Large Quasi-Tree Drawing: A Neighborhood Based Approach

Romain Bourqui, David Auber

Large Quasi-Tree Drawing: A Neighborhood Based Approach

Romain Bourqui and David Auber
Eindhoven University of Technology, Netherlands and LaBRI, Université Bordeaux I, France
R.Bourqui@tue.nl and david.auber@labri.fr

Abstract

In this paper, we present an algorithm to lay out a particular class of graphs coming from real case studies: the quasi-tree graph class. Protein and internet mappings projects have shown the interest of devicing dedicated tools for visualizing such graphs. Our method addresses a challenging problem which consists in computing a layout of large graphs (up to hundred of thousands of nodes) that emphasizes their tree-like property in an efficient time. In order to validate our approach, we compare our results on real data to those obtained by well known algorithms.

Keywords— **Graph visualization, graph drawing, graph clustering, quasi-tree graph.**

1 Introduction

Graphs are useful in many research areas such as: biology, microelectronics, social sciences, data mining and also computer science. Improvements in data acquisition techniques raise a challenging problem of visualization. Indeed the size and the complexity of these graphs do not enable to manually draw them. Therefore, graph drawing and information visualization communities focus on designing visualization of these data.

Theoritical solutions have been found to lay out particular classes of graphs such as trees, planar graphs, directed acyclic graphs, or biconnected outerplanar graphs classes. These algorithms give very good results not only in terms of time/space complexity but also in terms of aesthetic criteria. However, even if these algorithms offer good solutions for these particular classes of graphs, real application graphs do not usually belong to these classes.

To draw general graphs, force directed algorithms seem well suited since they produce visually pleasant and structurally significant results (e.g. [11, 22, 24, 14]). Main drawback of these classical force directed algorithms is their time complexities since they are usually not subquadratic and cannot therefore be applied on large graphs (with thousands of vertices and edges).

Finding an algorithm giving good results (in term of computation time, estheatic criteria and information emphasized) on general graph is a very difficult problem. Most of the works on visualization of large graphs do a trade-off between computation time and aesthetic criteria. In this article, we focus on a particular class of graphs to prevent from doing that trade-off. That class of graphs has been observed in real case studies (e.g. webgraphs, biology) and is called the **quasi-tree** graphs class. In [3], the authors claim that a quasi-tree graph is a graph with $O(n)$ *biconnected* components (where n is the number of vertices of the graph), but it seems complicated to formally define a quasi-tree graph. Unformally, we can consider that these graphs have a tree-like structure (for instances, see figures 6 and 7). In this article, we focus on a visualization of such graph which emphasizes its tree-like structure property.

The remainder of this paper is structured as follows. Section 2 reviews related work on large graph visualization. Section 3 describes quasi-tree graphs and how our method allows to emphasize this tree-like property. Then in section 4, we give results of our algorithm on real data and compare them to results of other well-known algorithms.

2 Related Work

In this section, we present the three main approaches to draw large graphs. Among them, the most famous is based on improvements of force directed methods ([15, 16, 2]). Another approach consists in representing the graph as a matrix and using linear algebra techniques ([18, 27]). Finally, a recent technique is based on topologies detection and the use of an appropriate algorithm for each topology ([1, 4, 3]).

2.1 Force directed based approach

To improve the time efficiency of the force directed methods, new algorithms use vertex (or edge) filtration.

This allows to reduce the number of elements considered when drawing the graph since each level of the filtration is drawn separatly.

In [15], Gajer and Kobourov present a vertex filtration called the *Maximal Independant Set Filtration*. This method consists in computing a set of sets $V_0, V_1, ..., V_k$ such that $V = V_0 \supset V_1 \supset ... \supset V_k$ where $\forall u, v \in V_i$, $dist_G(u, v) \geq 2^i$ and $|V_k| = 3$. The three vertices of V_k are laid out such that the euclidean distances respect the graph distances. Then, each other V_i, $0 \leq i < k$ (starting from $k-1$ and going down to 0) is drawn using either Kamada and Kawai algorithm [22] or Fruchterman and Reingold algorithm [24].

Hachul and Jünger describe in [16] another vertex filtration in which the graph is considered as a set of solar systems composed of *suns*, *planets* and *moons*. Then, each solar system is collapsed into a single vertex and that process is repeated to produce a serie of graphs $G = G_0, G_1, ..., G_k$, where G_k is considered as "small" enough. In addition to this multi-scale technique, the authors use a grid to approximate the repulsive forces, thus the overall complexity becomes $O(n \cdot log(n))$.

The LGL algorithm presented in [2] was first designed to visualize a protein map and can therefore handle weighted edges. The filtration in that algorithm is made by first computing a *minimum spanning tree* of the graph using the well-known Kruskal's algorithm [23] and then rooting that tree on the graph center. This rooted tree allows to know the order in which vertices of the graph must be inserted in the layout. Starting from the root, the algorithm draws one by one each level of the spanning tree. Again, to improve the computation time, the algorithm uses a grid to approximate repulsive forces.

2.2 Matrix based approach

Harel and Koren give in [18] an algorithm consisting in computing an m-dimensional embedding of the graph (typically m is set to 50) and then to project this embedding into 2 or 3 dimensions. To define these m axis, the authors compute a set $\{p_1, p_2, ..., p_m\}$ of m *pivots* (one for each dimension) such that the distance between all p_i is maximized. In that m-dimensional space, the i^{th} coordinate of a node u is its theoritical distance (weighted or not) to p_i. Then, to reduce the dimensionality of the embedding, the authors use the *Principal Component Analysis* (for more detail on this technique, readers are refered to [13]).

In [27], Koren *et al.* give an other matrix based approach using the eigenvectors of the Laplacian matrix to compute a projection of the graph. To speed up the process, the idea of that method is to compute an estimate for the eigenvectors of the original Laplacian matrix. To do so, this algorithm constructs a hierarchy of matrices such that the size of matrix of the higher level is enough "small". As the highest level matrix is "small", they can compute the exact eigenvectors of that matrix. Then, starting from the highest level, they recursively compute the eigenvectors of the underneath level, until the eigenvectors of the original matrix are found.

2.3 Topologies detection based approach

Recent articles refer to topology detection for large graph drawing (e.g. c[1, 3, 4]). The main idea of this approach is to detect interesting or typical topological subsustructures in the network and then to draw them using an appropriate algorithm.

In [1], Abello *et al.* present an algorithm that first search for the *peripheral forest* (i.e. proper subgraphs that are trees) using a technique called *pilling* and collapse them into single nodes. In the next step, the algorithm search for biconnected components of the resulting graph. Finally, if some biconnected components are "too" large (for more detail see [1]), they try to cluster it using MCL algorithm of [26]. In [1], Abello *et al.* also present the steerable exploration tool where they integrated this algorithm. Therefore, the graph is then drawn on demand using either a tree drawing or a force directed algorithm (depending on the structure to draw).

In [4], Archambault *et al.* use an other pipeline of topological structures searches. They look for proper subgraphs that are trees, biconnected components, HDE [18] suitable graphs (for more details see [4]), complete graphs and finally use *strength* [6]. Archambault *et al.* adapted this pipeline to the special properties of *quasi-tree* graphs in [3]. As mentioned above, according to Archambault *et al.*, quasi-tree graphs contains $O(n)$ biconnected components. Therefore, the technique of [3] consists in detecting the biconnected components of the graph. Then, each biconnected component is drawn using a modified version of LGL [2] and the biconnected components tree is drawn using an *area aware* version of RINGS [25]. Finally, an edge crossing reduction step is performed. This pipeline allows to improve the computation time (compared to [4]) since it only searches for caracteristical topological structures contained in *quasi-tree* graphs.

3 Quasi-tree graphs: features descriptions and method

To emphasize the tree-like property of quasi-tree graphs, our approach uses the following pipeline:

1. Group detection and coarsening

2. Bottom-up drawing

3. Edge bundling

The first step consists in finding caracteristical quasi-tree graph sub-structures using a hierarchical clustering algorithm. Then each level of the hierarchy tree is drawn using a bottom-up technique. Finally, an edge bundling step is performed to unclutter the resulting drawing.

In the next, we first describe the main topological features of quasi-tree graphs, then we present each of these three steps.

3.1 Quasi-tree graphs topological features

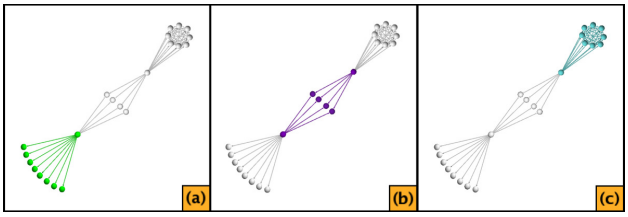

Figure 1. Typical topological structures of quasi-tree graphs: (a) a star (in green), (b) a diamond (in purple) and (c) a complete graph (in blue).)

In this paper, we consider that a quasi-tree graph contains three main types of topological structures: **stars**, **diamonds** and **cliques**.

First, a **star** is a subgraph of the quasi-tree induced by a set of vertices V' such that V' contains one *universal* vertex[1] and each other vertex of V' has a degree equal to 1 (see figure 1.a).

Then, a **diamond** is a subgraph induced by a set V', such that V' contains two vertices u and v linked to all other nodes in V' (except to u and v) and each other vertex has a degree equal to 2 (see the purple central nodes in figure 1.b). This definition can generalized to **multiple diamond**, i.e. a subgraph induced by a set of vertices $V' = V_1 \cup V_2$ such that each vertex of V_1 is linked to all vertices of V_2 and each vertex of V_2 has a degree equal to $|V_1|$ (for instance, see figure 2.a).

And the last topological structure is the **clique**, i.e. an induced complete subgraph (see figure 1.c).

In pratice, quasi-tree graphs do not contain many occurences such "perfect" structures. Therefore we add to these topological structure definitions a degree of freedom. This degree of freedom corresponds to either few edges linking "extremities" of a star (resp. "central" nodes of a diamond), or to few missing edges in a clique.

The property being characteristic of each of these structures is the similarity of their neighborhood (this notion is explained in section 3.2.1). Therefore, to detect such topological structures, our clustering algorithm is based on nodes neigborhood and groups vertices having "similar" neighborhoods.

[1] i.e. linked to all other vertices of V'.

3.2 Groups Detection

The first step of our approach is to compute groups of vertices having "almost" the same neighborhood. To do so, we have to compute a dissimilarity measure that we will call *neighborhood dissimilarity measure*. This measure is based on the Jaccard's index [20] where the attributes of one element (a vertex) are its neighbors. The next step consists in building a hierarchycal clustering of the graph according to that measure (figure 2 illustrates that process).

3.2.1 Neighborhood dissimilarity measure

To determine whether or not two vertices are "similar", we define the *neighborhood similarity measure* as follow : $\forall u \in V$ and $v \in V$:

$$sim(u,v) = \frac{|(N(u)\backslash\{v\}) \cap (N(v)\backslash\{u\})| + f(u,v)}{|(N(u)\backslash\{v\}) \cup (N(v)\backslash\{u\})| + f(u,v)},$$

where $N(u)$ is the neighborhood of u in the graph G and $f(u,v)$ is equal to 1 if u and v are neighbors, 0 otherwise.

Clearly, $sim(u,v) \in [0,1]$ and if two vertices u and v have exactly the same neighborhood then $sim(u,v) = 1$. We also define the *neighborhood dissimilarity measure* : $\forall u \in V$ and $v \in V$:

$$dissim(u,v) = 1 - sim(u,v)$$

It is straightforward to prove that $\forall u, v \in V$, if $dist_G(u,v) > 2$ then $sim(u,v) = 0$ and $dissim(u,v) = 1$. Thus, we only need to compute the similarity (dissimilarity) of each pair of vertices at distance less or equal to 2.

To find all pairs of vertices at distance less or equal to 2, we use n *Breadth-First Search* (BFS) of depth 2. This can be done in $\sum_{u \in V} (deg(u))^2$ which is $O(nd_{avg}d_{max})$ time and space complexity (where d_{avg} and d_{max} are respectively the average and the maximum degree) . Then to compute all similarities (dissimilarities), we have to compare the neighborhoods of the $O(nd_{avg}d_{max})$ pairs of vertices. In the data structure we use, the neighborhood of vertex is sorted, so that comparing the neighborhoods of two vertices u and v has computation cost of $O(deg(u) + deg(v))$.

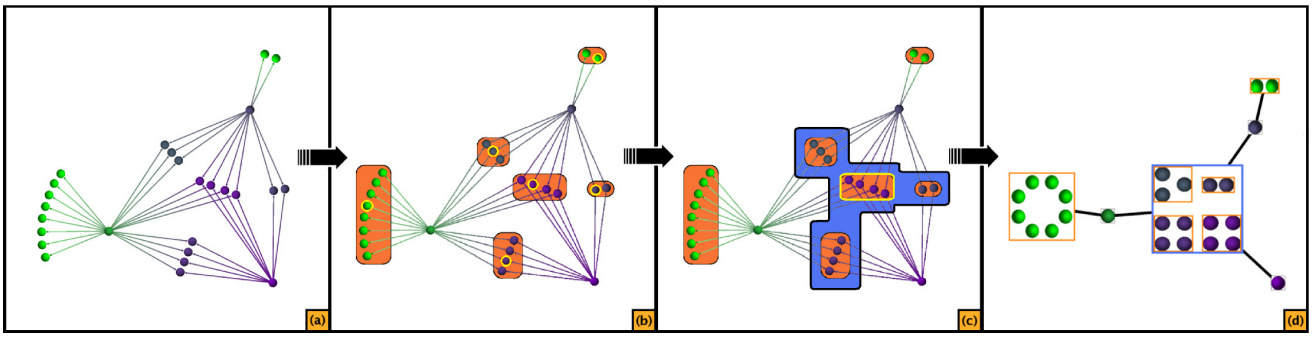

Figure 2. Illustration of the group detection process. (a) Subnetwork of the website of the Computer Science department of the University Bordeaux 1; (b) and (c), two successive passes of the clustering algorithm with ϵ respectively equal to 0.1 and 0.35. Nodes surrounded in yellow correspond to u_{root} defined in section 3.2.2; (d) the final corresponding quotient graph.

The overall process has a $O(nd_{avg}(d_{max})^2)$ time complexity and a $O(nd_{avg}d_{max})$ space complexity. If we consider that G is of bounded degree, we obtain a $O(n)$ time and space complexity.

3.2.2 Clustering algorithm

To compute our groups, we use an approach based on DBSCAN [12]. In this algorithm, groups of vertices are computed using two thresholds ϵ and k. If the similiraty between two nodes is more than ϵ, then these two nodes cannot be in the same group. Moreover a group cannot contain less than k vertices (in pratice, we use $k = 2$). Our algorithm repeats iteratively a group detection based on DBSCAN and thus needs two parameters : ϵ_{min} and ϵ_{max}, respectively the minimal and the maximal thresholds ϵ that will be used to compute the clusters.

The first step of the clustering algorithm consists in sorting the vertices by their average similarity with the vertices at distance less or equal to 2 from them. We call V_{sorted} this list of sorted vertices. Clearly, this step can be done in $O(m + nlog(n))$ time complexity and $O(n + m)$ space complexity.

The second step consists in finding the groups of the current level. To do so, we create a new cluster and add the first not yet considered vertex u_{root} of V_{sorted} in that cluster. Then, we add to this cluster all vertices v at distance at most 2 from u_{root} such that $dissim(u_{root}, v) \leq \epsilon$. This process is repeated for each vertex of V_{sorted} not already added in a cluster. It is straightforward to prove that the time and space complexities of that step are respectively $O(nd_{avg}d_{max})$ and $O(n + m)$.

And the third step consists in constructing the *quotient graph* correponding to these clusters (i.e. a graph obtained by collapsing each cluster into a single vertex). If a vertex does not belong to a cluster, we consider that these vertex belong to a cluster only containing it. To construct the quotient graph, we add to it one node (*metanode*) for each cluster in graph. And for two metanodes C_u and C_v respectively corresponding to $\{u_1, u_2, ...u_p\}$ and $\{v_1, v_2, ...v_q\}$, there exists an edge between C_u and C_v in the quotient graph if there exists i and j such that (u_i, v_j) is an edge of the original network. Moreover, the dissimilarity between C_u and C_v is set to the average dissmilarity between u_i and v_j, $\forall i \in \{1, .., p\}$ and $\forall j \in \{1, .., q\}$. To do so, we first map each vertex of the graph on the metanode of the quotient graph it correspond to. And then we just need to traverse each edge of the graph to find the edges of the quotient graph, thus the third step has a $O(n + m)$ time complexity.

This leads to a $O(nd_{avg}d_{max}+nlog(n))$ time complexity and a $O(n+m)$ space complexity to compute one level of the hierarchycal clustering.

Then, we increase the threshold ϵ to $\epsilon + c$, where c is a constant (in pratice we use $c = 0.05$) and we repeat these three steps on the quotient graph while $\epsilon < \epsilon_{max}$. As ϵ varies from ϵ_{min} to ϵ_{max} by steps of c, this three steps are repeated $(\epsilon_{max} - \epsilon_{min})/c$ times, thus the time and space complexities are respectively $O(nd_{avg}d_{max} + nlog(n))$ and $O(n + m)$.

The overall groups detection (including the dissimilarity computation) is done in $O(nd_{avg}(d_{max})^2 + nlog(n))$ time complexity and a $O(nd_{avg}d_{max})$ space complexity. At the end of the clustering algorithm, we obtain a multi-level quotient graph which is the *backbone* of the network. We will use this backbone to lay out the graph.

3.3 Bottom-up Drawing Algorithm

To draw the graph, we use a bottom-up approach that is to say we draw one by one the clusters of the quotient graphs corresponding to each level, starting from the deepest level and ending by the highest. The main advantage of the bottom-up techniques is that they allow to know the exact area needed to lay out each cluster. We make a distinction between the top level quotient graph and the other levels of the hierarchy. As the top level quotient graph can be large, we use modified version of LGL algorithm [2] while we use either a circular drawing algorithm or and the force directed algorithm described in [14] to draw clusters. All these algorithms are modified to take into account the sizes of the nodes.

3.3.1 Clusters drawings

To draw one cluster, we first detect if it is a complete graph or if on the contrary it is a stable graph (i.e. a graph with no edge). In these cases, we use a circular drawing algorithm. For the others topologies, we use the force directed algorithm describe in [14]. Even if this algorithm does not offer a good computation time, it gives good results in term of stretch and as groups do not contain a large number of nodes (or metanodes), it is usefull.

Making the circular algorithm area aware is straitforward. For the algorithm of [14], we take into account the sizes of the nodes when computing attractive forces. This solution does not give "perfect" result (see [17]) since some node-node overlaps remain in the drawing. To remove such overlaps, we use the algorithm of Dwyer *et al* [10] : this algorithm first generates the "separation" constraints according to the original nodes placements and then find a solution to these constraints

3.3.2 Top level drawing

To draw the top level quotient graph, we use the modified version of LGL described in [3]. This extension of LGL [2] reduce the number of node-node overlaps by modifing the size of the grid cells used in LGL. It is possible to completely avoid node-node overlap if the size of each cell is enought large. These modifications allow to take into account the sizes of the nodes (and the metanodes). The problem is that our clustering algorithm does not compute a balanced clustering, thus to avoid completely node-node overlaps we should set the size of each cell of the grid to an high value and it would conduct to a very large layout. Therefore we chose to allow some overlaps (by bounding grid size) and to remove it at the end of the drawing by using the algorithm of [10].

3.4 Edge Bundling

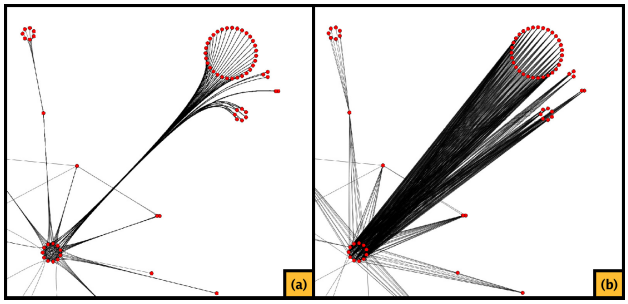

Figure 3. A subgraph of the internet network of the University of British Colombia visualized with edge bundling (a) and without edge bundling (b).

Last step of our drawing algorithm is based on the work of Holten [19]. In this article, the author use a standard tree visualization of the hierarchy. Then to embed, each non-hierarchical edges, he uses B-spline curves. For instance, if an edge e links to nodes u and v, the edge e "follows" the shortest path along the hierarchy from u to v. The idea is to "merge" edges linking "close" clusters in the hierarchy. In our approach, the hierarchy is determined by the clustering algorithm and is then lay out. We consider that the position of each node in the hierarchy tree is lay out at the barycenter of the nodes it corresponds to in the graph. This technique enables to reduce the edge cluttering in the final drawing (for instance see figure 3). It also allows to emphasize the "tree-like" property of the graph.

4 Results

In this section, we present results obtained on real application graphs: some webgraphs and a biological network. Firstly, we give some sample of computation times, and secondly, we compare two of these results to those obtained by some algorithms described in section 2.

4.1 Computation time results

All our experiments were performed on a Intel Core Duo 2.4 GHz processor with 2GB RAM. Figure 4 provides the results obtained on a sample of graphs.

These graphs can be sorted in three differents types: websites, biological networks and internet map. Websites graphs are extracted by starting from the home page of a website and then by following each hyperlink of the page in

Types	Graphs	Nodes	Edges	avg degree	max. degree	Times
Websites	UBC	2,002	4,802	4.79	150	6 s.
	Stanford	4,002	12,835	6.41	246	53 s.
	Bordeaux	6,865	8,242	2.4	460	38 s.
	Yale_15K	15,634	18,367	2.34	422	61 s.
	LaBRI	23,536	32,343	2.75	714	218 s.
	Yale_42K	42,496	66,386	3.12	745	686 s.
	San Jose	55,340	290,418	10.5	3446	8683 s.
Biological networks	b7_ferea	25,114	45,799	3.65	331	391 s.
	pgraph	30,727	1,206,654	78.51	1112	1.5 h.
Internet map	Net05	190,384	228,354	2.4	994	5 h.

Figure 4. Performance results of our algorithm. Computation times are given in secondes, except for pgraph and Net05.

a *Breadth First Search* (BFS) manner. Examples of figure 4 are subnetworks of american universities (Stanford, Yale and San Jose univerties), a canadian (University of British Colombia), a french university (University of Bordeaux 1) and its computer Science Department (LaBRI Laboratory, University of Bordeaux 1). Graphs of the second type are biological networks: b7_ferea is constructed by intagrating data coming from Gene Ontology [5], KEGG [21], Cellzome (see `www.cellzome.com`) and SWISSPROT [7]. This graph was provided by BLASTSETS [8]. And, pgraph is a protein homology graph provided by the LGL project and presented in [2]. Last type is internet map, more precisely this is the internet tomography dataset generated in 2005 by Cheswick's Internet Mapping Project [9] (see `www.cheswick.com/ches/map/`). Computation times given in figure 4 are clearly better than those offered by classic force directed algorithms since our approach allows to draw graphs with thousands of nodes in few seconds/minutes.

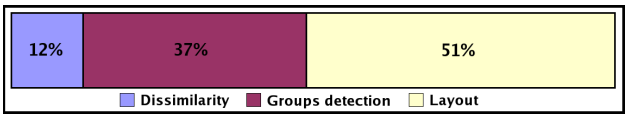

Figure 5. Average computation times spent during each step of the algorithm: dissimilarity computation, groups detection and layout of the graph.

Figure 5 shows the average percentage of time spend by each steps of our approach for laying out a graph. Most of the computation time of the algorithm is spent during the groups detection (37%) and the layout (51%) parts of the algorithm. Thus, it could be interesting to improve these two steps.

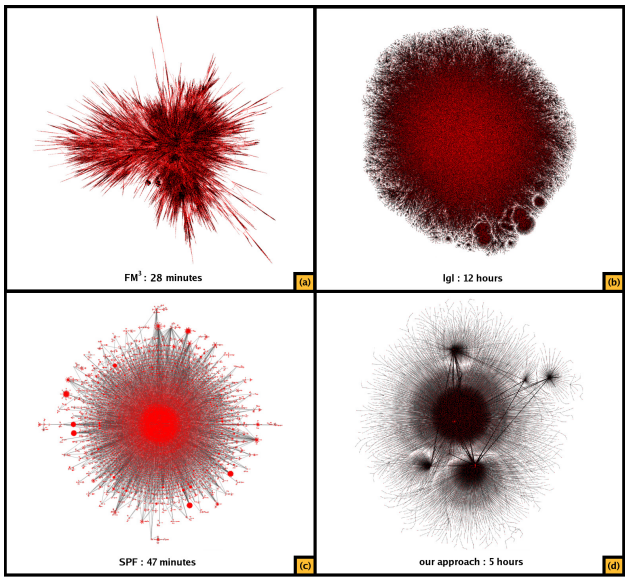

Figure 6. Drawing of Net05, obtained from the Internet Mapping project. Graph contains 190,384 nodes and 228,354 edges. Drawings produced by (a) FM^3, (b) LGL, (c) SPF, (d) our approach, with computation times indicated underneath.

4.2 Results comparisons

We compare results obtained on two real application graphs: a webgraph, Net05 from the Internet Mapping Project [9] and pgraph from the LGL Project [2]. These results are compared to results obtained by other algorithms. We decided to compare our results to FM^3 of [16], LGL of [2] and to SPF of [3]. The choice of these three algorithms is dictated by three different criteria: first of all, the computation time (FM^3 of [16]), then the esthetic criteria (LGL of [2]), and finally, the "tree-like" property emphasizing (SPF of [3]).

Figures 6 and 7 show the resutlts obtained by FM^3, LGL, SPF and our approach. In term of computation time, the algorithm FM^3 [16] is clearly the most efficient over all these algorithms. It is 10 times faster on Net05 and more than 20 times faster on pgraph than our approach. The second fastest algorithm is SPF [3] which offers quite good computation times (52 minutes on Net05 and 47 minutes on pgraph). For the LGL algorithm [2], computation time on Net05 is better using our algorithm whereas they are comparable on pgraph. This is certainly due to the average and maximum degree of pgraph that are high (see figure 4), furthermore the group detection of our algorithm takes a

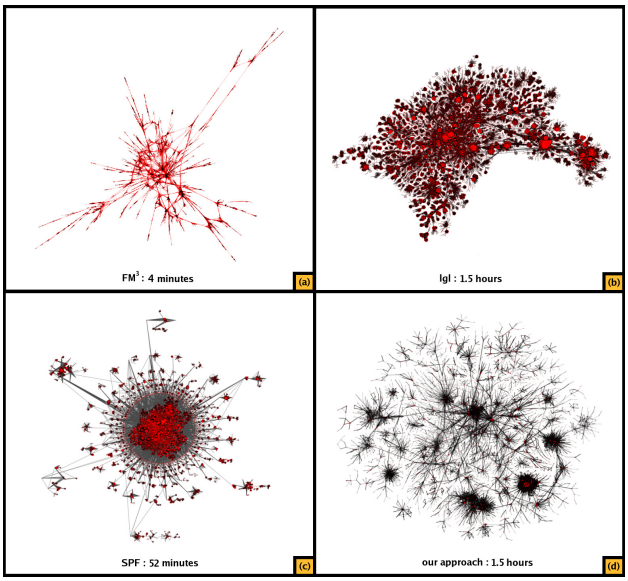

Figure 7. Drawing of pgraph, a protein map coming from the LGL Project [2]. Graph contains 30,727 nodes and 1,206,654 edges. Drawings produced by (a) FM^3, (b) LGL, (c) SPF, (d) our approach, with computation times indicated underneath.

long time on this graph.

In term of quality, FM^3 does not offer good results since it does not emphasize many information. Nodes are laid out along thin "branches" of the graph (for instance, see figure 7.a). Thus neither the global nor the local structures of the graph are shown. Results of LGL are better than those of FM^3. It succeeds in embedding the overall structure on pgraph (see figure 7.b) since we can easily see the branches (global tree-like structure) and highly connected groups of proteins (local structures). However, on Net05, LGL only succeeds to emphasize the peripheral tree-like structures but fails on the "central" part of the graph since all the inner part of the graph is embed in an undecipherable manner. Results of SPF are shown in figures 6.c and 7.c, this approach raises two main problems: first of all, if the graph does not contain many biconnected components or worst is biconnected (for instance in figure 7.c, the central part of the drawing is a biconnected component containing more than 21000 nodes, i.e. more than 70% of the vertices) then this technique will consist in using LGL. Secondly, the global structure of the graph is not shown, indeed given two biconnected component of 6.c, it is not possible to affirm if these two structures are close or not in the graph when it is an essential information.

Finaly, results of our approach are shown in figures 6.d and 7.d. The global tree-like property is well emphasized by our algorithm. In Net05 (see figure 6.d), high level branches of the graph appear clearly. Moreover another information is highlighted by our algorithm: these branches are "plugged" into several highly connected subnetworks. These subnetworks are the pivots of the network. Thus, one can see the peripheral and also the inner tree-like structure of the graph. Furthemore, the local structures of the graph are well shown, for instance in figure 7.d, groups of proteins are clearly visible.

5 Conclusion

In this paper, we presented an algorithm to lay out large quasi-tree graphs. This is done by first detecting typical topological features contained in quasi-tree graphs and then by drawing these structures using appropriate algorithms. Our method addresses a challenging problem which consists in computing a layout that emphasizes the tree-like property of the graph in an acceptable computation time.

We compared our results on real data coming from biology and an internet mapping project to those obtained by well knwon algorithms. We have shown that even if our method is not the most efficient in term of computation time, it improves the readability of the results. Indeed our approach allows to emphasize both global and local tree-like structure of the graph.

In the near future, we plan to improve the computation time by speeding up the layout step of our algorithm. It could be done by improving the time efficiency of the LGL algorithm. Another way could be to modify FM^3 to improve the quality of its results.

References

[1] J. Abello, F. Van Ham, and N. Krishnan. Ask-graphview : A Large Graph Visualisation System. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):669–676, 2006.

[2] A.T. Adai, S.V. Date, S. Wieland, and E.M. Marcotte. Lgl: creating a map of protein function with an algorithm for visualizing very large biological networks. *Journal Mol Biol*, 340(1):179–190, 2004.

[3] Daniel Archambault, Tamara Munzner, and David Auber. Smashing peacocks further: Drawing quasi-trees from biconnected components. *IEEE Transactions on Visualization and Computer Graphics (Proc. Vis/InfoVis 2006)*, 12(5):813–820, 2006.

[4] Daniel Archambault, Tamara Munzner, and David Auber. Topolayout: Multi-level graph layout by topological features. *IEEE Transactions on Visualization and Computer Graphics*, 13(2):305–317, 2007.

[5] M. Ashburner, C.A. Ball, J.A. Blake, D. Botstein, H. Butler, J.M. Cherry, A.P. Davis, K. Dolinski, S.S. Dwight, J.T. Eppig, M.A. Harris, D.P. Hill, L. Issel-Tarver, A. Kasarskis, S. Lewis, J.C. Matese, J.E. Richardson, M. Ringwald, G.M. Rubin, and G. Sherlock. Gene ontology: tool for the unification of biology. the gene ontology consortium. *Nat Genet.*, 25:25–29, 2000.

[6] D. Auber, Y. Chiricota, F. Jourdan, and G. Melançon. Multiscale visualization of small-world networks. In S. C. North and T. Munzner, editors, *Proc. of IEEE Information Visualization Symposium*, pages 75–81, Seattle, USA, 2003. IEEE Computer Press.

[7] A. Bairoch, R. Apweiler, C.H. Wu, W.C. Barker, B. Boeckmann, S. Ferro, E. Gasteiger, H. Huang, R. Lopez, M. Magrane, M.J. Martin, D.A. Natale, C. O'Donovan, N. Redaschi, and LS. Yeh. The universal protein resource (uniprot). *Nucleic Acids Res.*, 33:D154–159, 2005.

[8] R. Barriot, J. Poix, A. Groppi, A. Barré, N. Goffard, D. Sherman, I. Dutour, and A. de Daruvar. New strategy for the representation and the integration of biomolecular knowledge at a cellular scale. *Nucl. Acids Res*, 32(12):3581–3589, 2004.

[9] B. Cheswick, H. Burch, and S. Branigan. Mapping and visualizing the internet. In *Proc. USENIX*, 2000.

[10] T. Dwyer, K. Marriott, and P. Stuckey. Fast node overlap removal. In *Proc. Graph Drawing 2005 (GD'05)*, pages 153–164, 2005.

[11] Eades. A heuristic for graph drawing. In *Congressus Numerantium*, volume 42, pages 149–160, 1984.

[12] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proc. 2nd Int'l Conf. on Knowledge Discovery and Data Mining*, pages 226–231, 2003.

[13] B. S. Everitt and G. Dunn. *Applied Multivariate Data Analysis*. Arnold, 1991.

[14] A. Frick, A. Ludwig, and H. Mehldau. A fast adaptive layout algorithm for undirected graphs. In *Proceedings Graph Drawing 1994 (GD'94)*, pages 388–403, 1994.

[15] Pawel Gajer and Stephen G. Kobourov. GRIP: Graph dRawing with Intelligent Placement. In *Graph Drawing 2000 (GD'00)*, pages 222–228, 2000.

[16] S. Hachul and M. Jnger. Drawing large graphs with a potential-field-based multilevel algorithm. In *Proc. Graph Drawing 2004 (GD'04)*, pages 285–295, 2004.

[17] D. Harel and Y. Koren. Drawing graphs with non-uniform vertices. In *Proc. Working Conference on Advanced Visual Interfaces (AVI'02)*, pages 157–166. ACM Press, 2002.

[18] D. Harel and Y. Koren. Graph drawing by high dimensional embedding. In *Proc. Graph Drawing 2002 (GD'02)*, pages 207–219, 2002.

[19] D. Holten. Hierachical edge bundles: Visualization of adjacency relations in hierarchical data. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):805–812, 2006.

[20] P. Jaccard. Distribution de la flore alpine dans la bassin de dranses et dans quelques régions voisines. *Bulletin de la Société Vaudoise des Sciences Naturelles*, 37:241–272, 1901.

[21] M. Kanehisa and S. Goto. KEGG: Kyoto encyclopedia of genes and genomes. *Nucl. Acids Res.*, 28(1):27–30, 2000.

[22] Tomihisa Kamadaand Satoru Kawai. An algorithm for drawing general undirected graphs. In *Information Processing Letters*, volume 31, pages 7–15, apr 1989.

[23] J. B. Kruskal. On the shortest spanning subtree and the traveling salesman problem. In *Proceedings of the American Mathematical Society*, number 7, pages 48–50, 1956.

[24] Thomas M. J. Fruchtermanand Edward M. Reingold. Graph Drawing by Force-directed Placement. In *Software-Practice and Experience*, volume 21(11), pages 1129–1164. nov 1991.

[25] S. T. Teoh and K. Ma. Rings: A technique for visualizing large hierarchies. In *Proc. Graph Drawing 2002 (GD'02)*, pages 268–275, 2002.

[26] S. van Dongen. *Graph Clustering by Flow Simulation*. PhD thesis, Universiteit Utrecht, 2000.

[27] L. Carmel Y. Koren and D. Harel. Ace: A fast multiscale eigenvectors computation for drawing huge graphs. In *Proc. IEEE Symposium on Information Visualization*, pages 137–144, 2002.