A Space-based Layout Algorithm for the Drawing of Co-citation Networks

Arnaud Quirin and Oscar Cordón*

Abstract

We present in this paper a drawing algorithm to represent graphically co-citation networks (scientograms). These networks have some interesting and unusual topological properties which are often valuable to be visualized. In general, these networks are pruned with a network scaling algorithm, then visualized using a drawing algorithm [3]. However, typical drawing algorithms do not work properly, especially when the size of the networks grows. Edge crossings appear while the drawing space is not adequately filled resulting in an unsightly display. The approach presented in this paper is able to print the networks filling all the available space in an aesthetic way, while avoiding edge crossings. The algorithm is detailed and compared with the classical Kamada-Kawai drawing algorithm on two maps.

1 Introduction

Social networks have some interesting and unusual topological properties which are often valuable to be printed graphically. However, the raw networks cannot be often visualized easily, especially when their size grows proportionally with the number of data to be dealt with, and thus specific algorithms for simplifying such large networks have been developed. Network scaling algorithms, whose goal is to take proximity data and to obtain structures revealing the underlying organization of those data, use similarities, correlations or distances to prune a network based on the proximity between a pair of nodes. One of the most known, the Pathfinder algorithm, is used frequently due to its various mathematical properties [4]. The resulting network could then be graphically represented using a network drawing algorithm. The Kamada-Kawai [6] or the Fruchterman-Reingold [5] algorithms are usually applied for this task. This methodology ensures the network is represented in an aesthetic way, by the mean of some spatial constraints. If the network represents the complex tissue of relationships between individuals or organizations, we may be interested in viewing some specific properties. For instance, the backbone could be better highlighted if it is drawn in the center, and minor links could be represented in the border of the map.

There are some kinds of Social Network Analysis (SNA) applications which could benefit from such methodology, giving to the domain expert or even a simple user a simple access to the information contained in these networks. One of these applications is co-citation network analysis. Co-citation network models depict the complex tissue of relationships occurring in the scientific literature. The graphical representation of these kinds of networks while preserving their information is still a challenge. However, some work has been done using *scientograms* [3], visual representations showing the spatial distribution of the scientific actors in a given domain, where these actors can be as diverse as scientific categories, authors, journals or papers. Because of the complexity of the domain they aim to represent, these maps usually contain a large number of links and are really dense and hard to be directly represented.

The said methodology has already been described in the literature for the design of the scientograms [12]. But, until so far, no attention has been paid to the drawing algorithm itself. Classical drawing algorithms suffer from some aesthetic problems. For instance, the Kamada-Kawai algorithm has no explicit procedure to avoid edge crossings, and the Fruchterman-Reingold algorithm does not fill properly the full space allocated for the drawing. In fact, for the analysis of scientograms, the drawbacks of these algorithms could prevent an expert from an optimal interpretation of the relationships taking place inside the considered scientific domain. For instance, edge crossings can make a node and its labels overlap, thus avoiding a good reading of the map. Another point is the absence of some specific spatial constraints to avoid the links going back to the center of the map. Spatial artifacts, such as nodes appearing close together even if they are spatially-separated by several links, could convey false or misinterpreting information.

In this paper, we propose a new drawing algorithm to

*A. Quirin and O. Cordón are with European Centre for Soft Computing, Edificio Científico-Tecnológico, planta 3. Gonzalo Gutiérrez Quirós, s/n, 33600 - Mieres (Asturias), SPAIN {arnaud.quirin, oscar.cordon}@softcomputing.es

overcome these drawbacks. The structure of the current contribution is as follows. In the second section, we review the existing methodologies to design scientograms. In the third section, we describe our proposal. In the fourth section some experiments will be shown. Finally, some concluding remarks are pointed out in the last section.

2 A Methodology to Generate Scientograms

The achievement of a vast scientogram is a recurrent idea in the modern age. In 1998, Chen [3] was the first researcher to bring forth the use of Pathfinder Networks (PFNETs) in citation analysis. This is due to the fact that scientograms are the most appropriate means to represent the spatial distribution of research areas, while also affording information on their interactions [11]. Taking the latter as a base, Vargas and Moya [12] proposed a method for the visualization and analysis of vast scientific domains using the ISI[1]-JCR category co-citation information. They represented it as a social network, simplified that network by means of the Pathfinder algorithm, and graphically depicted its layout using the Kamada-Kawai algorithm, thus getting a structural model of the scientific research in a vast domain.

The different method stages are briefly described as follows. The last step is the one replaced by our proposal.

2.1 Category co-citation measure

Co-citation is a widely used and generally accepted technique for obtaining relational information about documents belonging to a domain. Because we strive to represent and analyze the structure of vast domains, whether they be thematic, geographic or institutional, we fall back on to ISI-JCR co-citation categories [12] as a tool for this purpose.

Hence, once the rough information of the ISI-JCR co-citation for the categories present in the domain to be analyzed is obtained, a co-citation measure CM is computed for each pair of categories i and j as follows:

$$CM(ij) = Cc(ij) + \frac{Cc(ij)}{\sqrt{c(i) \cdot c(j)}} \qquad (1)$$

where Cc is the co-citation frequency and c is the citation frequency.

2.2 Network pruning by Pathfinder

Then, the Pathfinder algorithm is applied to the co-citation matrix to prune the network. Since co-citation networks are usually very dense, Pathfinder is parameterized to $r = \infty$ and $q = n - 1$, in order to obtain an schematic representation of the most outstanding existing information by means of a network showing just the most salient links. In general, the weights of the links of the co-citation matrix belong to $\mathbb{R}$ and are all different, so the final result of the Pathfinder algorithm is a tree. To perform this step, the MST-Pathfinder algorithm, a quick version of the original Pathfinder algorithm based on *Minimum Spanning Trees* is used [10].

[1]Currently registered as *Thomson Scientific*.

2.3 Network layout by Kamada-Kawai

Kamada-Kawai algorithm [6] is then used to automatically produce representations of the pruned network resulting from the Pathfinder run on a plane, starting from a circular position of the nodes. It generates social networks with aesthetic criteria such as common edge lengths, forced separation of nodes, building of balanced maps, etc. Nevertheless, the satisfaction of some criteria are not directly implemented in the Kamada-Kawai algorithm. This is the case of the number of crossed links: in fact many links crossings appear making a lot of nodes overlap, and making the reading of the map harder. Another point is the fact that edges can go backwards to the center of the map, putting close two nodes linked by a long path. This can give a false impression of closeness to the expert due to the spatial distribution of the nodes. These are two of the main drawbacks we observed while using the Kamada-Kawai visualization applied to co-citation networks. We aim to solve them using a new visualization algorithm, as we will see in the remainder of this paper.

3 Overview of the algorithm

This section describes our drawing algorithm. As said, our methodology ensures that the result of the Pathfinder algorithm, the network we have to draw, is a tree. Thus, to develop our algorithm, we took as a base the tree visualization algorithm presented in [8], and extend it to make it applicable on scientograms. The basic version of the algorithm is first presented, then several variants are discussed for the specific case of scientogram design.

3.1 Main Algorithm

To ease the understanding of our proposal, some preliminary terminology is first introduced. In the following, we consider an ordered tree T, in which each node N has a parent P=PARENT(N), except the root node R=ROOT(T). CHILDREN(N) is the set of nodes having node N as their parent. ASCENDANT(N) is the chain of nodes from node N to the root node R, defined as { N, PARENT(N), PARENT(PARENT(N)), ..., R }. SUBTREE(N) is the subtree having node N as its root. SIZE(N) is equal to the number of nodes in SUBTREE(N), including its own root.

For instance, SIZE(N) is equal to 1 for a node having no children; 2 for a node having one child; etc. LEVEL(N) is the number of nodes in the set ASCENDANT(N). For instance, LEVEL(N) is equal to 1 for the root node; 2 for any of the children of the root node; etc. DEPTH(N) is the maximum value for LEVEL(M) for any node M in the tree SUBTREE(N). For instance, DEPTH(N) is equal to 1 for a node having no child; 2 for a node having any number of children, with none of them having children; etc. By convention, we will also use the notation ROOT(T) to define the node having the lowest level in a subtree T.

The algorithm is named *Vmap-Layout* due to the kinds of maps it draws, *Visual Science Maps*, another name for scientograms. It is divided itself in three sub-functions which are called in a sequential way. The first sub-function, *Attribute-Computation*, computes for each node the attributes needed for the remainder of the algorithm. The second sub-function, *Node-Positioning*, is a recursive function aiming to compute the coordinates of each node. The last one, *Node-Relocation*, adjusts the location of the nodes according to some specific criteria, thus improving the final visualization. The different sub-functions are detailed in the following sub-sections.

3.2 Attribute Computation

With the first sub-function, we compute several attributes assigned to each node: SIZE(N), LEVEL(N), and DEPTH(N). These attributes will be used later to facilitate the generation of the coordinates of each node and to improve the runtime of the algorithm. The first step is to select a root node, the one that will be printed in the center of the map. It will be used to compute some specific attributes which cannot be computed without the definition of a root node. The network generated by the Pathfinder algorithm does not self-contain any root node, thus we have to use an additional technique to select one. There are many ways to select a center in a graph. Many of them are described in the papers by Bavelas [2] and Parlebas [9]. The one used here, that gives good visual results, is the *deliverer criterion*: we compute the sum of the distances between any node and all the others, and we take as the root the one having the smaller value. Once we have selected the root node R, we can assign to each node N the values corresponding to SIZE(N), LEVEL(N) and DEPTH(N). As the tree is represented in memory using lists of children, the time complexity of all these operations is $O(n)$ where $n = SIZE(R)$. The complexity of *Attribute-Computation* is thus $O(n)$.

3.3 Node Positioning

Using the second sub-function, the algorithm fixes the location of each node as a pair of 2D coordinates. To do so, the global idea is to fill as much space as possible. The tree is drawn from the root node to the leaves, and the algorithm runs in a recursive way: the root node is drawn in the center of the map and at each iteration the algorithm draws all the nodes N having the same level L=LEVEL(N). The algorithm starts by selecting a region of the empty space in which it can draw the tree (we will call this region *initial polygon* in the following). Then, it assigns the root node to the center of this polygon, it divides the initial polygon into several slices (as many as the number of children in the root node), and it assigns one children of the root node to each slice. After the application of this function, the coordinates of the center of the polygons are assigned to each node.

There are several ways to design the initial polygon. As the whole tree will lie inside the initial polygon, the later will determine the final shape of the full map. This shape could be a square, a circle or any other n-sided polygon. Fig. 4 shows some possibilities for the initial polygon. The assignation of a center C to a given polygon could also be done in several ways. Some techniques are shown in Fig. 6. Lastly, the way of dividing a given polygon into different slices can also been done in several ways, which are detailed in the next sections.

Once the coordinates of the first level of the tree are fixed, the algorithm starts again, considering each of the slices as a new polygon and the corresponding node N as the root of a new subtree S=SUBTREE(N). Once no children has been found, the algorithm stops. The sub-function is outlined in Fig. 1. As we are using a recursive function based on the children on a given node, the complexity of *Node-Positioning* is thus $O(n)$.

1. Let P, a 2D-space region in which to draw the tree, and T, a tree.
2. Choose a central point C in P and assign to this point the root of the tree R=ROOT(T).
3. If CHILDREN(R) is empty, stop.
4. Divide P in different slices (*sub-polygons*), giving as many sub-polygons that the number of children of R. Let R_i be a child of R, the area of the corresponding sub-polygon P_i should be proportional to SIZE(R_i).
5. For each child R_i of R, run *Node-Positioning* on the region P_i and the tree R_i.

Figure 1. The *Node-Positioning* sub-function.

At the end of this sub-function, all the nodes of the tree have been assigned to a pair of coordinates in the 2D-space. An example of the execution of this sub-function on a tree is shown in Fig. 2.

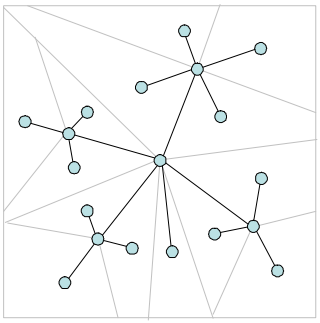

Figure 2. An example of the execution of the *Node-Positioning* sub-function.

3.4 Node Relocation

The goal of the third sub-function is only aesthetic. At the end of the application of the previous sub-function, some graphical elements, such as the nodes or the text labels, can overlap. Here, having the coordinates of the nodes generated previously, the algorithm fixes the final location of each node to avoid the presence of overlaps as much as possible. We say that two nodes overlap when they are too close from each other, according to a distance defined by the expert, and we call them *problematic* nodes. Problematic nodes cause text labels not to be read in a clear way, and reduces in general the readability of the map. The main idea of this function is to apply a node relocation process in which the problematic nodes are moved according to a repulsive force depending on the surrounding nodes, like if they were connected with repulsive springs. To avoid deadlocks in some cases (for instance, when a node is located exactly between other two nodes and at an equal distance), the problematic nodes are also slightly moved in a random direction, until they met a criterion set by the expert.

The sub-function is outlined in Fig. 3. As the time complexity of the KD-Tree preprocessing is $O(n \cdot log(n))$ and the time complexity of the KD-Tree search for one node is $O(log(n))$ [1], the time complexity of *Node-Relocation* is $O(n \cdot log(n))$. Thus, the complexity of the full algorithm is $O(n \cdot log(n))$.

At the end of the third function, the final coordinates of the nodes have been computed. During the final drawing of the nodes, additional improvements could be made in order to improve the aesthetic aspect of the map. For instance, nodes and labels sizes can be varied depending on their depth in the tree to better highlight the center of the map.

1. Apply a KD-Tree technique to compute the distance between all the nodes of T.
2. Select only the problematic nodes, i.e. the nodes close enough according to a criterion defined by the expert.
3. For each node of this set, do:
   - Apply a repulsive strength δ and move the node along this force.
   - Move it around its final position using a small random distance in the interval $[-\sigma,\sigma]$.
4. Execute again the *Node-Relocation* sub-function until a given amount of iterations defined by the expert has been reached.

Figure 3. The *Node-Relocation* sub-function.

3.5 Selecting different initial polygons

During the initialization of the Vmap-Layout algorithm, we have to select the initial polygon, within which the full tree has to be drawn. This polygon encloses all the layout and its shape will determine the global shape of the drawing.

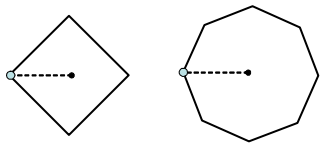

Figure 4. Initial polygons with 4 or 8 sides.

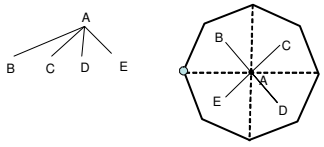

Figure 5. On the left, the tree to draw. On the right, the initial polygon and the center in black. The polygon is first divided in four slices, because the node A has four children, then we assign the corresponding nodes to the corresponding sub-polygons. The little circle is the starting point giving the direction of the polygon assignment.

In order to improve the general aspect of the final map and to customize the result for several uses (paper or on-

line drawing), several options can be used. The definition of this shape is controlled by an expert parameter, giving the number of sides the inital polygon should have (see Fig. 4). The larger this value is, the more circular the shape will be, but the slower will the algorithm run. This is due to the fact that, as at later stages, the computation of the areas and the angles of a sub-polygon would be more complex. This number of sides does not change in any manner the further execution of the algorithm but has only an aesthetic aspect.

Once the initial shape is designed, it is used to draw the initial tree, composed of its root and of all its first-level children (see Fig. 5). Any shape surrounding the graph could be used. For the application to co-citation networks, we opted by a circle because the SCImago Research Group[2] experts, with whom we collaborate, prefer this shape. Thus, in order to have a good compromise between time and aesthetic, a value of 15 sides seems to be well suited. Larger values than 30 will unnecessarily increase the runtime and smaller values than 12 would give an impression of dicontinuity.

3.6 Selecting different ways to compute the central point of a polygon

For each polygon, a central point has to be selected to become the starting point of the next sub-tree to print (see step 2 in Fig. 1). We have explored at least three different methods to choose the central point C of a polygon P (see Fig. 6). The first method, called '*Center of Mass*', takes for C the center of gravity of P. This center is defined in any case, but suffers from two problems: it could be outside of the polygon (this can occur when the polygon is non-convex) and a polygon with a lot of segments could attract the center far away from the *natural* center of the polygon. The second method, called '*Angle-based Central Point*', uses the angle to compute the central point. For a given polygon P, we first select a point on its border, that we call origin O. This origin O has itself to be defined by some methods. For instance, the origin could be the center of the parent polygon (the one used to generate the current polygon), or the left-most point of the polygon. We then draw a line dividing the angle O in two equal parts. Then we take the middle point of this line as the center C of the polygon. The third method, called '*Area-based Central Point*', applies the same procedure, but by dividing the polygon into two parts having the same area.

Our tests have shown that the *Area-based Central Point* is the best method. However, some problems could occur for some specific shapes of polygons in which it is impossible to divide a sub-polygon into two areas of equal sizes, pushing the central point C outside the polygon. This can happen when the polygon has internal angles greater than π (see Fig. 7), when the chosen initial polygon is non-convex

[2] http://www.scimago.es/

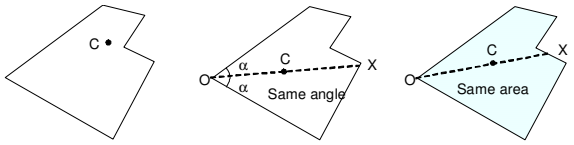

Figure 6. Centers positioned with the *Center of Mass*, the *Angle-based Central Point* and the *Area-based Central Point* methods.

or when the structure of the tree is quite uncommon. This is why other methods are provided.

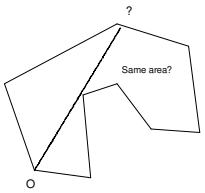

Figure 7. A problem that can occurs with the *Area-based Central Point* method: the two areas cannot be made equals.

For the *Angle-based Central Point* and the *Area-based Central Point* methods, once the line dividing the polygon in two parts has been determined, we still have to place the point C over this line. The usual way is to use the middle of the line, as described in the first paragraph of this section. But other values for the measure of the distance OC have been explored. This distance has a direct influence on the length of the edges and the location of the next sub-polygons, and playing with this value can lead to interesting results on the final drawing. An expert parameter has been set for this measure and is named *Cutpoint Value*. It is the ratio between the distance OC and the distance OX (see Fig. 6). With a small value for this parameter, we get maps where the centers are close among them, and with a larger value, we get maps where the centers are more far away among them. Several values for this parameter have been tried, such as 0.25, 0.5 and 0.6. The best results have been obtained with values lower or equal to 0.5.

3.7 Selecting different dividing slice methods

The way to divide a polygon in different slices (see step 4 in Fig. 1) will determine the respective areas for the drawing of the next sub-polygons. Small areas should be allocated to small sub-trees whereas larger areas should be allocated to larger sub-trees. This operation can be achieved by at

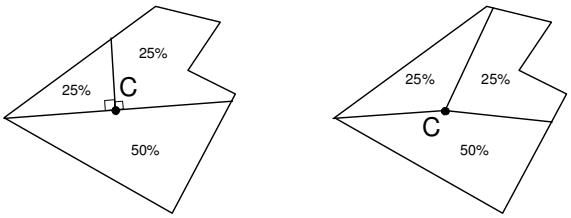

Figure 8. The result of the dividing using the *Angle-based Dividing* (on the left part) and the *Area-based Dividing* (on the right part) methods.

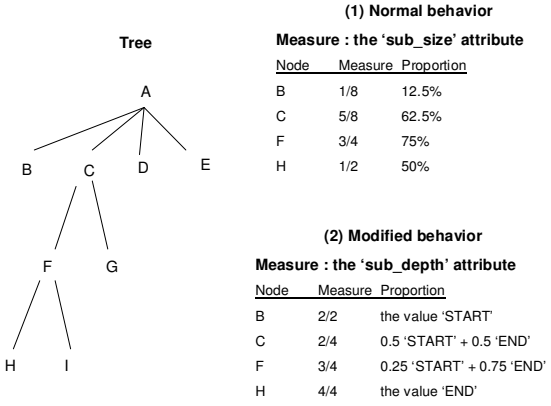

Figure 10. Different ratio computation methods for dividing the slices: an example of a tree (left); the proportions obtained using the *Sub-depth-based Ratio Computation* method (top); and the proportions obtained using the *Sub-depth-based Ratio Computation* method (bottom).

least two methods (see Fig. 8). With the first method, called *Angle-based Dividing*, the algorithm defines the size of the slices in order the angle around the center C is proportional to the size of each sub-tree R_i. With the second method, called *Area-based Dividing*, the algorithm defines the size of the slices in order the area around the center C is proportional to the size of each sub-tree R_i. In many scientograms generated using real world data, the second method does not work properly because the non-convex shapes of the polygons make the finding of a percentile using the area an impossible problem (see Fig. 7). Therefore, in our application, we always used the *Angle-based Dividing* method.

Figure 9. On the left, the normal behavior in which all the space is used to compute the size of each slice. On the right, a modified behavior in which a constraint is applied before computing the size of each slice, allowing us to direct the network in a given way.

Any use of the two previously described methods needs a measure defined for each node in order to compute the proportion in percentage allocated to the corresponding sub-polygon. The simplest way, called the *Sub-size-based Ratio Computation*, is to take the size of each node (defined by the SIZE(T) attribute), i.e. the number of nodes in a subtree T, to compute a proportional ratio assigned to the children of T. Then, this ratio is used as a percentage to compute the size of all the slices of the corresponding sub-polygons. Nevertheless, this method suffers from a lack of customization possibilities by the expert.

Another method, called the *Sub-depth-based Ratio Computation*, has been explored. It uses the depth of the trees, defined by the DEPTH(T) attribute, to modify the proportion allocated to each slice depending on if they are close or far away from the center of the map. The expert has to set two additional parameters, the proportion given for the allocation of the slice of the lowest level, corresponding to the initial root of the tree (this value has been named 'START'), and the proportion given for the allocation of the slice of the deepest level, corresponding to a given leaf of the tree (this value has been named 'END'). Because only these two values, START and END, have to be specified by the expert, the remaining values used to fix the proportion of the intermediate levels are computed using a linear regression. Using another point of view, the START value fixes the behavior of the nodes close to the center of the map (or the backbone), which are the most important ones, and the END value fixes the behavior of the minor nodes shown in the periphery. These parameters are thus useful for our application of co-citation networks. We have obtained the best results using 0.5 and 0.25 for the respective values of START and END.

The behavior of the *Sub-depth-based Ratio Computation* method is very simple. It allows us to constrain the angle of the links to go only forward when we are close to the center of the map (see Fig. 9). In fact, when drawing scientograms, having edges going mainly forward gives a better representation as nodes located far away in terms of number of edges are spatially dissociated. This is why we selected the *Sub-depth-based Ratio Computation* method as the default one for our experimentations.

To show how the proportion of the slices are computed using the *Sub-size-based Ratio Computation* and the *Sub-depth-based Ratio Computation* methods, an example is presented in Fig. 10 using a small tree. For some selected

nodes, the values are computed for both methods. The proportion is always computed as a ratio between two numbers, the current attribute of the node (respectively the SIZE and the DEPTH) and the maximum value for this attribute (so, the maximum size of the current sub-tree or the maximum depth if the second measure is considered). Note that the ratio computation method is totally independent from the method of dividing these slices, i.e. the ratio can be used independently with the *Area-based Dividing* or the *Angle-based Dividing* methods.

3.8 Details on the *Node-Relocation* function

Once the coordinates of the nodes have been found by the *Node-Positioning* sub-function (see Fig. 1), a node relocation stage occurs in order to improve the location of the overlapping nodes. This stage is done by the *Node-Relocation* sub-function (see Fig. 3).

The goal of this sub-function is to identify the nodes which are too close, according to an expert criterion, and to move them randomly in order to avoid the overlapping of the nodes. The process is iterated several times until a perfect configuration is found by the algorithm. Because of the cost of the computation of the distance between nodes, and since this process has to be iterated, we use a KD-Tree technique to compute these distances[3].

The *Node-Relocation* sub-function works as follows. A parameter named *radius* is defined by the expert to specify the minimum allowed distance between two nodes. Only the coordinates of any node having a smaller or equal distance to this radius will be modified by the algorithm, while the coordinates of the nodes located at a greater distance will not be changed. Two additional parameters are defined, the *spring-strength* δ giving the strength of the movements during the relocation of the nodes and the *random-strength* σ giving the quantity of randomness applied to the nodes that have to be relocated.

From each node N of the set defined by the *radius* parameter, we apply a force defined as the sum of all the repulsive forces generated by the nodes close to N, multiplied by the value defined by the *spring-strength* δ parameter (a repulsive strength), and add a random value chosen into the interval $[-\sigma,\sigma]$ to it. Fig. 11 shows how the repulsive forces generated by all the surrounding nodes apply on a given node, and how this node is moved. This is done until a given number of iterations have been completed. A higher value for this parameter can be used to establish the convergence of the coordinates of the nodes, but at the cost of slowing down the process. We have obtained good and fast results with a value of 100 iterations.

The *spring-strength* δ parameter is used to define the step size of movement applied to the node. A small value moves the nodes slowly through the iterations of the algorithm, while a bigger value allows sudden changes of the coordinates of the nodes. A value of 0.10 was used in our experiments. The *random-strength* defines the quantity of randomness applied to the location of the node at the end of each iteration. A value of zero disables any randomness during the movement of the nodes. A value of 0.05 was used.

[3] Actually, the ANN Library has been used for this purpose [7].

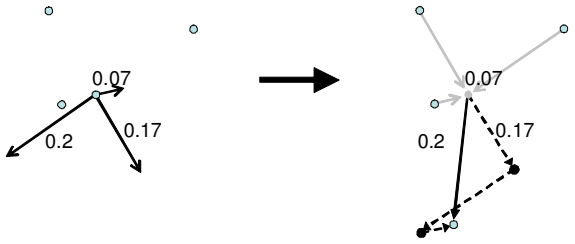

Figure 11. An example of the modification of the coordinates of a node after applying the *Node-Relocation* function.

4 Experiments

In this section, we will show the results obtained on two maps. The first map is a random network of 300 nodes. The second one is a network of 218 nodes extracted from a database of co-citation measures for Europe, generated in 2002. The resulting file encodes a fully connected network, with labeled nodes and weighted links, ready to be pruned. Thus, the first step is to use the MST-Pathfinder algorithm to prune these networks in order to get trees. The computing time for this step is roughly 9 ms on an Intel dual-core Pentium 3.2 GHz with 2 GB of memory.

The last step is to print the maps using the Vmap-Layout algorithm. The algorithm has been written in C++, compiled on Linux with the GNU `GCC` compiler with the `-O3` option, and is available upon request. The computing time using the Vmap-Layout algorithm is roughly 125 ms for the random map and 69 ms for the real-world scientogram. The main parameters used are as follows. The initial polygon has 15 sides. The central point has been computed using the Angle-based Central Point method. The Cutpoint Value has been set to 0.5. The slices have been divided using the Angle-based dividing method. Finally, the Sub-depth-based Ratio Computation has been used to position the central point in the polygon.

A comparison has been performed using the Kamada-Kawai algorithm. For this purpose, we have used the GraphViz library. GraphViz is an open source network drawing software, freely provided by AT&T Labs, and available at: http://www.graphviz.org/. It integrates the

Kamada-Kawai algorithm in the form of the *neato* utility. The computing time is roughly 1540 ms for the random map and 660 ms for the scientogram.

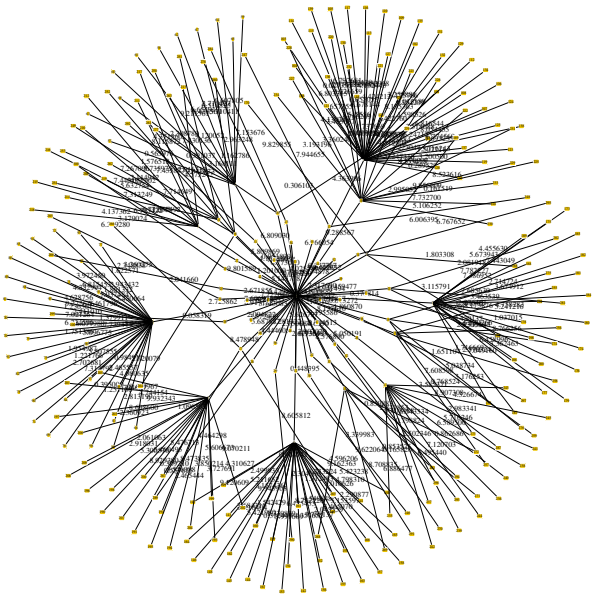

Figure 12. The scientogram of a random map, drawn with the Kamada-Kawai algorithm.

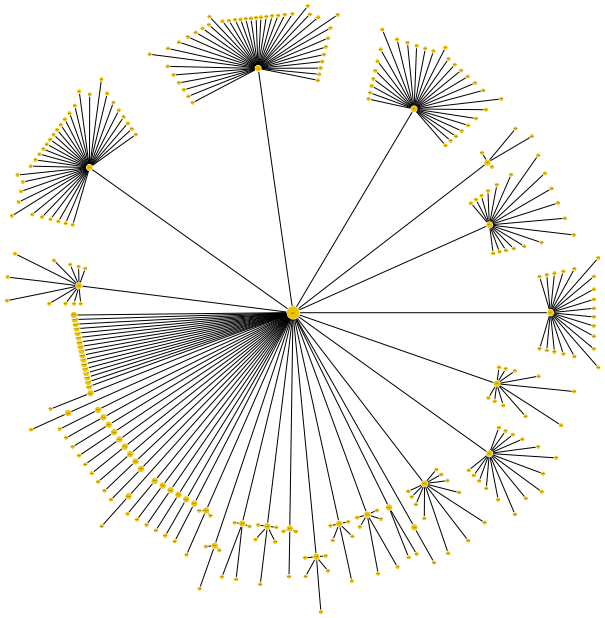

Figure 13. The scientogram of random map, drawn with the Vmap-Layout algorithm.

Figure 14. The scientogram of Europe in 2002, drawn with the Kamada-Kawai algorithm.

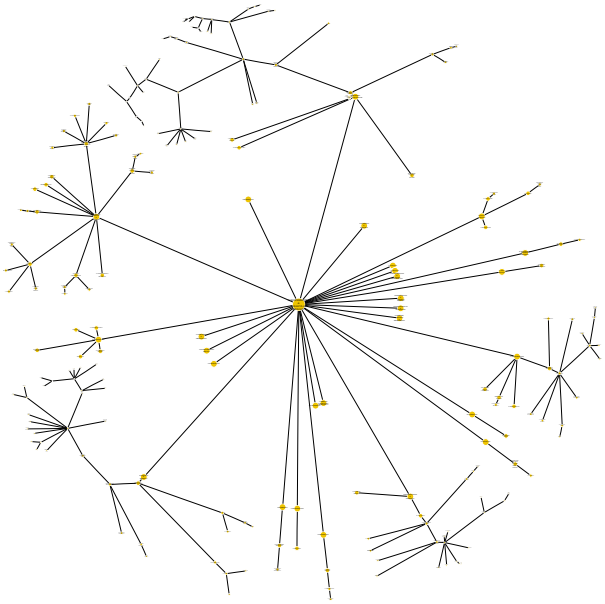

Figure 15. The scientogram of Europe in 2002, drawn with the Vmap-Layout algorithm.

The final results, for the Vmap-Layout and the Kamada-Kawai algorithms are shown in Fig. 12 and Fig. 13 for the random map and in Fig. 14 and Fig. 15 for the Europe scientogram. Notice that, the comparison of two visualiza-

tions is a subjective process and we are actually collaborating with the SCImago Research Group for the validation of the results. Nethertheless, several remarks can be made from these pictures. First, we have to notice that the Vmap-Layout algorithm avoids the edge crossings as expected, as each sub-tree is drawn in its own sub-space. Secondly, the nodes connected to the central node are properly spaced, and aligned on its own circle, allowing an expert to read properly the labels. In the case of the Kamada-Kawai maps, the reading of the top-level labels is not clear. Finally, all the space available in the figure is properly filled with the Vmap-Layout in the sense that more space is given to the larger sub-trees.

5 Conclusion

The Vmap-Layout algorithm is an effective and a fast technique - at least 10 times - for the representation of co-citation networks. Our algorithm can print real world networks in an aesthetic way, highlighting the backbone and pushing the less important links to the boundaries. Several variants have been described, allowing an expert to tune the representation depending on his needs. The only limitation we found with this approach corresponds to deep and large subtrees, which cannot be printed properly on a large area due to the recursive partitioning process.

We are currently investigating other improvements of this algorithm. One option is to allow it to use extra space over the polygons in order to reduce the white space between the edges. Another option is the use of different techniques for the node relocation, for instance based on the simulated annealing metaheuristic, to find a better positioning of the nodes. We are also planning some experiments on other kinds of real network datasets.

References

[1] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Y. Wu. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *J. ACM*, 45(6):891–923, 1998.

[2] A. Bavelas. *Réseaux de communications au sein de groupes placés dans des conditions expérimentales de travail, Les sciences de la politique aux États- Unis*. Armand Colin, Paris, 1951.

[3] C. Chen. Bridging the gap: the use of pathfinder networks in visual navigation. *Journal of Visual Languages and Computing*, 9:267–286, 1998.

[4] D. Dearholt and R. Schvaneveldt. Properties of pathfinder networks. In R. Schvaneveldt, editor, *Pathfinder associative networks: Studies in knowledge organization*, pages 1–30. 1990.

[5] T. M. J. Fruchterman and E. M. Reingold. Graph drawing by force-directed placement. *Software - Practice and Experience*, 21(11):1129–1164, 1991.

[6] T. Kamada and S. Kawai. An algorithm for drawing general undirected graphs. *Information Processing Letters*, 31(1):7–15, 1989.

[7] D. M. Mount and S. Arya, 2006. ANN: A Library for Approximate Nearest Neighbor Searching, version 1.1.1, online software available on http://www.cs.umd.edu/~mount/ANN/, released the 4/8/2006.

[8] Q. V. Nguyen and M. L. Huang. A space-optimized tree visualization. In *Proc. of the IEEE Symposium on Information Visualization (InfoVis 2002)*, pages 85–92, 2002.

[9] P. Parlebas. Centralité et compacité d'un graphe. *Mathématiques et Sciences Humaines*, 39:5–26, 1972.

[10] A. Quirin, O. Cordón, V. P. Guerrero-Bote, B. Vargas-Quesada, and F. Moya-Anegón. A quick MST-based algorithm to obtain pathfinder networks. *Journal of the American Society for Information Science and Technology*, 59(12):1912–1924, 2008.

[11] H. Small and E. Garfield. The geography of science: disciplinary and national mappings. *Journal of Information Science*, 11:147–159, 1985.

[12] B. Vargas-Quesada and F. Moya-Anegón. *Visualizing the Structure of Science*. New York: Springer, 2007.