Research Report #1 - Algorithms to generate PFNet maps

A. Quirin, O. Cordon - March 30, 2007

Abstract

Visual Science Maps are used to analyze and measure the information contained in the scientific papers. They are graphs and they represent the main connections between several entities, that could be scientific domains, journals or authors. The main connections between two entities are represented by an edge in the final graph. This can be used to print graphically the scientific contribution of a country, or the whole world. Several algorithms exist in the literature to start from an adjacence matrix (describing by weights the similarities between two entities), and to obtain a final pruned graph, in which only the main connections are remaining. One of this algorithm is called PFNet. In this paper, we first analyze the behavior of PFNet, by giving an intuitive interpretation, and then we propose several algorithms giving the same result as PFNet, but in a shorter time. Some maps are printed and the results are discussed.

1 Introduction

The representation named *Visual Science Maps* is a technique applied to scientific literature able to analyze and measure the information contained in the scientific papers. Bibliometrics is the study, or measurement, of texts and information. Content analysis is a type of bibliometrics. While it is most often used in the field of library and information science, it has wide applications in other areas. Historically bibliometric methods have been used to trace relationships amongst academic journal citations. Citation analysis, which involves examining an item's referring documents, is used in searching for materials and analyzing their merit. Citation indexes, such as Institute for Scientific Information's Web of Science, allow users to search forward in time from a known article to more recent publications which cite the known item [4].

Graphs represent a new representation of the statistics collected during citation analysis. Graphs can represent many aspects of statistical data, using different shapes, sizes and labels for the nodes and/or the links. Graphs can be used to represent the scientific contribution of a single team, or a country, or even the contribution of the whole world. In our case, nodes can represent authors, journals or scientific domains and a link between two nodes is valued by a weight. This weight is given by a similarity measure, corresponding to the co-citation degree between the two entities.

These graphs are generated using a modified PFNet algorithm, starting from the values of all the links between the nodes of a given graph. The next section provides a description of the original PFNet algorithm and the section 3 describes the modified algorithm. In the section 4, two adaptations of Shortest Path algorithms are compared and discussed, with some examples. In the section 5, three adaptations of Minimum Spanning Tree algorithms are discussed and compared and some examples are shown. Then, a conclusion is proposed.

2 The PFNet algorithm

In our research, these graphs are produced using the PFNet algorithm [3]. Its goal is the extraction of the main structure of a network by the mean of the analysis of the proximity among their variables. The result is a graph, not a tree. PFNet algorithm can only be applied on non-negative weights. Only the best paths connecting two nodes are kept, and these best paths are found according to a metric governed by two parameters. The similarity between two nodes is represented as a Minkowski distance with a

parameter r, which may be fixed to ∞, thus being equivalent to the maximum of the intermediate link weights found in the path connecting these two nodes. The second parameter, q, restricts the number of links among the elements. Every path connecting two nodes that violate the triangular inequality, having an associated Minkowski distance lesser than any other path between the same two nodes composed of up to q links, will be removed. The maximum possible value for q is $n-1$, with n being the number of nodes in the network.

To generate a PFNet network, two matrices are used.

- $W_{jk}^{(i)}$ indicates the minimum cost to go from the node j to the node k by following exactly i links, and this matrix is computed recursively using the matrix $W_{jk}^{(i-1)}$. $W^{(1)}$ is the original matrix of weights.
- $D_{jk}^{(i)}$ indicates the minimum cost to go from the node j to the node k by following i links *or less*, and this matrix is computed recursively using the matrices $W_{jk}^{(1)} \dots W_{jk}^{(i)}$.

The original PFNet algorithm is:

1. Compute $W^{(i+1)} = W \odot W^i$, as follows: $w_{jk}^{(i+1)} = \text{MIN}((w_{jm})^r + (w_{mk})^r)^{1/r}$, for $1 \leq m \leq n$.
2. Compute $D^{(i)}$, as follows: $d_{jk}^i = \text{MIN}(w_{jk}^1, \dots, w_{jk}^i)$, for $j \neq k$.
3. Continue until W^q and D^q are computed.
4. Compare W^1 and D^q : all the edges having the same values in these two matrices belong to the final PFNet graph.

For an intuitive interpretation of the original PFNet algorithm, we can consider that we look for a graph that keep only the smallest collars in the graph. A collar is the link with the maximum value on a path used to join a node from another one. Then, the minimum collar is found among all the possible paths, and correspond to the link with the minimum value. A simplified algorithm could be:

1. Sort the edges by their ascending values, and traverse them in this order.
2. Only keep the edges N_1-N_2 representing a minimum collar (all the other paths joining N_2 from N_1 should have a bigger value).
3. The suppressed edges should not be considered anymore, but this is not mandatory for the right terminaison of the algorithm.

The algorithm stops when the direct path between two nodes (W^1) is equal to the smallest collar (D^i) for at most q edges (when $i = q$). This is the stop criterion of the original PFNet algorithm.

3 The modified PFNet algorithm

In our case, we use the value $n-1$ for the parameter q, to be sure that the optimal path is found among *all* the nodes, and the value ∞ for the parameter r, to consider the path having the maximum value. In fact, because our research is applied to Visual Science Maps, we prefer to use a modified version of the original PFNet algorithm. In the modified version, the original expression $w_{jk}^{(i+1)} = \text{MIN}((w_{jm})^r + (w_{mk})^r)^{1/r}$ is replaced by $w_{jk}^{(i+1)} = \text{MAX}\{\text{MIN}(w_{jm}, w_{mk}^i)\}$. Looking for the best minimum path, instead of the best average (with $r = 2$) or any other path, gives special interesting properties to the final representation. For instance, this is equivalent to a flow problem when for each path, we look

for the minimum link (viewed as a *bottleneck*), and then we look for the larger *bottleneck* (this is the traduction of the MAX(MIN(...)) expression).

The modified PFNet algorithm is:

1. Compute $W^{(i+1)} = W \odot W^i$, as follows: $w_{jk}^{(i+1)} = \text{MAX}\{\text{MIN}(w_{jm}, w_{mk}^i)\}$, for $1 \leq m \leq n$.
2. Compute $D^{(i)}$, as follows: $d_{jk}^i = \text{MAX}(w_{jk}^1, \ldots, w_{jk}^i)$, for $j \neq k$.
3. Continue until W^{n-1} and D^{n-1} are computed.
4. Compare W^1 and D^{n-1} : all the edges having the same values in these two matrices belong to the final PFNet graph.

In a Visual Science Map, keeping the largest *bottleneck* link L between two nodes can be viewed intuitively as a way to only let connected the domains (or journals, or authors, ...) in which no other path can pass through nodes having a similarity greater than the value of L. So, the fact that two domains (or authors, ...) are connected means that these two domains have a strong relationship, better than any other relationship observed among any other path connecting these two domains.

Actually, this algorithm was implemented in C, and has a complexity of $O(qn^3)$. The input of this algorithm is a matrix of similarities, or a list of edges with their corresponding weights. The output is the list of edges in the PFNet graph, with their corresponding weights. Then we use a graph drawing algorithm based on energies [2] to convert the output of the algorithm in a graphical representation. The Kamada-Kawai algorithm insures that the resulting graph will not have many link crossings and that nodes having strong links (large weights) are closer.

Fig. 2 shows the result of the modified PFNet algorithm for the Argentina Science map (Argentina.net), using the Kamada-Kawai algorithm to redistribute the nodes in a better way. Fig. 3 gives some examples of the computing times observed on a 2.80 GHz CPU, with 2 GB of memory, running the Linux OS. As it can be seen, the delay are quite long, even on a recent processor.

4 Shortest Path algorithms

According to the long delay shown in Fig. 3, and because real time applications need fast algorithms, we have tried to design faster algorithms, producing exactly the same results than the modified PFNet algorithm. To design these new algorithms, the procedure was always the same: (1) modify some well-known graph processing algorithms in a way that the behavior of the parameters q, r and the *bottleneck* behavior remain the same, (2) analyze these new algorithms to know if the produced graphs are the same or not, and (3) if not, find properties characterizing the missing or the edges in excess to improve the algorithms. Only algorithms having an original complexity of $O(n^3)$ (or less) have been selected.

4.1 The Floyd-Warshall algorithm

The Floyd-Warshall algorithm is an algorithm giving the shortest paths between any source node and any destination node. The result includes two matrices: a cost matrix M_{ij} where m_{ij} is the minimum cost to join the node j from the node i, and a predecessor matrix P_{ij} where p_{ij} is the index of the predecessor of the node j among the optimal path to reach j from the node i. The Floyd-Warshall algorithm has been selected to substitute the costly computation of the matrices W and D in the original PFNet algorithm. As these matrices are also *cost* matrices, the substitution could be more effective, because the complexity of the Floyd-Warshall algorithm is $O(n^3)$.

The original Floyd-Warshall algorithm:

1. $D^{(0)} = W$

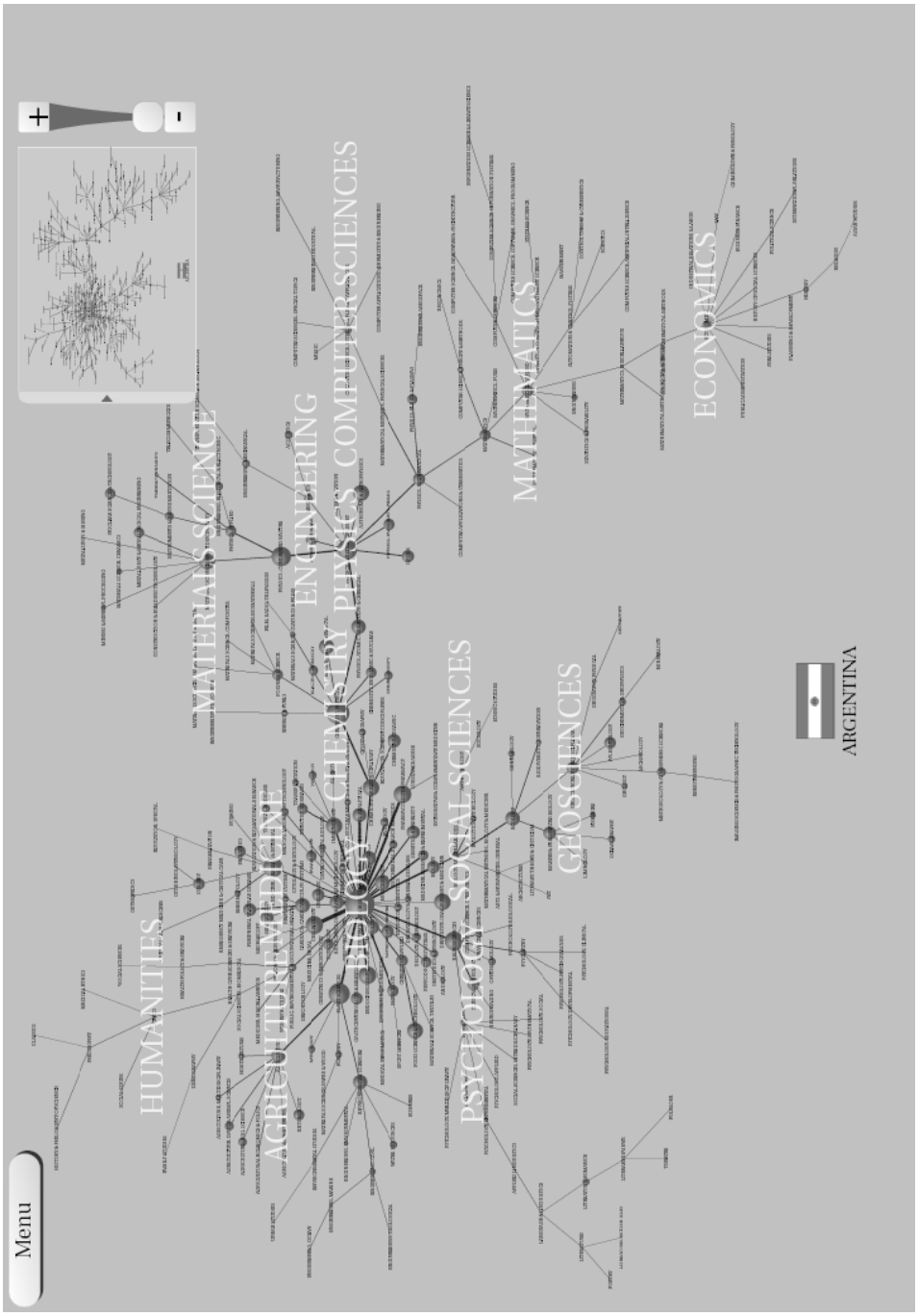

Figure 1: A typical Visual Science Map obtain with the modified PFNet algorithm, for the scientific contribution of Argentina (www.atlasofscience.net).

2. FOR k from 1 to $q+1$, do

3. FOR i from 1 to n, do

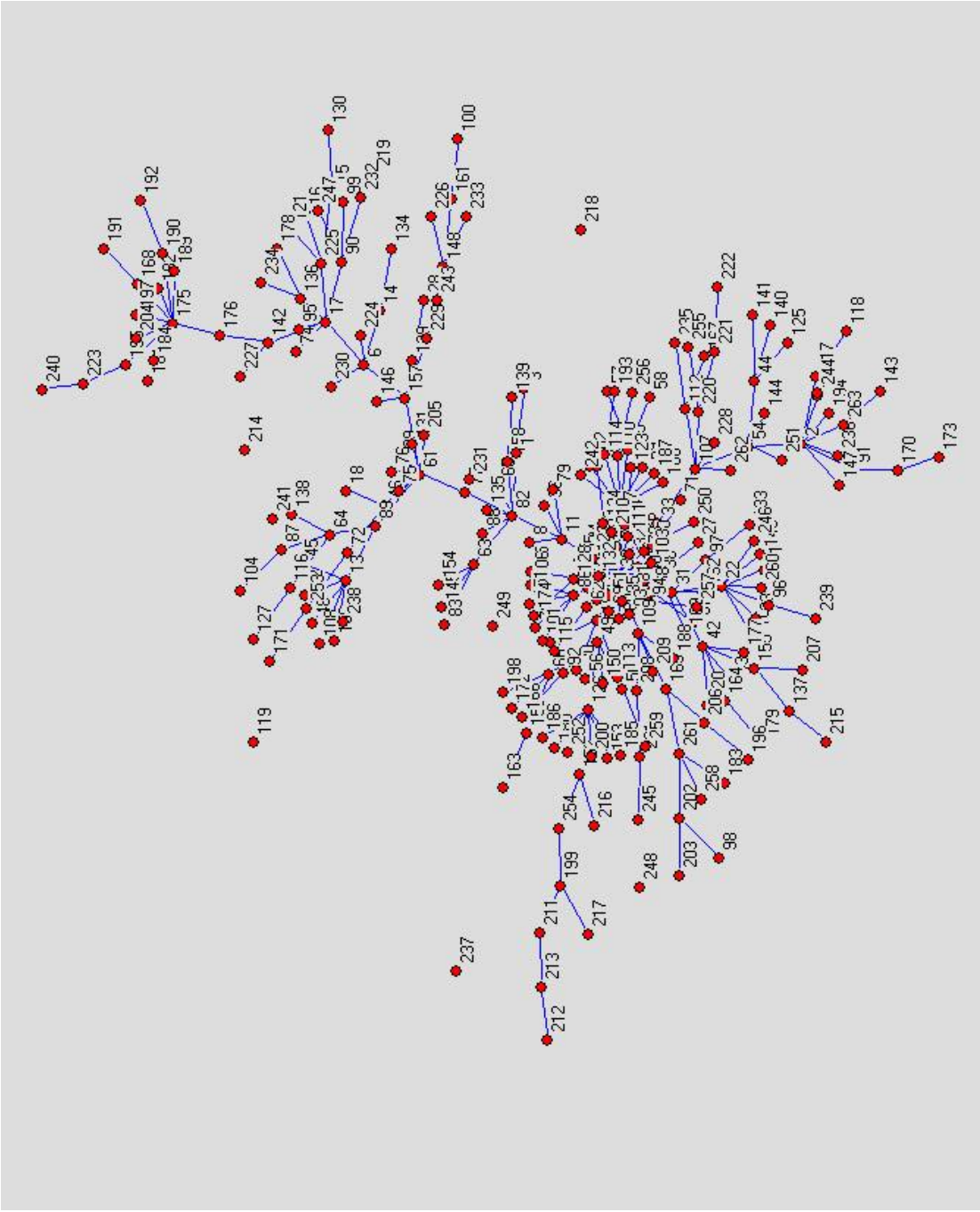

Figure 2: The result of the modified PFNet algorithm on the Argentina Science map.

4. FOR j from 1 to n, do

5. $d_{ij}^{(k)} = \text{MIN}(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)})$

Map	Time (s)
Argentina.net	146
Chile.net	107
Cuba.net	60.4
Espana.net	114
Mexico.net	133
Portugal.net	145

Figure 3: Computing time of the modified PFNet algorithm.

4.2 The Fast-PFNet algorithm

In fact, we need to modify this algorithm to obtain the same behaviors that the modified PFNet algorithm (r, q and MAX-MIN). This algorithm, called *Fast-PFNet* is:

1. $D^{(0)} = W$
2. FOR k from 1 to $q+1$, do
3. FOR i from 1 to n, do
4. FOR j from 1 to n, do
5. $d_{ij}^{(k)} = \text{MAX}(d_{ij}^{(k-1)}, \text{MIN}(d_{ik}^{(k-1)}, d_{kj}^{(k-1)}))$
6. IF $d_{ij}^{(k-1)} > \text{MIN}(d_{ik}^{(k-1)}, d_{kj}^{(k-1)})$
7. THEN $p_{ij}^{(k)} = p_{ij}^{(k-1)}$
8. ELSE $p_{ij}^{(k)} = p_{kj}^{(k-1)}$

By the substitution of the computation of the shortest path ($d_{ik}^{(k-1)} + d_{kj}^{(k-1)}$) by the computation of the *bottleneck* ($\text{MIN}(d_{ik}^{(k-1)}, d_{kj}^{(k-1)})$), we obtain the same matrix D that the one obtained by the modified PFNet algorithm.

For replacing the last part of the modified PFNet algorithm (the edge selection), two different ways have been proposed.

In the first way, we keep the original edge selection algorithm. The matrix D^q is given by the Fast-PFNet algorithm and the matrix W^1 is given as a parameter of this algorithm, so the comparison is straightforward. This edge selection has a complexity of $O(n^2)$. The final algorithm (called Fast-PFNet) has a complexity of $O(n^3) + O(n^2) = O(n^3)$. In this case, the computation of the predecessor matrix is not needed and the final algorithm is even faster. This way is the fastest way. The algorithm is:

1. FOR each node $i \in G$
2. FOR each node $j \in G$
3. IF $d_{ij}^{(0)} = d_{ij}^{(q+1)}$ AND $d_{ij}^{(0)} \neq 0$, THEN
4. (i,j) belongs to the final graph

In the second way, for $1 \leq k \leq n$, we keep only the edges connected to the node k, while analyzing the graph corresponding to the shortest paths from the single node k ($p_{kj}^{(q)}$ for $1 \leq j \leq n$). This corresponds

to the inclusion, in the final graph, of the edges connecting the node $p_{kj}^{(q)}$ to the node j with a positive weight, for $1 \leq k \leq n$ and $1 \leq j \leq n$. The mathematical equivalence with the first way comes from the fact that an edge connecting the node $s = p_{kj}^{(q)}$ to the node $e = j$ is included in the final P matrix only if $d_{se}^{(q)} = d_{se}^{(0)}$. This second way has also a complexity of $O(n^2)$, so the final algorithm has the same complexity than before, but the predecessor matrix has to be computed, so the final time is a bit slower (see figure 4). The algorithm is:

1. FOR each node $i \in G$
2. FOR each node $j \in G$
3. $s = p_{kj}^{(q)}$
4. $e = j$
5. IF $s \neq 0$ AND $s \neq i$, THEN
6. (s,e) belongs to the final graph

Map	Time for Mod-PFNet	Time for FastPFN (way 1)	Time for FastPFN (way 2)
Argentina.net	146	0.46	0.52
Chile.net	107	0.37	0.41
Cuba.net	60.4	0.27	0.30
Espana.net	114	0.37	0.41
Mexico.net	133	0.42	0.46
Portugal.net	145	0.44	0.50

Figure 4: Computing time (s) of the Fast-PFNet algorithms.

Note that the inclusion of all the links in the final predecessor matrix (instead of doing the edge selection presented previously) is *not* a PFNet graph in the general case, even with directed or undirected graphs. The proof is the following: if two different paths have the same minimum link in common (so they have the same value), but the first path is shorter than the second path (in terms of number of links), the Floyd-Warshall algorithm will always choose the shortest path, and the modified PFNet algorithm could suppress a link in this shortest path if this link does not respect the triangular condition.

4.3 The Dijkstra algorithm

The Dijkstra algorithm has been employed not to mimic the graphs produced by the modified PFNet algorithm, but to be able to extract trees in which the root node has a special property (for instance, the node with the maximum number of links, ...).

In this case, several trees have been produced, depending of the property of the root node:

- Starting or ending the arc having the maximum value
- Starting or ending the arc having the minimum value (and not null)
- Having the minimum or maximum number of entering or outgoing links
- Having the minimum or maximum sum of entering or outgoing links
- Having the minimum or maximum average of entering or outgoing links

This leads to 16 different properties, so 16 different trees. We can notice the fact that, on Visual Science Maps, nodes having two distinct properties could in general be the same, and this is also true for different maps. For instance, the node starting the edge having the maximum value, having the maximum number of links, having the maximum sum of links and having the maximum average of links, is the node number 30 for the Argentina, Chile and Portugal maps. To give some examples, the link with the greatest weight on the Argentina map is the link 30-38 (weight 8746), the node 30 has 235 outgoing edges, and the average of these edges is arround 755.

Let G be the original graph, $S[G]$ the nodes of G, $d[v]$ the cost attribute of the node v, $p[v]$ the predecessor of the node v, $w(u,v)$ the weight of the arc $u-v$ and s the chosen source node. EXTRACT-MIN(F) is a function that extract (and remove) the node with the smallest attribute in the set F. The original Dijkstra algorithm is:

1. FOR each node $v \in S[G]$
2. $\quad d[v] = \infty$
3. $\quad p[v] = \emptyset$
4. $d[s] = 0$
5. $E = \emptyset$
6. $F = S[G]$
7. WHILE $F \neq \emptyset$
8. $\quad u =$ EXTRACT-MIN(F)
9. $\quad E = E \cup \{u\}$
10. $\quad$ FOR each node $v \in$ NEIGHBORS(u)
11. $\quad\quad$ IF $d[v] > d[u] + w(u,v)$, THEN
12. $\quad\quad\quad d[v] = d[u] + w(u,v)$
13. $\quad\quad\quad p[v] = u$

4.4 The modified Dijkstra algorithm

A new modified Dijkstra algorithm has been designed to take into account the fact the maximum path having the minimum link has to be found, and not the shortest path (the sum should be replaced by MAX(MIN(...))). This modification has no influence of the computation time. The complexity of this algorithm is $O(n^2)$, but it can be proven that a complexity of $O(n.log(n) + A)$, where A is the number of edges, is possible using binary heaps for the EXTRACT-MAX function [1].

The new modified Dijkstra algorithm is:

1. FOR each node $v \in S[G]$
2. $\quad d[v] = 0$
3. $\quad p[v] = \emptyset$
4. $d[s] = \infty$
5. $E = \emptyset$

6. $F = S[G]$
7. WHILE $F \neq \emptyset$
8. $\quad u = \text{EXTRACT-MAX}(F)$
9. $\quad E = E \cup \{u\}$
10. $\quad$ FOR each node $v \in \text{NEIGHBORS}(u)$
11. $\quad\quad$ IF $d[v] < MIN(d[u], w(u,v))$, THEN
12. $\quad\quad\quad d[v] = MIN(d[u], w(u,v))$
13. $\quad\quad\quad p[v] = u$

Figures 5, 6 and 7 show some graphs obtained with the modified Dijkstra algorithm. The graphs produced by applying Dijkstra algorithm on the node 30 (Fig. 5) and the node 38 (Fig. 6) correspond graphically to what is expected with Visual Science Maps: only one cluster and nodes correctly distributed around the central node. The graph using the node 235 (Fig. 7) is not a well shaped graph. In fact, we have observed that graphs generated by the Dijkstra algorithm from *major* nodes (in which the statistical criterion is related to a maximum value) are better than from *minor* nodes.

4.5 Comparison between modified PFNet and Shortest Path algorithms

The procedure used in the Fast-PFNet algorithm is equivalent to run the modified Dijkstra algorithm for each node in G considering it as the source node s, and keep only the first level of the generated trees (only the edges connected to s for each generated tree). So we can say that a modified-PFNet graph generated from a graph G is equivalent to the set of all the top-level edges of the trees generated by the modified Dijkstra algorithm for each node of the graph G (see Fig. 10).

Proof: at the end of the modified PFNet algorithm, are kept all the edges $u - v$ where the cost to join directly the node v from the node u is equivalent to the best cost among all the other paths. Any edge of a Dijkstra tree corresponds to an edge included in the best path from a given source node s. But in the case in which the edge with the minimum value m among all the paths is connected to the node s, all these paths have the best value corresponding to m. In this case, only the edges connected to the node s corresponds to the best direct edge, where the cost corresponds to the best cost among all the other paths.

5 Minimum Spanning Tree algorithms

5.1 The MST algorithms

A Minimum Spanning Tree algorithm extract from a given graph the spanning tree (a tree connecting all the nodes in a graph) having the minimum cost (the sum of the weights of the edges)). Algorithms achieving this goal include Prim and Kruskal algorithms. In our study, we have looked for an algorithm giving the same result than the modified PFNet algorithm.

5.2 The Kruskal algorithm

In the Kruskal algorithm, all the edges are sorted (smallest values first). Edges are then added one by one in the final tree only if they do not connect the same connected component (in such a way that cycles are avoided). In the best case, the complexity of this algorithm is $O(A.log(n))$ [1].

The original Kruskal algorithm is:

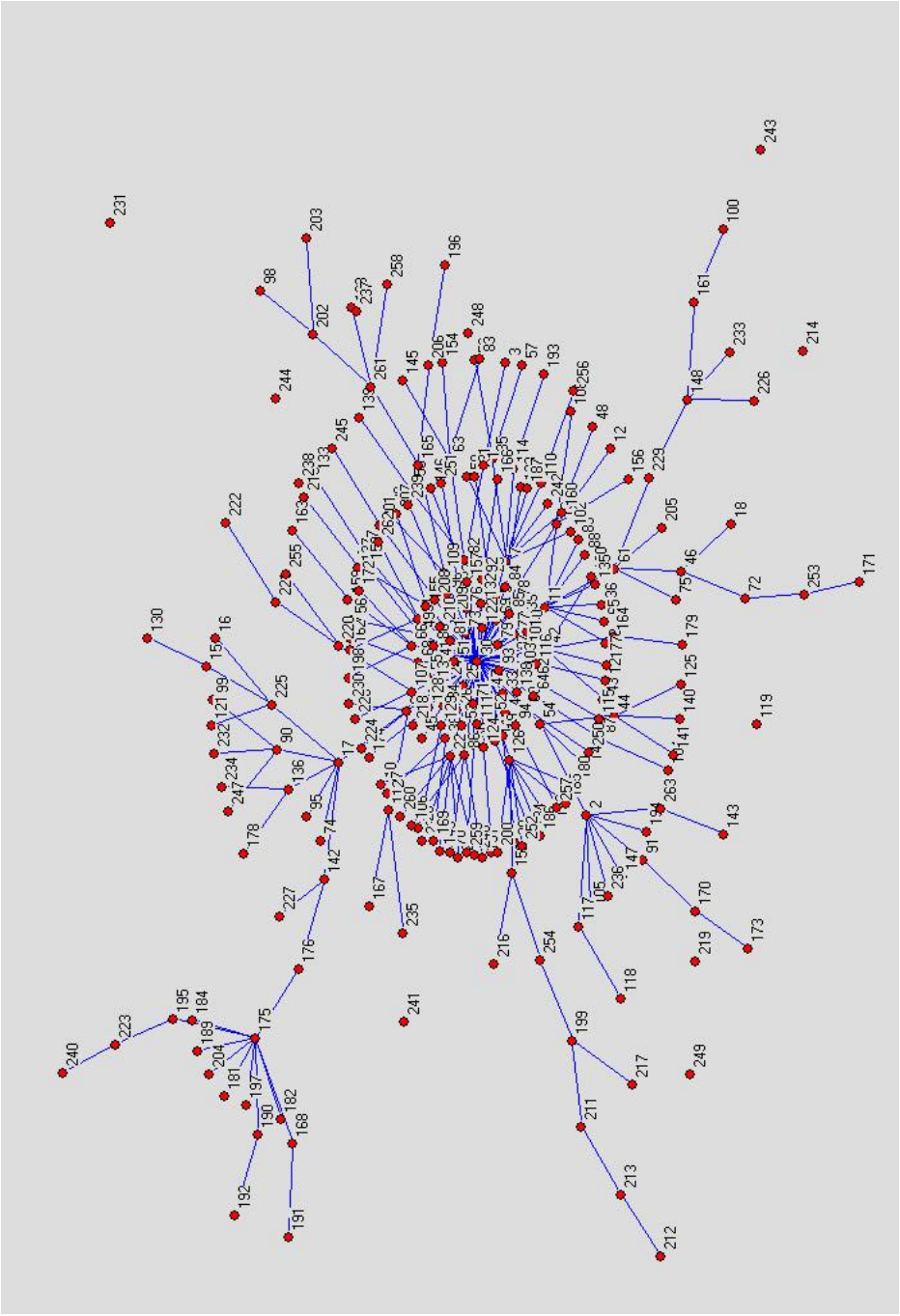

Figure 5: The result of the modified Dijkstra algorithm on the Argentina Science map, for the node 30 (in the same time the node starting the edge with the maximum value, the node having the maximum number of connected edges, and the node having the maximum average of the weights of the connected edges).

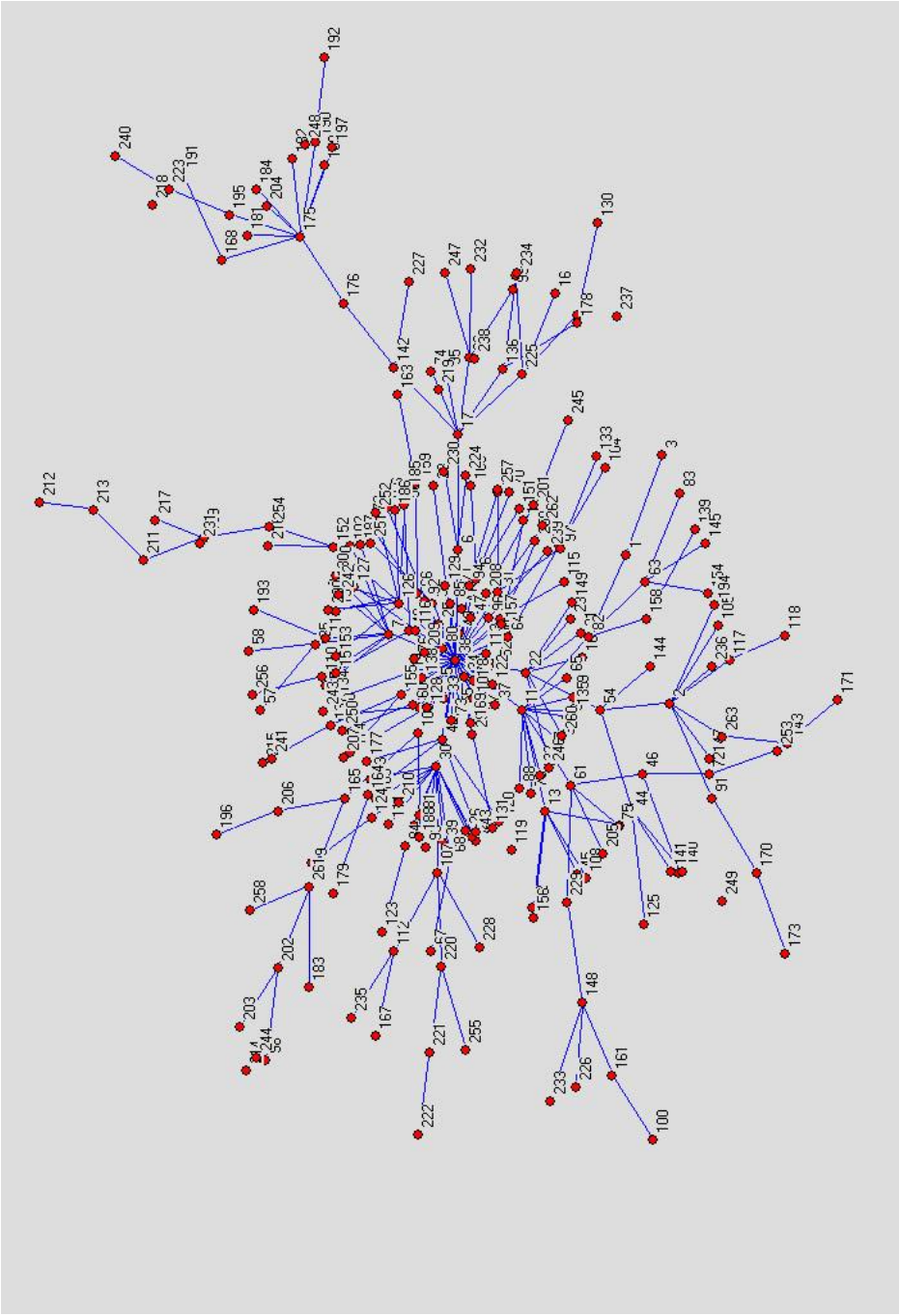

Figure 6: The result of the modified Dijkstra algorithm on the Argentina Science map, for the node 38 (the node ending the edge with the maximum value).

1. $E = \emptyset$
2. FOR each node $v \in S[G]$

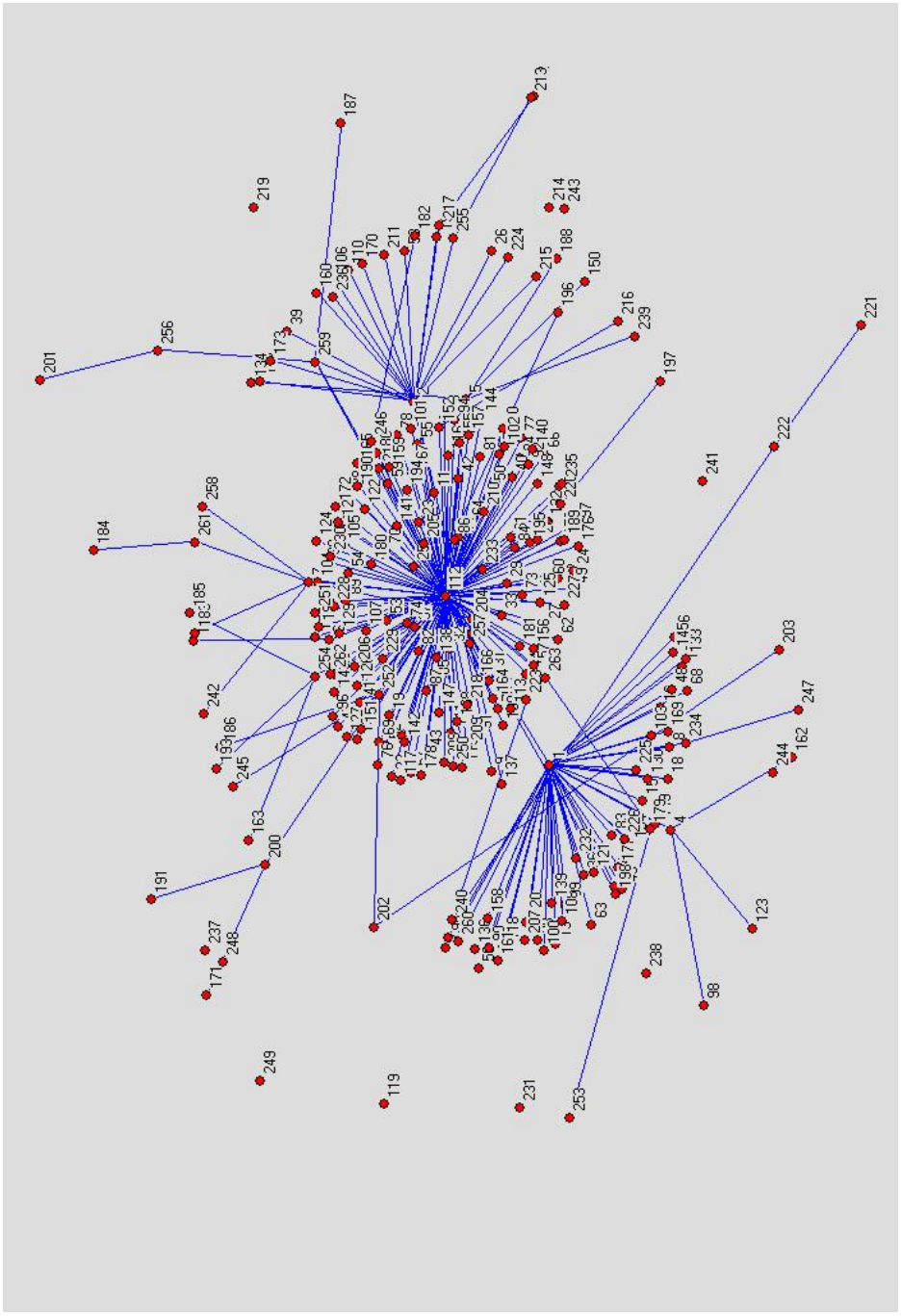

Figure 7: The result of the modified Dijkstra algorithm on the Argentina Science map, for the node 235 (the node having the minimum number of connected edges).

3. CREATE-CONNECTED-COMPONENT(v)

4. F = Set of all the edges of G sorted by their weights

5. FOR each edge $(u, v) \in F$
6. IF CONNECTED-COMPONENT$(u) \neq$ CONNECTED-COMPONENT(v), THEN
7. $E = E \cup \{(u, v)\}$
8. CONNECT-CONNECTED-COMPONENT(u, v)
9. Return E

5.3 The modified Kruskal algorithm

We have designed a new algorithm, to take into account the behavior observed in the modified PFNet algorithm with $q = n - 1$, $r = \infty$ and the *bottleneck* behavior. Because we are looking for the links with maximum weights in the graph, we have called this class of algorithms, *Maximum Spanning Tree*. This is the modified Kruskal algorithm:

1. $E = \emptyset$
2. FOR each node $v \in S[G]$
3. CREATE-CONNECTED-COMPONENT(v)
4. $F =$ Set of all the edges of G sorted by their weights, in a descending way
5. FOR each edge $(u, v) \in F$
6. IF CONNECTED-COMPONENT$(u) \neq$ CONNECTED-COMPONENT(v), THEN
7. $E = E \cup \{(u, v)\}$
8. CONNECT-CONNECTED-COMPONENT(u, v)
9. Return E

By traversing the edges using a decreasing order of their weights, the first edge that will be included in the network is the edge with the maximum value, corresponding to the best direct edge in the sense of PFNet (because no other path could provide a better value). In a recursive way, connecting (during the next steps) the edge with the best value to the current graph could lead in a graph similar to the one obtained with modified-PFNet. But any Kruskal-like algorithms will produce trees, and not graphs, as the modified PFNet algorithm does in the general case. First, have a look at a trivial example in which the Kruskal and the Dijkstra algorithms will not produce the same graphs.

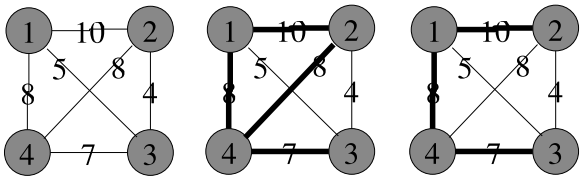

Figure 8: Left part: a graph G. Middle part: the modified-PFNet graph. Right part: the modified-Kruskal tree.

In the figure 8, the results obtained with the modified PFNet and the modified Kruskal algorithms are not the same. In spite of the fact that the edges with the largest values are selected first, the edge

$2-4$ is not included in the Kruskal Tree. This comes from the fact that, using the modified Kruskal algorithm, the nodes 2 and 4 are already in the same connected component (the edge $1-2$ was analyzed first because it has the largest value, and we suppose that the algorithm orders the edge $1-4$ before the edge $2-4$, so $1-2-4$ is a connected component at the step of the algorithm when $2-4$ is analyzed) and consequently the test in the line 6 is not validated. This edge is included in the PFNet graph, because in the modified PFNet algorithm, no strict-order comparison is applied. To be clear, in the PFNet graph, the edge $1-4$ with value 8 is included, in spite of the fact that the path $1-2-4$ has the same value; and the edge $2-4$ with value 8 is also included, in spite of the fact that the path $2-1-4$ has the same value.

So, we have guessed that these edges may have a special property, corresponding to an edge existing in a PFNet graph but not in a Kruskal tree. The property is that these edges form a group of edges with the same value, and all of them connect (if analyzed one by one) two different connected components (not necessarily the same). This group is named $\mathcal{G}$ in the following. In the Fig. 8, the edges $1-4$ and $2-4$ (with the same weight 8) connect (after the first step of the algorithm) the connected components {1,2} and {4}, so they form a group $\mathcal{G}$. In other words, edges connecting the same connected components (if analyzed independantly during the line 5 of the modified Kruskal algorithm) could also be included in the modified-Kruskal graph. Note that, in that case, the produced graph could not be a tree. This will lead in a graph equivalent to the PFNet graph and will be proven in the section 5.8.

5.4 The K-Fast-PFNet algorithm

Now we will analyze a new adaptation of the modified Kruskal algorithm, called *K-Fast-PFNet*, in which the edge-property previously explained is developped to add all the edges in a group $\mathcal{G}$ in the final graph, when they are met. So the graphs produced by modified-PFNet and K-Fast-PFNet will be always the same, even in the general case. In the previously seen modified Kruskal algorithm, two different connected components U and V are immediatly fusioned if they are linked with the current edge processed during the line 5. But in this case, in the next run, a just fusioned connected component introduces a bias in the test at the line 6, and some edges could be discarded. Here the idea is to wait until all the edges with the same values are processed (all edges of a group $\mathcal{G}$), and only fusion the corresponding components after. These edges are placed in a temporary set G (line 11), and the fusion occurs after (lines 12-13).

This is the K-Fast-PFNet algorithm:

1. $E = \emptyset$
2. FOR each node $v \in S[G]$
3. CREATE-CONNECTED-COMPONENT(v)
4. $F =$ Set of all the edges of G sorted by their weights, in a descending way
5. FOR each edge $(u, v) \in F$
6. $G = \emptyset$
7. FOR each edge $(u', v') \in F$
8. IF $w(u, v) = w(u', v')$ AND CONNECTED-COMPONENT(u') $\neq$ CONNECTED-COMPONENT(v'), THEN
9. $E = E \cup \{(u', v')\}$
10. $F = F - \{(u', v')\}$
11. $G = G \cup \{(u', v')\}$

12. FOR each edge $(u', v') \in G$
13. CONNECT-CONNECTED-COMPONENT(u', v')
14. Return E

Note that the K-Fast-PFNet algorithm has the same complexity of the modified Kruskal Algorithm (because the size of the set of edges F decreases each time), that is $O(A.log(n))$.

5.5 The Prim algorithm

In the Prim algorithm, the edge with the minimum value is selected, then a tree is growed from this edge, adding neighbor edges. In the best case, the complexity of this algorithm is $O(A + n.log(n))$ [1]. For the application on the Visual Science Maps, the complexity of the Prim algorithm is better than the Kruskal algorithm (for instance, with the Argentina maps, with 263 nodes and 19562 edges, the complexity is about 47300 for the Kruskal algorithm, and about 20200 for the Prim algorithm).

Let $d[u]$ and $p[u]$ be the distance and the predecessor attributes for the node u, and s be an arbitrarly chosen node. The original Prim algorithm is:

1. FOR each node $u \in S[G]$
2. $d[u] = \infty$
3. $p[u] = \emptyset$
4. $d[s] = 0$
5. $F = S[G]$
6. WHILE $F \neq \emptyset$
7. $u =$ EXTRACT-MIN(F)
8. FOR each node $v \in$ NEIGHBORS(u)
9. IF $v \in F$ and $w(u, v) < d[v]$, THEN
10. $d[v] = w(u, v)$
11. $p[v] = u$

5.6 The modified Prim algorithm

As with the previously seen modified Kruskal algorithm, the Prim algorithm can be rewritten to take into account the behavior of the PFNet algorithm. In this new algorithm, the node with the greatest distance attribute is extracted from F and is used as the root from which the growing of the tree is done. The analogy with a PFNet graph can be seen recursively. We suppose that in a given step, the current tree expressed by the $p[u]$ attributes is a PFNet-like tree, i.e. any path between two nodes in this tree contains the largest *bottleneck* compared to all the other possible paths contained in the original graph. This tree contains all the nodes of $S - F$. The next step is to grow the tree from the node m having the greatest value. For any edge $m - n$, if n is a unseen node, the edge is directly added to the tree; and if n is a previously seen node, the edge is added only if its weight is greater than the value of n. Because m has the greatest value, any edge added to the tree has a value greater than the link with the minimum weight already in the tree, so that the largest *bottleneck* behavior is respected.

The modified Prim algorithm is:

```
1. FOR each node u ∈ S[G]
2.         d[u] = 0
3.         p[u] = ∅
4. d[s] = ∞
5. F = S[G]
6. WHILE F ≠ ∅
7.         u = EXTRACT-MAX(F)
8.         FOR each node v ∈ NEIGHBORS(u)
9.                 IF v ∈ F and w(u, v) > d[v], THEN
10.                        d[v] = w(u, v)
11.                        p[v] = u
```

This algorithm produces trees and not graphs, so the results are not the same that ones produced with PFNet in the general case. But for the same reasons explained previously, this algorithm can produce the expected PFNet maps with Visual Science Maps, because the weights of the edges are not highly similar.

It is also theoretically possible to build a P-Fast-PFNet algorithm, i.e. a version of the Prim algorithm respecting the PFNet behavior in the general case. As seen with the K-Fast-PFNet algorithm, some edges are missing to form the expected PFNet graph. In this case, the missing edges are those having a value equal to the minimum weight already in the tree, so that the other possible best path are also included in the final graph. But this requires to compare the current edge value to those already existing in the graph, increasing in the same time the complexity of the algorithm. In the case of K-Fast-PFNet, this sort is done in a global way at the start of the algorithm. But in the Prim case, the tree is grown in a local way, so an additional loop is required to test for the missing edges. Note also that in this case, a more complex predecessor list should be save, having the ability to save more than one predecessor for each node (saving a graph, and not a tree), and being able to replace the predecessor list for a given node when a better edge is encountered.

5.7 Results

The results are presented in Fig. 9.

5.8 Comparison between modified PFNet and MST algorithms

A modified-PFNet graph generated from a graph G is equivalent to the tree generated by a Maximum Spanning Tree algorithm, including in addition some links having a special property (see Fig. 10). A typical Maximum Spanning Tree algorithm (such as the modified Kruskal algorithm) analyzes edges in an order given by their weights. But if several edges $e_1, \ldots, e_n$ have the same weight w, and if (at a given step of the algorithm) these edges connect two different connected components, an order is chosen in an *apriori* way (for instance, first encoded, first analyzed). In this case, a MaxST algorithm will give in fact *one* of the possible maximum spanning trees, including only one of these edges. It can be proven recursively that the modified PFNet algorithm includes all of these edges (the only point is that they have to connect different connected components if their are analyzed independantly) in the final produced graph.

Map	Mod-PFNet	Fast-PFNet (way 1)	Fast-PFNet (way 2)
Argentina.net	146	0.46	0.52
Chile.net	107	0.37	0.41
Cuba.net	60.4	0.27	0.30
Espana.net	114	0.37	0.41
Mexico.net	133	0.42	0.46
Portugal.net	145	0.44	0.50
Based on	PFNet	Floyd-Warshall	Floyd-Warshall
Time complexity	$O(qn^3)$	$O(n^3)$	$O(n^3)$
Space complexity	$O((3+q)n^2)$	$O(n^3)$	$O(2n^3)$

Map	Mod-Kruskal	K-Fast-PFNet	Mod-Prim
Argentina.net	0.043	0.042	0.033
Chile.net	0.054	0.051	0.042
Cuba.net	0.035	0.036	0.030
Espana.net	0.063	0.063	0.051
Mexico.net	0.061	0.064	0.047
Portugal.net	0.064	0.062	0.047
Based on	Kruskal	Kruskal	Prim
Time complexity	$O(A.log(n))$	$O(A.log(n))$	$O(A+n.log(n))$
Space complexity	$O(A+2n^2)$	$O(A+2n^2)$	$O(A+2n^2)$

Figure 9: Comparison of all the algorithms (computation times in seconds and complexities).

Proof: if each group $\mathcal{G}$ of size 1 and sorted by decreasing order of their weights is added one by one to the final graph, the best remaining direct edge is added each time, corresponding to the behavior of the modified PFNet algorithm. In the case of a group $\mathcal{G}$ of a size greater than 1, each edge of this group could be added by the modified PFNet algorithm during the corresponding step of the algorithm, because they connect two different connected components and they have the best value compared to any other edge able to connect these two components. So, all the edges of $\mathcal{G}$ are included in the best path connecting any nodes from these components, and the modified PFNet and K-Fast-PFNet algorithms are equivalent.

We can also discuss about the probability that the modified Kruskal algorithm produces the same results or not than the modified PFNet algorithm. These two algorithms are equivalent as long as they are applied on any graph not containing a group $\mathcal{G}$ of a size greater than 1. Commonly, this kind of graphs should have a lot of edges with similar values (at least two), and the situation is more frequently encountered with edges having small or integer weights. As Visual Science Maps contains real and big values, the modified-Kruskal algorithm could be used instead of the K-Fast-PFNet algorithm, having a high probability to produce the same graphs (in fact trees), but faster. Tests have been conducted on all the maps described in this paper (Fig. 3), and the results were the same. This is interesting, because modified-Kruskal is a really fast algorithm, compared to the modified PFNet or Fast-PFNet algorithms (see Fig. 9).

5.9 Mathematical comparison between MST + PFNet

The new variant of the Pathfinder algorithm is based on the Minimum Spanning Tree (MST) algorithms. Typical MST algorithms such as the Kruskal or the Prim algorithms are greedy approaches. As *Binary Pathfinder*, the current state-of-the-art algorithm used to generate PFNET is a dynamic programming algorithm, the algorithms based on the MST algorithms should be faster. Let $G = (V, E)$ a

#	**Domain (year)**	**#Nodes**	**#Links**	Original PF	Binary PF	Fast PF (predecessor)	Fast PF
1	Argentina (2005)	263	19562	110309.5	5447.66	320.66	268.28
2	Chile (2004)	242	17914	56025.26	2928.4	192.7	162.76
3	China (2002)	212	8541	37644.78	2544.5	179.7	151.36
4	Cuba (2004)	219	10644	45319.38	2065.04	200.1	169.82
5	Cuba (2006)	221	11286	33784.98	2813.92	144.96	122.8
6	Europe (2002)	218	17242	53357.56	2169.38	202.36	123.6
7	France (2002)	216	10087	30105.24	2909.78	187.1	159.1
8	Germany (2002)	218	11745	33631.5	2099.06	137.8	117.2
9	Japan (2002)	213	9028	27041.28	2288.76	177.26	182.88
10	Mexico (2005)	250	21264	100131.76	5569.52	304.16	183.98
11	Peru (2002)	218	8485	41196.14	2866.12	196.98	166.58
12	Portugal (2005)	254	22179	79733.1	4767.96	227.38	194.18
13	Spain (1994)	219	13478	49800.4	3022.02	197.92	169.28
14	Spain (1998)	223	16226	44860.4	2854.34	152.94	129.82
15	Spain (2002)	240	19183	77421.34	4723.66	266.86	225.52
16	Spain (2004)	240	23430	56890.9	4716.78	192.64	229.6
17	UK (2002)	218	13567	50484.46	2147.4	200.56	120.56
18	USA (2002)	218	18132	54046.88	2195.02	148.54	126.38
19	Venezuela (2005)	239	15415	50741.26	4248.92	257.08	154.86
20	World (2002)	218	20154	37976.02	2178.64	198.28	168.54

Table 1: Comparison of all the algorithms (computation times are expressed in milliseconds on an Intel dual-core 3.4 GHz CPU with 2 GB of memory)

non-directed connex graph. A MST of G is a sub-graph $T = (V, E')$ of G, $E' \subset E$, including all the vertex of G, where T is a tree and where the sum of the cost of each edge is minimal.

The relation between the result of a Minimum Spanning Tree (MST) algorithm and a Pathfinder algorithm is well known in the literature. For instance, [**?**] stated that for a given symmetric cost matrix W, r and q, the union of all the MST extracted from a PFNET(r, q) is PFNET$(\infty, n-1)$. Although it is not stated explicitly in his paper, the union of all the MST extracted from a given graph G is also PFNET$(\infty, n-1)$. This can be easily proved by contradiction: if $e(i, j)$ is an edge of a MST, this edge belongs also to the minimal cost path between the node i and j, so to the graph PFNET$(\infty, n-1)$. If not, it should be possible to build a better MST by adding the edge $e(i, j)$ to the tree, and by deleting another edge. In the same way, all the edges of the PFNET$(\infty, n-1)$ belongs to at least one MST of G. We have use this idea to implement a quicker algorithm for the generation of PFNET(r, q) when r is restricted to ∞ and q to $n-1$.

So to generate the PFNET$(\infty, n-1)$, we have to compute all the possible MST of a graph and return the union of the corresponding edges. The two corresponding well known algorithms, to compute a MST, are *Kruskal* and *Prim*. The *Prim* algorithm grows an initial tree by looking all the neighbors of a given vertex and by selecting the one than minimise the current cost of the tree. This neighbor-behavior seems to be more efficient when the networks are represented as a real graph (by the way of pointers, or some other structures). In our case, the scientific domains are represented by co-citation matrices and the *Kruskal* algorithm seems to be more suited. Another more technical and important reason for this choice will be explained later.

In the Kruskal algorithm, all the edges are sorted in an ascending order. The edges are then added one by one in the final tree only if they do not connect the same cluster (in such a way that cycles are avoided). In this case, at each iteration, two nodes belonging to the current tree are only connected by the edges having the minimal cost. The algorithm uses several sub-functions. The function CREATE-CLUSTER(v)

applied to a node v create a single cluster of size 1 and corresponding to v. The function CLUSTER(v) return the cluster associated to the node v. Usually this is done by returning an element identifying in an unique way the cluster containing v. The function MERGE-CLUSTER(u, v) union the cluster containing the node u with the one containing the node v.

The original Kruskal algorithm is:

1. Define a tree, $T = \emptyset$
2. FOR each node $v \in S[G]$
3. CREATE-CLUSTER(v)
4. Create F, a set of all the edges of G sorted by their weights
5. FOR each edge $(u, v) \in F$
6. IF CLUSTER(u) $\neq$ CLUSTER(v), THEN
7. $T = T \cup \{(u, v)\}$
8. MERGE-CLUSTER(u, v)
9. Return T

Our proposal concerns a way to compute the union of all the possible MST of a given graph with the same time complexity required to compute one of these MST. It could be noticed that the difference between the different MST comes from the edges having the same values, but still not present in the same cluster during a given step of the algorithm. At this step, the algorithm have the choice between different edges, and all these choices correspond to the same amount of different MST. In particular, if all edges have a different weights, the MST is unique. This corresponds to the only non-deterministic behavior of the original Kruskal algorithm, that let this algorithm to produce different trees. In the following, we will call *multiple-edges* the non-common edges between all the possible MST generated from a given graph.

So, to achieve our final goal of merging directly the different MST during the run of the algorithm, we have to detect these multiple-edges. The first property of these edges is that they have to have the same weights. This is computationally efficient because we can use the sort of the edges done in the initialisation of the algorithm. If the original algorithm is looked carefully, we can notice that once one of these edges are added to the tree T, the two clusters corresponding to this edge and to T are also merged. In this case, it is impossible in a further step to detect if two edges are or not multiple-edges.

The only solution is to store them in a temporary set H instead of adding them directly to the current tree, and process this set only when we are sure that this will not affect the detection of other edges. The processing of the set (i.e. the union of all the edges of the temporary set with the current tree) can be done once a new edge with a different weight has been found. These remarks let us to define directly the algorithm K-Fast Pathfinder, shown below:

1. Define a tree, $T = \emptyset$
2. FOR each node $v \in S[G]$
3. CREATE-CLUSTER(v)
4. Create F, a set of all the edges of G sorted by their weights
5. FOR each edges (u, v) remaining in F
6. $H = \emptyset$

```
7.      FOR each edges (u', v') remaining in F where w(u, v) = w(u', v')
8.          F = F − {(u', v')}
9.          IF CLUSTER(u') ≠ CLUSTER(v'), THEN
10.             T = T ∪ {(u', v')}
11.             H = H ∪ {(u', v')}
12.     FOR each edge (u', v') ∈ H
13.         MERGE-CLUSTER(u', v')
14. Return T
```

It is also interesting to note that this improvement can only be done with the Kruskal algorithm, and not the Prim algorithm. Indeed, in the Kruskal algorithm, the edges-sorting acts in a global way, allowing us to detect (and union) the different edges that would be present in the different MST trees. In the Prim algorithm, the growing of the tree is done in an incremental so a local way, by adding a non-explored edge to the current tree. In this condition, the detection of two edges with the same values that should be merged could not be achieved directly, because they could be located far away from each other, at least for a minimal cost (a sorting would be required at each iteration to do so).

As the K-Fast Pathfinder algorithm is based on the Kruskal algorithm, it as the same time complexity. The algorithm needs $O(|E|.log(|E|))$ operations to sort the list of the edges by their weights, where $|E|$ is the number of edges. To know in which cluster belongs each vertex, we can use a disjoint-set data structure with union by rank and path compression [1]. The cost of the CREATE-CLUSTER(v) function is $O(1)$. The cost of the functions CLUSTER(v) and MERGE-CLUSTER(v) can be proved to be $O(log(n))$. Although there is two imbricated FOR loops, at most $|E|$ CLUSTER(v) and MERGE-CLUSTER(v) operations are performed in the worst case. So, as we have $|E| < n^2$ and $log(|E|) = O(log(n))$, the theoretical cost of the full algorithm is $O(|E|.log(n))$. In conclusion, this algorithm is much more fast than the original Pathfinder algorithm, in spite of the recent improvements proposed in the literature. For instance, Fast Pathfinder, proposed in [] has a time complexity of $O(n^3)$ when applied with $q = n-1$.

It is important to mention that the time complexity developed previously has only a theoretical importance. In practice, due to the high number of calls to the CLUSTER(v) functions, compared to the number of calls to MERGE-CLUSTER(u, v), it is more efficient to consider another data structure than the disjoint-set to implement these functions. In our case, each node v is affected to a single index $c(v)$ encoding in a unique way the corresponding cluster, so each call to CLUSTER(v) is done in $O(1)$. Then, MERGE-CLUSTER(u, v) consists to affect the value $c(u)$ to each node having the value $c(v)$, and this can be done in $O(n)$. So the practical time complexity of the algorithm is $O(|E|.(n + log(n)))$.

Concerning the space complexity, we need to store three lists of edges with their weights, that are F, T and H. The record of the cluster can be done with a single additional attribute for each node, so the total space complexity of the algorithm is $3.|E| + n$.

ABSTRACT: The main idea concerning the K-Fast Pathfinder algorithm comes from the fact that the superposition of all the Minimum Spanning Trees (MST) extracted from a given network is equivalent to the result of the Pathfinder algorithm parametrized by a specific set of parameters ($r = \infty$ and $q = n-1$). Although this property is well known in the literature [**?**], it seems that no algorithm have been proposed to decrease the high computational cost of the Pathfinder algorithm, using ideas based on the MST algorithms.

It should also be noted that these parameters are the most used to produce PFNET networks for large domains [].

5.10 Comparison between Shortest Path and MST algorithms

Now, have a look about the results obtained with Shortest Path based algorithm (such as the modified Dijkstra algorithm) and a MST based algorithm (such as K-Fast-PFNet). All of these algorithms produce trees, but these trees are not equivalent. Proof: let G be a graph in which several paths connecting a source node and a destination node have a link L in common, and this link has the minimum value w (the remaining of all the links have a value greater than w). In G, the source and the destination nodes are connected with paths having the same value w (because the MAX of the values of each independant path has the same value w). So, a Shortest Path algorithm will select, for the final tree, one of these paths (it could be the first met, ...), but a Maximum Spanning Tree algorithm will select the path having the link with the maximum value, because of the pre-ordering step of all the edges. In conclusion, when several trees are possible using different paths with the same minimum value, a Shortest Path algorithm selects one of the path in a non-deterministic way, and a MST algorithm selects the path according to another criterion depending of the value of all the links, so, in the general case, these two trees will not be the same.

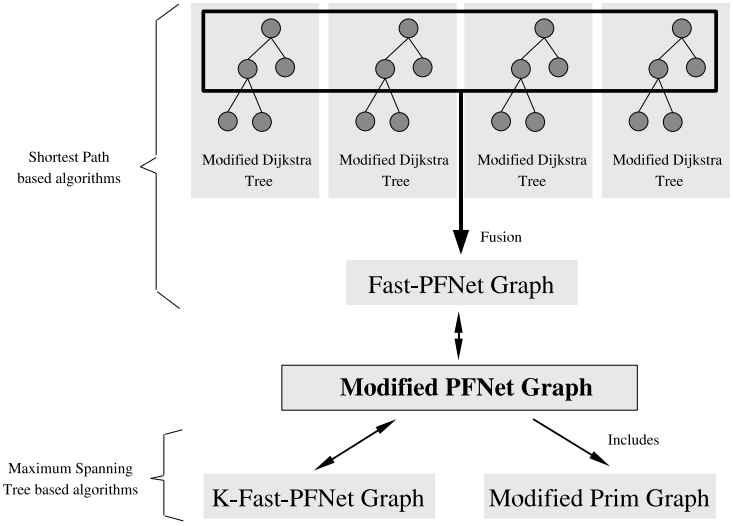

Figure 10: A comparison between the output of the modified PFNet algorihm, the modified Dijkstra algorithm, Fast-PFNet, K-Fast-PFNet, and the modified Prim algorithm. A double-sided arrow indicates an equivalence, and a single-sided arrow indicates a fusion or an inclusion.

6 Conclusion

In this paper, we have analyzed several algorithms producing Visual Science Maps similar to that ones produced by the modified PFNet algorithm. Algorithms coming from the Shortest Path class and from the Maximum Spanning Tree class have been developped, tested and discussed on several examples. These algorithms were first adapted to meet the specific behavior of a modified PFNet algorithm. The first class of algorithms, the Shortest Path algorithms such as Dijkstra and Floyd-Warshall, are usefull to produce shortest path trees from a given node, or any node. In the case of the Dijkstra algorithm, this node can be set up to give some interesting graphical properties to the output graph. For instance, a good solution is to take the node connecting the edge with the maximum value or the node connecting the maximum number of edges. The Floyd-Warshall algorithm gives several trees, each one corresponding to a part of the final solution (i.e. a graph equivalent to the modified PFNet algorithm) and a refinement

has been proposed to fusion these trees. The second class of algorithms, the Maximum Spanning Tree algorithms, provide algorithms able to generate PFNet graphs, but in a different way. In this class of algorithms, a tree is generated first, than edges are added (instead of deleted) to form the final graph. These edges can be described by a specific property. In the most common maps, this property is not observed, in a way that a simple adaptation of a Maximum Spanning Tree algorithm can lead to the same graph as modified-PFNet does. Based on this remark, we have developped a *fast* version of the Kruskal algorithm to solve the problem quickly, but not being able to solve the general case, and a *reliable* version of the same algorithm to solve the problem in the general case. All these algorithms have been tested on several real world examples to retrieve time statistics.

References

[1] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms, Second Edition.* The MIT Press, 2001.

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

[3] R. W. Schvaneveldt. *Pathfinder Associative Networks.* Ablex, Norwood, NJ, 1990.

[4] Wikipedia, 2006. Scientometrics, In: Wikipedia. The free encyclopedia. Available on http://en.wikipedia.org/wiki/Bibliometrics. Accessed on 11th, December, 2006.