Tak, S. & Cockburn, A. (2013). Enhanced Spatial Stability with Hilbert and Moore treemaps. IEEE Transactions on Visualization and Computer Graphics, 19(1), 141-148.
doi:ieeecomputersociety.org/10.1109/TVCG.2012.108

Enhanced spatial stability with Hilbert and Moore treemaps

Susanne Tak, Andy Cockburn

Abstract—Treemaps are a well known and powerful space-filling visualisation method for displaying hierarchical data. Many alternative treemap algorithms have been proposed, often with the aim being to optimise performance across several criteria, including spatial stability to assist users in locating and monitoring items of interest. In this paper we demonstrate that spatial stability is not fully captured by the commonly used 'distance change' metric, and we introduce a new 'location drift' metric to more fully capture spatial stability. An empirical study examines the validity and usefulness of the location drift metric, showing that it explains some effects on user performance that distance change does not. Next, we introduce 'Hilbert' and 'Moore' treemap algorithms, which are designed to achieve high spatial stability. We assess their performance in comparison to other treemaps, showing that Hilbert and Moore treemaps perform well across all stability metrics.

Index Terms—Treemap, space-filling curve, spatial stability.

1 Introduction

Treemaps are space-filling visualisations that use rectangular regions to reveal the hierarchical structure and quantitative attributes of datasets [1], [2]. They have been used in many application areas, including the stock market [3], photo browsers (PhotoMesa) [4], discussion forums [5], hierarchical data analysis [6], [7] and visual decision making [8].

Many treemap algorithms have been proposed, offering different advantages in different contexts. Metrics have also been proposed to assist comparison between treemap algorithms. These metrics include aspect ratio (whether treemap items are predominantly square or long and thin), readability and continuity of the layout, and measures of stability, including distance change and distance change variance. Stability is particularly important because it facilitates the development of expertise through learning of item locations [9]: with experience, users can quickly glance at the treemap to monitor salient data items.

We demonstrate that stability is not fully captured by the commonly used 'distance change' metric (e.g., [2], [4], [9]). Treemaps with low distance change values can sometimes be unstable, because items never 'settle' in a particular area of the screen. To address this shortcoming we introduce a new 'location drift' metric that better encapsulates stability. We report results of an empirical study that examines how the distance change and location drift metrics influence actual user performance in item retrieval.

We then introduce two new treemaps, based on Hilbert and Moore space filling curves [10] (see Fig. 1), and demonstrate that these treemaps perform well across the full set of stability metrics.

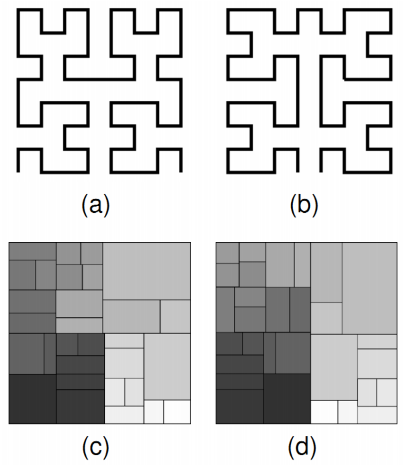

Fig. 1. (a) Hilbert and (b) Moore curve, (c) Hilbert and (d) Moore treemap (shading indicates item order).

The five specific contributions of the paper are:

- a mathematical formalisation of treemap metrics;
- introduction of the 'location drift' metric;
- an empirical study demonstrating that 'location drift' is a useful metric for evaluating stability;
- introduction of Hilbert and Moore treemaps;
- a comparison of the Hilbert and Moore treemaps and various other treemaps.

2 Related work

2.1 Treemap algorithms

Many treemap algorithms have been proposed since their introduction [1]. The early 'slice and dice' algorithm [11] (see Fig. 2a) recursively slices the available

• *S. Tak and A. Cockburn are with the Department of Computer Science and Software Engineering, University of Canterbury, Private Bag 4800, Christchurch 8140, New Zealand.*
Contact e-mail: andy@cosc.canterbury.ac.nz

space into parallel rectangles, but in doing so it is susceptible to producing unbalanced aspect ratios. Squarified [12] (see Fig. 2b) and cluster algorithms [3] produce more balanced aspect ratios.

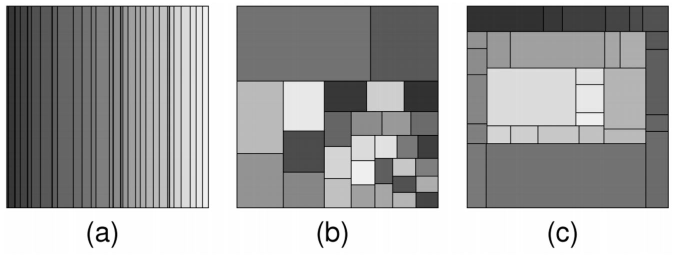

Fig. 2. Treemap algorithms: (a) slice and dice, (b) squarified, (c) spiral (shading indicates item order).

While visualisation aesthetics are often key design criteria, other factors can also be important, such as maintaining the underlying data ordering, strong representation of hierarchy, and spatial stability. Ordered treemap layouts preserve the underlying data order, with examples including slice and dice, pivot [9], strip [4], and spiral layouts [2] (see Fig. 2c). Fig. 2 uses shading to indicate item order; note that slice and dice (a) and spiral (c) maintain order, but squarified (b) does not. Many other designs have been proposed to meet the requirements of specific application areas. For example, Cushion treemaps [13] are designed to highlight hierarchical structure and Quantum treemaps [4] are designed to accommodate items with a fixed size, such as photos.

By definition, treemaps have rectangular items: "[a treemap is a] two-dimensional (2-d) space-filling approach in which each node is a rectangle whose area is proportional to some attribute such as node size" [11, p. 92]. However, this can lead to situations where the aspect ratios of the items are unavoidably unbalanced, as in the 'extreme' example when there are two items with respective weights of 999 and 1 [14]. To resolve this issue, some newly developed algorithms create a layout with non-rectangular items. For example, Voronoi treemaps [15] use arbitrary polygons rather than rectangles for visualising software metrics, and the Jigsaw layout [14] uses the space-filling H-curve [16] to create a treemap with non-rectangular puzzle-piece shaped items (in our paper we use the space-filling Hilbert and Moore curves [10]). Wattenberg [14] also provides a mathematical analysis of the use of space-filling curves for supporting space-filling visualisations. The use of non-rectangular items, however, does not come without its disadvantages. Wattenberg [14, p. 27] observes that "[...] irregular puzzle-piece shapes certainly look odd, and seem likely to make it more difficult to compare areas and understand the tree topology [...]".

2.2 Treemap metrics

Five metrics are commonly used (e.g., [2], [4], [9]) for evaluating and comparing treemap algorithms: aspect ratio, readability, continuity, distance change and distance change variance. To resolve ambiguity in their use, we formally define them below.

Aspect ratio: The aspect ratio of an item is the ratio of the longer dimension to the shorter one. Treemaps with a high mean aspect ratio are undesirable, because items are hard to recognise, select, and label, as well as being visually unattractive. We note that previous work suggests that the (perceptually) ideal mean aspect ratio is not necessarily the minimum possible value (of 1), but closer to 1.5 [17]. The mean aspect ratio of a treemap layout ranges from 1 (treemaps with perfectly square items) to very high (treemaps with many 'stretched' items) and is calculated using

$$\frac{1}{n}\sum_{i=1}^{n} max(\frac{width_i}{height_i}, \frac{height_i}{width_i}).$$

Readability: The readability [4] metric measures how easy it is to visually scan a treemap in order (see shading in Fig. 2). It is quantified by calculating the number of times the "reader's gaze" must change direction when scanning a treemap in order. When all item centres of consecutive items in the item ordering are assumed to be connected by vectors, the number of angle changes between successive vectors that are greater than .1 radians (about 6 degrees) can be counted (such an angle change represents a change in the direction of the reader's gaze). The readability metric is in the range [0,1], with 0 signifying very poor readability and 1 maximal readability, and defined by

$$1 - \frac{|angle\ >\ .1\ radians|}{n}.$$

Continuity: Similar to readability, continuity [2] quantifies how easy it is to visually scan a treemap in order. Continuity is calculated by counting how many items that are consecutive in the data ordering are adjacent in the treemap:

$$\frac{|adjacent\ consecutive\ items|}{n-1}.$$

Distance change: Distance change quantifies item movement when the underlying data is changed [4]. Distance change reflects the stability of the layout [2] with lower values indicating a more stable layout. If an item *i* in a treemap is defined as the rectangle (x, y, w, h), with x, y the position of one of the corners and w and h the width and the height, the distance change between two positions can be calculated and then averaged for all items, by using

$$\frac{1}{n}\sum_{i=1}^{n}\sqrt{(\Delta x_i)^2 + (\Delta y_i)^2 + (\Delta w_i)^2 + (\Delta h_i)^2}.$$

Distance change variance: Distance change variance [2] complements the distance change metric; a low mean distance change, but a high distance change variance means a few items move a lot.

3 Location drift

As noted in the previous section, distance change is commonly used to reflect the *stability* of a treemap. Stability is particularly important when the treemap is used to visualise dynamically updating data. In this section we demonstrate that although the distance change metric provides important insights into the stability of the treemap, it does not describe instability caused by 'drifting' item positions. For example, consider the two situations: one item flips between position A and B after each update, while another item drifts from A to B to C to D, constantly moving further away from its initial position A. It is possible that the distance change of these two items is identical at each update (if A-B = B-C = C-D), but the high *cumulative* movement distance of the latter item may detrimentally affect item retrieval because the item never 'settles' in one area of the screen.

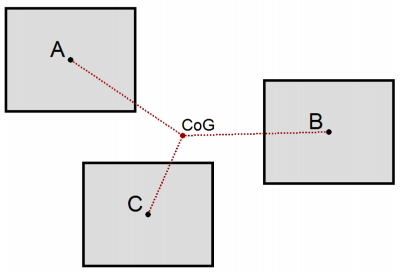

Fig. 3. Example of three positions of an item (*A*, *B*, and *C*), the centre of gravity (*CoG*) and the distances between the various positions and the centre of gravity.

We introduce the 'location drift' metric to overcome this limitation. The underlying assumption motivating 'location drift' is that human performance in item retrieval is facilitated when items are located in a particular spatial region of the display, and hindered when locations gradually drift. Location drift is calculated by determining the 'centre of gravity' (*CoG*) of all the locations an item has had across updates by averaging all x- and all y-coordinates of the various item (centre) positions (see Fig. 3). Next, the location drift of an item is determined by calculating the distance between the centre of gravity and the item at each update, and averaging these distances. The mean location drift of the treemap (with n items across m updates) is then described by

$$\frac{1}{n}\sum_{i=1}^{n}\frac{1}{m}\sum_{j=1}^{m}||position_{i,j}, CoG_i||.$$

Similar to distance change variance, location drift variance can also be calculated.

4 Empirical Study of Location Drift

This section presents an empirical validation of the location drift metric, examining how various levels of distance change and location drift affect user performance. In particular, we examine whether layouts with low distance change, but relatively high location drift impair user performance, as we proposed in Section 3.

The study used five different layout conditions with widely differing spatial properties resulting in substantially different metric values for distance change and location drift. The participants' tasks involved repeatedly selecting items within the layout as quickly and accurately as possible. All layouts consisted of 36 items in an 6 × 6 matrix. The rationale for using a grid layout, where all items are of equal size, rather than a treemap layout is to (1) minimise the risk of confounds introduced by different aspect ratios of different treemap layouts and (2) to maximally isolate effects of distance change and location drift.

4.1 Layouts

The five experimental layout conditions included two control conditions representing the end points of spatial stability: *random*, in which the location of every item was randomly assigned prior to each selection trial and *stable*, in which item locations were fixed. The *random* layout has a high mean distance change (*DC*) of 525 and a high mean location drift (*LD*) of 385, but both these values are 0 for the *stable* layout.

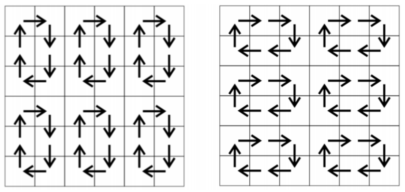

Fig. 4. Layout updates for *low distance change*.

The other three conditions were as follows:
Low distance change: Prior to each selection trial, the item locations changed according to one of the 'rotation' methods shown in Fig. 4 (randomly chosen). This layout has relatively low distance change, but high location drift (*DC*=171, *LD*=360).
Low location drift: All item locations within each quadrant (of nine items) were randomly assigned prior to each selection trial (i.e., all items always stay in the same quadrant, but within that quadrant their locations change). This layout has low location drift, but medium distance change (*DC*=250, *LD*=184).
Semi-random: All item locations were randomly assigned prior to each selection trial, with the caveat that 50% of items always stay in the same quadrant throughout all the selection trials. This layout has medium location drift and high distance change (*DC*=388, *LD*=284).

4.2 Design and procedure

The experimental interface consisted of a grid with 36 equal-sized squares, each containing a distinct coloured icon (e.g., a symbol of a house, a music note or a star) on the left side of the screen, and a cued target icon on the right side. Participants were instructed to click on the square containing the target as quickly and accurately as possible. After each selection, the grid was temporarily hidden from view and updated according to specifics of the layout. Participants pressed a 'next' button in the middle of the screen to proceed.

A Zipfian distribution of targets was used: one item was cued 18 times, one 9 times, then (6, 5, 4, 3, 3, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1) for the others (α=1.0, R^2=.97). The items were cued in random order. The rationale for using a Zipfian distribution is twofold. First, power distributions (such as Zipfian distributions) appear widely in many kinds of (natural) data, including biology, economics and finance and computer science [18]. Second, Zipfian distributions are often used for treemap evaluations [4], [9].

The experiment used a within-subject design, with all participants performing tasks in all layouts. The different layouts were presented in random order. After completion of the series of tasks in each layout the icons that had to be selected more than once in that layout were deleted from the icon collection, such that they were not re-used in following conditions. The experimental interface was displayed on a monitor with 1280 $\times$ 1024 pixels resolution.

4.3 Participants

Thirty students, naïve about the goal of the experiment, participated (15 male, 15 female, 18-39 years old). Participation lasted approximately 20 minutes.

4.4 Results and discussion

The selection times were analysed using a 5$\times$3 RM-ANOVA for factors *layout* and *experience* (low, medium or high). The experience factor allows us to examine how well the design supported formation and use of spatial memory for item locations. It was determined by assigning first-time item selections as low, 2^{rd}–14^{th} selections as medium, and 15^{th}–18^{th} selections as high experience. Any trial requiring more than one click to select the target was deemed an error and was removed from the analysis ($\sim$ 1% of selections).

There were significant main effects for both *layout* ($F_{4,116}$=22, p<.001) and *experience* ($F_{2,58}$=133, p<.001). The *stable* layout was the fastest (1.4s), followed by *low location drift* (1.7s), *semi-random* (1.9s), *low distance change* (1.9s) and *random* (1.9s). Post hoc comparisons show pairwise differences between all layouts and the *stable* layout. Fig. 5 shows a significant *layout* $\times$ *experience* interaction ($F_{8,232}$=13, p<.001), caused by relatively constant performance across experience with the *random*, *semi-random* and *low distance change* layouts in contrast to improvement of user performance with the *low location drift* and *stable* layouts.

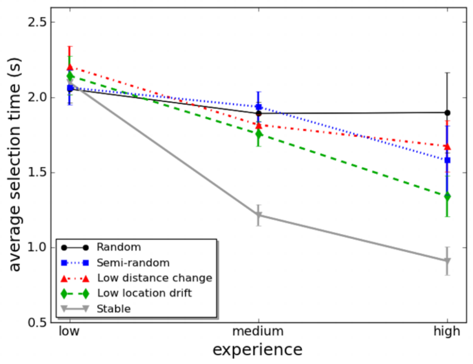

Fig. 5. Experiment mean selection times including 95% within-subjects confidence intervals for the five *layouts* by *experience*.

Many treemap designs aim to maximise stability because it facilitates rapid acquisition of *familiar* targets. To gain further insight into user performance with familiar targets, we separately analysed data from the 'high' experience level. The results show a significant main effect for *layout* ($F_{4,116}$=15, p<.001). The *stable* layout was fastest (0.9s), followed by *low location drift* (1.3s), *semi-random* (1.6s), *low distance change* (1.7s), and *random* (1.9s). Post hoc analysis (Bonferroni, α=.05) revealed that the *stable* layout was faster than all others (all p's <.001), and that *low location drift* was faster than *random* (p<.01) and *low distance change* (p<.05).

The finding that *low location drift* improves performance compared to *low distance change* is interesting, as *low location drift* has higher average distance change. However, its location drift is lower, which explains its better performance. When location drift is low, spatial location learning is aided; it is easier to learn item locations when they stay in the same area of the screen.

4.5 Results summary and discussion

The results show that a layout with low distance change, but high location drift impairs user performance. This demonstrates that location drift is a valuable metric for the evaluation of treemaps and to accurately capture the stability of a treemap layout location drift needs to be taken into account.

It is important to note that this experiment had a narrow objective: it aimed to understand the impact of location drift on user performance. As is typical of empirical work, this narrow objective raises validity concerns stemming from the trade-off between internal and external validity. In particular, our experimental context concerns user performance with treemaps, and ideally our results would readily generalise to any treemap deployment. However, treemap deployments vary widely in objective and presentation, complicating generalisation. Our experiment focussed on internal validity, facilitating rigor and replicability for our specific hypothesis;

but we stress that further work is needed to determine the impact of location drift in real treemap applications and in treemaps with imbalanced item aspect ratios.

5 Hilbert and Moore treemaps

Hilbert and Moore treemaps are intended to perform well across all measures of spatial stability, including location drift, and also perform well on the other metrics. Inspired by Wattenberg's analysis of space-filling curves for visualisations [14], we developed two new treemap algorithms based on *Hilbert* and *Moore* space-filling curves. The reason for using space-filling curves for generating treemaps is that we believed that the 'crumpled up' nature of space-filling curves would achieve low distance change and location drift, as it means items stay in the same area while migrating to new positions on the curve. However, unlike Wattenberg's work, our treemaps maintain the traditional rectangular item shape.

The Hilbert and Moore treemap algorithms are based on two (similar) space-filling curves [10], shown in Fig. 1. A space-filling curve is a self-similar continuous curve which completely covers a N-dimensional space without self-intersection. Examples of so-called level 0 and level 1 Hilbert curves are shown in Fig. 6. The level 0 curve evolves to a level 1 curve by applying a standard set of rules, as shown in Fig. 6. The level 1 curve comprises four quadrants with level 0 curves, albeit some rotated 90 degrees and/or flipped. In each of these quadrants the level 0 to 1 transition rules can be applied again to create a level 2 Hilbert curve (see Fig. 1a), and so forth.

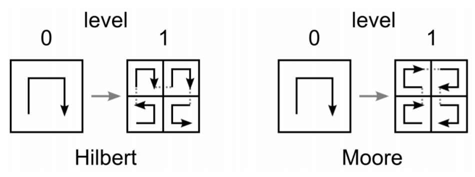

Fig. 6. Level 0 and 1 Hilbert and Moore curves.

The Moore curve is a variant of the Hilbert curve [10] with a different level 0 to level 1 transition. This results in a curve where the start and end are adjacent points (rather than adjacent corners, as is the case for the Hilbert curve). The level 0 to level 1 transition (see Fig. 6) creates an H-shaped curve. Subsequent transitions are the same as the Hilbert curve transitions. Fig. 1b shows a level 2 Moore curve.

To create a treemap based on the Hilbert and Moore space-filling curves we recursively divide the (ordered) data in four weighted quadrants, until each quadrant contains four or fewer items. Next, these quadrants are laid out on the screen. In the last step the actual items are laid out, during which some aspect ratio optimising is applied. We demonstrate the algorithm by using the example shown in Table 1.

Step 1: Recursively divide by weighted quadrants.
We divide the data in four weighted quadrants that have roughly equal weights, while the underlying ordering of the data is maintained. This is recursively repeated until each quadrant contains four or fewer items (i.e., if a quadrant contains more than four items it is divided in four quadrants again). For the example shown in Table 1, this process generates four quadrants labelled A, B, C and D with cumulative weights 12 (26%), 11 (24%), 13 (28%) and 10 (22%).

TABLE 1
Ten numbered items and their associated values and quadrants

item #	1	2	3	4	5	6	7	8	9	10
value	5	5	2	8	3	2	2	3	6	10
quadrant	A	A	A	B	B	C	C	C	C	D

Step 2: Lay out the quadrants.
The quadrants created in Step 1 are sequentially laid out on the screen following the directional rules dictated by the Hilbert/Moore curve (see Fig. 7a), starting with the highest level quadrants, and then the quadrants within these quadrants (if applicable).

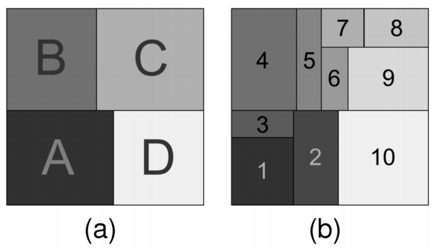

Fig. 7. Generation of a Hilbert treemap: (a) layout of the quadrants, (b) layout of the items.

Step 3: Lay out the items.
When the actual items are laid out (not the quadrants), aspect ratio optimising is applied by comparing several candidate layouts if a quadrant contains three or four items. These candidate layouts all maintain the underlying data order and place the first and the last item in adjacent corners. When a quadrant contains three items, the algorithm compares the aspect ratio of four layouts shown in Fig. 8. When a quadrant contains four items, three additional layouts are evaluated (see Fig. 9).[1] The layout with the lowest mean aspect ratio is used. The layout is then placed such that the underlying ordering of the whole treemap is maintained; the first item neighbours the last item in the previous quadrant and the last item neighbours the first item in the next quadrant. The resulting treemap is shown in Fig. 7b. Figures 1c and d show larger Hilbert and Moore treemaps.

1. There are more possible layout options than these seven, but using more layout options than the current seven did not result in a noticeable improvement of the aspect ratio of the layout.

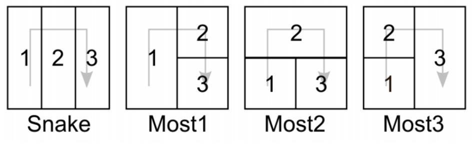

Fig. 8. Four layout options when the quadrant has three items: Snake, Most1, Most2 and Most3.

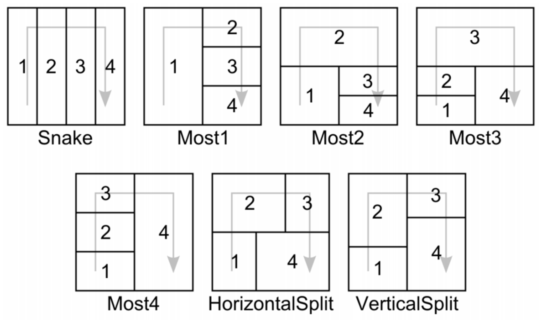

Fig. 9. Seven layout options when the quadrant has four items: Snake, Most1, Most2, Most3, Most4, HorizontalSplit and VerticalSplit.

6 Metric-based treemap comparison

We used the metrics described in Sections 2.2 and 3 to compare the following treemaps: Hilbert and Moore, slice and dice [11], squarified [12], strip [4], spiral, [2], and the three variants of pivot algorithm (pivot by size, pivot by middle and pivot by split size) [9].

Our method, detailed below, synthetically generates a large sample of treemaps. Large samples of treemaps are required to reduce the risks of random sampling producing an unrealistically favourable result for any particular treemap. The stability metrics are calculated for dynamic updates and item additions, which simulates changes to the underlying data. By simulating these events we can determine how the spatial properties of various treemaps respond.

6.1 Method

All metrics were calculated for treemaps with three different numbers of items: 10, 30, and 50 items. They were calculated for a square display (1:1 aspect ratio).

Three separate simulation batches were run, with each batch containing 300 treemaps for each algorithm (100 with 10 items, 100 with 30 items, and 100 with 50 items). The first batch was used to investigate how the spatial properties of the different treemap algorithms respond to *dynamic updates*. The second batch examined the same issues in the presence of *item addition*.[2] The third *remaining metrics* batch examined aspect ratio, continuity, and readability.

The items in each batch at each size were assigned values according to a Zipfian distribution (α=1), then randomly ordered: for example, a ten item dataset might be {*9, 21, 32, 12, 6, 7, 64, 16, 8, 10*}.

Dynamic Updates. Each of the 300 treemaps in the *dynamic update* batch was analysed through a series of 100 data updates (a method previously used in [2], [4]). At each update, the values in the treemap dataset were modified by multiplying each item by e^x, with x randomly drawn from a normal distribution with mean 0 and standard deviation 0.05. The resultant state of the treemap was then calculated by the treemap algorithm. The stability metrics (distance change and location drift) were calculated after every update.

Item Addition. For each of 300 treemaps in the *item addition* batch the distance change and distance change variance metrics were determined as follows. Having established 300 initial treemap states using the same procedure as *dynamic updates*, a single random item was inserted into each treemap (random location in the dataset and random size between the minimum and maximum of the dataset). The new state of each treemap was calculated by the treemap algorithm, and the metrics were calculated.

Remaining Metrics. Aspect ratio, continuity and readability were calculated by establishing 300 treemaps using the same procedure as *dynamic updates* and *item addition*, and the metric values calculated.

6.2 Results

6.2.1 Dynamic updates

Fig. 10 shows the results of the comparison of different treemaps on the distance change and location drift metrics when data is updated.

The first row of Fig. 10 shows that the average distance change of Hilbert and Moore is lower than squarified and pivot by split size, but higher than slice and dice and pivot by middle.

The second row of Fig. 10 shows that distance change variance of Hilbert and Moore is lower than all other treemaps except slice and dice. Also, the distance change variance of Hilbert and Moore is similar to that of pivot by middle, which has a lower value for the distance change metric.

The third row of Fig. 10 shows that location drift of Hilbert and Moore is lower than squarified, strip, spiral, pivot by size and pivot by split size. Also, similar to distance change, the location drift of Hilbert and Moore is higher than slice and dice and pivot by middle.

Finally, the fourth row of Fig. 10 shows that location drift variance of Hilbert and Moore is lower than all other treemaps except slice and dice.

6.2.2 Item addition

Fig. 11 shows the results of the comparison of different treemaps on the distance change metrics when an item is added to the layout.

The first row of Fig. 11 shows that distance change of Hilbert and Moore is lower than pivot by middle and

2. We observed that the effect of item *addition* on layout stability is the same as the effect of item *deletion*.

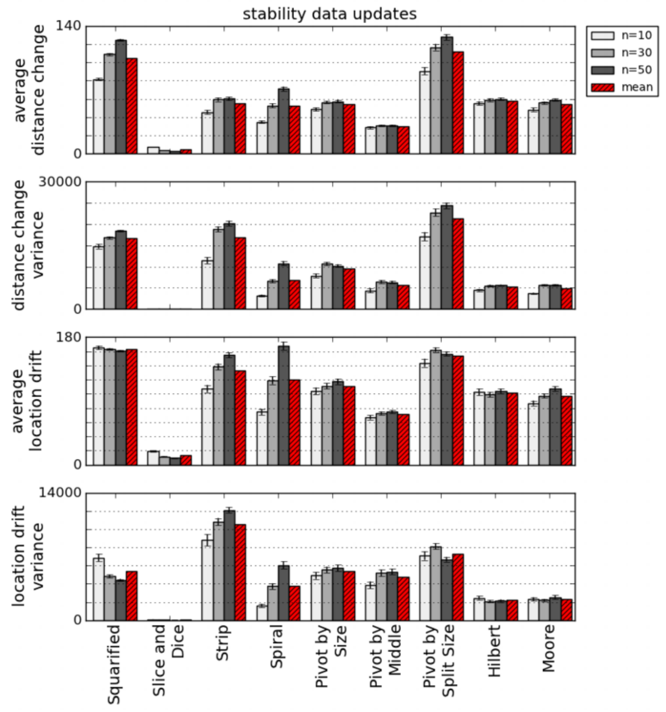

Fig. 10. Distance change and location drift metrics for data updates. Error bars represent +/- 1 SE.

by split size, but higher than slice and dice and pivot by size.

The second row of Fig. 11 shows that distance change variance of Hilbert and Moore is lower than most treemaps, but higher than slice and dice.

Most treemaps, including Hilbert and Moore, perform relatively similar in terms of stability when data is updated and when an item is added, but two layouts exhibit very different behaviours across these situations. The squarified treemap is quite unstable when the data is updated, but stability is not severely affected when an item is added. For the pivot by middle treemap the reverse is true: it is very stable when the data is updated, but not when an item is added.

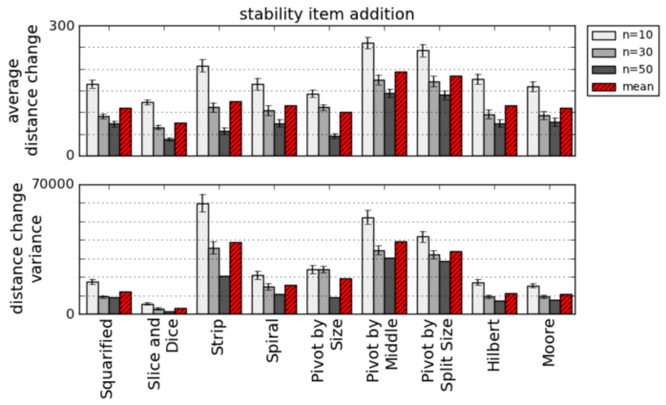

Fig. 11. Distance change metrics for item addition. Error bars represent +/- 1 SE.

TABLE 2
Stability scores and ranks for all treemaps. Lower scores mean better stability.

Treemap	Stability Score	Stability Rank
Slice and Dice	0.00	1
Moore	0.33	2
Hilbert	0.35	3
Spiral	0.42	4
Pivot by Size	0.45	5
Pivot by Middle	0.56	6
Squarified	0.63	7
Strip	0.75	8
Pivot by Split Size	0.90	9

6.2.3 *Stability score*

In total, six stability metrics were calculated for each of the three set sizes (see Figures 10 and 11). To determine how well the different treemaps perform *overall* in terms of stability a 'stability score' was calculated by linearly normalising the range of each stability metric (scaling values between 0 and 1) and averaging them for each treemap. The stability scores for all treemaps are shown in Table 2.

Table 2 shows that the Hilbert and Moore treemaps have the best (i.e., lowest) stability scores after the slice and dice treemap. The slice and dice treemap performs particularly well on the various stability metrics because it uses a simple linear order, and therefore does not suffer from large changes in the layouts when the underlying data is changed. However, the slice and dice treemap is not an attractive option for 'real life' use because of the very high aspect ratios of the items (see next section), which makes items hard, or even impossible, to label and select [9].

6.2.4 *Remaining metrics*

Fig. 12 shows the results of the comparison of different treemaps on the three other common treemap metrics: aspect ratio, continuity and readability.

The Hilbert and Moore algorithms create treemaps with aspect ratios that are similar to most other treemap algorithms. The top row of Fig. 12 shows that squarified performs particularly well (mean aspect ratio 1.3) and that slice and dice performs particularly poorly. All other algorithms produced mean aspect ratios in the range 1.8 to 2.8.

The second row of Fig. 12 shows that the readability of Hilbert and Moore treemaps is lower than slice and dice, strip, and spiral, but similar to pivot layouts. The low readability of the Hilbert and Moore treemaps stems from the localised reorientation ('crumpling') of Hilbert and Moore curves. However, this poor performance on the readability metric might not be detrimental for user performance because the layout is very stable. Stable layouts, such as Hilbert and Moore treemaps, eliminate the need to perform such a (slow) linear search as

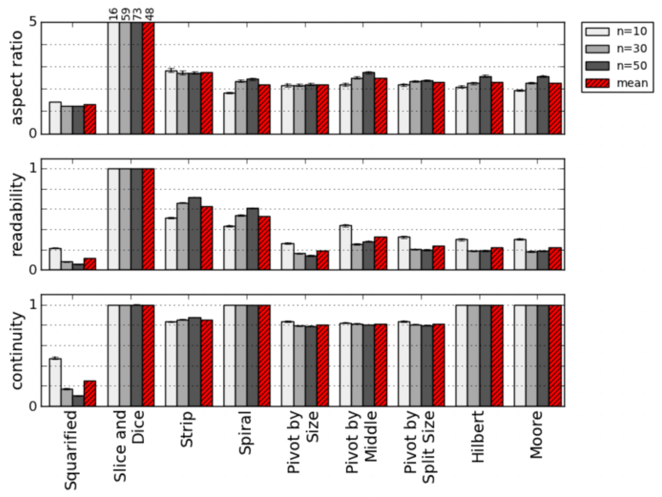

Fig. 12. Aspect ratio, readability, and continuity metrics. Error bars represent +/- 1 SE.

encapsulated by the readability metric, because users learn where items are located.

The third row of Fig. 12 shows that Hilbert and Moore treemaps have high continuity, as do slice and dice and spiral algorithms. Readability and continuity metrics both concern the ease with which a treemap visualisation can be visually scanned, so the disparity of results for Hilbert and Moore treemaps (poor for readability, best for continuity) indicates how relatively subtle differences in metric implication can have a substantial influence on metric-based conclusions. This is due to the readability metric being tailored for predominantly linear scanning, while the continuity metric predominantly concerns localised scanning.

6.2.5 Results summary

The results of the simulations show that Hilbert and Moore treemaps have the best overall performance in terms of stability after the slice and dice treemap. This is explained by the good stability of the Hilbert and Moore treemaps both when data is updated and when an item is added to the layout. Combined with their low aspect ratio and good continuity this suggests that Hilbert and Moore treemaps can be very useful in 'real life' applications, particularly those where layout stability is important. We note that Hilbert and Moore treemaps are very similar, not only visually (see Fig. 1), but also in terms of their metrics.

7 Conclusion and future work

We introduced a 'location drift' treemap metric and empirically demonstrated that it can explain some effects on user performance that are not captured by the commonly used distance change metric. We also proposed Hilbert and Moore treemaps, and demonstrated that they have a low mean aspect ratio, maximal continuity, and good stability both when the data changes and when the number of items changes.

Results of the empirical experiments (which used abstract tasks) and of the metric-based theoretical analyses suggest that Hilbert and Moore treemaps should be considered by designers seeking a spatially stable visualisation. However, further work is needed in broadening their performance analysis in 'real life' applications. Future work could also focus on comparing the various treemap algorithms for hierarchical data sets, as some metrics will differ for such data sets.

References

[1] B. Johnson and B. Shneiderman, "Tree-Maps: a space-filling approach to the visualization of hierarchical information structures," in *Proc. VIS '91*. IEEE, 1991, pp. 284–291.

[2] Y. Tu and H. Shen, "Visualizing changes of hierarchical data using treemaps," in *IEEE Trans. on Visualization and Computer Graphics*, 2007, pp. 1286–1293.

[3] M. Wattenberg, "Visualizing the stock market," in *Proc. CHI '99 extended abstracts*. ACM, 1999, pp. 188–189.

[4] B. B. Bederson, B. Shneiderman, and M. Wattenberg, "Ordered and quantum treemaps: making effective use of 2D space to display hierarchies," *ACM Trans. Graph.*, vol. 21, no. 4, pp. 833–854, 2002.

[5] B. Engdahl, M. Köksal, and G. Marsden, "Using treemaps to visualize threaded discussion forums on PDAs," in *Proc. CHI '05 extended abstracts*. ACM, 2005, pp. 1355–1358.

[6] K. Shi, P. Irani, and B. Li, "An evaluation of content browsing techniques for hierarchical space-filling visualizations," in *Proc. InfoVis '05*. IEEE, 2005, pp. 81–88.

[7] J. Stasko, "An evaluation of space-filling information visualizations for depicting hierarchical structures," *Int. J. Hum.-Comput. Stud.*, vol. 53, no. 5, pp. 663–694, 2000.

[8] T. Asahi, D. Turo, and B. Shneiderman, "Visual decision-making: using treemaps for the analytic hierarchy process," in *Proc. CHI '95*. ACM, 1995, pp. 405–406.

[9] B. Shneiderman and M. Wattenberg, "Ordered treemap layouts," in *Proc. InfoVis '01*, 2001, pp. 73–78.

[10] H. Sagan, *Space-Filling Curves*. Springer, 1994.

[11] B. Shneiderman, "Tree visualization with tree-maps: 2-d space-filling approach," *ACM Trans. Graph.*, vol. 11, no. 1, pp. 92–99, 1992.

[12] M. Bruls, K. Huizing, and J. v. Wijk, "Squarified treemaps," in *Proc. Joint Eurographics and IEEE*. IEEE, 2000, pp. 33–42.

[13] J. J. Van Wijk and H. van de Wetering, "Cushion treemaps: Visualization of hierarchical information," in *Proc. InfoVis '99*. IEEE, 1999, pp. 73–78.

[14] M. Wattenberg, "A note on space-filling visualizations and space-filling curves," in *Proc. InfoVis '05*. IEEE, 2005, pp. 24–29.

[15] M. Balzer, O. Deussen, and C. Lewerentz, "Voronoi treemaps for the visualization of software metrics," in *Proc. SoftVis '05*. ACM, 2005, pp. 165–172.

[16] R. Niedermeier, K. Reinhardt, and P. Sanders, "Towards optimal locality in mesh-indexings," in *Proc. Symp. Fundamentals of Computation Theory*. Springer-Verlag, 1997, pp. 364–375.

[17] N. Kong, J. Heer, and M. Agrawala, "Perceptual guidelines for creating rectangular treemaps," *IEEE Trans. on Visualization and Computer Graphics*, vol. 16, pp. 990–998, 2010.

[18] M. E. J. Newman, "Power laws, Pareto distributions and Zipf's law," *Contemporary Physics*, vol. 46, no. 5, pp. 323–351, 2005.

Susanne Tak Susanne Tak is a researcher interested in human-computer interaction, cognitive psychology, human problem solving and data visualisation. She completed her PhD at the Department of Computer Science and Software Engineering of the University of Canterbury, Christchurch, New Zealand in December 2011.

Andy Cockburn Andy Cockburn is a computer scientist with interests in understanding and modelling human performance with interactive systems. He is a full professor in the Department of Computer Science and Software Engineering of the University of Canterbury, Christchurch, New Zealand.