Vmap-layout, a Layout Algorithm for Drawing Scientograms

Arnaud Quirin and Oscar Cordón
European Centre for Soft Computing, Edf. Científico Tecnológico, Mieres, Spain
Phone: +34 985456545, FAX: +34 985456699
{arnaud.quirin,oscar.cordon}@softcomputing.es

Introduction

The purpose of this document is to describe a layout algorithm, i.e. an algorithm able to define the location of the nodes of a tree given by the user. The goal of the algorithm is to give to the tree an aesthetic layout. The tree is only described by its structure, i.e. by giving the set of nodes and links between them. The goal of the algorithm is to asign to each node a set of coordinates in the 2D-space, in order that the tree could be drawn. In the following, we will consider that the two terms *tree* and *graph* are synonyms, and in our graphs the links are directed, a node can only have one parent, there is no cycle, and no node have a specific meaning (there is no root node for instance).

The nodes have to be located in a *aesthetic* way in order to allow a user to understand properly the underlying structure of the tree. By using the term *aesthetic*, we would like to emphasize that the final layout of a tree is always subjective and defined by some criteria only guided by the final application of the layout algorithm. In our case, for the layout of scientograms (Chen, 1998), we would like to maximise the filling of the space, to avoid edge crossings, and to avoid the overlapping of nodes.

In fact, we will consider that the 2D coordinates of a node is an attribute of this node, and that the goal of this algorithm is to find the coordinate attributes of all the nodes. To do so, the algorithm will use a given amount of other attributes, computed for each node. So the algorithm has the following structure: first the algorithm takes into input the description of a tree and computes for each nodes some important attributes. Then, using these previously computed attributes it computes the coordinate attributes. Finally, the coordinates are given to the user in order he can generate a graphical output using an external library (this output could be an image, a web page, or whatever).

We will describe in the second section the global view of the algorithm. In the third section we will technicaly describe how to use the algorithm and what are the parameters used by this algorithm to generate the layout. Finally, in the last section, we will give an example of the output of our algorithm using the Spanish scientogram.

Overview of the algorithm

We start by a bit of terminology. In an ordered tree T, each node N has a parent P=PARENT(N), except the root node R=ROOT(T). CHILDREN(N) is the set of nodes having N as a parent. ASCENDANT(N) is the set { N, PARENT(N), PARENT(PARENT(N)), ..., R }. SUBTREE(N) is all the nodes which have N as one of its ascendant. SIZE(N) is equal to the number of nodes in SUBTREE(N). For instance, SIZE(N) is equal to 1 for a node having no child; 2 for a node having one child; etc. LEVEL(N) is the number of nodes in 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 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, but none of them has a child; 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 composed of three main functions. During the first function, the *attribute computation*, we compute an amount of attributes assigned to each node. These attributes will be used later to generate the coordinates of each node. As the graph provided by the user does not contain any root node (as said before), and as having a root node is required in order some of these attribtues can be computed, we have to elect one node as the root node. There is many ways of electing a centre in a graph. Many of them are described by Bavelas (1951) and Parlebas (1972). The one used here, that gives good results, is the *deliverer criterion*: we take the sum of the distances between a node and all the others, and we take as a root the one having the smaller value. Once we have elected the root node, a number of other attributes can be computed. We assign to each node N the values corresponding to SIZE(N), LEVEL(N) and DEPTH(N). At the end of this function, some attributes are assigned to all the nodes and a root node has been elected.

During the second function, the *coordinates computation*, having the graph structure and the attributes described before, 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. It is based on the space-optimized tree layout algorithm of Nguyen and Huang (2002). 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 centre 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), it will assign to the centre of this polygon the root node, it will cut the initial polygon into several slices (as many slices that the root node has children), and it will assign to each slice one of the children of the root

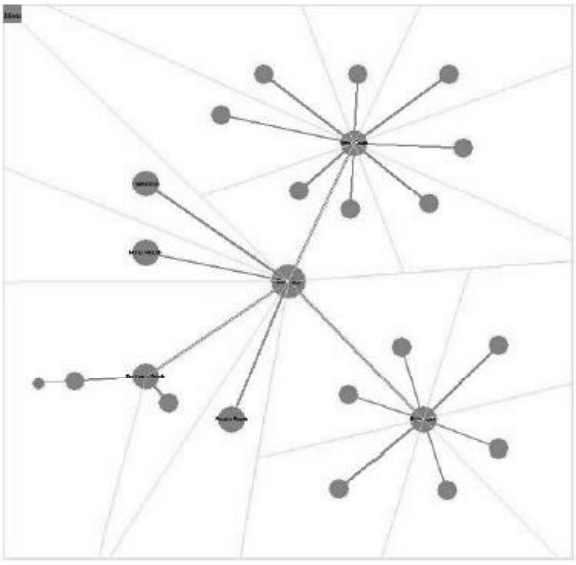

Figure 1. An example of the execution of the coordinates computation function.

node. In this function, the coordinates of the center of the polygons are assigned to each node. Then, the algorithm start again, considering one of the slice as a new polygon and the corresponding node N as the root of a new subtree S=SUBTREE(N). At the end of the second function, all the nodes of the tree are assigned to a pair of coordinates in the 2D-space. An example of the execution of this algorithm is shown in Fig. 1.

For the user, the nodes are represented as a little circle, with a text label inside. These graphical elements could overlap at the end of the second function. During the third function, the *node relocation*, we improve the placement of the overlapping elements (we will use the term *node* in the following to describe a graphical element which represents a node). Here, having the coordinates of the nodes generated previously, the algorithm fixes the final location of each node to avoid as much as possible that the node overlap. We say that two nodes overlap when they are too close from each other, according to a criterion defined by the user, and we call them *problematic* nodes. Problematic nodes avoid that the text labels could be read in a clear way, and avoid in general a high lisibility 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 springs. To avoid deadlocks in some cases (for instance, when a node is exactly located between two other nodes and at an equal distance), the problematic nodes are also slightly moved in a random direction, until they met a criterion set up by the user. At the end of the third function, the final coordinates of the nodes have been computed.

The algorithm, called the vmap-layout algorithm, takes as an input a tree T and assign to each node of this tree the coordinates of these nodes. It is defined as follows:

Algorithm VMAP-LAYOUT(T):

Function ATTRIBUTE-COMPUTATION(T).

1. Compute the root node using the *deliverer criterion*: compute the sum of the distances between a node and all the others, and take as a root the one having the smaller value.
2. Assign to each node N the values given by SIZE(N), LEVEL(N) and DEPTH(N).

Function COORDINATES-COMPUTATION(T).

3. Choose a 2D-space region P in which to draw T.
4. Choose a central point C in P and assign to this point the root of the tree R=ROOT(T).
5. If CHILDREN(R) is empty, stop.
6. Cut 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).
7. Run in a recursive way COORDINATES-COMPUTATION(R_i, P_i) for each child R_i of R.

Function NODE-RELOCATION(T).

8. Apply a KD-Tree technique[1] to compute the distance between all the nodes of T.
9. Select only the problematic nodes, i.e. the nodes close enough according to a criterion defined by the user.
10. For each node of this set, do:

- Apply a repulsive strength f and move the node along this force.
- Move it around its final position using a small random distance in the interval $[-\alpha,\alpha]$.

11. Execute again NODE-RELOCATION(T) until a given amount of iterations defined by the user has been reached.

As we can see, the algorithm tries to use all the space available for the drawing of the tree. Another very important point is that this algorithm is natively designed to avoid crossing of links, because a region of space will be allocated to only one tree.

Technical details for the use of the algorithm

This section describes some technical details about how to use the implementation of the vmap-layout algorithm (we call it the *program* in the following). In the first subsection, we list the parameters used by the program for each of the three main functions described previously. In the second subsection, we describe the format of the NET file loaded by the program. In the last subsection, we describe how we can use the output of the program (the list of the coordinates of all

[1] This technique computes the distance between a set of objects in order we can fastly know which objects are the closer to another one. The library we have used is called the ANN Library (Mount & Arya, 2006). The advantage of this library is that an approximate distance computation is used in order to speed up the process, provided the fact that the exact values of the distance are not important in our application.

the nodes) and how we can convert it into another format (an image for instance).

Parameters of the program

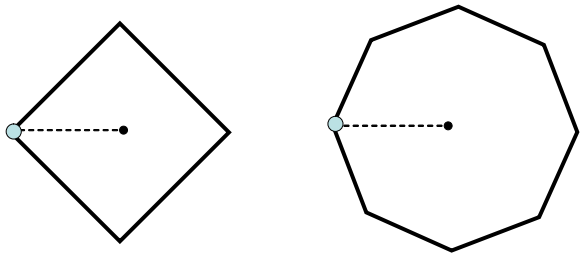

Figure 2. The initial polygons with a value of 4 or 8 for the 'INITIAL_SHAPE_SIDES' parameter.

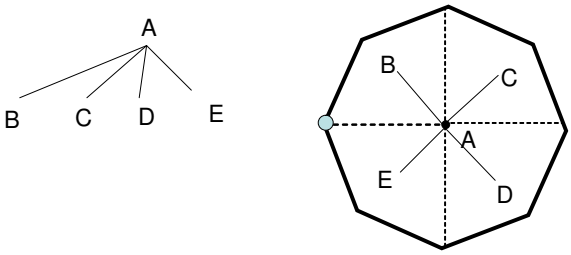

Figure 3. One the left, the tree to draw. On the right, the initial polygon and the centre *C* in black. The polygon is first cut in four slides, because the node *A* has four children, then we assign the corresponding nodes to the corresponding sub-polygons.

During the initialization of the Vmap-Layout algorithm (step 3 in the algorithm VMAP-LAYOUT), we have to select the initial polygon, in 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. Several options can be used, but in our case, it seems that using a circle is better. The definition of this shape is controled by the 'INITIAL_SHAPE_SIDES' parameter. It indicates how many sides this shape will contain. Larger this value is, more circular the shape will be, but slower will run the algorithm. A value of 15 is well suited, but other could be tried. See Fig. 2 for more details. Apart from that, the number of sides has nothing to do with the further execution of the algorithm, this option has only an aesthetic use. For instance, a high number could be used to simulate a perfect circle. See Fig. 3 for the detail of the execution of the algorithm.

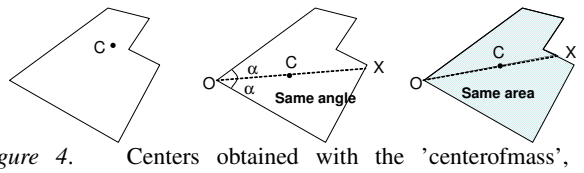

Figure 4. Centers obtained with the 'centerofmass', the 'centralpoint_angle' and the 'centralpoint_area' methods for the 'TYPE_CENTRALPOINT' parameter.

To choose the central point *C* of a polygon *P* (step 4 in the algorithm VMAP-LAYOUT), we have explored at least three different methods. The first one, called '**centerofmass**', takes as *C* the centre of gravity of *P*. This centre is defined in any case, but suffers from two problems: this centre could be outside of the polygon and a polygon with a lot of segments could attract the centre far away from the *natural* centre of the polygon. The second one, called '**centralpoint_angle**', uses the angle to compute the central point. For a given polygon *P*, we first select a point on the border of the polygon, that we call origin O[2]. We then draw a line cutting the angle *O* in two equal parts. Then we takes as the centre *C* the middle of this line. The last one, called '**centralpoint_area**', applies the same procedure, but by cutting the polygon into two parts having the same area. See Fig. 4 for more details. The selection of these methods is done by the 'TYPE_CENTRALPOINT' parameter. A value of 1 selects the 'centerofmass' method, a value of 2 selects the 'centralpoint_angle' method and a value of 3 selects the 'centralpoint_area' method. The original paper of Nguyen and Huang (2002) and our tests consider the 'centralpoint_area' as the best one. The problem is that for specific shapes of polygon (especially if they have internal angles greater than π), the area of a sub-polygon is not well defined. This is why other methods are provided. The default value for this parameter is fixed to 3 (centralpoint_area).

In the previous paragraph, we said that we place the centre *C* in the middle of the line, but other distance could be explored. The 'CUTPOINT' parameter select the ratio between the distance OC and the distance OX (see Fig. 4). With a small value for this parameter, we get maps where the centers are close from each other, and with a large value, we get maps where the centers are more far away from each other. The default value for this parameter is fixed to 0.5.

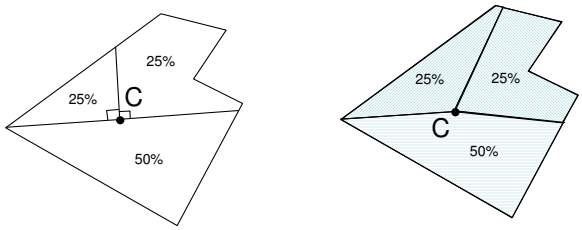

Figure 5. The result with the 'crosspolygon_angle' value (on the left part) and with the 'crosspolygon_area' value (on the right part) for the 'TYPE_CUTSLIDE' parameter.

The way of cutting a polygon in different slices (step 6 in the algorithm VMAP-LAYOUT) could be achieved by at least two methods and can be selected with the 'TYPE_CUTSLIDE' parameter (see Fig. 5. If we use the 'crosspolygon_angle' method (value equal to 1), the algorithm defines the size of the slides in order that the angle around the center *C* are proportional to the size of each tree R_i. If we use the 'crosspolygon_area' method (value equal to 2), the algorithm defines the size of the slides in order that the area around the center *C* are proportional to the size of

[2] This point could be for instance the centre of the polygon parent (the one used to generate the current polygon), or the left most point of the polygon.

each tree R_i. In many maps extracted from the real world, the second option (with the value equal to 2) does not work properly because the non-convex shapes of the polygons convert sometimes the problem of finding a polygon's percentile using the area into a nearly impossible problem. Therefore, the default value for this parameter is fixed to 1 (crosspolygon_angle).

Figure 6. On the left, the normal behaviour in which all the space is used to compute the size of each slice. On the right, a modified behaviour in which a constraint is applied before computing the size of each slice, allowing us to direct the graph in a given way.

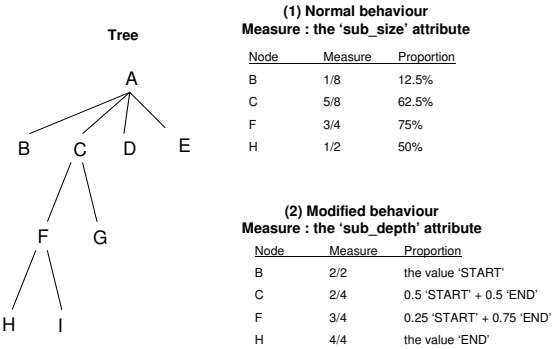

Figure 7. An example of the behaviour obtained with the 'NGUYEN_ANGLE_IDEA' parameter. On the left, an example of a tree. On the top, the proportions obtained using the value 1 for this parameter (given by the 'sub_size' measure). On the bottom, the proportions obtained using the value 2 for this parameter (given by the 'sub_depth' measure).

The way of cutting the slices described previously needs a measure defined for each node in order to compute the proportion in percentage allocated to the corresponding sub-polygon. The normal behaviour is to take the size of each node (defined previously as the 'sub_size' attribute), i.e. the number of nodes in the subtree of N, to compute a proportional ratio at the level of their parents. Then this ratio is used as a percentage to compute the size of all the slices of the corresponding sub-polygons. Note that this measure is totally independant from the *way* of cutting these slices, i.e. given the previously mentioned ratio, the size of a slice can be obtained using their angle or their area and is selected using the 'TYPE_CUTSLIDE' parameter. In fact, there is an other measure which can be used to compute the ratio. Using the depth of the trees (defined previously as the 'sub_depth' attribute), we can modify the proportion allocated to each slice depending on if they are close or far away from the center of the map. This could allow us to constrain the angle of the links to go only forward when we are close to the center of the map (see Fig. 6). The normal behaviour can be obtained using the value 1 for the 'NGUYEN_ANGLE_IDEA' parameter, while the modified behaviour is obtained using the value 2. The default value for this parameter is fixed to 2. In the modified behaviour, we can select the proportion applied to the node located close to the center of the map (with a small value for the 'sub_depth' attribute) by fixing the value of the parameter 'NG_ANGLE_CONSTRAINT_START'. By default its value is fixed to 0.5. We can also select the proportion applied to the node located far away from the center of the map (with a high value for the 'sub_depth' attribute) by fixing the value of the parameter 'NG_ANGLE_CONSTRAINT_END'. By default its value is fixed to 0.25. For any other value for the 'sub_depth' attribute, a linear regression is used to compute the correct value of the proportion of the slice. In the Fig. 7, we show for a given tree and four nodes, the values of the 'sub_size' and the 'sub_depth' measure. The first number (before the '/') indicates the current value for the node, the second number indicated the maximum value for the node, used to compute the proportion of the corresponding slice.

In the NODE-RELOCATION(T) function, some parameters can be fixed in order to have a trade off between the run time and the quality of the node coordinates computation. For instance, the NODE-RELOCATION(T) function could be desactivated using the 'KDTREE_RANDOM_IDEA' parameter. If the value of this parameter is fixed to 1, the algorithm jump over this function and directly let the coordinates of the nodes as the ones defined by the COORDINATES-COMPUTATION(T) function. If the value of this parameter is fixed to 2, the algorithm applies the node relocation function. The default value of this parameter is fixed to 2.

The KD-Tree technique (in the step 8 of the VMAP-LAYOUT algorithm) uses a 'KDTREE_EPSILON' parameter to define the appoximation during the computation of the distance used to select the problematic nodes. A high value implies a biggest error during the computation of the distances but also a fastest algorithm. A value of zero (0) implies an exact computation. As the computation of the distances is not useful for the final layout of this algorithm, but is only used to define the set of the problematic nodes, high values could be used to accelerate the algorithm. The default value for this parameter is fixed to 0 (exact computation).

The 'KDTREE_RADIUS' parameter is used to define the set of problematic nodes (step 9 in the algorithm VMAP-LAYOUT). The coordinates of any node having a distance smaller or equal to any other node will be modified by the algorithm, while the nodes located at a greater distance will not be moved. As the real distance values are used, this parameter is dependant to a lot of other parameters related to the scale of the final layout[3], so many care should be taken while the 'KDTREE_RADIUS' parameter is modified. The default value of this parameter is fixed to 0.30.

To each node N of the set defined by the 'KDTREE_RADIUS' parameter, a force is applied defined as the sum of all the repulsive force generated by the nodes close to N, multiplied by the value defined

[3] These other parameters are the 'INITIAL_POSITION', the 'INITIAL_DISTANCE', and the 'CANEVAS_SIZE' (see the next section).

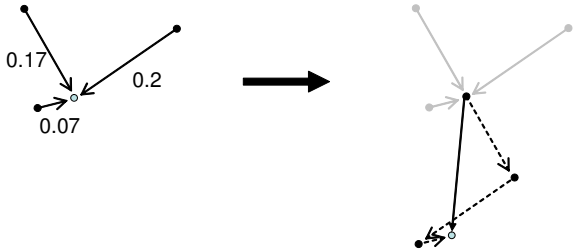

Figure 8. An example of the modification of the coordinates of a node after applying the node relocation function.

by the 'KDTREE_SPRINGSTRENGTH' parameter (the repulsive strength f), and added to a value defined by the 'KDTREE_RANDOMSTEP' parameter (defining the random interval $[-\alpha,\alpha]$, step 10 of the VMAP-LAYOUT algorithm). See Fig. 8 for more details. The 'KDTREE_SPRINGSTRENGTH' is used to define the amount of the quantity of movement applied to the node. A small value moves the nodes slowly through the iterations of the algorithm. The default value for this parameter is fixed to 0.10. The 'KDTREE_RANDOMSTEP' defines the quantity of randomness applied to the location of the node at the end of each iteration. A value of zero (0) disables any randomness during the movement of the nodes. The default value for this parameter is fixed to 0.05. As before, these parameters are closely dependant to the scale of the final layout and should be manipulated with care.

Another parameter is the 'KDTREE_ITERATION' parameter which defines how many iterations the algorithm have to do (step 11 of the VMAP-LAYOUT algorithm). A high value for this parameter can be used to establish the convergence of the coordinates of the nodes, but slow down the process. The default value for this parameter is fixed to 100.

The input file

To be complete, we give here the structure of the file loaded by the algorithm, called the .NET file. The graph given by the user is a set of nodes, each of them having a text label, and connected by links, each of them labeled by a floating value (its weights). The input file is a simple text file (ASCII file) and should follows the following structure:

```
*vertices <number of nodes>
<node index 1> "<label 1>"
<node index 2> "<label 2>"
...
<node index N> "<label N>"
*matrix
<W(1,1)> ... <W(1,n)>
<W(2,1)> ... <W(2,n)>
...
<W(n,1)> ... <W(n,n)>
```

where `W(i,j)` is the weight of the link (i, j). `n` is equal to the number of vertices and to the last node index and the node indices are 1-based.

Another structure is possible using `*edges` instead of `*matrix`:

```
*vertices <number of nodes>
<node index 1> "<label 1>"
<node index 2> "<label 2>"
...
<node index N> "<label N>"
*edges
<i> <j> <W(i,j)>
...
```

In this variant, `W(i,j)` is the weight of the link (i, j) described by the index of the nodes `<i>` and `<j>`. The second variant is more useful for sparse matrices, and the first one for dense matrices.

The output file

The output of the program is another text file describing the coordinates of the nodes in the DOT language (Gansner & North, 2000), an ASCII format used by the GraphViz library[4]. The final graphical output is obtained after a conversion from the DOT language to one of the many formats proposed by the GraphViz library (FIG, GIF, JPEG, PNG, PS, SVG, VRML, ...).

The program scales the drawing in order that it fits into a rectangular page and fixes the sizes of the graphical elements (the nodes, the font of the text labels and the edges). Some specific parameters related to the graphical output can be fixed by the user.

The size of the nodes and the size of the text font could be defined by a constant value or any combination of the node attributes seen previously. In the current version of the algorithm, the 'level' attribute of each node is used to define these values. This is hard-coded in the source code and could not be changed by the configuration file ('vmap-layout.ini') but the program should be recompiled. The size of the nodes is defined by default to $1/(0.9 * level + 1)$ units and the size of the font for the node's text labels is defined by default to $280/(12 * level + 20)$ units.

The 'INITIAL_POSITION' parameter defines the value of the X and the Y coordinates of the root node into the layout. The default value for this parameter is fixed to 24 units.

The 'CANEVAS_SIZE' parameter defines the scale of the layout. All the coordinates are fixed to have a smaller value than the value of this parameter. The default value for this parameter is fixed to 15 units.

For some output format (for instance, the PS format), this parameter has no effect because the GraphViz library applies its own scale algorithm.

The 'EDGE_SIZE' parameter defines the width of the edges. The default value for this parameter is fixed to 0.5.

The final output of the algorithm is a DOT file and has to be converted into an other format, to be printable. If the

[4] GraphViz is an open source network drawing software, freely provided by AT&T Labs, and is available at: http://www.graphviz.org/

output of the algorithm is saved into the 'mapfile.dot' file, the following command could be used to obtain a PDF file, once the GraphViz library has been installed:

```
/usr/local/bin/neato -Tps2 -o mapfile.ps
mapfile.dot
ps2pdf14 mapfile.ps mapfile.pdf
```

Examples

The last pages shown the algorithm applied on the Spanish scientogram. The first graph is the one obtained by the classical Kamada-Kawai algorithm. The second graph is the one obtained by the Vmap-layout algorithm, if all the special feature are disabled (NGUYEN_ANGLE_IDEA = 1, TYPE_CENTRALPOINT = 2, KDTREE_RANDOM_IDEA = 1). The third graph is the one obtained using the modified behaviour for the proportion of the slides (**NGUYEN_ANGLE_IDEA = 2**, TYPE_CENTRALPOINT = 2, KDTREE_RANDOM_IDEA = 1). Many nodes close to the center are more spaced, but many of them still overlap. The fourth graph is the one obtained using the node relocation function (NGUYEN_ANGLE_IDEA = 2, TYPE_CENTRALPOINT = 2, **KDTREE_RANDOM_IDEA = 2**). Now, all the nodes are spaced, especially the ones located in the top of the map. The last graph is the one obtained using the area instead the angle to compute the centre C (NGUYEN_ANGLE_IDEA = 2, **TYPE_CENTRALPOINT = 3**, KDTREE_RANDOM_IDEA = 2). Some nodes are better located and the space is better filled.

References

Bavelas, A. (1951). *Réseaux de communications au sein de groupes placés dans des conditions expérimentales de travail, les sciences de la politique aux États- unis*. Paris: Armand Colin.

Chen, C. (1998). Bridging the gap: the use of pathfinder networks in visual navigation. *Journal of Visual Languages and Computing*, *9*, 267-286.

Gansner, E. R., & North, S. C. (2000). An open graph visualization system and its applications to software engineering. *Software — Practice and Experience*, *30*(11), 1203–1233.

Mount, D. M., & Arya, S. (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)

Nguyen, Q. V., & Huang, M. L. (2002). A space-optimized tree visualization. In *Proc. of the ieee symposium on information visualization (infovis 2002)* (p. 85-92).

Parlebas, P. (1972). Centralité et compacité d'un graphe. *Mathématiques et Sciences Humaines*, *39*, 5-26.