Using Strahler numbers for real time visual exploration of huge graphs

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

March 8, 2003

Abstract

This paper studies the problem of real time navigation in huge graphs. When size of data is becoming too large, computers are not enough powerful to enable interactive navigation without loosing the relevant part of the graph. Here, we present a method to solve this problem. This solution is based on combinatorial properties of graphs. We first introduce the reader to our generalization to rooted maps of the so-called Strahler number[22]. Subsequently we present a way to use this parameter in order to display relevant part of the graph during the navigation. Finally we give experimental results of our method.

keywords: Information visualization, clustering, interaction, navigation

1 Introduction

Visualization of graphs has came to the fore during the last ten years. The huge amount of linked data on the Web as well as genome's research are some of the factors of this research domain evolution. One can refer to Munzer [18] for use of graphs in web analysis and to Robinson [20] or Guelzim[10] for their use in bioinformatics.

In this paper we focus on an efficient way to enable visual exploration of huge graphs. Visual data exploration is a very general concept, it includes: usual movements in two and three dimensional space such as zoom, rotate and scale, space distortion used in focus and context methods such as the fisheye view[21] and also coloration[11] of the graph's elements which can be used to highlight relevant part of the data. Thus visual exploration can be summarize to a stream of visual representations of data transmitted through the screen to the final end user.

To preserve the visual mental map of the final end user the transitions between the initial and the final frame must each preserve sufficient context that the users perceptual processes can track the movement within the virtual space. For instance, in the figure 1 one can see the entire structure of a small tree. By zooming, the user can go until the elementary data that we can see on the figure 2. If we zoom directly from the figure 1 to 2 we are completely lost in the data. But if we zoom progressively until the figure 2 user can keep in mind his/her position in the data.

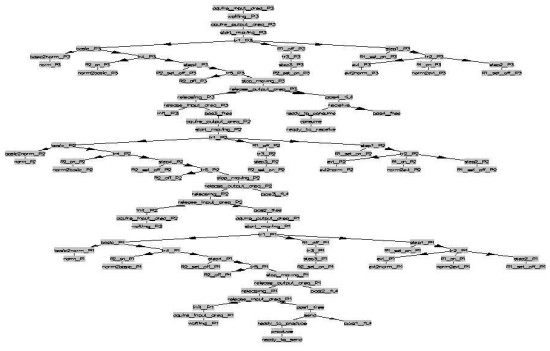

Figure 1: The full structure

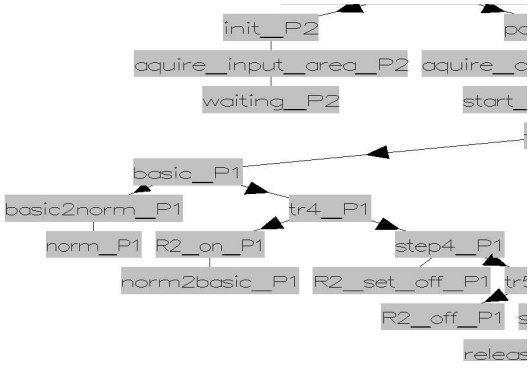

Figure 2: Some elementary data

The idea of making all change progressively can be generalized to the entire visualization system. The focus of a fisheye should move progressively to a new position. The layout of a graph should morph into a new layout as proposed by Carsten[9] or by Yee[27], in this case it enables the user to learn faster the new map of the data. Colors or shapes of elements can also follow the same rules. Thus by changing the data mapped on the size of elements one can see which elements are varying.

Viewing progressively modification of the picture presented to the user implies to display a large number of image to the user. Research in human perception has shown that the maximum delay between an action of the user and the displayed result on screen should be less than fifty milliseconds if one wants the user to believe in a causal link. One can refer to Ware[24] book on information visualization to obtain more detail on this subject. Thus we need a real time system which ensures that the displaying of a picture will take less than fifty milliseconds.

In the following we describe the incremental rendering method proposed by Graham[26] then we introduce the reader to the using of graphs parameter in order to enhance the result produce by this method. Subsequently, we introduce and study the computation complexity of the so called Strahler number[22] and of one of its generalization[8]. Then, we present an extension of this parameter to pointed map that can be used for graph visualization. Finally we conclude with experimental results of our method.

2 Incremental rendering

When visualizing huge graphs drawing of all elements on the screen can take more than fifty milliseconds. Therefore, to manage efficiently user or system interaction a visualization software must enable to receive new events even during the rendering of the visual representation of data. To solve this problem the method proposed by Graham[26] is to predict how many items can be drawn during 50 ms.Then, the system draw items during 50 ms before to check events. If an event is received, after executing associated actions, we restart the displaying from the early start. By this way, the system draw incrementally slices of the graph and ensure interaction in less than 50 ms. One of the advantage of this method is to enable graph rendering without expensive time checks.

Using this method in visualizing graphs representing file system or chemical reactions we have noticed that in case of huge graphs the rendering process could take more than 2 seconds and then we must split it in 40 parts. Thus, when user explores the graph and sends actions to the system at high frequency the graph rendering process never end. For instance, if the user move the focus of a fisheyes every fifty milliseconds, the system will only display a part of the elements of the graph. One of the problem in this case is that the displayed part of the graph must always be the same to avoid the image to shake during user interaction.

During experimentation on semantic network graph visualization[15], we have noticed that choosing randomly the set of elements to display first can produce quite bad results. For instance, when visualizing huge tree such as the one presented in the figure 13, if we can display only five thousand elements in fifty milliseconds , according to our choice, the picture presented to the user during animation can be completly different of the final one, and may actually display elements that the user is not interested in.

Therefore, when using incremental rendering, one must find efficient way to order the set of edges and the set of nodes. Several possibilities are available for solving this problem. One of them is to use the layout of the graph, by this method we can obtain result which is quite similar to the original one. However such a method becomes very hard to set up when one wants to use 3D visualization, space deformation, or layout morphing. One of the major reason is that the order depends of the graph drawing and thus each time the node's or edge's coordinates change one must recompute the order.

Using extrinsic parameter can also be an efficient way to produce an order. For instance, when visualizing web graph[6, 14] we can use the number of hits obtained by each page and each link to build the order. Such a method can give results in term of information visualization because it enables to take into account the user center of interest. For instance, in case of web graph, if the user wanted to see pages which are the most visited this solution seems to be appropriate. However, if the user gooal is to find pages which are difficult to access it doesn't seem to be the best approach.

Another way to treat the problem is to work on the graph structure. Such an approach is widely used in graph clustering algorithms [12, 3, 1]. Here, at the opposite of the method introduced above, we use intrinsic parameters to compute the ordering. Thus one must find graph parameters which provide an efficient ordering of elements. Simple parameter such as degree of nodes (number of edges connected to a node) enable to compute easily the needed order. In practice, we observe that the degree of nodes is bounded by a constant. For instance, trees coming from parallel compiler have a degree bounded by the number of processors. Then, when size of such trees is huge, it becomes equivalent to the using of a random order. Others parameters such as the activation metric proposed by Marshall[17] or the clustering measure[25] can produce more useful results, however, for both of them, they are not efficient for trees and directed acyclic graphs, and their complexity is $O(n^2)$ that means that they don't scale well in relation with data scale or that efficient using requires parallelization.

Our research focuses on a parameter called Strahler. This parameter enables to catch structural properties of graphs that is, in case of graph visualization, the information that we try to find. Furthermore its complexity enables to use it on huge graphs without using of expensive hardware that is one of our objective. One other interesting features of this parameter is that by thresholding it, we can retain an overall impression of the geometric structure of the graph which is useful for supporting awareness of spatial position while manipulating the graph.

3 Strahler numbers

We introduce here the first version of the Strahler number [22] which is defined on binary trees, subsequently we present a first extension proposed by Fedou[8] that generalizes this parameter to general trees and to directed acyclic graphs. In the following, the theorems and properties of these parameters are briefly given and proven in order to show that the algorithm complexity is $O(n)$ or $O(n\log(n))$ and therefore will scale better than parameters like spreading activation.

3.1 Strahler number on binary trees

The Strahler number on binary trees has been introduced in some work about the morphological structure of river networks[22]. It consists in associating an integer value to each node of a binary tree. These values give a quantitative information about the complexity of each sub-tree of the original tree[23]. Furthermore, if we consider an arithmetical expression A and its evaluation binary tree T Ershov[7] has proved that the Strahler number of T increased by one is exactly the minimal number of registers needed to compute A. Computation of the Strahler numbers is given by algorithm 1; figure 3 shows an example of the result.

Theorem 1 *The complexity of the algorithm 1 is linear.*

Proof 1 *The algorithm 1 can be resume to a depth first search algorithm. So it is linear.*

Theorem 2 *Let T be a tree having n nodes, an upper bound of the Strahler number value is $\lceil \log_2(n) \rceil$.*

Proof 2 *The Strahler value of a node is greater than its children if and only if its children have the same value. So, one can deduce that the maximal value is reached when the tree is well balanced. In this case an upper bound of the tree depth is $\lceil \log_2(n) \rceil$. The Strahler number can increase at most of one between two layers of a binary tree, thus an upper bound of its value is $\lceil \log_2(n) \rceil$.*

Lemma 1 *The maximal number of different values with the Strahler valuation on a tree T is bounded by $\lceil \log_2(n) \rceil$, where n is the number of nodes of T.*

Proof 3 *Direct from the theorem 2.*

Algorithm 1 Strahler algorithm.

```
binaryStrahler(node η of a binary tree T)
begin
  if η is a leaf of T return 1
  let η_left and η_right be respectively the left and the right child of η
  if binaryStrahler(η_left) is equal to binaryStrahler(η_right)
    return binaryStrahler(η_left) + 1
  else
    return max(binaryStrahler(η_left),binaryStrahler(η_right))
end binaryStrahler
```

3.2 Strahler number on general trees

Jean-Marc Fedou[8] has proposed an extension of the Strahler numbers to general trees. The idea is to consider n-ary operators instead of binary operators. Under this hypothesis, the Strahler number is defined as the minimal number of registers necessary to compute an n-ary expression. Knowing that any general trees can be viewed as an n-ary expression, the extended Strahler number is defined on all trees, we denote it by S_{ext}. The algorithm 2 summarizes the computation of this parameter and the figure 4 shows an instance of its valuation on a general tree.

Note that, in case of a binary tree, the Extended-Strahler number gives the same valuation as the original Strahler number.

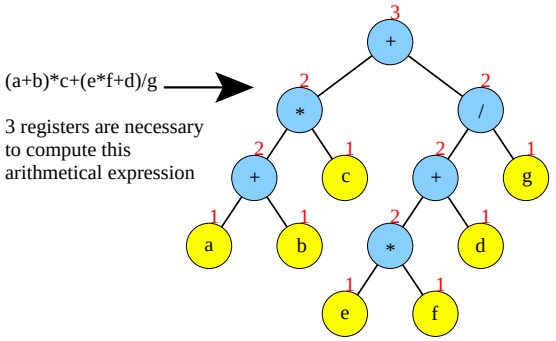

Figure 3: Arithmetical expression tree.

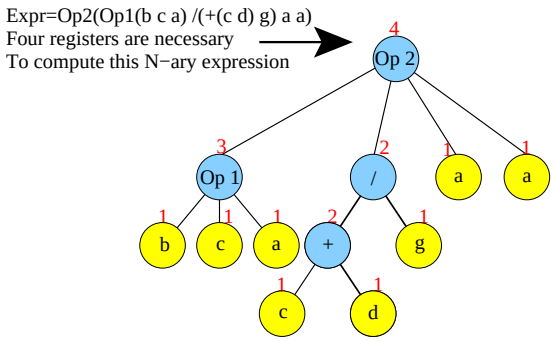

Figure 4: N-ary expression tree.

Theorem 3 *Let T be a tree having n nodes, for all node η of T, $S_{ext}(\eta) \leq n$.*

Proof 4 *By construction, it is easy to prove that it is true for trees which contains one to three nodes. Suppose that it is true for trees until n nodes. Let us build a tree T with $n+1$ nodes. The proposed algorithm ensures that the value of a node is less or equal to the sum of the values of its children plus one. Then, knowing that each subtree of T has less than $n+1$ nodes we can write that the sum of children's values of the root of T is less or equal than n. So, the maximal value of the root of T is $n+1$.*

Theorem 4 *Let T be a tree having n nodes, an upper bound of the complexity of the given algorithm is $O(n\log(n))$.*

Proof 5 *The proposed algorithm sorts all the children of a node in order to compute its valuation. Then it makes several operations which are linear in the number of children. Thus, the computation complexity C_η of a node η is bounded by the complexity of a sort which is in the general case $O(n\log(n))$.*
Let $deg^+(\eta)$ be the out-degree of a node η(ie, number of children) we have:

$$C_\eta \sim deg^+(\eta)\log(deg^+(\eta))$$

If we consider the valuation of all nodes, the complexity is :

$$C \sim \sum_{\eta\in T} deg^+(\eta)\log(deg^+(\eta))$$

Let $deg^+_{max}(T)$ be the maximal out-degree of the nodes of the tree T. We have :

$$C \sim \log(deg^+_{max}(T))\sum_{\eta\in T} deg^+(\eta)$$

Furthermore, we have:

$$\sum_{\eta\in T} deg^+(\eta) = n-1 \; and \; deg^+_{max}(T) < n$$

So, we can deduce that an upper bound of the algorithm is $O(n\log(n))$.

Theorem 5 *Let T be a tree having n nodes, an upper bound of the number of different values of the Strahler number on a T is $\lceil\sqrt{n}\rceil * \sqrt{2}$.*

Proof 6 *The proof of this theorem use a lot of combinatorial constructions and thus will be describe in an other paper.*

Theorem 6 *Under the therorem 1, the extended-Strahler number can be computed in linear time according to the number of nodes.*

Proof 7 *A number of different values bounded by $\lceil\sqrt{n}\rceil * \sqrt{2}$ and a range of value bounded by n makes linear the sort complexity. So, using the above proof we get a linear complexity.*

The generalization to directed acyclic graphs needs no modification of the algorithm. In fact, if we consider that no result are shared in the evaluation of an expression, we can transform a DAG in a forest and then compute the extended-Strahler number on each tree of the forest. In this case, under the theorem 5, the algorithm is linear according to the number of edges in the DAG.

Algorithm 2 Extended Strahler algorithm.

```
ExtendedStrahler(node η of a general tree T)
begin
  if (η is a leaf of T) then return 1
  freeRegisters = 0
  usedRegisters = 0
  for all children η_i of η in decreasing order induced by their treeStrahler values
  begin
    if (ExtendedStrahler(η_i) > freeRegisters) then freeRegisters=treeStrahler(η_i)
    usedRegisters=usedRegisters+1
    freeRegisters=freeRegisters-1
  end for
  return (freeRegisters+usedRegisters)
end ExtendedStrahler
```

4 Strahler number on rooted maps

In the following we denote by $G(V, E)$ a graph having a set of vertices (ie nodes) V and a set of edges (ie arcs) E. A rooted map is an embedding of the graph in the plane having a set of pointed vertices.
The idea of the Strahler number introduced above, is to view trees and directed acyclic graphs as n-ary expressions. To build a similar parameter on graphs, we consider that any graphs can be viewed as a sequential program. In a program:

- Registers are used to compute expressions. In our case, a spanning DAG enables to view a graph as an expression.
- Stacks are used to manage recursive calls. In our case, Cycles in the graph are interpreted as recursive calls in the program.

Thus, we define a two dimensional parameter on rooted maps. The first dimension ρ represents the number of registers needed to evaluate the program, the second dimension σ gives the number of necessary stacks (or nested call to the stack).

To compute σ we consider that edges which induce cycles in the graph represent recursive calls. However, for a given graph $G(V, E)$, finding a minimum set of edges $E_f \subset E$ such that the graph $G(V, E/E_f)$ is acyclic is well known as the feedback arc set problem. It has been proved[4][13] that it is a NP-Hard problem. Thus, given two isomorphic graphs, we couldn't ensure to produce the same valuation of nodes. In case of rooted maps, the problem of finding a unique set of edges E_u which verify the property (a) and (b) becomes linear.

- (a) $G(V, E/E_u)$ is acyclic.
- (b) $\forall \epsilon \in E_u \;\; G(V, E/E_u \cup \{\epsilon\})$ contains at least one cycle.

So, we consider the parameter only on such structures and we call it rooted map Strahler number, denoted by S_{rm}. In the following, we assume that the order of edges around nodes gives the execution order of the associated sequential program and that the root is its starting point.

For the purpose of graphs visualization, the use of rooted maps enables us to compute different valuations of the same graph depending on the user goal. For instance, by selecting a node, the user can choose the starting point of the algorithm. It also allows to take into account extrinsic parameters. For instance, we can use the map induced by the graph drawing or by edges weight.

In the following, we detail the construction of the rooted map Strahler number. For short, we will denote a rooted map by $G_S(V, E)$ assuming that the graph is already embedded in the plane and S is the root.

4.1 Edge decomposition

To compute S_{rm}, one must decompose the set of edges in four parts. Given a spanning tree T of a rooted map $G_S(V, E)$ these four sets are defined as following:

- Tree edge set $E_T = \{e \in E \mid e \in T\}$
- Descent edge set $E_D = \{e = (u, v) \in E \mid e \not\in T, \exists$ a path from u to v in $T\}$

- Return edge set E_R= $\{e = (u, v) \in E \mid e \not\in T, \exists$ a path from v to u in $T\}$
- Cross edge set E_C= $\{e \in E \mid e \not\in E_T \cup E_D \cup E_R\}$

This decomposition is used in some proofs of the strong component decomposition algorithm and can be done in linear time by using a depth first search algorithm for the spanning tree building.(fig 5)

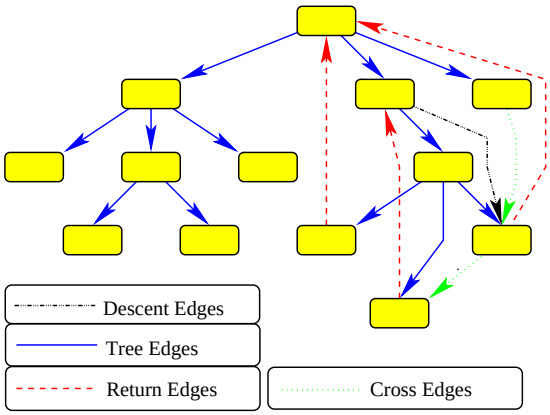

Figure 5: Edge decomposition.

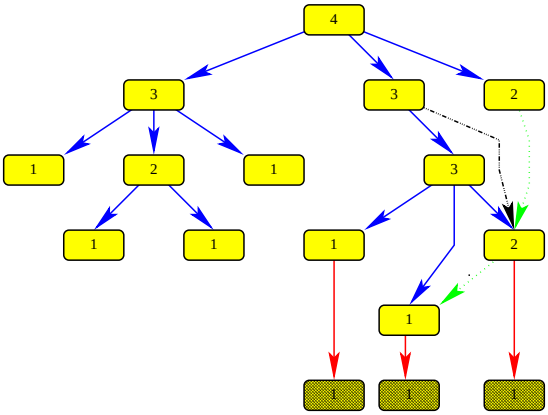

Figure 6: Temporary DAG.

4.2 Computation of the number of registers, ρ.

In order to compute the ρ parameter on a rooted map $G_S(V, E)$, one must build a temporary directed acyclic graph. This DAG D is obtain by unfolding the graph G of one step. The construction process is the following:

- Add all nodes of G in D.
- Add all edges e_i from $E_T \cup E_D \cup E_C$ in D.
- For all edges $e = (u_i, v_i)$ from E_R add a new node η_i in D and a new edge (u_i, η_i) in D. This new node allows to store in a register the result of a recursive call.

The figure 6 shows the DAG which has been built in order to compute the ρ parameter on the graph from the figure 5. The values of S_{ext} are given in the figure 6.

4.3 Computation of the number of necessary stacks σ.

In the following we present an algorithm which enables to compute the σ parameter. This algorithm uses three integers functions which are:

- Free: $V \rightarrow \mathbb{N}$: Number of free stacks, these stacks were used in previous evaluation and can be reused in others evaluations.
- Used: $V \rightarrow \mathbb{N}$: Number of used stacks used in the current evaluation.
- ToFree: $V \rightarrow \mathbb{N}$: Number of stacks to free after a node's child evaluation.

The figure 7 gives a view of the valuation of these three functions when one evaluates the node A after the node B. In the figure 7, large arrows represent sets of edges. To compute the used and free values of a node η we build a temporary list of integer-pairs(a,b). Each pair takes into account the value of a child η_i according that we access to it by the edge $\epsilon(\eta, \eta_i)$. Four cases arrise :

- If $\epsilon \in E_T$ (fig.7), the value of $\sigma(\eta)$ is at least $\sigma(\eta_i)$. In this $\sigma(\eta_i)$ stacks we can reuse those which are already free in η_i plus those which were necessary to compute the recursive calls which end in η. Thus, we add the pair [Free[η_i]+ToFree[η],Used[η_i]-ToFree[η]] to the temporary list.

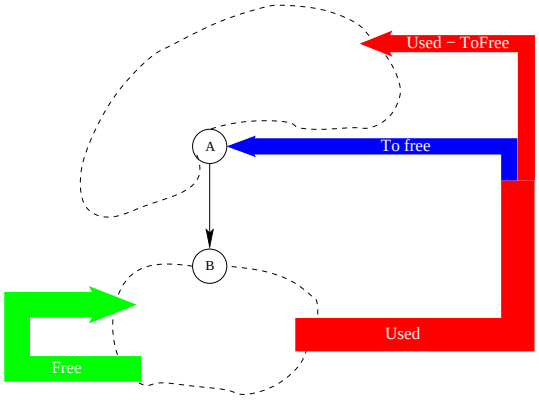

Figure 7: Tree edge processing.

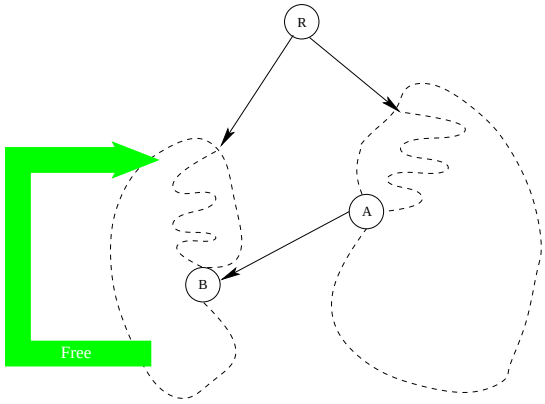

Figure 8: Cross edge processing.

- If $\epsilon \in E_C$ (fig.8), the value of $\sigma(\eta)$ is at least $\sigma(\eta_i)$. But, because the execution order is induced by the embedding of the graph, the value of η_i is already computed and so, its used stacks have already been free. Thus, we add to the temporary list the pair [Free[η_i]+used[η],0].
- If $\epsilon \in E_D$, it is the same as for a cross edge. However, computing [Free[η_i]+used[η_i],0] doesn't change the result for $\sigma(\eta)$ because η_i have already been treated by a tree edge. Thus, this case doesn't need to be computed.
- If $\epsilon \in E_R$, here it is a new recursive call. So, to compute η we need at least one stack and this stack will still used until that the η_i evaluation is finished. Thus, we add one to the number of stacks to free in η_i and we add the pair $[0, 1]$ in the temporary list.

Algorithm 3 Algorithm to compute Free,Used and ToFree.

$FreeUsedToFree$(node η)
begin
  Let L a list of pair, (i,j) with $i \in N, j \in N$.
  for all out going edges ϵ_i of η do begin
    η_i=target(ϵ_i)
    case $\epsilon_i \in of$:
      tree-edge set :
        ToFree[η]=0
        $FreeUsedToFree(\eta_i)$
        L.insert([Free[η_i]+ToFree[η],Used[η_i]-ToFree[η]])
      cross-edge set :
        L.insert([Free[η_i]+Used[η_i],0])
      return-edge set:
        ToFree[η_i]=ToFree[η_i]+1
        L.insert([0,1])
    end case
  end for
  Used[η]=Free[η]=0
  Sort L in the decreasing order induces by the first dimension.
  for all element $C_i = [a_i, b_i]$ in L do begin
    Used[η]=Used[η]+b_i
    Free[η]=max(Free[η],$a_i + b_i$)-b_i;
  end for
end FreeUsedToFree

Given the temporary list of pairs we can now compute $\sigma(\eta)$. To compute the minimal number of necessary stacks, we first sort the list such that $C_i = (a_i, b_i) < (a_j, b_j) = C_j$ if $a_i > a_j$. Then, we treat each pair in this order to compute the new used and free value of η. The principle is to compare the $a_i + b_i$ value to the current number of

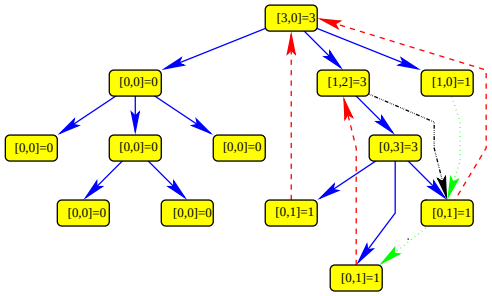

Figure 9: $[Free(\eta), Used(\eta)] = \sigma(\eta)$

free stacks in η. If $a_i + b_i$ is greater than Free(η) then we need more stacks to compute η. In the $a_i + b_i$ stacks a_i can be reused and b_i still used. So, we update Free(η) and Used(η) and then we treat an other pair. At the end we have:

$$\sigma(\eta) = Used(\eta) + Free(\eta)$$

Algorithm 3 summarized the method described above; Figure 9 shows the values computed by this algorithm on a given graph. The used map is the one induced by the drawing and the starting node is the one located at the top of the figure.

Note that, in case of general tree, σ is always equal to zero and $\rho = S_{ext}$.

Theorem 7 *Let G_S a rooted map having m edges, an upper bound of the time-complexity of the algorithm which computes S_{rm} on G_S is $O(m \log(m))$.*

Proof 8 *The algorithm is composed of three parts, the first one is the edge decomposition which can be done in linear time. subsequently we build a temporary DAG, this operation is linear according to the number of edges m. Then we compute the S_{ext} parameter in $O(m \log(m))$ using the theorem 4. Finally, using the same proof as for the theorem 4 one can prove that the complexity of σ computation is $O(m \log(m))$. Thus, an upper bound of the complexity the algorithm is $O(m \log(m))$.*

The given algorithm is a simplification of the final one. In fact, the three above steps can be merged together, by this way we do not change the theoretical complexity of the algorithm but, in practice, it reduces the constant term in the time and space complexity of the algorithm. One of the trick is that we don't need to build the temporary DAG in order to compute S_{ext} on it. An open source implementation of this algorithm is given as a plug-in in the Tulip software[2].

5 Strahler numbers for real time navigation.

It is well known that we can automatically extract important information about relational data by only using the structure of graphs. For instance, in software reverse engineering one can automatically detect modules or components of a program by analyzing the graph structure induced by the source file inclusion [16]. However, to our knowledge, using it in incremental rendering has not been studied yet. Here, we discuss about how to use the Strahler numbers introduced above for it.

First of all we project the two dimensional parameter S_{rm} into a one dimensional space. This projection is done by computing the euclidian norm of $[\sigma, \rho]$ noted:

$$\beta(\eta) = \sqrt{\sigma^2(\eta) + \rho^2(\eta)}$$

We have chosen the euclidian norm in order to preserve the original value of Sthraler numbers in case of trees, and to reduce the differences between components which are almost acyclic ($\sigma \sim 0$) or almost only cycles ($\rho \sim 1$) and those which are the both ($\rho \sim \sigma$).

Then, we assign to each edge the value:

$$\phi(\epsilon) = min(\beta(source(\epsilon)), \beta(target(\epsilon)))$$

By using the minimum value between source and target we ensure to give higher values to edges between important elements. For instance, when computing S_{rm} on the graph of human metabolism, the node representing water receives a high value for S_{rm} as it is involved in comparatively many of the chemical reactions. However,

during visualization the schematic graph must only display links between water and elements which take place into complex reactions else too many edges are displayed. In practice, the results with other methods such as the average between source and target don't give better approximation of the final drawing.

The idea of this heuristic is that the more registers and stacks we use to evaluate a node (or a a sub-program) the more relevant the node should be. And so, by displaying first the more complex part of the graph we should display the more important part of it. Thus, we order the set of nodes such that :

$$\eta_i < \eta_j \; if \; \beta(\eta_i) < \beta(\eta_j)$$

and the set of edges such that:

$$\epsilon_i < \epsilon_j \; if \; \phi(\epsilon_i) < \phi(\epsilon_j)$$

Then, during the navigation we use these two ordered sets for the incremental rendering of the graph.

In terms of time complexity, this method can be used to draw huge graphs (500K nodes) on standard PC hardware. For trees and directed acyclic graphs all the process can be done in linear time using the theorem 5 and for the general graphs it can be done in $O(m \log(m))$ time. One must notice that with real data, the time used in order to sort elements is almost linear with general graphs.

During visualization, we can compute the order of elements at the early start and then use always the same. It is not necessary to rebuild the order after each actions. However, in order to take into account the user operations we can recompute this order during navigation. For instance, we can use the nearest node of the fish eyes center as starting node of the Strahler parameter. The computation complexity of the method allows such operations even with huge graphs.

6 Experimentation

We give here practical results of our method. The experimentation has been made with the Tulip program[2]. The samples are directed graphs. To compute the S_{rm} parameter on the samples we have automatically chosen the starting nodes by using the algorithm 4. The idea of this automatic selection is that if a node have no incoming edge it should be an entry in the network, and if it doesn't exist such a node, we hope that nodes with maximum degree are important nodes in the network. Figures 10 to 17 are snapshots of the images displayed to the screen at different stages of the incremental rendering.

Algorithm 4 Automatic selection of input nodes

while (all nodes are not computed)
Begin
  if it exists uncomputed source of the graph(with an in-degree=0) use it as entry nodes and run the algorithm
  else use the uncomputed node with the greater out degree.
End

The figures 10 to 13 show the incremental drawing of a binary tree with 520,000 elements. The layout has been generated automatically by using the Reingold and Tilford[19] tree drawing algorithm. One can see on this sample that even with 4600 nodes (less than 1% of the entire graph), there is an impression that the abstracted graph retains some elements of the overall shape of the full graph. The hope is (and this would require empirical evaluation to evaluate) that sufficient structure is maintained to preserve the mental map of the user during graph navigation operations. During experimentation we have compare this results with other tree parameters, in all cases the results don't seem better than the ones obtain with the Strahler parameter.

The figures 14 to 17 show the incremental rendering of a general graph. The sample is the graph of the structure of our laboratory web site. The 3D layout has been generated with a spring electrical algorithm[5]. As for the previous sample the first picture with only 200 elements (less than 6.5 % of the entire graph) gives a good abstraction of the final one. Furthermore, during the experimentation on such graphs, the heuristic has given good result by displaying first the relevant part of the graph. On the figure 14 the displayed graph is the most important part of our web-site. By most important we mean the kernel of our web site.

7 Conclusion and future work.

Presented results are directly usable and encourage the use of such methods for graph visualization. Future works will be to identify others intrinsic parameters which improves the result of this heuristic and to put a special emphasis on evaluation method of this kind of navigation.

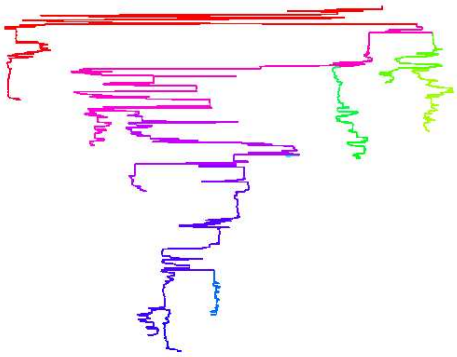

Figure 10: 4,600 elements

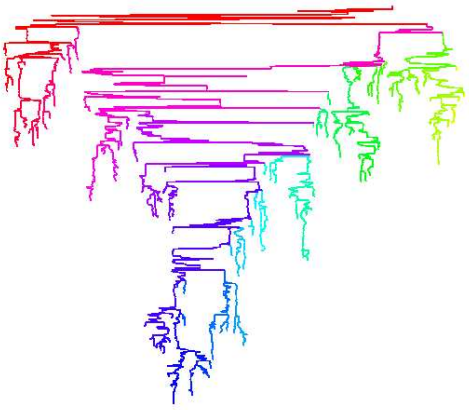

Figure 11: 16,400 elements

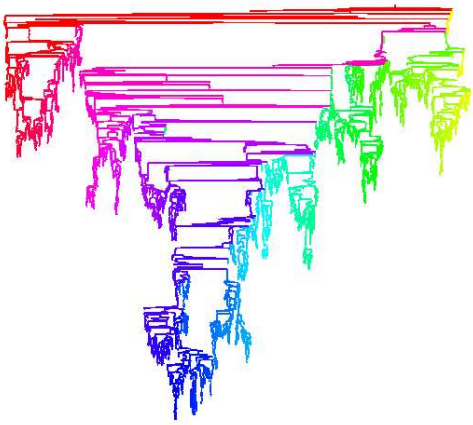

Figure 12: 66,000 elemenst

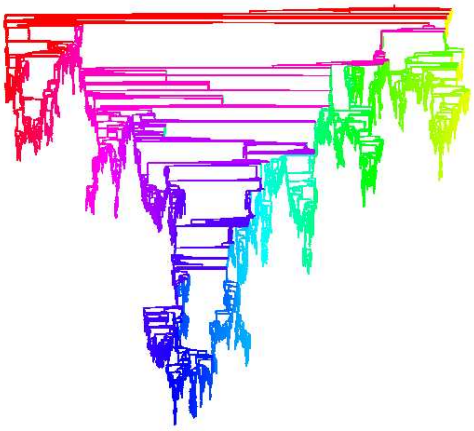

Figure 13: 520,000 elements

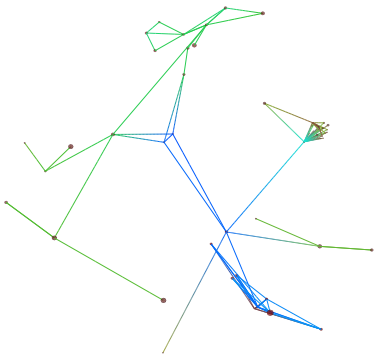

Figure 14: 200 elements

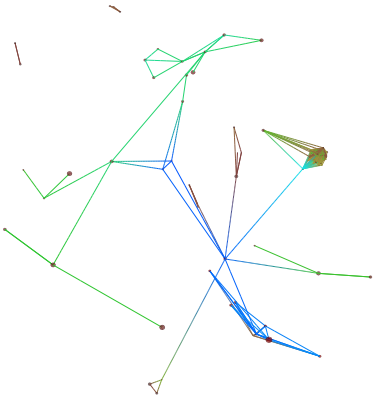

Figure 15: 600 elements

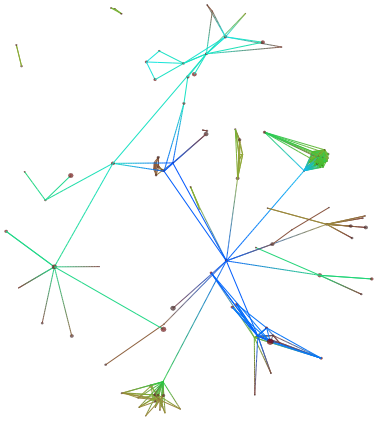

Figure 16: 1,000 elements

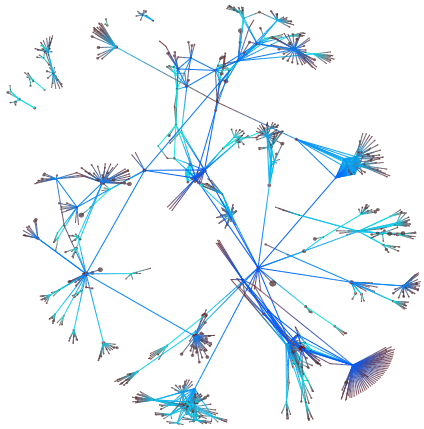

Figure 17: 3,200 elements

The experimentation has shown that the part of data with the greater value is not without sense. More precisely we are currently working on clustering algorithm which use the Strahler number introduce above. Even if the clustering methods will be slower than the one presented above, first results on interaction protein graph are pretty good.

All the features proposed in this article are available through our graph visualization software called Tulip[2][1].

8 Acknowledgment

We gratefully acknowledge David Duke and Maylis Delest for their comments on previous version of this paper.

References

[1] C.J. Alpert and A.B. Kahng, *Recent directions in netlist partitioning: A survey*, Integration: The VLSI J. **19** (1995), 1–18.

[2] D. Auber, *Tulip*, Proc. 9th Symp. Graph Drawing, GD (Sebastian Leipert Petra Mutzel, Mickael Junger, ed.), Lecture Notes in Computer Science, LNCS 2265, Springer-Verlag, 2001, pp. 335–337.

[3] D. Auber and M. Delest, *A clustering algorithm for huge trees*, Advances in Applied Mathematics (To appear).

[4] J. Bang-Jensen and G. Gutin, *Digraphs: Theory, algorithms and applications*, Springer-Verlag, 2000.

[5] G. Battista, P. Eades, and al, *Graph drawing, algorithms for the visualization of graphs*, no. 1, 1999.

[6] A. Broder, F. Maghoul, and al, *Graph structure in the web*, Proc. 9th International World Wide Web Conference, Computer Networks, vol. 33, 2000, pp. 309–320.

[7] A.P. Ershov, *On programming of arithmetic operations*, Com, of the A.C.M **1** (1958), no. 8, 3–6.

[8] J.M. Fedou, *Nombre de strahler sur les arbres généraux, ecole jeunes chercheur en algorithmique et calcul formel, gdr alp, bordeaux*, may 1999.

[9] C. Friedrich and M.E. Houle, *Graph drawing in motion ii*, Proc. 9th Symp. Graph Drawing, GD (Sebastian Leipert Petra Mutzel, Mickael Junger, ed.), Lecture Notes in Computer Science, LNCS 2265, Springer-Verlag, 2001, pp. 220–231.

[10] N. Guelzim, S. Bottani, and al, *Topological and causal structure of the yeast transcriptional regulatory network.*, Nature genetics **31** (2002), 60–63.

[1] Tulip, software is under GPL license and is available at : $http://www.tulip-software.org$

[11] I. Herman, M. Marshall, and al, *Density functions for visual attributes and effective partitioning in graph visualization*, 2000.

[12] A.K. Jain and R.C. Dubes, *Algorithms for clustering data, prentice-hall, englewood cliff s*, NJ **88** (1988), 1988.

[13] R. Karp, *Reducibility among combinatorical problems.*, Complexity of Computer Computations (1972), 85–103.

[14] S.R. Kumar, P. Raghavan, and al, *The web as a graph*, Proc. Symposium on Principles of Database Systems, 2000, pp. 1–10.

[15] B. LeBlanc, D. Dion, and al, *Constitution et visualisation de deux réseaux d'associations verbales*, Actes du colloque Agents Logiciels, Coop´eration, Apprentissage et Activit´e Humaine (ALCAA), 2001, pp. 37–43.

[16] S. Mancoridis, B.S. Mitchell, and al, *Using automatic clustering to produce high-level system organizations of source code*, Intl. Workshop on Program Comprehension (1998).

[17] S. Marshall, *Methods and tools for the visualization and navigation of graphs.*, Ph.D. thesis, University Bordeaux I, June 2001.

[18] T. Munzner, *Drawing large graphs with h3viewer and site manager*, Proc. 5th Int. Symp. Graph Drawing, GD, Lecture Notes in Computer Science, LNCS, Springer-Verlag, 1998, pp. 384–393.

[19] E.M. Reingold and J.S. Tilford, *Tidier drawings of trees*, IEEE Transactions on Software Engineering **7** (1981), no. 2, 223–228.

[20] A. Robinson, *Visualisation of microarray gene expression data*, http://industry.ebi.ac.uk/ alan/.

[21] M. Sarkar and M.H. Brown, *Graphical fisheye views*, Communications of the ACM **37** (1994), no. 12, 73–84.

[22] A.N. Strahler, *Hypsomic analysis of erosional topography*, Bulletin Geological Society of America 63 1117-1142. (1952).

[23] G. Viennot, *Trees everywhere*, Colloquium on Trees in Algebra and Programming, 1990, pp. 18–41.

[24] C. Ware, *Information visualization: Perception for design*, Moragn Kaufmann, 2000.

[25] D.J. Watts, *Small worlds: The dynamics of networks between order and randomness*, 1999.

[26] G.J. Wills, *NicheWorks — interactive visualization of very large graphs*, Proc. 5th Int. Symp. Graph Drawing, (Giuseppe Di Battista, ed.), Lecture Notes in Computer Science, LNCS, no. 1353, Springer-Verlag, 1997, pp. 403–414.

[27] K.P. Yee, D. Fisher, and al, *Animated exploration of graphs with radial layout*, Proc. IEEE Symposium on Information Visualization, San Diego, 2001, pp. 43–50.