Enhanced Hexagon-Tiling Algorithm for Map-Like Information Visualisation

Muye Yang
Dept. of Computer and Information Science
Faculty of Science and Technology
University of Macau
Macau S.A.R., China
muye.yang@gmail.com

Robert P. Biuk-Aghai
Dept. of Computer and Information Science
Faculty of Science and Technology
University of Macau
Macau S.A.R., China
robertb@umac.mo

ABSTRACT

Hierarchical information is ubiquitous, and analysing such information is one of the common tasks performed by its users. Map-like visualisations, which resemble geographic maps, can be a useful tool for perceiving hierarchical information. Maps have the advantage that the majority of ordinary users can easily understand them without prior training. This paper presents a novel method for visualising hierarchical data in a form resembling a geographic map, using hexagon tiling. Our method can be used for visualising general hierarchical data, such as organisational structures, library catalogues, file systems, and many more. Like treemaps, our new method is area-based, thus it not only reveals hierarchy but also other attributes. Unlike treemaps, however, our method produces visualisations that resemble geographic maps in shape and appearance. Several tactics are employed to control the space-division process and aspect ratio. In this paper we present experiments involving our visualisation, evaluations of results, comparisons with other methods, and discuss advantages and potential issues.

Categories and Subject Descriptors

I.3.8 [**Computer Graphics**]: Applications

Keywords

Information visualisation, hierarchy, treemap, geographic map

1. INTRODUCTION

Since the wide-spread use of the computer and the Internet, creating and exchanging information has been greatly facilitated, and human capability to process and manage information has increased enormously. Over the past two decades, the emergence of web technologies and social media have created huge amounts of content such as email, news websites, social media posts and many others. People are exposed to various types of data on a daily basis, and frequently have to perform tasks such as analysing data and making decisions based on these data. Among them, hierarchical data such as are used in information catalogues, website navigation structures, or organisation structures, is of special interest to us because of their ubiquity.

Information visualisation takes advantage of the human visual ability, creating a meaningful graphic representation from abstract data to allow users to perceive and assimilate large amounts of information at once. An everyday example of this are geographic maps.

Many visualisations have been devised over the years and proven to be effective. However, many of these visualisations are designed for use by experts and require prior training. Usability issues arise when the majority of a visualisation's audience are non-professional users without prior training.

Suitable metaphors that connect with users' real-world experiences facilitate understandability of a visualisation. This is the case with methods that visualise hierarchical and other relational data in the form of a geographic map. Maps consist of land and sea, of countries, provinces and counties that are separated by borders and that are labelled with their name. Maps are usually also coloured to represent a certain aspect of the terrain, such as elevation in the case of topographic maps, or another attribute such as economic output, unemployment rate, language spoken or others in the case of a thematic map.

When the map form is used to represent non-geographic data, clusters of data can be represented as regions in the map, and attributes of this data can be mapped to visual attributes of map objects. The use of the map form has the significant advantage that most people understand geographic maps easily thanks to early exposure to maps in school. Even among preschool children essential mapping abilities are well developed [7]. Elements such as mountains, valleys, land, sea, rivers, and cities, as well as the meaning of each, are readily recognized by people even without special training. Therefore visualising information structures in the form of a geographic map enables people to relate to such representations more easily without requiring prior instruction.

A number of attempts have been made to create visualisations based on the metaphor of maps, for instance the treemapping family of visualisations which was devised more than 20 years ago [13] and has spawned many extensions. Recently, Biuk-Aghai et al. devised several methods to visualise hierarchical data with the appearance of a geographic map [5, 6]. However, issues remain with the performance of their algorithms, the complexity of their methods, and their ability to correctly reflect size attributes of the input data in the visualisation.

To address these issues, we have devised a new method as an extension of our previous work [5, 6]. At its core is the Enhanced Hexagon Tiling Algorithm (EHTA). The algorithm creates maps from hierarchical input data, and uses simple back-tracking to resolve conflicts and recover from failure. In addition, some tactics

VINCI '15 August 24–26, 2015, Tokyo, Japan
 ISBN 978-1-4503-3482-2…$15.00
DOI: http://dx.doi.org/10.1145/2801040.2801056

are used to enforce an ideal aspect ratio of the final result, and also to greatly reduce the chance of tiling failure, thereby improving performance.

In this paper we present our new method. In Section 2 we briefly introduce existing visualisations for hierarchical data and the previous work that our new method is based on. Section 3 gives a detailed description of the principles and implementation of our new method. Sections 4 and 5 then present an evaluation and discussion of our method, respectively, and finally we give conclusions in Section 6.

2. RELATED WORK

Metaphors are widely used in the field of visualisation and software interface design. They relate concepts that are difficult to comprehend to things that users are familiar with. In the case of hierarchical information visualisation, many methods are based on the metaphors of map and tree.

Many early methods are based on node-link diagrams. Even in early uses of computers, developers created node-link diagrams to represent information such as directory structures. Cone trees [18] present data as a node-link diagram with the root at the top and its children attached to its bottom, thereby appearing like a cone. Cone trees embed the result in a three-dimensional space, requiring a complex user interface and 3D animation. Hyperbolic Trees [15] are similar to cone trees, but preserve a two-dimensional layout and arrange child nodes in a circle centric to the root in a hyperbolic plane, making them more space-efficient. Bertin [4] offers a similar idea using a radial layout and shrinking the size of the nodes with their distance from the root, but the use of hyperbolic geometry is more elegant. The fractal approach of Koike and Yoshihara [14] provides a similar technique to Hyperbolic Trees. There are many other methods that are modified or improved based on these, and are similar in visual appearance. One of the disadvantages of these methods is that they cannot encode information other than the hierarchy.

In addition to node-link diagrams, area-based visualisations are also widely used. Mosaic plots [10], for example, use rectangular tiling to display joint distributions. Another famous method, the Treemap [13], introduced by Johnson and Shneiderman, was originally designed to visualise disk space usage. Compared with the tree algorithms, Treemaps have the advantage that they can not only visualise the hierarchy of nodes, but also their sizes. Treemaps are more space-efficient than other methods. They fully utilize all the given space by representing the data as nested rectangles. Treemaps have many applications in various areas [21, 24, 12], as well as numerous variants, such as Ordered Treemaps [19], Squarified Treemaps [8], Mixed Treemaps [23], Cushion Treemaps [22], and others.

Some algorithms use shapes other than rectangles, for example Jigsaw Treemaps [25] and Convex Treemaps [17]. Voronoi Treemaps [2, 3, 16] use arbitrary polygons instead of rectangles, laying them out by the iterative relaxation of Voronoi tessellations.

Despite their many merits and wide use, these algorithms create trees and maps that differ considerably from the trees and maps we normally see. This is not a big problem because these visualisations are usually intended for professional users. However, with the popularisation of web-based data and analytics, the demand for visualisations usable by non-professional users without training is increasing. Given that geographic maps represent hierarchic data, and that even preschool children are able to read maps [7], the map form is an attractive representation for easy-to-use visualisations.

Skupin has used the map form to represent the geographic knowledge domain [20]. Taking 2200 conference abstracts, he applied the Self-Organizing Map (SOM) method to produce a map-like representation of the topics contained in this document collection. Topics were clustered hierarchically, and this hierarchy was mapped to areas in the map. The final map output and labelling were produced by a GIS (geographic information system) software. Although Skupin's visualisations are map-like, the visual size of regions does not correspond to any given size attribute.

Hu et al. have devised another method for representing dynamic relational data in map form [11]. Their approach, called GMap, transforms an arbitrary graph into a map representation. It represents clusters of nodes as areas in the map, and displays most areas fused together into one large continent, with only a few small separate islands, if any.

Gronemann and Jünger have visualised clustered graphs in the form of topographic maps [9]. Their input is a graph with an overlaid tree, expressing both relationships and hierarchies among nodes. This graph data is preprocessed to determine node placement, from which a triangle mesh is generated that assigns an elevation to each node. The result data is fed into a GIS software that outputs a map showing islands of areas whose elevation is above sea level, whereas other areas that have a negative elevation appear as submerged below the sea.

Auber et al. have proposed another method that produces visualisations resembling a geographic map, which they term GosperMap, as their method relies on the use of a Gosper curve [1]. The input data has a tree structure, the leaves of which are projected to a 2D space-filling curve. Finally, regions containing nodes are filled to become areas corresponding to sub-trees in the input tree. The resulting maps consist of a single continent with countries that have highly irregular borders.

Biuk-Aghai and Pang have devised a different method to create map-like visualisations [6]. Their method takes a tree of hierarchical data and constructs a map that represents nodes at different levels as nested areas, reflecting the hierarchy. Areas are constructed by tiling hexagons in a hexagon matrix, starting from the area's center and moving outwards in random order.

Biuk-Aghai and Ao have created a different method of map-like visualisation based on expanding polygons instead of hexagon tiling [5]. It uses the metaphor of liquids poured onto a surface, where each liquid expands outwards until the column of liquid reaches a minimal height. In this approach, areas are constructed as 100-sided polygons where each vertex can expand outward until it touches a neighbouring polygon. As a result, areas in this visualisation often have rounded edges.

Node-link diagrams are widely used in many applications, but suffer the disadvantage that they usally only represent the hierarchy. Area-based visualisations such as treemaps can not only present the hierarchical structure through nesting relationships, but also encode one more dimension into area size. This makes them suitable for wider applications. However, these visualisations are not so intuitive and require some learning before users are able to make sense of them. Existing map-like visualisations also have their shortcomings. Skupin's method cannot represent a numeric attribute as area size. Gronemann and Jünger's method has an additional dependency on GIS software. Auber et al. always generate a map consisting of only one continent. We devised our new method to address all these issues. It uses a simple and effective algorithm to layout hierarchical, two-dimensional data, and has no dependency on other software packages.

3. VISUALISATION METHOD

This section discusses our new method for visualising hierarchical data in a form resembling a geographic map. The complete

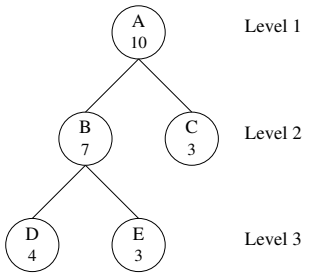


Figure 1: Example hierarchy

process consists of following three main steps. (1) Preprocessing: we retrieve the input data and create a tree representation from it. Transforming and pruning operations may be necessary to ensure constraints required by our algorithm. (2) Layout: we run our layout algorithm, transforming the tree to a map representation. This consists of a set of regions, each of which corresponds to a tree node, with the region size corresponding to a size attribute of the node. The layout uses hexagon tiling to create the map, as this is simple and effective. (3) Decoration: colours and boundaries are added to enrich the information content of the visualisation, which also makes it appear more like a geographic map.

3.1 Hierarchical Data Model

Our method has a few requirements on the data to be visualised. The data needs to be in the structure of a directed tree with a single root node, and each node needs to have a size attribute with a positive value. The size of a branch node must equal the sum of all its children's sizes. This corresponds to the hierarchy of regions in a geographic map: at the root is the whole world, below which are nodes for all the countries, each of which consists of nodes for provinces or states, etc. Each higher-level region's size is the sum of its sub-regions' sizes. We shall take the hierarchy represented by Figure 1 as an example in the following explanation. It consists of five entities, A, B, C, D, E. Out of these, B and C are sub-entities of A, and D and E are sub-entities of B. It is easy to see that the size of B equals the sum of the sizes of D and E which are B's children.

In practice, we may ignore the size of branch nodes, populating their size attributes from the sum of the sizes of their descendent leaf nodes. Leaf nodes of zero size (if any) are discarded. To improve the performance of certain queries, such as the lowest common ancestor (LCA), optimized data structures and algorithms can be used for the tree representation.

3.2 Layout Algorithm

Our method uses a space-filling hexagon tiling approach. To imitate the irregular appearance of geographic maps, we use randomness in the tiling process. However, in order to be able to reproducibly create the same output from a given input, we fix the random seed.

Each node in our input tree is represented by a set of hexagonal tiles in our visualisation. We use a hexagon grid of unbounded size, able to shrink and grow as necessary, which always has enough space to accommodate the whole map.

To simplify the following explanation, from now on we assume we are using geographic data as our input, and we refer to nodes as entities. For example, the first level nodes or entities represent countries, second level nodes represent provinces, and so on. Each tile represents a tract of land and can be claimed by at most one entity, i.e. node.

Our algorithm involves several frequent operations. One of them is calculating the lowest common ancestor (LCA) of a node. The LCA, $lca(v,w)$, of two nodes v and w is defined as the lowest node u that has both v and w as its descendants. We define each node as a descendant of itself. In this way, the LCA of any two nodes in a tree is defined. For instance, in Figure 1, $lca(D,C) = A$, and $lca(D,E) = B$.

In the next step, we claim tiles for each node. Tiles are claimed one at a time, and only from among the unclaimed tiles adjacent to the area of an entity. This process is performed recursively from the root down. Taking the example hierarchy of Figure 1 we start with the root A. To lay out the root, we proceed to lay out each of its children B and C, and for B we lay out its children D and E. Because a sub-entity is a part of another entity, the tiles claimed by D and E are also claimed by B. As a result, in order to lay out the tree of Figure 1 we only need to process its three leaf nodes D, E, C instead of all of its five nodes A, B, C, D, E.

Tiling mainly mainly involves one operation: selecting a tile to claim. For this we distinguish two cases, selecting the first tile and selecting the remaining tiles.

Selecting the first tile.

To begin claiming tiles for an entity we need to decide where to start. As we constrain tiling to select only adjacent unclaimed tiles for an entity, this implies that a sub-entity must also be adjacent to another sub-entity that belongs to the same entity. Our algorithm, EHTA, first checks for each entity if its parent has claimed any tiles. If it has, then it checks if there is any unclaimed tile adjacent to it. If there is, then the first tile to be claimed is chosen from it, otherwise, it checks its grand parent, and so on. This process will stop at the root. The root must have claimed some tiles, because all tiles of its sub-entities are also part of itself. If the algorithm has just started and no tiles have been claimed yet, then we either manually or randomly specify a starting tile as we are certain that no tiles have been claimed and thus there can be no clash.

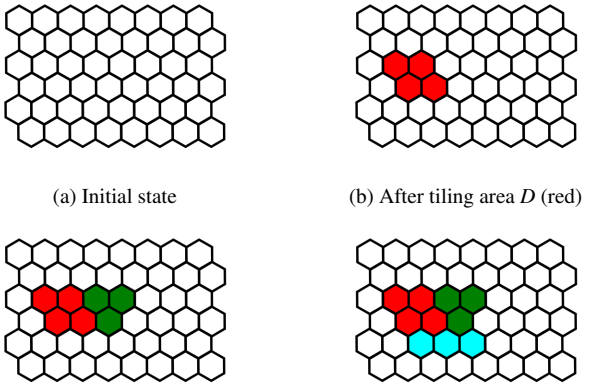
We use the example hierarchy from Figure 1 to illustrate this, with the process shown in Figure 2. We start with no tiles being claimed and let D claim tiles. We find that its parent B and grand parent A have not claimed any tiles. Because A is the root, we know all tiles are unclaimed, and we simply select a random first tile for D to start. After D is finished, we continue by letting E claim tiles. At this time, we find that D's parent B has already claimed some tiles because its child D has claimed some tiles and there are unclaimed ones around it. So the first tile for E is selected from this area, and E will be adjacent to D.

Selecting the remaining tiles.

Once the first tile has been selected, selecting the remaining tiles is straightforward. The only constraint is to make sure that the next tile should be adjacent to the tiles that have already been claimed so that the resulting area is contiguous. For a small number of nodes, the next tile can be randomly selected and the result has a natural ragged appearance like geographic boundaries. But when the number of nodes is large, random selection tends to lead to too many holes, which also causes another problem explained later.

To eliminates holes to the extent possible while preserving the natural ragged appearance, the strategy that EHTA adopts is to employ an estimating function. This function works as follows.

In the first step, it gathers a list of eligible tiles, i.e. unclaimed tiles adjacent to the entity. In the second step, for each eligible tile it counts how many among all its neighbouring tiles are already claimed. If a tile is surrounded by many claimed tiles, it is more likely to be a hole. If it is surrounded by 6 claimed tiles, then it definitely is a hole. Let this number be n. In the next step, we



(a) Initial state (b) After tiling area D (red)

(c) After tiling area E (green) (d) After tiling area C (blue)

Figure 2: Tiling of the example hierarchy involving entities A,B,C,D,E. Only leaf nodes D,E,C need to be tiled as the area of branch nodes A and B is equal to that of their child nodes.

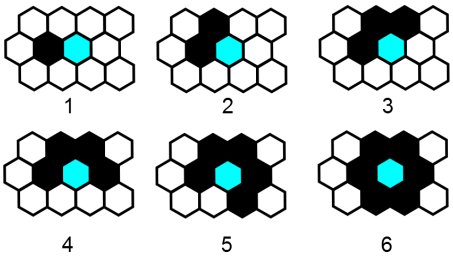


Figure 3: Selecting tiles to occupy

calculate the score s_n of node n by the formula $s_n = b^n$, where b is a constant. At last, we randomly select one tile based on its score. The probability of one tile getting selected is directly proportional to the score. If 4 is chosen for b, the chance of a tile surrounded by 6 claimed tiles getting selected is four times as large as for a tile surrounded by 5 claimed tiles. Figure 3 illustrates this: starting from the blue tile in the centre, neighbouring tiles are occupied one by one.

Choosing a larger value of b results in fewer holes and more compact and regular shapes, as shown in Figure 4.

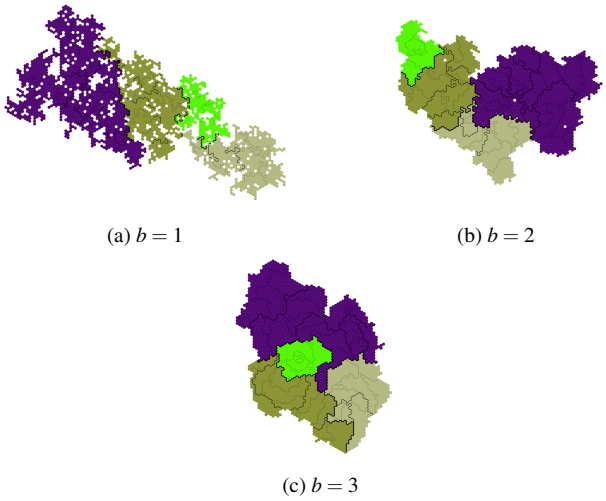
Backtracking.

Sometimes it happens when claiming tiles that after choosing a starting tile, there is not enough space to accommodate the whole entity, for example if the starting position happens to be a hole. This problem is solved through backtracking. Our EHTA algorithm keeps a list of attempted start positions. Once an entity fails to claim the number of tiles it requires, it will give up all claimed tiles (effectively performing a *rollback* of all its already performed tiling), and re-attempt tiling at another position.

By choosing a good value for constant b, the likelihood of having to backtrack is generally low.

Adding separating space.

Our layout method allows areas of entities to be separated by some amount of space. This is akin to land separated by sea in a map. Doing so can enhance readability as the boundaries of separate areas become more easily discernible. To achieve this visual separation we employ transparent tiles. Moreover, we can control at which level of the input data hierarchy this visual separation should be performed. For example, if we require separation at level 2 then all level 2 areas are laid out with separating space in-between each other, but all level 3 areas within the same level 2 area are tiled without any separating space between each other.



(a) $b = 1$ (b) $b = 2$

(c) $b = 3$

Figure 4: Comparison of results with different values of b, using random input data

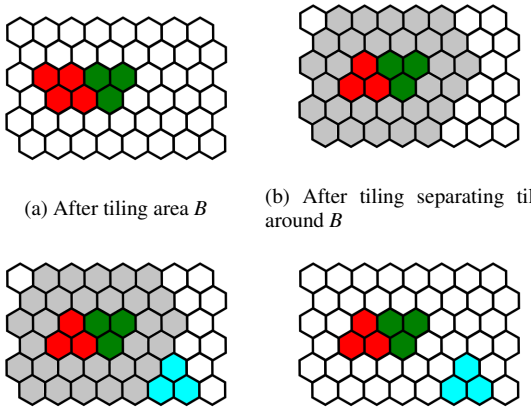
To add separating space, our layout method performs an additional step. After the tiling of an area at the required level of separation is complete, we claim "sea" tiles to surround its "land" tiles. To illustrate this process we use the example hierarchy of Figure 1 and assume that we decide to separate at level 2, i.e. each of areas B and C should be separated, and that the amount of spacing should be at least two tiles wide. We now perform the recursive algorithm as usual to lay out the first area, in this case B, consisting of areas D and E, as shown in Figure 5 (a). At this time the tiling of B is finished and the tiling of C is next. Because B and C are level 2 entities, and we decided to separate at level 2, we need to add separating tiles between them. To do this, we simply claim all unclaimed tiles within 2 units of B, shown as the gray tiles in Figure 5 (b). Because this area is now claimed as separating tiles, it cannot be claimed by other areas as their regular area tiles. Then we tile area C adjacent to the area occupied by B and its separating tiles, as shown in Figure 5 (c). In the final result the separating tiles appear transparent, as shown in Figure 5 (d).

3.3 Decoration

The final step in our visualisation method is to decorate the map. The previous step has output a layout consisting of sets of tiles representing nodes in our input data set. Before rendering these areas on the screen they are decorated: areas are filled with background colours, down to the desired level (e.g. in Figure 4 the colouring is at the level of countries); borders between areas are drawn with different thickness and/or colour depending on the type of border (e.g. Figure 4 shows three kinds of borders: national, provincial, and county borders, with decreasing thickness); and labels are added to areas, producing a final result map.

4. EVALUATION

To evaluate our method we implemented it in a prototype visualisation application. This application was written in Java (using JDK1.8), implementing all parts of our method described above.



(a) After tiling area *B*

(b) After tiling separating tiles around *B*

(c) After tiling area *C*

(d) Final result

Figure 5: Separating areas at level 1 (areas *B* and *C*)

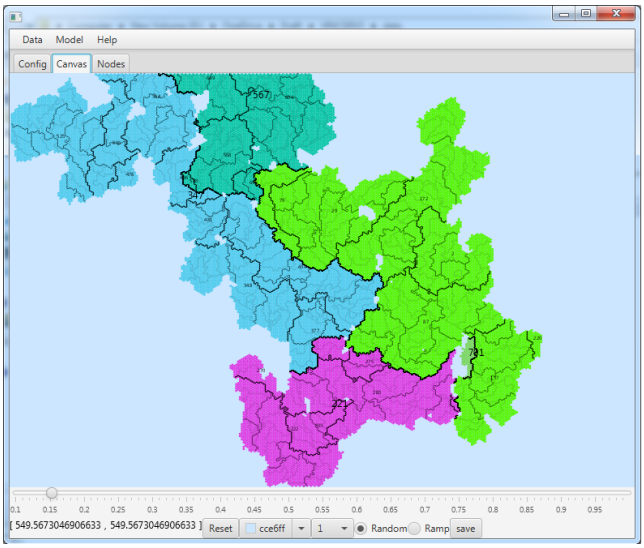


Figure 6: Visualisation prototype

Demo Program.

We developed an interactive application (Figure 6) to test and diagnose our method. This program includes a GUI written in Java. It allows us to generate random data of different size according to given parameters. The data is then fed to the layout algorithm and the result rendered on demand. The map can also be zoomed and panned for navigation, and the background colour and colour scheme can also be adjusted. Moreover, we can export the result to an image file in bitmap format.

Environment.

Our evaluation tests were run on a desktop computer with Intel i7-4770 CPU and 8GB RAM, running the Windows 7 operating system.

Dataset.

We generated random datasets of different size. We use three parameters to control the generation process: height, width and size. Height controls the maximal height of the tree, width the maximal number of children of a node, and size the number of tiles that should be allocated for each node. We recorded the time it takes to run the layout algorithm for each dataset. This time does not include the time required to generate the dataset or to generate the image. The result is shown in Table 1.

Table 1: Evaluation data and performance

Nodes	Height	Width	Size	time (ms)
137	3	7	100	404
155	2	10	100	419
569	3	10	100	1099
569	3	10	200	1889
2272	4	10	100	2757
1744	3	12	100	2183
4627	3	16	100	8818

5. DISCUSSION

Our new method was inspired by and improves on Biuk-Aghai et al.'s previous work [6]. The new method is superior to its predecessor and other methods in several aspects.

First, our new method makes use of the area size to correctly represent the input data's size attribute. This provides an intuitive way to make sense of the magnitude of data, which node-link diagrams such as hyperbolic trees do not show. Like the treemap, the size of a node is preserved during layout and is proportional to the data's size attribute. Biuk-Aghai et al.'s previous method also attempts to represent a size attribute as the corresponding area's size, but it does not guarantee that the area size is always accurate, and in some cases it may fail if an area that is surrounded on all sides by other areas is unable to expand further. Our new method does not suffer from this problem as it makes use of backtracking whenever it runs out of space. Our new method has a small limitation: as the size attribute of an entity maps to the number of tiles allocated to its area, that size attribute must be an integer, i.e. if it is a real number it needs to be rounded up to the nearest integer, resulting in a small loss of accuracy. A related issue is when the size attribute is very large, such as in the millions or billions, the performance is heavily affected when laying out that many tiles. In these cases scaling down the size attribute can effectively solve this issue, however this may result in a small loss of precision due to rounding errors.

Second, our new method resembles geographic maps in appearance. Unlike the treemap which comprises only rectangles, our visualisation has irregular shapes and ragged borders, which makes it appear more like a geographic map. Users can therefore more easily relate to our visualisation without any special instruction, taking advantage of their pre-existing knowledge and experience of reading geographic maps.

Third, the performance is reasonable. It our test environment it takes about 2 milliseconds to lay out a node with an average size of 50 tiles, which is acceptable for real-time visualisation. Drawing the map on a screen takes considerably more time. However, in practice we find that an interactive approach that only displays a small set of nodes and enables users to zoom in and out is both attractive and efficient.

6. CONCLUSION

We have presented a new method for visualising hierarchical data in a form resembling a geographic map. We demonstrated and evaluated this method with a Java program and attested its viability. The method has several advantages and has the potential to

be used for non-professional users in perceiving hierarchical data. Potential applications include exploring organizational structures, library catalogues, computer disk usage, and many others. Our research is a work in progress and is being actively developed, including through user evaluations that will allow us to identify areas of further improvement.

Acknowledgements

This research was supported by the University of Macau under grant number MYRG2014-00172-FST.

7. REFERENCES

[1] D. Auber, C. Huet, A. Lambert, B. Renoust, A. Sallaberry, and A. Saulnier. Gospermap: Using a gosper curve for laying out hierarchical data. *IEEE Transactions on Visualization and Computer Graphics*, 19(11):1820–1832, 2013.

[2] M. Balzer and O. Deussen. Voronoi treemaps. In *Proceedings of the IEEE Symposium on Information Visualization 2005*, pages 49–56, 2005.

[3] M. Balzer, O. Deussen, and C. Lewerentz. Voronoi treemaps for the visualization of software metrics. In *Proceedings of the 2005 ACM Symposium on Software Visualization*, pages 165–172. ACM, 2005.

[4] J. Bertin. *Semiology of Graphics*. University of Wisconsin Press, 1983.

[5] R. P. Biuk-Aghai and W. H. Ao. A novel map-based visualization method based on liquid modelling. In *Proceedings of the 6th International Symposium on Visual Information Communication and Interaction*, VINCI '13, pages 97–104, New York, NY, USA, 2013. ACM.

[6] R. P. Biuk-Aghai, C.-I. Pang, and Y.-W. Si. Visualizing large-scale human collaboration in Wikipedia. *Future Generation Computer Systems*, 31:120–133, Feb. 2014.

[7] M. Blades, J. M. Blaut, Z. Darvizeh, S. Elguea, S. Sowden, D. Soni, C. Spencer, D. Stea, R. Surajpaul, and D. Uttal. A cross-cultural study of young children's mapping abilities. *Transactions of the Institute of British Geographers*, 23(2):269–277, 1998.

[8] M. Bruls, K. Huizing, and J. van Wijk. Squarified treemaps. In *Proceedings of the Joint Eurographics and IEEE TCVG Symposium on Visualization*, pages 33–42. Springer-Verlag, 1999.

[9] M. Gronemann and M. Jünger. Drawing clustered graphs as topographic maps. In W. Didimo and M. Patrignani, editors, *Graph Drawing*, volume 7704 of *Lecture Notes in Computer Science*, pages 426–438. Springer Berlin Heidelberg, 2013.

[10] H. Hofmann. Multivariate categorical data – mosaic plots. In *Graphics of Large Datasets*, Statistics and Computing, pages 105–124. Springer New York, 2006.

[11] Y. Hu, S. Kobourov, and D. Mashima. Visualizing dynamic data with maps. *IEEE Transactions on Visualization and Computer Graphics*, 18(9):1424–1437, 2012.

[12] L. Jin and D. Banks. Tennisviewer: a browser for competition trees. *Computer Graphics and Applications, IEEE*, 17(4):63–65, Jul 1997.

[13] B. Johnson and B. Shneiderman. Tree-maps: A space-filling approach to the visualization of hierarchical information structures. In *Proceedings of the 2nd conference on Visualization '91*, VIS '91, pages 284–291, Los Alamitos, CA, USA, 1991. IEEE Computer Society Press.

[14] H. Koike and H. Yoshihara. Fractal approaches for visualizing huge hierarchies. In *Proceedings 1993 IEEE Symposium on Visual Languages*, pages 55–60, Aug 1993.

[15] J. Lamping, R. Rao, and P. Pirolli. A focus+context technique based on hyperbolic geometry for visualizing large hierarchies. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '95, pages 401–408, New York, NY, USA, 1995. ACM Press/Addison-Wesley Publishing Co.

[16] A. Nocaj and U. Brandes. Computing voronoi treemaps: Faster, simpler, and resolution-independent. *Comp. Graph. Forum*, 31(3pt1):855–864, June 2012.

[17] K. Onak and A. Sidiropoulos. Circular partitions with applications to visualization and embeddings. In *Proceedings of the 24th ACM Symposium on Computational Geometry*, pages 28–37, 2008.

[18] G. G. Robertson, J. D. Mackinlay, and S. K. Card. Cone trees: Animated 3D visualizations of hierarchical information. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '91, pages 189–194, New York, NY, USA, 1991. ACM.

[19] B. Shneiderman and M. Wattenberg. Ordered treemap layouts. In *IEEE Symposium on Information Visualization*, pages 73–73. IEEE Computer Society, 2001.

[20] A. Skupin. The world of geography: Visualizing a knowledge domain with cartographic means. In *Proceedings of the National Academy of Sciences*, volume 101 (Suppl. 1), pages 5274–5278, 2004.

[21] D. Turo and B. Johnson. Improving the visualization of hierarchies with treemaps: design issues and experimentation. In *Proceedings of the 3rd conference on Visualization'92*, pages 124–131. IEEE Computer Society Press, 1992.

[22] J. J. Van Wijk and H. Van de Wetering. Cushion treemaps: visualization of hierarchical information. In *Proceedings 1999 IEEE Symposium on Information Visualization*, pages 73–78, 147, 1999.

[23] R. Vliegen, J. van Wijk, and E.-J. van der Linden. Visualizing business data with generalized treemaps. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):789–796, Sept 2006.

[24] M. Wattenberg. Visualizing the stock market. In *CHI '99 Extended Abstracts on Human Factors in Computing Systems*, pages 188–189. ACM, 1999.

[25] M. Wattenberg. A note on space-filling visualizations and space-filling curves. In *IEEE Symposium on Information Visualization*, pages 181–186, Oct 2005.