VP-Viewer: Keeping track of your query from a vantage point*†

Daniel L. Jasbick[1], Thaylon Guedes[2], Rodolfo A. Oliveira[1], Lúcio F. D. Santos[3], Daniel de Oliveira[2], and Marcos V. N. Bedo[1]

[1]Fluminense Northwest Institute – Fluminense Federal University (UFF)

{danieljasbick, rodolfooliveira, marcosbedo}@id.uff.br

[2]Institute of Computing – Fluminense Federal University (UFF)

thaylongs@id.uff.br, danielcmo@ic.uff.br

[3]Federal Institute of Technology North of Minas Gerais (IFNMG)

lucio.santos@ifnmg.edu.br

Abstract. *VP-Tree is the metric sibling of Binary Tree and k-Dimensional Tree indexing structures. However, visual exploration of VP-Trees is an open, yet important, issue as drawing crisp borders of VP-Tree partitions over a 2D transformation of metric data is often unachievable. In this demonstration, we present VP-Viewer, an index visualization tool for the investigation of the VP-Tree structure and the inspection of query paths. VP-Viewer builds upon a metric space library and enables the construction of parameterized VP-Trees, in which methods for distance calculation, pivot selection, and index balancing, besides the datasets themselves, are provided by the users. VP-Viewer renders VP-Trees by distinguishing directory nodes, which include vantage points, partition characteristics, and pivot-based distance distributions, from leaf nodes, which encompass the data elements and their distance to vantage points. Accordingly, users can easily explore the partitioning of a dataset for distinct parameterizations. Finally, VP-Viewer also enables the submission of range and kNN queries so that users can evaluate the tree branches examined by the searching algorithms.*

1. Introduction

Similarity searching is a base paradigm for the handling of data that are "alike" but not "equal". Such paradigm supports a variety of computational tasks, such as distance-based classification and content-based retrieval [Chávez et al. 2001, Padmanabhan and Deshpande 2015]. In practice, two of the most requested similarity searches are the range and neighborhood queries. An example of a range query in the bioinformatics domain is **(Q1)** Select all polypeptide chains that are different from a given chain by at most 3 codons, whereas a neighborhood (kNN) query example in the biomedical domain is **(Q2)** Find the 15 images of Magnetic Resonance Imaging (MRI) from distinct studies which are the most similar to a given MRI image of an (undiagnosed) patient. Range and neighborhood queries can be modeled upon *metric spaces*, where the

*The authors thank FAPEMIG, FAPERJ, CNPq and CAPES for their financial support.

†https://github.com/Jasbick/VP-Viewer

elements (polypeptide chains and MRI images, in the aforementioned examples) are represented as points and the (dis)similarity between each pair of points is evaluated by a distance function that complies to the symmetry, positivity and triangle inequality properties [Hetland 2009, Padmanabhan and Deshpande 2015].

Several metric access methods have been proposed to speed up similarity-based queries [Chávez et al. 2001, Chen et al. 2017]. Such methods accelerate similarity searches by targeting an optimization criterion, such as the number of disk accesses, the number of distance calculations, or the overhead caused by the searching algorithm [Chávez et al. 2001, Hetland 2009]. *Vantage-Point Trees* (VP-Trees) access methods are particularly versatile for enhancing query executions, as they enable the organization of the search space in a hierarchical and disjoint fashion [Li et al. 2014]. VP-Trees are indexing structures that extend the concept of a Binary Trees for the querying of metric spaces within logarithm time complexity once each decision of searching either a left or right node may halve the number of subtrees to be evaluated [Yianilos 1993].

Roughly speaking, VP-Trees organize data from a set $\mathcal{S}$ using a *pivot* p, a median μ of distances from p to elements in $\mathcal{S}$, a maximum distance d_m between p and any element in $\mathcal{S}$, and two partitions generated from p: *left* and *right* nodes. Elements whose distances to p fall inside the $[0, \mu)$ interval are assigned to the left node, whereas elements of the $[\mu, d_m]$ interval are set to the right node. The left and right nodes are datasets themselves and can be recursively divided until either each node becomes a unitary set, or a maximum number of elements per leaf node is reached. The last criterion generates the VP-Tree variation called vp^{sb}-tree, which we shall examine hereafter. Median μ is a careful choice for the disjointed partitioning of $\mathcal{S}$ as unique medians would split the dataset into a perfectly balanced tree. Such *uniqueness*, however, depends on the distance distribution so that the resulting tree may be unbalanced if leaf nodes are unable to handle overflow. Likewise, the method for selecting VP-Tree pivots directly affects tree balancing and branching-based search quality [Li et al. 2014]. VP-Tree pivot set includes the data elements that *maximize* the variance of the distance distribution [Hetland 2009], but the solution for fetching such an optimal set is polynomial [Ruiz et al. 2013]. Alternatively, several heuristics can be used for reducing the pivot selection costs for large databases, such as *Randomness*, *Sampling*, and *Convex Hull Points* [Chávez et al. 2001].

Finding the most suitable setting of a VP-Tree, *i.e.*, pivot selection and tree balancing, is often unintuitive and experimentally burdensome [Li et al. 2014, Chen et al. 2017]. In this demonstration, we present VP-Viewer, a tool for assisting users in both understanding and assessment of VP-Trees. Although existing applications, *e.g.*, the C++-based MAM-View [Chino et al. 2010] or a JavaScript-based web solution[1], can be used for the visualization of indexed metric data, they focus on rendering 2D representations of data partitioning and can express neither the relationship between VP-Tree nodes nor the distance distributions within rooting nodes. Unlike these previous approaches, VP-Viewer distinguishes VP-Tree *directory* nodes, which include vantage points, partition lower and upper bounds, covered number of elements, and pivot-based distance distributions, from *leaf* nodes, which encompass the data elements and their distance to vantage points. Accordingly, users can easily explore the partitioning of a dataset for distinct parameterizations. Furthermore, VP-Viewer is not only limited to generate VP-Tree visualizations.

[1] `https://fribbels.github.io/vptree/vptree.html`

Our tool also enables the user to submit range and neighborhood queries so that they can interactively evaluate the tree nodes that were examined by the VP-Tree branch-and-bound algorithms, *i.e.*, users can compare distinct VP-Tree searching performances.

2. The VP-Viewer Architecture

VP-Viewer is a cross-platform desktop application that assists users in the exploration and assessment of VP-Trees. Internally, VP-Viewer is composed of a set of connected modules, as in Figure 1. The tool builds upon the Arboretum metric space library[2] for *(i)* reading user-provided datasets, and *(ii)* casting them into a common abstraction regardless of the domain (number, string, etc.) – Figure 1(1–3). VP-Viewer provides its own implementation of VP-Trees by employing both Arboretum data abstraction and metric distance function interface. We also employ open-source Graphviz library[3] for the graph representation of the resulting VP-Tree structure – Figure 1(4).

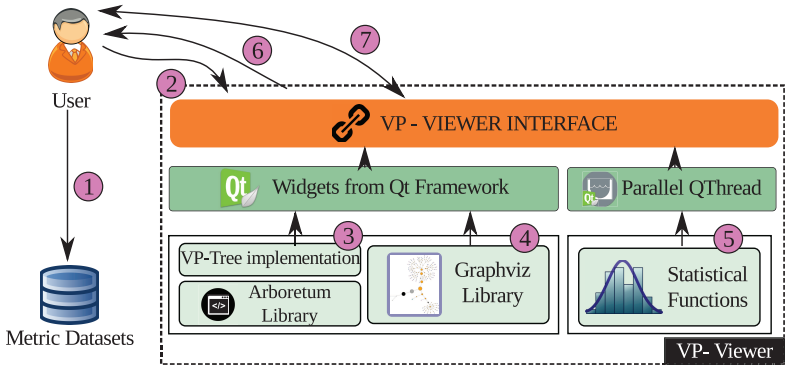


Figure 1. Overview of VP-Viewer's architecture.

Qt framework[4] is used for the incorporation of both VP-Tree implementation and Graphviz graph representation into *widgets*, which are ready-to-use GUI components. We also implemented a group of routines for the selection of pivots and for the gathering of pivot-based distance distributions – Figure 1(5). Such routines are implemented by extending the Qt thread interface so that they run along with the VP-Tree construction. Finally, we integrated all resources into a single GUI interface that supports data loading, zoom-in/out, inspection of pivot-based distance distributions within directory nodes, and submission-and-solving of range and neighborhood queries – Figure 1(6 – 7). The visual exploration of similarity searches involves the interaction between VP-Tree implementation, Graphviz library, and Statistical Functions modules.

In the first step, VP-Tree implementation executes a branching-based algorithm for the query execution and labels the evaluated tree nodes, whereas the Statistical module gathers the number of both distance calculations and inspected nodes. Next, while Graphviz renders the labeled nodes by highlighting them, the differences between the branching-based costs are juxtaposed to the brute-force solution (sequential scan) costs in as a bar percentage chart. Finally, the highlighted query path and the searching bar costs are embedded into widgets and displayed to the user.

[2]https://www.bitbucket.org/gbdi/arboretum
[3]https://www.graphviz.org/
[4]https://www.qt.io/

3. Demonstration of VP-Viewer

Here, we use VP-Viewer on three real-world datasets aiming at providing an exemplification of the scenarios covered by our tool. Table 1 details the demonstration datasets regarding cardinality, dimensionality and distance function (δ).

Table 1. Description of the datasets used in the demonstration.

Dataset	Card.	Dim.	δ	Description	Available at
CITIES	$5,507$	2	L_2	Lat-long coordinates of $5,507$ Brazilian cities.	www.ibge.gov.br
WORDS	$5,757$	–	$Edit$	All data elements are English words composed of precisely 5 characters.	www-cs-faculty.stanford.edu/~knuth
ISOLET	$6,238$	617	L_1	Features extracted from records of persons speaking each alphabet letter.	archive.ics.uci.edu/ml/datasets/isolet

Upon accessing VP-Viewer, users can visualize the main panel (Figure 2) that requests the disk location of the dataset to be indexed (1), the data cardinality (2) and dimensionality (3), the maximum allowed number of elements per leaf node (4), the distance function to be used (5), the pivot selection method (6), and, finally, the authorization parameter for node overflow (7) that may be set to "True" if balanced tree is a hard constraint, or "False" otherwise. At this point, users can request the VP-Tree construction by clicking on "generate" button and explore the resulting structure. Although VP-Viewer includes L_1, L_2, and $Edit$ functions, other distances can be plugged in through the Arboretum library interface. VP-Viewer alternatives for pivot selection are "Random", "Yianilos's Sampling", and "Convex Hull".

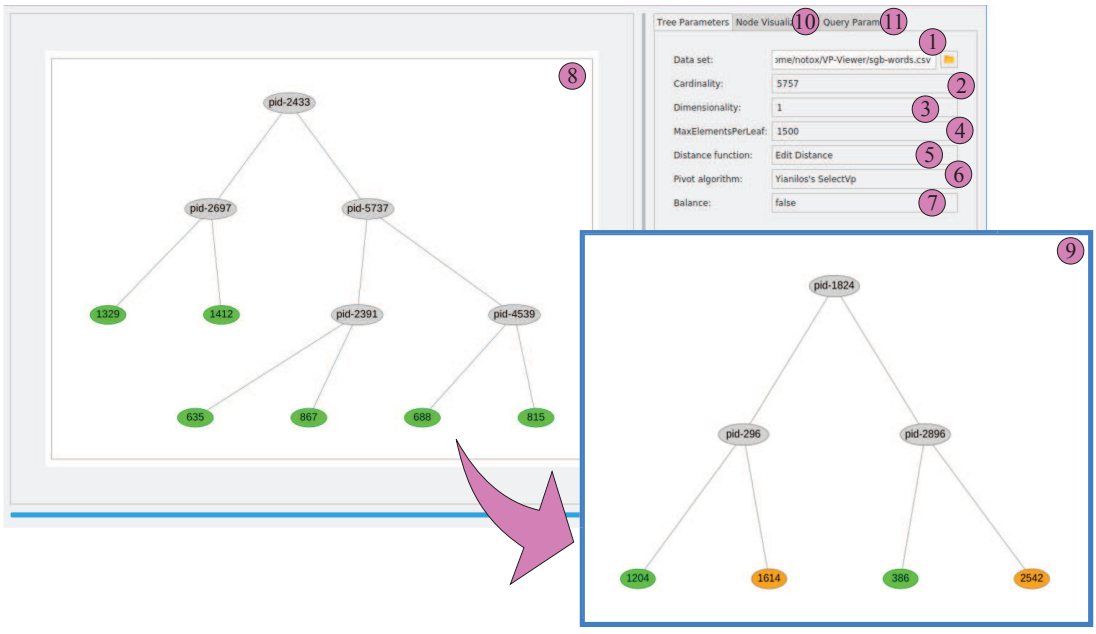


Figure 2. VP-Viewer main interface.

Figure 2 illustrates two different VP-Trees constructed for WORDS dataset with distinct constraints on node overflow. In this scenario, tied elements at the median distance to the pivots are expected because the $Edit$ distance returns discrete measures in the $[0, 5]$ interval. Figure 2(8) shows the resulting VP-Tree for a relaxed overflow constraint, which means VP-trees are allowed to be unbalanced. Green nodes indicate all leaves contain no more than the user-specified number of elements. Alternatively, a balanced tree requires the right children of directory nodes to handle overflow, as in Figure 2(9). Orange nodes indicate *(i)* leaves are currently storing more elements than specified in the user parameters, and *(ii)* additional disk paging may be necessary whenever the number of elements is related to the disk page size.

Before changing the VP-Tree parameterization for softening the tie issue, users can verify the *pivot-based distance distribution* on the VP-Viewer Node Visualization tab – Figure 2(10)). Such a resource enables the user to explore the distance distribution within each directory node. By selecting a particular node, users can visualize the distance histogram of the elements rooted by the directory node to its pivot. Figure 3(a) shows an example for the unbalanced case of WORDS dataset. Orange histogram bar is the bucket of the median and expresses the probability of ties in terms of frequencies. The probability is recursively propagated to the right-most node of the structure.

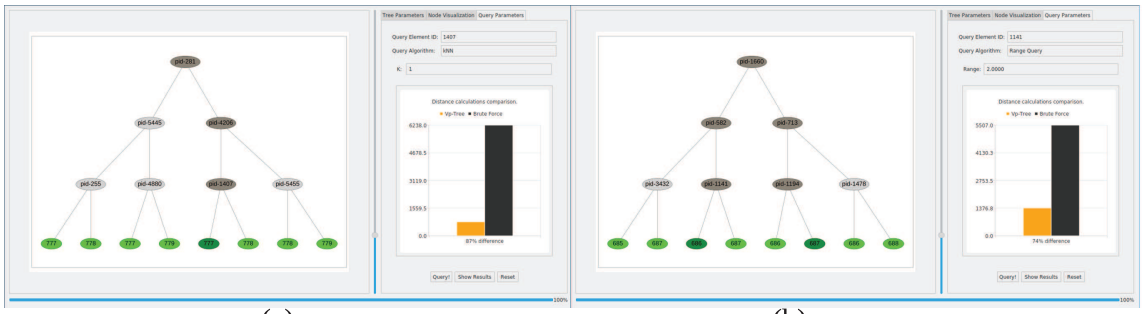
Although, choosing a new pivot selection method may diminish (or derail) the ties of median values, distance concentration around medians can be unavoidable for certain cases. For instance, Figure 3(b) shows the distance distribution for a VP-Tree directory node constructed for the ISOLET dataset with Yianilo's sampled pivots, 800 elements per leaf node, and unbalanced tree constraint setting. In this high-dimensional case, the majority of pivot-based distance distributions resemble the Standard distribution in which median behaves analogously to the mean. The switch of Yianilo's sampled pivots to either Random or Convex Hull criteria did not change the Standard distribution behavior of distances in the evaluations we performed.

(a) (b)

Figure 3. Exploration of pivot-based distance distribution. (a) WORDS dataset. (b) ISOLET dataset.

The last aspect we consider for VP-Viewer is experimental evaluation may be necessary for finding the most suitable VP-Tree partitioning. VP-Viewer enables users to request range and neighborhood queries on the Query parameters tab – Figure 2(11)). Figure 4(a) provides an example of a neighborhood query on ISOLET dataset, whereas Figure 4(b) shows an example of a range query on CITIES dataset. VP-Viewer executes both branching-based VP-Tree search and brute-force algorithms for each requested

query, and presents the performance differences between the two searching routines as a bar plot. Our tool also labels the inspected nodes that are highlighted as the *query path* in the main interface. As a result, users can visualize and interpret VP-Tree searching parameters and also compare distinct VP-Trees settings by evaluating their performances over a (sequence of) similarity query.



(a) (b)

Figure 4. Similarity searching on VP-Viewer. (a) A neighborhood query on `CITIES` dataset. (b) A range query on `ISOLET` dataset.

4. Conclusions

In this demonstration, we presented VP-Viewer, a tool that supports users in the understanding and assessment of VP-Trees. VP-Viewer is composed of a set of individual modules, which enable users the coupling and testing of several methods for distance calculation and pivot selection. We examine the capabilities of VP-Viewer and discuss usage scenarios for three real-world datasets. Our application has presented interesting resources in the handling of metric data from different perspectives.

References

Chávez, E., Navarro, G., Baeza-Yates, R., and Marroquín, J. L. (2001). Searching in metric spaces. *ACM Computing Surveys*, 33(3):273–321.

Chen, L., Gao, Y., Zheng, B., Jensen, C. S., Yang, H., and Yang, K. (2017). Pivot-based metric indexing. *PVLDB*, 10(10):1058–1069.

Chino, F. J. T., Vieira, M. R., Traina, A. J. M., and Jr., C. T. (2010). MAMView: A Framework for Visualization of Metric Trees. In *SBBD – Demo Section*, pages 1–6.

Hetland, M. L. (2009). The basic principles of metric indexing. In *Swarm Intelligence for Multi-objective Problems in Data Mining*, pages 199–232. Springer.

Li, Q., Z., H., Lei, F., L., G., Lu, M., and Mao, R. (2014). Excluded Middle Forest vs. VP-Tree: An Analytical and Empirical Comparison. In *PAIS*, pages 431–437. Springer.

Padmanabhan, D. and Deshpande, P. M. (2015). *Operators for Similarity Search - Semantics, Techniques and Usage Scenarios*. Springer.

Ruiz, G., Santoyo, F., Chávez, E., Figueroa, K., and Tellez, E. S. (2013). Extreme pivots for faster metric indexes. In *SISAP*, pages 115–126. Springer.

Yianilos, P. N. (1993). Data structures and algorithms for nearest neighbor search in general metric spaces. In *ACM-SIAM SDA*, pages 311–321. SIAM.