Volume xx (*200y*), *Number z, pp. 1–10*

Tulip an information visualization software for huge graphs

D. Auber

LaBRI-Université Bordeaux 1, 351 Cours de la Libération, 33405 Talence, France
email:auber@labri.fr

Abstract

This paper presents some of the most important features of a graph visualization framework called Tulip, developed in order to experiment with tools such as clustering, graph drawing, and metric coloring for the purpose of information visualization. Tulip's main characteristics are a graph hierarchy that permits and encourages sharing of elements, and a general property evaluation mechanism that makes the software reusable and easily extendable. We demonstrate the efficiency of our approach by providing a complete program called Tulip which can display, modify, cluster and automatically draw of graphs with up to 1,000,000 elements on personal computers with 256MB of memory.

1. Introduction

Information visualization is becoming more and more useful for the correct analysis of existing data sets. This need results from progress in data acquisition methods, and from the huge effort made to build computer access to the human knowledge. As an example, for the human genome database, the raw data acquisition phase seems to be completed; however, to reach the ultimate goal of providing new medical treatment, it is necessary to understand these data. In such an application, the information visualization challenge is to devise tools that give humans understandable views of the data in order to explore and extend knowledge.

Here we focus on data that can be represented by a graph. In most of cases a graph structure can be extracted from existing data sets. The most well-known is the World Wide Web where links between pages can be considered as edges and pages as nodes. Another one is the human metabolism data-set where chemical reactions can be embedded in a Petri net, literature co-citations are modeled as edges between nodes of this network, and metabolic pathway are considered as clusters of the resulting graph.

Systems to visualize graphs have come to the fore during the last ten years. [17, 12, 19, 26, 18, 6, 14, 2]. To our knowledge, no one provides the following capabilities simultaneously :

- Visualization and navigation in 2 or 3 dimensions.
- Support of huge graphs.
- Support of graph modifications.
- Management of clusters.
- Management of unbounded number of shared properties between graphs.
- A mechanism for evaluating internal properties.
- Extension and reuse without recompilation of the software.
- Free of use and open source.

To experiment with tools to handle graphs of the size of those induced by the human genome data set, one needs a software solution with all these capabilities. That's why we decided to build our own graph visualization software that meets these requirements. Tulip has been developed in C++, and uses two well-known libraries, OpenGL[1] and Qt[23]. The final program enables visualization, clustering and automatic drawing of graphs with up to 1,000,000 elements on personal computer with 256MB of memory.

After some definitions, this paper presents an overview of the Tulip data structure. Then it introduces the reader to the set of properties implemented in Tulip in order to make a graph visualization program. Subsequently, after a presentation of the software human computer interface, we give a sample implementation of a plug-in and then we conclude with a brief overview of existing plug-ins.

2. Definitions

This section introduce some definitions about graphs and the data structure that we use in what following.

Definition 1 (Graph) Let V and $E \subset V \times V$ be two sets. We call graph the structure $< V, E >$. We note it G(V,E). Elements of V are called nodes, and elements of E are called edges. Let $e = (u, v)$ be an edge; we call u the source and v the target.

Definition 2 (Sub-Graph) Let $G(V, E)$ and $G'(V', E')$ two graphs. $G'(V', E')$ is a sub-graph of $G(V, E)$ if $V' \subset V$ and $E' \subset E$. We note it $G' \prec G$. Respectively, we call $G(V, E)$ an upper-graph of $G'(V', E')$.

Definition 3 (Induced Sub-Graph) Let $G(V, E)$, $G'(V', E')$ with $G' \prec G$, G' is an induced sub-graph of G iff $E' = \{e(u, v) \in E | u, v \in V'\}$.

Definition 4 (Cluster Tree) A cluster tree is a tree $T(V, E)$ that represents a hierarchy of graphs. Each node $n \in V$ represents a graph G_n and for each edge $\varepsilon = (u, v) \in E$, $G_v \prec G_u$.

Definition 5 (Property) Let K_1,K_2 two sets. A property of a graph $G(V, E)$ is a pair of functions $\{F_V : V \to K_1, F_E : E \to K_2\}$. Such a property will be denoted by π^G.

The above definition allows one to abstract the concept of graph attributes without using object programming terminology.

Definition 6 (PropertyContainer) We call a PropertyContainer of a graph $G(V, E)$ the structure $< L, P, F >$. L is a set of labels, P is a set of properties and F is a function from L to P. For each $\pi^{G_i} \in P$, $G_i = G$ or $G \prec G_i$. We note it Π^G.

This definition is also given in order to abstract the concept of PropertyContainer, in the following we will discuss how the set of properties and labels are built, and we describe precisely how the function F maps labels to properties.

Definition 7 (SuperGraph) We call SuperGraph the structure $< G(V, E), \Pi^G, C >$. C is a Cluster Tree in which the root node represents G.

Definition 8 (Graph View) Let $G'(V', E') \prec G(V, E)$. We call graph view the structure $< G'(V', E'), \Pi^{G'}, S_G, \eta >$. S_G is a SuperGraph, η is the node in the cluster tree of S_G that represents $G'(V', E')$.

3. Data structure.

We want to handle graphs having 1,000,000 of elements (nodes and edges). In this case, memory management is a crucial factor for building an efficient framework. In order to limit memory swapping and the processor cache faults, one must put a special emphasis on the trade-off between memory space and processor time use. Furthermore, to our knowledge the best size bounds of grid layout algorithms are $O(n^2)$[15]. Thus, taking into account resolution of computer screens, it is hard to give understandable view of graphs with more than 1,000 nodes. Therefore, for huge graphs, we must provide clustering mechanisms for reducing manually or automatically the size of data during the visualization process.

3.1. Graph hierarchy

In Tulip, clustering is done by using the SuperGraph and GraphView structures introduced above. Viewing a graph hierarchy through a cluster tree and a set of subgraphs allows us to have only one graph in memory and to provide access to it through views. Thus, a view can be considered as a filter on the graph stored in memory. In the figure 1, one can see a graph G and a set of clusters $\{G_1, G_2, G_3, G_4, G_5\}$. In this example, clusters are sub-graphs induced by the nodes inside colored boxes. The figure 2 is the cluster tree that represents the hierarchy of graphs of the figure 1. On the tree edges of the figure 2, boxes represent adjustable filters. The idea of adjustable filters is to store in memory a minimum of information in order to rebuild clusters dynamically. For instance, for the filter $F2$ it is better to store the difference between G_2 and G if one wants to minimize the memory usage. At the contrary, for the filter $F3$ it is better to store all nodes and edges of G_3.

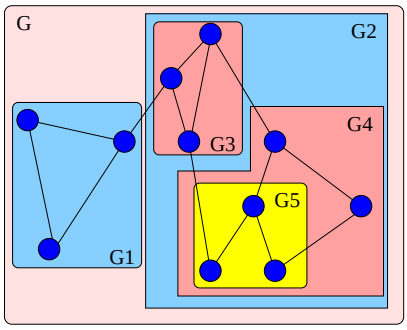

Figure 1: *A clustered graph*

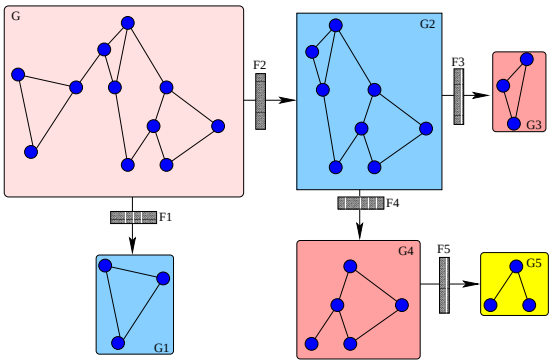

Figure 2: *The cluster hierarchy*

To maintain the coherence of the data structure one must add some precision on the modification operations. When one wants to add a node or edge in a cluster, this node or edge must be inserted into all upper-graphs of the cluster. When one wants to delete a node or edge in a cluster, this node or edge must be deleted from all the sub-graphs of the cluster.

In practical cases of graph visualization, the processor time cost due to the management of the hierarchy coherence is offset by reduction in memory use and by the improvement of important operations such as graph cloning or element sharing.

3.2. Graph attributes

We now introduce attribute management in Tulip. An attribute is a value attached to a node or an edge. In a graph visualization framework we can divide the attributes in two types, the ones predefined in the data or set by the user and the ones computed by algorithms provided by the framework. For instance, when we load a labeled graph, the nodes' labels are predefined attributes, and when we compute a graph layout, the nodes' coordinates are computed attributes. It is easy to see that in both cases, we can abstract the attributes with the Property definition introduced above. In Tulip, this abstraction allows optimization of the memory use. For instance, setting to zero all elements of a graph $G(V,E)$ is equivalent to building a property $(F_N : N \rightarrow \{0\}, F_E : E \rightarrow \{0\})$. It also enables one to include useful mechanisms such as buffered evaluation of recursive functions. To include such an improvement we construct properties called proxies which use other properties. Information about proxies can be found in the Design Pattern book written by Gamma and al[7]. The plug-in code example presented in section 6 illustrates the utility of such a mechanism.

In graph visualization, one must manipulate several properties at the same time. For example, if one visualizes a file system, files name, files size and files type are attributes of the graph nodes. In Tulip, the PropertyContainer introduced above is used to manage property sets. It allows one to store an unbounded number of properties and to modify dynamically the set of properties attached to a graph. For instance, it makes it possible to store several graph drawing at the same time.

During clustering, one wants to keep attributes on elements. This is equivalent to sharing attributes between graphs. For example, in a file system, the files names never change even during visualization of a sub-directory. However, one also needs to be able to locally change a property of a cluster. For instance, if a file system has been drawn with a tree map algorithm[3], it can be useful to change the drawing of a sub-directory by using a tree walker algorithm[20]. In this case, one can look at the structure of the sub-directory instead of the file sizes. In Tulip, both cases are taken into account by using the PropertyContainer and the Cluster tree. The mechanism set-up is analogous to those included in object languages that permit inheritance, redefinition and dynamic change of object structure[25]. The difference in Tulip is that we allow the inheritance of data. A complete example is presented below.

In figure 3 one can see a set of graphs with attributes. G is a loaded graph that contains two properties: layout and labels. G_1 is a spanning directed acyclic graph (DAG) of G, due to the inheritance of properties we have directly the layout and the labels of its elements without any cloning of data. G_2 is also a spanning DAG of G that has been redrawn using a DAG dedicated algorithm. In this case, we redefine the layout properties locally to G3. Thus, labels are still inherited from G and layout is contained in G_2. In the graph G_3, one can see that to define the set of inherited properties, we choose the nearest ones in the cluster tree. Therefore the layout of G_3 is the one of G_2 and not the one of G.

3.3. Extension mechanism

To include new features easily, a plug-in mechanism has been built inside Tulip. It works as follows. During initialization of Tulip data structure, Tulip's configuration directories are scanned to find new computed properties. Those found are added to a property factory. Now, when one asks for a graph property, if it doesn't exist (locally or inherited) the PropertyContainer queries the property factory for a computed property. If one exists, after initialization and calculability checks, the property is added to the graph. If such a computed property doesn't exist then a data property is created. Tulip includes other plug-ins mechanism in order to support the addition of algorithms that are not properties. The existing ones are the import/export of graphs, and the clustering of graphs.

4. Graph visualization

The data structure presented above efficiently manages hierarchies of graphs with attributes. In this section, we present the set of properties provided inside Tulip in order to make a graph visualization program. All the presented properties work as the general one introduced above.

4.1. Layout

When visualizing huge graphs, hand-made drawing cannot be done by the user. Therefore one needs algorithms to automatically assign coordinates to graph elements. Such algorithms are widely studied in the literature. One can reefer to the Dorothea Wagner and Michael Kaufmann book[16] or to the annotated bibliography written by Giuseppe Di Battista and al[10] to obtain further informations on this subject. Few algorithms are briefly presented in the section 6. In trying to use these algorithms for information visualization we noticed that using only one kind of drawing for the visualization is not enough to get inside data. For instance, force-model algorithms shows up symmetries in the graph in which it is useful to localize parts of the data which have the same structure and upward-forward drawing shows up information about the hierarchy inside the data. Therefore in Tulip the layout property is the following $(F_V : V \rightarrow \mathbb{R}^3, F_E : E \rightarrow [\mathbb{R}^3]^*)$. To our knowledge this representation enables to represent the results of all graph drawing algorithm.

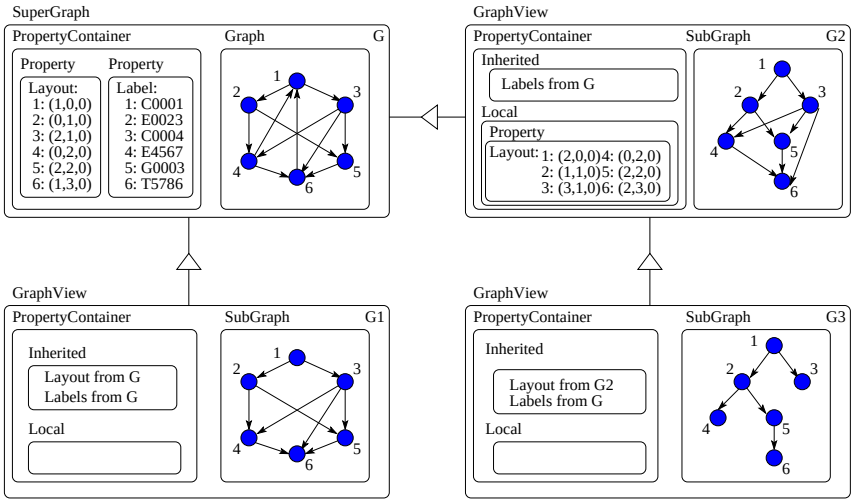

Figure 3: *Inheritance of Properties*

4.2. Metric

Metrics are used to make measures on graph elements. By mapping the results of this measures to visual attributes such as element color one can clearly show very interesting features in the data. For instance, flow measures can be used on graphs representing networks to highlight most used paths during communication. To our knowledge, complete analysis on how the graph theoretical measure can help to understand complex data structure has not been done yet. In Tulip the property ($F_V : V \rightarrow \mathbb{R}$, $F_E : E \rightarrow \mathbb{R}$) can be easily extended to new graph measures for studying their impact in terms of graph visualization.

4.3. Selection

As we cannot picture exactly what the final user is looking for inside data, interaction between user and data representation must be addressed. In Tulip the selection property ($F_V : V \rightarrow \{true, false\}$, $F_E : E \rightarrow \{true, false\}$) enables writing of complex selection algorithm. As simple example, one can select the induced subgraph by nodes at distance k of a node; by making a cluster with this selection it is possible to study locally (what is going on near a node) the properties of a graph. In metabolism data set analysis we use it to visualize in which reactions an element participates. Selection algorithms are also used in other algorithms such as graph drawing. As an example some algorithms need to remove self loops in the graph. This action is equivalent to building a cluster using a selection algorithm that selects all nodes and edges which are not self loops.

4.4. Color

As seen before, colors can be used to highlight information. However, mapping graph attributes to a color model cannot be done in the same way regarding to the attributes type and to the attributes values. For instance, Guy Melançon and al[13] posed and solved the problem of mapping real values into colors. One of the problems is that if one makes a linear mapping of real values into a color model, if the distribution of values is not linear users cannot perceive the color difference. The solution proposed is based on an analysis of histogram of values. The color property include in Tulip enables to provdie such algorithms.

4.5. Size

Just like color, the size of elements can be used to show up interesting information. As an example, when working with two dimensional drawing the height of elements can be used. By looking to the graph in three dimensions the user is able to distinguish relevant elements easily. An example is given on figure 4 where the graph as been drawn with the 2D force directed algorithm GEM[9] and elements size is computed automatically by mapping an extrinsic metric (predefined in the data).

4.6. Shape

Until now we have discussed colors, coordinates and size of elements, however computer graphics also enables to use the shape of elements. In Tulip we consider a shape as an integer, the reason of this simplification is that it enables to easily use rendering acceleration. So to assign shape to an element one must assign it an integer value. In practice the view module remaps these integer values to OpenGl[1] display lists which enable a fast displaying of huge set of elements. Just like color and size the problem of mapping attributes to a set of shapes needs to be solved; for this reason the Shape property has been provided to devise new mapping algorithm easily.

4.7. Label

This property has been provided in order to attach labels to graph elements. Even so it is hard to display labels when presenting huge data sets. In practice, after metric computation which highlight elements the user should, by using the selection mechanism or an automatic clustering algorithm, obtain a graph which can be displayed quite well even with labels.

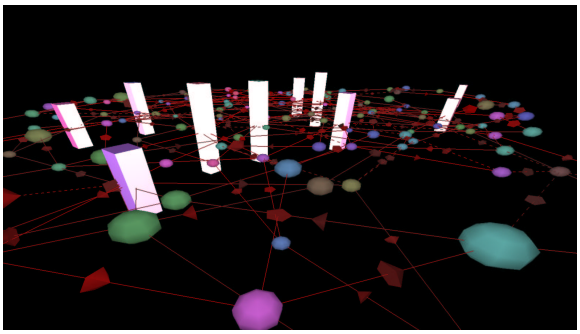

Figure 4: *Example of how visual attribute can show up information.*

5. Tulip software

Carrying out tests with end users and algorithm tuning both need a complete visualization program. The Tulip program provides a graphical interface to visualize and edit the data structure and properties introduced above. In this section we describe briefly its human computer interface (HCI).

For all of the design of the HCI we have chosen to follow the rules that emerge from image manipulation software. This choice comes from an analysis of actions that need to be performed to understand the meaning of data in huge graphs. If we consider the set of operations the differences are not so great as we can think. We need to select, to filter, to work on different graphs at the same time, to cluster, to move, to rotate, to scale etc... All these operations can be found with slight differences in an image manipulation program.

The software is composed of two kinds of windows. The first one is the application window which provides the general software tools. The second is the super-graph window, that gives access to super-graph specific operations. One can have several super-graph windows at the same time. A snapshot of Tulip HCI is given in figure 5.

5.1. Application window

The application window is composed of two parts, the first being the menu where we find all available import plug-ins. The menu is dynamically built, thus if one adds new import plug-ins in the framework they are automatically added to the menu without any modification and compilation of the software. The second part of the application window is a tool bar where mouse operations are available.

5.2. Super-Graph window

The super-graph window menu bars are dynamically built by querying the factories. It thus enables one to run the existing plug-ins and it is updated automatically if new ones are added. The existing plug-ins types taken into account are: color, clustering, export, layout, metric, shape and size. Other features such as the saving in image formats (eps,bmp,jpg etc) are also available.

Visualization and navigation are available through the 3Dview sub-window. The navigation possibilities are zoom, rotation and translation. In order to render the graph we use the OpenGl library. When hardware acceleration is available it enables one to display graph with about 300,000 elements at a usable frame rate . However, when rendering is too slow or graphs too large, we use an incremental rendering method. The principle is to draw only part of the graph each time in order to enable user interaction. The choice of the part to display first is done by using the Strahler metric[4]. The 3Dview sub-window also allows one to move, create, delete and select graph elements in three dimensional space.

A spread-sheet visualization of properties is also provided through the Property Editor sub-window. It enables one to edit, create, modify and clone the local and inherited properties of a graph. Combining both 3D displaying and spread-sheet displaying of data must be addressed in order to make the software usable for the final end-user.

Another sub-window called the Cluster Tree window lets one navigate and modify the cluster tree. The operations available are those that preserve the $\prec$ relation between graphs.

6. Plug-in example

Tulip has been built to be easily extendable, and here we give an instance of a metric source code. This source program is the entire code needed to add a metric. One only has to compile it as a shared library and put it into Tulip's plug-in directory to make it available. It will be automatically inserted into menu bars, and all others plug-in will have access to it by querying the PropertyContainer for a property having its name (here "Leaf"). This plug-in computes the number of leaves reachable from a node of a tree. Due to the buffer mechanism included in the framework, one only needs to implement the core of the recursive function and the check function.

```
#include <Metric.h>
struct LeafMetric:public Metric {
 //Property context is used to give arguments to the algorithm
 LeafMetric(PropertyContext *context):Metric(context){}
 ~LeafMetric(){}
```

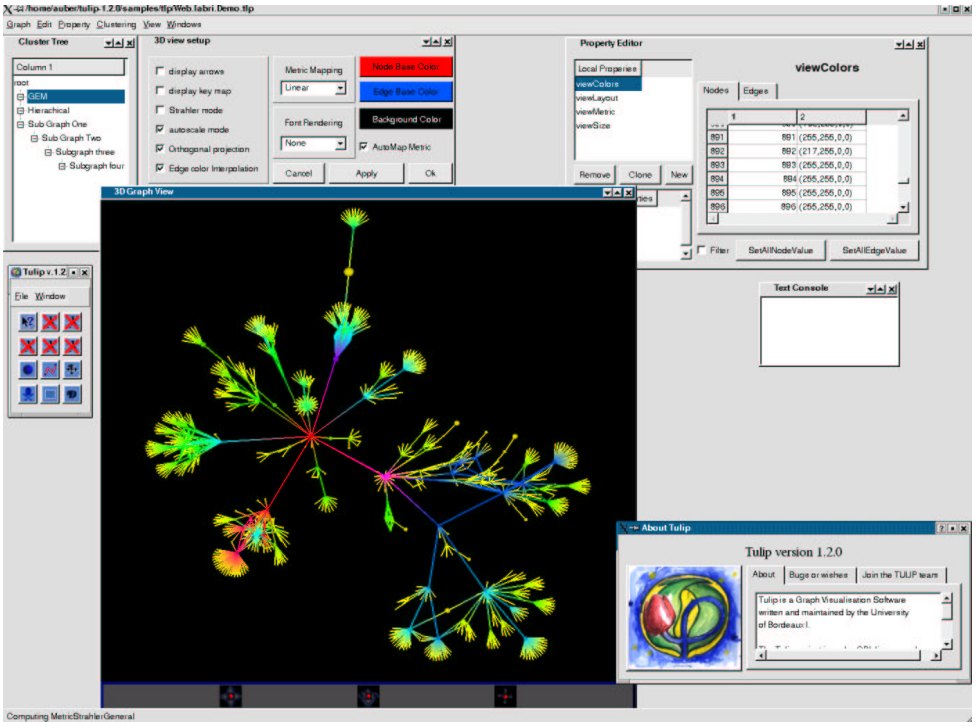

Figure 5: *The Tulip software HCI*

```
double getNodeValue(const node n){
 double result=0;
 //We use iterator pattern to access to graph elements.
 Iterator<node> *itN=superGraph->getOutNodes(n);
 for(;itN->hasNext();) {
  node itn=itN->next();
  //The property that we are computing is recursive,
  //therefore a buffer is used. If the value of itn has not
  //been already set in metricProxy then the proxy do the recursive
  //call, else the proxy return the buffered value.
  result+=metricProxy->getNodeValue(itn);
 }delete itN;
 if (result==0) return 1.0; else return result;
}
//This function is automatically called by the system
//when necessary, it enables to verify that the algorithm
//can be performed on the graph.
bool check(string &erreurMsg){
 if (!superGraph->isTree()) {
  erreurMsg="The Graph must be a tree";
  return false;}
 return true;
 }
};
//We call a macro which automatically builds the necessary
//object for linking the plug-in with Tulip.
METRICPLUGIN(LeafMetric,"Leaf","Author","Date","Ver","0","1")
```

7. Implemented plug-ins

We are already using Tulip for our own research, it permits us to quickly experiment with tools for exploring graph structure. In this section we present briefly some of the implemented plug-ins.

7.1. Metrics Plug-ins

- Tree metrics : Basic tree parameters are implemented in Tulip: max degree, path length, Strahler[8, 21], number of leaves etc. These metrics are useful when visualizing data which can be represented by a tree. For instance, when visualizing a file system tree (we omit symbolic links), the arity max parameter, which gives the measure of the maximum arity of nodes in a subtree, enables to show directories which contain too much (more than others) files or sub-directories even if we are displaying huge file system.
- DAG metrics : Two DAG metrics are available, they are used more for graph drawing algorithm than for visualization. The barycenter one is used to reduce the number of crossing in the layout and the layer assignment one is used to compute y-axis coordinates of layout.
- Graph metrics : Several graph parameters are included in Tulip, for instance decomposition into strong components is a parameter which assigns different values to nodes if they are not in the same strong component. Using it for

web-graph visualization enables one to show up instantaneously which parts of a web site unreachable from an other. Other metrics such as cluster, and fission are also available. Another one called Strahler graph[4] is used to give information about the complexity of the graph near a node.

7.2. Layout Plug-ins

Tree Walker : This plug-in implements the so-called tree walker[20] algorithm. Its principle is to draw separately each subtree of a node, the resulting subtrees are moved on x coordinates until the distance between them is minimal without superposition. The algorithm continues recursively until the whole tree is drawn. See figure 6.

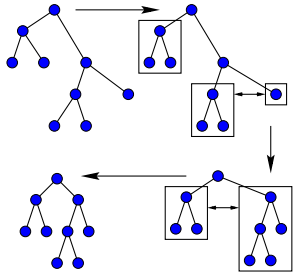

Figure 6: *Tree Walker principle*

Tutte layout : It is an implementation of the so-called Tutte[24] algorithm. The principle of this algorithm is to place the nodes of a face (a cycle in the graph) of the graph on a circle. The others nodes are placed to the barycenter of their neighbors until no nodes move. The advantage of this algorithm is that if the graph is 3-connected planar the drawing is planar (without any edge crossing). See figure7.

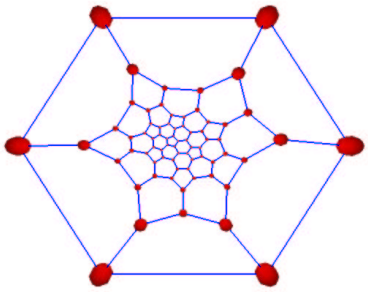

Figure 7: *Tutte Layout of the C_{70} molecule*

Spring Electrical : This is an implementation of a 3D/2D force directed algorithm. It replaces each edge by a spring and puts an electrical repulsion force between unlinked nodes. Then it lets the system go until there is no move or the time limit is elapsed. In practice there is a lot of ways to write such algorithms[9] and the result is often good especially for symmetrical graphs. See figure 8.

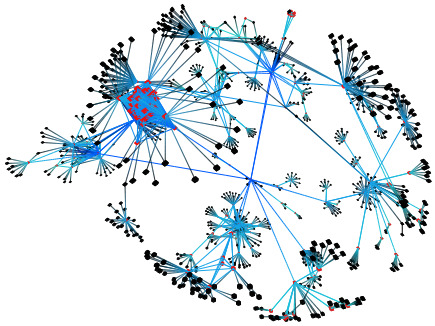

Figure 8: *3D Spring Electrical of our web site*

DAG Layout : This algorithm gives an upward forward drawing($\forall e = (u,v) \in G$ u is over v) of directed acyclic graphs. The principle is to augment the structure of the original DAG to obtain a proper DAG. This augmentation replaces edges by two dummy edges and one dummy node until the layer assignment metric of two linked nodes is equal to one. After that it extracts a spanning tree of this DAG. Then it draws the spanning tree by using a tree algorithm and restores the original DAG by adding bends corresponding to nodes that have been added during the augmentation step. To reduce edge crossings in the drawing the tree map (order of the children of a node) is not chosen randomly. We use the barycenter metric to choose the map. This kind of algorithm are called Sugiyama[22] in the literature. In Tulip we have put a special emphasis on this algorithm in order to run it on huge graphs and to manage different size of elements. See figure 9.

General Graph : This algorithm replaces all self loops of the graph by three dummy edges and two dummy nodes. Then it extracts a spanning dag of this graph and reverses the sense of all the edges out of the spanning dag. Like this it transforms the original graph in an acyclic directed graph, then it calls the Dag Layout algorithm. After it restores the original graph by restoring the edges orientation and replacing dummy nodes added for self loops by bends on the original edges. See figure 10.

7.3. Clustering Plug-ins

- Tree clustering : Three tree clustering algorithms are available. All of them use the statistical properties of tree parameters in order to prune progressively the tree[5]. These

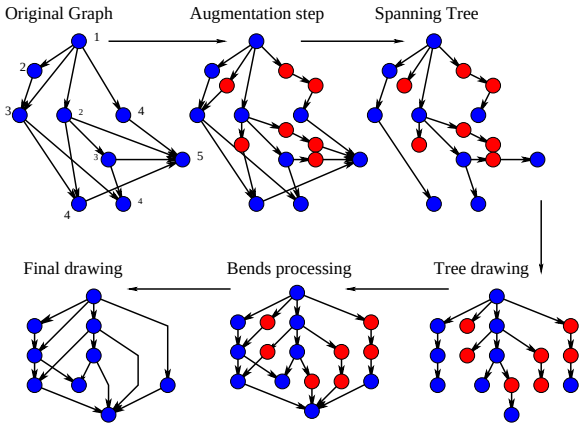

Figure 9: *Dag Layout principle*

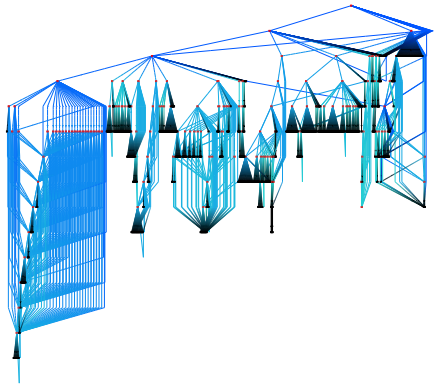

Figure 10: *Hierarchical drawing of our web site*

algorithm has been used in order to study trees coming from a parallel compiler.

- Hierarchical : This algorithm recursively splits in two parts the set of nodes of a graph depending to the value of a Metric.

7.4. Selection Plug-ins

As we said before selection algorithms are useful to enable interaction. Several algorithms are available: induced subgraph selection enables one to automatically select edges between a set of nodes, the reachable subgraph described above, a spanning DAG selection algorithm, a spanning tree selection algorithm. Other selection plug-ins such as the selection of all elements with a metric in a given range are also available and improve the search for information inside data.

8. Comparison and benchmarks

In this section we compare execution time of simple task on Tulip to two different softwares. The first one is AGD[19] which is based on the well-known LEDA library and is one of the most famous graph drawing software. The second one is GraphViz[6] developed by AT&T research center. The benchmark test suite can be download at the following url: www.tulip-software.org. The test has been done on an notebook running at 800MHz, with 256Mo of memory. For the displaying, the OpenGl hardware acceleration was disable in order to limit the difference between the programs.

8.1. Loading

This test consists in measuring loading time of a graph into memory. For each program, we have used the graph format specific to the software. For each graph format, the layout of the graph has been stored inside the file in order to measure only the loading time. The results are summarized in figure 11. For each kind of graph one can see that Tulip is faster than others, especially when the size of graphs become huge.

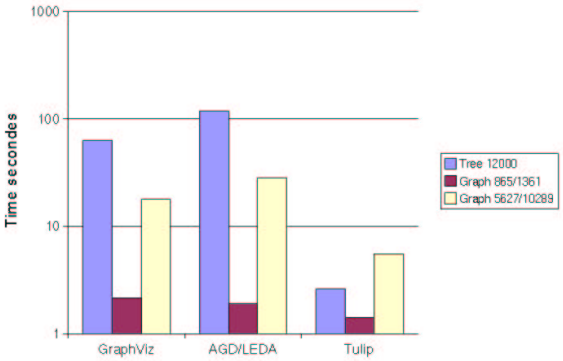

Figure 11: *Loading time of graphs.*

8.2. Displaying

In order to test the displaying time, we have chosen to measure the time between two redisplays after the window resizing. The results are summarized in figure 12.

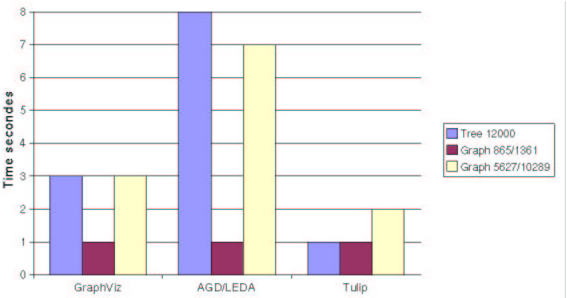

Figure 12: *Displaying time of graphs.*

8.3. Layout

Comparison of layout time is a more difficult task. The reason is that graph drawing algorithms use heuristics in order to solve NP complete problems such as crossing number minimization. For all the following results we have used the fastest method proposed in each program in order to lay out the graph. For tree drawing, the tree walker algorithm has been used in AGD and Tulip; the drawings are similar, thus the algorithm implementation should be the same. For graphs, we have used the Sugyiama layout of AGD and GraphViz with the fastest options, and the hierarchical drawing of Tulip which is a Sugyiama algorithm optimized for huge graphs. The results are summarizes in figure 13. For each kind of graph Tulip is faster than the others. However, for graph layout one must take into account the quality of the drawing which seems better on other programs, but during the visualization process user interaction is the most important thing and we cannot wait one hour before the displaying of the first picture.

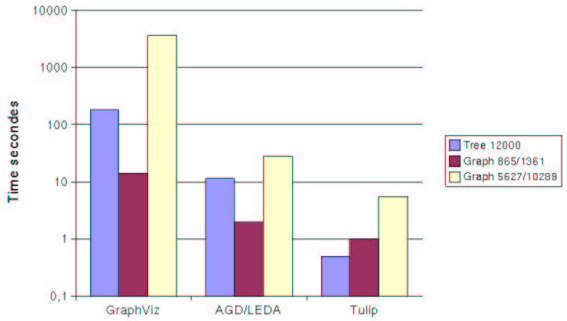

Figure 13: *Layout computing time of graphs.*

9. Conclusion

We have presented Tulip, a graph visualization program that enables researchers to experiment with new tools easily and obtains good speed results and memory usage. It makes it possible to implement graph layout algorithms as well as graph metric algorithms and can be extended to treat any kind of graph properties. It also permits one to visualize graphs in two or three dimension and to modify it, and includes graph clustering function. This program is open source and free for use under Linux, and only uses libraries that follow the same rules.

Future work on Tulip will be to improve the tools library. We are already planning to add graph drawing algorithms based on topological maps such as the so-called mixed-model[11] algorithm. These algorithms will be used for the visualization of small graphs obtained with clustering methods. We are also working on graph clustering algorithms based statistical graph structure and on a general way to integrate view tools such as the fish eye in the view modules.

Tulip software can be downloaded at this URL : *"www.tulip – software.org"*

References

1. Mesa 3-d graphics library. `http://www.Mesa3D.org`.
2. S. Bridgeman, A. Garg, and R. Tamassia. A graph drawing and translation service on the WWW. *Lecture Notes in Computer Science*, 1190:45–52, 1997.
3. C. Huizing J.J. van Wijk. Bruls, D.M. Squarified treemaps. In *Data Visualization 2000*, proceedings of the joint Eurographics and IEEE TCVG Symposium on Visualization, pages 33–42. Springer, 2000.
4. D.Auber. Using strahler numbers for real time navigation in huge graphs. Technical report, LaBRI, Department of Computer Science, Bordeaux University, 2002.
5. D.Auber and M.Delest. Web tree clustering. Technical report, LaBRI, Department of Computer Science, Bordeaux University, 2001.
6. John Ellson, Emden Gansner, Eleftherios Koutsofios, and Stephen North. Graphviz. Available in the World Wide Web at http://www.research.att.com/sw/tools/graphviz.
7. Richard Helm Erich Gamma and al. *Design Patterns*. International Thomson, january 1995.
8. J.M Fedou. Nombre de strahler sur les arbres généraux, école jeunes chercheur en algorithmique et calcul formel, may 1999.
9. Arne K. Frick, H. Mehldau, and A. Ludwig. A fast adaptive layout algorithm for undirected graphs. In Roberto Tamassia and Ioannis G. Tollis, editors, *Proc. DIMACS Int. Work. Graph Drawing, GD*, number 894, pages 388–403, Berlin, Germany, 10–12 1994. Springer-Verlag.
10. Roberto Tamassia Giuseppe Di Battista, Peter Eades and Ioannis Tollis. Annotated bibliography on graph drawing algorithms. *Computational Geometry: Theory and Applications*, 4:235–282, 1994.
11. C. Gutwenger and P. Mutzel. Planar polyline drawings with good angular resolution. *Lecture Notes in Computer Science*, 1547:167–182, 1998.
12. Herman, Melançon, de Ruiter, and Delest. Latour – A tree visualisation system. In *GDRAWING: Conference on Graph Drawing (GD)*, 1999.
13. I. Herman, M. Marshall, and G. Melançon. Density functions for visual attributes and effective partitioning in graph visualization, 2000.
14. Mao Lin Huang and P. Eades. A fully animated interactive system for clustering and navigating huge graphs.

Lecture Notes in Computer Science, 1547:374–383, 1998.

15. G. Battista. P. Eades. R. Tamassia. Ioannis and G. Tollis. *GRAPH DRAWING, Algorithms for the Visualization of Graphs*. Number 1. 1999.

16. Dorothea Wagner. Michael Kaufmann. *Drawing Graphs, Methods and Models.* january 2001.

17. Tamara Munzner. Drawing large graphs with h3viewer and site manager. In *Proc. 5th Int. Symp. Graph Drawing, GD*, Lecture Notes in Computer Science, LNCS, pages 384–393. Springer-Verlag, 1998.

18. Maurizio Patrignani and Francesco Vargiu. 3DCube: A tool for three dimensional graph drawing. In Giuseppe Di Battista, editor, *Proc. 5th Int. Symp. Graph Drawing, GD*, number 1353 in Lecture Notes in Computer Science, LNCS, pages 284–290. Springer-Verlag, 18–20 September 1997.

19. C.Gutwenger P.Mutzel and al. A library of algoritms for graph drawing. *LNCS, Graph Drawing*, 1547(13):456–457, 1998.

20. Edward M. Reingold and John S. Tilford. Tidier drawings of trees. *IEEE Transactions on Software Engineering*, 7(2):223–228, March 1981.

21. A. N. Strahler. Hypsomic analysis of erosional topography. *Bulletin Geological Society of America 63 1117-1142.*, 1952.

22. Kozo Sugiyama, Shojiro Tagawa, and Mitsuhiko Toda. Methods for visual understanding of hierarchical system structures. *IEEE Trans. Systems, Man and Cybernetics*, SMC-11(2):109–125, 1981.

23. Trolltech. Qt the crossplatform c++ gui framework. `http://www.trolltech.com/`.

24. W.T. Tutte. How to draw a graph. *Proc. London Math. Soc.*, 3(13):743–768, 1963.

25. David Ungar, Craig Chambers, Bay-Wei Chang, and Urs Hölzle. Organizing programs without classes. 4(3):223–242, July 1991.

26. Graham J. Wills. NicheWorks — interactive visualization of very large graphs. In *Proc. Graph Drawing*, LNCS. Springer-Verlag, 1997.