Vol. 11, No. 3, March 2013

Comparative Analysis between Split and HierarchyMap Treemap Algorithms for Visualizing Hierarchical Data

Aborisade D. O. (Corresponding Author)
Department of Computer Science, College of Natural Sciences, Federal University of Agriculture, Abeokuta, (FUNAAB) Ogun State, Nigeria.

Oladipupo, O. O.
Department of Computer and Information Sciences, Covenant University, Ota, Ogun State, Nigeria.

Oyelade, O. J.
Department of Computer and Information Sciences, Covenant University, Ota, Ogun State, Nigeria.

Obembe O. O.
Department of Biological Sciences, Covenant University, Ota, Ogun State, Nigeria.

Ewejobi, I. T.
Department of Computer and Information Sciences, Covenant University, Ota, Ogun State, Nigeria.

Obagbuwa, I. C.
Department of Computer Sciences,
Lagos State University, Lagos, Nigeria.(LASU)

Abstract

We carried out comparative analysis between Split treemap algorithm and a more recently introduced treemap algorithm called HierarchyMap. HierrachyMap and Split are Treemap Visualization methods for representing large volume of hierarchical information on a 2-dimensional space. Split layout algorithm has been developed much earlier as an ordered layout algorithm with capability to preserve order and reduce aspect ratio. HierarchyMap is a newer ordered treemap algorithm developed to overcome certain deficiencies of the Split layout algorithm. The two algorithms were analyzed to compare their rate of complexity. They were also implemented using object-oriented programming tool and compared using a number of standard metrics for measuring treemap algorithms. Their implementation shows that HierarchyMap and Split although maintain the same level of data ordering and usability but HierarchyMap algorithm has better aspect ratio, better readability, low run-time, and less number of thin rectangles compared to Split treemap algorithm. Since aspect ratio is an important metric for determining the efficiency of treemaps on 2-D and small screens, and the result of the analysis shows that HierarchyMap is better efficient than Split treemap alagorithm, we conlude that HierarchyMap is more efficient than Split treemap algorithm.

Keywords: Treemap algorithm, Aspect ratio, HierarchyMap, 2-D space, Data Visualization.

I. Introduction

Hierarchical structure is a structure comprising a series of ordered class of elements or entities within a particular system. Hierarchical structures such as Family structure, a University Structure and Manual Directory have been found to be very useful in representing information in almost all systems of life. Arranging information in hierarchical structures has also been observed to be more useful in bringing out meaning in the system being represented to the observer more than other known ways of representing it. Representations in hierarchical structures also help to clearly reveal the relationship between the components in the system. Data that are modeled into such structures are referred to as Hierarchical data. It was later observed that Hierarchical structure is only efficient for representing small and manageable data items. Efforts geared towards improving the Visualization of hierarchical data especially when voluminous data items are involved brought to mind the concept of Treemaps in early part of the Nineties by [2]. Treemap involves turning a tree into a planar space-filling map. Treemap visualization method maps hierarchical information into a rectangular 2-dimensional display in a space-filling manner such that 100% of the designated display space is utilized. [3]. It is described as space-filling visualization method capable of representing large hierarchical collections of quantitative data [5]. It works by dividing the display area into a nested sequence of rectangles whose areas correspond to an attribute of the dataset, effectively combining aspects of a Venn diagram and a pie chart. With the development of algorithm for early treemaps like Slice and dice, and Cluster and ordered treemap algorithms like Strip, Split and HierarchyMap, very large volume of data sets can be visualized on a 2D space like a computer screen with little or no difficulty. In this paper, a comparative analysis is made between a recently developed ordered algorithm called HierarchyMap and using metrics such as readability, aspect ratio, run time, and number of thin rectangles. The remaining sections are organized as follows; Section two reported the review of related literature, section 3 analyses the complexity of the algorithms and compares the two treemap algorithms (Split and HierarchyMap) using standard treemaps metrics, while section 4 discusses the implementation and results based on standard treemap metrics.

II. Related Works

From the time the idea of Treemaps was first conceived and original treemap developed to solve the problem of space usage by using the full display space to visualize the contents of the tree, many algorithms have been introduced to display hierarchical information structures [2]. These treemap algorithms in the order of their introduction and successive improvement include Slice and Dice, Cluster, Squarified, Pivot by Split Size, Pivot by Middle, Split Strip, and HierarchyMap treemap algorithm [8]. Of great importance to this paper are the ordered treemap algorithms like Pivot by middle, Pivot by Split Size, Strip, Split and HierarchyMap treemaps algorithms. The idea that lead to algorithms for ordered treemaps is that it is possible to create a layout in which items that are next to each other in the given order are adjacent in the treemap [6] . Treemap algorithm where the first step is to choose a special item, the pivot, which is placed at the side of rectangle R. In the second step, the remaining items in the list are assigned to three large rectangles that make up the rest of the display area. Finally, the algorithm is then applied recursively to each of these rectangles [5]. This algorithm has some minor variations, depending on how the pivot is chosen. There are three pivot-selection strategies; the first is the algorithm where the pivot with the largest area is chosen. The motivation for this choice is that the largest item will be the most difficult to place, so it should be done first [5].The alternate approaches to pivot selection are pivot-by-middle and pivot-by-split-size. Pivot-by-middle selects the pivot to be the middle item of the list i.e. if the list has n items, the pivot is item number n/2, rounded down. The motivation behind this choice is that it is likely to create a balanced layout. In addition, because the choice of pivot does not depend on the size of the items, the layouts created by this algorithm may not be as sensitive to changes in the data as pivot by size. Pivot-by-split-size selects the pivot that will split the list into approximately equal total areas. With the sub-lists containing a similar area, they expected to get a balanced layout, even when the items in one part of the list are a substantially different size than items in the other part of the list. The Strip treemap algorithm is a modification of the existing Squarified Treemap algorithm [4]. It works by processing input rectangles in order, and laying them out in horizontal (or vertical) strips of varying thicknesses. It is efficient in that it produces a layout with better readability than the basic ordered treemap algorithm, and comparable aspect ratios and stability [5]. The inputs in a Strip treemap are the subdivision of rectangle *R* and a list of items that are ordered by an index and have given areas. As with all treemap algorithms, the inputs are a rectangle *R* to be subdivided and a list of items that are ordered by an index and have given areas. A current strip is maintained, and then for each rectangle, a check is done to know if adding the rectangle to the current strip will increase or decrease the average aspect ratio of all the rectangles in the strip. If the average aspect ratio decreases (or stays the same), the new rectangle is added. If it increases, a new strip is started with the rectangle [5]. The result is the Split treemap which, like the Pivot, is a partially ordered algorithm. It produces a layout where the natural ordering of the data set is roughly preserved, while in most cases producing better aspect ratios than the Pivot and the Strip treemaps [6].

III. Method

Algorithms Complexity Analysis

This section describes the two treemap algorithms (Split and HierarchyMap) and their complexity analysis, as its helps to compare algorithms to see which one is better.

Split Algorithm:

Inputs to the algorithm are an ordered list, $L = \{l_1, l_2, \ldots l_n\}$ of items to layout and a rectangle, *R*, in which the items are distributed. Weight *w(L)* is defined to be the sum of the sizes of all the elements in the list. The algorithm follows a recursive process, where *L* is split into two halves, *L*1 and *L*2, such that *w*(*L*1) is as close as possible to *w*(*L*2). Noting that the ordering of the elements must not be changed. *L*1 and *L*2 are both ordered, and all the elements of *L*1 have an index less than those of *L*2 to give. $w(L_1) \approx w(L_2) \approx w(L)/2$ *and* $\forall l_i \in L_1$, $\forall l_j \in L_2 : l_i \leq l_i + 1 \leq \cdot l_j \leq l_j + 1$

$\alpha(R)$ is then defined to be the area of a rectangle *R*. The rectangle *R* is split, either horizontally or vertically depending on whether the width is bigger than the height, into two sub rectangles, *R*1 and *R*2 such that their areas corresponds to the size of the elements of *L*1 and *L*2, that is ;

$$\frac{\alpha(R1)}{\alpha(R)} = \frac{w(L1)}{w(L)}, \quad \frac{\alpha(R2)}{\alpha(R)} \quad \frac{w(L2)}{w(L)}$$

Hence, recursively layout the contents of $L1$ and $L2$ in $R1$ and $R2$ according to the algorithm [6].

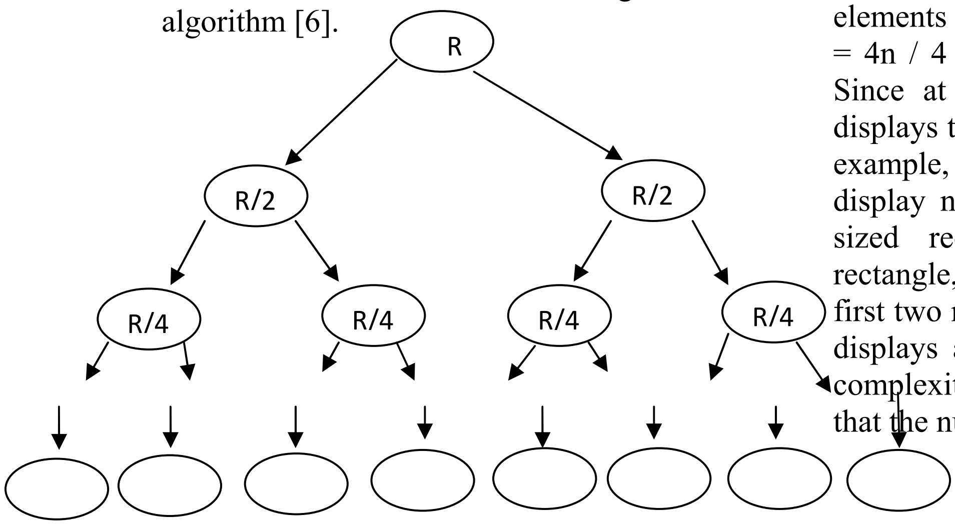


Figure 1: Split treemap recursion model

Here the Split treemap algorithm is modeled by a recursive tree where each circle represents a node (or rectangle R in which the items are distributed.) and the number written in the circle indicates the items ($l_{1,}$ $l_{2,\dots\dots}l_n$) to layout. The first node stands for the original rectangle R to be sub-divided in layouts. The arrows indicate recursive calls made between nodes. Since the algorithm follows a recursive process, where L is split into two halves, $L1$ and $L2$, such that $w(L1)$ is as close as possible to $w(L2)$. The call to the next row shows the division of the first set of into 2 halves (i.e. n / 2). This is indicated by the two arrows at the top. In turn, each of these also makes calls to the next .row for further sub-division of n / 4 each, and so forth until all the items are displayed. If the total the total number of items to be displayed in figure1 is taken to be n, and the total number of items in each level of the tree is n. The first row contains only one call the next row with an array of size *n*, so the total number of elements is *n*. The second row has two calls to the next level (row) each of size n / 2. But n / 2 + n / 2 = n and so again in this row the total number of elements is *n*. In the third row, we have 4 calls each of which is applied on an n / 4-sized rectangle, giving a total number of elements equal to n / 4 + n / 4 + n / 4 + n / 4 = 4n / 4 = n. So again we get *n* elements. Since at each level of the tree rectangle displays the items from the input values. For example, the left node in level 1 has to display n / 2 elements. It splits the n / 2-sized rectangle into two n / 4-sized rectangle, calls recursively to display those first two nodes from the left in level 3), then displays all. This argument shows that the complexity for each row is Θ(n). And since that the number of levels in the is log(n).

We have log(n) rows and each of them is Θ(n), therefore the complexity of Split treemap algorithm is Θ(n * log(n)).

HierarchyMap treemap Algorithm

Inputs to the algorithm as ordered data in tree-like form. infotree(treedata nodes)=T={t_1,t_2 ,t_3, ………., t_n} and a 2-D space divided into four equal rectangles.

Step 1: If the number of hierarchical items to be displayed is zero (i.e. T=0) , then no display.

Step 2: If the number of hierarchical items to be displayed is 1 (i.e T=1), then Set 2-D space to the item **Step 3**: If the number of items is greater than 1, divide the rectangular 2-D space into four equal sizes and recursively divided each of the resultant item into fours until all items in the list are exhausted. Such that $\forall\, t_i \in T_{1,} \forall\, t_j \in T_{2,} \forall\, t_K \in T_{3,}\dots\dots\dots \forall\, t_n \in T_n$ **:** $\mathbf{t}_i \le t_{i+1} \le t_j \le t_{j+1} \le t_k \le t_{k+1} \le \dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots.\ t_n \le t_{n+1.}$

Step 4: an attribute of the each hierarchical item corresponds to an area of each of the nested rectangles defined as **area(R)** in such a manner that their areas correspond to the size of the elements of T_1, T_2 T_3, and T_4 where **area (R_1) ≈ area (R_2) ≈ area (R_3)** ≈ ………………**area (R_n) [8]**.

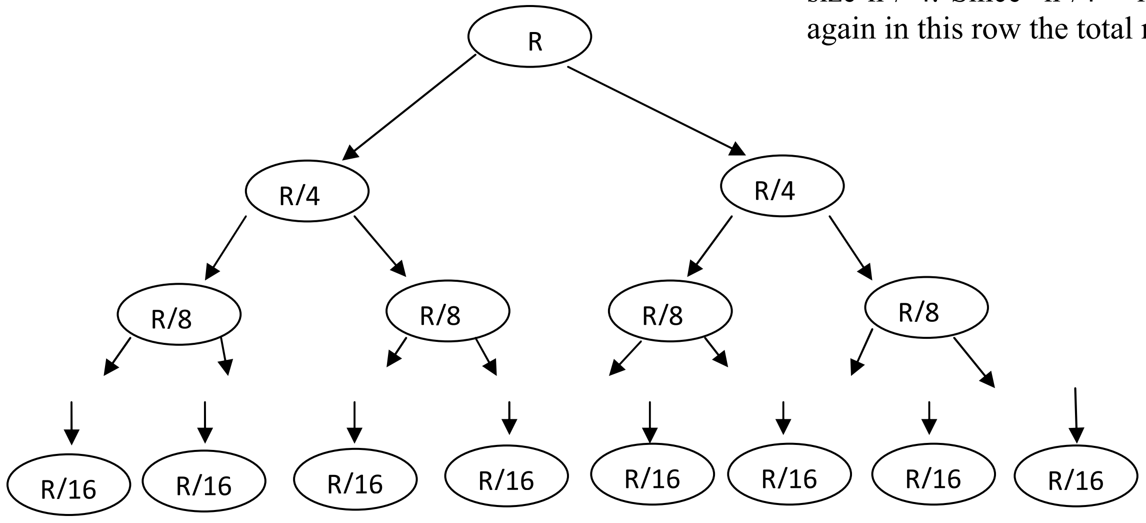


Figure 2: HierarchyMap treemap recursion model

In a similar manner, HierarchyMap algorithm is represented by a recursive tree in figure 2 where each circle represents a node (or rectangle *R* in which the items are distributed.) and the number written in the circle indicates the items ($l_{1,}$ $l_{2,.....}l_n$) to layout. The root node stands for the original rectangle R to be sub-divided in layouts. The arrows indicate recursive calls made between nodes. HierarchyMap recursively processes the display of the items on rectangular space by sub-dividing the first rectangle R into four parts. T1, T2, T3, and T4 where area (R_1) ≈ area (R_2) ≈ area (R_3) ≈ …………area (R_n). The call to the next row shows the division of the first set of into 4 parts (i.e. n /4). This is indicated by the two arrows at the top. In turn, each of these also makes calls to the next .row for further sub-division of n / 8 each, and so forth until all the items are displayed. If the total the total number of items to be displayed is n, and the total number of items in each level of the tree is 0.5n. For example, the first row contains only one call. The second level with items of size n and hence has total number of elements is 0.5*n*. The third level has two calls to the next level (row) each of size n / 4. Since n /4 + n /4 = 0.5n and so again in this row the total number of elements is *n*. In the fourth level, we have 4 calls each of which is applied on an n / 16-sized rectangle, giving a total number of elements equal to n / 16 + n / 16 + n / 16 + n /16 + n/16+n/16+n/16+n/16= 0.5n, giving us 0.5n again. Since at each level of the tree, rectangle displays the items from the input values. For example, the left node in level 2 has to display n /4 elements. It splits the n / 4-sized rectangle into two n / 8-sized rectangle, calls recursively to display those first two nodes from the left in level 3), then displays all. This argument shows that the complexity for each row is Θ(0.5 n). And since that the number of levels in the is log(n). We have log(n) rows and each of them is Θ(0.5n), therefore the complexity of Split treemap algorithm is Θ(n * log(0.5 n)) which is approximately equal to the complexity of the split treemap derived earlier as Θ(n * log(n)). But the constant multiplier in the HierarchyMap makes the difference. Since the constant multiplier is 0.5, it means that it grows more slowly than that of Split treemap and it is better and capable of quickly displaying items on rectangular space. This shows that the derived algorithm for Split is worse than that of HierarchyMap.

Implementation and Other Analysis Metrics

This section shows the implementation of the two algorithms (Split and HierarchyMap) and compares them on the basis of the standard treemap algorihm metrics like Aspect Ratio, Ordering, Readability, number of thin rectangles, Run time, and Usability. The behavior of each of the algorithm is observed with respect to the standard metrics when the treemap displays no (zero) item (Fig. 3a and Fig. 3b), displays between 10-15 items (Fig 4a and Fig 4b), displays between 20-25 items (Fig 5a and Fig 5b), displays between 30-60 items (Fig 6a and Fig 6b). Further discussion of these results is found in the remaining part of this Section.

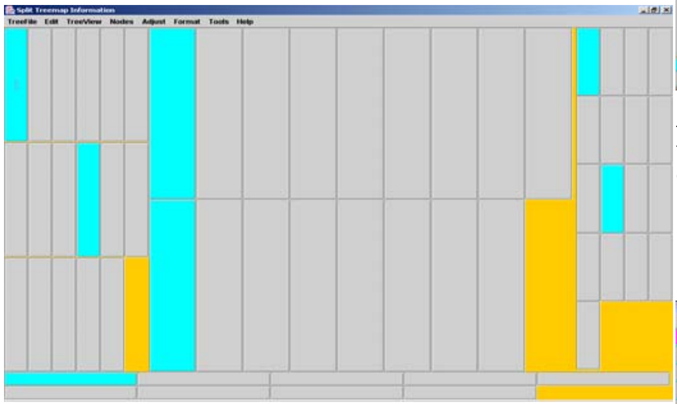

Split Treemap Information
TreeFile Edit TreeView Nodes Adjust Format Tools Help

Figure 3a: Split treemap implementation with no item displayed (Aspect ratio is 2.92)

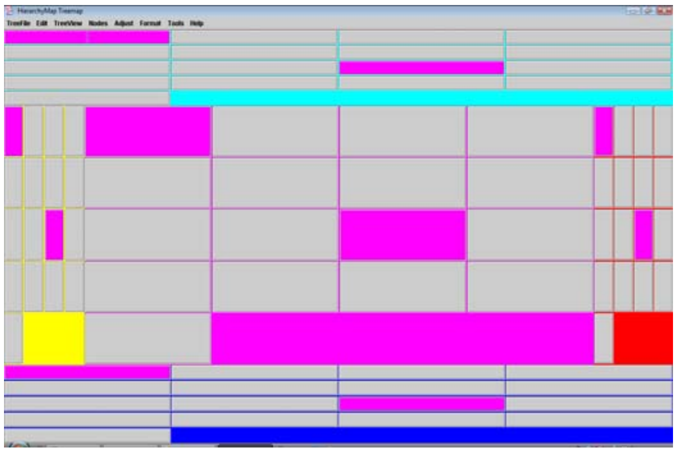


Figure 3b: HierarchyMap showing nested rectangles with no item displayed and Aspect Ratio of 1.72)

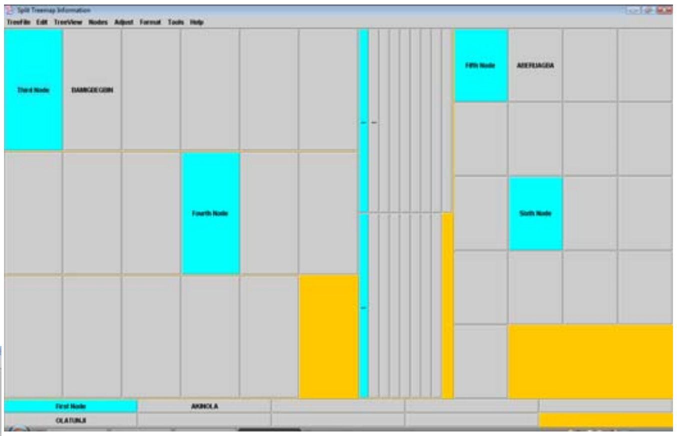


Figure 4a: Split Treemap with an average of 10 and 15 items giving Aspect Ratio 1.72

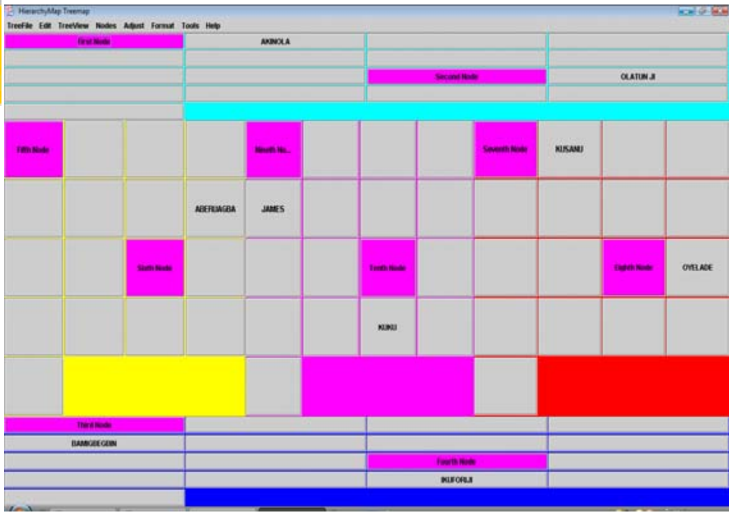


Figure 4b: HierarchyMap with an average of 10 and 15 items and Aspect Ratio of 1.72

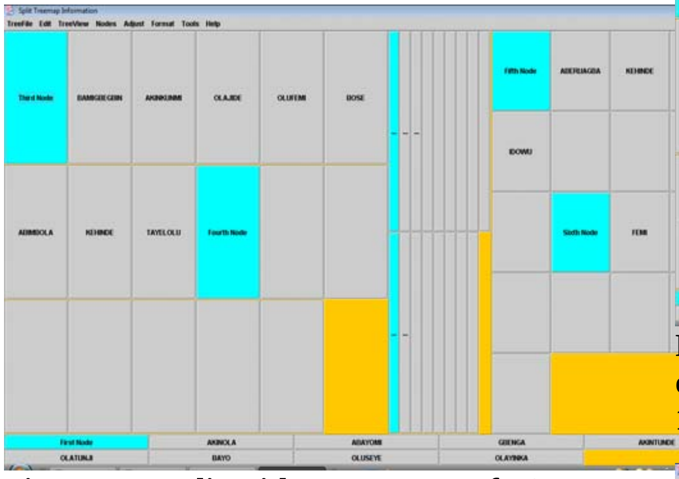


Figure 5a: Split with an average of 20 to 25 items displayed maintains Aspect Ratio of 1.72

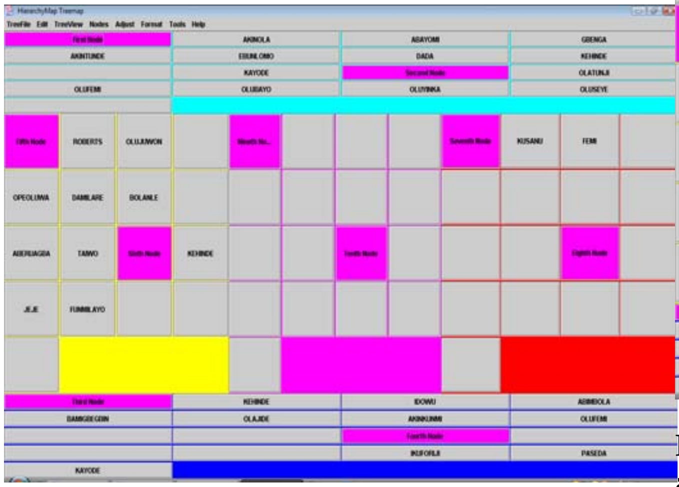


Figure 5b: HierarchyMap with average of 20-25 items displayed (Aspect Ratio 1.72)

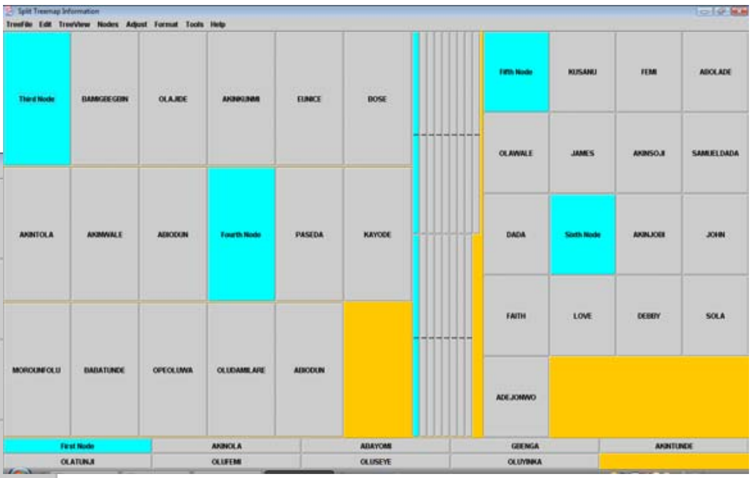


Figure 6a: Split treemap with an average of 30-60 items displayed (Aspect Ratio 1.72)

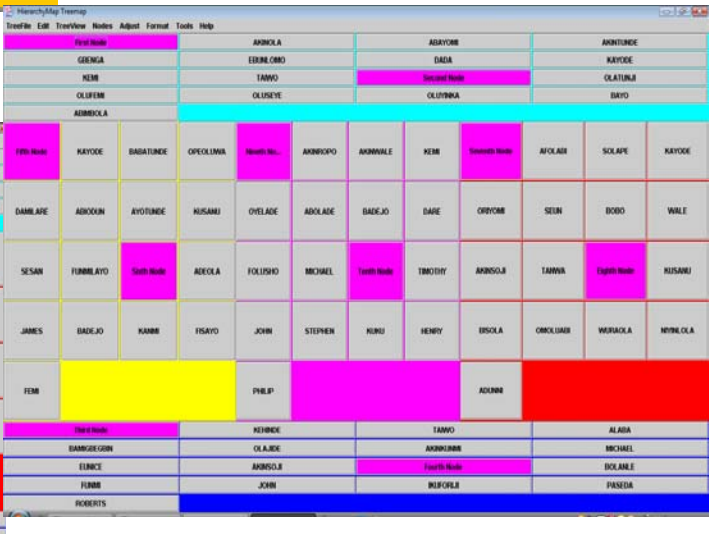


Figure 6b: HierarchyMap with an average of 30-60 items displayed (Aspect Ratio 1.72)

IV. DISCUSSION OF RESULTS

This section discusses the results of implementing the two algorithms (HierarchyMap and Strip algorithm) with respect to the standard treemap metrics such as Aspect ratio, Ordering, Readability, Run time, Number of thin rectangles and Usability.

4.0.1 Aspect ratio

Aspect ratio is the defined as the longest side of a rectangle divided by it shortest side. It is also defined as Max(Width/Height, Height/Width) of a rectangle. The lower the aspect ratio of a rectangle, the more nearly square it is. The aspect ratio for the two algorithms were determined using the same set of data. The Height/Width of each of the rectangles generated by each of the Treemap algorithm program are calculated (in cm). The result of the calculated values are added together and divided by four to get the average height and average width. The results of the calculated aspect ratios are represented in Figure 7 below.

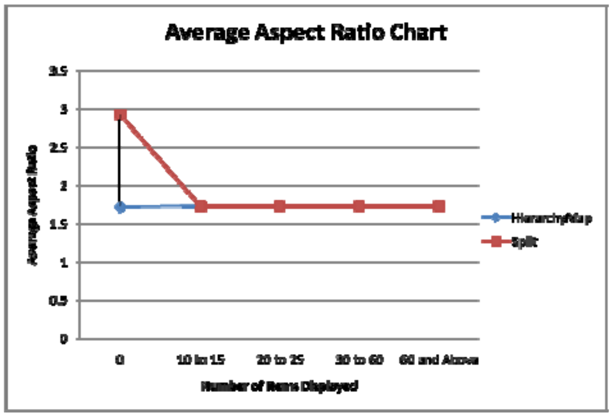


Figure 7: The graph plotted Average Aspect Ratio against Number of Items represents the relationship between Aspect ratio and the Number of rectangles generated in HierarchyMap and Split Treemap Algorithm.

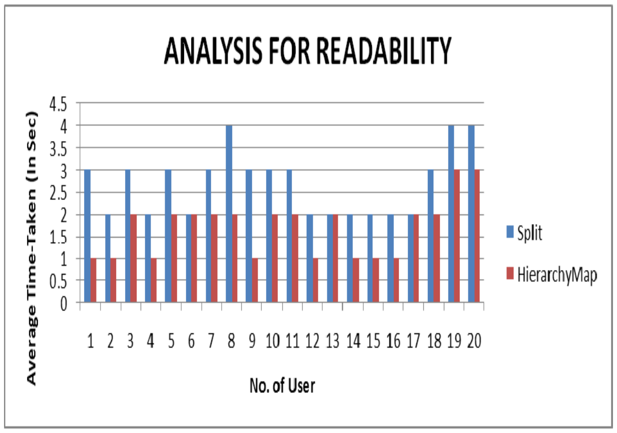
The graph shows that HierarchyMap Treemap Algorithm has an Aspect ratio of 1.73 while Split Treemap Algorithm has Aspect ratio of about 2.92 when no rectangle is displayed. Both treemap algorithms maintain Aspect ratio of 1.73 when number of rectangles displayed are between 10, 60 and above in their treemaps. Hence, HierarchyMap is observed to have better aspect ratio than Split treemap.

4.0.2 Ordering

Ordering is a metric that determines the ability of the algorithm to create a layout in which items that are next to each other in a given order are placed adjacent to each other (Berderson et al., 2002). Implementation of HierarchyMap and Split treemap algorithms as indicated above in the treemaps diagrams show that the two algorithms maintain items in the ordered manner.

4.0.2 Readability

Readability describes the measure of the number of times a user eye will have to change direction when scanning the treemap in order (Berderson et al., 2002). This test is used to measure how easy it is to locate a particular information between the layouts generated by the Split and HierarchyMap algorithms. In this experiment, twenty (20) persons (users) were carefully selected to scan through the treemap generated from the implementation of the two algorithms to locate a particular information. The time taken each of them was presented in Figure 8.



:

Figure 8: Analysis for Readability: **A**verage time is plotted against the number of users for both Split and HierarchyMap

The graph shows that in HierarchyMap, readers use less time in most cases to locate information compared to Split treemap where more time is used in most cases by users to locate information of their choice on the treemap. This shows that HierarchyMap has better readability than Split. This reflects the property of the Split layout, which changes direction more often than the HierarchyMap layouts that use several sub-lists instead of two. The results from the test indicate a slightly worse readability for the Split layout. HierarchyMap gives better readability because of the pivot. Assigning a pivot and then splitting the list in two, four, and then several parts generates a more consistent layout than the Split layout, which splits the list into two parts. Since the layout direction can alter between horizontal and vertical every time the list is split, the HierarchyMap algorithm is more predictable, since all the four sub lists will be laid out in the same directions, whereas the Split layout, with only two sub lists, will change direction more frequently.

4.0.3 Run Time

Run time is another important metric for evaluating treemap algorithm usability. In this case, run time for the implementation of the two algorithms is compared. This is done ten (10) different times for each algorithm on a Laptop Computer with the specification such as: Intel® Core ™ 2 CPU T5200, 1.60 GHz, RAM 1015MB, 32-bit Operating System. The readings obtained are presented in Figure 9.

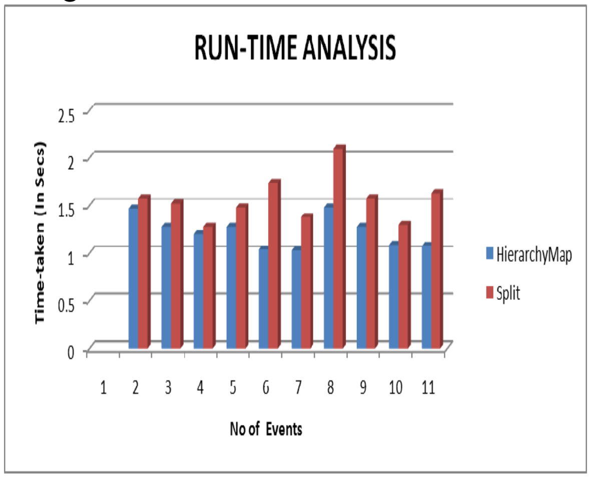


Figure 9: Column graph showing the Run-time Analysis for the two Algorithms

It is observed in Figure 9 above that HierarchyMap has a lower run time in all the events compared with Split treemap algorithm.

4.0.4 Number of thin rectangles

Another treemap efficiency metric very close to that of aspect ratio is the number of thin rectangles. The number of thin rectangles in a treemap determines the aspect ratio in the treemap. A treemap with a high number of thin rectangles has a high aspect ratio while a low number of thin rectangles has low aspect ratio. Figure 10 shows the number of thin rectangles generated by Split and HierarchyMap algorithms for different number of items displayed.

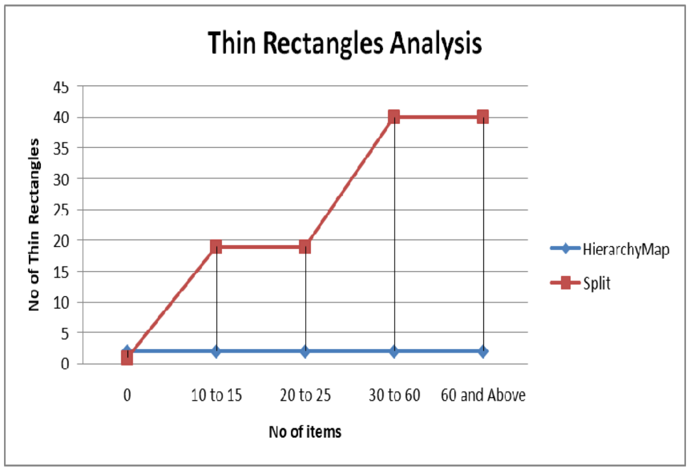


Fig. 10 Thin rectangle analysis

The thin rectangle analysis in Figure 10 shows that the number of thin rectangles generated by Split is more than the number of thin rectangles generated by HierarchyMap Treemap. Hence, Split has high aspect ratio than HierarchyMap treemap

4.0.5 Usability: HierarchyMap treemap algorithm by its implementation has been observed to be capable of generating high volumes of hierarchical information on a 2-D space than Split treemap algorithm. It was interesting to observe that when the number of items to be displayed was more than 60, HierarchyMap treemap became more stable and did not flicker.Hence, HierarchyMap treemap algorithm is more efficient than Split algorithm in laying out hierarchical data in a 2-D space like a Computer screen.

V. Conclusion and Future Work

In this work, we compared the efficiency of two Ordered treemap algorithms called HierarchyMap and Split algorithms developed to represent hierarchical data on 2-D space. In comparing the two algorithms, the two algorithms were first analyzed measure their complexity. Then standard treemap algorithm metrics like aspect ratio, readability, ordering, usability, number of thin rectangles, and run time were also used as the basis of comparing them. The measure of complexity of the two algorithms shows that HierarchyMap is more efficient in laying out items on rectangular space and results of implementation using standard treemap algorithms metrics showed that HierarchyMap and Split although maintained the same level of data ordering and usability but HierarchyMap algorithm was observed to have better aspect ratio, readability, low Run-time, and less number of thin rectangles compared to Split treemap algorithm. Since aspect ratio is one of the most important properties when using treemaps on 2-D and small screens, HierarchyMap can therefore be said to be more efficient than the Split treemap algorithm. The future effort on this work is intended to improve on HierarchyMap algorithm to have better ordering and usability.

VI. References

[1] Bruggemann-Klein, A. and D. Wood. Drawing Trees nicely with Tex. Electronic Publishing, 2(2):101–115, 1989.

[2]B. Johnson and B. Shneiderman. Treemaps: A space-filling approach to the Visualization of Hierarchical Information Structures. *In Proc. of the 2nd International IEEE Visualization Conference*, pages 284–291, October 1991.

[3] B. Shneiderman. Tree visualization with treemaps:A 2-D space-filling approach. *ACM Transactions on Graphics*, 11(1):92–99, September 1992.

[4] Bruls S., M., Huizing, K., and Van Wijk, J., 2000. Squarified treemaps. *In Proceedings of the Joint Eurographics and IEEETCVGSymposium*onVisualization(VisSym), 33–42.

[5] Bederson, B., Shneiderman, B., and Wattenberg, M. 2002. Ordered and quantum treemaps: Making effective use of 2D space to display hierarchies. ACMTransactions on Graphics *21*, 4, 833–854.

[6] B. Engdahl, 2005. Ordered and Unordered Treemap Algorithms and Their Applications on Handheld Devices. Master's Thesis in Computer Science at the School of Computer Science and Engineering,Royal Institute of Technology year 2005.

[7] D.E. Knuth. Fundamental algorithms. Art of computer programming. Volume 1. Addison-Wesley, Reading, MA, 1973.

[8] D. O. Aborisade and O.J. Oyelade. HierarchyMap: A New Approach to Treemap Visualization of Hierarchical Data. Global Journal of Computer Science and Technology.Vol. 9 Issue 5,Online ISSN-0975-4172,Print ISSN 0975-4350. Pages 77-81. January, 2010.

[9] G.W. Furnas. Generalized fisheye views. In Proc. of ACM CHI'86, Conference on Human
Factors in computing systems, pages 16–23, 1986. Herman H, Maurer. Data Structures
and Programming Techniques. Prentice- All Incorporation. 1977.

[10] J. Bingham and S. Sudarsanum. Visualising large hierarchical clusters in
Hyperbolic space. Bioinfomatics Chapter 16:pg. 660-661, 2000.

[11] Malin Koksal, Visualization of threaded discussions forums on hand-held
devices, Masters Thesis at NADA, 2005.

[12] Russel Winder and Graham Roberts, Developing Java Software, John Wiley & Sons.
1998.

[13] S.K. Card, G.G. Robertson, and J.D. Mackinlay. The information visualizer, an
Information workspace. In Proc. of ACM CHI'91, Conference on Human Factors in
Computing Systems, pages 181–188, 1991.

[14] Wattenberg, M. 1999. Visualizing the stock market. In Extended Abstracts on Human Factors in Computing Systems (CHI), ACM Press, 188–189.

Authors's Profile

Aborisade Dada Olaniyi is a PhD student and Lecturer in the Department of Computer Science, College of Natural Sciences, Federal University of Agriculture, Abeokuta, Ogun State, Nigeria. He bagged his first degree in in B.Sc Mathematical Sciences (Computer Science option) in 2000 from University of Agriculture, Abeokuta, Ogun State, Nigeria and Msc in Computer Science of the University of Ibadan, Oyo State, Nigeria in 2007. His research interests are in the area of Human Computer Interaction (HCI) and Computer Information Security. He's a member of Microsoft Information Technology Academy (MITA) and Nigeria Computer Society (NCS).

Oyelade Olanrewaju Jelili recieved his Bachelor degree in Computer Science with Mathematics (Combined Hons) and M.Sc degree in Computer Science from Obafemi Awolowo Univ ersity, Ile-Ife, Nigeria. He obtained his Ph. D in Covenant University, Ota, Nigeria. **Dr. Oyelade, O. J.** is a senior faculty member in the department of Computer and Information Sciences, Covenant University, Ota, Nigeria. His research interests are in Bioinformatics, Clustering, Fuzzy logic and Algorithms. He is a member of International Society for Computational Biology (ISCB), Africa Society for Bioinformatics and Computational Biology (ASBCB), Nigeria Society of Bioinformatics and Computational Biology (NISBCB), the Nigerian Computer Society (NCS), and Computer Professional Registration Council of Nigeria (CPN).

Obagbuwa Ibidun Christiana is a lecturer in the Department of computer science, Lagos state University Ojo, Lagos state, Nigeria. She obtained her first degree (B.Sc Computer Science) in 1997 from University of Ilorin, Ilorin, Kwara state. She proceeded to University Of PortHarcourt, Rivers state and obtain Degree of master in Computer science in 2005. She is currently working on her Doctoral degree (PhD) in Computer science. Her area of specialization include Computer security, Computational intelligence/softcomputing,Telecommunicati on & Networking and Databases. She is happily married with Three children. She is a member of Nigeria Computer Society (NCS), and Computer Professionals (Registration Council) of Nigeria (CPN)

Oladipupo O. O. recieved her Bachelor degree in Computer Science in University of Ilorin and M.Sc degree in Computer Science from Obafemi Awolowo Univ ersity, Ile-Ife, Nigeria. She obtained her Ph. D in Covenant University, Ota, Nigeria. **Dr. Oladipupo, O. O.** is a senior faculty member in the department of Computer and Information Sciences, Covenant University, Ota, Nigeria. Her research interests are in Artificial Intelligence, Data Mining, and Soft Computing Technique. She is a member of Nigerian Computer Society (NCS), and Computer Professional Registration Council of Nigeria (CPN).

Itunuoluwa Ewejobi received her Bachelor's degree (First Class honours) in Computer Science and M.Sc degree in Computer Science from Covenant University, Ota, Nigeria. She is a Ph.D student in the Bio-informatics research group of the Department of Computer and Information Sciences, Covenant University, Nigeria. She is currently on a a DAAD (German Academic Exchange Service) Sandwich Scholarship at the Ruprecht-Karls Universität,Heidelberg, Germany to carry out some part of her Ph.D research titled ***"Transcription Factor(s)-Target Detection in the malaria parasite Plasmodium falciparum"***. Her research interests include; Artificial Intelligence, Transcriptomics and Modeling of biological systems and Algorithms.