Eurographics Symposium on Parallel Graphics and Visualization (2020)
S. Frey, J. Huang, F. Sadlo (Editors)

Alternative Parameters for On-The-Fly Simplification of MergeTrees

K. Werner[1] and C. Garth[1]

[1]Technische Universität Kaiserslautern, Germany

Abstract

Topological simplification of merge trees requires a user specified persistence threshold. As this threshold is based on prior domain knowledge and has an unpredictable relation to output size, its use faces challenges in large-data situations like online, distributed or out-of-core scenarios. We propose two alternative parameters, a targeted percentile size reduction and a total output size limit, to increase flexibility in those scenarios.

CCS Concepts

• ***Human-centered computing*** → ***Scientific visualization;*** • ***Mathematics of computing*** → *Topology;* • ***Software and its engineering*** → *Ultra-large-scale systems;*

1. Introduction

The contour tree [vKvOB*97] is one of the most fundamental and widely used structures for topology driven analysis, for example mesh cleaning [WHDS04], noise removal [RWS*17] or allowing visualization for complex data [BG15]. Contour trees can be constructed from the two merge trees of the domain [CSA03], and some applications can be performed on the merge trees themselves.

Current merge tree construction algorithms are often output-sensitive [GFJT17, CLLR05] or tailored for memory-constrained [MSD*12, CWSA16] or distributed [MW14, LPG*14] settings and tend to have a large memory footprint [AN15]. Because of this, computing the complete merge tree on unsimplified data faces challenges in large-data applications. To overcome these challenges, isolated efforts to perform simplification in a single-pass, online and concurrently during construction have arisen in large data-focused topological analysis [PSBM07]. This on-the-fly simplification proved to be beneficial, as it was able to avoid multiple iterations and reduce output size without ever storing the full output.

For merge tree simplification a user specified persistence threshold ε is required. Merge tree arcs are tested bottom up and arcs with less persistence than ε are removed from the output. However it is not possible to estimate resulting output sizes or relative size reduction of the simplification based on ε, see figure 1. Additionally, ε has to be expressed within the scale of the data values and thus a satisfactory choice for ε requires domain knowledge, which often requires multiple costly iterations of analysis to gain. This is in stark contrast to the above mentioned benefits of on-the-fly simplification and hurts applicability in large-data scenarios.

We propose the use of two alternative user specified parameters for simplification, to allow a more direct control over the simplified output size and increase flexibility in large-data applications. First, simplification can be controlled by a total desired output size N, pruning all but the N most persistent merge tree arcs. Second, simplification can be controlled by a desired relative size reduction p, pruning all but the p percent most persistent merge tree arcs. We introduce algorithms to guide simplification based on N or p. This requires adaptive estimation techniques in online applications of p.

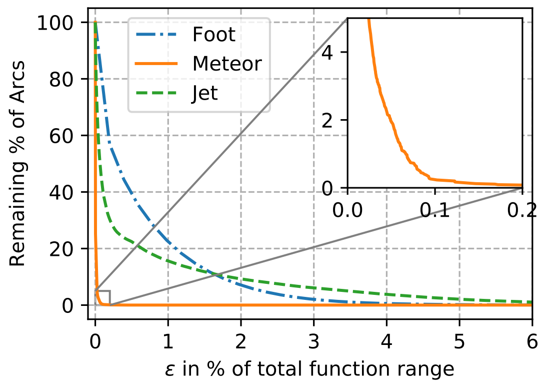

Figure 1: *The percentage of remaining arcs after simplification decreases with larger choices for the persistence threshold ε. The relation is highly non-linear and depends strongly on the specific data, making it difficult for the user to control simplification results by choosing ε. Data was acquired from join trees of data sets presented in section 5.*

DOI: 10.2312/pgv.20201077

2. Related Work

A well-defined approach to topological simplification of scalar functions is based on persistence pairs [ELZ02], minimum-saddle or maximum-saddle pairs, that can be visualized in the so called persistence diagram [CSEH07]. From these pairs the ε-*simplification* [EMP06] was derived. It allows for the simplification of a scalar function on manifolds, by cancelling persistence pairs, i.e. eliminating exactly all persistence pairs with persistence less than ε, while guaranteeing a maximal function value change of ε. However under this guarantee, up to all persistence pairs with persistence less than 2ε can be eliminated [BLW12].

A combinatorical approach to scalar function simplification is to maintain user-specified minima and maxima of the function and eliminate the remainder through perturbation of function values [TP12]. A hierarchical representation called branch decomposition [PCMS05] of merge trees accumulates consecutive tree arcs into branches according to persistence. Cancelling a persistence pair of the scalar function results in the same tree as discarding the corresponding leaf-edge of the merge tree branch decomposition. Note however, that this does not hold true for contour trees [HC19].

A further merge tree simplification technique is based on identifying Y-shapes in the tree and comparing the smaller persistence of the involved non-leaf arcs to a threshold [TTF04]. A similar approach that is applicable for multi-saddles instead works on a per-arc basis [CSvdP10]. Both techniques achieve results equivalent to persistence pair cancelling like in the branch decomposition, when applied to a merge tree. Again, when applied to contour trees, results deviate both from branch decomposition and ε-simplification.

For the tree based simplification techniques different alternative measures, including surface area or volume of represented level-sets, can be used as a weight for edges to be compared to a threshold for simplification [CSvdP04, ZT11]. Although we use topological persistence in this paper, any arc-weight that is compared to a fixed threshold for simplification purposes can be used in conjunction with our methods.

To apply simplification during construction, instead of outputting the entire tree MT at the end of the construction, one can consider the output as a concurrent stream S of arcs, with their associated persistence as weight w. The merge tree MT is now the full aggregation of all arcs in this stream S. On-the-fly simplification is the process of pruning arcs from S according to w and ε as they arrive and add only unpruned arcs to MT. First appearances of this idea are found in online reeb graph construction [PSBM07].

Contemporary contour tree construction algorithms follow [CSA03] in first constructing the merge trees with an ordered sweep through the data and then combining them to create the contour tree. This naturally sequential approach has received a divide-and-conquer based parallelization, both with regular spatial subdivision [VP03,LPG*14] and subdivision along isosurfaces [GFJV16].

Contour tree construction without a globally sorted progression was introduced by output-sensitive parallel algorithms based on monotone paths between critical points [CLLR05,MSD*12,RS14]. Such approaches are suitable for massively data-parallel computation [CWSA16]. Recently, construction methods were introduced, that are able to identify merge tree arcs independently (with an exception of inner arcs depending on their children) in a task-parallel [GFJT17] and ad-hoc network [SZG*08] setting.

The proposed parameters can be used in conjunction with any of the above simplification and construction methods, as N and p will be used to find an appropriate threshold ε, that can be used as before. However, to be applicable to on-the-fly simplification, simplification or construction methods have to fulfill some conditions described in section 4. In section , we describe the proposed alternative parameters and algorithms to calculate or estimate an ε that realizes their application.

3. Alternative Parameters for Simplification

3.1. Constrained Branch Count N

One setting of interest is to set ε such that a given number N of branches remain after simplification. This is for example necessary if work is performed in a memory-constrained environment.

For conventional post-processing, finding an ε so that N branches have a larger weight than ε is easily done by an inverse rank query. However for on-the-fly simplification the decision to prune or keep a branch must be done before weights for all branches are known.

To this end, we propose to use a priority queue [Pug89] Q. For each arc in the stream S, the arc is enqueued in Q, with its weight as priority. If more than N elements have been enqueued, Q is immediately dequeued from, resulting in the arc with $N+1$ largest weight so far. This arc cannot be among the N largest weighted arcs overall and thus has a weight smaller than the hypothetical ε we search. Thus it can be immediately pruned.

Substituting ε for N as a decision basis for topological simplification allows to maintain maximum detail in a memory constrained setting. Subsequent simplification may then be performed within main memory outside of the large-data application.

3.2. Percentile Size Reduction by p

Persistence is expressed in terms of the scalar function values and thus requires knowledge about the scale of that function to interpret and use. Reducing the output size by a given percentage however does not rely on that knowledge. Thus another interesting problem is to choose ε, such that a given percentage p of branches remain after simplification.

For on-the-fly simplification based on a given p the problem is the following: For each arc in the stream S, calculate the percentile rank of the arc and prune it if it is smaller than p. Of course precise ranks are only known a posteriori, thus an estimation based on the streamed arcs so far has to be made.

We next turn to the problem of estimating the percentile rank of an arc from all previously streamed arcs with minimal memory overhead.

3.2.1. Quantile Summary

To this end we propose the use of a biased quantile (bq-)summary [CKMS06]. When restricting the range of possible weights for the arcs, we can store those arcs as leafs in a binary tree over this range. The bq-summary instead stores a subset of nodes of this tree with associated counts, to approximate the distribution of stored leafs. By maintaining a set of invariants upon insertion, and running an amortized compression of the tree, sublinear memory consumption, insertion and estimation runtimes are achieved.

The data structure as proposed by the authors depends on a discretized range restriction of possible weights, containing U different weights. Insertion of a weight to the summary has an amortized cost of $\mathcal{O}(\log\log U)$. Rank estimation technically has the same cost, however as we will estimate the rank of every inserted weight (thus for every arc) we can slightly adapt the insertion method to yield the rank estimation as a byproduct. Memory consumption of the data structure is $\mathcal{O}(\frac{\log U}{\varepsilon}\log(\varepsilon N))$, with N the overall size of the stream and ε the maximal relative error of the estimation.

Since we do not want to rely on previous domain knowledge, we choose the range restriction to contain the whole range representable by floating point variables. Overall estimation accuracy achieved on real world data sets and runtime penalties paid for maintaining the data structure will be shown in Section 5.

3.2.2. Statistical Estimation

Online rank estimation inevitably suffers from irregular distribution of weights within the stream. With this, ranks of arcs within the history of the stream upon their arrival will deviate from the ranks of those arcs in the overall data. In other words, if a lot of short arcs are finalized first, the resulting summary data structure will rank short arcs too high.

To alleviate this problem one can try to introduce a measure of uncertainty into the summary, that represents size and variance of the observed part of the stream. If uncertainty is high, rank estimation can be adjusted to, for example, prune less arcs.

The simplest approach to statistical online rank estimation, is to assume arc weight distribution to be Gaussian. If arc weights are distributed according to a normal distribution, we can estimate this distribution by interpreting the previously observed stream as a sample. Small sample sizes will result in pessimistically estimated distributions, that will prune less arcs. Consider the following update mechanism for each arc a:

1. Filter the weight of a with Tukey's Fences [Tuk77] to reduce impact of outliers. Small outliers are pruned, large outliers are stored to the output.
2. If a is not an outlier, increase the sample size n by 1 and update overall empiric mean and empiric variance of the sample with a numerically stabilized Steiner Translation [CGL83].
3. From the sample, calculate a Students t and χ squared distribution with $n-1$ degrees of freedom. For a given significance ε find the smallest explainable mean and variance.
4. These mean and variance correspond to the most pessimistic normal distribution that can explain the sample with significance ε. Evaluate the upper p percent quantile of this distribution and compare it to the value of a.

With this, after each arrival of an arc a, we calculate a confidence interval around the empiric mean, in which the true mean of the arc weights lies with ε percent certainty. To be most pessimistic and thus try to keep arcs instead of pruning them when in doubt, we choose the smallest mean in this range. Similarly we calculate an interval around the empiric variance, in which the true variance lies with ε percent certainty and choose the smallest variance. From this mean and variance we derive a pessimistic normal distribution. The upper p percent quantile of this distribution is a value, below which most probably at most $1-p$ percent of the actual arc weights lie. Thus if the weight of a is below that value it can most probably be pruned.

4. Application

To perform benchmarks for performance impact and estimation precision of the techniques above, an implementation of contour tree construction has been extended with on-the-fly simplification. The used algorithm, see Algorithm 1, is a task-parallel sweeping method that identifies arcs individually; it is a variant of the algorithm used in the TTK [TFL*17, GFJT17]. Growing monotone regions around local extrema allows identification of their saddles, once all child arcs have identified a given saddle, all but the largest are tested and either pruned or added to the output. Simplification was done symbolically without augmentation, thus the pruned arcs were simply removed from the output, potentially resulting in a vertex reduction of the saddle.

Algorithm 1 TASK-PARALLEL ON-THE-FLY SIMPLIFIED MERGE TREES

```
procedure MAIN(Scalar function $f$, Domain $M$)
    for each Vertex v in $M$ do
        if is_local_extremum(v, $f$, $M$) then
            schedule_task(SWEEP(v, $f$, $M$));

procedure SWEEP(Vertex v, $f$, $M$)
    while Vertex saddle == null do
        saddle = visit_next_neighbor(v, $f$, $M$);
    schedule_task(UPDATE_SADDLE(saddle, v, $f$, $M$));

procedure UPDATE_SADDLE(Vertex s, Vertex v, $f$, $M$)
    s.children.add(v);
    if all_children_arrived(s) then
        schedule_task(SWEEP(s, $f$, $M$))
        for each Vertex c in s.children do
            if on-the-fly-test(persistence(c, s)) then
                add_arc_to_output(c, s);
```

The function is_local_extremum returns a bool, that is true, if v is a local minimum (for join trees) or maximum (for split trees) in M regarding f. The procedure schedule_task creates a new task for the parallel runtime environment to execute. For details on the functions visit_next_neighbor and all_children_arrived please refer to the original algorithm [GFJT17]. schedule_task() represents the use of the task-parallel runtime scheduler and persistence() is a function that returns the arc weight for simplification w, add_arc_to_output creates an arc in the resulting merge tree. Lastly, the function on-the-fly-test() uses our methods in section 3 (e.g. insert the argument to the BQ-Summary and test resulting rank against p) to decide whether the arc is pruned or added to the output, the consecutive concurrent calls to this function can be interpreted as the stream S.

Note, that this implementation was derived for benchmarking and represents only one possible application of on-the-fly simplification and the novel simplification parameters. On-the-fly simplification as described in section 2 can be applied to different simplification schemes and merge tree construction algorithms, under the following conditions.

First, it is assumed that the merge tree construction output can be expressed as a stream of finalized arcs. Thus individual arcs are identified gradually over time and will not be modified afterwards. This assumption for example only holds partially true for parallel peak pruning [CWSA16], where the stream would be filled in batches, after each iteration of the algorithm.

Second, it is assumed that the decision whether to prune an arc is solely dependent on its persistence and ε. This is generally not the case, as in most simplification methods described above, arcs are pruned from leaves upwards, and an arc is protected from pruning if it has the largest w among all its siblings. However, in the construction methods described above, construction of an arc only happens after construction of all its children, thus this assumption can be enforced by buffering finalized arcs until a larger valued or all siblings appeared in S. For example in [GFJT17] all merge tree arcs connected to the same saddle are finalized at once.

Third, it is noteworthy, that while constructing a contour tree from simplified merge trees is possible, it may not result in the same tree as simplifying the contour tree itself, even when utilizing the same simplification method. This is due to the existence of w-structures in contour trees, that may protect additional arcs from pruning, which can not be detected in merge trees alone.

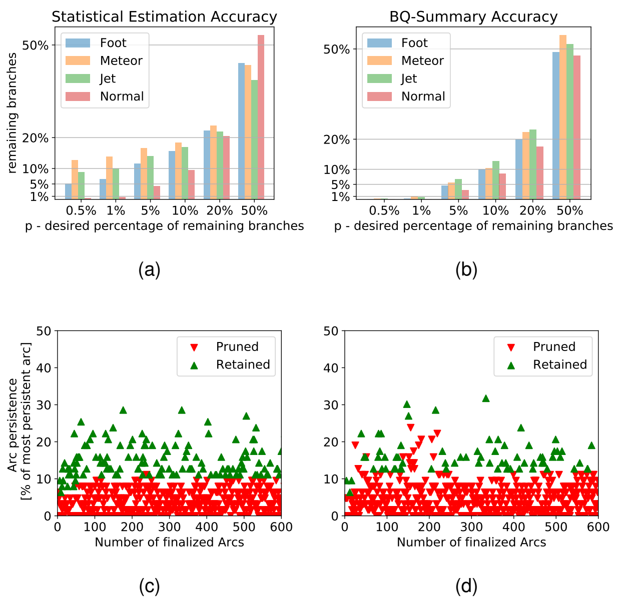

Figure 2: *(a) and (b) show achieved estimation accuracy for the Gaussian estimation and the bq-summary with different p on all data sets. (c) and (d) show the first 600 (of ca. 400.000) decisions/arcs for the Gaussian estimation and bq-summary. Each finalized arc is represented by a triangle in sequence of their arrival in the stream on the x-axis and their (relative) persistence/weight w on the y-axis. One can see some initial fluctuation, that stabilizes towards a mostly constant threshold (for the rest of the 400.000 decisions). (c) and (d) stem from the Foot data set and $p = 20\%$*

5. Results

All results emerged from experiments run on processors of type Intel XEON SP 6126 (2.6 GHz, 12 CPU cores, 96GB RAM).

Resulting estimation accuracy has been benchmarked for both percentile estimation techniques on four datasets, seen in Figure 2. The used data sets are a CT scan of a foot [TFL*17], a simulation of a meteor impacting in the ocean [PG17], and a jet fluid stream simulation. Additionally, a fourth data set is identical to the meteor simulation, but arcs are assigned a random weight, sampled from a Gaussian normal distribution. Note, that the Gaussian based statistical estimator has excellent accuracy for this data set. However it suffers heavily from skewness in the weight distribution of arcs in real life data sets, thus pruning not enough arcs. Additionally it is outperformed by the more accurate BQ-Summary.

Table 1: *Runtimes of simplification based on classical fixed threshold, summary and statistical based percentile threshold and fixed memory budget*

Data set	Fixed ε	BQ-Summary	Gaussian	Budget
Foot	6.24	7.01	7.17	6.48
Meteor	0.42	0.48	0.44	0.44
Jet	46.81	48.03	48.98	47.2

Performance penalties for maintaining the data sets for fixed memory limit and percentile estimation are shown in Table 1. All figures given represent the average of 50 measurements.

6. Conclusion

For on-the-fly simplification and especially within large-data related scenarios, two alternatives to a user specified fixed persistence threshold ε become both interesting and non-trivial to use: Simplifying all but the N branches with highest persistence and simplifying all but the p percent branches with the highest persistence of the output. This allows for example to reduce the output size by 90% or force the output to fit into a fixed memory budget.

We introduced and evaluated algorithmic structures to guide simplification with these alternative parameters. Both performance impact and estimation accuracy showed satisfactory in the benchmarks. For future research, replacing the Gaussian density estimation by more sophisticated kernel density estimation techniques might be an interesting topic.

7. Acknowledgements

Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) - 398122172.

References

[AN15] ACHARYA A., NATARAJAN V.: A parallel and memory efficient algorithm for constructing the contour tree. *2015 IEEE Pacific Visualization Symposium (PacificVis)* (2015), 271–278. 1

[BG15] BIEDERT T., GARTH C.: Contour Tree Depth Images For Large Data Visualization. In *Eurographics Symposium on Parallel Graphics and Visualization* (2015), Dachsbacher C., Navrátil P., (Eds.), The Eurographics Association. doi:10.2312/pgv.20151158. 1

[BLW12] BAUER U., LANGE C., WARDETZKY M.: Optimal topological simplification of discrete functions on surfaces. *Discrete & Computational Geometry 47*, 2 (Mar 2012), 347–377. doi:10.1007/s00454-011-9350-z. 2

[CGL83] CHAN T. F., GOLUB G. H., LEVEQUE R. J.: Algorithms for computing the sample variance: Analysis and recommendations. *The American Statistician 37*, 3 (1983), 242–247. doi:10.1080/00031305.1983.10483115. 3

[CKMS06] CORMODE G., KORN F., MUTHUKRISHNAN S., SRIVASTAVA D.: Space- and time-efficient deterministic algorithms for biased quantiles over data streams. In *Proceedings of the Twenty-fifth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems* (New York, NY, USA, 2006), PODS '06, ACM, pp. 263–272. doi:10.1145/1142351.1142389. 2

[CLLR05] CHIANG Y.-J., LENZ T., LU X., ROTE G.: Simple and optimal output-sensitive construction of contour trees using monotone paths. *Computational Geometry 30* (02 2005), 165–195. doi:10.1016/j.comgeo.2004.05.002. 1, 2

[CSA03] CARR H., SNOEYINK J., AXEN U.: Computing contour trees in all dimensions. *Computational Geometry 24*, 2 (2003), 75 – 94. Special Issue on the Fourth CGC Workshop on Computational Geometry. doi:https://doi.org/10.1016/S0925-7721(02)00093-7. 1, 2

[CSEH07] COHEN-STEINER D., EDELSBRUNNER H., HARER J.: Stability of persistence diagrams. *Discrete & Computational Geometry 37*, 1 (Jan 2007), 103–120. doi:10.1007/s00454-006-1276-5. 2

[CSvdP04] CARR H., SNOEYINK J., VAN DE PANNE M.: Simplifying flexible isosurfaces using local geometric measures. In *Proceedings of the Conference on Visualization '04* (Washington, DC, USA, 2004), VIS '04, IEEE Computer Society, pp. 497–504. doi:10.1109/VISUAL.2004.96. 2

[CSvdP10] CARR H., SNOEYINK J., VAN DE PANNE M.: Flexible isosurfaces: Simplifying and displaying scalar topology using the contour tree. *Computational Geometry 43*, 1 (2010), 42 – 58. Special Issue on the 14th Annual Fall Workshop. doi:https://doi.org/10.1016/j.comgeo.2006.05.009. 2

[CWSA16] CARR H. A., WEBER G. H., SEWELL C. M., AHRENS J. P.: Parallel peak pruning for scalable smp contour tree computation. In *IEEE Symposium on Large Data Analysis and Visualization 2016, LDAV 2016* (2016), IEEE. 1, 2, 4

[ELZ02] EDELSBRUNNER, LETSCHER, ZOMORODIAN: Topological persistence and simplification. *Discrete & Computational Geometry 28*, 4 (Nov 2002), 511–533. doi:10.1007/s00454-002-2885-2. 2

[EMP06] EDELSBRUNNER H., MOROZOV D., PASCUCCI V.: Persistence-sensitive simplification functions on 2-manifolds. In *Proceedings of the Twenty-second Annual Symposium on Computational Geometry* (New York, NY, USA, 2006), SCG '06, ACM, pp. 127–134. doi:10.1145/1137856.1137878. 2

[GFJT17] GUEUNET C., FORTIN P., JOMIER J., TIERNY J.: Task-based augmented merge trees with fibonacci heaps. In *2017 IEEE 7th Symposium on Large Data Analysis and Visualization (LDAV)* (Oct 2017), pp. 6–15. doi:10.1109/LDAV.2017.8231846. 1, 2, 3, 4

[GFJV16] GUEUNET C., FORTIN P., JOMIER J., VIJAY: Contour forests: Fast multi-threaded augmented contour trees. In *IEEE Symposium on Large Data Analysis and Visualization 2016, LDAV 2016* (2016), IEEE. 2

[HC19] HRISTOV P., CARR H.: W-structures in contour trees. 2

[LPG*14] LANDGE A. G., PASCUCCI V., GYULASSY A., BENNETT J., KOLLA H., CHEN J., BREMER P.: In-situ feature extraction of large scale combustion simulations using segmented merge trees. In *SC* (2014), IEEE Computer Society, pp. 1020–1031. 1, 2

[MSD*12] MAADASAMY, SENTHILNATHAN, DORAISWAMY, HARISH, NATARAJAN, VIJAY: A hybrid parallel algorithm for computing and tracking level set topology. In *HiPC* (2012), IEEE Computer Society, pp. 1–10. 1, 2

[MW14] MOROZOV D., WEBER G.: Distributed contour trees. In *Topological Methods in Data Analysis and Visualization III* (2014), pp. 89–102. URL: http://escholarship.org/uc/item/9k99z474. 1

[PCMS05] PASCUCCI V., COLE-MCLAUGHLIN K., SCORZELLI G.: Multi-resolution computation and presentation of contour trees. 2

[PG17] PATCHETT J., GISLER G.: *Deep Water Impact Ensemble Data Set*. Tech. rep., 2017. LA-UR-17-21595. URL: https://datascience.dsscale.org/wp-content/uploads/2017/03/DeepWaterImpactEnsembleDataSet.pdf. 4

[PSBM07] PASCUCCI V., SCORZELLI G., BREMER P.-T., MASCARENHAS A.: Robust on-line computation of reeb graphs: Simplicity and speed. *ACM Trans. Graph. 26*, 3 (July 2007). doi:10.1145/1276377.1276449. 1, 2

[Pug89] PUGH W.: Concurrent maintenance of skip lists. *Institute for Advanced Computer Studies, Department of Computer Science, University of Maryland, College Park, CS-TR-2222.1* (1989). 2

[RS14] RAICHEL B., SESHADHRI C.: A mountaintop view requires minimal sorting: A faster contour tree algorithm. *CoRR abs/1411.2689* (2014). 2

[RWS*17] ROSEN P., WANG B., SETH A., MILLS B., GINSBURG A., KAMENETZKY J., KERN J., R. JOHNSON C.: Using contour trees in the analysis and visualization of radio astronomy data cubes. 1

[SZG*08] SARKAR R., ZHU X., GAO J., GUIBAS L. J., MITCHELL J.: Iso-contour queries and gradient descent with guaranteed delivery in sensor networks. In *INFOCOM 2008. The 27th Conference on Computer Communications* (2008), IEEE. 2

[TFL*17] TIERNY J., FAVELIER G., LEVINE J. A., GUEUNET C., MICHAUX M.: The Topology ToolKit. *IEEE Transactions on Visualization and Computer Graphics* (2017). URL: https://topology-tool-kit.github.io/downloads.html. 3, 4

[TP12] TIERNY J., PASCUCCI V.: Generalized topological simplification of scalar fields on surfaces. *IEEE Transactions on Visualization and Computer Graphics 18*, 12 (Dec. 2012), 2005–2013. doi:10.1109/TVCG.2012.228. 2

[TTF04] TAKAHASHI S., TAKESHIMA Y., FUJISHIRO I.: Topological volume skeletonization and its application to transfer function design. *Graph. Models 66*, 1 (Jan. 2004), 24–49. doi:10.1016/j.gmod.2003.08.002. 2

[Tuk77] TUKEY J. W.: *Exploratory Data Analysis*. Addison-Wesley, 1977. 3

[vKvOB*97] VAN KREVELD M., VAN OOSTRUM R., BAJAJ C., PASCUCCI V., SCHIKORE D.: Contour trees and small seed sets for isosurface traversal. In *Proceedings of the Thirteenth Annual Symposium on Computational Geometry* (New York, NY, USA, 1997), SCG '97, ACM, pp. 212–220. doi:10.1145/262839.269238. 1

[VP03] V. PASCUCCI K. C.-M.: Parallel computation of the topology of level sets. *Algorithmica 38(1)* (2003), 249–268. 2

[WHDS04] WOOD Z., HOPPE H., DESBRUN M., SCHRÖDER P.: Removing excess topology from isosurfaces. *ACM Trans. Graph. 23*, 2 (Apr. 2004), 190–208. doi:10.1145/990002.990007. 1

[ZT11] ZHOU J., TAKATSUKA M.: Importance driven contour tree simplification. *Internet Computing and Information Services, International Conference on 0* (09 2011), 265–268. doi:10.1109/ICICIS.2011.169. 2