 2*) requires $O(\ell + |D|) = O(|D|)$ (since $|D| \geq \ell$). Using the computational costs for the first and the third part from Section 2.3.1, we have that in worst case, the overall computational cost for the HETree-R construction is $O(|D|log|D| + |D| + \frac{d^2 \cdot \ell - d}{d-1}) = O(|D|log|D| + \frac{d^2 \cdot \ell - d}{d-1})$.

2.5. *Estimating the HETree Parameters*

In our working scenario, the user specifies the parameters required for the HETree construction (e.g., number of leaves ℓ). In this section, we describe our approach for automatically calculating the HETree parameters based on the input data, when no user preferences are provided. Our goal is to derive the parameters by the input data, such that the resulting HETree can address some basic guidelines set by the visualization environment. In what follows, we discuss in detail the proposed approach.

An important parameter in hierarchical visualizations is the minimum and maximum number of objects that can be effectively rendered in the most detailed level[5]. In our case, the above numbers correspond to the number of triples contained in the leaf nodes. The proper calculation of these numbers is crucial such that the resulted tree avoids overloaded and scattered visualizations.

Therefore, in HETree construction, our approach considers the minimum and the maximum number of triples per leaf node, denoted as λ_{min} and λ_{max}, respectively. Besides the number of objects rendered in the lowest level, our approach consider perfect m-ary trees, such that a more "uniform" structure (i.e., all the groups are divided into same number of groups) is resulted. The following example illustrates our approach to calculate the HETree parameters.

Example 4. Assume that based on an adopted visualization technique, the ideal number of data objects to be rendered on a specific screen is between 25 and 50. Hence, we have that $\lambda_{min} = 25$ and $\lambda_{max} = 50$.

Now, let's assume that we want to visualize the RDF triples set D_1, using an HETree-C, where $|D_1| = 500$. Based on the number of triples and the λ bounds, we can estimate the bounds for the number of leaves. Let ℓ_{min} and ℓ_{max} denote the lower and the upper bound for the number of leaves. There-

[5]Similar bounds can also be defined for other tree levels.

fore, we have that $\left\lceil \frac{|D_1|}{\lambda_{max}} \right\rceil \leq \ell \leq \left\lceil \frac{|D_1|}{\lambda_{min}} \right\rceil \Leftrightarrow$ $\left\lceil \frac{500}{50} \right\rceil \leq \ell \leq \left\lceil \frac{500}{25} \right\rceil \Leftrightarrow 10 \leq \ell \leq 20$.

Hence, our HETree-C should have between $\ell_{min} = 10$ and $\ell_{max} = 20$ leaf nodes. Since, we consider perfect m-ary trees, from Table 1 we can identify the tree characteristics that conform to the number of leaves guideline. The candidate settings (i.e., leaf number and degree) are indicated in Table 1, using dark-grey colour. The setting with $d = 2$ is rejected since visualizing two groups of objects in each level, can be considered a small number under most visualization settings. Note that, in our work, in any case we assume settings with $d \geq 3$ and $height \geq 2$. Therefore, an HETree-C with $\ell = 16$ and $d = 4$ is a suitable structure for our case.

Now, let's assume that we want to visualize the RDF triples set D_2, where $|D_2| = 1000$. Following a similar approach, we have that $20 \leq \ell \leq 40$. The candidate settings are indicated in Table 1 using light-grey colour. Also, here the setting with $d = 2$, is rejected. Hence, we have the following settings that satisfy the considered guideline: $S1$: $\ell = 27$, $d = 3$; $S2$: $\ell = 25$, $d = 5$; and $S3$: $\ell = 36$, $d = 6$.

In the case, where more than one setting satisfies the considered guideline, we select the preferable one according to following set of rules. From the candidate settings, we prefer the setting which results in the highest tree[6]. (1st *Criterion*). In case that the highest tree is constructed by more than one settings, we consider the distance c, between ℓ and the centre of ℓ_{min} and ℓ_{max} (2nd *Criterion*); i.e., $c = |\ell - \frac{\ell_{min}+\ell_{max}}{2}|$. The setting with the lowest c value is selected. Note that, based on the visualization context, different criteria and preferences may be followed.

In our example, from the candidate settings, the setting $S1$ is selected, since it will construct the highest tree (i.e., $height = 3$). On the other hand, the settings $S2$ and $S3$ will construct trees with lower heights (i.e., $height = 2$).

Now, assume a scenario where only the settings $S2$ and $S3$ are candidates. In this case, since both settings result to trees with equal heights, the 2nd *Criterion* is considered. Hence, for the $S2$ setting we have $c_2 = |25 - \frac{20+40}{2}| = 5$. Similarly, for the $S3$ setting $c_3 = |36 - \frac{20+40}{2}| = 6$. Therefore, between the settings $S2$ and $S3$, the setting $S2$ is preferable, since $c_2 < c_3$.

In case of HETree-R, a similar approach is followed, assuming normal distribution over the values of the triples objects. □

[6]Depending on the scenario, the shortest tree may be preferable

Table 1. Number of leaf nodes for perfect m-ary trees

	Degree				
Height	**2**	**3**	**4**	**5**	**6**
1	2	3	4	5	6
2	4	9	16	25	36
3	8	27	64	625	216
4	16	81	256	3125	1296
5	32	243	1024	15625	7776
6	64	729	4048	78125	46656

2.6. Statistics Computations over HETree

Data statistics is a crucial aspect in the context of hierarchical visual exploration and analysis. Statistical informations over groups of objects offer rich insights into the underlying data. In this way, useful information regarding different set of objects with common characteristics is provided. Additionally, this information may also guide the users through their navigation over the hierarchy.

In this section, we present how statistics computation is performed over the nodes of the HETree. Statistics computations exploit two main aspects of the HETree structure: (1) the internal nodes aggregate their child nodes; and (2) the tree is constructed in bottom-up fashion. Statistics computation is performed during the tree construction; for the leaf nodes, we gather statistics from the triples they contain, whereas for the internal nodes we aggregate the statistics of their children.

For simplicity, here, we assume that each node contains the following extra fields, used for simple statistics computations, although more complex or RDF-related (e.g., most common subject, subject with the minimum value, etc.) statistics can be computed. Assume a node n, as $n.N$ we denote the *number* of RDF triples covered by n; as $n.\mu$ and $n.\sigma^2$ we denote the mean and the variance of the triples objects values covered by n, respectively. Additionally, we assume the minimum and the maximum values, denoted as $n.min$ and $n.max$, respectively.

Statistics computations can be easily performed in the construction algorithms (Algorithm 1) without any modifications. The follow example illustrates these computations.

Example 5. In this example we assume the HETree-C presented in Figure 2. Figure 4 shows the HETree-C with the computed statistics in each node. When all the leaf nodes have been constructed, the statistics for each leaf is computed. For instance, we can see from Figure 4, that for the rightmost leaf h we have: $h.N = 2$, $h.\mu = \frac{80+100}{2} = 90$ and $h.\sigma^2 = \frac{1}{2} \cdot ((80-90)^2 + (100-90)^2) = 100$. Also, we have $h.min = 80$ and $h.max = 100$. Following the above process, we compute the statistics for all leaf nodes.

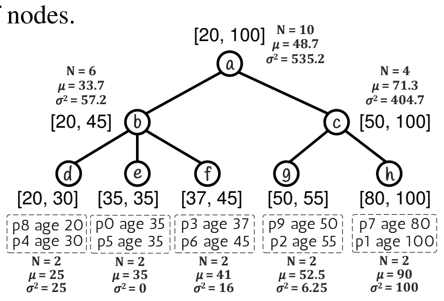


Fig. 4. Statistics computation over HETree

Then, for each parent node we construct, we compute its statistics using the computed statistics of its child nodes. Considering the c internal node, with the child nodes g and h, we have that $c.min = 50$ and $c.max = 100$. Also, we have that $c.N = g.N + h.N = 2 + 2 = 4$. Now we will compute the mean value by combining the children mean values: $c.\mu = \frac{g.N \cdot g.\mu + h.N \cdot h.\mu}{g.N + h.N} = \frac{2 \cdot 52.5 + 2 \cdot 90}{2+2} = 71.3$. Similarly, for variance we have $c.\sigma^2 = \frac{g.N \cdot g.\sigma^2 + h.N \cdot h.\sigma^2 + g.N \cdot (g.\mu - c.\mu)^2 + h.N \cdot (h.\mu - c.\mu)^2}{g.N + h.N} = \frac{2 \cdot 6.25 + 2 \cdot 100 + 2 \cdot (52.5 - 71.3)^2 + 2 \cdot (90 - 71.3)^2}{2+2} = 404.7$.

The similar approach is also followed for the case of HETree-R. □

Computational Analysis. Most of the well known statistics (e.g., mean, variance, skewness, etc.) can be computed linearly w.r.t. the number of elements. Therefore, the computation cost over a set of numeric values S is considered as $O(|S|)$. Assume a leaf node n containing k triples, then the cost for statistics computations for n is $O(k)$. Also, the cost for all leaf nodes is $O(|D|)$. Let an internal node n, then the cost for n is $O(d)$; since the statistics in n are computed by aggregating the statistics of the d child nodes. Considering that $\frac{d \cdot \ell - 1}{d-1}$ is the maximum number of internal nodes (Section 2.3.1), we have that in the worst case the cost for the internal nodes is $O(\frac{d^2 \cdot \ell - d}{d-1})$. Therefore, the overall cost for statistics computations over an HETree is $O(|D| + \frac{d^2 \cdot \ell - d}{d-1})$.

3. Efficient Hierarchical Exploration

In this section, we exploit the HETree structure in order to efficiently handle different exploration scenarios over hierarchically organized content. Essentially, we propose two methods for efficient hierarchical exploration over large datasets. The first method incrementally constructs the hierarchy via user interaction; the second one achieves dynamic adaptation of the data organization based on user's visual preferences.

3.1. Exploration Scenarios

In a typical hierarchical exploration scenario, referred here as *Basic exploration scenario* (BSC), the user explores a dataset in a top-down fashion. The user first obtains an overview of the data through the root level, and then drills down to more fine-grained contents for accessing the actual data objects (i.e., LD resources) at the leaves. In BSC, the root of the hierarchy is the starting point of the exploration and, thus, the first element to be visualized.

The described scenario offers basic exploration capabilities; however it does not assume use cases with user-specified starting points, other than the root, such as starting the exploration from a specific resource, or from a specific range of values.

Consider the following example, in which the user wishes to explore the *DBpedia* infoboxes dataset to find places with very large population. Initially, she selects the *populationTotal* property and starts her exploration from the root node, moves down the right part of the tree and ends up at the rightmost leaf that contains the highly populated places. Then, she is interested in viewing the area size (i.e., *areaTotal* property) for one of the highly populated places and, also, in exploring places with similar area size. Finally, she decides to explore places based on the water area size (i.e., *areaWater*) they contain. In this case, she prefers to start her exploration by considering places that their water area size is within a given range of values.

In this example, besides BSC one we consider two additional exploration scenarios. In the *Resource-based exploration scenario* (RES), the user specifies a resource of interest (i.e., an IRI) and a specific property; the exploration starts from the leaf containing the specific resource and proceeds in a bottom-up fashion.

Thus, in RES the data objects contained in the same leaf with the resource of interest are visualized first. We refer to that leaf as *leaf of interest*.

The second scenario, named *Range-based exploration scenario* (RAN) enables the user to start her exploration from an arbitrary point in the hierarchy providing a range of values; the user starts from a set of internal nodes and she can then move up or down the hierarchy. The RAN scenario begins by visualizing all sibling nodes that are children of the node covering the specified range of interest; we refer to these nodes as *nodes of interest*.

Note that, regarding the adopted rendering policy for all scenarios, we only consider nodes belonging to the same level. That is, sibling nodes or data objects contained in the same leaf, are visualized.

Regarding the "navigation-related" *operations*, the user can move down or up the hierarchy by performing a *drill-down* or a *roll-up* operation, respectively. A drill-down operation over a node n enables the user to focus on n and visualize its child nodes. If n is a leaf node, the set of data objects (i.e., LD resources) contained in n are visualized. On the other hand, the user can perform a *roll-up* operation on a set of sibling nodes S. The parent node of S along with the parent's sibling nodes are visualized. Finally, the roll-up operation when applied to a set of data objects O will visualize the leaf node that contains O along its sibling leaves, whereas a drill-down operation is not applied to a data object.

3.2. Interactive HETree Construction

In the Web of Data, the dataset might be dynamically retrieved by a remote site (e.g., via an SPARQL endpoint), as a result, in all exploration scenarios, we have assumed that the HETree is constructed on-the-fly at the time the user starts her exploration. In the previous DBpedia example, the user explores three different properties; although only a small part of their hierarchy is accessed, the whole hierarchies are constructed and the statistics of all nodes are computed. Considering the recommended HETree parameters for the employed properties, this scenario requires that 29.5K nodes will be constructed for *populationTotal* property, 9.8K nodes for the *areaTotal* and 3.3K nodes for the *areaWater*, amounting to a total number of 42.6K nodes. However, the construction of the hierarchies for large datasets poses a time overhead (as shown in the experimental section) and, consequently, increased response time in user exploration.

In this section, we introduce ICO (**I**nteractive HETree **Co**nstruction) method, which incrementally constructs the HETree, based on user interaction. The proposed method goes beyond the incremental tree construction, aiming at further reducing the response time during the exploration process by "pre-constructing" the parts of the tree that will be visited by the user in her next roll-up or drill-down operation. Hence, a node n is not constructed when the user visits it for the first time; instead, it has been constructed in a previous exploration step, where the user was on a node in which n can be reached by a roll-up or a drill-down operation. This way, our method offers incremental construction of the tree, tailored to each user's exploration. Finally, we show that, during an exploration scenario, ICO constructs the minimum number of HETree elements.

Employing ICO method in the DBpedia example, the *populationTotal* hierarchy will only construct 76 nodes (the root along its child nodes and 9 nodes in each of the lower tree levels) and the *areaTotal* will construct 3 nodes corresponding to the leaf node containing the requested resource and its siblings. Finally, the *areaWater* hierarchy initially will contain either 6 or 15 nodes, depending on whether the user's input range corresponds to a set of sibling leaf nodes, or to a set of sibling internal nodes, respectively.

Example 6. We demonstrate the functionality of ICO through the following example. Assume the dataset used in our running examples, describing persons and their ages. Figure 5 presents the interactive construction of the HETree presented in Figure 3 for the RES and RAN exploration scenarios. Blue color is used to indicate the HETree elements that are visualized (rendered) to the user, in each exploration stage.

In the RES scenario (upper flow in Figure 5), the user specifies "http://persons.com/p6" as her resource of interest; all data objects contained in the same leaf (i.e., e) with the resource of interest are initially rendered to the user. The ICO initially constructs the leaf e, along with its siblings, i.e., leaves d and f. These leaves correspond to the nodes that the user can reach in a next (roll-up) step. Next, the user rolls up and the leaves d, e and f are rendered to her. At the same time, parent node b and its sibling c are constructed. Note that all elements which are accessible to the user by moving either down (i.e., d, e, f data objects), or up (i.e., b, c nodes) are already constructed. Finally, when the user rolls up b and c

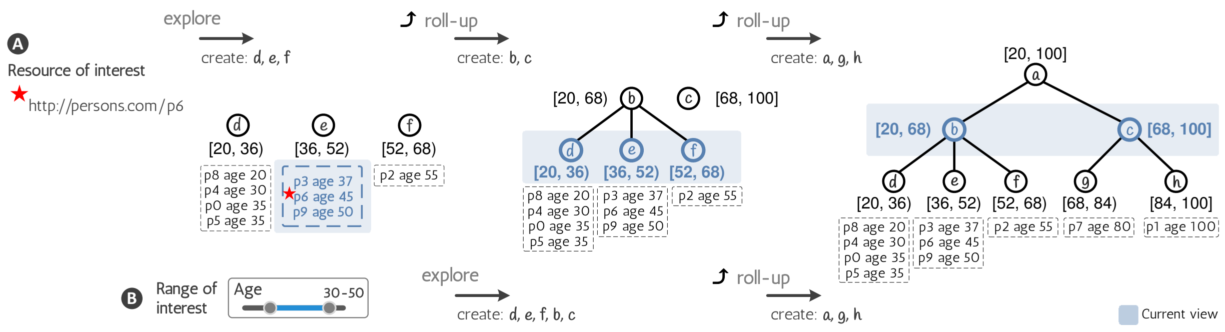


Fig. 5. Interactive HETree construction example. A) Resource-based (RES) exploration scenario; B) Range-based (RAN) exploration scenario

nodes are rendered and parent node a, along with the children of c, i.e., g and h, are constructed.

In the RAN scenario (lower flow in Figure 5), the user specifies [20, 50] as her range of interest. The nodes covering this range (i.e., d, e) are initially visualized along with their sibling f. Also, ICO constructs the parent node b and its sibling c because they are accessible by one exploration step. Then, the user performs a roll-up and ICO constructs the a, g, h nodes (as described in the RES scenario above). $\square$

In the beginning of each exploration scenario, ICO constructs a set of *initial nodes*, which are the nodes initially visualized, as well as the nodes potentially reached by the user's first operation (i.e., required HETree elements). The *required HETree elements* of an exploration step are nodes that can be reached by the user by performing one exploration operation. Hence, in the RES scenario, the initial nodes are the leaf of interest and its sibling leaves. In the RAN, the initial nodes are the nodes of interest, their children, and their parent node along with its siblings. Finally, in the BSC scenario the initial nodes are the root node and its children.

In what follows we describe the *construction rules* adopted by ICO through the user exploration process. These rules provide the correspondences between the types of elements visualized in each exploration step and the elements that ICO constructs. Note that these rules are applied after the construction of the initial nodes, in all three exploration scenarios. The correctness of these rules is verified later in Proposition 1.

Rule 1: *If a set of internal sibling nodes C is visualized*, ICO constructs: (i) the parent node of C along with the parent's siblings, and (ii) the children of each node in C.

Rule 2: *If a set of leaf sibling nodes L is visualized*, ICO does not construct anything (the required nodes have been previously constructed).

Rule 3: *If a set of data objects O is visualized*, ICO does not construct anything (the required nodes have been previously constructed).

The following proposition shows that, in all case, the required HETree elements have been constructed earlier by ICO[7].

Proposition 1. In any exploration scenario, the HETree elements, a user can reach by performing one operation (i.e., required elements), have been previously constructed by ICO.

Also, the following theorem shows that over any exploration scenario ICO constructs only the required HETree elements.

Theorem 1. ICO constructs the minimum number of HETree elements in any exploration scenario.

3.2.1. *ICO Algorithm*

In this section, we present the interactive HETree construction algorithm[8].

Here, we assume that each node n contains the following extra fields. Let a node n, $n.p$ denotes the parent node of n, and $n.h$ denotes the height of n in the hierarchy. Additionally, given a dataset D, $D.minv$ and $D.maxv$ denote the minimum and the maximum value for all objects in D, respectively. The user preferences regarding the exploration's starting point are represented as an interval U. In the RES scenario, given

[7]Proofs are included in Appendix A.

[8]Here, we include the pseudocode only for the HETree-R version, since the only difference with the HETree-C version is in the way that the nodes' intervals are computed and that the dataset is initially sorted. In the analysis of the algorithms, both versions are studied.

Algorithm 2. ICO-R(D, ℓ, d, U, cur, H)

```
Input: $D$: set of triples; $\ell$: number of leaf nodes; $d$: tree degree;
       $U$: interval representing user's starting point; $cur$: currently
       visualized elements; $H$: currently created HETree-R
Output: $H$: updated HETree-R
Variables: $len$: the length of the leaf's interval
1  if $cur$ = null then                                  //first ICO call
2      $len \leftarrow \frac{D.maxv - D.minv}{\ell}$
3      from $U$ compute $I_0$, $h_0$        //used for constructing initial nodes
4      $cur$ , $H \leftarrow$ constrSiblingNodes-R($I_0$, null, $D$, $h_0$)
5      if RES then return $H$
6  if $cur[1].p$ = null and $D \neq \varnothing$ then
7      $H \leftarrow$ constrRollUp-R($D$, $d$, $cur$, $H$)
8      if $cur[1].h > 0$ then                            //$cur$ are not leaves
9          $H \leftarrow$ constrDrillDown-R($D$, $d$, $cur$, $H$)
10 return $H$
```

that the value of the visualized property for the resource of interest is o, we have $U^- = U^+ = o$. In the RAN scenario, given that the range of interest is R, we have that $U^- = \max(D.minv, R^-)$ and $U^+ = \min(D.maxv, R^+)$. In the BSC scenario, the user does not provide any preferences regarding the starting point, so we have $U^- = D.minv$ and $U^+ = D.maxv$. Finally, according to the definition of HETree, a node n *encloses* a triple (i.e., data object) tr if $n.I^- \geq tr.o$ and $n.I^+ \leq tr.o$.

The algorithm ICO-R (Algorithm 2) implements the interactive method for HETree-R. The algorithm uses two procedures to construct all required nodes. The first procedure constrRollUp-R (Procedure 4) constructs the nodes which can be reached by a roll-up operation, whereas constrDrillDown-R (Procedure 5) constructs the nodes which can be reached by a drill-down operation. Additionally, the aforementioned procedures exploit two secondary procedures (available in Appendix A): computeSiblingInterv-R and constrSiblingNodes-R, which are used for nodes' intervals computations and nodes construction.

The ICO-R algorithm is invoked at the beginning of the exploration scenario, in order to construct the initial nodes, as well as every time the user performs an operation. The algorithm takes as input the dataset D, the tree parameters d and ℓ, the starting point U, the currently visualized elements cur, and the constructed HETree H. ICO-R begins with the currently visualized elements cur equal to $null$ (*lines 1-5*). Based on the starting point U, the algorithm computes the interval I_0 corresponding to the sibling nodes that are first visualized to the user, as well as its hierarchy height h_0 (*line 3*). For sake of simplicity, the details for computing I_0 and h_0 are omitted. For example, the interval I for the leaf that contains the resource of interest with object value o, is computed as $I^- = D.minv + len \cdot \left\lfloor \frac{o - D.minv}{len} \right\rfloor$ and $I^+ = \min(D.maxv, I^- + len)$. Following a similar approach, we can easily compute I_0 and h_0.

Procedure 4: constrRollUp-R(D, d, cur, H)

```
Input: $D$: set of triples; $d$: tree degree; $cur$: currently visualized
       elements; $H$: currently created HETree-R
Output: $H$: updated HETree-R
//Computed in ICO-R: $len$: the length of the leaf's interval
1  create an empty node $par$                             //$cur$ parent node
2  $par.h \leftarrow cur[1].h + 1$
3  $par.I^- \leftarrow cur[1].I^-$
4  $par.I^+ \leftarrow cur[|cur|].I^+$
5  for $i \leftarrow 1$ to $|cur|$ do                  //create parent-child relations
6      $par.c[i] \leftarrow cur[i]$
7      $cur[i].p \leftarrow par$
8  insert $par$ into $H$
9  $l_p \leftarrow par.I^+ - par.I^-$                      //$par$ interval length
10 $I^-_{ppar} \leftarrow D.minv + d \cdot l_p \cdot \left\lfloor \frac{par.I^- - D.minv}{d \cdot l_p} \right\rfloor$
11 $I^+_{ppar} \leftarrow \min(D.maxv, I^-_{ppar} + d \cdot l_p)$
                               //compute interval for $par$ parent, $I_{ppar}$
12 $l_{sp} \leftarrow (len \cdot d^{cur[1].h})$        //interval length for a $par$ sibling node
13 $I_{spar} \leftarrow$ computeSiblingInterv-R($I^-_{ppar}$, $I^+_{ppar}$, $l_{sp}$, $d$)
                               //compute intervals for all $par$ sibling nodes
14 remove $par.I$ from $I_{spar}$  //remove $par$ interval, $par$ already constructed
15 $S \leftarrow$ constrSiblingNodes-R($I_{spar}$, null, $D$, $cur[1].h + 1$)
16 insert $S$ into $H$
17 return $H$
```

Based on I_0, the algorithm constructs the sibling nodes that are first visualized to the user (*line 4*). Then, the algorithm constructs the rest initial nodes (*lines 6-9*). In the RES case, as I_0 we consider the interval that includes the leaf that contains the resource of interest along with its sibling leaves. Hence, all the initial nodes are constructed in line 4 and the algorithm terminates (*line 5*) until the next user's operation.

After the first call, in each ICO execution, the algorithm initially checks if the parent node of the currently visualized elements is already constructed, or if all the nodes that enclose data objects[9] have been constructed (*line 6*). Then, procedure constrRollUp-R (*line 7*) is used to construct the cur parent node, as well as the parent's siblings. In the case that cur are not leaf nodes or data objects (*line 8*), procedure constrDrillDown-R (*line 9*) is used to construct all cur children. Finally, the algorithm returns the updated HETree (*line 10*).

The constrRollUp-R procedure initially constructs the cur parent node par (*lines 1-7*). Next, it computes the interval I_{ppar} corresponding to par parent node interval (*lines 10-11*). Using I_{ppar}, it computes the intervals for each of par sibling nodes (*line 13*). Finally,

[9]Note that in the HETree-R version, we may have nodes that do not enclose any data objects.

```
Procedure 5: constrDrillDown-R($D$, $d$, $cur$, $H$)
  Input: $D$: set of triples; $d$: tree degree; $cur$: currently visualized
         elements; $H$: currently created HETree-R
  Output: $H$: updated HETree-R
  //Computed in ICO-R:  $len$: the length of the leaf's interval
1 $l_c = len \cdot d^{cur[1].h-1}$                    //length of the children's intervals
2 for $i \leftarrow 1$ to $|cur|$ do
3     if $cur[i].c[0] =$ null then continue //nodes previously constructed
4     $I_{ch} \leftarrow$ computeSiblingInterv-R($cur[i].I^-$, $cur[i].I^+$, $l_c$, $d$)
                                          //compute intervals for $cur[i]$ children
5     $S \leftarrow$ constrSiblingNodes-R($I_{ch}$, $cur[i]$, $cur[i].data$, $cur[1].h-1$)
      for $k \leftarrow 1$ to $|S|$ do
6         $cur[i].c[k] \leftarrow S[k]$
7     insert $S$ into $H$
8 return $H$
```

the computed sibling nodes' intervals I_{spar} are used for the nodes construction (*line 15*).

In the constrDrillDown-R procedure, for each node in cur, its children are constructed as follows (*line 2*). First, the procedure computes the intervals I_{ch} of each child and then it constructs all children (*line 5*). Finally, the child relations for the parent node $cur[i]$ are constructed (*line 6-7*).

Computational Analysis. Here we analyse the ICO algorithms for both HETree versions.

$Number\ of\ Constructed\ Nodes$. Regarding the initial nodes constructed in each scenario: in RES scenario, at most d leaf nodes are constructed; in RAN scenario, at most $2d+d^2$ nodes are constructed; finally in BSC scenario, $d+1$ are constructed.

Regarding the maximum number of nodes constructed in each operation in RES and RAN scenarios: (1) A *roll-up operation* constructs at most $d + d \cdot (d-1) = d^2$ nodes. The d nodes are constructed in constrRollUp, whereas the $d \cdot (d-1)$ in constrDrillDown. (2) A *drill-down operation* constructs at most d^2 nodes in constrDrillDown. As for the BSC scenario: (1) A *roll-up operation* does not construct any nodes. (2) A *drill-down operation* constructs at most d^2 nodes in constrDrillDown.

$Discussion$. The worst case for the computational cost is higher in HETree-R than in HETree-C, for all exploration scenarios. Particularly, in HETree-R worst case, ICO must build leaves that contain the whole dataset and the computational cost is $O(|D|log|D|)$ for all scenarios. In HETree-C, for the RES and RAN scenarios, the cost is $O(d^2 + \frac{d-1}{d}|D|)$, and for the BSC scenario the cost is $O(d^2 + |D|)$. A detailed computational analysis for both HETree-R and HETree-C is included in Appendix B.

3.3. Adaptive HETree Construction

In a visual exploration scenario, users wish to modify the visual organization of the data by providing user-specific preferences for the whole hierarchy or part of it. The user can select a specific subtree and alter the number of groups visualized in each level (i.e., the tree degree) or the size of the groups (i.e., number of leaves). In this case, a new tree (or a part of it) pertaining to the new parameters provided by the user should be constructed on-the-fly.

For example, consider the HETree-C of Figure 6 representing ages of persons[10]. A user may navigate to node b, where she prefers to increase the number of groups visualized in each level. Thus, she modifies the degree of b from 2 to 4 and the subtree is adapted to the new parameter as depicted on the bottom tree of Figure 6. On the other hand, the user prefers exploring the right subtree (starting from node c) with less details. She chooses to increase the size of the groups by reducing (from 4 to 2) the number of leaves for the subtree of c. In both cases, constructing the subtree from scratch based on the user-provided parameters and recomputing statistics entails a significant time overhead, especially, when user preferences are applied to a large part of or the whole hierarchy.

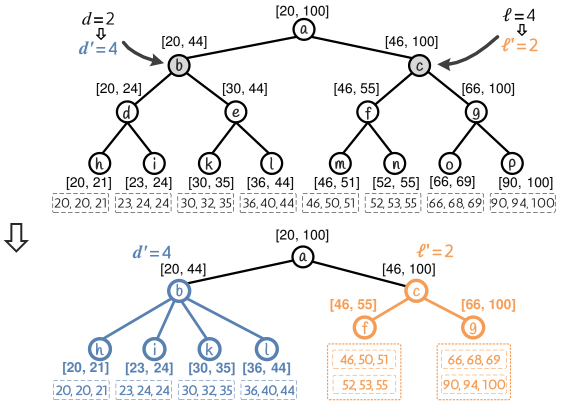


Fig. 6. Adaptive HETree example

In this section, we introduce ADA (**Ada**ptive HETree Construction) method, which dynamically adapts an existing HETree to a new set of user-defined parameters. Instead of both constructing the tree and computing the nodes' statistics from scratch, our method re-

[10]For simplicity, Figure 6 presents only the objects (i.e., numeric values) of the triples.

Table 2. Summary of Adaptive HETree Construction*

		Modify Degree				**Modify Num. of Leaves**			
	Full Construction	$d' = d^k$	$d' = k \cdot d$	$d' = \sqrt[k]{d}$	*elsewhere*	$\ell' > \ell$	$\ell' = \frac{\ell}{d^k}$	$\ell' = \frac{\ell}{k}$	$\ell' = \ell - k$
Tree Construction									
Complexity	$O(mlogm + d'e)$	$O(mlog_{\sqrt[k]{d'}} m)$	$O(d'e)$	$O(d'^k r)$	$O(d'e)$	$O(m + d'e)$	$O(m)$	$O(m + d'e)$	$O(mlogm + d'e)$
#leaves$_0$	ℓ'	0	0	0	0	ℓ'	0	0	ℓ'
#leaves$_+$	0	0	0	0	0	0	ℓ'	ℓ'	0
#internals$_0$	e	0	e	$e - r$	e	e	0	e	e
#internals$_+$	0	0	0	0	0	0	0	0	0
Statistics Computations									
Complexity	$O(m + d'e)$	$O(1)$	$O(\frac{k\ell'}{d'} + d'e)$	$O(d'(e - r))$	$O(d'e)$	$O(m + d'e)$	$O(1)$	$O(m + d'e)$	$O(m + d'e - \ell' - k)$
#leaves$_0$	ℓ'	0	0	0	0	ℓ'	0	0	$\ell' - \frac{\ell'^2}{d'}$
#leaves$_+$	0	0	0	0	0	0	0	ℓ'	$\frac{\ell'^2}{d'}$
#internals$_0$	e	0	$e - \lceil \frac{\ell'}{d'} \rceil$	$e - r$	e	e	0	e	e
#internals$_+$	0	0	$\lceil \frac{\ell'}{d'} \rceil$	0	0	0	0	0	0

* $m = |D|$, $e = \frac{d'\ell' - 1}{d' - 1}$ (maximum number of internal nodes), and $r = \frac{d'^k \ell' - 1}{d'^k - 1}$

constructs the new part(s) of the hierarchy by exploiting the existing elements (i.e., nodes, statistics) of the tree. In this way, ADA achieves to reduce the overall construction cost and enables the on-the-fly reorganization of the visualized data. In the example of Figure 6, the new subtree of b can be derived from the old one, just by removing the internal nodes d and e, while the new subtree of c results from merging leaves together and aggregating their statistics.

Let $\mathcal{T}(D, \ell, d)$ denote the existing HETree and $\mathcal{T}'(D, \ell', d')$ is the new HETree corresponding to the new user preferences for the tree degree d' and the number of leaves ℓ'. Note that $\mathcal{T}$ could also denote a subtree of an existing HETree (in the scenario where the user modifies only a part of it). In this case, the user indicates the *reconstruction root* of $\mathcal{T}$.

Then, ADA identifies the following elements of $\mathcal{T}$: (1) The elements of $\mathcal{T}$ that also exist in $\mathcal{T}'$. For example, consider the following two cases: the leaf nodes of $\mathcal{T}'$ are internal nodes of $\mathcal{T}$ in level x; the statistics of $\mathcal{T}'$ nodes in level x are equal to the statistics of $\mathcal{T}$ nodes in level y. (2) The elements of $\mathcal{T}$ that can be reused (as "building blocks") for constructing elements in $\mathcal{T}'$. For example, consider the following two cases: each leaf node of $\mathcal{T}'$ is constructed by merging x leaf nodes of $\mathcal{T}$; the statistics for the node n of $\mathcal{T}'$ can be computed by aggregating the statistics from the nodes q and w of $\mathcal{T}$.

Consequently, we consider that an element (i.e., node or node's statistics) in $\mathcal{T}'$ can be: (1) constructed/computed from scratch[11], (2) reused as is from $\mathcal{T}$ or (3) derived by aggregating elements from $\mathcal{T}$.

Table 2 summarizes the ADA reconstruction process. Particularly, the table includes: (1) the computational complexity for constructing $\mathcal{T}'$, denoted as $Complexity$; (2) the number of leaves and internal nodes of $\mathcal{T}'$ constructed from scratch, denoted as $\#leaves_0$ and $\#internals_0$, respectively; and (3) the number of leaves and internal nodes of $\mathcal{T}'$ derived from nodes of $\mathcal{T}$, denoted as $\#leaves_+$ and $\#internals_+$, respectively. The lower part of the table presents the results for the computation of node statistics in $\mathcal{T}'$. Finally, the second table column, denoted as $Full\ Construction$, presents the results of constructing $\mathcal{T}'$ from scratch.

The following example demonstrates the ADA results, considering a DBpedia exploration scenario.

Example 7. The user visually explores the *populationTotal* property of the DBpedia dataset. The default system organization for this property is a hierarchy with degree 3. The user modifies the tree parameters in order to fit better visualization results as following. First, she decides to visualize more groups in each hierarchy level and increases the degree from 3 to 9 (1st *Modification*). Then, she observes that the results overflow the visualization area and that a smaller degree fits better; thus she re-adjusts the tree degree to a value of 6 (2nd *Modification*). Finally, she navigates through the data values and decides to increase the groups' size by a factor of three (i.e., dividing by three the number of leaves) (3rd *Modification*). Again, she corrects her decision and readjusts the final group size to twice the default size (4th *Modification*).

[11]Note that it is possible for a from scratch constructed node in $\mathcal{T}'$ to aggregate statistics from nodes in $\mathcal{T}$.

Table 3. Full Construction vs. ADA over DBpedia Exploration Scenario (cells values: Full / ADA)

	Modify Degree		Modify Num. of Leaves	
	1st Modification	2nd Modification	3rd Modification	4th Modification
Tree Construction				
#nodes	22.1K / 0	23.6K / 3.9K	9.8K / 6.6K	14.7K / 4.9K
Statistics Computations				
#nodes	22.1K / 0	23.6K / 659	9.8K / 0	14.7K / 4.9K

Table 3 summarizes the number of nodes, constructed by a *Full Construction* and ADA in each modification, along with the required statistics computations. Considering the whole set of modifications, ADA constructs only the 22% (15.4K vs. 70.2K) of the nodes that are created in the case of the full construction. Also, ADA computes the statistics for only 8% (5.6K vs. 70.2K) of the nodes. □

In the next sections, we present in detail the reconstruction process through the example trees of Figure 7. Figure 7a presents the initial tree $\mathcal{T}$ that is an HETree-C, with $\ell = 8$ and $d = 2$. Figures 7b~7e present several reconstructed trees $\mathcal{T}'$. Blue colour is used to indicate the elements (i.e., nodes, edges, statistics) of $\mathcal{T}'$ which do not exist in $\mathcal{T}$. Regarding statistics, we assume that in each node we compute the mean value. In each $\mathcal{T}'$, we present only the mean values that are not known from $\mathcal{T}$. Also, in mean values computations with red colour, we highlight the values that are reused from $\mathcal{T}$. All reconstruction details and computational analysis for each case are included in Appendix C.

3.3.1. The User Modifies the Tree Degree

Regarding the modification of the degree parameter, we distinguish the following cases:

The user increases the tree degree. We have that $d' > d$; based on the d' value we have the following cases:

(1) $d' = d^k$, with $k \in \mathbb{N}^+$ and $k > 1$: Figure 7a presents $\mathcal{T}$ with $d = 2$ and Figure 7d presents the reconstructed $\mathcal{T}'$ with $d' = 4$ (i.e., $k = 2$). $\mathcal{T}'$ results by simply removing the nodes with height 1 (i.e., d, e, f, g) and connecting the nodes with height 2 (i.e., b, c) with the leaves.

In general, $\mathcal{T}'$ results from $\mathcal{T}$ by simply removing tree levels from $\mathcal{T}$. Additionally, there is no need for computing any new statistics, since the statistics for all nodes of $\mathcal{T}'$ remain the same as in $\mathcal{T}$.

(2) $d' = k \cdot d$, with $k \in \mathbb{N}^+$, $k > 1$ and $k \neq d^\nu$ where $\nu \in \mathbb{N}^+$: An example with $k = 3$ is presented in Figure 7c, where we have $d' = 6$. In this case, the leaves of $\mathcal{T}$ (Figure 7a) remain leaves in $\mathcal{T}'$ and all internal nodes up to the reconstruction root of $\mathcal{T}$ are constructed from scratch. As for the node statistics, we can compute the mean values for $\mathcal{T}'$ nodes with height 1 (i.e., μ_b, μ_c) by aggregating already computed mean values (e.g., μ_d, μ_e, etc.) from $\mathcal{T}$.

In general, except for the leaves, we construct all internal nodes from scratch. For the internal nodes of height 1, we compute their statistics by aggregating the statistics of $\mathcal{T}$ leaves, whereas for internal nodes of height greater than 1, we compute from scratch their statistics.

(3) $elsewhere$: In any other case where the user increases the tree degree, all internal nodes in $\mathcal{T}'$ except for the leaves are constructed from scratch. In contrast with the previous case, the leaves' statistics from $\mathcal{T}$ can not be reused and, thus, for all internal nodes in $\mathcal{T}'$ the statistics are recomputed.

The user decreases the tree degree. Here we have that $d' < d$; based on the d' value we have the following two cases:

(1) $d' = \sqrt[k]{d}$, with $k \in \mathbb{N}^+$ and $k > 1$: Assume that now Figure 7d depicts $\mathcal{T}$, with $d = 4$, while Figure 7a presents $\mathcal{T}'$ with $d' = 2$. We can observe that $\mathcal{T}'$ contains all nodes of $\mathcal{T}$, as well as a set of extra internal nodes (i.e., d, e, f, g). Hence, $\mathcal{T}'$ results from $\mathcal{T}$ by constructing some new internal nodes.

(2) $elsewhere$: This case is similar to the previous case (3) where the user increases the tree degree.

3.3.2. The User Modifies the Number of Leaves

Regarding the modification of the number of leaves parameter, we distinguish the following cases:

The user increases the number of leaves. In this case we have that $\ell' > \ell$; hence, each leaf of $\mathcal{T}$ is split into several leaves in $\mathcal{T}'$ and the data objects (i.e., LD resources) contained in a $\mathcal{T}$ leaf must be reallocated to the new leaves in $\mathcal{T}'$. As a result, all nodes (both leaves and internal nodes) in $\mathcal{T}'$ have different contents compared to nodes in $\mathcal{T}$ and must be constructed from scratch along with their statistics.

In this case, constructing $\mathcal{T}'$ requires $O(|D| + \frac{d^2 \cdot \ell' - d}{d-1})$ (by avoiding the sorting phase).

The user decreases the number of leaves. In this case we have that $\ell' < \ell$; based on the ℓ' value we have the following three cases:

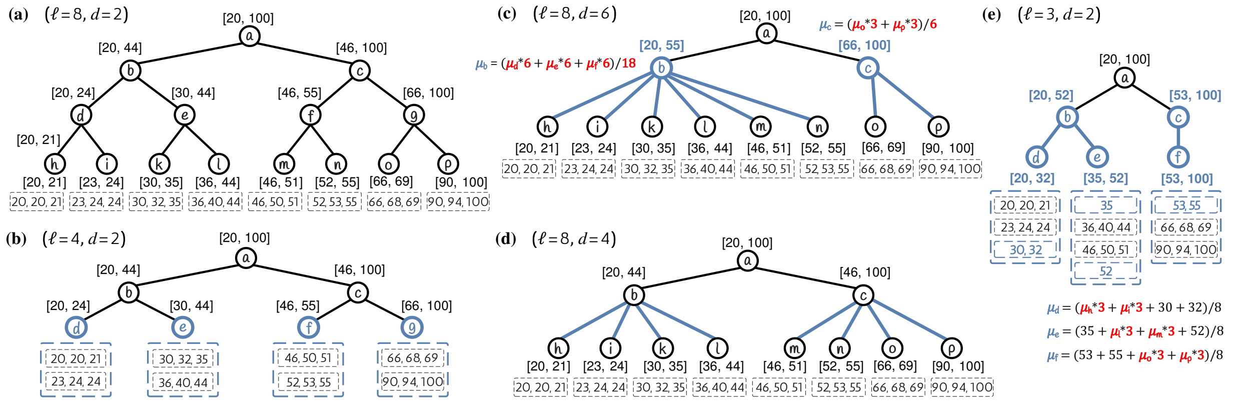

(a) (ℓ=8, d=2)
(b) (ℓ=4, d=2)
(c) (ℓ=8, d=6)
(d) (ℓ=8, d=4)
(e) (ℓ=3, d=2)
μc = (μo*3 + μp*3)/6
μb = (μd*6 + μe*6 + μf*6)/18
μd = (μh*3 + μi*3 + 30 + 32)/8
μe = (35 + μl*3 + μm*3 + 52)/8
μf = (53 + 55 + μo*3 + μp*3)/8

Fig. 7. Adaptive HETree construction examples

(1) $\ell' = \frac{\ell}{d^k}$, with $k \in \mathbb{N}^+$: Considering that Figure 7a presents $\mathcal{T}$ with $\ell = 8$ and $d = 2$. A reconstruction example of this case with $k = 1$, is presented in Figure 7b, where we have $\mathcal{T}'$ with $\ell' = 4$. In Figure 7b, we observe that the leaves in $\mathcal{T}'$ result from merging d^k leaves of $\mathcal{T}$. For example, the leaf d of $\mathcal{T}'$ results from merging the leaves h and i of $\mathcal{T}$. Then, $\mathcal{T}'$ results from $\mathcal{T}$, by replacing the $\mathcal{T}$ nodes with height k (i.e., b, e, f, g), with the $\mathcal{T}'$ leaves. Finally, the nodes of $\mathcal{T}$ with height less than k are not included in $\mathcal{T}'$.

Therefore, in this case, $\mathcal{T}'$ is constructed by merging the leaves of $\mathcal{T}'$ and removing the internal nodes of $\mathcal{T}'$ having height less or equal to k. Also, we do not recompute the statistics of the new leaves of $\mathcal{T}'$ as these are derived from the statistics of the removed nodes with height k.

(2) $\ell' = \frac{\ell}{k}$, with $k \in \mathbb{N}^+$, $k > 1$ and $k \neq d^{\nu}$, where $\nu \in \mathbb{N}^+$: As in the previous case, the leaves in $\mathcal{T}'$ are constructed by merging leaves from $\mathcal{T}$ and their statistics are computed based on the statistics of the merged leaves. In this case, however, all internal nodes in $\mathcal{T}'$ have to be constructed from scratch.

(3) $\ell' = \ell - k$, with $k \in \mathbb{N}^+$, $k > 1$ and $\ell' \neq \frac{\ell}{\nu}$, where $\nu \in \mathbb{N}^+$: The two previous cases describe that each leaf in $\mathcal{T}'$ $fully$ contains k leaves from $\mathcal{T}$. In this case, a leaf in $\mathcal{T}'$ may *partially* contains leaves from $\mathcal{T}$. A leaf in $\mathcal{T}'$ fully contains a leaf from $\mathcal{T}$ when the $\mathcal{T}'$ leaf contains all data objects belonging to the $\mathcal{T}$ leaf. Otherwise, a leaf in $\mathcal{T}'$ partially contain a leaf from $\mathcal{T}$ when the $\mathcal{T}'$ leaf contains a subset of the data objects from the $\mathcal{T}$ leaf.

An example of this case is shown in Figure 7e that depicts a reconstructed $\mathcal{T}'$ resulted from the $\mathcal{T}$ presented in Figure 7a. The d leaf of $\mathcal{T}'$ fully contains leaves h, i of $\mathcal{T}$ and partially leaf k for which value 35 belongs to a different leaf (i.e., e).

Due to this partial containment, we have to construct all leaves and internal nodes from scratch and recalculate their statistics. Still, the statistics of the fully contained leaves of $\mathcal{T}$ can be reused, by aggregating them with the individual values of the data objects included in the leaves. For example, as we can see in Figure 7e, the mean value μ_d of the leaf d is computed by aggregating the mean values μ_h and μ_i corresponding to the fully contained leaves h and i, with the individual values 30, 32 of the partially contained leaf k.

4. The rdf:SynopsViz Tool

Based on the proposed hierarchical model, we have developed a Web-based prototype called *rdf:SynopsViz*[12]. The key features of rdf:SynopsViz are summarized as follows: (1) It supports the aforementioned *hierarchical model* for RDF data visualization, browsing and analysis. (2) It offers *automatic* on-the-fly hierarchy construction, as well as *user-defined* hierarchy construction based on users' preferences. (3) Provides *faceted* browsing and filtering over classes and properties. (4) Integrates *statistics with visualization*; visualizations have been enriched with useful statistics and data information. (5) Offers several visualization techniques (e.g., timeline, chart, treemap). (6) Provides a large number of dataset's *statistics* regarding the: *data-level* (e.g., number of sameAs triples), *schema-level* (e.g., most common classes/properties), and *structure level* (e.g., entities with the larger in-degree). (7) Provides numerous *metadata* related to the dataset: licens-

[12]synopsviz.imis.athena-innovation.gr

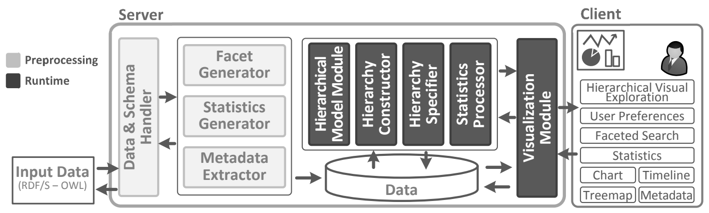


Fig. 8. System architecture

ing, provenance, linking, availability, undesirability, etc. The latter can be considered useful for assessing data quality [70].

In the rest of this section, Section 4.1 describes the system architecture, Section 4.2 demonstrates the basic functionality of the rdf:SynopsViz. Finally, Section 4.3 provides technical information about the implementation.

4.1. System Architecture

The architecture of rdf:SynopsViz is presented in Figure 8. Our scenario involves three main parts: the Client UI, the rdf:SynopsViz, and the Input data. The *Client* part, corresponds to the system's front-end offering several functionalities to the end-users. For example, hierarchical visual exploration, facet search, etc. (see Section 4.2 for more details). rdf:SynopsViz consumes RDF data as *Input data*; optionally, OWL-RDF/S vocabularies/ontologies describing the input data can be loaded. Next, we describe the basic components of the rdf:SynopsViz.

In the preprocessing phase, the *Data and Schema Handler* parses the input data and inferes schema information (e.g., properties domain(s)/range(s), class/ property hierarchy, type of instances, type of properties, etc.). *Facet Generator* generates class and property facets over input data. *Statistics Generator* computes several statistics regarding the schema, instances and graph structure of the input dataset. *Metadata Extractor* collects dataset metadata. Note that the model construction does not require any preprocessing, it is performed online, according to user interaction.

During runtime the following components are involved. *Hierarchy Specifier* is responsible for managing the configuration parameters of our hierarchy model, e.g., the number of hierarchy levels, the number of nodes per level, and providing this information to the Hierarchy Constructor. *Hierarchy Constructor* implements our tree structure. Based on the selected facets, and the hierarchy configuration, it determines the hierarchy of groups and the contained triples. *Statistics Processor* computes statistics about the groups included in the hierarchy. *Visualization Module* allows the interaction between the user and the back-end, allowing several operations (e.g., navigation, filtering, hierarchy specification) over the visualized data. Finally, the *Hierarchical Model Module* maintains the in-memory tree structure for our model and communicates with the Hierarchy Constructor for the model construction, the Hierarchy Specifier for the model customization, the Statistics Processor for the statistics computations, and the Visualization Module for the visual representation of the model.

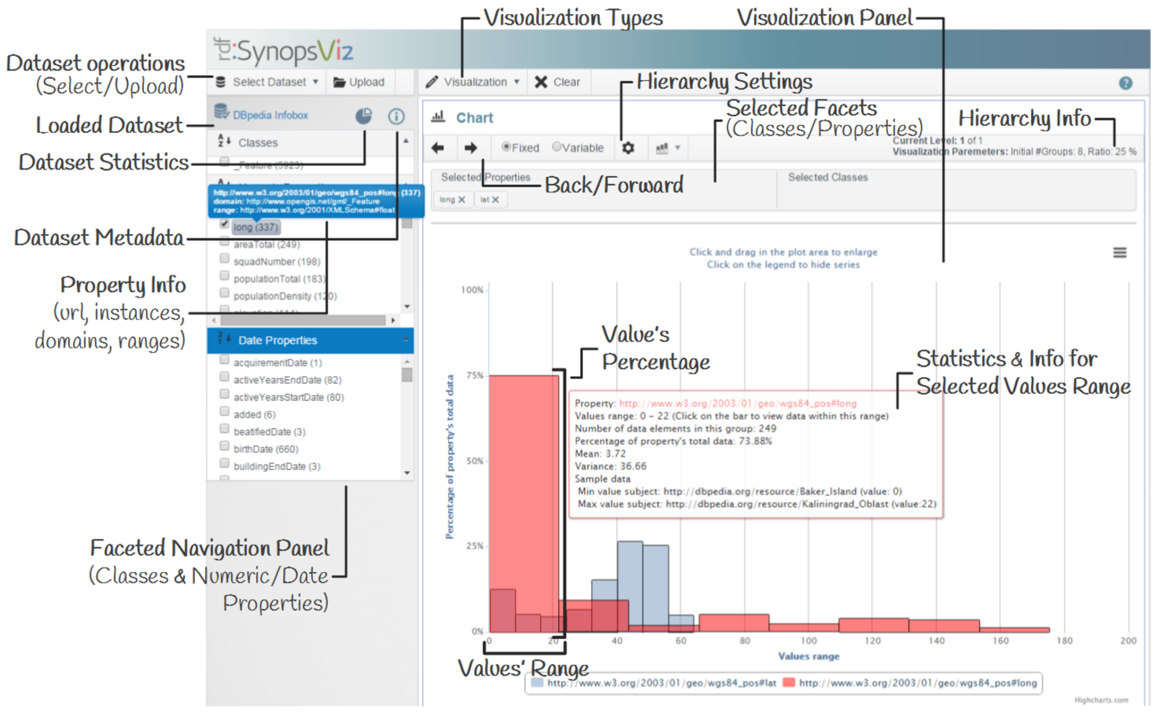
4.2. rdf:SynopsViz In-Use

In this section we outline the basic functionality of rdf:SynopsViz prototype. Figure 9 presents the Web user interface of the main window. rdf:SynopsViz UI consists of the following main panels: *Facets panel*: presents and manages facets on classes and properties; *Input data control panel*: enables the user to import and manage input datasets; *Visualization panel*: is the main area where interactive charts and statistics are presented; *Configuration panel*: handles visualization settings.

Initially, users are able to select a dataset from a number of offered real-word LD datasets (e.g., DBpedia, Eurostat) or upload their own. Then, for the selected dataset, the users are able to examine several of the *dataset's metadata*, and explore several *datasets's statistics*.

Using the *facets panel*, users are able to navigate and *filter* data based on classes, numeric and date properties. In addition, through facets panel several information about the classes and properties (e.g., number of instances, domain(s), range(s), IRI, etc.) are provided to the users through the UI.

Users are able to visually explore data by considering properties' values. Particularly, *area charts* and *timeline-based area charts* are used to visualize the resources considering the user's selected properties. Classes' facets can also be used to *filter* the visualized data. Initially, the top level of the hierarchy is presented providing an *overview* of the data, organized into top-level groups; the user can interactively *drill-down* (i.e., zoom-in) and *roll-up* (i.e., zoom-out) over


Visualization Types
Visualization Panel
Dataset operations (Select/Upload)
Hierarchy Settings
Selected Facets (Classes/Properties)
Hierarchy Info
Loaded Dataset
Dataset Statistics
Back/Forward
Dataset Metadata
Property Info (url, instances, domains, ranges)
Value's Percentage
Statistics & Info for Selected Values Range
Faceted Navigation Panel (Classes & Numeric/Date Properties)
Values' Range
SynopsViz
Select Dataset
Upload
Visualization
Clear
DBpedia Infobox
Classes
Chart
Fixed
Variable
Selected Properties
Selected Classes
Date Properties
Click and drag in the plot area to enlarge
Click on the legend to hide series
Percentage of property's total data
Values range

Property: http://www.w3.org/2003/01/geo/wgs84_pos#long
Values range: 0 - 22 (Click on the bar to view data within this range)
Number of data elements in this group: 249
Percentage of property's total data: 73.88%
Mean: 3.72
Variance: 36.66
Sample data
Min value subject: http://dbpedia.org/resource/Baker_Island (value: 0)
Max value subject: http://dbpedia.org/resource/Kaliningrad_Oblast (value:22)

Fig. 9. Web user interface

the group of interest, up to the actual values of the input data (i.e., LD resources). At the same time, statistical information concerning the hierarchy groups as well as their contents (e.g., mean value, variance, sample data, range) is presented through the UI (Figure 10a). Regarding the most detailed level (i.e., LD resources), several visualization types are offered; i.e., area, column, line, spline and areaspline (Figure 10b).

In addition, users are able to visually explore data, through class hierarchy. Selecting one or more classes, users can interactively navigate over the class hierarchy using treemaps (Figure 10c) or pie charts (Figure 10d). Properties' facets can also be used to filter the visualized data. In rdf:SynopsViz the treemap visualization has been enriched with schema and statistical information. For each class, schema metadata (e.g., number of instances, subclasses, datatype/object properties) and statistical information (e.g., the cardinality of each property, min, max value for datatype properties) are provided.

Finally, users can interactively modify the hierarchy specifications. Particularly, they are able to increase or decrease the level of abstraction/detail presented, by modifying both the number of hierarchy levels, and number of nodes per level.

A video presenting the basic functionality of our prototype is available at `youtu.be/n2ctdH5PKA0`. Finally, a demonstration of rdf:SynopsViz tool is presented in [16].

4.3. Implementation

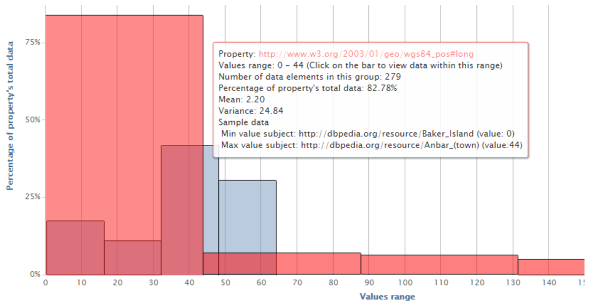
rdf:SynopsViz is implemented on top of several open source tools and libraries. The back-end of our system is developed in Java, Jena framework is used for RDF data handing and Jena TDB is used for disk-based RDF storing. The front-end prototype, is developed using HTML and Javascript. Regarding visualization libraries, we use Highcharts, for the area, column, line, spline, areaspline and timeline-based charts and Google Charts for treemap and pie charts.

5. Experimental Analysis

In this section we present the evaluation of our approach. In Section 5.1, we present the dataset and the experimental setting. Then, in Section 5.2 we present the performance results and in Section 5.3 the user evaluation we performed.

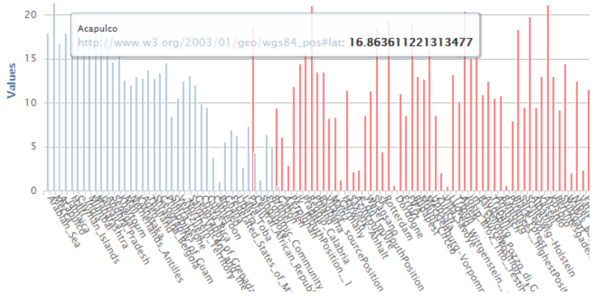
5.1. Experimental Setting

In our evaluation, we use the well known *DBpedia* 2014 LD dataset. Particularly, we use the *Mapping-*



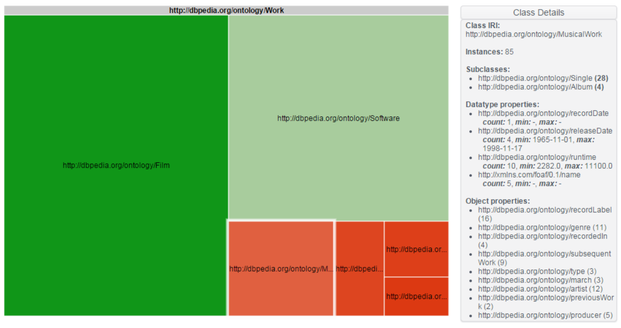
Property: http://www.w3.org/2003/01/geo/wgs84_pos#long
Values range: 0 – 44 (Click on the bar to view data within this range)
Number of data elements in this group: 279
Percentage of property's total data: 82.78%
Mean: 2.20
Variance: 24.84
Sample data
Min value subject: http://dbpedia.org/resource/Baker_Island (value: 0)
Max value subject: http://dbpedia.org/resource/Anbar_(town) (value:44)

(a) Groups of numeric RDF data (Area chart)



Acapulco
http://www.w3.org/2003/01/geo/wgs84_pos#lat: 16.86361221313477

(b) Numeric RDF data (Column chart)


http://dbpedia.org/ontology/Work
http://dbpedia.org/ontology/Film
http://dbpedia.org/ontology/Software
http://dbpedia.org/ontology/M...
http://dbpedi...
http://dbpedia.or...
http://dbpedia.or...

Class Details

Class IRI:
http://dbpedia.org/ontology/MusicalWork

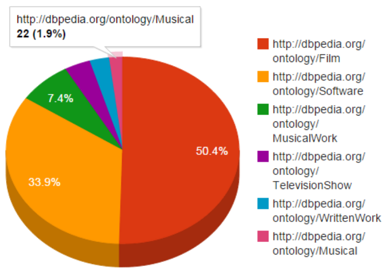
Instances: 85

Subclasses:
- http://dbpedia.org/ontology/Single **(28)**
- http://dbpedia.org/ontology/Album **(4)**

Datatype properties:
- http://dbpedia.org/ontology/recordDate
  count: 1, ***min:*** -, ***max:*** -
- http://dbpedia.org/ontology/releaseDate
  count: 4, ***min:*** 1965-11-01, ***max:*** 1998-11-17
- http://dbpedia.org/ontology/runtime
  count: 10, ***min:*** 2282.0, ***max:*** 11100.0
- http://xmlns.com/foaf/0.1/name
  count: 5, ***min:*** -, ***max:*** -

Object properties:
- http://dbpedia.org/ontology/recordLabel (16)
- http://dbpedia.org/ontology/genre (11)
- http://dbpedia.org/ontology/recordedIn (4)
- http://dbpedia.org/ontology/subsequentWork (9)
- http://dbpedia.org/ontology/type (3)
- http://dbpedia.org/ontology/march (3)
- http://dbpedia.org/ontology/artist (12)
- http://dbpedia.org/ontology/previousWork (2)
- http://dbpedia.org/ontology/producer (5)

(c) Class hierarchy (Treemap chart)



(d) Class hierarchy (Pie chart)

Fig. 10. Numeric data & class hierarchy visualization examples

based Properties (cleaned) dataset[13] which contains high-quality data, extracted from Wikipedia Infoboxes. This dataset contains 33.1M triples and includes a large number of numeric and temporal properties of varying sizes. The largest numeric property in this dataset has 534K triples, whereas the largest temporal property has 762K.

Regarding the methods used in our evaluation, we consider our HETree hierarchical approaches, as well as a simple non-hierarchical visualization approach, referred as *FLAT*. FLAT is considered as a competitive method against our hierarchical approaches. It provides one level visualizations, rendering only the actual data objects; i.e., it is the same as the visualization provided by rdf:SynopsViz at the most detailed level. In more detail, FLAT approach corresponds to a column chart in which the resources are sorted in ascending order based on their object values, the horizontal axis contains the resources' names (i.e., triples' subjects), and the vertical axis corresponds to objects' values. By hovering over a resource, a tooltip appears including the resource's name and object value.

Regarding the HETree approaches, the tree parameters (i.e., number of leaves, degree and height) are automatically computed following the approach described in Section 2.5. In our experiments, the lower and the upper bound for the objects rendered at the most detailed level have been set to $\lambda_{min} = 10$ and $\lambda_{max} = 50$, respectively. Considering the visualizations provided by the default Highcharts settings, these numbers are reasonable for our screen size and resolution.

Finally, our backend system is hosted on a server with a quad-core CPU at 2GHz and 8GB of RAM running Windows Server 2008. As client, we used a laptop with i5 CPU at 2.5GHz with 4G RAM, running Windows 7, Firefox 38.0.1 and ADSL2+ internet connection. Additionally, in the user evaluation, the client is employed with a 24" (1920×1200) screen .

5.2. Performance Evaluation

In this section, we study the performance of the proposed model, as well as the behaviour of our tool, in terms of construction and response time, respectively. Section 5.2.1 describes the setting of our performance evaluation, and Section 5.2.2 presents the evaluation results.

5.2.1. Setup

In order to study the performance, a number of numeric and temporal properties from the employed

[13]downloads.dbpedia.org/2014/en/mappingbased_properties_cleaned_en.nt.bz2

dataset are visualized using the two hierarchical (i.e., HETree-C/R) and the FLAT approach. We select one set from each type of properties; each set contains 15 properties with varying sizes, starting from small properties having 50-100 triples up to the largest properties.

In our experiment, for each of the three approaches, we measure the tool response time. Additionally, for the two hierarchical approaches we also measure the time required for the HETree construction.

Note that in hierarchical approaches through user interaction, the server sends to the browser only the data required for rendering the current visualization level (although the whole tree is constructed at the back-end). Hence, when a user requests to generate a visualization we have the following workflow. Initially, our system constructs the tree; then, the data regarding the top-level groups (i.e., root node children) are sent to the browser which renders the result. Afterwards, based on user interactions (i.e., drill-down, roll-up), the server retrieves the required data from the tree and sends it to the browser. Thus, the tree is constructed the first time a visualization is requested for the given input dataset; for any further user navigation over the hierarchy, the response time does not include the construction time. Therefore, in our experiments, in hierarchical approaches as response time we measure the time required by our tool to provide the first response (i.e., render the top-level groups), which corresponds to the slower response in our visual exploration scenario. Thus, we consider the following measures in our experiments:

$Construction\ Time$: the time required to build the HETree structure. This time includes (1) the time for sorting the triples; (2) the time for building the tree; and (3) the time for the statistics computations.

$Response\ Time$: the time required to render the charts, starting from the time the client sends the request. This time includes (1) the time required by the server to compute and build the response. In hierarchical approaches, this time corresponds to the *Construction Time*, plus the time required by the server to build the JSON object sent to the client. In FLAT approach, it corresponds to the time spent in sorting the triples plus the time for the JSON construction; (2) the time spent in the client-sever communication; and (3) the time required by the visualization library to render the charts on the browser.

5.2.2. Results

Table 4 presents the evaluation results regarding the numeric (upper half) and the temporal properties (lower half). The properties are sorted in ascending order of the number of triples. For each property, the table contains the number of triples, the characteristics of the constructed HETree structures (i.e., number of leaves, degree, height, and number of nodes), as well as the construction and the response time for each approach. The presented time measurements are the average values from 50 executions.

Regarding the comparison between the HETree and the FLAT, the FLAT approach can not provide results for properties having more than 305K triples, indicated in the last rows for both numeric and temporal properties with "–" in the FLAT response time. For the rest properties, we can observe that the HETree approaches clearly outperform the FLAT in all cases, even in the smallest property (i.e., *rankingWin*, 50 triples). As the size of properties increases, the difference between the HETree approaches and the FLAT increases, too. In more detail, for properties having more than 1.400 triples (i.e., the numeric properties larger than the *hsvCoordinateValue* -5th row-, and the temporal properties larger than the *lastAirDate* -4th row-), the HETree approaches outperform the FLAT by about one order of magnitude. Finally, for the largest property that FLAT can handle (i.e., *populationTotal*, 305K triples), the difference between the HETree and the FLAT is about two order of magnitude.

Regarding the time required for the construction of the HETree structure, from Table 4 we can observe the following. The performance of both HETtree structures is very close for all examined properties, with the HETree-R performing slightly better than the HETree-C. Furthermore, we can observe that the response time follows a similar trend as the construction time. This is expected since the communication cost, as well as the times required for constructing and rendering the JSON object are almost the same for all cases. Particularly, in our HETree setting, the Highchart requires approximately 90 msec for rendering the charts in the browser.

Regarding the comparison between the construction and the response time in the HETree approaches, from Table 4 we can observe the following. For properties having up to 5.5K triples (i.e., the numeric properties smaller than the *width* -8th row-, and the temporal properties smaller than the *decommissioningDate* -7th row-), the response time is dominated by the communication cost, and the time required for the JSON construction and rendering. For properties with only a small number of triples (i.e., *waistSize*, 241 triples), only the 1.5% of the response time is spent on con-

Table 4. Performance Results for Numeric & Temporal Properties

	Tree Characteristics				**HETree-C**		**HETree-R**		**FLAT**
Property (#Triples)	**#Leaves**	**Degree**	**Height**	**#Nodes**	**Construction Time** (msec)	**Response Time** (msec)	**Construction Time** (msec)	**Response Time** (msec)	**Response Time** (msec)
Numeric Properties									
rankingWins (50)	9	3	2	13	5	324	1	323	415
distanceToBelfast (104)	9	3	2	13	7	337	4	329	419
waistSize (241)	16	4	2	21	10	346	9	336	440
fileSize (492)	27	3	3	40	18	347	16	345	575
hsvCoordinateValue (995)	81	3	4	121	74	403	50	383	980
lineLength (1, 923)	81	3	4	121	77	409	55	391	1,463
powerOutput (5, 453)	243	3	5	364	234	560	217	540	2,583
width (11, 049)	729	3	6	1,093	506	830	467	799	6,135
numberOfPages (21, 743)	729	3	6	1,093	2,888	3,219	2,403	2,722	12,669
inseeCode (36, 780)	2,187	3	7	3,280	4,632	4,962	4,105	4,436	19,119
areaWater (40, 564)	2,187	3	7	3,280	4,945	5,134	5,274	5,457	29,538
populationDensity (52, 572)	2,187	3	7	3,280	6,803	7,127	6,080	6,404	44,262
areaTotal (140, 408)	6,561	3	8	9,841	16,158	16,482	13,298	13,627	219,018
populationTotal (304, 522)	19,683	3	9	29,524	31,141	31,473	25,866	26,196	1,523,675
lat (533, 900)	19,683	3	9	29,524	73,528	73,862	71,784	72,106	—
Temporal Properties									
retired (155)	9	3	2	13	8	330	4	327	425
endDate (341)	27	3	3	40	17	339	16	339	468
lastAirDate (704)	64	4	3	85	34	359	30	359	853
buildingStartDate (1, 415)	81	3	4	121	73	406	53	384	1,103
latestReleaseDate (2, 925)	243	3	5	364	162	496	146	480	1,804
orderDate (3, 788)	243	3	5	364	210	542	195	523	2,011
decommissioningDate (7, 082)	243	3	5	364	405	735	383	717	3,423
shipLaunch (15, 938)	729	3	6	1,093	1,772	2,094	1,595	1,919	6,935
completionDate (17, 017)	729	3	6	1,093	1,987	2,311	1,793	2,121	7,814
foundingDate (19, 694)	729	3	6	1,093	2,745	3,069	2,583	2,905	8,699
added (44, 227)	2,187	3	7	3,280	5,912	5,943	6,244	6,265	33846
activeYearsStartDate (98, 160)	6,561	3	8	9,841	10,368	10,702	8,952	9,282	107,587
releaseDate (169, 156)	6,561	3	8	9,841	19,122	19,451	16,526	16,856	950,545
deathDate (321, 883)	19,683	3	9	29,524	32,990	33,313	27,936	28,271	—
birthDate (761, 830)	59,049	3	10	88,573	85,797	86,120	83,982	84,314	—

structing the HETree. Moreover, for a property with a larger number of triples (i.e., *buildingStartData*, 1.415 triples), 18% of the time is spent on constructing the HETree. Finally, for the largest property for which the construction time is dominated by the other costs (i.e., *powerOutput*, 5.453 triples), 42% of the time is spent on constructing the HETree.

Figure 11 summarizes the results from Table 4, presenting the response time for all approaches w.r.t. the number of triples. Particularly, Figure 11a includes all properties sizes, whereas Figure 11b focus on the properties having up to 20K triples. We observe also here that the HETree-R performs slightly better than the HETree-C. Additionally, from Figure 11b we can indicate that for up to 10K triples the performance of the HETree approaches is almost the same. We can also observe the significant difference between the FLAT and the HETree approaches. Finally, an important observation is that, in practice both the construction and the response time, and thus the overall performance of our tool, grow sub-linearly to the number of triples.

Overall, our hierarchical approaches exhibit reasonable time performance (i.e., sub-linear w.r.t. number of triples), handling properties with 762K objects in about 1min 26s. Finally, it is shown that the hierarchical approaches clearly outperform the employed non-hierarchical approach.

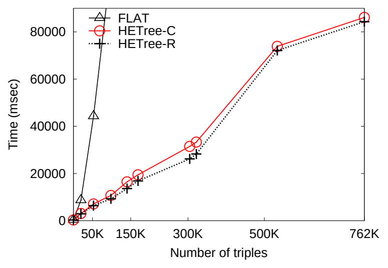
5.3. User Evaluation

In this section we present the user evaluation of our tool, where we have employed three approaches: the two hierarchical and the FLAT. Section 5.3.1 describes the user tasks, Section 5.3.2 outlines the evaluation procedure and setup, Section 5.3.3 summarizes the evaluation results, and Section 5.3.4 discusses issues related to the evaluation process.

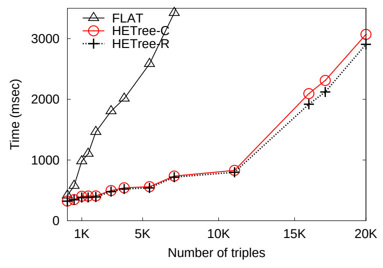
5.3.1. Tasks

In this section we describe the different types of tasks that are used in the user evaluation process.

Type 1 [Find resources with specific value]: This type of tasks requests the resources having value v (as object). For this task type, we define task T1 by selecting a value v that corresponds to 5 resources. Given this task, the participants are asked to provide the number of resources that pertain to this value. In order to solve this task, the participants first have to find a re-



(a) Properties sizes: 50 to 762K triples



(b) Properties sizes: 50 to 20K triples

Fig. 11. Response Time w.r.t. the number of triples

source with value v and then check which of the nearby resources also have the same value.

Type 2 [Find resources in a range of values]: This type of tasks requests the resources having value greater than v_{min} and less than v_{max}. We define two tasks of this type, by selecting different combinations of v_{min} and v_{max} values, such that tasks which consider different number of resources are defined. Particularly, in the first task, named T2.1, we specify the values v_{min} and v_{max} such that a small set of (approximately 10) resources are included, whereas the second task, T2.2, considers a larger set of (approximately 50) resources. Given these tasks, the participants are asked to provide the number of resources included in the given range. This task can be solved by first finding a resource with a value included in the given range, and then explore the nearby resources in order to identify the resources in the given range.

Type 3 [Compare distributions]: This type of tasks requests from the participant to identify whether more resources appear above or below a given value v. For this type, we define task T3, by selecting the value v near to median. Given this task, the participants are asked to provide the number of resources appearing either above or below the value v. The answer for this tasks requires from the participants to indicate the value v and determine the number or resources appearing either before or after this value.

5.3.2. Setup & Procedure

In order to study the effect of the property size in the selected tasks, we select two properties of different sizes from the employed dataset (Section 5.1). The *hsvCoordinateHue* numeric property containing 970 triples, is referred as $Small$, and the *maximumElevation* numeric property, containing 37.936 triples, is referred as $Large$. The first one corresponds to a hierarchy of height 4 and degree 3, and the latter corresponds to a hierarchy of height 7 and degree 3. We should note here, that through the user evaluation, the hierarchy parameters are fixed for all the tasks, and the participants are not allowed to modify them, such that the setting is the same for everyone.

In our evaluation, 10 participants take part, with their age ranging from 25 to 40. The participants are computer science graduate students and researchers who are familiar with information visualization tools. At the beginning of the evaluation, each participant is introduced to the system by an instructor who provides a brief tutorial over the features required for the tasks. After the instructions, the participant is allowed to familiarize themselves with the system. Note that we have integrated in the rdf:SynopsViz the FLAT approach along with the HETree approaches.

During the evaluation, each participant has to perform the previously described four tasks, using all approaches (i.e., HETree-C/R and FLAT), over both the small and large properties. In order to reduce the learning effects and fatigue we define three groups. In the first group, the participants start their tasks with the HETree-C approach, in the second with HETree-R, and in the third with FLAT. Finally, the property (i.e., small, large) first used in each task is counterbalanced among the participants and the tasks. The entire evaluation does not exceed 75 minutes.

Furthermore, for each task (e.g., T2.1, T.3), three task instances are specified by slightly modifying the task parameters. As a result, given a task, a participant has to solve a different instance of this task, in each approach.

For example, in task T2.1, for the HETree-R, the selected v corresponds to a solution of 11 resources, in HETree-C, to 9 resources, whereas for FLAT v corresponds to a solution of 8 resources. The task instance assigned to each approach varies among the participants. Finally, the task instances are selected such that we reduce the possibility for a participant to give a correct answer by guess. Essentially, we try to avoid easily guested answers like 5, 10, 50, etc. That is the reason why, in the aforementioned example for T.2.1, 10 is not considered in solutions.

Table 5. Average Task Completion Time (sec)

	Small Property				Large Property			
	FLAT	HETree-C	HETree-R	p	FLAT	HETree-C	HETree-R	p
T1	54	29	28	★★	85	52	47	★★
T2.1	63	57	64	◆	74	60	69	★
T2.2	120	69	74	★★	128	72	77	★★
T3	262	41	40	★★	—	64	62	—

★ $(p < 0.05)$ ★★ $(p < 0.01)$ ◆ $(p > 0.05)$

Table 6. Error Rate (%)

	Small Property				Large Property			
	FLAT	HETree-C	HETree-R	p	FLAT	HETree-C	HETree-R	p
T1	0	0	0	◆	0	0	0	◆
T2.1	0	0	0	◆	0	0	0	◆
T2.2	20	0	0	◆	20	0	10	◆
T3	70	0	0	★★	—	0	0	—

★ $(p < 0.05)$ ★★ $(p < 0.01)$ ◆ $(p > 0.05)$

During the evaluation the instructor measures the time required for each participant to complete a task, as well as the number of incorrect answers. Tables 5 present the average time required for the participants to complete each task. The table contains the measurements for all approaches, and for both properties. Although we acknowledge that the number of participants in our evaluation is small, we have computed the statistical significant of the results. Essentially, for each property, the p-value of each task is presented in the last column. The p-value is computed using one-way repeated measures ANOVA.

In addition, the results regarding the number of tasks that have been solved incorrectly are presented in Table 6. Particularly, the table presents the percentage of incorrect answers for each task and property, referred as *error rate*. Additionally, for each task and property, the table includes the p-value. Here, the p-value has been computed using Fisher's exact test.

5.3.3. Results

Task T1. Regarding the first task, as we can observe from Table 5, HETree approaches outperform the FLAT, in both property sizes. Note that the time results on T1 are statistical significant ($p < 0.01$).

As expected, all approaches require more time for the Large property compared to the Small one. This overhead in FLAT is caused by from the larger number of resources that the participants have to scroll over and examine, until they indicate the requested resource's value. On the other hand, in HETree, the overhead is caused by the larger number of levels that the Large property hierarchy has. Hence, the participants have to perform more drill-down operations and examine more groups of objects, until they reach the LD resources.

We can also observe that in this task, the HETree-R outperforms the HETree-C in both property sizes. This is due to the fact that, in HETree-R structure, resources having the same value are always contained in the same leaf. As a result the participants have to always inspect only one leaf. On the other hand, in HETree-C this does not always hold, hence the participants may have to explore more than one leaf.

Finally, as we can observe from Table 6, in all cases only correct answers are provided. However, none of those results are statistically significant ($p > 0.05$).

Task T2.1. In the next task, where the participants have to indicate a small set of resources in a range of values, the FLAT performance is very close to the HETree, especially in the Small property (Table 5). In addition, we can observe that the HETree-C approach performs slightly better than the HETree-R. Finally, regarding the statistical significance of the results, in Small property we have $p > 0.05$, while in Large we have $p < 0.005$.

The poor performance of the HETree approaches in this task can be explained by the small set of resources requested and the HETree parameters adopted in the user evaluation. In this setting, the resources contained in the task solution are distributed over more than one leaves. Hence, the participants are required to perform several roll-up and drill-down operations in order to find all the resources. On the other hand, in FLAT once the participants have indicated one of the requested resources, it is very easy for them to find out the rest of the solution's resources. To sum up, in FLAT, most of the time is spent on identifying the first of the resources, while in HETree the first resource is identified very quickly. Regarding the difference in performance between the HETree approaches we have the following. In HETree-C due to the fixed number of objects in each leaf, the participants have to visit at most one or two leaves in order to solve this task. On the other hand, in HETree-R, the number of objects in each leaf is varied, so in most times the participants have to inspect more than two leaves in order to solve the task. Finally, again, for this task, only correct answers are provided (Table 6).

Task T2.2. In this task the participants have to indicate a larger set (compared to the previous task) of resources given a range of values. HETree approaches noticeable outperform the FLAT approach with statistical significance ($p < 0.01$), while similar results are observed in both properties.

In FLAT approach a considerable time is required for the participant to identify and navigate over a large number of resources. On the other hand, due to the large number of resources involved in the task's solution, there are groups in the hierarchy that explicitly contain solution's resources (i.e., they do not contain resources not included in the solution). As a result, the participants in HETree can easily indicate and compute the whole solution by combining the information related to the groups (i.e., number of enclosed resources) and individual resources. Due to the same reasons stated in the previous task (i.e., T2.1), similarly in T2.2 the HETree-C performs slightly better than the HETree-R. Finally, we can observe from Table 6 (but without statistical significance), that it is more difficult for participants to solve correctly this task with FLAT than with HETree.

Task T3. In the last task, participants are requested to find which of the two ranges contain more resources. As it is expected, from Table 5 we can observe that the HETree approaches clearly outperform the FLAT approach with statistical significance in the Small property. This is due to the fact that the participants in FLAT approach have to overview and navigate over almost half of the dataset. As a result, except for the long time required for this process, it is also very difficult to find the correct solution. This is also verified by Table 6 on a statistically significant level. On the other hand, in HETree approaches, the participants can easily find out the answer by considering the resources enclosed by several groups.

Regarding the Large property, as it is expected, it is impossible for participants to solve this task with FLAT, since this requires to parse over and count about 19K resources. As a result none of the participants completed this task using FLAT (indicated with "–" in Table 5), considering the 5 minute time limit used in this task.

5.3.4. Discussion

The user evaluation showed that the hierarchical approaches can be efficient (i.e., require short time in solving tasks) and effective (i.e., have lower error rate) in several cases. In more detail, the HETree approaches perform very well on indicating specific values over a dataset, and given the appropriate parameter setting are marginally affected by the dataset size. Also note that, due to the "vertical-based" exploration, the position (e.g., towards the end) of the requested value in the dataset does not affect the efficiency of the approach. Furthermore, it is shown that the hierarchical approaches can efficiently and effectively handle visual exploration tasks that involve large numbers of objects.

At the end of the evaluation, the participants gave us valuable feedbacks on possible improvements of our tool. Most of the participants criticize several aspects in the interface; since our tool is an early prototype. Also, several participants mentioned difficulties in obtaining their "position" (e.g., which is the currently visualized range of values, or the previously visualized range of values) during the exploration. Finally, some participants mentioned that some hierarchies contained more levels than needed. As previously mentioned, the adopted parameters are not well suited for the evaluation, since hierarchies with larger than 3 degree (and as result less levels) are required.

Finally, additional tasks for demonstrating the capabilities of our model can be considered. However, most of these tasks were not selected in this evaluation, because it was not possible for the participants to perform them with the FLAT approach. An indicative set includes: (1) Find the number of resources (and/or statistics) in the 1st and 3rd quartile; (2) Find statistics (e.g., mean value, variance) for the top-10 or 50 resources; (3) Find the decade (i.e., temporal data) in which most events take place; etc.

6. Related Work

This section reviews works related to our approach on hierarchical visualization and exploration of Linked data. Section 6.1 presents tools and techniques for LD visualization and exploration, Section 6.2 discusses techniques on LD statistical analysis, Section 6.3 present hierarchical data visualization techniques, and finally, Section 6.4 discusses works on data structures & processing related to our HETree data structure.

In Table 7 we provide an overview and compare several visualization tools that offer similar features to our *rdf:SynopsViz* tool. The LD column indicates tools that target the Semantic Web and Linked Data area (i.e., RDF, RDF/S, OWL). The Hierarchical column indicates tools that provide hierarchical visualization of non-hierarchical data. The Statistics column captures the provision of statistics about the visualized data. The Recomm. column indicates tools, which offer recommendation mechanisms for visualization settings (e.g., appropriate visualization type, visualization

Table 7. Visualization Tools Overview

Tool	LD	Hierarchical	Data Types*	Vis. Types**	Statistics	Recomm.	Preferences	Domain	App. Type
Rhizomer [18]	✓	✗	N, T, S, H, G	C, M, T, TL	✗	✓	✗	generic	Web
Payola [43]	✓	✗	N, T, S, H, G	C, CI, G, M, T, TL, TR	✗	✗	✗	generic	Web
LDVM [17]	✓	✗	S, H, G	B, M, T, TR	✗	✓	✗	generic	Web
Vis Wizard [68]	✓	✗	N, T, S	B, C, M, P, PC, SG	✗	✓	✓	generic	Web
LDVizWiz [6]	✓	✗	S, H, G	M, P, TR	✗	✓	✗	generic	Web
SemLens [37]	✓	✗	N	S	✗	✗	✓	generic	Web
LODeX [13]	✓	✗	G	G, M, P	✓	✗	✗	generic	Web
LODWheel [65]	✓	✗	N, S, G	C, G, M, P	✗	✗	✗	generic	Web
RelFinder [36]	✓	✗	G	G	✗	✗	✓	generic	Web
Fenfire [35]	✓	✗	G	G	✗	✗	✗	generic	Web
Lodlive [19]	✓	✗	G	G	✗	✗	✓	generic	Web
IsaViz [54]	✓	✗	G	G	✗	✗	✓	generic	Desktop
graphVizdb [15]	✓	✗	G	G	✗	✗	✓	generic	Web
ViCoMap [56]	✓	✗	N, T, S	M	✓	✓	✓	generic	Web
EDT [50]	✗	✗	N, T, H	C, CM, T, SP	✓	✗	✗	OLAP	Desktop
Polaris [64]	✗	✓	N, T, S, H	C, M, S	✓	✗	✓	OLAP	Desktop
GrouseFlocks [5]	✗	✓	G	G	✗	✗	✓	generic	Desktop
GMine [40]	✗	✓	G	G	✗	✗	✗	generic	Desktop
Gephi [10]	✗	✓	G	G	✓	✗	✓	generic	Desktop
CGV [67]	✗	✓	G	G	✗	✗	✓	generic	Desktop
rdf:SynopsViz	✓	✓	N, T, H	C, P, T, TL	✓	✓	✓	generic	Web

* N: Numeric, T: Temporal, S: Spatial, H: Hierarchical (tree), G: Graph (network)

** B: *bubble chart*, C: *chart*, CI: *circles*, CM: *colormap*, G: *graph*, M: *map*, P: *pie*, PC: *parallel coordinates*, S: *scatter*, SP: *solarplot*, SG: *streamgraph*, T: *treemap*, TL: *timeline*, TR: *tree*.

parameters, etc.). Finally, the Preferences column captures the ability of the users to apply data (e.g., aggregate) or visual (e.g., increase abstraction) operations.

6.1. Linked Data Visualization & Exploration

A large number of works studying issues related to LD visual exploration and analysis have been proposed in the literature [23,51,3]. In what follows, we classify these works into the following categories: (1) Generic visualization tools, (2) Domain, vocabulary & device-specific visualization tools, and (3) Graph-based visualization tools.

6.1.1. Generic Visualization Tools

In the context of LD visual exploration, there is a large number of generic visualization frameworks, that offer a wide range of visualization types and operations. Next, we outline the most known tools in this category.

Rhizomer [18] provides LD exploration based on a overview, zoom and filter workflow. Rhizomer offers various types of visualizations such as maps, timelines, treemaps and charts. *Payola* [43] is a generic framework for LD visualization and analysis. The framework offers a variety of domain-specific (e.g., public procurement) analysis plugins (i.e., analyzers), as well as several visualization techniques (e.g., graphs, tables, etc.). In addition, Payola offers collaborative features for users to create and share analyzers. In Payola the visualizations can be customized according to ontologies used in the resulting data. The *Linked Data Visualization Model* (LDVM) [17] provides an abstract visualization process for LD datasets. LDVM enables the connection of different datasets with various kinds of visualizations in a dynamic way. The visualization process follows a four stage workflow: Source data, Analytical abstraction, Visualization abstraction, and View. LDVM considers several visualization techniques, e.g., circle, sunburst, treemap, etc. *Balloon Synopsis* [58] provides an usable LD visualizer based on HTML and JavaScript. It adopts a node-centric visualization approach in a tile design. Additionally, it supports automatic information enhancement of the local RDF data by accessing either remote SPARQL endpoints or performing federated queries over endpoints using the Balloon Fusion service. Balloon Synopsis offers customizable filters, namely ontology templates, for the users to handle and transform (e.g., filter, merge) input data. *Vis Wizard* [68] is a Web-based visualization tool, which exploits data semantics to simplify the process of setting up visualizations. Vis Wizard is able to analyse multiple datasets using brushing and linking methods. Similarly, *Linked Data Visualization Wizard* (LDVizWiz) [6] provides a semi-automatic way for the production of possible visualization for LD datasets. *SemLens* [37] is a visual tool that combines scatter plots and semantic lenses, offering visual discovery of correlations and patterns in data. Objects are arranged in a scatter plot and are analysed using user-defined semantic lenses. *LODeX* [13] is a tool that generates a representative summary of an LD source. The tool takes as input an SPARQL endpoint and generates a visual (graph-based) summary of the LD source, accompanied by statistical and structural information of the source. *LODWheel* [65] is a Web-based visualizing tool which combines JavaScript libraries (e.g., MooWheel, JQPlot) in order to visualize RDF data in charts and graphs. *Hide the stack* [24] proposes an approach for visualizing LD for mainstream end-users. Underlying Semantic Web technologies (e.g., RDF, SPARQL) are utilized, but are "hidden" from the end-users. Particularly, a template-based visualization approach is adopted, where the information for each resource is presented based on its rdf:type.

6.1.2. Domain, Vocabulary & Device-specific Visualization Tools

In this section, we present tools that target visualization needs for specific types of data and domains, RDF vocabularies or devices.

Several tools focus on visualizing and exploring geo-spatial data. *Map4rdf* [47] is a faceted browsing tool that enables RDF datasets to be visualized on an OSM or Google Map. *Facete* [63] is an exploration and visualization tool for SPARQL accessible data, offering faceted filtering functionalities. *SexTant* [14] focus on visualizing and exploring time-evolving geo-spatial data. Finally, the *LinkedGeoData Browser* [62] is a faceted browser and editor which is developed in the context of LinkedGeoData project. Furthermore, in the context of linked university data, *VISUalization Playground* (VISU) [4] is an interactive tool for specifying and creating visualizations using the contents of linked university data cloud. Particularly, VISU offers a novel SPARQL interface for creating data visualizations. Query results from selected SPARQL endpoints are visualized with Google Charts.

A variety of tools target multidimensional LD modelled with the Data Cube vocabulary. *CubeViz* [28,57] is a faceted browser for exploring statistical data. The tool provides data visualizations using different types of charts (i.e., line, bar, column, area and pie). The *Payola Data Cube Vocabulary* [38] adopts the LDVM stages [17] in order to visualize RDF data described by the Data Cube vocabulary. The same types of charts as in CubeViz are provided in this tool. The *OpenCube Toolkit* [41] offers several tools related to statistical LD. For example, *OpenCube Browser* explores RDF data cubes by presenting a two-dimensional table. Additionally, the *OpenCube Map View* offers interactive map-based visualizations of RDF data cubes based on their geo-spatial dimension. The *Linked Data Cubes Explorer* (LDCE) [42] allows users to explore and analyse statistical datasets. Finally, [53] offers several map and chart visualizations of demographic, social and statistical linked cube data[14].

Regarding device-specific tools, *DBpedia Mobile* [12] is a location-aware mobile application for exploring and visualizing DBpedia resources. *Who's Who* [20] is an application for exploring and visualizing information focusing on several issues that appear in the mobile environment. For example, the application considers the usability and data processing challenges related to the small display size and limited resources of the mobile devices.

6.1.3. Graph-based Visualization Tools

A large number of tools visualize LD datasets adopting a *graph-based* (a.k.a., node-link) approach. *RelFinder* [36] is a Web-based tool that offers interactive discovery and visualization of relationships (i.e., connections) between selected LD resources. *Fenfire* [35] and *Lodlive* [19] are exploratory tools that allow users to browse LD using interactive graphs. Starting from a given URI, the user can explore LD by following the links. *IsaViz* [54] allows users to zoom and navigate over the RDF graph, and also it offers several "edit" operations (e.g., delete/add/rename nodes and edges). In the same context, *graphVizdb* [15] is built on top of spatial and database techniques offering interactive visualization over very large (RDF) graphs.

6.1.4. Discussion

In contrast to the aforementioned approaches, our work does not focus solely on proposing techniques for LD visualization. Instead, we introduce a model for building, visualizing and interacting with hierarchically organized numeric and temporal LD resources. The underlying model is not bound to any specific type of visualization (e.g., chart); rather it can be adopted by several "flat" techniques and offer multi-level visualizations over non-hierarchical data. Also, we present a prototype system that employs the introduced hierarchical model and offers efficient multi-level visual exploration over LD datasets, using charts and timelines.

6.2. Linked Data Statistical Analysis

A second area related to the analysis features of the proposed model deals with LD statistical analysis. *RDFStats* [45] calculates statistical information

[14]www.linked-statistics.gr

about RDF datasets. *LODstats* [8] is an extensible framework, offering scalable statistical analysis of LD datasets. *RapidMiner LOD Extension* [55,52] is an extension of the data mining platform RapidMiner[15], offering sophisticated data analysis operations over LD. *SparqlR*[16] is a package of the R[17] statistical analysis platform. SparqlR executes SPARQL queries over SPARQL endpoints and provides statistical analysis and visualization over SPARQL results. Finally, *ViCoMap* [56] combines LD statistical analysis and visualization, in a Web-based tool, which offers correlation analysis and data visualization on maps.

6.2.1. Discussion

In comparison with these tools, our work does not focus on new techniques for LD statistics computation and analysis. We are primarily interested on enhancing the visualization and user exploration functionality by providing statistical properties of the visualized datasets and objects, making use of existing computation techniques. Also, we demonstrate how in the proposed structure, computations can be efficiently performed on-the-fly and enrich our hierarchical model. The presence of statistics provides quantifiable overviews of the underlying LD resources at each exploration step. This is particularly important in several tasks when you have to explore a large number of either numeric or temporal data objects. Users can examine next levels' characteristics at a glance, this way are not enforced to drill down in lower hierarchy levels. Finally, the statistics over the different hierarchy levels enables analysis over different granularity levels.

6.3. Hierarchical Visualization

The wider area of data and information visualization has provided a variety of approaches for hierarchical analysis and presentation.

Treemaps [60] visualize tree structures using a space-filling layout algorithm based on recursive subdivision of space. Rectangles are used to represent tree nodes, the size of each node is proportional to the cumulative size of its descendant nodes. Finally, a large number of treemaps variations have been proposed (e.g., Cushion Treemaps, Squarified Treemaps, Ordered Treemaps, etc.).

Moreover, hierarchical visualization techniques have been extensively employed to visualize very large graphs using the node-link paradigm. In these techniques the graph is recursively decomposed into smaller sub-graphs that form a hierarchy of abstraction layers. In most cases, the hierarchy is constructed by exploiting clustering and partitioning methods [1,7,10,40,67]. In other works, the hierarchy is defined with hub-based [48] and density-based [71] techniques. Finally, *GrouseFlocks* [5] supports ad-hoc hierarchies which are manually defined by the users.

In the context of data warehousing and *online analytical processing* (OLAP), several approaches provide hierarchical visual exploration, by exploiting the predefined hierarchies in the dimension space. [50] proposes a class of OLAP-aware hierarchical visual layouts; similarly, [66] uses OLAP-based hierarchical stacked bars. *Polaris* [64] offers visual exploratory analysis of data warehouses with rich hierarchical structure.

Finally, several hierarchical techniques have been proposed in the context of *ontology visualization and exploration* [30,26,33,46]. *CropCircles* [69] adopts a hierarchical geometric containment approach, representing the class hierarchy as a set of concentric circles. *Knoocks* [44] combines containment-based and node-link approaches. In this work, ontologies are visualized as nested blocks where each block is depicted as a rectangle containing a sub-branch shown as tree map. Finally, a different approach is followed by *OntoTrix* [9] which combine graphs with adjacency matrices.

6.3.1. Discussion

In contrast to above approaches that target graph-based data, our work focuses on the hierarchical visualization of numeric and temporal data. Moreover, it is not purpose to introduce a new technique for hierarchically-organized data, such as OLAP data. Instead, we propose a model that can be adopted by "flat" visualization techniques, in order to provide multi-level visualizations over non-hierarchical data. Thus, our approach can be considered more flexible compared to the techniques that rely on predefined hierarchies, as it can enable exploratory functionality on dynamically retrieved datasets, by constructing hierarchies on-the-fly, and allowing users to modify these hierarchies.

6.4. Data Structures & Data Processing

In this section we present the data structures and the data (pre-)processing techniques which are the most relevant to our approach.

[15]rapidminer.com
[16]cran.r-project.org/web/packages/SPARQL/index.html
[17]www.r-project.org

R-Tree [32] is disk-based multi-dimensional indexing structure, which has been widely used in order to efficiently handle spatial queries. R-Tree adopts the notion of minimum bounding rectangles (MBRs) in order to hierarchical organize multi-dimensional objects.

Data discretization [31,25] is a process where continuous attributes are transformed into discrete. A large number of methods (e.g., supervised, unsupervised, univariate, multivariate) for data discretization have been proposed. *Binning* is a simple unsupervised discretization method in which a predefined number of bins is created. Widely known binning methods are the *equal-width* and *equal-frequency*. In equal-width approach, the range of an attribute is divided into intervals that have equal width and each interval represents a bin. In equal-frequency approach, an equal number of values are placed in each bin.

By recursively applying discretization techniques, a hierarchical discretization of attribute's values can be produced (a.k.a. *concept/generalization hierarchies*). [59] proposes a dynamic programming algorithm for generating numeric concept hierarchies. The algorithm attempts to maximize both the similarity between the objects stored in the same hierarchy's node, as well as the dissimilarity between the objects stored in different nodes. The generated hierarchy is a balanced tree where different nodes may have different number of children. [34] constructs hierarchies based on data distribution. Essentially, both the leaf and the interval nodes are created in such a way that an even distribution is achieved. The hierarchy construction considers also a threshold specifying the maximum number of distinct values enclosed by nodes in each hierarchy level. Finally, binary concept hierarchies (with degree equal to two) are generated in [21]. Starting from the whole dataset, it performs a recursive binary partitioning over the dataset's values; the recursion is terminated when the number of distinct values in the resultant partitions is less than a pre-specified threshold.

6.4.1. Discussion

The basic concepts of HETree structure can be considered similar to a simplified version of a static 1D R-Tree. However, in order to provide efficient query processing in disk-based environment, R-Tree considers a large number of I/O-related issues (e.g., space coverage, nodes overlaps, fill guarantees, etc.). On the other hand, we introduce a lightweight, main memory structure that efficiently constructed on-the-fly. Also, the proposed structure aims at organizing the data in a practical manner for a (visual) exploration scenario, rather than for disk-based indexing and querying efficiency.

Compared to discretization techniques, our tree model exhibits several similarities, namely, the HETree-C version can be considered as a hierarchical version of the equal-frequency binning, and the HETree-R of the equal-width binning. However, the goal of data organization in HETree is to enable visualization and hierarchical exploration capabilities over dynamically retrieved non-hierarchical data. Hence, compared to the binning methods we can consider the following basic differences. First, in contrast with binning methods that require from the user to specify the number or the size of the bins, our approach is able to automatically estimate the hierarchy parameters and adjust the visualization results by considering the visualization environment characteristics. Secondly, the proposed tree structure is exploited in order to allow efficient statistics computations over different groups of data; then, the statistics are used in order to enhance the overall exploration functionality. Finally, the construction of the model is tailored to the user interaction and preferences; our model offers incremental construction considering the user interaction, as well as efficiently adaptation to the users preferences.

7. Conclusions

In this paper we have presented HETree, a generic model that combines user-customized multi-level exploration with online analysis of numeric and temporal data. Our model is built on top of a lightweight tree-based structure, which can be efficiently constructed on-the-fly for a given set of data. We have presented two variations for constructing our model: the HETree-C structure organizes input data into fixed-size groups, whereas the HETree-R structure organizes input data into fixed-range groups. In that way the users can customize the exploration experience, allowing them to organize data into different ways, by parametrizing the number of groups, the range and cardinality of their contents, the number of hierarchy levels, etc. We have also provided a way for efficiently computing statistics over the tree, as well as a method for automatically deriving from the input dataset the best-fit parameters for the construction of the model. Regarding the performance of multi-level exploration over large datasets, our model offers incremental HETree construction and efficient HETree adaptation based on user's preferences. Based on the introduced model, a Web-based prototype system, called rdf:SynopsViz, has been developed. Finally, the efficiency and the effectiveness of the presented approach are demonstrated via a through performance evaluation and an empirical user study.

Some possible insights for future work include the support of sophisticated methods for data organization in our approach (e.g., organize data in a way, so specific statistics properties hold for the resulted groups). Additionally, the extension of our approach in order to effectively handle multidimensional numeric and temporal data, as well as offer "multidimensional-based" visual exploration operations. Regarding the rdf:SynopsViz tool, we are planning to redesign and extend the graphical user interface, so our tool to be able to use data resulted from SPARQL endpoints, as well as to offer more sophisticated filtering techniques (e.g., SPARQL-enabled browsing over the data). Finally, we are interested in including more visual techniques and libraries.

References

[1] J. Abello, F. van Ham, and N. Krishnan. ASK-GraphView: A Large Scale Graph Visualization System. *IEEE Trans. Vis. Comput. Graph.*, 12(5), 2006.

[2] S. Agarwal, B. Mozafari, A. Panda, H. Milner, S. Madden, and I. Stoica. BlinkDB: queries with bounded errors and bounded response times on very large data. In *EuroSys*, 2013.

[3] F. Alahmari, J. A. Thom, L. Magee, and W. Wong. Evaluating Semantic Browsers for Consuming Linked Data. In *Australasian Database Conference (ADC)*, 2012.

[4] M. Alonen, T. Kauppinen, O. Suominen, and E. Hyvönen. Exploring the Linked University Data with Visualization Tools. In *ESWC*, 2013.

[5] D. Archambault, T. Munzner, and D. Auber. GrouseFlocks: Steerable Exploration of Graph Hierarchy Space. *IEEE Trans. Vis. Comput. Graph.*, 14(4), 2008.

[6] G. A. Atemezing and R. Troncy. Towards a linked-data based visualization wizard. In *Workshop on Consuming Linked Data*, 2014.

[7] D. Auber. Tulip - A Huge Graph Visualization Framework. In *Graph Drawing Software*. 2004.

[8] S. Auer, J. Demter, M. Martin, and J. Lehmann. LODStats - An Extensible Framework for High-Performance Dataset Analytics. In *EKAW*, 2012.

[9] B. Bach, E. Pietriga, and I. Liccardi. Visualizing Populated Ontologies with OntoTrix. *Int. J. Semantic Web Inf. Syst.*, 9(4), 2013.

[10] M. Bastian, S. Heymann, and M. Jacomy. Gephi: An Open Source Software for Exploring and Manipulating Networks. In *Conference on Weblogs and Social Media (ICWSM)*, 2009.

[11] L. Battle, M. Stonebraker, and R. Chang. Dynamic reduction of query result sets for interactive visualizaton. In *IEEE Conference on Big Data*, 2013.

[12] C. Becker and C. Bizer. Exploring the Geospatial Semantic Web with DBpedia Mobile. *J. Web Sem.*, 7(4), 2009.

[13] F. Benedetti, L. Po, and S. Bergamaschi. A Visual Summary for Linked Open Data sources. In *ISWC*, 2014.

[14] K. Bereta, C. Nikolaou, M. Karpathiotakis, K. Kyzirakos, and M. Koubarakis. SexTant: Visualizing Time-Evolving Linked Geospatial Data. In *ISWC*, 2013.

[15] N. Bikakis, J. Liagouris, M. Krommyda, G. Papastefanatos, and T. Sellis. Towards Scalable Visual Exploration of Very Large RDF Graphs. In *ESWC*, 2015.

[16] N. Bikakis, M. Skourla, and G. Papastefanatos. rdf:SynopsViz - A Framework for Hierarchical Linked Data Visual Exploration and Analysis. In *ESWC*, 2014.

[17] J. M. Brunetti, S. Auer, R. García, J. Klímek, and M. Necaský. Formal Linked Data Visualization Model. In *IIWAS*, 2013.

[18] J. M. Brunetti, R. Gil, and R. García. Facets and Pivoting for Flexible and Usable Linked Data Exploration. In *Interacting with Linked Data Workshop*, 2012.

[19] D. V. Camarda, S. Mazzini, and A. Antonuccio. LodLive, exploring the web of data. In *I-SEMANTICS*, 2012.

[20] A. E. Cano, A. Dadzie, and M. Hartmann. *Who's Who* - A Linked Data Visualisation Tool for Mobile Environments. In *ESWC*, 2011.

[21] W. W. Chu and K. Chiang. Abstraction of High Level Concepts from Numerical Values in Databases. In *AAAI Workshop on Knowledge Discovery in Databases*, 1994.

[22] A. Dadzie, V. Lanfranchi, and D. Petrelli. Seeing is believing: Linking data with knowledge. *Information Visualization*, 8(3), 2009.

[23] A. Dadzie and M. Rowe. Approaches to visualising Linked Data: A survey. *Semantic Web*, 2(2), 2011.

[24] A. Dadzie, M. Rowe, and D. Petrelli. *Hide the Stack:* Toward Usable Linked Data. In *ESWC*, 2011.

[25] J. Dougherty, R. Kohavi, and M. Sahami. Supervised and Unsupervised Discretization of Continuous Features. In *International Conference on Machine Learning*, 1995.

[26] M. Dudás, O. Zamazal, and V. Svátek. Roadmapping and Navigating in the Ontology Visualization Landscape. In *EKAW*, 2014.

[27] N. Elmqvist and J. Fekete. Hierarchical Aggregation for Information Visualization: Overview, Techniques, and Design Guidelines. *IEEE Trans. Vis. Comput. Graph.*, 16(3), 2010.

[28] I. Ermilov, M. Martin, J. Lehmann, and S. Auer. Linked Open Data Statistics: Collection and Exploitation. In *Knowledge Engineering and the Semantic Web*, 2013.

[29] D. Fisher, S. M. Drucker, and A. C. König. Exploratory Visualization Involving Incremental, Approximate Database Queries and Uncertainty. *IEEE Computer Graphics and Applications*, 32(4), 2012.

[30] B. Fu, N. F. Noy, and M.-A. Storey. Eye Tracking the User Experience - An Evaluation of Ontology Visualization Techniques. *Semantic Web Journal (to appear)*, 2015.

[31] S. García, J. Luengo, J. A. Sáez, V. López, and F. Herrera. A Survey of Discretization Techniques: Taxonomy and Empirical Analysis in Supervised Learning. *IEEE Trans. Knowl. Data Eng.*, 25(4), 2013.

[32] A. Guttman. R-Trees: A Dynamic Index Structure for Spatial Searching. In *SIGMOD*, 1984.

[33] F. Haag, S. Lohmann, S. Negru, and T. Ertl. OntoViBe: An Ontology Visualization Benchmark. In *Workshop on Visualizations and User Interfaces for Knowledge Engineering and*

Linked Data Analytics, 2014.
[34] J. Han and Y. Fu. Dynamic Generation and Refinement of Concept Hierarchies for Knowledge Discovery in Databases. In *AAAI Workshop on Knowledge Discovery in Databases*, 1994.
[35] T. Hastrup, R. Cyganiak, and U. Bojars. Browsing Linked Data with Fenfire. In *WWW*, 2008.
[36] P. Heim, S. Lohmann, and T. Stegemann. Interactive Relationship Discovery via the Semantic Web. In *ESWC*, 2010.
[37] P. Heim, S. Lohmann, D. Tsendragchaa, and T. Ertl. SemLens: visual analysis of semantic data with scatter plots and semantic lenses. In *I-SEMANTICS*, 2011.
[38] J. Helmich, J. Klímek, and M. Necaský. Visualizing RDF Data Cubes Using the Linked Data Visualization Model. In *ESWC*, 2014.
[39] J. Im, F. G. Villegas, and M. J. McGuffin. VisReduce: Fast and responsive incremental information visualization of large datasets. In *IEEE Conference on Big Data*, 2013.
[40] J. F. R. Jr., H. Tong, J. Pan, A. J. M. Traina, C. T. Jr., and C. Faloutsos. Large Graph Analysis in the GMine System. *IEEE Trans. Knowl. Data Eng.*, 25(1), 2013.
[41] E. Kalampokis, A. Nikolov, P. Haase, R. Cyganiak, A. Stasiewicz, A. Karamanou, M. Zotou, D. Zeginis, E. Tambouris, and K. A. Tarabanis. Exploiting Linked Data Cubes with OpenCube Toolkit. In *ISWC*, 2014.
[42] B. Kämpgen and A. Harth. OLAP4LD - A Framework for Building Analysis Applications Over Governmental Statistics. In *ESWC*, 2014.
[43] J. Klímek, J. Helmich, and M. Necaský. Payola: Collaborative Linked Data Analysis and Visualization Framework. In *ESWC*, 2013.
[44] S. Kriglstein and R. Motschnig-Pitrik. Knoocks: New Visualization Approach for Ontologies. In *Conference on Information Visualisation*, 2008.
[45] A. Langegger and W. Wöß. RDFStats - An Extensible RDF Statistics Generator and Library. In *DEXA*, 2009.
[46] M. Lanzenberger, J. Sampson, and M. Rester. Visualization in Ontology Tools. In *CISIS*, 2009.
[47] A. d. Leon, F. Wisniewki, B. Villazón-Terrazas, and O. Corcho. Map4rdf- Faceted Browser for Geospatial Datasets. In *Using Open Data: policy modeling, citizen empowerment, data journalism*, 2012.
[48] Z. Lin, N. Cao, H. Tong, F. Wang, U. Kang, and D. H. P. Chau. Demonstrating Interactive Multi-resolution Large Graph Exploration. In *IEEE Data Mining Workshops*, 2013.
[49] Z. Liu, B. Jiang, and J. Heer. *imMens*: Real-time Visual Querying of Big Data. *Comput. Graph. Forum*, 32(3):421–430, 2013.
[50] S. Mansmann and M. H. Scholl. Exploring OLAP aggregates with hierarchical visualization techniques. In *SAC*, 2007.
[51] N. Marie and F. L. Gandon. Survey of Linked Data Based Exploration Systems. In *IESD*, 2014.
[52] H. Paulheim. Generating Possible Interpretations for Statistics from Linked Open Data. In *ESWC*, 2012.
[53] I. Petrou, M. Meimaris, and G. Papastefanatos. Towards a methodology for publishing Linked Open Statistical Data. *eJournal of eDemocracy & Open Government*, 6(1), 2014.
[54] E. Pietriga. IsaViz: a Visual Environment for Browsing and Authoring RDF Models. In *WWW*, 2002.
[55] P. Ristoski, C. Bizer, and H. Paulheim. Mining the Web of Linked Data with RapidMiner. In *ISWC*, 2014.
[56] P. Ristoski and H. Paulheim. Visual Analysis of Statistical Data on Maps using Linked Open Data. In *ESWC*, 2015.
[57] P. E. R. Salas, F. M. D. Mota, K. K. Breitman, M. A. Casanova, M. Martin, and S. Auer. Publishing Statistical Data on the Web. *Int. J. Semantic Computing*, 6(4), 2012.
[58] K. Schlegel, T. Weißgerber, F. Stegmaier, C. Seifert, M. Granitzer, and H. Kosch. Balloon Synopsis: A Modern Node-Centric RDF Viewer and Browser for the Web. In *ESWC*, 2014.
[59] C. Shen and Y. Chen. A dynamic-programming algorithm for hierarchical discretization of continuous attributes. *European Journal of Operational Research*, 184(2), 2008.
[60] B. Shneiderman. Tree Visualization with Tree-Maps: 2-d Space-Filling Approach. *ACM Trans. Graph.*, 11(1), 1992.
[61] B. Shneiderman. The Eyes Have It: A Task by Data Type Taxonomy for Information Visualizations. In *IEEE Symposium on Visual Languages*, 1996.
[62] C. Stadler, J. Lehmann, K. Höffner, and S. Auer. LinkedGeoData: A core for a web of spatial open data. *Semantic Web*, 3(4), 2012.
[63] C. Stadler, M. Martin, and S. Auer. Exploring the web of spatial data with facete. In *WWW*, 2014.
[64] C. Stolte, D. Tang, and P. Hanrahan. Query, analysis, and visualization of hierarchically structured data using Polaris. In *SIGKDD*, 2002.
[65] M. Stuhr, D. Roman, and D. Norheim. LODWheel - JavaScript-based Visualization of RDF Data. In *Workshop on Consuming Linked Data*, 2011.
[66] K. Techapichetvanich and A. Datta. Interactive Visualization for OLAP. In *ICCSA*, 2005.
[67] C. Tominski, J. Abello, and H. Schumann. CGV - An interactive graph visualization system. *Computers & Graphics*, 33(6), 2009.
[68] G. Tschinkel, E. E. Veas, B. Mutlu, and V. Sabol. Using Semantics for Interactive Visual Analysis of Linked Open Data. In *ISWC*, 2014.
[69] T. D. Wang and B. Parsia. CropCircles: Topology Sensitive Visualization of OWL Class Hierarchies. In *ISWC*, 2006.
[70] A. Zaveri, A. M. Anisa Rula, R. Pietrobon, J. Lehmann, and S. Auer. Quality assessment methodologies for linked open data. *Semantic Web Journal (to appear)*, 2015.
[71] M. Zinsmaier, U. Brandes, O. Deussen, and H. Strobelt. Interactive Level-of-Detail Rendering of Large Graphs. *IEEE Trans. Vis. Comput. Graph.*, 18(12), 2012.

Appendix

A. Interactive HETree Construction

Remark 1. Each time ICO constructs a node (either as part of initial nodes or due to a construction rule), it also constructs all of its sibling nodes.

Proof of Proposition 1. Considering the different cases of currently visualized HETree elements and the available exploration operations, we have the following.

(1) *A set of (internal or leaf) sibling nodes S are visualized and the user performs a roll-up action.* Here,

the roll-up action will visualize the parent node of S along with parent's sibling nodes. In the case that S are the nodes of interest (RAN scenario), the visualized nodes have been constructed in the beginning of the exploration (as part of RAN initial nodes). Otherwise, the visualized nodes have been previously constructed due to construction $Rule\ 1(i)$.

(2) *A set of internal sibling nodes C are visualized and the user performs a drill-down action over a node $c \in C$.* In this case, the drill-down will visualize c child nodes. If C are the nodes of interest (RAN scenario), then the child nodes of c have been constructed at the beginning of the exploration (as part of RAN initial nodes). Else, if C is the root node (BSC scenario), then again the child nodes of c have been constructed at the beginning of the exploration (as part of BSC initial nodes). Otherwise, the children of c, have been constructed before due to construction $Rule\ 1(ii)$.

(3) *A set of leaf sibling nodes L are visualized and the user performs a drill-down action over a leaf $l \in L$.* In this case the drill-down action will visualize data objects contained in l. Since a leaf is constructed together with its data objects, all data objects here have been previously constructed along with l.

(4) *A set of data objects O are visualized and the user performs a roll-up action.* Here, the roll-up action will visualize the leaf that contains O along with the leaf's siblings. In RAN and BSC exploration scenarios, data objects are reachable only via a drill-down action over the leaf over the leaf that are contained, whereas in the RES scenario, the data objects, contained in the leaf of interest, are the first elements that are visualized to the user.

In the general case, since O are reached only via a drill-down, their parent leaf has already been constructed. Based on Remark 1, all sibling nodes of this leaf have also been constructed. In the case of the RES scenario, where O includes the resource of interest, the leaf that contains O along with leaf's siblings have been constructed at the beginning of the exploration.

Thus, it is shown that, in all cases, the HETree elements that a user can reach by performing one operation, have been previously constructed by ICO. This concludes the proof of Proposition 1. ■

Proof of Theorem 1. We will show that, during an exploration scenario, in any exploration step, ICO constructs only the required HETree elements. Considering an exploration scenario, ICO constructs nodes only either as initial nodes, or via construction rules. The initial nodes are constructed once, at the beginning of the exploration process; based on the definition of the initial nodes, these nodes are the required HETree elements for the first user operation.

During the exploration process, ICO constructs nodes only via the construction rules. Essentially, from construction rules, only the $Rule\ 1$ construct new nodes. Considering the part of the tree visualized when $Rule\ 1$ is applied, it is apparent that the nodes constructed by $Rule\ 1$ are only the required HETree elements.

Therefore, it is apparent that in any exploration step, ICO constructs only the required HETree elements. By considering all the steps comprising a user exploration scenario, the overall number of elements constructed is the minimum. This concludes the proof of Theorem 1. ■

Procedure 6: computeSiblingInterv-R(low, up, len, n)

```
Input: $low$: intervals' lower bound; $up$: intervals' upper bound;
       $len$: intervals' length; $n$: number of siblings
Output: $I$: an ordered set with at most $n$ equal length intervals
$I_t^-$ , $I_t^+ \leftarrow low$
for $i \leftarrow 1$ to $n$ do
    $I_t^- \leftarrow I_t^+$
    $I_t^+ \leftarrow \min(up\,, len + I_t^-)$
    append $I_t$ to $I$
    if $I_t^+ = up$ then break
return $I$
```

Procedure 7: constrSiblingNodes-R(I, p, A, h)

```
Input: $I$: an ordered set with equal length intervals $p$: nodes' parent
       node; $A$: available data objects; $h$: nodes' height
Output: $S$: a set of HETree-R sibling nodes
$l = I[1]^+ - I[1]^-$                                   //intervals' length
$T[\,] \leftarrow \varnothing$
foreach $tr \in A$ do                   //indicate enclosed data for each node
    $j \leftarrow \left\lfloor \frac{tr.o - I[1]^-}{l} \right\rfloor + 1$
    if $j \geq 0$ and $j \leq |I|$ then
        insert triple $tr$ into $T[j]$
        remove triple $tr$ from $A$
for $i \leftarrow 1$ to $|I|$ do                           //construct nodes
    if $T[i] = \varnothing$ then continue
    create a new node $n$
    $n.I^- \leftarrow I[i]^-$
    $n.I^+ \leftarrow I[i]^+$
    $n.p \leftarrow p$
    $n.c \leftarrow$ null
    $n.data \leftarrow T[i]$
    $n.h \leftarrow h$
    if $h = 0$ then                                      //node is a leaf
        sort $n.data$ based on objects values
    append $n$ to $S$
return $S$
```

B. ICO Computational Analysis

In this section, we analyse in details the worst case of ICO algorithm, i.e., when the construction cost is maximized.

B.1. The HETree-R Version

The worst case in HETree-R occurs when the whole dataset D is contained in a set of sibling leaves nodes L, where $|L| \leq d$.

Considering the above setting, in the RES scenario, the cost is maximized when ICO-R constructs L (as initial nodes). In this case, the cost is $O(|D| + |D|log|D|) = O(|D|log|D|)$.

In a RAN scenario, the cost is maximized when the parent node p of L along with p's sibling nodes are considered as nodes of interest. First, let's note that in this case p has no sibling nodes, since all the sibling nodes are empty (i.e., they do not enclose data). Hence, the p has to be constructed in ICO-R as initial nodes, as well as the L in constrDrillDown-R, and the parent of p in constrRollUp-R. The p construction in ICO-R requires $O(|D|)$. Also, the L construction in constrDrillDown-R requires $O(d + |D| + |D|log|D| + d)$. Finally, the construction of the parent of p in constrRollUp-R requires $O(1)$. Therefore, in RAN the overall cost in the worst case is $O(|D| + d + |D| + |D|log|D| + d) = O(|D|log|D|)$.

Finally, in BSC scenario, the cost is maximized when the L have to be constructed by constrDrillDown-R, which requires $O(d + |D| + |D|log|D| + d) = O(|D|log|D|)$.

B.2. The HETree-C Version

First let's note that in HETree-C version, the dataset is sorted at the beginning of the exploration and the leaves contain equal number of data objects. As a result, during a node construction, the data objects, enclosed by it, can be directly identified by computing its position over the dataset and without the need of scanning the dataset or the enclosed data values. However, in ICO we assume that the node's statistics are computed each time the node is constructed. Hence, in each node construction, we scan the data objects that are enclosed by this node. In RES scenario, the worst case occurs, when the user rolls up for the first time to the nodes at level 2 (i.e., two levels below the root). In this case, ICO has to construct the d nodes at level 1, as well the children for the $d-1$ nodes in level 2. Note that the construction of the parent of the nodes in level 2 does not require to process any data objects or construct children, since these nodes are already constructed. Now regarding the construction of the rest $d-1$ nodes at level 1, ICO will process at most the $\frac{d-1}{d}$ of all data objects[18]. Thus, the cost for constrRollUp-C is $O(d + \frac{d-1}{d}|D| + d - 1)$. Finally, for constructing the child nodes for the $d-1$ nodes in level 2, we are required to process at most the $\frac{d-1}{d^2}$ of all data objects. Hence, the cost for constrDrillDown-C is $O(d^2 + \frac{d-1}{d^2}|D| + d^2)$. Therefore, in RES the cost in worst case is $O(d + \frac{d-1}{d}|D| + d - 1 + d^2 + \frac{d-1}{d^2}|D| + d^2) = O(d^2 + \frac{d-1}{d}|D|)$.

In RAN scenario, the worst case occurs, when the user starts from any set of sibling nodes at level 2. Hence, the cost is maximized at the beginning of the exploration. In this case, ICO has to construct the d initial nodes at level 2, the d nodes at level 1, and the children for all the d nodes in level 2. First the d initial nodes at level 2 are constructed by ICO-R, which can be done in $O(d + |D| + d)$. Then, the d nodes at level 1 are constructed by constrRollUp-C. Similarly as in RES scenario, this can be done in $O(d + \frac{d-1}{d}|D| + d - 1)$. Finally, the construction of the child nodes for all the d nodes in level 2 requires to process $\frac{|D|}{d}$ data objects. Hence, the cost for constrDrillDown-C is $O(d^2 + \frac{|D|}{d} + d^2)$. Therefore, in RAN the cost, in the worst case, is $O(|D| + d + d + \frac{d-1}{d}|D| + d - 1 + d^2 + \frac{|D|}{d} + d^2) = O(d^2 + \frac{d-1}{d}|D|)$.

Finally, in BSC scenario, the worst case occurs, when the user visits for the first time any of the node at level 1. In this case, ICO has to construct the children for the d nodes in level 1. Hence, constrDrillDown-C has to process $|D|$ data objects in order to construct the d^2 child nodes. Therefore, in BSC the cost in worst case is $O(d^2 + |D| + d^2) = O(d^2 + |D|)$.

C. Analysis of ADA Reconstruction Process

C.1. Preliminaries

In order to perform traversal over the levels of the HETree (i.e., level-order traversal), we use an array $\mathcal{H}$ of pointers to the ordered set of nodes at each level, with $\mathcal{H}[0]$ referring to the set of leaf nodes and $\mathcal{H}[k]$ referring to the set of nodes at height k. Moreover, we

[18]This number can be easily computed by considering the number of leafs enclosed by these nodes.

consider the following simple procedures that are used for the ADA implementation:

– $\mathsf{mergeLeaves}(L, m)$, where L is an ordered set of leaf nodes and $m \in \mathbb{N}^+$, with $m > 1$. This procedure returns an ordered set of $\lceil \frac{L}{m} \rceil$ new leaf nodes, i.e., each new leaf merges m leaf nodes from L. The procedure traverses L, constructs a new leaf for every m nodes in L and appends the data items from the m nodes to the new leaf. This procedure requires $O(|L|)$.

– $\mathsf{replaceNode}(n_1, n_2)$, replaces the node n_1 with the node n_2; it removes n_1, and updates the parent of n_1 to refer to n_2. This procedure requires constant time, hence $O(1)$.

– $\mathsf{createEdges}(P, C, d)$, where P, C are ordered sets of nodes and d is the tree degree. It creates the edges (i.e., parent-child relations) from the parent nodes P to the child nodes C, with degree d. The procedure traverses over P and connects each node $P[i]$ with the nodes from $C[(i-1)d+1]$ to $C[(i-1)d+d]$. This procedure requires $O(|C|)$.

C.2. The User Modifies the Tree Degree

C.2.1. The user increases the tree degree

(1) $d' = d^k$, with $k \in \mathbb{N}^+$ and $k > 1$

$Tree\ Construction.$ For the $\mathcal{T}'$ construction, we perform a reverse level-order traversal over $\mathcal{T}$, using the $\mathcal{H}$ vector. Starting from the leaves ($\mathcal{H}[0]$), we skip (i.e., remove) $k-1$ levels of nodes. Then, for the nodes of the above level ($\mathcal{H}[k]$), we create child relations with the (non-skipped) nodes in the level below. The above process continues until we reach the root node of $\mathcal{T}$.

Hence, in this case all nodes in $\mathcal{T}'$ are obtained "directly" from $\mathcal{T}$. Particularly, $\mathcal{T}'$ is constructed using the root node of $\mathcal{T}$, as well the $\mathcal{T}$ nodes from $\mathcal{H}[j \cdot k], j \in \mathbb{N}^0$.

The $\mathcal{T}'$ construction requires the execution of createEdges procedure, j times. For computing j, we have that $j \cdot k \leq |\mathcal{H}| \Leftrightarrow j \cdot k \leq log_d \ell$. Considering that $d' = d^k$, we have that $k = log_d d'$. Hence, $j \cdot log_d d' \leq log_d \ell \Leftrightarrow j \leq log_d(\ell - d')$. So, considering that worst case complexity for createEdges is $O(\ell)$, we have that the overall complexity is $O(\ell \cdot log_d(\ell - d'))$. Since we have that $\ell \leq |D|$, then in worst case the $\mathcal{T}'$ can be constructed in $O(|D| log_d(|D|)) = O(|D| log_{\sqrt[k]{d'}}(|D|))$.

(2) $d' = k \cdot d$, with $k \in \mathbb{N}^+$, $k > 1$ and $k \neq d^\nu$ where $\nu \in \mathbb{N}^+$

$Tree\ Construction.$ As in $\mathcal{T}'$ the leaves remain the same as in $\mathcal{T}$, we only use the constrInterNodes (Procedure 2) to build the rest of the tree. Therefore, in the worst case, the complexity for constructing the $\mathcal{T}'$ is $O(\frac{d'^2 \cdot \ell - d'}{d'-1})$.

$Statistics\ Computations.$ The statistics for $\mathcal{T}'$ nodes of height 1 can be computed by aggregating statistics from $\mathcal{T}$. Particularity, in $\mathcal{T}'$ the statistics computations for each internal node of height 1, require $O(k)$ instead of $O(d')$, where $k = \frac{d'}{d}$. Hence, considering that there are $\lceil \frac{\ell}{d'} \rceil$ internal nodes of height 1 in $\mathcal{T}'$, the cost for their statistics is $O(k \cdot \lceil \frac{\ell}{d'} \rceil) = O(\frac{k \cdot \ell}{d'} + k)$.

Regarding the cost of recomputing them from scratch, consider that there are $\frac{\ell - 1}{d'-1}$ internal nodes[19] with heights greater than 1; the statistics computations for these nodes require $O(\frac{d' \cdot \ell - d'}{d'-1})$. Therefore, the overall cost for statistics computations is $O(\frac{k \cdot \ell}{d'} + k + \frac{d' \cdot \ell - d'}{d'-1})$.

(3) $elsewhere$

$Tree\ Construction.$ Similar to the previous case, the $\mathcal{T}'$ construction requires $O(\frac{d'^2 \cdot \ell - d'}{d'-1})$.

$Statistics\ Computations.$ In this case, the statistics should be computed from scratch for all internal nodes in $\mathcal{T}'$. Therefore, the complexity is $O(\frac{d'^2 \cdot \ell - d'}{d'-1})$.

C.2.2. The user decreases the tree degree

(1) $d' = \sqrt[k]{d}$, with $k \in \mathbb{N}^+$ and $k > 1$.

$Tree\ Construction.$ For the $\mathcal{T}'$ construction we perform a reverse level-order traversal over $\mathcal{T}$ using the $\mathcal{H}$ vector and starting from the nodes having height of 1. In each level, for each node n we call the constrInterNodes (Procedure 2) using[20] as input the d child nodes of n and the new degree d'. Hence, the complexity of constrInterNodes for one call is $O(d)$. Considering that, we perform a procedure call for all the internal nodes, as well as that the maximum number of internal nodes is $\frac{d \cdot \ell - 1}{d-1}$, we have that, in the

[19]Take into account that the maximum number of internal nodes (considering all levels) is $\frac{d^{\lceil log_d \ell \rceil} - 1}{d-1}$.

[20]Note that, in this reconstruction case, the constrInterNodes does require to construct the root node; the root node here is always corresponding to the node n.

worst case the $\mathcal{T}'$ can be constructed in $O(\frac{d^2 \cdot \ell - d}{d-1}) = O(\frac{d'^{2k} \cdot \ell - d'^k}{d'^k - 1})$.

Regarding the number of internal nodes that we have to construct from scratch. Since $\mathcal{T}'$ has all the nodes of $\mathcal{T}$, for $\mathcal{T}'$ we have to construct from scratch $\frac{d' \cdot \ell - 1}{d' - 1} - \frac{d \cdot \ell - 1}{d-1}$ new internal nodes, where the first part corresponds to the number of internal nodes of $\mathcal{T}'$, and the second part corresponds to $\mathcal{T}$. Considering that, $d' = \sqrt[k]{d}$, we have to build $\frac{d' \cdot \ell - 1}{d' - 1} - \frac{d'^k \cdot \ell - 1}{d'^k - 1}$ internal nodes.

Statistics Computations. Statistics should be computed only for the new internal nodes of $\mathcal{T}'$. Hence, the cost here is $O(d' \cdot (\frac{d' \cdot \ell - 1}{d' - 1} - \frac{d'^k \cdot \ell - 1}{d'^k - 1}))$

C.3. The User Modifies the Number of Leaves

C.3.1. The user decreases the number of leaves

(1) $\ell' = \dfrac{\ell}{d^k}$, with $k \in \mathbb{N}^+$

Tree Construction. In this case[21], each leaf in $\mathcal{T}'$ is resulted by merging d^k leaves from $\mathcal{T}$. Hence, $\mathcal{T}'$ leaves are constructed by calling mergeLeaves(ℓ, d^k). So, considering the mergeLeaves complexity, in worst case the new leaves construction requires $O(|D|)$. Then, each leaf of $\mathcal{T}'$ replace an internal nodes of $\mathcal{T}$ having height of k. Therefore, in worst case ($k = 1$), we call $\lceil \frac{\ell}{d} \rceil$ times the replaceNode procedure, which requires $O(\lceil \frac{\ell}{d} \rceil)$. Therefore, the overall cost for constructing $\mathcal{T}'$ in worst case is $O(|D| + \lceil \frac{\ell}{d} \rceil) = O(|D|)$.

(2) $\ell' = \dfrac{\ell}{k}$, with $k \in \mathbb{N}^+$, $k > 1$ and $k \neq d^\nu$, where $\nu \in \mathbb{N}^+$

Tree Construction. In this case, each leaf in $\mathcal{T}'$ is resulted by merging k leaves from $\mathcal{T}$. Hence, the $\mathcal{T}'$ leaves are constructed by calling the mergeLeaves(ℓ, k), which in worst case requires $O(|D|)$. Then, the rest of the tree is constructed from scratch using the constrInterNodes. Therefore, the overall cost for $\mathcal{T}'$ construction is $O(|D| + \frac{d^2 \cdot \ell' - d}{d-1})$.

Statistics Computations. The statistics for all internal nodes have to be computed from scratch. Regarding the leaves, the statistics for each leaf in $\mathcal{T}'$ are computed by aggregating the statistics of the $\mathcal{T}$ leaves it includes. Essentially, for computing the statistics in each leaf in an HETree-C, we have to process k values instead of $\frac{|D|}{\ell'}$. However, in the worst case (i.e., $\ell = |D|$), we have that $k = \frac{|D|}{\ell'}$. Therefore, in the worst case (for both HETree versions) the complexity is the same as computing statistics from scratch.

(3) $\ell' = \ell - k$, with $k \in \mathbb{N}^+$, $k > 1$ and $\ell' \neq \dfrac{\ell}{\nu}$, where $\nu \in \mathbb{N}^+$

Tree Construction. In order to construct $\mathcal{T}'$ we have to construct all nodes from scratch, which in the worst case requires $O(|D| log |D| + \frac{d^2 \cdot \ell' - d}{d-1})$.

Statistics Computations. The statistics of the $\mathcal{T}$ leaves that are fully contained in $\mathcal{T}'$ can be used for calculating the statistics of the new leaves. The worst case is when the number of leaves that are fully contained in $\mathcal{T}'$ is minimized. For HETree-C (resp. HETree-R), this occurs when the size of leaves in $\mathcal{T}'$ is $\lambda' = \lambda + 1$ (resp. of length $\rho' = \rho + \frac{\rho}{\ell}$). In this case, for every λ (resp. ρ) leaves from $\mathcal{T}$ that are used to construct the $\mathcal{T}'$ leaves, at least 1 leaf is fully contained. Hence, when we process all ℓ leaves, at least $\frac{\ell^2}{|D|}$ leaves are fully contained in $\mathcal{T}'$.

Hence in HETree-C, in statistics computations over the leaves, instead of processing $|D|$ values, we process at most $|D| - \frac{\ell^2}{|D|} \cdot \lambda = |D| - \ell$. The same also holds in HETree-R, if we assume normal distribution over the values in D. Therefore, the cost for computing the leaves statistics is $O(|D| - \ell) = O(|D| - \ell' - k)$.

[21]Note that in this, as well as in the following case, we assume that $\mathcal{T}$ is a perfect tree. In case where $\mathcal{T}$ is not perfect, we can use as $\mathcal{T}$ the perfect tree that initially proposed by our system.