Fast Similarity Search in Scalar Fields using Merging Histograms

Himangshu Saikia, Hans-Peter Seidel, and Tino Weinkauf

Abstract Similarity estimation in scalar fields using level set topology has attracted a lot of attention in the recent past. Most existing techniques match parts of contour or merge trees against each other by estimating a best overlap between them. Due to their combinatorial nature, these methods can be computationally expensive or prone to instabilities. In this paper, we use an inexpensive feature descriptor to compare subtrees of merge trees against each other. It is the data histogram of the voxels encompassed by a subtree. A small modification of the merge tree computation algorithm allows for obtaining these histograms very efficiently. Furthermore, the descriptor is robust against instabilities in the merge tree. The method is useful in an interactive environment, where a user can search for all structures similar to an interactively selected one. Our method is conservative in the sense that it finds all similar structures, with the rare occurrence of some false positives. We show with several examples the effectiveness, efficiency and accuracy of our method.

1 Introduction and Related Work

Finding structural similarities in scalar fields is of prime importance when one needs to analyze repeating patterns or symmetric arrangements in the data. To this end, several methods involving feature-based analysis using the topology formed by level-sets have been proposed. The *merge tree* is one such arrangement of features which traces the connectivity evolution of sub/super-level sets in the data. It is easy to see that similar structures exhibit similar level-set arrangements and hence similar branchings in the merge tree, each such branch or subtree representing a unique structurally important region. The *contour tree* [2] is an extension of the merge tree as it contains the combined information for both sub and super-level sets.

H. Saikia (✉) • T. Weinkauf
KTH Royal Institute of Technology, Stockholm, Sweden
e-mail: saikia@kth.se; weinkauf@kth.se

H.-P. Seidel
MPI Informatik, Saarbrücken, Germany
e-mail: hpseidel@mpi-inf.mpg.de

H. Carr et al. (eds.), *Topological Methods in Data Analysis and Visualization IV*,
Mathematics and Visualization, DOI 10.1007/978-3-319-44684-4_7

The *Reeb Graph* [6] is a generalized form of the contour tree to include non-simply connected manifolds. The *Morse-Smale Complex* [7] is a segmentation of the data into regions of uniform gradient flow. These topological arrangements give rise to graph representations like the extremum graph [17] or topological spines [5].

Beketayev et al.[1] compare two merge trees by comparing all of their possible branch decompositions. This method provides an accurate ϵ-match with respect to noisy perturbations, but is not practical for interactive applications. This is because computing all possible branch decompositions has exponential complexity and a memoized solution that the authors propose still involves a higher order polynomial runtime. Precisely, $O(n^5)$ for comparing two trees of n nodes each for a given threshold ϵ.

Thomas et al. [16, 18] present methods to visualize all symmetric structures in a scalar field using the contour tree. In [16], the authors cluster the different branches in the branch decomposition tree of the contour tree to display symmetric arrangements. Using a single branch decomposition tree, however, is less robust against noise. In [18] they extract iso-surfaces using the contour tree and cluster them in a feature space. In another work,[17], the authors achieve similar results using the extremum graph.

Saikia et al. [12] perform a similarity search for any structurally significant region as given by a subtree in the merge tree by first pre-computing the similarity of all possible subtrees to all other existing subtrees. The method involves computing branch decomposition trees for every subtree and overlaying them in the best way possible. The similarity is then computed as the minimum cost of this overlay. Although the method is fast owing to a memoized algorithm to compute and compare branch decomposition trees, the result can be affected by perturbations that lead to a different order in the hierarchy of branching.

We employ the method by Saikia et al. [12] in the sense that we compare all subtrees of a merge tree against each other. However, in this paper, instead of overlaying branch decomposition trees obtained from the subtrees, we describe every subtree using a feature vector. This is given by the intensity distribution of the member voxels within a subtree region and is thus appropriately termed as a histogram. These histograms can be efficiently computed using just a small modification to the merge tree computation algorithm, exploiting the fact that the tree is created bottom-up. This augmented algorithm allows us to compute the histograms for every subtree on-the-fly, i.e., while the merge tree is being computed. The merge tree computation is done by a single sweep through a sorted (by function value) array of the voxels in the data as described in [2, 15]. Augmented contour tree algorithms have also been used before for topological simplification by computing local geometric measures such as volume in [3].

Histograms have had a long standing use in data visualization [8], automatic transfer function generation [9, 10] and various volume rendering techniques [14, 20]. Histograms have also been used in shape retrieval, where they are defined on the distances from the barycentric center of a simplicial mesh to its surface triangles [19]. This distance metric is known as a *cord*. A histogram presents itself as a simple and powerful statistical representation of the data distribution. It is also shown in

[4, 13] that the histogram has a close relationship with isosurface area. However, we do not focus on a precise calculation of the distribution within a region for the sake of simplicity and efficiency.

In the following sections, we provide some background and notation in Sect. 2, the method to compute merging histograms in Sect. 3, how similarity search is performed using the new method in Sect. 4, show a few results obtained using our method in Sect. 5 and conclude after some evaluation and discussion of our method in Sect. 6.

2 Background and Notation

Given a Morse scalar field $f : \mathbb{R}^n \rightarrow \mathbb{R}$, and any value c in the range of the function f, super-level sets L_c^+ and sub-level sets L_c^- are defined as follows

$$L_c^+ = \{\mathbf{x} | f(\mathbf{x}) \geq c\} \tag{1}$$

$$L_c^- = \{\mathbf{x} | f(\mathbf{x}) \leq c\}. \tag{2}$$

For the sake of convenience let us only talk about super-level sets from here on. Each super-level set can have one or more disconnected components. These components are born at the *maxima*, merge with other components at *saddles*, and finally merge and disappear at the *global minimum*.

We define a region $\mathscr{R}$ as the set of voxels belonging to any component *just* before it merges with another component. A is the representation of the connectivity evolution of these regions. Every birth or merge is represented as a node in this tree, and these nesting relationships are denoted by edges. Every region is represented as a non-empty subtree, the largest region being the entire merge tree. In case of super-level sets, the merge tree is often called a .

We denote a merge tree as $M = (\mathscr{N}, \mathscr{E})$ where $\mathscr{N} = \{n_1, n_2, \ldots, n_p\}$ are the nodes and $\mathscr{E} = \{e_1, e_2, \ldots, e_{p-1}\}$ are the edges of the tree. Note that a tree with p nodes has $p - 1$ edges.

Figure 1 illustrates the idea of regions and edges. Region $\mathscr{R}_A$ is given by $\{e_A\}$ and corresponds to the entire blue area, $\mathscr{R}_B$ by $\{e_B\}$—the entire red area and $\mathscr{R}_C$ by $\{e_A, e_B, e_C\}$—the entire green, red and blue areas together.

3 Merging Histograms

3.1 Histograms as Feature Descriptors

Our objective is to compare subtree regions of a merge tree. Given that these regions are similar in data value, size and also to a certain extent the geometry, a

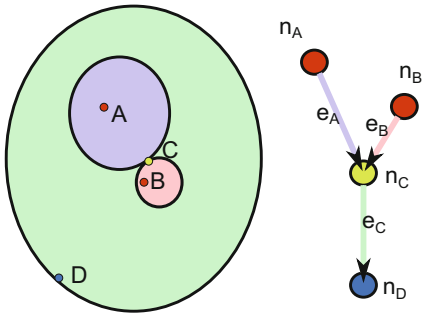

Fig. 1 A region in a scalar field and its join tree $J = (\mathcal{N}, \mathcal{E})$. Here $\mathcal{N} = \{n_A, n_B, n_C, n_D\}$ and $\mathcal{E} = \{e_{A\to C}, e_{B\to C}, e_{C\to D}\}$ or simply $\{e_A, e_B, e_C\}$. The two *red* nodes are maximas and leaf nodes, the *yellow* node is a saddle, and the *blue* node is the global minimum and the root. The edges signify the corresponding areas in the same color and the *arrows* show the unidirectional path from every maximum to the global minimum

of the distribution of the member voxels in the data range is a good measure for comparison. This is because a histogram roughly encapsulates the surface area of a region at various iso-levels. This gives us a good measure of the distribution of intensity in the region.

Given a scalar function f, and a subtree region $\mathscr{R}$, the histogram is computed by binning all voxels (or pixels for 2D) in this region into a number of bins by their function value.

Figure 2 shows an illustration of this feature descriptor. Two structurally different subtree regions of a dataset can be distinguished from each other by comparing their individual histograms.

3.2 Computation

The good thing about constructing these histograms is that they can be computed incrementally during the construction of the merge tree itself. The part *not* marked in red in Algorithm 1 shows the classic merge tree computation. Let us look closely at the three different cases encountered while sweeping through the sorted data, and how the histograms have to be modified during this process.

Case 1: extremum This is the *if* clause in line 10 of Algorithm 1. In this case a new node n_i is added to the set of nodes and a new component in the union-find c_i starts. The initial histogram is set to zero for all bins, i.e., $\mathbf{h}_i = \mathbf{0} = [0, \ldots, 0]$.

Case 2: regular voxel This is the *else-if* clause in line 15 of Algorithm 1. Here the current voxel is added to the component it belongs to. The appropriate histogram bin has to be incremented for the corresponding component. Let us assume the histogram bin that this point falls into is b. Then, $h_{i,b} \leftarrow h_{i,b} + 1$, where c_i is the component to which this voxel belongs to.

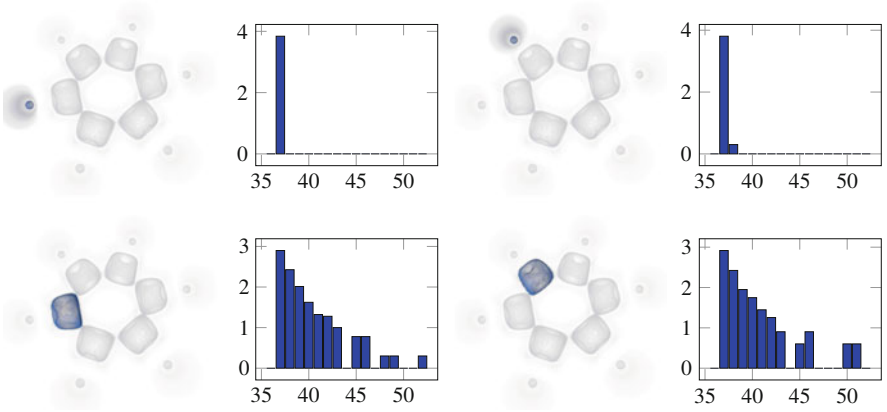

Fig. 2 Four different subtree regions in the benzene dataset along with their corresponding histogram feature vectors. The *x*-axis in the plots shows the corresponding histogram bins and the *y*-axis shows the log scaled values of the total number of voxels in each bin. A total of 100 bins were used in all cases. The *x*-range with only non-zero values is shown. As can be seen, the regions corresponding to the hydrogen atoms have similar histograms as do the regions corresponding to the carbon atoms

Case 3: saddle This is the *else* clause in line 19 of Algorithm 1. This is the case when a point is in contact with two or more edges. All edges pertaining to every component are added to the set of edges. A new component is constructed and a new node corresponding to this voxel is added to the set of nodes. The histogram corresponding to this component has to be initialized using the sum of all histograms pertaining to all of the components in set C_G. Thus, for a saddle node n_i and number of bins B we have $\mathbf{h}_i = [h_{i,1}, h_{i,2}, \ldots, h_{i,B}]$, where each bin is given by

$$h_{i,b} = \sum_{c_j \in C_G} h_{j,b} \tag{3}$$

The only modification that needs to be done to the algorithm is to incorporate these three cases. Thus, when the merge tree is computed, the feature vectors are also pre-computed as a result.

4 Similarity Search Using Merging Histograms

After the computation is performed using Algorithm 1, in addition to the merge tree, we also obtain the feature descriptors for every subtree region in the form of a histogram of values. Using these feature descriptors we compare every subtree region with each of the others and store the results in a distance matrix. Later we

Data: The scalar field f, with vertices $\mathbf{x}_1, \ldots, \mathbf{x}_m$ in sorted order.
Result: If $f(\mathbf{x}_i) \geq f(\mathbf{x}_j)$ for $i < j$ then Join Tree $J = (\mathcal{N}, \mathcal{E})$. If $f(\mathbf{x}_i) \leq f(\mathbf{x}_j)$ for $i < j$ then Split Tree $S = (\mathcal{N}, \mathcal{E})$. Also the set $H = \{h_1, \ldots, h_i, \ldots\}$ containing all histograms corresponding to every edge $e_i \in \mathcal{E}$ and subtree region $\mathscr{R}_i$.

begin
  $\mathcal{N} := \emptyset$, $\mathcal{E} := \emptyset$, $H := \emptyset$, UnionFind U
  for $i \leftarrow 1$ **to** $m - 1$ **do**
    Set of neighbors of $\mathbf{x}_i : G = \{\mathbf{g}_1, \ldots, \mathbf{g}_p\}$
    Set of components containing $G : C_G := \emptyset$
    for $j \leftarrow 1$ **to** p **do**
      $C_G \leftarrow C_G \bigcup findComponent_U(\mathbf{g}_j)$
    end
    $b := bin(f(\mathbf{x}_i))$ // Finding the bin value.
    if $|C_G| = 0$ **then**
      $c_v \leftarrow createNewComponent_U(\mathbf{x}_i)$
      $\mathcal{N} \leftarrow \mathcal{N} \bigcup \{n_i\}$
      $\mathbf{h}_v = [0, \ldots, 0]$ // Initializing.
      $h_{v,b} \leftarrow h_{v,b} + 1$
    else if $|C_G| = 1$ **then**
      $C_G = \{c_v\}$
      $addMemberToComponent_U(c_v, \mathbf{x}_i)$
      $h_{v,b} \leftarrow h_{v,b} + 1$ // Incrementing the bin.
    else
      $C_G = \{c_a, \ldots, c_k\}$
      $\mathcal{N} \leftarrow \mathcal{N} \bigcup \{n_i\}$
      $\mathcal{E} \leftarrow \mathcal{E} \bigcup \{e_{a \rightarrow i}, \ldots, e_{k \rightarrow i}\}$
      $c_v \leftarrow createNewComponent_U(\mathbf{x}_i)$
      $H \leftarrow H \bigcup \{\mathbf{h}_a, \ldots, \mathbf{h}_k\}$ // Adding to output.
      $\mathbf{h}_v = \mathbf{h}_a + \ldots + \mathbf{h}_k$ // Merging.
    end
  end
  $\mathcal{N} \leftarrow \mathcal{N} \bigcup \{n_m\}$
end

Algorithm 1: An augmented version of the classic merge tree algorithm to account for merging histograms. The augmented parts are shown in red.

can reference any of these subtree regions by means of interactively selecting it, and querying for its best matches in the distance matrix.

4.1 Distance Measure

The distance measure between two histograms should be as discriminative as possible. To compare two histograms we use the L_2-norm of the log-scaled bin values. Log scaling helps to smooth the histograms a little bit and make the comparison function slightly robust to noise. Thus, the distance d between two subtree regions $\mathscr{R}_i$ and $\mathscr{R}_j$ can be given as

$$d(\mathscr{R}_i, \mathscr{R}_j) = \sum_{b \in [1,B]} ||g(h_{i,b}) - g(h_{j,b})|| \tag{4}$$

where g is given by

$$g(x) = \begin{cases} 0, & \text{if } x = 0 \\ \log x, & \text{otherwise.} \end{cases} \tag{5}$$

4.2 Querying the Distance Matrix

A distance matrix is then constructed using the distance values for every pair of subtree regions computed using Eq. 4. Any row and column in this matrix refers to a subtree region. As has been seen before in Sect. 2 the number of subtree regions is equal to the number of edges in the merge tree. This shows that the time complexity of computing a distance matrix and its size are dependent only on the merge tree and not on the size of the data.

In an interactive setting, a user picks any voxel in the dataset. Since every voxel is contained within an edge, the corresponding edge can be queried for. And since every edge corresponds to a unique subtree, we can immediately identify which subtree region is selected by the user. Once this region is known, similar regions to it can be queried simply by looking for the smallest values in the corresponding row of the distance matrix. A distance threshold slider also allows the user to increase or decrease the threshold and show correspondingly more or lesser close matches.

5 Results

Now we show a few results obtained using our method. Figure 3 shows a volume rendering of the Benzene data set and two different search regions. As can be seen, the sixfold symmetry in the molecule is evident from the closest matches to the selected subtree region. Figure 4 shows the EMDB-1603 data set and a few different search regions. The dataset had nearly 38,000 edges in its merge tree, which were then simplified to around 900 edges by eliminating low persistent edges (below 2%). The particle exhibits a ninefold symmetry as seen in all three selections and their closest matches. As can be observed, interesting structures which could otherwise not be seen clearly are apparent when singled out. Figures 5 and 6 show two other protein datasets alongwith some interesting self-similar structures. Figure 7 shows another EMDB dataset with a helical structure. This is identified in the selected subtree region and its closest matches. Figure 8 shows another complicated dataset where the symmetric arrangements are revealed during exploration.

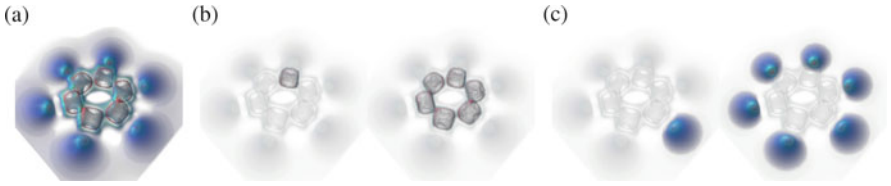

Fig. 3 Scalar field depicting the potential around a Benzene molecule. Two different sixfold symmetry regions are seen. (**a**) Full volume rendering. (**b**) First selected region and its closest matches. (**c**) Second selected region and its closest matches

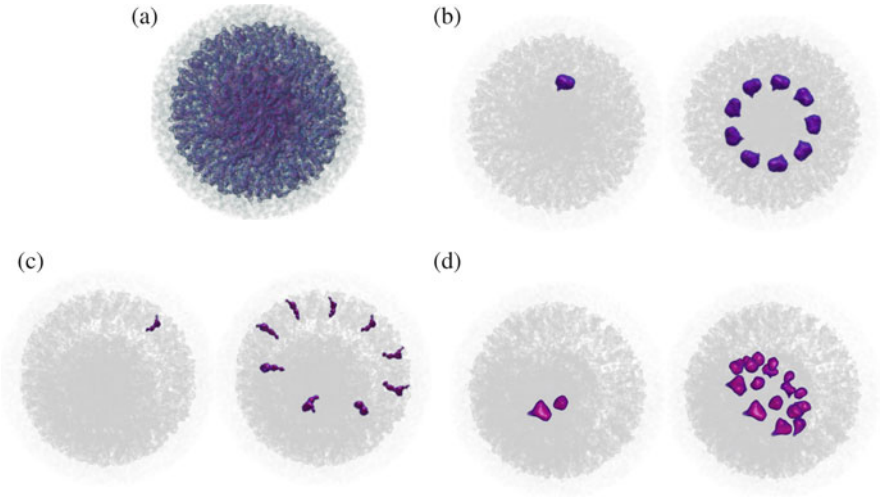

Fig. 4 EMDB-1603. A cryo-electron microscopy reconstruction of a recombinant active ribonucleoprotein particle of influenza virus. The ninefold symmetry is apparent in the matchings shown. Different transfer functions are used for the three different selections for better visibility. (**a**) Full volume rendering. (**b**) First selection and its corresponding best matches. (**c**) Second selection visualized at a slightly different angle and its best matches. (**d**) Third selection and its best matches

All EMDB data sets are obtained from the Protein Data Bank Japan (pdbj.org) online archive. The results are all rendered using the Voreen volume rendering engine (voreen.uni-muenster.de).

6 Discussion

6.1 Runtime Comparison with Tree Overlay Methods

Computing feature vectors on the fly while computing the merge tree itself leads to a more efficient implementation as opposed to performing the two steps sequentially

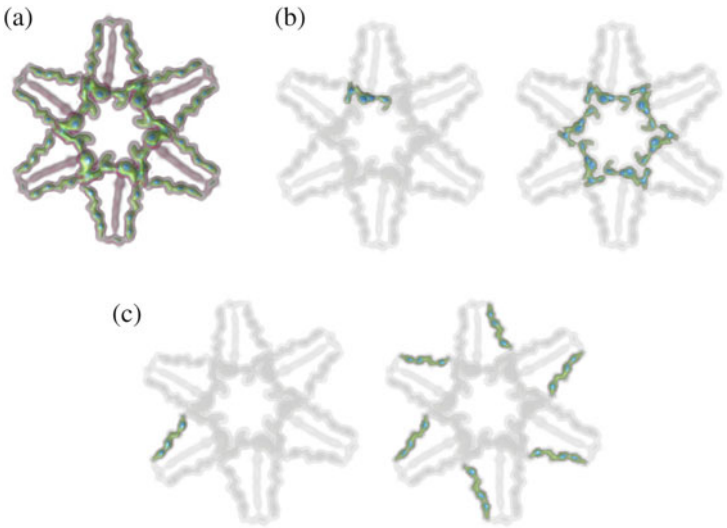

Fig. 5 EMDB-1201. The myosin V inhibited state obtained by cryo-electron tomography. There exists a sixfold symmetry. (**a**) Full volume rendering. (**b**) first selection and its best matches. (**c**)Second selection and its best matches

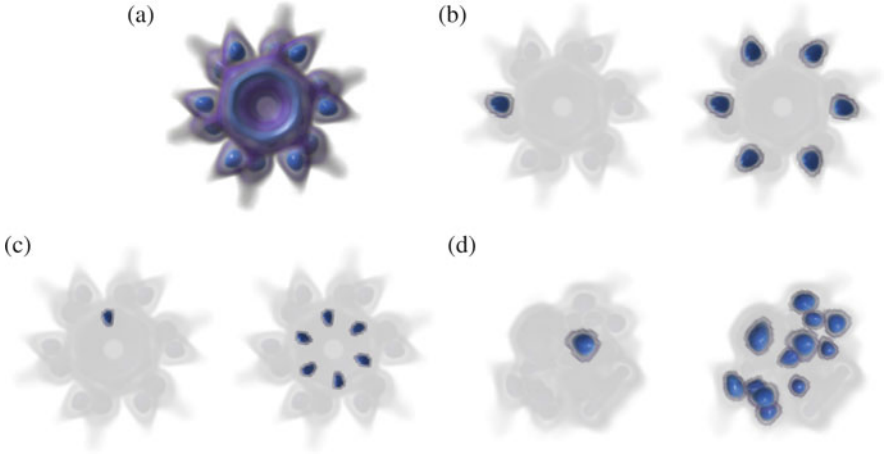

Fig. 6 EMDB-1706. Cryo-electron reconstruction of Lactococcal phage p2 baseplate. There exists a sixfold symmetry. (**a**) Full volume rendering. (**b**) First selection and its closest matches. (**c**) Second selection and its matches. (**d**) Selection in (**b**) from a different angle. A few more best matches are shown to reveal a duplicate sixfold symmetric pattern

as in the Extended Branch Decomposition Graph method in [12]. There is almost no overhead of running the augmented merge tree computation algorithm as opposed to the classic algorithm, as can be seen in Table 1.

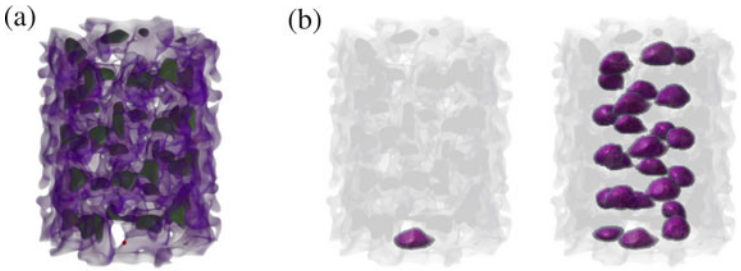

Fig. 7 EMDB-2400. MuB is an AAA+ ATPase that forms helical filaments to control target selection for DNA transposition. (**a**) Full volume rendering. (**b**) A selection and its closest matches. The helical structure is apparent

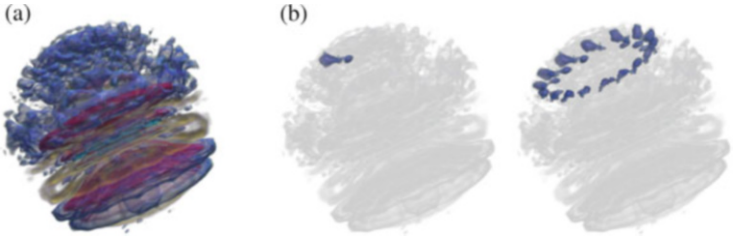

Fig. 8 EMDB-5300. Structural Diversity of Bacterial Flagellar Motors: Campylobacter jejuni. (**a**) Full volume rendering. (**b**) A selection and its closest matches.

For searching a subtree region, a distance matrix has to be constructed and this can be achieved in $O(n^2B)$ time as opposed to $O((n \log n)^2)$ in [12]. Note that both methods, the one in this paper and the method in [12], compare all subtrees to all others and then allow for similarity searching interactively in real-time. Comparing two subtree regions using our method is very fast as it just compares the two individual histograms in $O(B)$ time. Hence our method is orders of magnitude faster. This can be seen in Table 2.

6.2 Robustness

Merging histograms perform well under small perturbations in the data. This can be immediately observed from the fact that there is no ordering of hierarchy in this representation unlike a branch decomposition tree. Since every ROI is defined only by its complete underlying structure and not on the precise order in which its containing iso-contours evolved, this method is immune to slight changes in the merge tree due to noise (see Fig. 9).

For more complicated branchings however, the histogram cannot accurately represent the branching hierarchy. This means that two trees with extremely

Table 1 Merge tree computation times (in milliseconds) for various data sets—without histograms and with histograms of bin sizes 10 and 100

			Merge tree computation time in *ms*		
				Augmented algorithm 1	
Dataset	Vertices	Edges in join tree	Classic algorithm	With 10 bins (% increase)	With 100 bins (% increase)
Benzene	101^3	23	678	752 (10.9)	700 (3.2)
Neghip	64^3	252	148	157 (6.1)	156 (5.1)
EMDB-1603	160^3	38,671	4934	5470 (10.9)	6336 (28.4)
EMDB-1201	180^3	41,169	1589	1873 (17.9)	2531 (59.3)
EMDB-1706	130^3	155	935	1112 (18.9)	1141 (22.0)
EMDB-2400	128^3	2003	1973	2189 (11.0)	2083 (5.6)
EMDB-5300	60^3	3685	191	210 (10.0)	237 (24.1)

As can be seen there is only a slight overhead to adding histogram information to the computation phase. All operations were performed on a machine with a 2.66 GHz Intel Xeon processor and 12 GB main memory

Table 2 Total feature computation and comparison times (in milliseconds) for various data sets—using the eBDG method in [12] and the histogram method

			Feature computation and comparison time in *ms*	
Dataset	Vertices	Edges in join tree (after simplification)	eBDG approach	Our approach
Benzene	101^3	23	506	489
Neghip	64^3	252	268	142
EMDB-1603	160^3	38,671 (910)	54,487	6572
EMDB-1201	180^3	41,169 (198)	13,955	3749
EMDB-1706	130^3	155	672	702
EMDB-2400	128^3	2003	764,629	3632
EMDB-5300	60^3	3685	6,805,101	7081

As can be seen, with more number of edges the eBDG method requires far more time to compare all features against each other than the histogram method. All operations were performed on a machine with a 2.3 GHz Intel i7 processor and 16 GB main memory

different branchings (and hence underlying topological structure) might end up having very similar histograms (see Fig. 10). This might result in false negatives. However, note that similar subtrees have similar histograms, i.e., the method finds similar structures independent of their complexity.

6.3 Histogram Resolution and Comparison

The bin number reflects the resolution at which we sample our data. The higher the bin number, the greater the resolution. In our examples we observe that a bin number

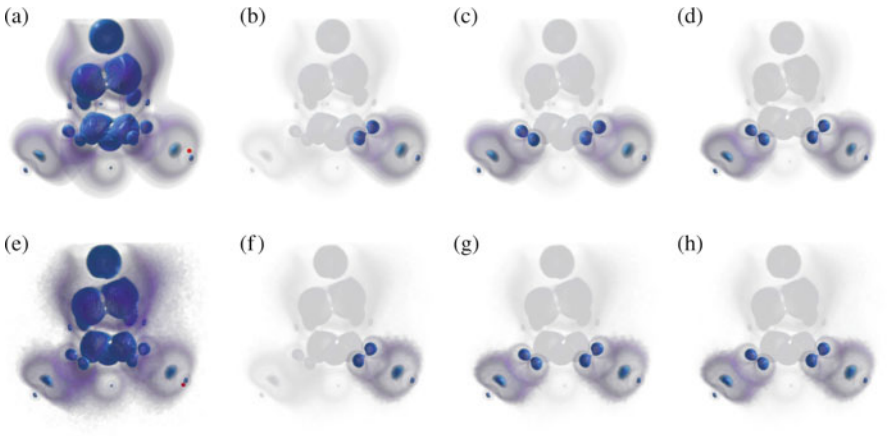

Fig. 9 Neghip dataset. The results are in accordance with the eBDG method described in [12]. The matching results are stable even in presence of noise. (**a**) Full volume rendering. (**b**) A selection. (**c**) Similar structures to (**b**) using merging histograms. (**d**) Similar structures to (**b**) using eBDG. (**e**) 5% random noise added to (**a**). (**f**) Same selection. (**g**) Similar structures to (**f**) using merging histogram. (**h**) Similar structures to (**f**) using eBDG.

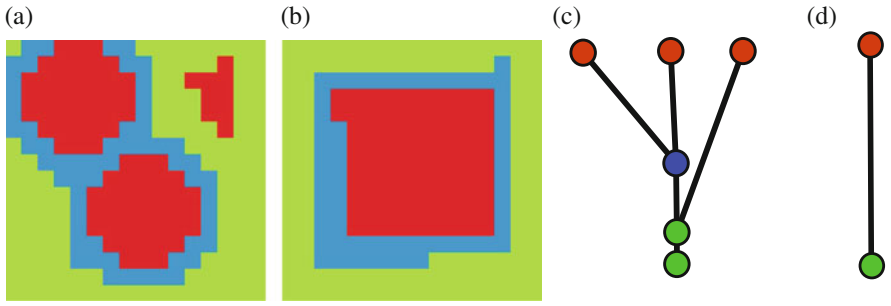

Fig. 10 An example of false positives in the histogram approach. Two simple yet topologically different datasets were designed to have the exactly same pixel distribution (*Red* = 83, *Blue* = 57 and *Green* = 116) and hence identical histograms. (**c**) and (**d**) show their corresponding join trees. As can be seen, a tree overlay method will be able to find the differences between these datasets but the histogram method will not. (**a**) Simple dataset A. (**b**) Simple dataset B. (**c**) Join tree for A. (**d**) Join tree for B

of 100 is a good estimate for most cases, and using a higher number does not alter the results much. In our implementation, we do not impose a necessary condition on the bin number. Sometimes it may so happen that two non-overlapping iso-valued regions are assigned to the same bin due to the resolution being too low. This issue can be addressed in future work. Another alternative binning strategy would be to assign more bins to more dense parts of the dataset and vice versa, thereby choosing a non-linear binning mechanism based on the intensity distribution of the entire dataset.

The current implementation considers applications which require finding similar regions at *nearly the same* iso-range. For applications which require finding similar structures at different iso-ranges, the method can be modified to finding the best overlap between histograms which minimizes the distance between them. This can be achieved by using standard dynamic programming techniques such as the Earth Mover's Distance [11].

7 Conclusion

Interactive similarity search in scalar fields using merge trees present a lot of useful possibilities like real-time exploration of the data, and reveal interesting patterns in volume renderings that are otherwise hard to see. Finding such patterns involve the general idea of finding similar subtrees to the corresponding subtree of the underlying pattern. Instead of trying to compare these subtrees, we focused on comparing feature descriptors of the subtree regions themselves and showed that a faster method can be used to achieve similar results.

We presented merging histograms—a feature descriptor defined for all subtrees of a merge tree, which was shown to be easily computed on-the-fly as part of the merge tree computation step. We presented a few simple modifications to the merge tree computation algorithm to achieve this. The comparison was shown to be quite discriminative and robust to small perturbations.

Computing these self-similarities between all pairs of subtree regions very quickly, provides for a rich interactive possibility.

A direction for future work could be to display self-similar structures at various iso-levels automatically without any user intervention.

References

1. Beketayev, K., Yeliussizov, D., Morozov, D., Weber, G.H., Hamann, B.: Measuring the distance between merge trees. In: Topological Methods in Data Analysis and Visualization III, pp. 151–165. Springer, Berlin (2014)
2. Carr, H., Snoeyink, J., Axen, U.: Computing contour trees in all dimensions. In: Proceedings ACM-SIAM Symposium on Discrete Algorithms, SODA '00, pp. 918–926. Society for Industrial and Applied Mathematics, Philadelphia (2000)
3. Carr, H., Snoeyink, J., van de Panne, M.: Simplifying flexible isosurfaces using local geometric measures. In: Proceedings IEEE Visualization, pp. 497–504. IEEE Computer Society (2004)
4. Carr, H., Brian, D., Brian, D.: On histograms and isosurface statistics. IEEE Trans. Vis. Comput. Graph. **12**(5), 1259–1266 (2006)
5. Correa, C., Lindstrom, P., Bremer, P.T.: Topological spines: a structure-preserving visual representation of scalar fields. IEEE Trans. Vis. Comput. Graph. **17**(12), 1842–1851 (2011)
6. Doraiswamy, H., Natarajan, V.: Efficient algorithms for computing Reeb graphs. Comput. Geom. **42**(6–7), 606–616 (2009)

7. Günther, D., Reininghaus, J., Seidel, H.P., Weinkauf, T.: Notes on the simplification of the Morse-Smale complex. In: Topological Methods in Data Analysis and Visualization III, pp. 135–150. Springer, Berlin (2014)
8. Ioannidis, Y.: The history of histograms (abridged). In: Proceedings International Conference on Very Large Data Bases (VLDB), vol. 29, pp. 19–30. VLDB Endowment (2003)
9. Kindlmann, G.L.: Semi-automatic generation of transfer functions for direct volume rendering. In: Proceedings IEEE Symposium on Volume Visualization, pp. 79–86 (1998)
10. Lundström, C., Ljung, P., Ynnerman, A.: Local histograms for design of transfer functions in direct volume rendering. IEEE Trans. Vis. Comput. Graph. **12**(6), 1570–1579 (2006)
11. Rubner, Y., Tomasi, C., Guibas, L.J.: A metric for distributions with applications to image databases. In: Proceedings International Conference on Computer Vision, pp. 59–66. IEEE (1998)
12. Saikia, H., Seidel, H.P., Weinkauf, T.: Extended branch decomposition graphs: structural comparison of scalar data. Comput. Graph. Forum **33**(3), 41–50 (2014)
13. Scheidegger, C.E., Schreiner, J.M., Duffy, B., Carr, H., Silva, C.T.: Revisiting histograms and isosurface statistics. IEEE Trans. Vis. Comput. Graph. **14**(6), 1659–1666 (2008)
14. Sereda, P., Vilanova Bartroli, A., Serlie, I.W.O., Gerritsen, F.A.: Visualization of boundaries in volumetric data sets using LH histograms. IEEE Trans. Vis. Comput. Graph. **12**, 208–218 (2006)
15. Tarasov, S.P., Vyalyi, M.N.: Construction of contour trees in 3D in O(n log n) steps. In: Proceedings Annual Symposium on Computational Geometry, SCG, pp. 68–75. ACM, New York (1998)
16. Thomas, D.M., Natarajan, V.: Symmetry in scalar field topology. IEEE Trans. Vis. Comput. Graph. **17**(12), 2035–2044 (2011)
17. Thomas, D.M., Natarajan, V.: Detecting symmetry in scalar fields using augmented extremum graphs. IEEE Trans. Vis. Comput. Graph. **19**(12), 2663–2672 (2013)
18. Thomas, D., Natarajan, V.: Multiscale symmetry detection in scalar fields by clustering contours. IEEE Trans. Vis. Comput. Graph. **20**(12), 2427–2436 (2014)
19. Tung, T., Schmitt, F.: Augmented Reeb graphs for content-based retrieval of 3D mesh models. In: Proceedings Shape Modeling Applications, pp. 157–166. IEEE (2004)
20. Younesy, H., Möller, T., Carr, H.: Visualization of time-varying volumetric data using differential time-histogram table. In: Proceedings International Workshop on Volume Graphics, pp. 21–224. IEEE (2005)