Algorithmic Improvements on Hilbert and Moore Treemaps for Visualization of Large Tree-structured Datasets

W. Scheibel[1], C. Weyand[1], J. Bethge[1], and J. Döllner[1]

[1]Hasso Plattner Institute, Digital Engineering Faculty, University of Potsdam, Germany

Abstract

Hilbert and Moore treemaps are based on the same named space-filling curves to lay out tree-structured data for visualization. One main component of them is a partitioning subroutine, whose algorithmic complexity poses problems when scaling to industry-sized datasets. Further, the subroutine allows for different optimization criteria that result in different layout decisions. This paper proposes conceptual and algorithmic improvements to this partitioning subroutine. Two measures for the quality of partitioning are proposed, resulting in the min-max and min-variance optimization tasks. For both tasks, linear-time algorithms are presented that find an optimal solution. The implementation variants are evaluated with respect to layout metrics and run-time performance against a previously available greedy approach. The results show significantly improved run time and no deterioration in layout metrics, suggesting effective use of Hilbert and Moore treemaps for datasets with millions of nodes.

CCS Concepts

• ***Human-centered computing*** → ***Treemaps;*** *Information visualization;*

1. Introduction

Much of the data measured and collected today has a tree structure and tree visualization techniques are versatile tools to convey this data, its structure, relationships, and meaning [SHS11]. One specific technique for visualization is the treemap [STLD20] as a family of visualization techniques with different underlying layouts [SLD20] and visual representation [LSDT19]. Especially for the layout algorithm, the different approaches come with different characteristics regarding visual representation (shape), perceptive properties (readability, recognition), and implementations (run-time performance, source code availability). One goal for optimization is the stability of the layout over time, i.e., adhering to the *principle of visual-data correspondence* [KS14]. As such, a high layout stability facilitates in maintaining a users'Designing a layout algorithm that ensures good layout stability is an ongoing aspiration and resulted, for example, in the proposal for rectangular Hilbert and Moore treemaps [TC13].

Depending on the domain, tree-structured data can be of larger size, e.g., when measuring software system information with hundreds of thousands of nodes [LSDT19] (see Fig. 1). Although the visual display and cognition of users are usually the limiting factors when providing visualization of large-scale datasets, the underlying layout of a treemap should be computed on the whole dataset. This enables the use of multiscale visualizations with level-of-detail and level-of-abstraction approaches, as these require information about positions and extents of the treemap layout elements, even if they are not displayed at all times [LSHD17]. Previous applications for these treemap algorithms seem to have used datasets with multiple levels of hierarchy and up to 60 000 nodes [VSC*20]. Nonetheless, we identified run-time limitations when applying Hilbert and Moore treemaps ourselves, especially on even larger datasets with more than 100 000 nodes. The main run-time complexity comes from the partitioning subroutine to "divide the data in four weighted quadrants that have roughly equal weights" [TC13]. In addition, the vague definition of *roughly* further allows for a number of choices on what qualities a partitioning of the weight quadrants should have. Existing implementations range from greedy approaches in $O(n^2)$ to brute-force implementations with a cubic run time. The greedy approach can be applied for dataset sizes of up to 60 000 nodes, more sophisticated approaches are strongly limited with an extrapolated run time of several days.

Contributions. We add a variation of optimization criteria to Hilbert and Moore treemaps and provide novel partitioning algorithms, implementation specifics, and both a metrics-based and a run-time evaluation. This includes:

- a formalization of the the list of weight partitioning subroutine and two optimization criteria for possible solutions, namely *min-max* and *min-variance*,
- two novel partitioning algorithms that provide optimal solutions,
- pseudo code for and run-time complexity assessment of both algorithms in comparison to the already available *greedy* approach,
- and experimental results on run-time performance and layout metrics, applied to mid-sized and large-sized software system data for datasets with up to 2,4 million nodes.

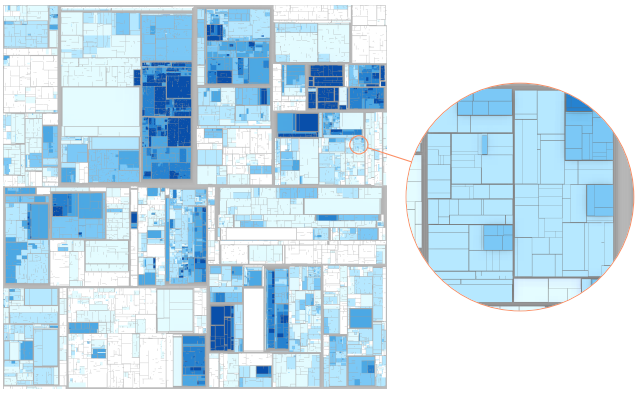

Figure 1: *Hilbert layout of the Firefox project with over 100 000 leaf nodes using min-max partitioning.*

Related Work The Hilbert and Moore space-filling curves and the partitioning of the list of weights are the basic building blocks for both treemap algorithms. The concept of space-filling curves was previously considered for treemap layout algorithms with the Strip treemap [BSW02] and later conceptually extended by Spiral and S-Shape treemaps [TS07]. A space-filling Hilbert curve was previously used for non-convex Jigsaw Maps [Wat05]. The approach to partition the list of weights to perform divide-and-conquer layouts was first proposed with the Binary treemap [SP14] and has been established since then [LSNH13, FGDH19].

2. Partitioning Subroutine

The part of the layout algorithm with most impact on run time and presumed impact on layout quality is the recursive partitioning of the list of weights into four parts with "roughly equal weights" [TC13]. However, the original paper provides no detailed approaches and no discussion on possible optimization goals of this partitioning, limiting expressiveness and comparability of implementations. As of now, one specific implementation is publicly available, implementing a *greedy* approach. In the following, we formalize the problem of this list-of-weight partitioning and propose two quality measures, namely *min-max* and *min-variance*. We further provide linear-time algorithms to find an optimal solution for each quality measure. Although not discussed for each algorithm, the use of a prefix sum on the list of weights is a strong factor on the run-time complexity and should be considered a standard optimization for such range queries [CLRS09].

Problem Formalization. Let $A = (a_0, \dots, a_{n-1})$ be an array of n positive numbers with $W = \sum a_i$. For two indices $i < j$, we denote the subarray $(a_i, \dots, a_{j-1})$ as $A[i, j]$. A set of $k+1$ indices $c_0 = 0, c_1, \dots, c_{k-1}, c_k = n$ with $c_i < c_{i+1}$ induces a subdivision of A into k disjoint consecutive subarrays $P_1, \dots, P_k$ where $P_i = A[c_{i-1}, c_i]$. We call $P = \{P_1, \dots, P_k\}$ a k-partitioning of the array A and the indices $c_1, \dots, c_{k-1}$ its cuts (omitting c_0 and c_k). Further, we denote P_i as a segment and $W_i = \sum_{a \in P_i} a$ its weight. The term quadrant is used for segment when k is four. To measure the quality of a partitioning P, we propose to measure the heaviest segment

$$\max(P) = \max_{1 \le i \le k} W_i, \tag{1}$$

or the variance of segment weights

$$\mathrm{Var}(P) = \sum_{1 \le i \le k} (W/k - W_i)^2 . \tag{2}$$

In both cases the goal is to minimize the respective measure. Analogous to minimizing the heaviest segment (min-max), one could maximize the lightest segment (max-min). Note that a perfectly balanced partition ($W_i = W/k$) is optimal under all three measures, but unbalanced partitionings are penalized differently. We restrict our analysis to min-max and min-variance.

Finding a Min-Max Partition. When searching an optimal solution with respect to the measure given in Eq. 1, the number of cuts is fixed to 3 while the weight of the heaviest quadrant is to be minimized. We derive a linear algorithm for this optimization task by first considering a slightly modified problem. Suppose an upper bound B on the weight of the heaviest segment were given and the number of cuts to achieve this bound should be minimized. In this setting, it is possible to compute a solution using a scan-line approach. That is, one extends the current segment as long as the summed weight stays below the bound and otherwise makes a cut. The correctness of this approach can be proven via a *greedy stays ahead* argument. The naive algorithm runs in linear time but can be reduced to $O(\log n)$ per cut using a binary search over the prefix sum as shown in Algorithm 1. In the pseudo code, the binary search function returns the first index i where the prefix sum exceeds the value $used + B$, that is, the rightmost index to which we can extent the current segment without violating the limit B.

Since, the number of required cuts is monotone decreasing in B, we can use binary search over B to find the smallest bound that can be met with at most 3 cuts. With Algorithm 1 as a subroutine, we obtain an algorithm for the original problem with exactly three cuts. To maintain logarithmic running time for one invocation of Algorithm 1, we limit the number of found segments. In total this yields an $O(n + \log(\frac{W}{\varepsilon}) \log(n))$ algorithm with W being the total weight and ε the desired precision, e.g., $\varepsilon = 1$ for integral numbers.

Finding a Min-Variance Partition. While a naive approach tests all combinations for the three cuts leading to a run time of at least $O(n^3)$, there is a dynamic programming solution for this problem that runs in $O(n^2 k)$ where k is the number of desired cuts [Sta13a, Sta13b]. Improving on this, we provide an algorithm that computes an optimal partitioning with 3 cuts in linear time (see Algorithm 2). The idea is to test all positions for a middle cut m, while maintaining cuts l and r that bisect $A[0, m]$ and $A[m, n]$, respectively.

Regarding the correctness of this algorithm, we test all possible positions for the middle cut so at one point we find the optimal one. What remains to show is that, given a middle cut, we place the other cuts optimally. First, we observe that the left and right cut are independent. Now, suppose some subarray $A[b, e]$ that is to be bisected into two segments by a cut c. Let $M = W/4$ be the desired quadrant weight. We search an index c before which to cut with $b < c < e$ such that $(M - \sum_{i=b}^{c-1} a_i)^2 + (M - \sum_{i=c}^{e-1} a_i)^2$ is minimized according to Eq. 2. Let $S = \sum_{i=b}^{e-1} a_i$ be the total weight of

Algorithm 1: Pseudo code of the min-max partitioning subroutine. This algorithm is performed as subroutine of a binary search over the weight of the heaviest segment.

```
   Input: segment bound B, prefix sum S
1  cuts ← [ ], used ← 0
2  while used < W and |cuts| < 4 do
3  |  i ← binary_search(S, used + B)
4  |  cuts.append(i)
5  |  used ← S[i − 1]
6  return cuts
```

Algorithm 2: Pseudo code of the min-variance partitioning algorithm.

```
1  cuts ← [1,2,3], l ← 1, r ← 3
2  for m ← 2 to n − 2 do
3  |  while l + 1 cuts A[0,m] more balanced do
4  |  |  l ← l + 1
5  |  while r + 1 cuts A[m,n] more balanced do
6  |  |  r ← r + 1
7  |  if variance(l,m,r) < variance(cuts) then
8  |  |  cuts ← [l,m,r]
9  return cuts
```

the given subarray. If we could cut the sum S at any point x, instead of only at discrete indices, then the problem would be to minimize the imbalance given by the function

$$f(x) = (M-x)^2 + (M-(S-x))^2$$
$$= 2x^2 - 2Sx - 2SM + 2M^2 + S^2.$$

The root of the derivative $f'(x) = 4x - 2S$ is at $x = \frac{S}{2}$ and since f is a quadratic function, we know that the value of f is lower the closer we get to $\frac{S}{2}$. This means that the optimal way to place the left and right cut is to split the subarrays up to and from the middle cut as balanced as possible. Lastly, if the most balanced bisection of $A[0,m]$ is at index l, then the most balanced bisection l' of $A[0,m+1]$ will be to the right of l. Therefore, we can incrementally move l as we move m. The same holds for the right cut.

Regarding the run time of our algorithm, the outer loop tries all $n-3$ positions for the middle cut. The while loops advance the left and right cut, but each cut individually can only be incremented at most $n-4$ times. Thus, amortized over all iterations of the outer loop, the while loops add an overhead of $O(n)$. Both the imbalance of a bisected subarray (lines 3,5) and the added variance of all four quadrants (line 7) can be computed in constant time using prefix sums. This yields an overall run time of $O(n)$.

Greedy Partition. The only public implementation we are aware of uses a greedy approach to determine the quadrants [Son18, VSC*20]. This partitioning approach extends the current quadrant while the difference to the optimal quadrant weight decreases. The implementation handles additional edge cases, e.g., $A = (1,47,26,26)$ would result in three segments, whereas $A = (20,20,20,20,20)$ yields five segments. The referenced implementation recomputes the sum of the current quadrant in each iteration, leading to quadratic run time. With this, our larger datasets do not finish in reasonable time (i.e., 1 hour) and we optimized this algorithm using a running total to achieve linear time.

Table 1: *Lists of weights and optimal solutions w.r.t. introduced quality measures. The sum is 100 which results in an optimal quadrant size at 25. Example 1: The solution for min-max is not unique. The solution for min-variance is also optimal for min-max. Example 2: A bad instance for the min-max partitioning with three cuts.*

Example 1	**Weights**						
Optimization Goal	20	9	16	17	8	29	1
min-variance	20 |	25		25		30	
min-max	29		16 |	25		30	
greedy	29		33	|	37	|	1

Example 2	**Weights**					
Optimization Goal	1	33	22	11	11	22
min-variance	34		22 |	22	|	22
min-max	1 |	33	33	|	33	
greedy	34		22 |	22	|	22

Quality Guarantees. The min-max and the min-variance approach provide strong guarantees, namely optimality regarding their respective measure. In contrast, there are no such or similar guarantees for the greedy approach. In fact, Example 1 in Table 1 shows an instance where the greedy approach finds a very unbalanced solution although a good alternative exists. While the intuitively best solution in the Example 1 is optimal with respect to variance as well as the heaviest quadrant, a less balanced partitioning that is also optimal regarding the heaviest quadrant is given. Moreover, there are instances where the optimal min-max solution is arguably worse (less balanced) than a solution with a larger heaviest segment (see Example 2 in Table 1). We find that min-variance best captures the notion of quadrants being *of roughly equal size*.

3. Evaluation

The three algorithms were evaluated by their layout quality and their computation time. We use twelve datasets containing multiple snapshots of a software system, two datasets containing a single snapshot but with a larger number of nodes, and one dataset containing part of the Github project landscape with 1.9 million leaf nodes. Details on the datasets including example layouts are provided with the additional material and we provide the measurement prototype in source as an open source repository [SWB21].

Layout Metrics. The optimization criteria for treemap layouts revolve around readability and stability [TC13]. These two abstract targets are approximated using layout metrics. For readability, we use the average aspect ratio (AAR). For stability, the different layouts of tree-structured datasets are measured against different types of changes: (1) the average distance change w.r.t. position and extent (ADC) [BSW02], (2) the relative position change w.r.t. neighborhood and adjacency (RPC) [SSV18], (3) the average angular displacement that measures the change in direction for adjacent nodes (AAD) [WD08, GCB*15], (4) the rotation-invariant adaption that is the relative direction change (RDC) [HBD17], and (5) the

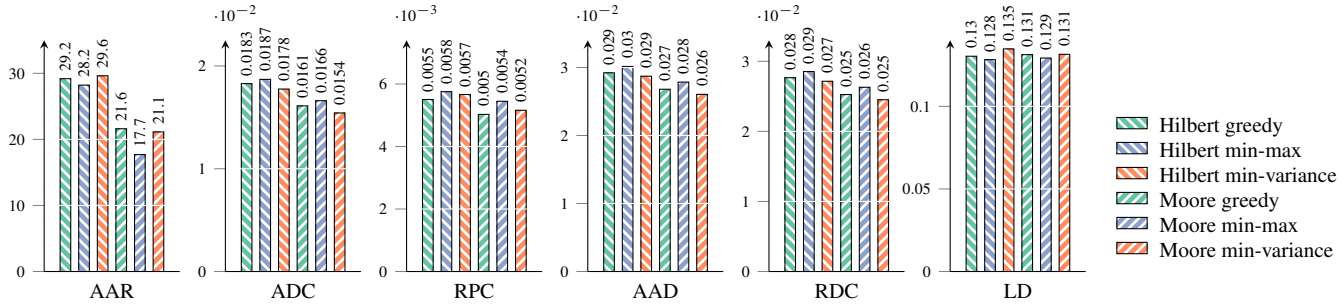

Figure 2: *Comparison of the layout metrics. A lower value is preferred for each metric.*

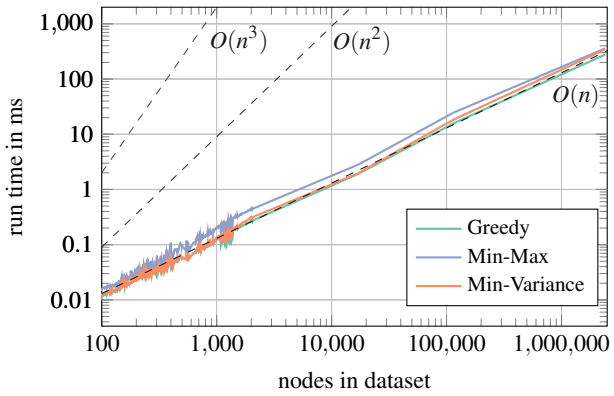

Figure 3: *Algorithm run time for* greedy, min-max, *and* min-variance *implementations. Both the x-axis and the y-axis use logarithmic scaling. For reference, symbolic plots are added for the complexity classes $O(n)$, $O(n^2)$, and $O(n^3)$.*

location drift, that considers the overall displacement of a layout element (LD) [TC13]. Our layout quality tests were applied to both the Hilbert and the Moore treemap variants using all three partitioning algorithms, resulting in six measurement series. The layout metric results show an overall similar stability and quality of treemap layouts (Figure 2). There is a slight difference between using a Hilbert and a Moore curve for the base layout, as the layout metrics are slightly lower when using a Moore curve except for the LD metric. Further, the *min-max* partitioning algorithm results in slightly better aspect ratios for the layout elements than the *greedy* and *min-variance* approaches. Although the underlying curve slightly influences the layout readability and stability, the partitioning algorithm does not show a similar effect.

Run-time Performance. The algorithm run times were measured for each partitioning algorithm, each dataset, and each snapshot. We use the average layout computation time of 500 test runs that are measured after 500 additional warmup runs. The runs were executed using a single thread on an Ubuntu 20.04 system with an Intel Core™ i7-6850K at 3.6 GHz base speed. In general, the results for the three implementations perform similarly well (Figure 3). The measurements suggests a linear run time complexity for each implementation. Our improvement to the *greedy* implementation allows for a fair comparison of the implementations. Without the optimization, the plot would follow the $O(n^2)$ symbolic plot. As overall observation, we see a ranking of the implementation variants regarding run-time performance with the *greedy* approach being first, the *min-variance* approach being second, and the *min-max* approach being third. The fastest execution of the greedy approach is expected as it uses minimal logic to distribute weights among quadrants. However, the main result is that the execution time for each implementation and dataset is similar and always less than one second for each measurement. More specifically, the largest dataset with 2.4 million nodes was computed in 357 milliseconds using the *min-max* approach and even faster for the other two approaches.

4. Conclusions

Summarizing, the partitioning subroutine of Hilbert and Moore treemaps allows for different interpretations and optimization goals. We presented two optimization goals, the *min-max* and the *min-variance*, and linear-time algorithms to solve them optimally. Further, the previously used greedy approach was optimized to linear time as well. Our improvements allow Hilbert and Moore treemaps to be used on large real-world datasets over multiple revisions while accounting for both layout stability and run-time performance. Our preliminary experiments confirm the linear run time and show no deterioration in layout quality. The choice of datasets resulted in no significant differences, which may come from their number or variety and them not being standardized datasets. Further, an open question is which measure is most natural and best reflects the balance of a partitioning. For future work, we plan to extend the evaluation w.r.t. dataset sizes, further metrics [BSW02], and aspect ratios [KHA10], as well as to explore the continuity of the space-filling curve across hierarchy levels and sibling nodes.

Acknowledgements

We want to thank the anonymous reviewers for their valuable comments and suggestions to improve this article. This work has been supported by the German Federal Ministry of Education and Research (BMBF) through grant 01IS19006 (“KI-Labor ITSE”). Further, this work is part of the “Software-DNA” project, which is funded by the European Regional Development Fund (ERDF – or EFRE in German) and the State of Brandenburg (ILB).

References

[BSW02] BEDERSON B. B., SHNEIDERMAN B., WATTENBERG M.: Ordered and quantum treemaps: Making effective use of 2d space to display hierarchies. *Transactions on Graphics 21*, 4 (2002), 833–854. doi:10.1145/571647.571649. 2, 3, 4

[CLRS09] CORMEN T. H., LEISERSON C. E., RIVEST R. L., STEIN C.: *Introduction to algorithms*, 3 ed. MIT press, 2009. 2

[FGDH19] FENG C., GONG M., DEUSSEN O., HUANG H.: Treemapping via balanced partitioning. CVM '19. 2

[GCB*15] GHONIEM M., CORNIL M., BROEKSEMA B., STEFAS M., OTJACQUES B.: Weighted maps: treemap visualization of geolocated quantitative data. In *Proc. Visualization and Data Analysis* (2015), vol. 9397 of *VDA '15*, International Society for Optics and Photonics, SPIE, pp. 163–177. doi:10.1117/12.2079420. 3

[HBD17] HAHN S., BETHGE J., DÖLLNER J.: Relative direction change - a topology-based metric for layout stability in treemaps. In *Proc. 12th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications - Volume 3: IVAPP* (2017), IVAPP '17, INSTICC, SciTePress, pp. 88–95. doi:10.5220/0006117500880095. 3

[KHA10] KONG N., HEER J., AGRAWALA M.: Perceptual guidelines for creating rectangular treemaps. *Transactions on Visualization and Computer Graphics 16*, 6 (2010), 990–998. doi:10.1109/TVCG.2010.186. 4

[KS14] KINDLMANN G., SCHEIDEGGER C.: An algebraic process for visualization design. *Transactions on Visualization and Computer Graphics 20*, 12 (2014), 2181–2190. doi:10.1109/TVCG.2014.2346325. 1

[LSDT19] LIMBERGER D., SCHEIBEL W., DÖLLNER J., TRAPP M.: Advanced visual metaphors and techniques for software maps. In *Proc. 11th International Symposium on Visual Information Communication and Interaction* (2019), VINCI '19, ACM, pp. 11:1–8. doi:10.1145/3356422.3356444. 1

[LSHD17] LIMBERGER D., SCHEIBEL W., HAHN S., DÖLLNER J.: Reducing visual complexity in software maps using importance-based aggregation of nodes. In *Proceedings of the 12th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications – Volume 3: IVAPP* (2017), IVAPP '17, INSTICC, SciTePress, pp. 176–185. doi:10.5220/0006267501760185. 1

[LSNH13] LIANG J., SIMOFF S., NGUYEN Q. V., HUANG M. L.: Visualizing large trees with divide & conquer partition. In *Proc. 6th International Symposium on Visual Information Communication and Interaction* (2013), VINCI '13, ACM, pp. 79–87. doi:10.1145/2493102.2493112. 2

[SHS11] SCHULZ H.-J., HADLAK S., SCHUMANN H.: The design space of implicit hierarchy visualization: A survey. *Transactions on Visualization and Computer Graphics 17*, 4 (2011), 393–411. doi:10.1109/TVCG.2010.79. 1

[SLD20] SCHEIBEL W., LIMBERGER D., DÖLLNER J.: Survey of treemap layout algorithms. In *Proc. 13th International Symposium on Visual Information Communication and Interaction* (2020), VINCI '20, ACM, pp. 1:1–9. doi:10.1145/3430036.3430041. 1

[Son18] SONDAG M.: IncrementalTreemap Repository, 2018. https://gitaga.win.tue.nl/max/IncrementalTreemap, last accessed on 2021-04-16. 3

[SP14] SHNEIDERMAN B., PLAISANT C.: *Treemaps for space-constrained visualization of hierarchies*. Tech. rep., Human-Computer Interaction Lab, 2014. http://www.cs.umd.edu/hcil/treemap-history, last accessed on 2021-04-16. 2

[SSV18] SONDAG M., SPECKMANN B., VERBEEK K.: Stable treemaps via local moves. *Transactions on Visualization and Computer Graphics 24*, 1 (2018), 729–738. doi:10.1109/TVCG.2017.2745140. 3

[Sta13a] STACKEXCHANGE CONTRIBUTION: Partition array into k subsets, each with balanced sum, 2013. https://cs.stackexchange.com/questions/19181/partition-array-into-k-subsets-each-with-balanced-sum, last accessed on 2021-04-16. 2

[Sta13b] STACKOVERFLOW CONTRIBUTION: Algorithm to split an array into p subarrays of balanced sum, 2013. https://stackoverflow.com/questions/14120729/algorithm-to-split-an-array-into-p-subarrays-of-balanced-sum, last accessed on 2021-04-16. 2

[STLD20] SCHEIBEL W., TRAPP M., LIMBERGER D., DÖLLNER J.: A taxonomy of treemap visualization techniques. In *Proc. 15th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications – Volume 3: IVAPP* (2020), IVAPP '20, INSTICC, SciTePress, pp. 273–280. doi:10.5220/0009153902730280. 1

[SWB21] SCHEIBEL W., WEYAND C., BETHGE J.: Hilbert and Moore Treemap Layouts Prototype, 2021. https://github.com/varg-dev/hilbert-moore-treemap-layouts-prototype, last accessed on 2021-04-16. 3

[TC13] TAK S., COCKBURN A.: Enhanced spatial stability with hilbert and moore treemaps. *Transactions on Visualization and Computer Graphics 19*, 1 (2013), 141–148. doi:10.1109/TVCG.2012.108. 1, 2, 3, 4

[TS07] TU Y., SHEN H. W.: Visualizing changes of hierarchical data using treemaps. *Transactions on Visualization and Computer Graphics 13*, 6 (2007), 1286–1293. doi:10.1109/TVCG.2007.70529. 2

[VSC*20] VERNIER E., SONDAG M., COMBA J., SPECKMANN B., TELEA A., VERBEEK K.: TreemapComparison GitHub Repository, 2020. https://github.com/tue-alga/TreemapComparison/, last accessed on 2021-04-16. 1, 3

[Wat05] WATTENBERG M.: A note on space-filling visualizations and space-filling curves. In *Proc. Symposium on Information Visualization* (2005), INFOVIS '05, IEEE, pp. 181–186. doi:10.1109/INFVIS.2005.1532145. 2

[WD08] WOOD J., DYKES J.: Spatially ordered treemaps. *Transactions on Visualization and Computer Graphics 14*, 6 (2008), 1348–1355. doi:10.1109/TVCG.2008.165. 3