Vision, Modeling, and Visualization (2017)
M. Hullin, R. Klein, T. Schultz, A. Yao (Eds.)

Improving Layout Quality by Mixing Treemap-Layouts Based on Data-Change Characteristics

Joseph Bethge, Sebastian Hahn and Jürgen Döllner

Hasso Plattner Institute, Faculty of Digital Engineering, University of Potsdam, Germany

Abstract

This paper presents a hybrid treemap layout approach that optimizes layout-quality metrics by combining state-of-the-art treemap layout algorithms. It utilizes machine learning to predict those metrics based on data metrics describing the characteristics and changes of the dataset. For this, the proposed approach uses a neural network which is trained on artificially generated dataset,s containing a total of 15.8 million samples. The resulting model is integrated into an approach called Smart-*Layouting. This approach is evaluated on real-world data from 100 publicly available software repositories. Compared to other state-of-the-art treemap algorithms it reaches an overall better result. Additionally, this approach can be customized by an end user's needs. The customization allows for specifying weights for the importance of each layout-quality metric. The results indicate, that the algorithm is able to adapt successfully towards a given set of weights.*

Categories and Subject Descriptors (according to ACM CCS): [Human-centered computing]: Visualization—Treemaps; [Human-centered computing]: Visualization—Empirical studies in visualization

1. Introduction

Treemaps have been researched for over 25 years [JS91, Sch11] as a visualization technique for the depiction of hierarchical data. Geometric shapes, e.g., rectangles, are laid out in a space-filling non-overlapping manner to display the hierarchical structure of a dataset and additional attributes. This allows for an efficient use of limited space making the technique especially useful for large datasets. The size of the shapes usually corresponds to a weight attribute defined on a per-element basis. Other visual variables, such as color or shading can be added to the visualization to display additional attributes [Ber83, Car03]. Furthermore, multiple treemaps can be used to depict the data at different times or in different states, so called snapshots.

Depicting hierarchical structures and artifact-related information within the context of the surrounding structure is an important visualization aspect in different domains, such as visualizing business data maps [VvWVdL06], showing software quality measures of software system artifacts [BE95, BD11], and image collection browsing [Bed01]. Depending on the use case of the created depictions, certain qualities of the created treemap layouts are of varying importance. These aspects are optimal aspect ratios (near one), maintaining the order of the depicted dataset, or producing highly stable layouts for varying data that changes over time. The latter is important if recognizing and memorizing the position of familiar elements between snapshots is intended. For example, weekly depictions software metric data [BD11] require more layout stability than static reports of sports results [JB97]. Producing optimal results for all possible use cases presents an algorithmic challenge. Within the high number of treemap algorithms, designed to improve certain aspects of previously published ones, no algorithm is superior in all aforementioned aspects and under all circumstances. In fact, they all have certain strong and weak points regarding these aspects, which further depend on characteristic in the data.

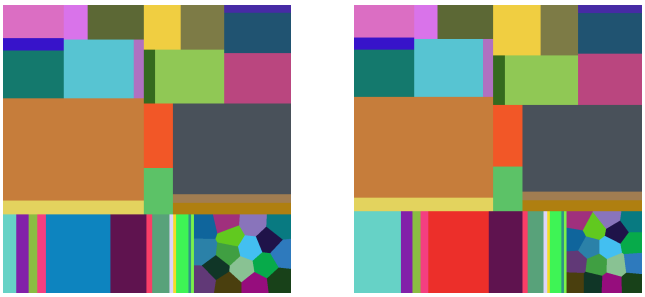


Figure 1: *Two treemaps showing the* real lines of code *of files in the* format *module of the* d3 *Javascript library created by the* Smart *layout algorithm. The base algorithms* Hilbert, Slice&Dice, *and* Voronoi *were used for different sub-hierarchies of the data.*

For example, any rectangular algorithm necessarily produces high aspect ratios for unbalanced data, such as two items with

DOI: 10.2312/vmv.20171261

weights 999 and 1 [Wat05]. Only algorithms, which depart from the traditional rectangular regions, can achieve low aspect ratios for this case.

Based on the these currently incomprehensible dependencies between the dataset and the algorithm's performance, the question arises, whether the differences between datasets can be recognized and exploited by an algorithm. This thought can be extended to respect the hierarchy of both the dataset and the depiction. It could be possible to not only use the heterogeneity between different datasets but within different parts of a single dataset as an advantage. This leads us to one central question: *Can different treemap layout algorithms be combined into one dynamic algorithm, which achieves better layout qualities than any of the individual algorithms, by predicting individual algorithm performance for each sub-hierarchy based on descriptive data metrics?*

In this paper we present an approach which uses multiple neural networks to predict individual algorithm performance and optimizes algorithm selection. A combination of existing and novel data metrics is used to describe the input data. For the evaluation of the system, established layout quality metrics are used. The outline of this paper remains as follows: An overview about treemap layout algorithms and the research regarding layout quality is given in Section 2. The descriptive data metrics and the layout-quality metrics are described in Section 3. In Section 4 the training process of the neural network and the *Smart* algorithm is shown. The validation of different model architectures and testing of the final *Smart* algorithm is summarized in Section 5. Finally, we conclude this paper and present areas for future work in Section 6.

2. Related Work

Treemap Layout Algorithms have been used for more than 25 years [Sch11]. The first treemap layout algorithm **Slice&Dice** was presented by Johnson and Shneiderman [JS91]. *Slice&Dice* divides the available space into linear regions for each level in the tree and alternates between vertical and horizontal division, depending on the tree depth. This potentially leads to thin elongated rectangles, reducing the visibility of single elements and occurs more severely if elements in the tree have a large number of children. The **Squarified** algorithm, published by Bruls et al. [BHVW00], focuses on achieving square-like aspect ratios. Although the algorithm leads to very favorable aspect ratios close to one, it introduces more instability in consecutive treemap layouts, if weights are changed or elements are added to the tree. In 2002, Bederson et al. presented the **Strip** algorithm [BSW02], which is a modification of the *Squarified* algorithm. It aims at a compromise between decent aspect ratios and stability, but items are processed in their original order and the alignment and direction of the strips are constant, e.g. from left to right. These modifications create parallel lines of items and make it more stable than *Squarified*, but it has worse aspect ratios. A variant of this algorithm, called **Strip-Inv**, alternates the directions of the created strips in each new row. This slightly increases stability, since when items at the ends of a strip move, they only move to an adjacent position on the next strip, instead of jumping to the opposite end. The **Spiral** treemap layout algorithm was developed by Tu and Shen [TS07]. The algorithm uses a flow concept similar to a spatial curve, following a spiral pattern, to lay out items, such that neighboring items in the data are adjacent in the treemap. Tak and Cockburn [TC13] presented the **Hilbert** and **Moore** algorithm based on the identically named space-filling curves to determine the item positions. The result has low aspect-ratios and high stability on average. Balzer et al. published the **Voronoi** algorithm based on Voronoi diagrams which uses non-rectangular areas in 2005 [BD05]. It was adapted by Hahn et al. by adding an initial stable distribution [HTMD14]. It has further been extended by Rinse van Hees and Jurriaan Hage with a stable placement based on a scaled Hilbert curve which also decreases computation time [vHH15].

Only very few approaches were presented combining different algorithms in this way. The first approach, called **Mixed** treemaps, was reported by Vliegen et al. [VvWVdL06] in 2006. They use the algorithms *Slice&Dice*, *Strip*, and *Squarified* and slight modifications of these algorithms. In contrast to our work, they use a smaller number of algorithms and suggest a fixed configuration of algorithms, created by an expert for a specific visualization. This means, their algorithm is adaptable to a dataset, however, this must be done manually by an expert. A slightly different idea, which is most similar to the approach in this work is presented by Hahn and Döllner [HD17] in 2017. They use statistical analysis over a large set of inputs to identify which base algorithm should by applied for a given sub-hierarchy of a dataset. The resulting **Hybrid** algorithm combines eight rectangular and non-rectangular based algorithms and achieves superior performance in visualizing ten different software projects. Their approach is similar to the one presented in this work, as they also choose the most suitable layout algorithm among a set of existing algorithms. However, their approach relies on statistical analysis, rather than on machine learning, and only considers the number of elements as data characteristics. In addition, it is limited to optimization of a fixed weighting of layout-quality metrics, whereas the optimization objective of the dynamic approach presented in this work can be adapted by an end user later on.

Different measures have been proposed to measure the usefulness of created treemap depictions. Possible tasks which can be performed with treemaps include comparing or estimating sizes of elements and locating certain elements. Comparing sizes of items in treemaps with different aspect ratios is mentioned as being frustrating by Stasko et al. [SCGM00]. The *aspect ratio* of items tries to measure the degree of difficulty for this task and was introduced by Bruls et al. [BHVW00]. The research regarding layout stability is strongly connected to the model of a mental map for graphs defined by Misue et al., who specify three main aspects: orthogonal *ordering*, *proximity relations*, and *topology* [MELS95]. The first measure which captures layout stability **distance change** was introduced by Bederson et al. to measure layout stability [BSW02]. It measures the Euclidean distance between consecutive positions and aspect ratios of items. The *relative direction change* metric was introduced by Hahn et al. [HBD17]. It captures changes between the angles of each pair of elements in a treemap, independent of rotations of whole groups of elements and serves as a mean to measure the topological and arrangement stability of treemap items. Furthermore, they state, that a combination of average aspect ratio, average distance change and relative direction can be used to significantly predict user performance in recovery tasks. Therefore these three metrics were also used to evaluate the *Smart* approach.

3. Input Data

This section explains which metrics were used to model the dependency between dataset characteristics and algorithm performance. The following terms are used throughout this section:

- **Snapshot:** One instance of a tree with an additional numeric attribute for each element. It is visualized in one treemap.
- **Dataset:** One complete set of multiple snapshots, either generated artificially or derived from real data, such as a version control system.

3.1. Data Metrics

To allow prediction of the outcome, the neural network needs the receive all relevant inputs for the task. The first important factor is the orientation and size of the boundaries for the current sub-hierarchy. It is conveyed to the neural network by using the width and height, normalized by the maximum of both. Six additional metrics describe each individual snapshot: the maximum, minimum and median weight, the number of elements, and the sum and variance of weights. Four metrics are used to describe changes between snapshots directly, the relative weight change proposed by Steinbrückner [Ste12], and two novel metrics, the positional change and the hierarchy change. The positional change measures positional shifts of elements between their sibling nodes. Let p_i, p_{i+1} denote the positions of one data element and n_i, n_{i+1} be the total number of children in the sub-hierarchy. Then the positional change PC is defined as $PC = \frac{1}{2 \cdot \max(n_i, n_{i+1})}(|p_{i+1} - p_i| + |(n_{i+1} - p_{i+1}) - (n_i - p_i)|)$. The hierarchy change measures deletions and insertions in the hierarchy with the Jaccard index [Jac08]. If A and B denote the set of elements in a sub-hierarchy in two snapshots, then the hierarchy change HC is $HC = 1 - \frac{|A \cap B|}{|A \cup B|}$.

3.2. Layout Metrics

The aim of the neural network is to predict the layout quality of an arbitrary input dataset for each algorithm. Three layout metrics were used to measure different aspects of layout quality:

- **Average Aspect Ratio:** Reflects the readability of the treemap and how well sizes can be distinguished and compared.
- **Average Distance Change:** Measures stability of individual treemap items.
- **Relative Direction Change:** Measures stability between the treemap items (arrangement and topology of treemap items).

They are defined in the literature [HBD17, TC13].

3.3. Normalization

Before the metric data was used for training and prediction, it was normalized. The different metrics, which are used as input data are preprocessed and normalized in different ways (see Table 1). All methods are applied to one sample in a fixed way, i.e. the normalization is not dependent on maximum or minimum values over the whole dataset. Instead it sometimes relies on the maximum and minimum value of some other metrics of that particular sample. Preliminary tests showed, that the accuracy of the models increased, when the skewed distributions of values for the input metrics were transformed to be more evenly distributed. Therefore, in addition to achieving values between zero and one another goal of the normalization process was to achieve a distribution similar to a uniform distribution for these metrics. Further, the normalization functions are bijective, so the inverse function can be used to retrieve the actual predictions from the predicted values.

Table 1: *List of metrics used as machine learning data and their normalization functions. These metrics are used to approximate the dependency between two snapshots, and the layout-quality metrics between two corresponding depictions. Metrics with a star (*), only refer to one of these snapshots. Therefore, they appear twice in the actual machine learning data, representing either the first or the second snapshot respectively (as indicated by the a or b in the super script in formulas). The neural network predicts the metrics with type* out*, based on the value of the metrics with type* in*.*

Metric name	Short	Type	Normalization
Width of Parent (*)	P_w	In	$\frac{x}{\max(P_w, P_h)}$
Height of Parent (*)	P_h	In	$\frac{x}{\max(P_w, P_h)}$
Weight Sum (*)	W_s	In	$\frac{x}{\max(W_\mathrm{s}^a, W_\mathrm{s}^b)}$
Largest Weight (*)	W_max	In	x
Smallest Weight (*)	W_min	In	$f_m(x)$
Weight Variance (*)	W_v	In	$f_m(x)$
Weight Median (*)	W_med	In	$f_m(x)$
Number of Children (*)	W_n	In	$f_e(x)$
Relative Weight Change	RWC	In	$f_m\left(\frac{x}{\max(W_\mathrm{s}^a, W_\mathrm{s}^b)}\right)$
Positional Change	PC	In	$f_m(x)$
Hierarchy Change	HC	In	$f_m(x)$
Average Distance Change	ADC	Out	$f_m(x)$
Average Aspect Ratio (*)	AAR	Out	$f_i(x)$
Relative Direction Change	RDC	Out	$f_m(x)$

Four different methods of normalization were used. Both, the prediction of the average aspect ratio in the first and in the second snapshot is normalized with the inverse function $f_i(x) := \frac{1}{x}$. Because the original average aspect ratio x is always $x \geq 1$, the inversed average aspect ratio is always $0 < f_i(x) \leq 1$. The logarithmic normalization function $f_e(x) = \frac{1}{2}(\log_{10}(x+10) - 1)$ is used for the *number of children* to provide values between 0 and 1 for the input interval $[1, 1000]$. The logarithmic normalization function $f_m(x) = \log_{100}(99x+1)$ is used for multiple metrics, for example, for the *relative weight change* to distribute the values more evenly in the interval between 0 and 1. The number of children W_num is represented by natural numbers with no real limits. A logarithmic function $f_e(x) := \frac{1}{2}(\log_{10}(x+10) - 1)$ is used instead to normalize these values. No actual hierarchy, neither in the training data, nor in the evaluation data had more than 1000 children in a single sub-hierarchy, therefore, this function could be used as a normalization method. If any instances were encountered, the value could simply be clipped at 1.

4. Smart Treemap Layouts

The *Smart* layout algorithm was implemented based on one trained neural network for each algorithm. These neural networks were

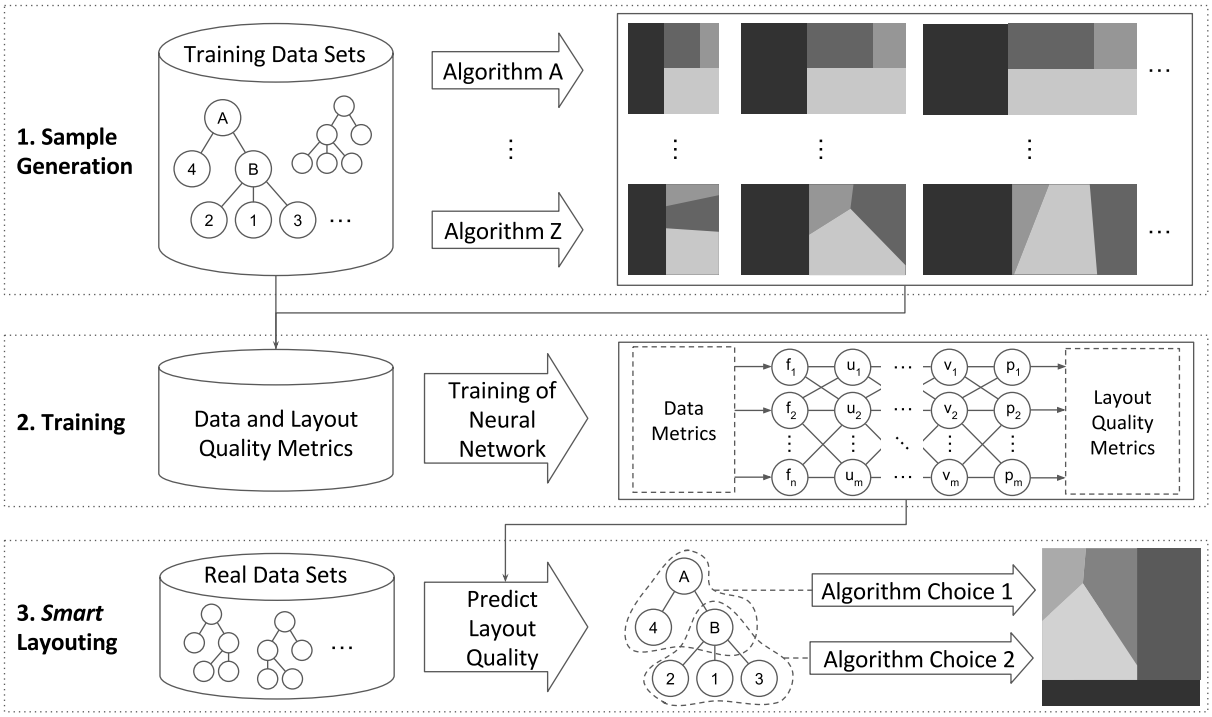


Figure 2: *The general procedure used to create a* Smart *algorithm. First, training samples are generated, by calculating data and layout-quality metrics for a large number of datasets. Secondly, a neural network is trained, to predict the layout-quality metrics, based on the data metrics. Finally, the resulting model can be used to predict the layout-quality metrics for each algorithm for unknown data. Based on this information, the optimal algorithm can be chosen for each sub-hierarchy.*

trained from a large number of treemaps which was generated by all algorithms from artificially generated datasets. This section describes the generation of the artificial data, the training process and how the predictions are used to select the optimal algorithm. An overview can be seen in Figure 2.

4.1. Training the Model

The training data was generated to produce data which is similar to real data, but can be generated in the desired amount. Different parameters were used to ensure creating artificial data which has a high diversity and coverage of edge cases. The general process is similar to the approach described by Tak and Cockburn [TC13] for their evaluation process. However, the process is executed six times, with an adaptation of some of the parameters. For reference we denote the six combinations of parameters with A, B, C, D, E, and F. Three log-normal distributions with different variances (1.05 (A), 1.5 (B, C, D, E), 1.8 (F)) are used instead of the original Zipfian distribution for the initial weight assignment. Additionally, before modifying the weight attribute for each snapshot some of the dataset types contained a chance to add or remove items. The actual chance is calculated per element in the treemap with chances of 0% (C), .5% (A, B, E, F), or 1% (D). This covers real datasets, where the addition and removal items can regularly happen. Random values x are drawn from a Gaussian distribution to modify the weights in each step by multiplying them with e^x. The variance of the Gaussian distribution is 0 (E), 0.05 (A, B, C, F), or 0.1 (D). These six combinations were used to generate datasets with 44 different numbers of initial elements. Every number between 2 and 20, and every third number between 20 and 50 is used. Furthermore, we used every fifth number between 50 and 100, and every tenth number between 100 and 150. Each combination of initial size and type of dataset is created 5 times with different random seeds. Afterwards the eight algorithms (*Slice&Dice*, *Strip*, *Strip-Inverted*, *Squarified*, *Spiral*, *Moore*, *Hilbert*, *Voronoi*) are used to create multiple single-level treemaps with different bounding rectangles. In addition to a square with width and height of 1, both narrow and wide variations are created by reducing either the width or height to $\frac{9}{10}$, $\frac{3}{4}$, $\frac{2}{3}$, $\frac{1}{2}$, $\frac{1}{3}$, $\frac{1}{4}$, and $\frac{1}{10}$. This results in a total of almost 2 million samples for each treemap algorithms, 15.8 million in total.

4.2. The Smart Algorithm

To allow the optimal prediction of the best algorithm for each sub-hierarchy in a dataset, the layout process is executed by order of the snapshots. Therefore, the treemap for the children of the root element is calculated first for all snapshots. Then, the recursive layout process for the sub-hierarchies starts for all snapshots in parallel. Because of this, the bounding rectangles are known for all snapshots and can be used for the prediction. Before each layout process starts, the corresponding data metrics can be calculated between each pair of snapshots. Then the trained models are used to predict the layout-quality metric values for each algorithm and each pair of snapshots. Based on the predicted layout-quality met-

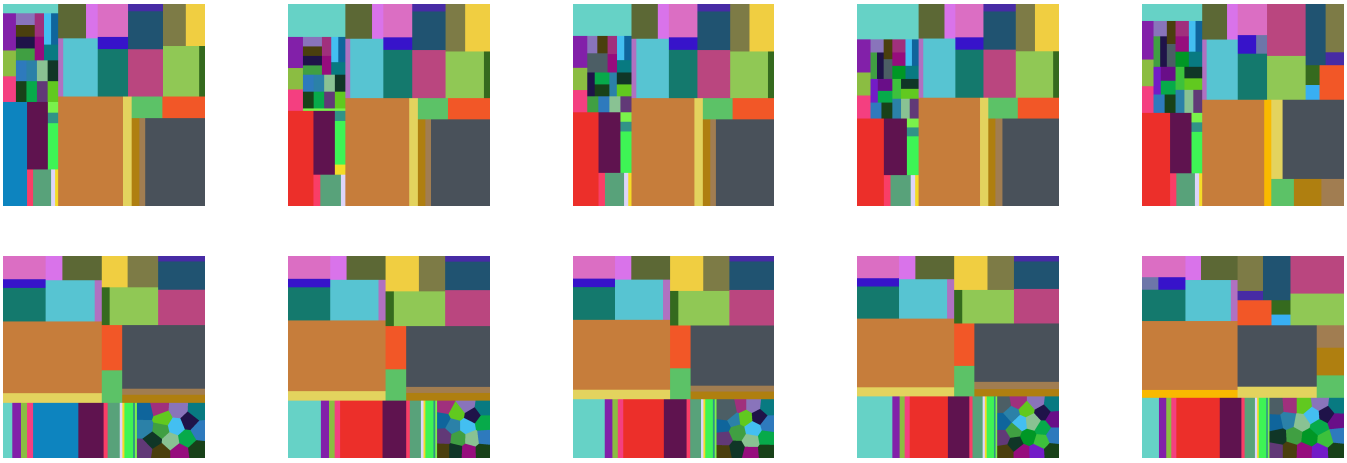


Figure 3: *Five snapshots of the format module of the d3 library as calculated by the* Hilbert *(top) and the* Smart *algorithm (bottom).*

rics and importance weighting of each metric the average score is calculated between all snapshots. Afterwards *Smart* algorithm selects the layout algorithm which achieves optimal values according to the predictions. In this process, the polygonal layout algorithm (*Voronoi*) is used only for sub-hierarchies which do not contain any other sub-hierarchies, i.e. they only contain leaf elements. Otherwise the recursive layout process could no longer use the rectangular algorithms.

Instead of using machine learning an exhaustive search can be used, by running all algorithms for every sub-hierarchy to compare the actual results. However, this approach would increase the total time needed for creating the final layout by a factor of the number of algorithms. Therefore, this might be feasible for a low number of algorithms, but not for a larger number. Since it is expected that new algorithms for treemaps are added over time, this gap will further increase. By using machine learning, the decision can be made based on the results of the trained neural network, which can be computed faster than some of the more complex layout algorithms. To achieve this speedup, additional time is needed to train the initial models, which can then be used over and over.

5. Evaluation

The evaluation data consists of software source code repositories publicly available from Github. The snapshots were created based on the revision history, with one snapshot being taken every month. The data contained snapshots for each month from November 2013 to January 2016, resulting in at most 26 snapshots for each repository. One example depiction can be seen in Figure 3.

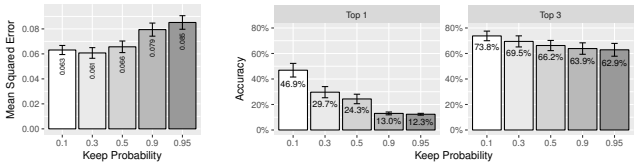
The file structure of the repository represented the hierarchy, where each leaf node represents a file in a repository and the intermediary nodes represent folders, containing other folders and files. The *real lines of code* metric was used as a weight attribute. Furthermore, duplicate snapshots were removed, i.e., when no activity occurred in the repositories for a whole month. Finally, different repositories were used for comparing the different models (*validation* data), and evaluating the final model compared to the other layout algorithms (*test* data).

These two different data collections, denoted with either *validation* or *test*, both include data from 100 repositories. The *validation* data contains 128,350 sub-hierarchies. The calculated data and layout metrics for each of the eight algorithms were used as validation samples, about 1 million in total. The number of snapshots for each repository was between 2 and 26 ($\text{Mean} = 6.41, \text{sd} = 5.91$). The *validation* data is used for evaluating different model parameters and configurations. The *test* data contains 198,101 sub-hierarchies. In this part of the data, each repository had between 2 and 22 ($\text{Mean} = 6.14, \text{sd} = 5.45$) snapshots. This data is used solely for evaluating the *Smart* algorithm compared to the base algorithms, based on the best model from the previous experiments. Therefore, the data and layout metrics were calculated only for the evaluation of the resulting layouts.

5.1. Preliminary Experiments

Multiple experiments were conducted to evaluate the optimal selection of various training and model parameters and the general size and structure of the model. The mean squared error for the predicted and the actual value was measured to validate individual model performance. The lower this error, the better a model is able to predict the metric values. Furthermore, the performance across all models was validated by measuring the accuracy of correctly predicting the best algorithm according to each metric (*Top 1*). This means, the predictions were cast by all models individually, and the result was considered correct, if the presumably best algorithm according to the predictions was actually the best algorithm for that layout-quality metric. Furthermore, this accuracy was calculated for a second variation, where all of the three actual best algorithms were considered correct (*Top 3*).

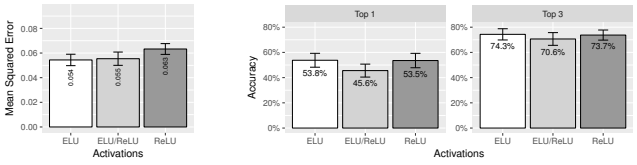
The structure of the neural network consists of a number of fully connected layers each followed by an activation layer and a dropout layer. The final layer produces 4 outputs, two values for the Average Aspect Ratio (one for each input snapshot) and the Relative Direction Change and the Average Distance Change. The number of layer and the sizes of each layer is defined by a list notation, where each element stands for the number of intermediary elements be-



(a) *The performance based on mean squared error (lower is better).*

(b) *The performance based on prediction accuracy (higher is better).*

Figure 4: *The performance of different keep probabilities for the dropout layer, based on the two different measures. The tests show best results for probabilities of 0.1 and 0.3.*



(a) *The performance based on mean squared error (lower is better).*

(b) *The performance based on prediction accuracy (higher is better).*

Figure 5: *The performance of the different activation functions, based on the two different measures. The results slightly favor the ELU activation function, but are overall similar and insignificant.*

tween two layers. For example, a model with the input vector of size 19, one hidden layer with output size 10 and a second hidden layer with output size 5, and the final prediction layer with output size 4 would be written as $[10,5]$.

The first decision was to use a learning rate $lr = 0.001$, and the *AdamOptimizer* [KB14] instead of a *GradientDescent* optimizer [Bot10]. This was based on a grid search, with a model of size $[20,10]$, without dropout and ReLU [NH10] activations. After these initial parameters were determined, five probabilities $0.95, 0.9, 0.5, 0.3, 0.1$ for retaining weights in the dropout layer the *keep probability* p were tested (see Figure 4). The results show the lowest mean squared error for $p = 0.3$, slightly below $p = 0.1$ and $p = 0.5$ (see Figure 4a). However, the prediction accuracy reaches their highest values for $p = 0.1$ in both the *Top 1* and the *Top 3* category. Therefore, we decided to use a keep probability of $p = 0.1$ for the dropout layers.

In the same manner, three different configurations of activation functions were tested:

1. ELU [CUH15] for the whole network
2. ELU for the hidden layers and ReLU for the final prediction layer
3. ReLU [NH10] for the whole network

The results show, that either ELU or the combination of ELU/ReLU achieves the best model performance (see Figure 5). The mean squared error of both variants is slightly better than the error obtained by using ReLU (see Figure 5a). However, the prediction accuracy is much lower for ELU/ReLU (see Figure 5b). Therefore, the optimal function seems to be using the ELU in the whole network.

As for model size and structure, we determined, that models with less than two layers were significantly worse than those with two or more layers. However there was no significant difference between the models $[20,10]$, $[40,20,10]$, and $[20,10,5]$ according to the mean squared error or the prediction accuracy.

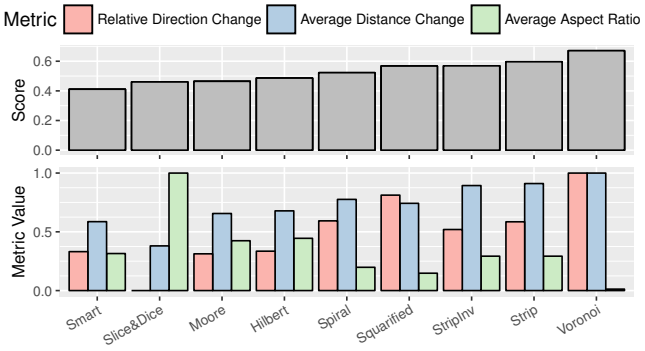
5.2. Evaluation of the Smart Algorithm

Three different objectives were used to evaluate the *Smart* algorithm and its ability to adapt to user input. They are presented by different weights of importance for the layout-quality metrics. The first objective, called **balanced**, uses equal weights for all three layout-quality metrics. The second objective represents a use case where **high visibility** is more important, the weight of the Average Aspect Ratio (AAR) is increased to be four times as high as for the Average Distance Change (ADC) and the Relative Direction Change (RDC). Finally, the last objective represents a use case where **high stability** is more important, the weight of ADC and RDC is four times as high as the weight of AAR. The resulting *layout quality score* is then calculated by the weighted average, using the previously defined weights. Since a low value is better for all metrics, as it indicates higher stability and small aspect ratios, a lower layout quality score is better. The performance of the *Smart* layout algorithm was evaluated based on this layout quality score.

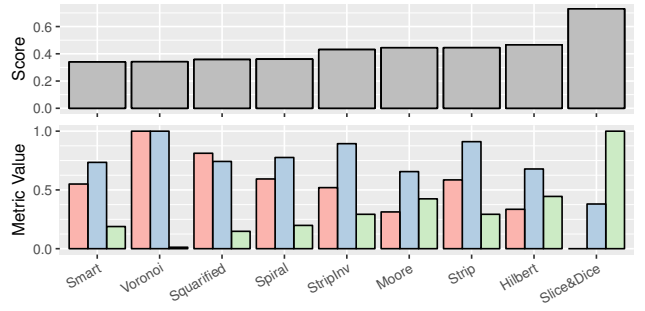
The results show *Smart* as the best algorithm with a much better AAR than the second best algorithm *Slice&Dice* (see Figure 6a) for the balanced objective. But, the RDC and ADC of *Slice&Dice* are higher at the same time. *Moore* and *Hilbert* achieve third and fourth best results. *Moore* achieves slightly better metric values than *Hilbert* for all three metrics. *Moore* also achieves a lower RDC better than *Smart*, but is worse in the other two metrics. *Spiral* is the next best algorithm, and even though it has a better AAR than all of the previous algorithms, the RDC and ADC are larger. *Squarified* achieves the sixth best result through its low AAR. The seventh and eight best results are achieved by *StripInv* and *Strip*. They both have slightly worse ADC and AAR, than *Spiral*. But, *StripInv* has a better RDC, hence being better than *Strip* overall. *Voronoi* achieves the worst result, even though it has very low AAR. However, the ADC and RDC are very high as expected, and are the reason for the algorithm achieving the worst overall score.

The individual metric values of all base algorithms were equal to those in the *balanced* dataset, since they can not be adapted to specific optimization objectives. Therefore, the main question was, whether the *Smart* algorithm could be better than the algorithms, which were biased towards a certain metric, such as stability for *Slice&Dice* and low aspect ratios for *Voronoi* and *Squarified*.

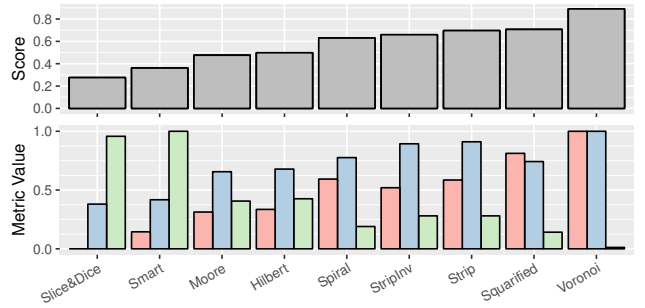
The *Smart* algorithm achieved the best score for the *high visibility* objective (see Figure 6b). However, *Voronoi* achieved an overall score almost as good. The *Smart* algorithm had a slightly worse AAR than *Voronoi* and *Squarified*, but achieved higher stability measured by RDC and ADC. As expected all other base algorithms performed worse for the *high visibility* objective. *Slice&Dice* in particular was second best performing for the *balanced* objective, but now had the worst result overall.



(a) *Algorithm performance based on the* balanced *objective.*



(b) *Algorithm performance based on the* high visibility *objective.*



(c) *Algorithm performance based on the* high stability *objective.*

Figure 6: *The performance of all algorithms on the test data, based on different optimization objectives. The score represents the weighted average between all layout-quality metrics. The smaller the individual metrics values and the score, the better. The algorithms are ordered by their average score.*

For the *high stability* objective the *Smart* algorithm only achieved the second best result (see Figure 6c). *Slice&Dice* achieved the best score for this case. This was also the only experiment, where the *Smart* algorithm achieved a worse AAR, than *Slice&Dice*. Furthermore, the stability of *Slice&Dice* was also higher than that of the *Smart* algorithm.

Therefore, we conclude that adjusting to a *higher stability* does work with the *Smart* algorithm. But, in a real-world use case, the *Slice&Dice* algorithm might be a better choice instead, if the flexibility of adapting the objective is not important. Also, other objectives, such as weighing AAR by only a factor of 2 or 3 instead of 4 could show different results. On the contrary, adaptation towards a *higher visibility* worked well, as the *Smart* was able to achieve the best results also for that objective.

6. Conclusion

We presented a new dynamic treemap layout algorithm, the *Smart* algorithm, which combines eight existing treemap layout algorithms. A neural network was used to predict the layout qualities metrics *relative direction change*, *average distance change*, and *average aspect ratio*. Based on these predictions, the presumably best algorithm was chosen. Artificial data was generated for training the neural network, and different parameters and structures were evaluated. The *Smart* algorithm based on the trained model was evaluated against the existing treemap layout algorithms based on data from software repositories. The resulting decisions of the *Smart* algorithm were analyzed, depending on the optimization objective.

The results in the evaluation suggest, that this specific approach can be used successfully to achieve overall better results than any individual existing algorithm. It can further provide flexibility to end users, who can directly balance the importance of the layout metrics. This also provides a potential topic for future work, as to whether this flexibility can be used effectively for actual users, who create treemap visualizations. It might be possible, that other layout-quality metrics, reflect the requirements desired in treemaps in a better way. Maybe the set of metrics needs to be extended, or other metrics should be used instead. Furthermore, a user study remains to be conducted to validate the usefulness of mixing multiple layout algorithms.

In a similar way, the data metrics used as features for the neural network could be extended, modified or otherwise improved. The predictions of the machine learning models only reached about 60% accuracy on the validation data, which is quite low, when compared to other more well-known machine learning problems, such as hand-written digit recognition, where the error rate of the state-of-the-art methods is 0.21% [WZZ*13]. Therefore, there is still room for improvement. Four main factors could possibly be limiting the current machine learning solution. First, the amount and quality of training data might not be sufficient to reach better results. Secondly, the data provided to the algorithm does simply not provide enough information. Thirdly, the model is not sufficient to capture the dependencies between the data in a better way. The last reason might be, that the desired predictions are inherently unpredictable, which is unlikely since the algorithms used are deterministic. Furthermore, we think the third reason can be ruled out, since extensive testing with different model architectures was done. Therefore, the two remaining reasons are best candidates for further improvement.

Acknowledgements

The authors would like to thank the anonymous reviewers for their valuable comments. This work was funded by the Federal Ministry of Education and Research (BMBF), Germany, within the "BIMAP" project. We also want to thank seerene™ for providing us access to their massive data sets.

References

[BD05] BALZER M., DEUSSEN O.: Voronoi treemaps. In *Proceedings of the IEEE Symposium on Information Visualization (InfoVis 2005)* (2005). 2

[BD11] BOHNET J., DÖLLNER J.: Monitoring code quality and development activity by software maps. In *Proceedings of the 2nd Workshop on Managing Technical Debt* (2011), ACM, pp. 9–16. 1

[BE95] BAKER M. J., EICK S. G.: Space-filling software visualization. *Journal of Visual Languages & Computing 6*, 2 (1995), 119–133. 1

[Bed01] BEDERSON B. B.: Quantum treemaps and bubblemaps for a zoomable image browser. *Proc. User Interface Systems and Technology* (2001), 71–80. 1

[Ber83] BERTIN J.: Semiology of graphics: Diagrams, networks, maps. 1

[BHVW00] BRULS M., HUIZING K., VAN WIJK J. J.: *Squarified treemaps*. Springer, 2000. 2

[Bot10] BOTTOU L.: Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*. Springer, 2010, pp. 177–186. 6

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

[Car03] CARPENDALE M.: Considering visual variables as a basis for information visualisation. 1

[CUH15] CLEVERT D.-A., UNTERTHINER T., HOCHREITER S.: Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289* (2015). 6

[HBD17] HAHN S., BETHGE J., DÖLLNER J.: Relative direction change: A topology-based metric for layout stability in treemaps. In *Proceedings of the 8th International Conference of Information Visualization Theory and Applications (IVAPP 2017)* (2017). 2, 3

[HD17] HAHN S., DÖLLNER J.: Hybrid-treemap layouting. In *Proceedings of EuroVis 2017 - Short Papers* (2017). 2

[HTMD14] HAHN S., TRÜMPER J., MORITZ D., DÖLLNER J.: Visualization of varying hierarchies by stable layout of voronoi treemaps. In *Information Visualization Theory and Applications (IVAPP), 2014 International Conference on* (2014), IEEE, pp. 50–58. 2

[Jac08] JACCARD P.: Nouvelles researches sur la distribution florale. *Bull soc vaud sci nat 44* (1908), 223–270. 3

[JB97] JIN L., BANKS D. G.: Tennisviewer: A browser for competition trees. *IEEE Computer Graphics and Applications 17*, 4 (1997), 63–65. 1

[JS91] JOHNSON B., SHNEIDERMAN B.: Tree-maps: A space-filling approach to the visualization of hierarchical information structures. In *Visualization, 1991. Visualization'91, Proceedings., IEEE Conference on* (1991), IEEE, pp. 284–291. 1, 2

[KB14] KINGMA D., BA J.: Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014). 6

[MELS95] MISUE K., EADES P., LAI W., SUGIYAMA K.: Layout adjustment and the mental map. *Journal of visual languages and computing 6*, 2 (1995), 183–210. 2

[NH10] NAIR V., HINTON G. E.: Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)* (2010), pp. 807–814. 6

[SCGM00] STASKO J., CATRAMBONE R., GUZDIAL M., MCDONALD K.: An evaluation of space-filling information visualizations for depicting hierarchical structures. *International Journal of Human-Computer Studies 53*, 5 (2000), 663–694. 2

[Sch11] SCHULZ H.-J.: Treevis. net: A tree visualization reference. *Computer Graphics and Applications, IEEE 31*, 6 (2011), 11–15. 1, 2

[Ste12] STEINBRÜCKNER F.: Consistent software cities: supporting comprehension of evolving software systems. 3

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

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

[vHH15] VAN HEES R., HAGE J.: Stable voronoi-based visualizations for software quality monitoring. In *Software Visualization (VISSOFT), 2015 IEEE 3rd Working Conference on* (2015), IEEE, pp. 6–15. 2

[VvWVdL06] VLIEGEN R., VAN WIJK J. J., VAN DER LINDEN E.-J.: Visualizing business data with generalized treemaps. *Visualization and Computer Graphics, IEEE Transactions on 12*, 5 (2006), 789–796. 1, 2

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

[WZZ*13] WAN L., ZEILER M., ZHANG S., CUN Y. L., FERGUS R.: Regularization of neural networks using dropconnect. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)* (2013), pp. 1058–1066. 7