RELT: Visualizing Trees on Mobile Devices

Jie Hao Kang Zhang
Dept. of Computer Science
Erik Jonsson School of Engineering and Computer Science
The University of Texas at Dallas
Richardson, TX 75083-0688, USA
{jxh049000, kzhang}@utdallas.edu

Mao Lin Huang
Department of Computer Systems
Faculty of Information Technology
University of Technology, Sydney
PO Box 123 Broadway, NSW 2007, Australia
maolin@it.uts.edu.au

ABSTRACT

The small screens on increasingly used mobile devices challenge the traditional visualization methods designed for desktops. This paper presents a method called "Radial Edgeless Tree" (RELT) for visualizing trees in a 2-dimensional space. It combines the existing connection tree drawing with the space-filling approach to achieve the efficient display of trees in a small geometrical area, such as the screen that are commonly used in mobile devices. We recursively calculate a set of non-overlapped polygonal nodes that are adjacent in the hierarchical manner. Thus, the display space is fully used for displaying nodes, while the hierarchical relationships among the nodes are presented by the adjacency (or boundary-sharing) of the nodes. It is different from the other traditional connection approaches that use a node-link diagram to present the parent-child relationships which waste the display space. The hierarchy spreads from north-west to south-east in a top-down manner which naturally follows the traditional way of human perception of hierarchies. We discuss the characteristics, advantages and limitations of this new technique and suggestions for future research.

KEYWORD*: Tree visualization, mobile interface, screen estate, aesthetic layout.*

1 Introduction

There is a dramatic increase in the population who use mobile computing devices. Although the hardware is becoming more powerful, online browsing and navigation tend to be not user-friendly. For example, when the user wishes to search for a favorite music on a mobile phone, many clicks or button-pushes are required. This is mostly due to the limited screen space where few music pieces can be presented on one screen.

Most current mobile online search is linear, possibly with scroll bars. Web browsing on mobile devices is also primarily based on the desktop browsing approach with scaled versions. Therefore, finding information on a mobile device has not been as fast as needed. There have been growing research activities in effective and efficient mobile user interfaces. Yet few hierarchical search methods that aim at minimizing the number of clicks and button-pushes have been developed for small screens.

This paper presents a new RELT method for visualizing hierarchical information on mobile devices. The remainder of this paper is organized as follows. Section 2 covers the related work by reviewing desktop-based tree visualization methods and mobile visualization methods. RELT is introduced in Section 3 and an application for music classification depicting this new algorithm in Section 4. Section 5 builds an estimate function for RELT and an optimized RELT algorithm. Section 6 concludes the paper and mentions the future work.

2 Related Work

2.1 Tree Visualization on Desktops

The current research on tree visualization can be generally classified into two categories:

- **Connection:** This method uses nodes to represent tree leaves, and edges to represent parent-child relationships. Much research has been done in this category, such as balloon view [4, 6], radial view [1, 6], and space-optimized tree visualization [7]. The connection-based approaches match the human perception of hierarchy. Their layouts are also easy to understand with clear structures.

- **Enclosure:** This method represents nodes as rectangles. The display area is recursively partitioned to place all the nodes inside their parent's regions. The tree map view [5, 9] is a typical enclosure-based approach. Enclosure-based approaches achieve economic screen usage, but do not provide a clear hierarchical view.

2.2 Visualization on Mobile Devices

Researchers have developed methods for mobile displays by deriving them from those for desktop displays.

Yoo and Cheon [12] introduced a preprocessor to classify the input information. It divides the input information into different types and each type maintains its corresponding visualization method. For hierarchical information, their approach applies the radial layout method [1, 3] with the mobile devices' restrictions [2, 11]. They also use the fisheye view algorithm to help the user to see highlighted regions.

Although, the above approach works for mobile devices, it fails to efficiently utilize the space, evidenced by the examples provided [12].

The main difference between the presented RELT approach and traditional radial approaches is that the latter performs 360 degree circular partitioning while RELT uses 90 degree polygon partitioning, that is more appropriate to fit on mobile screens.

3 Radial Edgeless Tree Visualization

The RELT algorithm is designed to not only utilize the screen space but also maintain the tree layout. The following subsections first give an intuitive explanation of RELT, then describe the algorithm in detail, and finally discuss the complexity of the algorithm.

3.1 Basic Ideas

A tree is a connected graph $T=(V, E)$ without a cycle. A rooted tree $T=(V, E, r)$ consists of a tree T and a distinguished vertex r of T as the root. Each vertex v has an associated value $w(v)$, which we call the weight.

The entire rectangular display area is partitioned into a set of none-overlapping geometrical polygons (or nodes) $P(v_1), P(v_2), \dots, P(v_n)$ that are used to visually represent vertexes $v_1, v_2, \dots, v_n$. Each polygonal node $P(v)$ is defined by three or four cutting edges which may be shared with other nodes. These boundaries are defined as below:

1. A common boundary sharing with its parent represents the child-parent relationship.
2. A common boundary sharing with all its children represent the patent-child relationship.
3. One or two boundaries sharing with its siblings represent the sibling relationships.

The geometrical size of a node $P(v)$ is calculated based on its weight $w(v)$. We, therefore, use boundary-sharing to represent the parent-child relationships among nodes, rather than a node-link diagram. Thus, the display space utilization is maximized. The entire tree T is drawn hierarchically from the north-west at the root to the south-east in the top-down manner, which naturally follows the traditional way of human perception of hierarchies. Note the approach can be easily adapted to move the root to other screen locations.

An intuitive method combining the previous two approaches is constructed with three steps. First, a normal connection-based method, the classical hierarchical view [8] for example, is applied. Second, consider each node as a balloon and inflate all the balloons until they occupy the whole screen. Third, these anomalistic non-overlapping balloons are relocated to simulate the tree structure. Although the above step appears like a space-filling approach, the result is more like a radial display. As presented next, the difference from the typical radial approaches, such as InterRing [10], is that our approach computes area allocations based on the nodes' weights, rather than their angles.

3.2 Algorithm

For a given tree, the method recursively calculates the weight for each vertex. Vertexes are classified into four types and each type is assigned with a corresponding rule. The root is assumed to locate at the upper left corner. We employ depth-first search to traverse the tree. Whenever a new vertex is met, the corresponding rule is applied. Every rule considers two operations. One is the node area distribution operation. The other is how to recursively divide its area for its children. After completely traversing the tree, the entire display area is partitioned into a set of non-overlapping polygons which are used to represent vertexes $v_1, v_2 \dots v_n$. In this case, the display area is fully utilized and a set of graphical links that are commonly used in traditional connection-based methods are avoided. The algorithm is given in peudocode as below:

```
procedure RELT (matrix ad_matrix)
begin
  Para_Creator (ad_matrix)
  // Calculate the necessary parameters for each node.
  DFS (ad_matrix)
  // Depth first search to traverse the tree
  if vertex  v is new then
      int L= Test (v)
      // Return rule L  to v
      Polygonal_Node (v, L)
      // Assign a region to v  with rule L
      Partition_Area(v)
      // Divide the area depending on the weights of v's children
      fi
end
```

The RELT algorithm shown above consists of four major functions that are explained next.

Para_Creator(matrix *ad_matrix***)** calculates the necessary parameters, including weight, depth, parent and children for each vertex. A vertex *v* is assigned with a weight *w(v)*, which is calculated in the following way:

- If vertex *v* is a leaf, *w(v)* =1.
- Otherwise, if *v* is not a leaf and has *m* children $\{v_1, v_2, \ldots, v_n\}$ then

$$w_N = 1 + \sum_{i=1}^{m} w_i \qquad (1)$$

Test(vertex *v***)** returns the rule that should apply to vertex *v*. We classify all vertexes into four types according to their characteristics in the tree. Specifically, a vertex *v* is of type:

1. If *v* is the only child of its parent.
2. If the parent of *v* has more than one child AND *v* is the first child of its parent.
3. If the parent of *v* has more than one child AND *v* is the last child of its parent (Note that child vertices are numbered from left to right. The left most child is the first and the right most child is the last).
4. If the parent of *v* must have more than one child AND *v* is neither the first nor the last child of its parent.

Before defining the function **Polygonal_Node(Vertex** *v*, **int** *L*), several notations and definitions should be introduced.

- L_{Ni}: is the i_{th} cutting edge of node *P(v)* . The cutting edges are numbered from left to right, as illustrated as Figure 1.

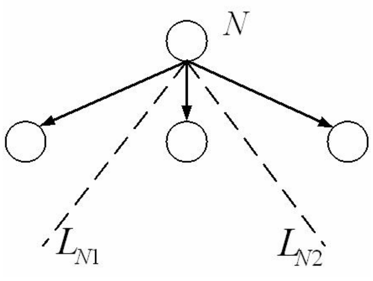

Figure 1 Cutting edges

- N_{NAS}: *N*'s nearest ancestor with sibling(s) as illustrated in Figure 2, is the nearest ancestor of *N* that has at least one sibling.
- N_{NAS_P}: The parent of N_{NAS}.
- N_{NASL}: N_{NAS} that is not the leftmost sibling.

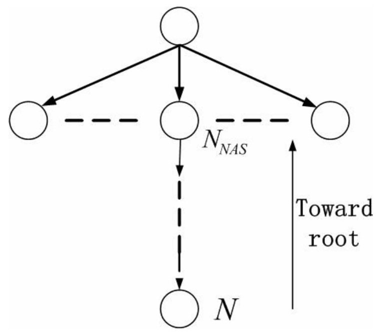

Figure 2 N's nearest ancestor with siblings

- N_{NASL_P}: The parent of N_{NASL}.
- N_{NASR}: N_{NAS} that is not the rightmost sibling.
- N_{NASR_P}: The parent of N_{NASR}.

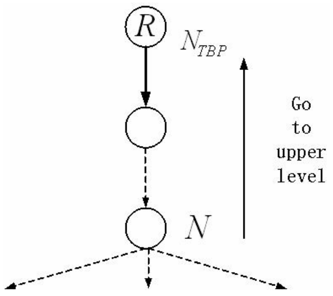

Figure 3 Special case with the root being named N_{NAS}

Consider such as case, as illustrated in the Figure 3, where a node *N* has no N_{NAS}. We treat it as a special case by making the root R as N_{NAS}.

Polygonal_Node(Vertex *v***, int** *L***)** represents a vertex in a polygonal shape, rather than a rectangular or circular shape that is commonly used in other connection-based visualizations. Each node is constructed by linking two division line, so this function essentially dictates how to choose these two cutting edges.

1. *L* = 1 (Node type 1)

   The cutting edges used by node *P(v)* are the same as those by N_{NAS}. Rule1 first finds cutting edges used by N_{NAS} and then link them together to form *N's region*.

2. *L* = 2 (Node type 2)

   The first cutting edge chosen by Rule2 is $L_{N^1{}_i}$ where $N^1 = N_{RTBP\_P}$ and N_{NAS_P} is the $(i+1)_{th}$ child of N_{NASR_P}. The second cutting edge is $L_{N^2{}_1}$ where N^2 is the parent of *N*.

3. *L* = 3 (Node type 3)

   The first cutting edge selected by Rule3 is $L_{N^1{}_1}$ where N^1 is the parent of *N*. The second cutting edge is $L_{N^2 i}$ where $N^2 = N_{NASL\_P}$ and N_{NASL} is the $(i+1)_{th}$ child of $N_{NASL\_P.}$

4. $L = 4$ (Node type 4)

The two cutting edges chosen by Rule4 are $L_{N^1(i-1)}$ and $L_{N^1 i}$ where N^1 is the parent of N and N is the i_{th} child of N^1.

Partition_Area (Vertex *v***),** this function divides the remaining area of a vertex *v* based on the total weight of its children. The partitioned areas are allocated for the branches rooted at *v*'s children.

Figure 4 shows an example tree. *R*, the tree's root, partitions the whole 90 degree angle to its children P(*v1), P(v2)* and P(*v3)*, the shade area in Figure 5 is the area given to the branch rooted at *N1***.** *N1* uses cutting edge $L_{N1,1}$ to recursively partition the area into *N11* and *N12* surrounded by a dashed line. Because *N11* and *N12* have the same weight by Equation <3>,they occupy the same sized areas.

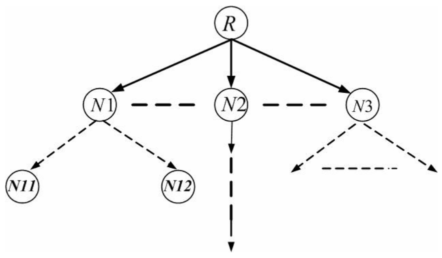

Figure 4 An example tree

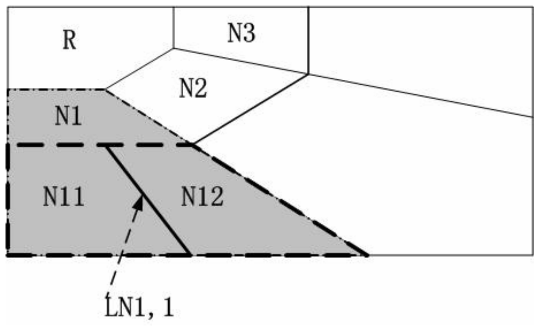

Figure 5 Drawing of the branch for the tree in Figure 4

3.3 Complexity Analysis

For a n-node tree, the complexity of function **Para_Creator** is O(n^2) because adjacent matrix is used for information structure storage. Th depth first search is used to traverse the tree, for each new node, functions **Test()**, **Polygonal_Node()** and **Partition_Area()** are applied. All these three functions are O(*1*). So the complexity of this function is O(n^2).

Using the above RELT algorithm, the tree structure is clearly displayed, as shown in Figure 6. For any node, its parent is in the upper left direction, and its children are in the lower right direction. The nodes at the same level locate along the dashed curve. In addition, this method recursively divides the whole display area, maximizes the screen usage. The next section presents an application of the algorithm.

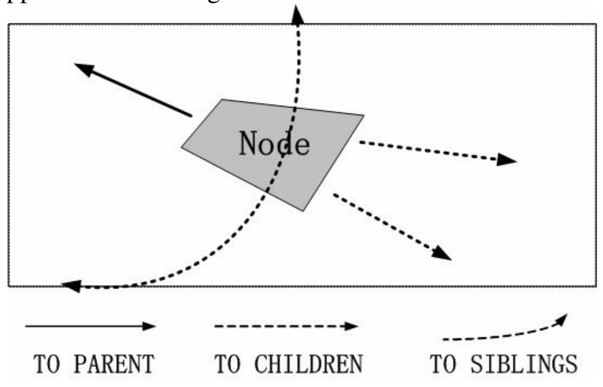

Figure 6 General directions in parent-child relationships

4 A Case Study: Music Selection

RELT works well for hierarchical information, especially with the overall structure revealed on a limited screen estate.

One of the current trends is to combine mobile phones with MP3 player. With the available storage capacity and improved sound effect, consumers can download many music pieces from the Internet. With the increasing amount of music selections available on the Internet, there is an urgent need for a commonly accepted music classification system that can assist navigation and selection. The most common approach is using a menu bar. The structure of menu bar is very simple, like artist – album – track and may be defined by users (Ipod, for example). It however does not show clearly the structure of the music categories.

Figure 7 shows an example music classification, whose. RELT display is shown in Figure 8.

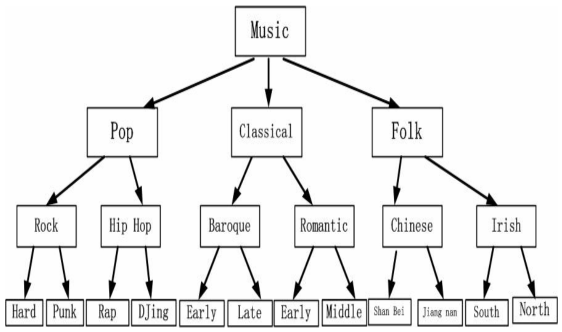

Figure 7 A music classification example – a balanced tree

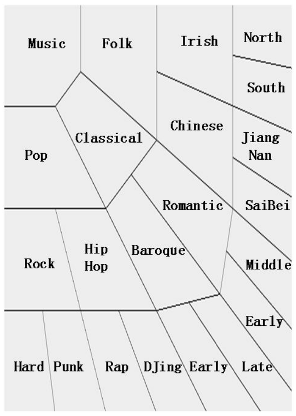

Figure 8 RELT for the example music classification in Figure 7

This method works well when the input tree is almost balanced. The smaller difference between $w^{max}(l)$ and $w^{min}(l)$, the maximal and minimal weights at level l, the tree is more balanced and the layout is more aesthetic.

The example shown in Figure 7 is totally balanced because all the nodes on the same level have the same weight. Figure 9 gives an unbalanced tree since the node Folk's (in a shaded ellipse) weight is 3 which is smaller than 5 of node Classical (in a white ellipse). This leads to an unaesthetic layout, as shown in Figure 10. Nodes Irish and Chinese, in shaded ellipses, are long, across several levels. Those long nodes make the hierarchical levels unclear. The next section discusses an optimization technique that reduces the number of such long nodes.

5 An Optimization

In an unbalanced tree, some leaf nodes may over-represent their areas. More specifically, such nodes take more than one level in the final RELT representation. Take the node "Chinese" in Figure 9 for example, it is a level 3 node but it covers across levels 3 to 5 as shown in Figure 10. The following subsections first introduce the notations and a layout estimate function, and then present the optimized algorithm in details.

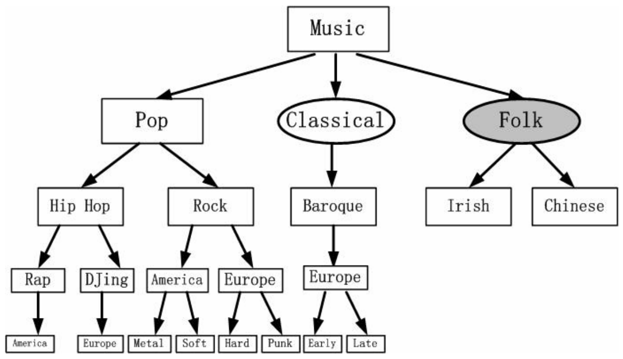

Figure 9 An unbalanced tree

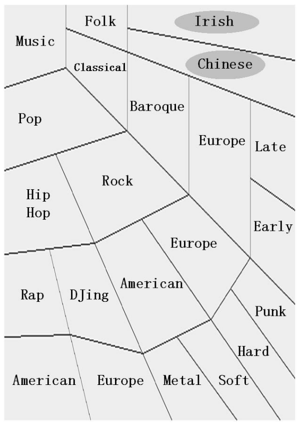

Figure 10 RELT for the example in Figure 9

5.1 Estimating Node Overrepresentation

The more number of levels over-represented by nodes in a RELT layout, the more misunderstanding of the tree structure may the layout lead to. The total number of over-represented levels is therefore a critical criterion in estimating the effectiveness of RELT for easy human perception.

Definition 1: Assume each leaf node is counted once, the *depth* of a node *N*, denoted *N.depth*, is the number of nodes from *N* (including *N*) to its nearest leaf.

Definition 2: A leaf node *N* is over-representing iff *N.depth* < *i.depth*, where *i* is another leaf node. We denote the set of over-representing leaf nodes as *OR*.

According to the RELT algorithm, non-leaf nodes will never over-represent. Every node in *OR* shares at least one of its boundaries with some other nodes at more than one level. For example, node "Chinese" in Figure 9 is in *OR* because it shares one of its boundaries with "Classical", "Baroque" and "Late" that are at different levels. Node "Irish" is not in *OR* because it shares each boundary with exactly one other node.

Definition 3: The number of *levels* over-represented by node *N* (in *OR*) is denoted L_OR_N. If both side boundaries of *N* are shared with *N*'s sibling nodes, L_OK_N is the sum of the levels on these two boundaries.

The estimate function NOR for overall node over-representation can be constructed as follows:

$$NOR = \sum L\_OR_N \qquad <2>$$

Clearly, a small NOR is desirable.

5.2 Minimizing Overrepresentation

We consider the *state* of a tree as the one that uniquely determines the parent-child relationships and ordered (left to right) sibling relationships. For example, exchanging a node's left and right children will change the tree state. A change of relative positions of any two nodes will change the tree's state. A given state of a tree uniquely determines its layout by the RELT algorithm.

To obtain the best RELT layout, we need to investigate how to obtain the best state of a tree. A tree's nodes are initially divided into groups according to their levels as described in Section 3.2. At each level, the nodes are numbered from left to right. In the following, we will use $TN\_Depth(l_i)$ to represent a function that measures the maximum difference between the depths of any two leaf nodes at a given level *i* for a tree state.

Let

- N_{ij} be the node at level *i* numbered *j*.
- $N_{ij}.depth$ be the depth of node N_{ij}.
- l_{num} be the number of levels of *T*.
- l_{i_num} be the number of nodes at level *i*.

$$TN\_Depth(l_i) = \sum_{j=1}^{l_{i\_num}-1} \left|N_{i(j+1)}.depth - N_{ij}.depth\right|$$

$$\left|N_{i(j+1)}.depth - N_{ij}.depth\right| = \begin{cases} \left|N_{i(j+1)}.depth - N_{ij}.depth\right| & (1) \\ 0 & (2) \end{cases} \qquad <3>$$

(1) If one of N_{ij} and $N_{i(j+1)}$ is a leaf and the other is a non-lead node.
(2) If both N_{ij} and $N_{i(j+1)}$ are leaves or are non-leaf nodes.

The following function measures the maximum difference in depth between any two leaf nodes for a tree state:

$$TN\_Depth(State) = \sum_{i=1}^{l_{num}} TN\_Depth(l_i) \qquad <4>$$

Theorem 1:
Let s_{old} and s_{new} be the old and new final states, and L_{old} and L_{new} be the corresponding old and new layouts of a tree, if $TN\_Depth(s_{new})$ is smaller than $TN\_Depth(s_{old})$, then $NOR(L_{new})$ is also smaller than $NOR(L_{old})$.

Proof: *According to Equation <4>, the depth difference between two adjacent nodes contributes to the function only if one node is leaf and the other is a non-leaf node. Thus the leaf node is over-representing in the layout and the depth difference is exactly the number of levels over represented. For a given state of a tree and its layout, the values of TN_Depth and NOR are the same. The only difference is that TN_Depth explores estimates based on the tree's state and NOR estimates the outcome layout.*

Now the strategy of minimizing overrepresentation becomes how to use function *TN_Depth(State)* to find the state that can result in the best layout of a tree. For a tree, the best state s_{BEST} satisfies Equation <5>:

$$TN\_Depth(s_{BEST}) = \min_{s_i \in S} TN\_Depth(s_i) \qquad <5>$$

where *S* is the set of all the tree's states.

Next finding the minimum number of *TN_Depth(State)* becomes the key. This can be done in two steps. The first step is expressed in Equation <3> and the second step in Equation 4. To simplify the problem, a basic assumption is initially constructed. Equation <4> will achieve its minimum point when all the results for each level in Equation <3> are the smallest. The process is then simplified to how to obtain the smallest number for each level in Equation <3> as described below.

Definition 4: A *unit* is the set of all the siblings who have the same parent.

A unit's position at its level is the number counted from left to right. Let U_{ij}, denotes the unit at level *i* and position *j*. For example, in Figure 10, "Hip Hop" and "Rock" make up a unit U_{31}, "Classical" is a unit U_{32} by itself, and "Chinese" and "Irish" make up the unit U_{33}.

Assume level *i* contains *m* units and each unit has exactly two nodes, a naive way to obtain the best layout is to compute overrepresentation by Equation <4> for the all the unit combinations and choose the state with the smallest depth. The complex is $O(2^m)$ because there are two possibilities in each unit. It is explicitly not a good choice especially when *m* is large.

Theorem 2:
For a single unit, Equation <3> derives the minimum number if the nodes are sorted by their depths.

Proof: *If there is initially a sorted unit including m nodes and then two nodes D_i and D_j are exchanged.*

U_{SORT}:

$$(D_1, D_2 \ldots D_{i-1}, D_i, D_{i+1} \ldots D_{j-1}, D_j, D_{j+1} \ldots D_m)$$

$$1 \leqslant h \leqslant t \leqslant m \quad D_h \leqslant D_t$$

U_{UNSORT}:

$$(D_1, D_2 \ldots D_{i-1}, D_j, D_{i+1} \ldots D_{j-1}, D_i, D_{j+1} \ldots D_m)$$

$$\begin{aligned}
&TN\_Depth(U_{SORT}) - TN\_Depth(U_{UNSORT}) \\
&= (/ D_i - D_{i-1} / + / D_{i+1} - D_i / + / D_j - D_{j-1} / + / D_{j+1} - D_j /) \\
&\quad - (/ D_j - D_{i-1} / + / D_{i+1} - D_j / + / D_i - D_{j-1} / + / D_{j+1} - D_i /) \\
&= (D_{i+1} - D_{i-1} + D_{j+1} - D_{j-1}) - (2 D_j - D_{i-1} - D_{i+1} + D_{j-1} + D_{j+1} - 2 D_i) \\
&= 2(D_i + D_{i+1}) - 2(D_{j-1} + D_j) \geqslant 0
\end{aligned}$$

$TN\_Depth(U_{SORT}) = TN\_Depth(U_{UNSORT})$ becomes true when $i + 1 = j$ or $D_i = D_j$.

Now we can derive an optimized RELT algorithm. Based on Theorem 2, function **Sort (Node** *N***)** is inserted between **Polygonal_Node(Node** *N***, int** *L***)** and **Partition_Area(Node** *N***).**

For every new non-leaf node *N,* function **Sort (Node** *N***)** sorts its children by their depths. Assuming the j_{th} non-leaf node has n_i children, the complexity of function **Sort (Node** *N***)** is $\sum_{j=1}^{n_{in}} n_j \lg n_j$.

Proof of complexity is given as follows:

$$\sum_{j=1}^{n_{in}} n_j \lg n_j \leqslant \sum_{j=1}^{n_{in}} n_j \lg n_{in} \leqslant \lg n_{in} \sum_{j=1}^{n_{in}} n_j$$
$$< n \lg n_{in} < n \lg n$$

So, the complexity of optimized algorithm is O(n^2) +O(*nlgn*).

6 Conclusions and Future Work

The small screen on mobile devices severely challenges the traditional hierarchy information visualization methods for desktop screens. Based on the analysis of the traditional tree visualization method, we observe the following:

- The two major approaches, i.e. connection and enclosure, for tree visualization are no longer suitable for mobile devices without proper adaptation.
- To achieve both a clear hierarchical structure and the maximum use of the display area, the current algorithms for desktop displays do not work well.

This paper has presented the RELT approach to tree visualization on small screens. The RELT algorithm traverses the tree with depth first, recursively partitions the remaining area, and allocates each partitioned area for a node. In this way, the entire display area is fully used while the hierarchical structure is clearly visualized.

The algorithm has been tested on several tree applications. Although our evaluation work is still underway, this approach has demonstrated to be feasible for visualizing tree-based hierarchical information on mobile devices. We plan to further optimize the RELT algorithm for more aesthetic layout while maintaining the clear tree structure. Adaptive layout features in response to browsing and labeling techniques will also be investigated.

Reference:

1. P. Eades, "Drawing Free Trees", Bulleting of the Institute for Combinatorics and Its Applications, 1992, pp.10-36.
2. G. di Battista, P. Eades, R. Tamassia, and I. G. Tollis, *Graph Drawing: Algorithms for the Visualization of Graphs*. 1999, Prentice Hall.
3. I. Herman, G. Melançon, and M.S. Marshall, "Graph Visualization in Information Visualization: a Survey", *IEEE Transactions on Visualization and Computer Graphics*, 2000, pp. 24-44.
4. C. S Jeong and A. Pang, "Reconfigurable Disc Trees for Visualizing Large Hierarchical Information Space", *Proc. 1998 IEEE Symposium on Information Visualization (InfoVis'98)*, IEEE CS Press, 1998, pp.19-25.
5. B. Johnson and B. Shneiderman, "Tree-maps: A Space-filling approach to the visualization of hierarchical information structures", *Proc. 1991 IEEE Symposium on Visualization*, IEEE, Piscataway, NJ, 1991, pp. 284-291.
6. C. C. Lin and H. C. Yen, "On Balloon Drawings of Rooted Trees" *Proc. 13th International Symposium on Graph Drawing (GD'05)*, Limerick, Ireland, September 12-14, 2005. pp 285-296.
7. Q. V. Nguyen and M. L.Huang, "A Space-Optimized Tree Visualization" *Proc. 2002 IEEE Symposium on Information Visualization (InfoVis'02)*, pp.85-92.

8. E.M. Reingold and J.S. Tilford, "Tidier Drawing of Trees," *IEEE Trans. Software Eng.,* Vol. 7, No. 2, 1981, pp. 223-228.

9. B. Shneiderman, "Treemaps for Space-Constrained Visualization of Hierarchies", December 26, 1998, last updated April 26, 2006, http://www.cs.umd.edu/hcil/treemap-history/.

10. J. Yang, M.O. Ward, and E.A. Rundensteiner "InterRing: An Interactive Tool for Visually Navigating and Manipulating Hierarchical Structures", *Proc. 2002 IEEE Symposium on Information Visualization (InfoVis'02)*, pp.77-84.

11. K. P. Yee, D. Fisher, R. Dhamija, and M.Hearst "Animated Exploration of Dynamic Graphs with Radial Layout", *Proc. 2001 IEEE Symposium on Information Visualization (InfoVis'01)*, San Diego, CA, USA, 2001, pp.43-50.

12. H.Y. Yoo and S.H. Cheon, "Visualization by information type on mobile device", *Proc. 2006 Asia-Pacific Symposium on Information Visualization* - Volume 60, 2006, pp. 143-146.