Strahler based graph clustering using convolution

David Auber, Maylis Delest
LaBRI - Université Bordeaux 1
351 Cours de la Libération
33405 Talence CEDEX, France
david.auber@labri.fr, maylis.delest@labri.fr

Yves Chiricota
Département d'informatique et mathématique
Université du Québec à Chicoutimi
555, boul. de l'Université, Chicoutimi (Qc)
Canada G7H 2B1
Yves_Chiricota@uqac.ca

Abstract

We propose a method for the visualization of large graphs. Our approach is based on the calculation of a density function resulting from the application of a metric on the vertices of a graph. The density function is then filtered using a convolution, leading to a partition of the graph. The choice of an appropriate kernel for the convolution makes it possible to control the number of clusters, and their size. Our algorithm can be executed automatically, but the parameters can also be interactively fixed by the user. We applied the algorithm to the problem of legacy code extraction from inclusion relation of C++ source files and film sequence analysis. The metric used here is defined from Strahler numbers, which measure the "ramification" level of graph vertices.

1 Introduction

Given a graph, it is natural to want to group vertices with common characteristics, taking into account various criteria. Such vertex groupings allow for the apprehension of a graph structure. Particularly, it is possible to obtain hierarchic views of a graph from fragments. These views not only simplify graph visualization, but also help better understand the relations between fragments. Thus, automatic fragmentation of large graphs is the object of vigorous research endeavours in graph theory, and other fields as well.

Fragmentation has multiple applications. In biology for example, graphs are used to represent proteins from vast data bases. The vertices correspond to the proteins, and the edges are weighed as a function of similarity between proteins. In this context, fragmentation is used to determine protein families. Enright *et al.* [8] use a heuristic method based on Markov chains to determine graph fragments. Kawaji *et al.* [14] also use a heuristic method in their work. In this case, the fragmentation algorithm minimises a function that measures the level of separation between fragments.

Fragmentation is also applied in data bases. Chen *et al.* [6] use this approach to group items belonging to data bases as a function of their similarity. Here also, fragmentation calculation is done through a function that measures similarity between data base elements. Another fragmentation technique was introduced by Hermann *et al.* [13], and consists in fragmenting the graph based on a vertex metric (a metric is a function which associates a non-negative real number to each vertex of a graph). The density of metric function is then used to group vertices.

Genetic algorithms are among the heuristic methods used for fragmentation. Maio *et al.* [16, 19] introduced a genetic optimisation algorithm to minimize an objective function that measures the quality of the hierarchic fragmentation of graphs. The function to be optimised is derived from six criteria used to measure fragmentation quality. These graphs are used in the robotics industry to represent the space in which an intelligent mobile system moves. A number of fragmentation techniques constrained by the minimisation of edges between fragments are presented in [2].

Graph fragmentation is also applied in software engineering to analyse in retrospect the structure of very large programs. A recent paper by Wiggerts [25] deals with computer system modules and contains algorithms designed for the extraction of legacy code. In the same vein, Mancoridis *et al.* [18, 17] propose a genetic algorithm for the extraction of software structures from dependence graphs in C++ source files. Here, the function that needs to be optimized corresponds to a measure of fragmentation quality. Their method was implemented into a program called Bunch. Also, Tzerpos *et al.* [22, 23] introduced a fragmentation algorithm based on the use of patterns inherent to the implementation of computer systems.

Most of these methods do not allow for dynamic control of fragment size, which help the user to better manage detail level when viewing graphs. This function is very useful when exploring large graphs. It is in fact closely linked to classic "details on demand" techniques (see Schneiderman [20]). Further, calculation time for heuristic algorithms being relatively long (many minutes), these methods do not allow for interaction with the fragmentation program.

The algorithm we propose palliates these problems. For a given metric we use convolution to filter the histogram representing metric distribution. This characteristic allows us to introduce parameters that control fragment size. The algorithm presented here can be executed automatically, but the parameters can also be interactively fixed by the user. We ap-

plied the algorithm to the problem of legacy code extraction from inclusion relation of C++ source files and to multimedia files. To this end, we used the σ metric, based on Strahler number calculations. These numbers were used in hydrogeology [21], but are also extremely relevant for other types of applications using tree structures [24]. This metric measures the "ramification" level of graph vertices. Our choice of metric relies on previous experimentation [4], in which we noticed that Strahler numbers helped find file groups belonging to the same components of a program. Particularly, we noticed that histograms corresponding to Strahler numbers calculated for a graph were made up of intervals corresponding to non-zero values, relatively distant from each other, and corresponding to significant program components.

An important characteristic resulting from the use of Strahler numbers in our method is fast calculation time for fragmentation. The approach thus allows for the implementation of interactive tools to navigate in the graph representing a computer system. As opposed to heuristic methods, with which fragmentation calculations can take several seconds, sometimes minutes, our approach allows for real-time navigation and viewing.

The rest of the paper is structured in the following manner: Section 2 gives a few fundamental definitions and describes Strahler metric σ for directed acyclic graphs (DAG). Our fragmentation method is then presented in Section 3, in which we describe how histogram values are filtered and fragments calculated from the results. Finally, we have implemented the method in our visualisation software[1] Tulip [5]. An example of the application of our method in the context of software engineering and multimedia is given in Section 5.

2 Definitions

Strahler numbers for binary trees were introduced by Horton [12] and Strahler [21] for hydrographic studies. In computer science, they correspond to the minimum number of registers needed to evaluate a arithmetical expression made up of binary operators [9, 11]. These numbers have also been used by Viennot [24] in the field of image synthesis, to calculate realistic images of trees.

The Strahler numbers were generalised to any rooted tree by Fedou [10] and correspond to the calculation of the minimum number of registers needed to calculate arithmetical expressions with operators of any arity. Given a rooted tree A, the calculation is made for a vertex x, of which the children are $x_1, x_2, \ldots, x_n$ and is achieved using the following recursive formula:

$$\sigma(x) = \begin{cases} 1, & \text{if } x \text{ is a leaf;} \\ \max(\{\sigma(x_i) + i - 1\}_{1 \leq i \leq n}), & \text{else.} \end{cases}$$

By convention we define $\sigma(A) = \sigma(r)$, where r is the root of A. Figure 1 illustrates this calculation.

The generalisation of these numbers to DAGs is direct. In fact, given a DAG, it suffices to consider a forest obtained from the DAG, and to associate to the DAG the Strahler number of

[1]GPL product, download from www.tulip-software.org.

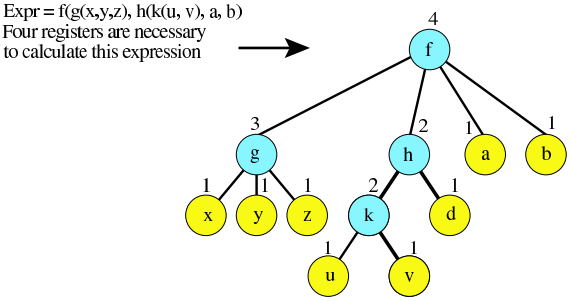

Figure 1. Strahler number calculation for a tree.

that forest. A generalisation of Strahler numbers to any type of graph could be found in [4]. This generalisation consists in considering a rooted map as a recursive expression and determining the number of stacks and registers needed to evaluate the expression. One of the important properties of Strahler numbers is that they measure tree ramifications of trees and graphs. Moreover, it can be shown [3] that for a DAG with n vectices, the number of distinct values is bounded by $\sqrt{2 * n}$. The calculation of Strahler numbers can be done in $O(n)$. We will use these characteristics of Strahler numbers for the fragmentation of graphs.

We have chosen these numbers in our system due to their ability to fragment graphs according to their clusters structure. We have verified this property experimentally.

Now let us introduce a few others definitions used in the following. Let $G = (V, E)$ be a graph. Let K be a set and $\phi : E \longrightarrow K$ a function. The histogram of values (or density function) of ϕ is the function $h : K \rightarrow \mathbb{N}$, which counts the number of elements $x \in E$ such that $\phi(x) = v$. Hence, we have $h(v) = |\phi^{-1}(v)|$. In the case of graphs, it consists in counting the number of vertices having the same value.

We will use the following definition of convolution:

Definition 2.1 *Let f and h be two functions $f, h : \mathbb{Z} \rightarrow \mathbb{R}$, we call the* convolution *of f and h function $f * h$ defined by:*

$$(f * h)(r) = \sum_{s=-\infty}^{+\infty} h(r - s) f(s).$$

The function f is called the kernel *of the convolution.*

Definition 2.2 *Let $f : \mathbb{N} \rightarrow \mathbb{N}$; $m \in \mathbb{N}$ is called a* local minimum *of f if $f(m-1) > f(m)$ and $f(m+1) > f(m)$.*

3 Fragmentation

When one looks to process a metric in order to fragment a graph, it often happens that it does not follow a well-known distribution, even if, in other respects, vertex metrics express graph properties. For example, for Strahler metrics, experimenting [4] shows that the metric may make possible the grouping of vertices having close values, that is to say similar importance in the graph. On the other hand, Strahler number distribution in binary trees is complex [11] and thus of

little use for visualisation. Of course, the same applies to general trees for which this distribution is unknown. However, by observing the values, one can see that the distribution (*the histogram*) of these values is made up of intervals grouping elements and nul sub-intervals of more or less spread. A simplistic approach would consist in fragmenting the graph in accordance with these nul sub-intervals. However, this is only relevant if the sub-intervals are of constant width, which is not verified in the case of Strahler numbers (or even generally for that matter). Figure 2 illustrates this phenomenon (typical distribution of Strahler parameter). The figure shows a

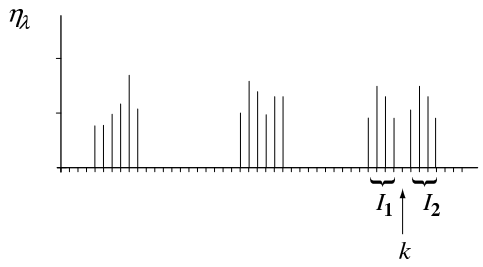

Figure 2. Nul sub-interval.

nul sub-interval at position k, which generates two parts in the simplistic approach. A more convenient method would consist in obtaining only one part in this case. The splitting of this part into two would then be postponed to a later step in the process of hierarchical fragmentation of data. Let us see how to use a convolution to deal with this situation.

The clustering algorithm presented here relies on the calculation of a partition of V from the density function η. In order to calculate this partition of vertices, one may observe that any partition $\Pi = \cup_k I_k$ of $\mathrm{supp}(\eta)$ leads to a partition of V. The partition of V is defined by the following equivalence relation: $u \approx v$ if and only if $\phi(u)$ and $\phi(v)$ belong to the same subset in partition Π. We will apply this idea in conjunction with the convolution of η with a well chosen kernel, to get a partition of V. Such a partition leads to a clustering of G in a natural manner.

The algorithm we propose consists of three steps: discretisation, convolution filtering, and fragmentation.

3.1 Discretisation

Usually, a metric ϕ defined from a graph $G(V, E)$ takes its value in $\mathbb{R}$, resulting in a density function $h : \mathbb{R} \to \mathbb{N}$. In order to reduce the complexity of the calculations, a discrete density function $\eta : \mathbb{N} \to \mathbb{N}$ must be used. The construction of this function η requires the discretisation of the original histogram h.

The calculation of η is achieved by translating ϕ in such a way that its smallest value is equal to 0. From this point forward, we will suppose that ϕ satisfies this property. Next, we define the discrete density function η by

$$\eta(k) = \eta_\lambda(k) = |\{v \in V \ : \ k\lambda \leq \phi(v) < (k+1)\lambda\}|,$$

where λ acts as a scaling factor.

We also define $\mu = \mu_\lambda = \max(\mathrm{supp}(\eta_\lambda))$. The filtering will consist in building a partition of interval $I = [0, \mu]$.

3.2 Filtering

To take into account the phenomenon encountered in Figure 2, we propose filtering density function η, using a convolution. The application of a correctly chosen convolution to η will mean that intervals will be merged as a function of their width, and of the density around them. Moreover, the choice of the kernel for the convolution allows to vary the number of sub-intervals, thus making it possible to obtain a hierarchical fragmentation method. For example, if we choose a kernel so that there are only three sub-intervals, we will then be able to apply the method once again, recursively, on the three sub-intervals, by adapting the kernel each time.

In our study, we use the following function as a kernel:

$$\zeta_{A,\omega}(k) = \begin{cases} 0 & \text{if } k < -\omega, \\ \frac{A}{\omega}k + \omega & \text{if } -\omega \leq k < 0, \\ -\frac{A}{\omega}k + \omega & \text{if } 0 \leq k < \omega, \\ 0 & \text{if } k > \omega. \end{cases}$$

In what follows, we will postulate that $A = 1$ and the kernel will be denoted by ζ_ω. The ω parameter allows for varying the number of partition elements of I, and thus of sub-graphs in the corresponding fragmentation. The filtering $\widetilde{\eta}$ of the density function η is derived from the convolution:

$$\widetilde{\eta} = \widetilde{\eta}_\omega = \eta * \zeta_\omega.$$

Figure 3 illustrates histogram filtering. Partition Π associated to this histogram contains three parts: X_1, X_2, and X_3. One will notice that the nul sub-interval is "absorbed" by parts X_2 and X_3 following convolution. This is related to the fact that the weight of these parts is relatively high, compared to the width of the nul sub-interval.

3.3 Fragmentation

Construction of density function η requires discretisation of the set of values. For discretisation to be as near as possible to the initial histogram, we use the following formula to calculate scale parameter λ :

$$\lambda = \min(\{\| \phi(s_i) - \phi(s_j) \|\}_{s_i, s_j \in V \text{ and } \phi(s_i) \neq \phi(s_j)}).$$

So, λ is the minimal difference between any two values of ϕ. The size of the discretised histogram to be built being equal to μ_λ, calculation complexity of the convolution and memory space required for the storage of η_λ imposes that the maximum value of μ_λ must be limited. This limit value is denoted μ_{thres}. If $\mu_\lambda > \mu_{\mathrm{thres}}$, value λ is increased to obtain $\mu_\lambda = \mu_{\mathrm{thres}}$. In our experimentation, the value of μ_{thres} has been established at 2^{15}, which gives enough flexibility since, in practice, it is rarely attained.

Some relevant nul sub-interval of η may disappear if one considers the convoluted function $\widetilde{\eta}$. In fact, if the parameter ω

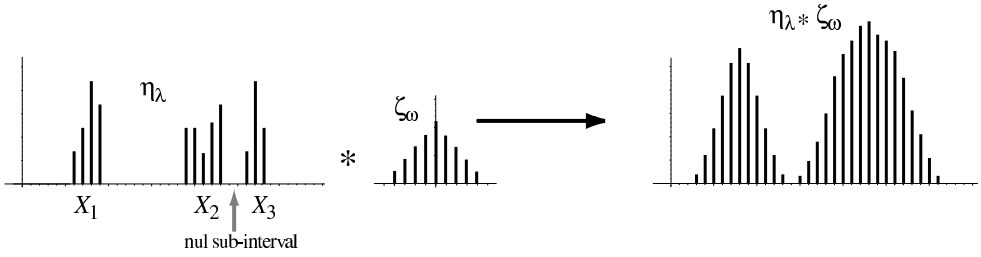

Figure 3. Filtering using a convolution.

is sufficiently large, it is even possible to remove every nul sub-interval. To deal with this situation, we also use local minima of $\widetilde{\eta}$ to define the partitions. So, in the calculation of the partition, we thus take into account the nul sub-intervals, which can obviously be interpreted as a local minimum, as well as other local minima of $\widetilde{\eta}$. Let $(m_1, \ldots, m_k)$ be the sequence of local minima of $\widetilde{\eta}$, ordered in such a way that $m_i < m_{i+1}$. An element, X_i, of partition $\Pi = \{X_1, X_2, \ldots, X_{k+1}\}$ of $\mathrm{supp}(\eta)$ is then defined in the following manner:

$$X_i = \{x \mid x \in \mathrm{supp}(\eta) \text{ and } m_{i-1} < x < m_{i+1}\}$$

with, $m_0 = -1$ and $m_{k+1} = \mu + 1$. This partition of $\mathrm{supp}(\eta)$ leads to a clustering of G using the method described above.

For automatic and progressive partitioning of the graph into sub-graphs, to obtain a hierarchy, we use parameter ω. In fact, it allows for varying the number of local minima. Figure 4 illustrate the process. The slider named "Width" is linked to parameter ω. The vertical lines correspond to local minima. The chosen value of ω is that which allows to obtain a minimum number (greater than zero) of local minima. It is noted ω_c.

Once fragmentation is done, the process can be repeated for each sub-graph created. A fragmentation tree of the initial graph is thus obtained, and the maximum size of that tree is $2 \cdot |\mathrm{supp}(\eta)| - 1$.

4 Complexity

This section describes the complexity of the three steps of the algorithm, which are: discretisation, filtering, and construction of the hierarchy. We use the value of $|\mathrm{supp}(\eta)|$ in our calculations. As we will see in Section 5, knowledge of the properties of metric ϕ allows to obtain information on $|\mathrm{supp}(\eta)|$, and in that way obtain a more precise value of the upper bound of algorithm complexity.

Proposition 4.1 *Calculation complexity of the discretised histogram is in*

$$O(|V| \log(|V|) + \mu).$$

Proof 1 *To calculate λ, the elements of V must be sorted and the ordered sequence iterated. Calculation of λ can thus be obtained in $O(|V| \log(|V|))$. To build μ, a table of size μ must be initialized to zero and inside affectations $|V|$ need to be made.*

Proposition 4.2 *Complexity of calculation of the filtered histogram $\widetilde{\eta}$ is in*

$$O(|\mathrm{supp}(\eta)|\omega + \mu).$$

Proof 2 *To calculate the histogram $\widetilde{\eta}$, function ζ_ω must be calculated for each value of η. In general, an upper bound for calculation complexity of $\widetilde{\eta}$ is thus in $O(\mu\omega)$. However, to calculate $\widetilde{\eta}$, it suffices to take into account the non-zero values of η ($\mathrm{supp}(\eta)$). Calculation complexity is then in $O(|\mathrm{supp}(\eta)|\omega + \mu)$. Algorithm 4.1 summarizes calculation of $\widetilde{\eta}$.*

Algorithm 4.1 *Calculation of $\widetilde{\eta}$*
for $i \in [0..\mu]$ **do**
  $\widetilde{\eta}(i) = 0$
for $i \in \mathrm{supp}(\eta)$ **do**
  for $j = -\omega$ *to* ω **do**
    $\widetilde{\eta}(j+i) = \widetilde{\eta}(j+i) + \eta(i)\zeta_\omega(j)$

Proposition 4.3 *Calculation complexity of ω_c is in*

$$O(|\mathrm{supp}(\eta)|\mu^2).$$

Proof 3 *Let $m_\eta(\omega)$ be the number of local minima of $\widetilde{\eta}_\omega$. Function $m_\eta(\omega)$ having no other characteristic than the fact that it is globally decreasing, to find ω_c, all possible solutions need to be looked over, that is to say from 1 to μ. For each operation, function $\widetilde{\eta}_{\omega_i}$ must be calculated, as well as its number of local minima. The complexity of an operation is then in $O(|\mathrm{supp}(\overline{\phi})|\omega_i + 2\mu)$. The complexity of an operation also depends on the value of ω_i. The upper limit of complexity C_{ω_c}, of the search for ω_c, verifies the following inequality:*

$$\begin{aligned} C_{\omega_c} &\leq \sum_{i=1}^{\mu} (2\mu + |\mathrm{supp}(\eta)|\, i) \\ &= 2\mu^2 + \frac{\mu(\mu+1)}{2} |\mathrm{supp}(\eta)| \\ &\leq 2\mu^2 |\mathrm{supp}(\eta)| + \frac{\mu(\mu+1)}{2} |\mathrm{supp}(\eta)| \\ &= O(|\mathrm{supp}(\eta)|\mu^2). \end{aligned}$$

Proposition 4.4 *Calculation of graph hierarchy is in*

$$O(|\mathrm{supp}(\eta)|^2\mu^2).$$

Proof 4 *It is trivial considering the fact that the size of the fragmentation tree is equal to $2|\mathrm{supp}(\eta)| - 1$.*

[htb]

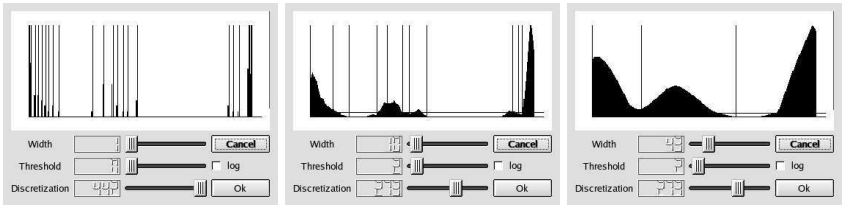

Figure 4. Variation of the fragmentation in function of ω.

5 Applications

In this section, the results of experimentation using our fragmentation method are presented.

5.1 Include graphs

We apply our method to *include graphs* obtained from C++ source code used to compile a program. Given a set of source files, this graph is obtained from the `#include` preprocessor directive, and is defined as follows :

- Vertices s_i correspond to source files;
- If file `a`, corresponding to vertex s_i, contains the instruction `#include "b"`, where `b` is the file corresponding to vertex s_j, then there is an edge $e(s_i, s_j)$ in the graph.

The analysis of result relevancy requires in-depth knowledge of the source files used, which led us to choose the source files from our visualisation program, Tulip, to build our example. We used the Strahler valuation (cf. Section 2) to fragment the program. Let us note that the initial graph (Figure 5) has only one connected component. The results shown here required two-step fragmentation, and they are the smallest fragments obtained from our application. They correspond to the second hierarchical fragmentation step. Further, in order to get more relevant information, we proceeded to decompose the resulting sub-graphs into connected components.

In Figure 6, the quotient graph was constructed from the six clusters obtained by the mean of our algorithm. The other clusters shown are

- All the Tulip plug-ins,
- All the files required for the management of each property,
- Files from the graphic interface library intermixed with those of Tulip. One can see that only the files required for the use of Tulip in interactive mode (with GUI) make up part of this cluster.

Diagram 6 clearly shows that the resulting clusters yield an interesting decomposition of the program. A first observation is that the plug-ins use all parts of the program, but are independent, and the loading mechanisms of the plug-ins are tied to both the Tulip library and the C++ standard library. This experiment also allowed us to detect inconsistencies in our files. For example, in our first experiment, a link between cluster $property$ and cluster $TulipGUI$ appeared in the quotient graph. Verification showed that it was the result of errors in our files (useless data).

Figures 7 and 8 show fragment selection resulting from our method. By making correspondence between the graphs produced by our program architecture, the clusters have, for the most part, a sense in terms of program components. Figure 7 shows the files involved in the management of Tulip types, as well as the "abstract construction" mechanisms used for the management of plug-ins related to these same types.

Figure 8 shows that the great majority of the C++ standard library files were grouped.

5.2 Multimedia files

Structuring video documents into logical video segments, such as shots, scenes, objects, and indexing them by characterizing their motion, color, or texture is a key feature of the MPEG7 [1] standard. Numerous descriptors can be used and several techniques have already been proposed to describe video content by means of low-level features. In a video document, where each shot is characterized by a vector or scalar descriptor, a distance between shot descriptors can be computed. Based on distance matrix for shots, a video document is a weighted complete graph for which the distance matrix is in fact an incidence matrix. The vertices of the graph are the shots and the edges between two vertices are weighted by the corresponding entry in the distance matrix. After selecting a subset of relevant distances (edges) for grouping shots into scenes, by statistical methods [7], Strahler numbers on graphs can be used to build clusters of similar shots according to Strahler values of vertices. Clustering is fulfilled interactively, based on user-satisfaction of resulting clusters. Clusters are displayed as a quotient graph, i.e. each cluster is a node in the graph of clusters. Users can assess the quality of clusters by viewing the key-frame that is mapped on each shot-node. Using this process, we decided to split "Aquaculture" SFRS®into 5 clusters. Figure 9 shows the clusters of "Aquaculture" in which one can navigate, using the mouse, seeing the image abstracting a shot. Figures 14 and 15 illustrate the performance of the clustering method: the majority of shots with similar color-content and even scene layout were grouped together in one cluster.

6 Conclusion

One of the applications of our method concerns systems to help in the design and understanding of relations regarding the inclusion of source files in programs. As noted by Koschke [15], most methods used to solve this kind of problem yield only approximations. Of course, users of these types of systems must expect a certain degree of imprecision. Our method however yields acceptable results, as shown in the previous section. It is also important to note that if the structuring of a computer system is not properly done, an algorithm cannot be used to correct the problem.

The algorithm presented herein is an exploration tool to visualise the physical architecture of a computer system and other structures relying on graphs, like multimedia files. Visualisation then makes possible the optimisation of these system, and to detect certain errors, as is the case for our design. In fact, the initial inclusion graph (from which results the graph in Figure 6) comprised several inconsistent edges between groups. Verification showed that it was the result of errors in our files (useless data).

References

[1] *ISO/IEC JTC 1/SC 29/WG 11/M6156, MPEG-7 Multimedia Description Schemes WD (Version 3.1), Beijing*. July 2000.

[2] C. J. Alpert and A. B. Kahng. Recent directions in netlist partitioning: a survey. *Integration: the VLSI Journal*, 19:1–81, 1995.

[3] D. Auber. *Outils de visualisation de larges structures de données*. Thèse Université Bordeaux 1, LaBRI, 2002.

[4] D. Auber. Using strahler numbers for real time visual exploration of huge graphs. In *International Conference on Computer Vision and Graphics*, ICCVG2002, pages 56–69, September 2002.

[5] D. Auber. Tulip - a huge graph visualization framework. In M. J. P. Mutzel, editor, *Graph Drawing Software Book*, Mathematics and visualization. Springer-Verlag, 2003. To appear.

[6] N. Chen, A. Chen, L. Zhou, and L. Lu. A graph-based clustering algorithm in large transaction databases. *Intelligent Data Analysis*, 5(4):327–338, 2001.

[7] M. Delest, A. Don, and J. Benois-Pineau. Graph-based interfaces for navigation in indexed video content. pages 49–55, Renne, France, 2003. CBMI03.

[8] A. Enright, S. V. Dongen, and C. Ouzounis. An efficient algorithm for large-scale detection of protein families. *Nucleic Acids Research*, 30(7):1575–1584, 2002.

[9] A. P. Ershov. On programming of arithmetic operations. *Com, of the A.C.M*, 1(8):3–6, 1958.

[10] J. Fédou. Nombre de strahler sur les arbres généraux. In G. ALP, editor, *Ecole jeunes chercheur en algorithmique et calcul formel*, may 1999.

[11] P. Flajolet, J. Raoult, and J. Vuillemin. The number of registers required for evaluating arithmetic expressions. *Theoritical Computer Science*, (8):99–125, 1979.

[12] R. E. Horton. Erosional development of streams and their drainage basins; hydro-physical approach to quantitative morphology. *Bulletin of the Geological Society of America*, 56:275–370, 1945.

[13] G. M. I. Herman, M.S. Marshall. Density functions for visual attributes and effective partitioning in graph visualization. In S. Roth and D. Keim, editors, *IEEE Information Visualization Symposium*, pages 49–56. IEEE CS Press, 2000.

[14] H. Kawaji, Y. Yamaguchi1, H. Matsuda, and A. Hashimoto. A graph-based clustering method for a large set of sequences using a graph partitioning algorithm. In *Genome Informatics*, volume 12, pages 93–102, 2001.

[15] R. Koschke and T. Eisenbarth. A framework for experimental evaluation of clustering techniques. In I. C. S. Press, editor, *International Workshop on Program Comprehension*, pages 201–210, 2000.

[16] D. Maio, D. Maltoni, and S. Rizzi. Dynamic clustering of maps in autonomous agents. *Transactions on Pattern Analysis and Machine Intelligence*, 18(11):1080–1091, 1996.

[17] S. Mancoridis, B. Mitchell, Y. Chen, and E. Gansner. Bunch: A clustering tool for the recovery and maintenance of software system structures. In *International Conference on Software Maintenance*, pages 50–62. IEEE Computer Society, 1999.

[18] S. Mancoridis, B. Mitchell, C. Rorres, Y. Chen, and E. Gansner. Using automatic clustering to produce high-level system organizations of source code. In *International Workshop on Program Comprehension*. IEEE Computer Society, 1998.

[19] S. Rizzi. Genetic operators for hierarchical graph clustering. *Pattern Recognition Letters*, 19(14):1293–1300, 1998.

[20] B. Shneiderman. The eyes have it: A task by data type taxonomy for information visualization. In Boulder, editor, *IEEE Conference on visual languages*, pages 336–343, 1996.

[21] A. N. Strahler. Hypsometric (area-altitude) analysis of erosional topology. *Bulletin of the Geological Society of America*, 63:1117–1142, 1952.

[22] V. Tzerpos and R. Holt. Mojo : A distance metric for software clustering. In I. C. Society, editor, *6th Working Conference on Reverse Engineering*, pages 187–193, October 1999.

[23] V. Tzerpos and R. Holt. On the stability of software clustering algorithms. In I. C. Society, editor, *8th International Workshop on Program Comprehension*, pages 211–220, June 2000.

[24] G. Viennot, G. Eyrolles, N. Janey, and D.Arques. Combinatorial analysis of ramified patters and computer imagery of trees. In *SIGGRAPH Conference*, volume 23 of *Computer Graphics*, pages 31–40, 1989.

[25] T. A. Wiggerts. Using clustering algorithms in legacy systems remodularization. In *4th Working Conference on Reverse Engineering*, pages 33–43. Institute of Electrical and Electronics Engineers, 1997.

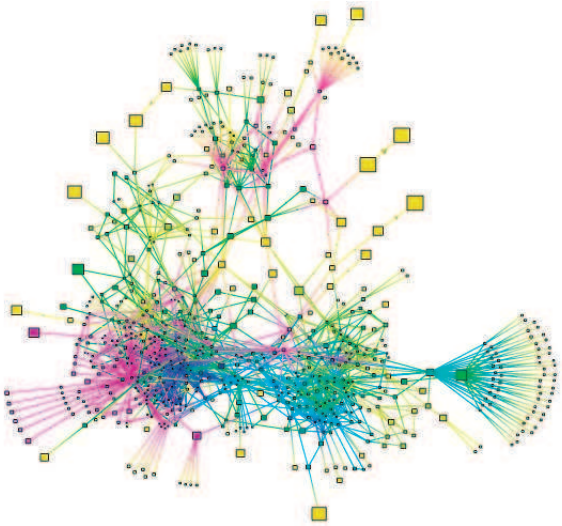

Figure 5. Include graph of the Tulip software.

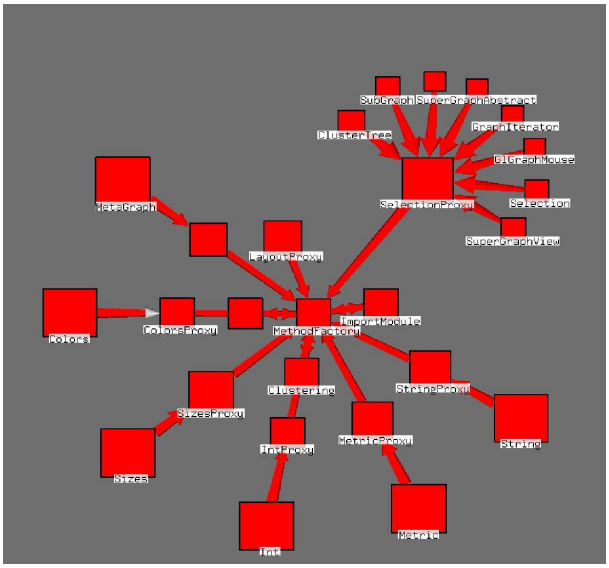

Figure 7. Tulip types

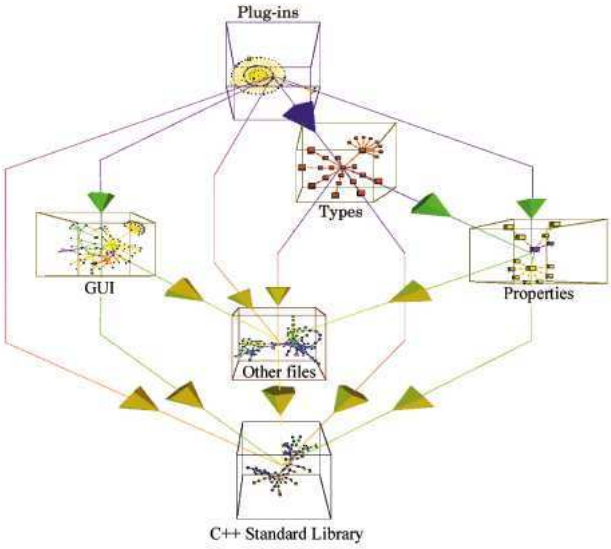

Figure 6. Quotient graph .

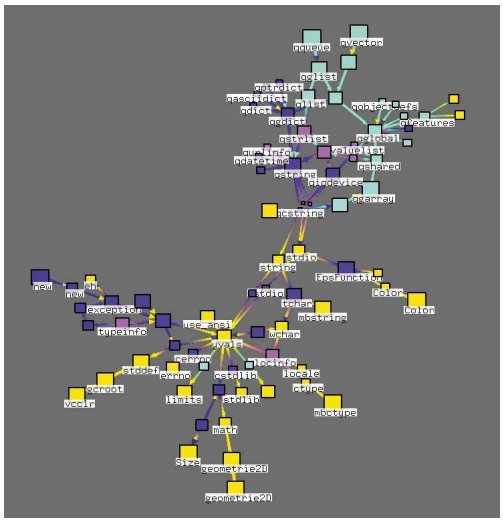

Figure 8. C++ library

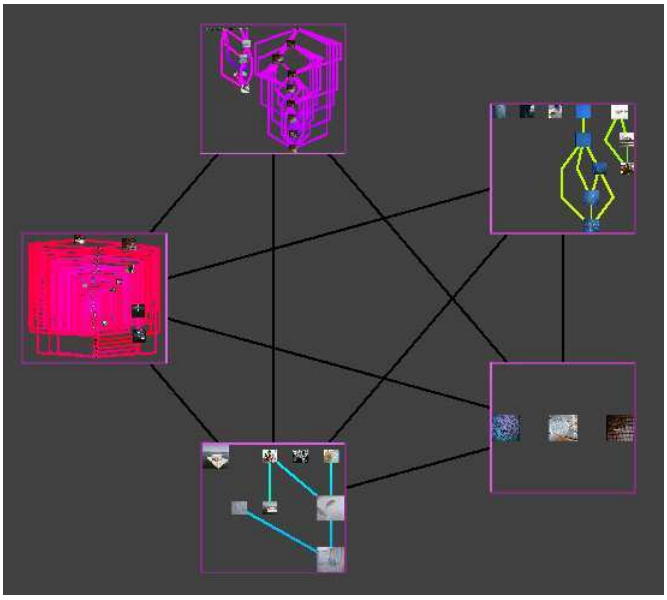

Figure 9. The 5 clusters of "Aquaculture".

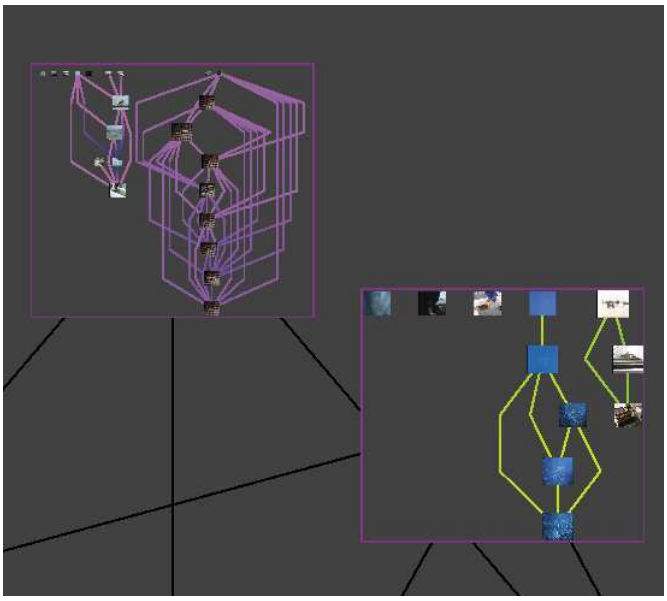

Figure 10. Zoom on clusters of "Aquaculture".

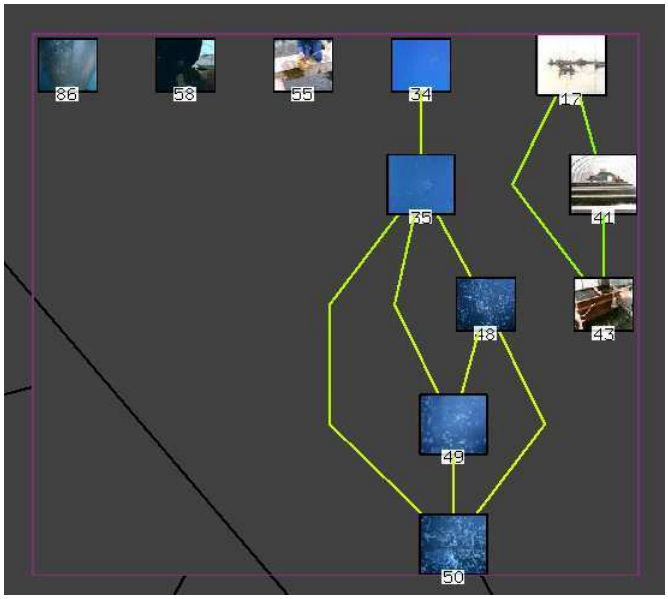

Figure 11. Navigating in a cluster of "Aquaculture".