Latour — a Tree Visualisation System

Ivan Herman[1], Guy Melançon[1], Maurice M de Ruiter[1], and Maylis Delest[2]

[1] Centrum voor Wiskunde en Informatica, P.O. Box 94079
1090 GB Amsterdam, The Netherlands
{Ivan.Herman,Guy.Melancon, Behr.de.Ruiter}@cwi.nl
[2] LaBRI, Université Bordeaux I
351, cours de la Libération, 33405 Talence Cedex, France
Maylis.Delest@labri.u-bordeaux.fr

Abstract. This paper presents some of the most important features of a tree visualisation system called Latour, developed for the purposes of information visualisation. This system includes a number of interesting and unique characteristics, for example the provision for visual cues based on complexity metrics on graphs, which represent general principles that, in our view, graph based information visualisation systems should generally offer.

1 Introduction

Information visualisation is one of the relatively new areas of research and development in computer science; its fundamental goal, i.e., the ability to visualise and to navigate in large, abstract datastructures, is often regarded as one of the crucial tasks in bringing computers closer to the general public [2]. Visualising graphs plays a very special role in this area, because they can often be used to visualise abstract datastructures. Practical examples include hypermedia structures (like the Web), database query results, or organisational charts of companies. Systems to visualise large graphs have come to the fore in the last years; the NicheWorks system of Wills [17], the fsviz system of Carrière and Kazman [3], or daVinci of the University of Bremen [6] are just some typical examples. These systems usually draw on the rich research heritage in the graph drawing community which, over the years, has explored some of the mathematical problems related to graph drawings [1]. Putting these research results into practice is not a simple task, however. Practical issues raised by, for example, the large size of graphs in information visualisation, the need for navigation and interaction, user interface and ergonomic issues, etc., create new challenges, or cast a new light on well–accepted practices [13]. Consequently, none of the current graph drawing systems could claim to be complete; experiences with these systems are still to be gathered to gain a better understanding of the kind of drawing and navigation facilities which are necessary for a really successful system.

The goal of this paper is to contribute to this “gathering”. It describes an application framework called Latour, whose goal is to incorporate interactive graph (primarily tree) visualisation and navigation techniques into other applications. At present, Latour is used or tested as a toolkit to visualise, and to interact with, various application data, for example internal data structures of programs, deployment results of large Petri nets, evolution of genetic algorithms, etc.

While developing this framework, some of the practical problems required more concentrated research efforts, which also led to interesting and general results [8, 9, 12]. The goal of this paper is to describe a number of issues which, albeit not deserving separate articles by themselves, together constitute a body of experiences which we felt is worth sharing with the R&D community. A technical report available online [10] contains further details which, because of lack of place, could not be included in the present paper.

2 Graph/Tree layout

In spite of all the results on graph drawing [1], it is not simple to choose a specific algorithm for information visualisation. Information visualisation, which is inherently interactive, raises a number of issues that are not necessarily covered by the classical research. Apart from obvious problems such as speed (in the case of a graph with 3–4000 nodes, the display of the graph should not take more than a second), there remain two important aspects:

– Predictability. Two different runs of the algorithm, involving the same or similar graphs, should not lead to radically different visual representation. This is very important if the graph is interactively changed, for example by (temporarily) hiding some nodes or making them visible again. Great care should be taken on which layout algorithm is chosen. For example, a number of graph layout algorithms use optimisation techniques; if the graph changes, a new local minimum may lead to a dramatically different visual representation, which is unacceptable for interactive use. (The term "preserving the mental model" is also used to describe this requirement, see [11].)
– Navigation on large or unusual graphs. Practical applications lead to thousands, or possibly tens of thousands of nodes. To cope with such numbers, navigation tools, search facilities, hierarchical views, etc., are necessary. The implementation of such tools may also require the usage of suboptimal layout algorithms.

The bulk of the Latour system concentrates on trees, where the usual layout algorithms are quite predictable and fast. It was not the goal of Latour to develop new layout algorithms; instead, the goal was to concentrate on the issues raised by data exploration and interaction. Three different tree layout algorithms have been implemented. Various user communities have their own traditions, habits, or requirements, and an application framework cannot impose one single view on its users. In what follows, a short overview of these views will be given.

2.1 Hierarchical view

The hierarchical view of the tree is based on the well–known algorithm of Reingold and Tilford [14] revisited by Walker [16]. The layout algorithm is simple, fast, and completely predictable. Various variants exist: grid–based, left–to–right or top–down, etc. All these variations are mathematically identical and implementers may be tempted to include arbitrarily one of these variations only. This would be a mistake: one should recognise that the way of looking at trees may depend on the application areas. For example, the top–down grid view is the widespread way of looking at family trees, whereas biological evolution schemas often use a left–to–right grid. The conclusion is simple, albeit important: give the user the choice; he/she should be able to choose among the different views.

2.2 Radial view

The radial view is based on an algorithm described in Eades [5] (see also [1]). This algorithm recursively places the children of a subtree into circular wedges; the central angle of these wedges are proportional to the width of the respective subtrees, i.e., the number of leaves. If this was the only layout rule, additional edge intersections would occur if the angle on the node became too large; to avoid this, a "convexity constraint" is introduced which, essentially, forces the wedge to remain convex. This type of view is favoured, for example, by some web site viewers, which do not want to overemphasise the role of a root.

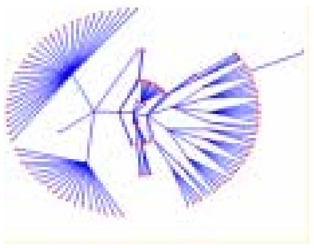
Fig. 1: Radial view without convexity check

The algorithm is very simple, but it is not optimal in using the available space. We spent some time in trying to optimise the algorithm. The idea was to use the statistical distribution of the width of a subtree at a node, which can be approximated with a normal distribution (see [4]). The improvements were not significant, however; this turned out to be the consequence of the convexity constraint whose effect seems to dominate other optimisation attempts. A possibility to overcome this problem is to simply drop the convexity check. Although this is not mathematically correct, the occurrence of extra intersections is not very frequent after all. It is not necessary to look for a mathematically perfect algorithm for a graph layout; the mathematical "faults" may not be significant in practice. Problems with navigation, zooming, etc. (see the next section) should become predominant in that case, and it is not really worth to optimise the layout any further. For the sake of completeness, we decided to include both the optimal (i.e., with convexity check) and the, shall we say, sub–optimal radial layout algorithm into Latour. See [10] for a comparison of both layouts including figures.

2.3 Balloon view

The request for a "balloon" view (see Fig. 2) came from an application dealing with the retrieval of keywords and their relations from a database. The notion of a "root" is temporary for such application: the user should be able to move from one node to the other interactively, and the tree on the screen should reflect the relationships using this temporary focus. The balloon view seems to fulfil this need. The detailed explanation of the algorithm would go beyond the scope of this paper [12]. Other placement algorithms could also be used [3].

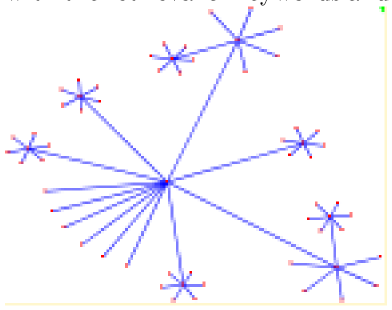
Fig. 2: Balloon view

3 Interaction and Navigation

Information visualisation is an inherently interactive application; the user has to move around in information space, explore details, hide unnecessary parts of a tree, etc. Obviously, a good system must offer a whole range of tools in order to make the exploration of a graph easy, or indeed possible.

3.1 Zoom, pan, fish–eye

Some of the techniques, implemented in Latour are now standard: zoom, pan, and geometric fish–eye [15]. As much as possible, the factors controlling these effects (e.g., the distortion factor of the fish–eye view) are settable interactively by the end–user.

The fish–eye view has one drawback, though, which implementers should be aware of. The essence of a fish–eye view is to distort the position of each node, using a concave function applied on the distance between the focal point and the node's position. If the distortion were to be applied faithfully, the edges connecting the nodes should be distorted into general curves. Usual graphics systems do not offer the necessary facilities to draw these curves easily. The implementer's only choice is to approximate these curves with dense polylines. This leads to a prohibitively large amount of calculation and makes the responsiveness of the system sink to an unacceptably low level. The only viable solution is to apply the fish–eye distortion on the nodes only, and to connect them by straight–line edges. The consequence of this inexact solution is that new edge intersections might occur. Though inelegant, this brute force approach did not prove to be disturbing in practice.

3.2 Complexity visual cues

The well–known problem in using zoom and pan is that the user looses the "context". This is why fish–eye view is used: it provides a "focus+context" view of the tree. However, when the tree is large, zoom and pan cannot be avoided and other techniques become necessary, too. A unique feature of Latour is a technique to provide visual cues based on the structural complexity of the tree. This technique works as follows.

A metric value is calculated for each node of the tree. This metric should represent the complexity of the subtree stemming at the node. Several different metric functions are possible and, ultimately, the choice among these should be application dependent. Examples for such metric include the width of the subtree (i.e., the number of leaves), the sum of the lengths of all paths between the node and the leaves of the subtree, the so–called Strahler numbers [8], or the "degree of interest" function used by Furnas [7]. Using these metric values, and visual tools like colour saturation, linewidth, etc, Latour can highlight the "backbone" of a tree, i.e., those edges which hold larger, more complex subtrees. The effect is clearly visible on Fig. 3. Without the backbone the user would barely know where to move with the pan, if complex areas are searched. The backbone on the figure clearly shows, for example, that one of the edges going toward the left leads to a complex portion of the tree, whereas the other ones are probably less interesting.

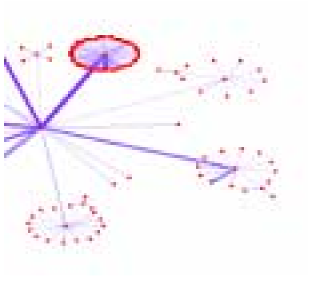
Fig. 3: Visual Cues

Another possible usage of the metric numbers is presented on Fig. 4: this is the so–called schematic view of a tree. Based on the complexity metrics of the nodes, Latour displays only those nodes whose metric value is greater than a specific cut–off, yielding what we have called the skeleton of the tree. All other nodes are encapsulated in triangular shapes, whose size and geometry is proportional to the hidden portion of the tree. The result is a better overall view of the tree which, combined with other navigation techniques, provides a powerful interactive tool. It is worth noting that, although all our examples so far were for trees, the visual cue techniques based on a complexity metric represent a general principle which can be applied for more general graphs, too; the interested reader should refer to [9].

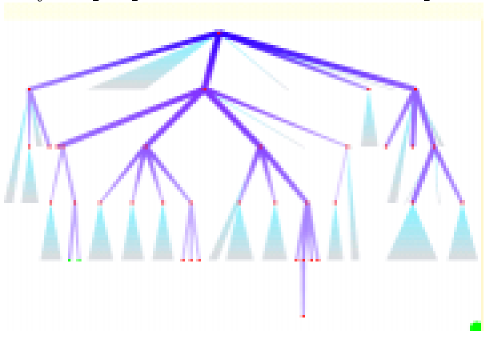
Fig. 4: Schematic view of a tree

3.3 Animation

Latour is an interactive system; the user navigates in different portions of the tree, zooms, pans, etc. Some of these actions result in an immediate, real–time feedback (for example, zoom), some other actions may lead to a more radical reorganisation of the screen (for example, folding a subtree into a node, or unfolding a folded subtree). Latour animates all possible changes from one view to the other, avoiding any radical changes as far as possible. Although originally only included in Latour to reduce possible ergonomic problems, this basic animation feature turned out to be a very useful tool for various applications exploring a sequence of trees, instead of a single one. This is the case, for example, of the application exploring genetic algorithms, or the traces of parallel program runs. Therefore, the input possibilities of Latour have been extended: it can not only accept the description of a single tree, but also a "generation" of trees, i.e., a basic tree plus a sequence of difference trees. This sequence of trees can then be visualised systematically with again a graceful animation at each change.

4 Beyond trees

Latour is primarily a tree visualisation tool but, obviously, applications may want to handle more general structures, too. We have added some extensions to Latour which are worth mentioning here.

4.1 Packed forests

Packed forests are special data structures. The need for these data structures has arisen through an application concerned with the visualisation of the internal data structures of compilers, but has proven to be useful in general, too.

Instead of giving an abstract definition, the concept is presented through an example. For a compiler, the standard internal representation of a string is a list. The leaves of the list represent the individual characters of the string, and intermediate nodes are used to build up a list structure. Such list can be represented as a simple tree, like the left–hand one in Fig. 5. However, such a representation may be too "verbose". An expert in compiler technology knows the internal representation for a string and does not necessarily need the full list version of the relevant portion of the graph; the tree on the right–hand side of Fig. 5 is enough to convey all the necessary information. What the user wants is to be able to interactively "switch" between different representations. Latour has the possibility to store, internally, a set of such alternatives for each node, and offers interactive means to switch among those.

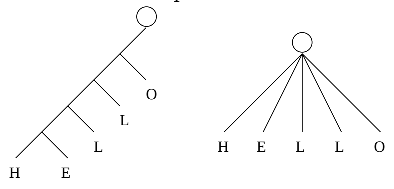

Fig. 5: A packed forest

Packed forests turned out to be extremely useful in practice. As a slightly extreme example, some of the demonstration graphs used by our compiler builder partner is, initially, a tree consisting of 2–3 nodes only. However, when the same graph has all its most complex alternatives extended, it turns into a tree of about

100 nodes. Similar data structures are used routinely in computational linguistics; the concept of "level of details", of an utmost importance in virtual reality scenes, is another example which can be represented through these structures. Packed forests provide a very efficient way of imposing a manageable hierarchy on the visualised data structures.

4.2 Dag's

Dag's (Directed Acyclic Graphs) represent the next logical step when trying to generalise from trees. This is achieved by a simple extension of Latour, which allows the storage of additional links for each node of the underlying tree. This means that a spanning tree is provided, and Latour uses its tree–related structure to visualise the dag by simply adding the additional links to the tree picture. The spanning tree may have two origins: either the application generates it, or the spanning is tree is calculated for the dag.

Requesting the application to generate a spanning tree is not such a strong requirement. A number of applications have an inherent tree structure in the data, and visualising this tree, with the additional edges added to the tree, yields a natural representation of the dag.

Fig. 6 shows an example where a spanning tree is used to visualise a dag. The interesting feature in this case is that the spanning tree consists of three branches and most of the "non–tree" edges are used to connect nodes in different branches. We can refer to such graphs as "multipartite" trees. Similar, but bipartite trees occur when describing virtual reality scenes, for example (where one branch describe an object hierarchy, the other the real instances). These "multipartite" graphs occur frequently in applications, and constitute a set of examples where the simple extension of Latour works out very well in practice. Automatic generation of spanning trees raises a number of issues not developed here. For further details see [10].

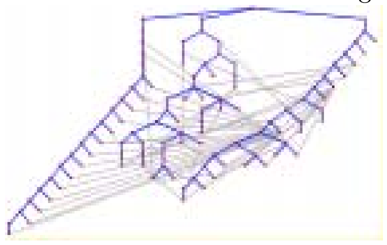

Fig. 6: A tree with added links

5 Conclusions

The implementation of Latour has resulted in a very flexible system, which is well adaptable to various user communities. It concentrates on interaction and visual feed–back, rather than complicated layout algorithms, which makes it one of its strengths. It has also taught us some important lessons: that a proper balance has to be found between the mathematical correctness and the requirements of navigation and interaction, that the end–user has to have a maximal control over the appearance and the attributes of the visual representation, we learned about the importance of metric functions on graphs in general. In developing a more general graph–based information visualisation framework these experiences will become of an utmost importance.

References

1. Battista, G. di, Eades, P., Tamassia, R., Tollis, I.G.: *Graph drawing: algorithms for the visualisation of graphs.* Prentice Hall (1999).
2. Card, S.K., Mackinlay, J.D., Shneiderman, B. (eds): *Readings in Information Visualization.* Morgan Kaufmann Publishers (1999).
3. J. Carrière, J., Kazman, R.: Interacting with huge trees: beyond cone trees. In: *Proceedings IEEE Information Visualisation '95*, IEEE CS Press (1995), 74–81.
4. Drmota, M.: Systems of functional equations. In: *J. Random Structures and Algorithms*, **10**(1–2), (1997), 103–124.
5. Eades, P.: Drawing free trees. In: *Bulletin of the Institute for Combinatorics and its Applications*, **5**, (1992), 10–36.
6. Fröhlich, M., Werner, M.: Demonstration of the interactive graph visualization system daVinci. In: *Proceedings of DIMACS Workshop on Graph Drawing '94, Springer–Verlag*, (1995).
7. Furnas, G.W.: Generalized fisheye views. In: *Human Factors in Computing Systems, CHI'95 Conference Proceedings*, ACM Press (1995), 16–23.
8. Herman, I., Delest, M., Melançon, G.: Tree visualisation and navigation clues for information visualisation. In: *Computer Graphics Forum*, **17**(2), (1998), 153–165.
9. Herman, I., Marshall, S.M., Melançon, G., Duke, D.J., Delest, M., Domenger, J.–P.: Skeletal images as visual cues for graph visualisation. In: *Data Visualization '99, Proceedings of the Joint Eurographics IEEE TCVG Symposium on Visualization*, Springer–Verlag, (1999), 13–22.
10. Herman, I., Melançon, G., Ruiter, M.M. de, Delest, M.: *Latour — a tree visualisation system.* Reports of the Centre for Mathematics and Computer Sciences (CWI), INS–R9904, `http://www.cwi.nl/InfoVisu/papers/LatourOverview.pdf`, (1999).
11. Misue, K.,Eades, P., Lai W., Sugiyama, K.: Layout adjustment and the mental map. In: *Journal of Visual Languages and Computing*, **6**, (1995), 183–210.
12. Melançon, G., Herman, I.: *Circular drawing of rooted trees.* Reports of the Centre for Mathematics and Computer Sciences (CWI), INS–R9817, `http://www.cwi.nl/InfoVisu/papers/circular.pdf`, (1998).
13. Purchase, H.: Which Aesthetic has the Greatest Effect on Human Understanding? In: *Proceedings of the Symposium on Graph Drawing GD'97*, Springer–Verlag (1998), 248–261.
14. Reingold, E.M., Tilford, J.S.: Tidier drawing of trees. In: *IEEE Transactions on Software Engineering*, **SE–7**(2), (1981), 223–228.
15. Sarkar, M., Brown, M.H.: Graphical fisheye views. In: *Communication of the ACM*, **37**(12), (1994), 73–84.
16. Walker II, J.Q.: A node–positioning algorithm for general trees. In: *Software — Practice and Experience*, 20(7), (1990), 685–705.
17. Wills, G.J.: Niche Works — interactive visualization of very large graphs. In: *Proceedings of the Symposium on Graph Drawing GD'97*, Springer–Verlag (1998), 403–415.