ΟΠΤΙΚΟΠΟΙΗΣΗ ΠΛΗΡΟΦΟΡΙΑΚΩΝ ΟΙΚΟΣΥΣΤΗΜΑΤΩΝ ΜΕ ΚΥΚΛΙΚΕΣ ΜΕΘΟΔΟΥΣ ΑΠΕΙΚΟΝΙΣΗΣ ΓΡΑΦΗΜΑΤΩΝ

Η
ΜΕΤΑΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ ΕΞΕΙΔΙΚΕΥΣΗΣ

Υποβάλλεται στην

ορισθείσα από την Γενική Συνέλευση Ειδικής Σύνθεσης
του Τμήματος Μηχ. Η/Υ και Πληροφορικής
Εξεταστική Επιτροπή

από την

Ευθυμία Κοντογιαννοπούλου

ως μέρος των Υποχρεώσεων

για τη λήψη

του

ΜΕΤΑΠΤΥΧΙΑΚΟΥ ΔΙΠΛΩΜΑΤΟΣ ΣΤΗΝ ΠΛΗΡΟΦΟΡΙΚΗ

ΜΕ ΕΞΕΙΔΙΚΕΥΣΗ ΣΤΟ ΛΟΓΙΣΜΙΚΟ

Ιανουάριος 2014

Table of Contents

CHAPTER 1. Introduction 1

CHAPTER 2. Related Work 3

2.1 Visualization Fundamentals 4

2.2 Software Visualization 6

2.3 Graph Drawing 7

2.3.1 Circular Methods 7

2.3.2 Multi-Circular Clustered Graph Visualizations 8

2.4 Relationship of our Approach to State of the Art 9

CHAPTER 3. Graph Layout Methods for Data-Intensive Ecosystems 11

3.1 Overview of our Method 11

3.2 Clustering of Modules 13

3.3 Cluster Preprocessing 15

3.4 Layout of Cluster Circle(s) 17

3.4.1 Circular cluster placement with variable angles 18

3.4.2 Clusters on Concentric Circles 24

3.4.3 Clusters on Concentric Arcs 27

3.5 Layout of Nodes inside a Cluster 29

3.6 Refactoring Hecataeus: The Case of Zoom-In for Modules 35

3.6.1 Tabbed Pane 38

3.6.2 Overview Map 39

CHAPTER 4. Experiments 43

4.1 Experimental Method 43
4.2 Aesthetic Criteria 44
4.3 Assessment of Objective Criteria 48
4.4 Comparison to Alternative Methods 56
CHAPTER 5. Conclusions and Future Work 61
5.1 Conclusions 61
5.2 Future Work 62
References 67
Short CV 88

List of Tables

Table 1. Datasets Used (R: Relations, V: Views, Q: Queries, E: Edges) 44
Table 2 Aesthetic criteria and how we address them.....46
Table 3 How the "Visual Information Seeking Mantra" is applied in our implementation. ...47
Table 4 Objective measures for all four data sets 48
Table 5 Area occupied by graph (pixels^2).....51
Table 6 Percentage of occupied area by graph (pixels^2) 51
Table 7 Average time needed for clustering and visualization in milliseconds 53

List of Figures

Fig. 1 Alternative visualizations for the ecosystem of Drupal (an open-source Content Management Platform). Left: Circular layout; Right: Layout based on Concentric Circles. 1
Fig. 2 Alternative visualizations for Drupal. Left: Concentric Arcs. Right: zoom in a cluster of Drupal. 2
Fig. 3 Example of module similarity 12
Fig. 4. Clustering algorithm 14
Fig. 5. Algorithm for the identification of circles, along with their nodes and radii. 16
Fig. 6 A cluster from the BioSQL ecosystem 16
Fig. 7 Calculating the radius R of the embedding circle 18
Fig. 8 Calculating angle $\varphi/2$ for the case where ρ is smaller than R. 19
Fig. 9 Calculating φusing the law of cosines 20
Fig. 10 Calculating $\varphi/2$ for the case where ρ is greater than R 20
Fig. 11 Original assignment of clusters to segments 22
Fig. 12 Enlarging R by *w* to apply visual borders between clusters 23
Fig. 13 Visualizations for BioSQL and Drupal with circular cluster placement. 23
Fig. 14 2^k cluster placement 24
Fig. 15 Avoiding overlaps between clusters on subsequent circles 26
Fig. 16 Visualizations for BioSQL and Drupal with concentric cluster placement method. 27
Fig. 17 Visualizations for BioSQL and Drupal with concentric arc placement. 28
Fig. 18 Algorithm for placing nodes in a cluster. 30
Fig. 19 Layout of query nodes in a cluster with respect to the relation node they reference. 31
Fig. 20 Layout of nodes in a cluster from Drupal. 31
Fig. 21 Layout of nodes in a cluster, the case of queries that reference more than one tables. 32
Fig. 22 Decide circles for the view nodes. 34
Fig. 23 Strata S1, S2, S3 as a result of view stratification produced from Algorithm 4. 35

Fig. 24 Initial non clustered layout of Hecataeus 36
Fig. 25 Hecataeus window. 38
Fig. 26 Application window with two viewers: left the detailed clustered view and right the overview map. 40
Fig. 27 Objective measures for all datasets. 48
Fig. 28 The area used by the graph for every layout. 52
Fig. 29 The connection between number of the clusters and their size with the area they cover. 53
Fig. 30 Time needed for clustering. 54
Fig. 31 Average time needed for visualization. 54
Fig. 32 Visualization time per method and dataset 55
Fig. 33 BioSql visualized via a circular algorithm by Jung. 57
Fig. 34 BioSql visualized with FR algorithm by Jung. 57
Fig. 35 BioSql visualized with ISOM algorithm by Jung. 58
Fig. 36 BioSql visualized with KK algorithm by Jung. 58
Fig. 37 BioSql visualized with a spring layout algorithm by Jung. 59
Fig. 38 First variation of concentric circles method for Drupal data set. 63
Fig. 39 Second variation of concentric circles method for Drupal dataset 64
Fig. 40 Spiral layout for Drupal data set. 65
Fig. 41 Part of Drupal create table statements. 70
Fig. 42 Modified create table statements from Drupal data set to become parsable. 71
Fig. 43 Sample queries from Drupal data set. 72
Fig. 44 A sample of modified queries to be parsable from taxonomy folder. 73
Fig. 45 BioSql data set visualized with the circular method. 74
Fig. 46 BioSql data set visualized with the concentric circle method. 75
Fig. 47 BioSql data set visualized with the concentric arcs method 76
Fig. 48 Biggest cluster of BioSql data set. 77
Fig. 49 Drupal data set visualized with the circular method. 78
Fig. 50 Drupal data set visualized with the concentriccircles method. 79
Fig. 51 Drupal data set visualized with the concentric arcs method. 80

Fig. 52 Biggest cluster form Drupal data set. 81
Fig. 53 ZenCart data set visualized with the circular method. 82
Fig. 54 ZenCart data set visualized with the concentric circles method. 83
Fig. 55 ZenCart data set visualized with the concentric arcs method. 84
Fig. 56 OpenCart data set visualized with the circular method. 85
Fig. 57 OpenCart data set visualized with the concentric circles method. 86
Fig. 58 OpenCart data set visualized with the concentric arcs method. 87

Abstract

Efthimia Kontogiannopoulou.
MSc, Computer Science Department, University of Ioannina, Greece.
Visualization of Data-Intensive Information Ecosystems Via Circular Methods.
Supervisor: Panos Vassiliadis

Data-intensive ecosystems are collections of databases along with software applications that are built on top of them. The main characteristic of such systems is their strong dependence on the underlying layer of data. The database layer not only facilitates the surrounding applications of the information system (scripts, web forms, stored procedures, spreadsheets, or any other client application) but also, deeply affects their architecture and maintenance. *The research question that this Thesis addresses is the provision of a visual map of the ecosystem that can be exploited by all the involved stakeholders in order to show the correlation of the developed code to the underlying database and support operations like program comprehension, impact analysis (for potential changes at the database layer), documentation etc. To provide this visual map, we represent the internal structure of ecosystems as a graph, we cluster its nodes and we visualize the clustered graph with three circular layout methods.*

Περίληψη στα Ελληνικά

Ευθυμία Κοντογιαννοπούλου.
MSc, Τμήμα Πληροφορικής, Πανεπιστήμιο Ιωαννίνων.
Οπτικοποίηση Πληροφοριακών Οικοσυστημάτων με Κυκλικές Μεθόδους Απεικόνισης Γραφημάτων
Επιβλέπων καθηγητής: Παναγιώτης Βασιλειάδης.

Τα οικοσυστήματα δεδομένων αποτελούνται από βάσεις δεδομένων και εφαρμογές οι οποίες εξυπηρετούν ερωτήσεις και συνεπώς εξαρτώνται σε μεγάλο βαθμό από αυτές. Το κύριο χαρακτηριστικό αυτών των εφαρμογών (scripts, web forms, stored procedures, υπολογιστικά φύλλα, κ.ά.) είναι η ισχυρή εξάρτησή τους από το υποκείμενο στρώμα των δεδομένων, το οποίο, όχι απλώς είναι απαραίτητο για τη λειτουργία των εφαρμογών, αλλά επηρεάζει και την αρχιτεκτονική του κώδικα και την συντήρησή του. Το θέμα αυτής της εργασίας είναι η παροχή ενός οπτικού χάρτη του οικοσυστήματος που απεικονίζει την συσχέτιση του πηγαίου κώδικα με την βάση δεδομένων και μπορεί να υποστηρίξει λειτουργίες, όπως η κατανόηση του προγράμματος, η ανάλυση των επιπτώσεων (για ενδεχόμενες αλλαγές στο επίπεδο της βάσης δεδομένων), η τεκμηρίωση, κλπ. Για την παροχή αυτού του οπτικού χάρτη, αναπαριστούμε την εσωτερική δομή των οικοσυστημάτων ως γράφημα το οποίο απεικονίζουμε με τρεις κυκλικές μεθόδους διάταξης.

CHAPTER 1. Introduction

Data-intensive ecosystems are conglomerations of one or more databases along with software applications that are built on top of them. The main characteristic of such systems is their strong dependence on the underlying layer of data. The database layer not only facilitates the surrounding applications of the information system (scripts, web forms, stored procedures, spreadsheets, or any other client application) but also, deeply affects their architecture and maintenance. The research question that this Thesis addresses is the provision of a visual map of the ecosystem that can be exploited by all the involved stakeholders (database administrators, designers, application developers, managers, testers, etc) in order to show the correlation of the developed code to the underlying database and support operations like program comprehension, impact analysis (for potential changes at the database layer), documentation etc.

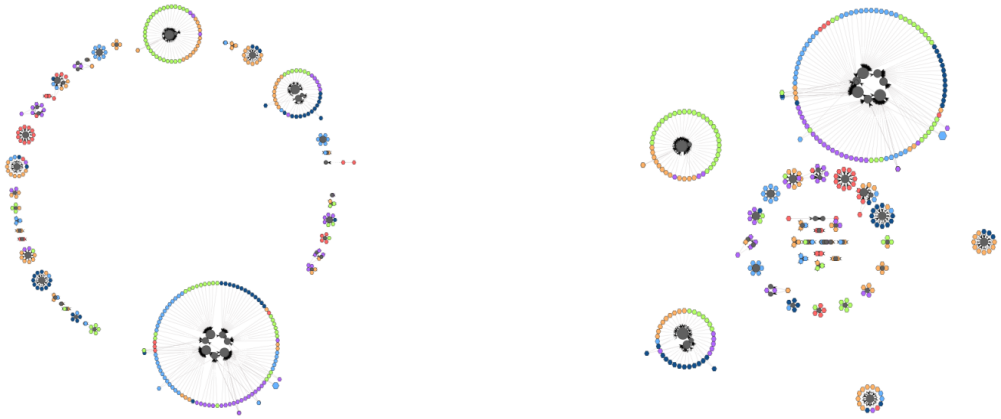


Fig. 1 Alternative visualizations for the ecosystem of Drupal (an open-source Content Management Platform). Left: Circular layout; Right: Layout based on Concentric Circles.

To highlight the internal structuring of the ecosystem and the co-existence and interdependence of software modules and parts of the database schemata, we visualize the ecosystem as a graph where all modules are modeled as nodes of the graph and the provision of data from a database module –e.g., a table—to a software module is denoted by an edge. To detect "regions" of the

graph with dense interconnections (and to visualize them accordingly) we cluster the ecosystem's nodes. Then, we employ three circular graph drawing methods for the visualization of the graph (see Fig. 1). Our first method, *Circular Layout*, places all clusters on an embedding "cluster" circle. Our second method, to which we refer as *Concentric Circles*, instead of employing a single circle for the clusters, splits the space in layers of concentric circles of clusters. Finally, our last method, *Concentric Arcs*, uses arcs instead of concentric circles. In all our methods, the internal visualization of each cluster involves the placement of relations, views and queries in concentric circles, in order to further exploit space and minimize edge crossings.

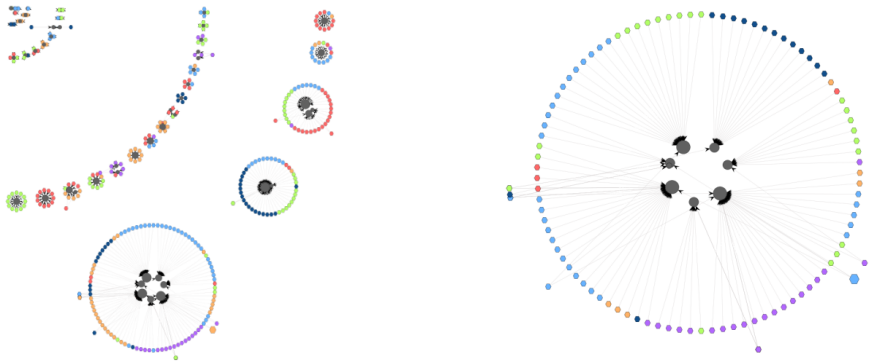


Fig. 2 Alternative visualizations for Drupal. Left: Concentric Arcs. Right: zoom in a cluster of Drupal.

The problem is novel and the state of the art does not sufficiently address it -- see Chapter 2 for a discussion) and Sec. 4.4 for a visual demonstration. To the best of our knowledge, *this is the first time that visual "maps" for the particular case of data-intensive information systems are produced in a principled manner that addresses aesthetic and objective layout criteria*. Moreover, a second contribution of this thesis is the exploration of new ground in the area of circular graph drawing methods by *introducing concentric and arc-based methods for clustered graphs with guarantees on cluster disjointness*.

CHAPTER 2. Related Work

2.1 Visualization Fundamentals
2.2 Software Visualization
2.3 Graph Drawing
2.4 Relationship of our Approach to the State of the Art

In this chapter we cover the fundamentals and research efforts that were related to the problem we study. The most related areas are (a) visualization fundamentals (covered in section 2.1), (b) software visualization (covered in section 2.2) and (c) graph drawing methods (covered in section 2.3).

The basic graph drawing problem simply put is given a set of nodes with a set of edges representing relations between nodes calculate the position of the nodes. Graph visualization techniques among others have to respect a set of principles, the aesthetic criteria. The aesthetic criteria make a graph easier for humans to understand. For example nodes and edges should be evenly distributed, edges should have the same length, edge crossings should be kept to minimum and many other aesthetic rules. However not all aesthetic rules have the same importance. Some authors demonstrate that reducing the edge crossings is by far the most important aesthetic, while minimizing the number of bends and maximizing symmetry have a lesser effect. Other authors report differences in the perception of a graph depending on its layout.

Another issue we take under consideration when visualizing graphs is their size. The size of the graph to visualize is very important in graph visualization. Large graphs cause several problems. If the number of elements to visualize is large it can compromise performance or even exceed the limits of the viewing area. Few systems can claim to deal effectively with thousands of nodes

although graphs with this order of magnitude appear in a wide variety of applications. The size of a graph can make a normally good layout algorithm completely unusable. In fact, a layout algorithm may produce good layouts for graphs of several hundred nodes, but this does not guarantee that it will scale up to several thousand nodes. When visualizing graphs, many constraints are expressed in the form of aesthetic rules. Aesthetic rules have a huge impact on the graph layout. An effective graph drawing algorithm should respect a set of aesthetic criteria regarding the distribution of vertices and edges, the edge length and crossings, the complexity of the layout, the proximity of visual elements in regard to their similarity, the avoidance of visual clutter and many other criteria.

2.1 Visualization Fundamentals

The fundamental concepts that govern user perception of visually demonstrated information have been investigated by the Gestalt school of psychology founded in 1912 and can be summarized as follows [Ware04]:

- *Proximity* - objects close to each other tend to be perceived as similar.
- *Similarity* - objects of the same shape, color, orientation and size are perceived as similar by individuals.
- *Connectedness* - to express semantic relationship among visually connected objects.
- *Closure* - the eye tends to create perceptions of closed space, even if they do not exist -- best served when the depicted objects tend to create a "border" around similar objects along with blobs of whitespace.
- *Continuity* - the eye tends to perceive as related objects that are aligned together intersections create the perception of single uninterrupted groups.
- *Symmetry* - as a means to emphasize non-typical behavior or emphasis when symmetry is broken by an object. In principle asymmetry is used for emphasis while symmetry is used in cases where we do not want to target on something specific.
- *Contrast* - creates emphasis in sharp antithesis to the similarity principle. Contrast can be achieved in terms of chromatic, size or shape choices.

- *Proportion* - where an object placed in an area of the visualization is scaled according to its semantic significance, as the difference in proportion creates a visual attraction to the eye

We also take into consideration best practices [Tidw06] closely related to the above Gestalt principles like

- *Clutter avoidance* - the avoidance of noise on the diagram via uninterrupted areas of whitespace that act as separators of the groups of objects
- *Isolation* - to promote emphasis for an object in sharp antithesis to the continuity of the vast majority of the "regular" objects
- *Visual hierarchy* - to denote a semantic hierarchy in the depicted objects
- *Focal points* to guide visual flow (i.e., objects that intentionally stand out in the representation and whose sequence guides the eye in the visual flow of exploring the diagram).

In addition to all these criteria, there are also other principles we have to respect when designing graphical user interfaces as it is important to provide the users with the information they are looking for effectively. As stated by [Shne96] among many visual design guidelines for effective visualizations, the basic principle of Information Visualization can be summarized as the famous "Visual Information Seeking Mantra": Overview first, zoom and filter, then details-on-demand. In further detail, these four attributes represent:

- *Overview*: Gain an overview of the entire collection. Overview strategies include zoomed out views of each data type to see the entire collection plus an adjoining detail view.
- *Zoom* : Zoom in on items of interest. Users typically have an interest in some portion of a collection, and they need tools to enable them to control the zoom focus and the zoom factor. Smooth zooming helps users preserve their sense of position and context. Zooming could be on one dimension at a time by moving the zoombar controls or in two dimensions. A very satisfying way to zoom in is by pointing to a location and issuing a

zooming command, usually by clicking on a mouse button for as long as the user wishes or clicking on a node or edge to view further details.

- *Filter*: filter out uninteresting items.
- *Details-on-demand*: Select an item or group and get details when needed. Once a collection has been trimmed to a few dozen items it should be easy to browse the details about the group or individual items. The usual approach is to simply click on an item to get a pop-up window with values of each of the attributes, also helpful to keep a history of user actions and support other actions the user may need like undo or replay.

In section 4.2 Aesthetic Criteria we explain how we address the above criteria.

2.2 Software Visualization

A particular area of concern for this thesis involves software visualization. Efforts in this area are concerned with properly presenting the structure & dependencies of software modules to a user. Efforts in the past include tree maps and nested tree maps. Resent efforts are following the IDE approach (like Eclipse or Netbeans). Notably we mention the Code Canvas [DeRo10] and the Code Bubbles [BZRK10] projects.

The state of the art on software visualization described in Code Canvas [DeRo10] suggests that an effective visualization display should provide orientation, selection of regions, dynamic feedback. A display that partitions the main visualization area into separate areas, something that requires the use of many layers of visualization helps the user multitask and access lots of information simultaneously as the user can create separate visualization displays by selecting/filtering items on the main visualization display. The state of the art approach on visualization environments suggests that the use of many visualization displays does not necessarily mean that the user should launch a new display to browse details on the filtered items but the use of a zoomable display allows the user to zoom in to view detailed information and zoom out to get overview. The initial visualization display should provide the user with the full

system structure, a map view of the graph allowing the selection of items of interest by zooming into them.

In Code Bubbles [BZRK10] the authors propose a novel user interface that is based on collections of lightweight editable fragments, which they call bubbles, when the bubbles are grouped together they form concurrently visible working sets. Again the authors of this paper suggest that the user needs to multitask effectively. They argue in favor of a new approach, where the IDE shows multiple editable fragments simultaneously, letting the user see and work with complete working sets. The result reduces navigations like switching among different views using tabs, forward/back buttons, etc. that are proved to occupy lots of the programmers time. The evaluation of their approach showed that users were able to perform complex code understanding tasks significantly more efficiently when using bubbles than when using other IDEs like Eclipse due to reduced navigation.

In our implementation we enable multitasking by providing the user with the option to create many tabs to view parts of the graph; we also provide the user with an overview map to get a more general view with less details of the graph simultaneously without the need of extra navigation and the initial view of the graph provides the user with the full system structure.

2.3 Graph Drawing

Layout algorithms can be categorized with respect to type of layout they generate. Among the many effective ways to visualize a graph our related work focuses on the following categories: Circular Methods and Multi-Circular Clustered Graph Visualizations.

2.3.1 Circular Methods

In terms of related work, the research that mostly pertains to our method involves circular graph drawing. This is due to the increased ability of circular methods to clearly demonstrate natural group structures – clusters – within the overall graph [SixT06]. [SixT06] proposes a technique for

producing circular drawings, using fixed angles on biconnected graphs with the goal of minimizing edge crossings. The method places (a) edges towards the circumference of the embedding circle and (b) the neighbors of a node as close as possible to the node. [Misu06] proposes a method for drawing bipartite graphs in circular layouts. In this method, the nodes of the graph are divided into two groups, the "anchor nodes" that are arranged on the circumference of a circle and "free nodes" that are positioned in the circular disk in relation to the adjacent anchor maps. A simulated annealing algorithm provides the final graph arrangement via the iterative computation of a cost for misplacing free nodes with respect to anchor nodes.

2.3.2 Multi-Circular Clustered Graph Visualizations

[ItKM10] in *Drawing Clustered Bipartite Graphs in Multi-Circular Style* described a way to visualize clustered bipartite graphs in with multi circular methods. They expanded their previous single circle layout we discussed above and they proposed a technique in which anchor nodes in a bipartite graph are arranged in a hierarchal multi-circular layout with different circles for each cluster placed on the periphery of a single big circle. They also develop their method to respect certain aesthetic criteria like minimizing edge crossings, edge lengths and increase the area efficiency as high as possible.

In *RADAR* [Vass11] concentric layouts are used to visualize graphs. According to this work each node has a type and different types on nodes are placed on different concentric circles. The main parts of the *RADAR* algorithm consist of (I) placing relation nodes in the innermost circle of the graph and arranges query nodes around them, (II) distributing query nodes at appropriate angles around the relations circle such that each query is close in terms of angle as possible to the relation it accesses. (III) de-cluttering the graph by placing query nodes of similar degree in concentric cycles of increasing distance from the center, (IV) resolving problems of conflicts when query nodes should be placed closely in the graph by slightly shifting the conflicting nodes. The *RADAR* method possesses several good properties such as hiding a part of the graphs noise

generated by edge crossings and it allows the progressive drawing of the graph. This work is very similar to ours in the case of node placement inside a cluster.

In *Pining Balloons with Perfect Angles and Optimal Area* [HaSk12] the authors visualize nodes (balloons) with different sizes on a disk. The number of nodes placed on a disk is a power of two. If the number of nodes is not a power of two they place the remaining nodes in a disk concentric to the previous but with larger radius. To avoid overlaps they sort the nodes with ascending radius order and they divide the disk in as many as the number of nodes equal in size partitions such that the biggest node fits perfectly in one partition. Since the nodes they visualize are not connected they don't have to consider any aesthetic criteria regarding the clutter produced by edge crossings. [HaSk12] proposes a technique that uses fixed angles to place disks on the circumference of a circle. The disks are either touching the circumference, or in case their size is greater than the angle that is predefined for them, they are moved further from the circle, till they fit in the predefined angle.

2.4 Relationship of our Approach to State of the Art

Compared to related work, we provide a pre-visualization clustering based on node similarity of our graph and we introduce a method with variant angles for cluster placement on circle (as opposed to fixed angles of the related work) and we also propose two concentric methods. All are methods guarantee that the clusters do not intersect. Moreover, we exploit the inherent stratified nature of nodes (tables, views and queries) for the placement of nodes within clusters (as views are defined over tables and queries over both views and tables). To the best of our knowledge, *this is the first time that visual "maps" for the particular case of data-intensive information systems are produced in a principled manner that addresses aesthetic and objective layout criteria*.

CHAPTER 3. Graph Layout Methods for Data-Intensive Ecosystems

3.1 Overview of our Method
3.2 Clustering of Modules
3.3 Cluster Preprocessing
3.4 Layout of Cluster Circle(s)
3.5 Layout of Nodes inside a Cluster
3.6 Refactoring Hecataeus: The Case of Zoom-In for Modules

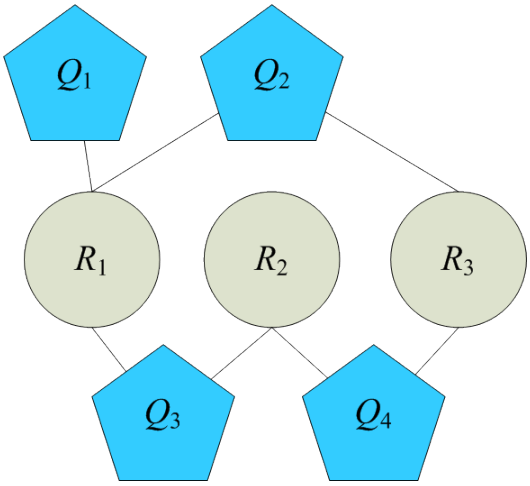
In this Thesis, our main task is to provide the user with several graph visualization layouts through Hecataeus. Hecataeus is system that allows the modeling, visualization, and evolution management of data-intensive ecosystems. In this thesis Hecataeus was re-engineered to support new clustered graph layouts as well as multiple viewing panes to enable the new zoom-in features that provide the user with extra details on demand and filtering of information.

Hecataeus visualizes database constructs as a directed graph $G = (V, E)$ V represents the vertices of the graph and E represents the edges. The components of such a graph are the high level constructs, such as relations, views and queries, which we call modules of the graph and the lower level constructs such as the attributes.

3.1 Overview of our Method

The fundamental modeling pillar upon which we base our approach is the *Architecture Graph* $G(V, E)$ of a data-intensive ecosystem. The Architecture Graph is a skeleton, in the form of graph

that traces the dependencies of the application code from the underlying database. In our previous research [MaVP13], we have employed a detailed representation of the queries and relations involved; in this paper, however, it is sufficient to use a summary of the architecture graph as a zoomed-out variant of the graph that comprises only of modules (relations, views and queries) as nodes and edges denoting data provision relationships between them. Formally, a Graph Summary is a directed acyclic graph $G(V, E)$ with V comprising the graph's module nodes and E comprising relationships between pairs of data providers and consumers.



$$sim(Q_1, Q_3) = \frac{|\{R_1\} \cap \{R_1, R_2\}|}{|\{R_1\} \cup \{R_1, R_2\}|} = \frac{1}{2}$$

$$sim(R_1, Q_3) = \frac{|\{Q_1, Q_2, Q_3\} \cap \{R_1, R_2\}| + 2}{|\{Q_1, Q_2, Q_3\} \cup \{R_1, R_2\}|} = \frac{2}{4}$$

$$sim(R_1, Q_4) = \frac{|\{Q_1, Q_2, Q_3\} \cap \{R_2, R_3\}|}{|\{Q_1, Q_2, Q_3\} \cup \{R_2, R_3\}|} = 0$$

Fig. 3 Example of module similarity

In terms of visualization methods, the main graph layout we use is a circular layout. Circular layouts are beneficial due to a better highlight of node similarity, along with the possibility of minimizing the clutter that is produced by line intersections. We place clusters of objects in the periphery of an embedding circle or in the periphery of several concentric circles or arches. Each cluster will again be displayed in terms of a set of concentric circles, thus producing a simple, familiar and repetitive pattern.

Our method for visualizing the ecosystem is based on the principle of clustered graph drawing and uses the following steps:

1. Cluster the queries, views and relations of the ecosystem, into clusters of related modules. Formally, this means that we partition the set of graph nodes V into a set of disjoint subsets, i.e., its clusters, $C_1, C_2, \dots, C_n$.
2. Perform some initial preprocessing of the clusters to obtain a first estimation of the required space for the visualization of the ecosystem.
3. Position the clusters on a two-dimensional canvas in a way that minimizes visual clutter and highlights relationships and differences.
4. For each cluster, decide the positions of its nodes and visualize it.

3.2 Clustering of Modules

To accomplish a successful visualization it is often required to reduce the amount of visible elements being viewed by placing them in groups. This reduces visual clutter and improves user understanding of the graph as it applies the principle of proximity: similar nodes are placed next to each other. To this end, in our approach we use clustering to group objects with similar semantics in advance of graph drawing.

We have implemented an average-link agglomerative clustering algorithm [Dunh02] described in (Fig. 4) which works as follows.

- First, we compute the distances for every pair of nodes in the graph.
- Then, we iteratively perform cluster merging:
    i. we find the minimum distance pair of clusters,
    ii. we merge the components of the pair into a new cluster, and,
    iii. we calculate the new distances.

This process (Fig. 3) starts with each node being a cluster on its own and stops when the minimum distance of all pairs of clusters is greater than a user-defined threshold of cluster distance (due to the user interface, a threshold is always set).

Algorithm 1. Clustering

Input: *G* : all the graph objects (relations, queries, views), list with solutions (initially every object as a cluster) *T*: the user defined threshold for the distance of two clusters (below which the user deems that the merge of the clusters is without meaning)

Variables: *mindist*: the min distance between clusters

Output: *C:* a set of clusters

```
Begin
1.  Create a set C = { {t1}, {t2}, …, {tn} } with all the objects of G as clusters
2.  Do
3.      mindist = ∞
4.      For each pair ci , cj, i ≠ j
5.          Compute pairwise distances between them
6.          If a pair has smaller distance than mindist
7.              Update mindist with smaller distance
8.              Update mindist pair
9.          End if
10.     End for
11.     Merge mindist pair
12.     Add pair to C
13.     Remove mindist objects from C
14.     If mindist >= T return C
15. While number of clusters != 1
16. Return C
End
```

Fig. 4. Clustering algorithm

The distance function used in our method evaluates node similarity on the grounds of common neighbors. So, for nodes of the same type (i.e., the similarity of two queries, or the similarity of two tables), similarity is computed via the Jaccard formula, i.e., the fraction of the number of common neighbors over the size of the union of the neighbors of the two modules. When it comes to assessing the similarity of nodes of different types (like, e.g., a query and a relation), we must take into account whether there is an edge among them. If this is the case, the numerator is increased by 2, accounting for the two participants. Formally, the distance of two modules, i.e., nodes of the graph, M_i, M_j is expressed in formula (1):

$$\mathrm{dist}(\mathrm{M_i}, \mathrm{M_j}) = 1 - \begin{cases} \dfrac{|\mathrm{neighborhood_i} \cap \mathrm{neighborhood_j}|}{|\mathrm{neighborhood_i} \cup \mathrm{neighborhood_j}|}, \text{if } \nexists \text{ edge } (\mathrm{i}, \mathrm{j}) \\ \dfrac{|\mathrm{neighborhood_i} \cap \mathrm{neighborhood_j}| + 2}{|\mathrm{neighborhood_i} \cup \mathrm{neighborhood_j}|}, \text{if } \exists \text{ edge } (\mathrm{i}, \mathrm{j}) \end{cases} \quad (1)$$

or, equivalently, $\mathrm{dist}(\mathrm{M_i}, \mathrm{M_j}) = 1 - \dfrac{|\mathrm{closedNeighborhood_i} \cap \mathrm{closedNeighborhood_j}|}{|\mathrm{neighborhood_i} \cup \mathrm{neighborhood_j}|}$

To clarify the terminology used in the formula (1), we explain the employed terms as follows.

- For a given node v_i in a graph $G(V, E)$, *neighborhood*(v_i) = {v_j, s.t., ∃ edge (v_i, v_j) or (v_j, v_i) in E}
- For a given node v_i in a graph $G(V, E)$, *closedNeighborhood*(v_i) = *neighborhood*(v_i) ∪{v_i}

3.3 Cluster Preprocessing

Before proceeding any further, the method requires the computation of the area that each cluster will possess in the final drawing. As already mentioned, each cluster includes at least three concentric circles: the innermost circle for the relations, an intermediate band of circles for the views (which are stratified by definition, and can thus, be placed in strata) and the outermost band of circles for the queries that pertain to the cluster. The latter includes two circles: a circle of relation-dedicated queries (i.e., queries that hit a single relation) and an outer circle for the rest of the queries (see Fig. 3). This heuristic is due to the fact that in all the studied datasets, there was a vast majority of relation-dedicated queries; thus, the heuristic allows a clearer visualization of how queries access relations and views.

In order to obtain an estimation of the required space for the visualization of the ecosystem, we need to perform two computations (see Fig. 5 for the algorithm). First, we need to determine the circles of the drawing and the nodes that they contain, and second, we need to compute the radius for each of these circles. Then, the outer of these circles gives us the space that this cluster needs in order to be displayed.

Algorithm 2. Circle Identification

Input: a cluster C

Variables: T: the tables of C, V: the views of C, Q: the queries of C, S: a list of strata (to be topologically sorted) over $T \cup V \cup Q$, *nodes:* $T \cup V \cup Q$

Output: a list of circles $K=\{K_0, \ldots, K_n\}$, each annotated with its nodes, *nodes*(K_i), and its radius R_i

Begin

1. Topologically sort T, V and Q and organize their nodes in strata; then, S is a list of strata, $S = T \cup V \cup Q$, with $T= \{S_0\}$, $V= \{S_1, \ldots, S_m\}$, $Q = \{S_{m+1}, S_{m+2}\}$
2. For every stratum S_i of $S = \{T \cup V \cup Q\}$
3. Append a new circle K_i to K, *nodes*$(K_i) = V_i$
4. Compute its radius $R_i = \log(nodes^3) + nodes$

End

Fig. 5. Algorithm for the identification of circles, along with their nodes and radii.

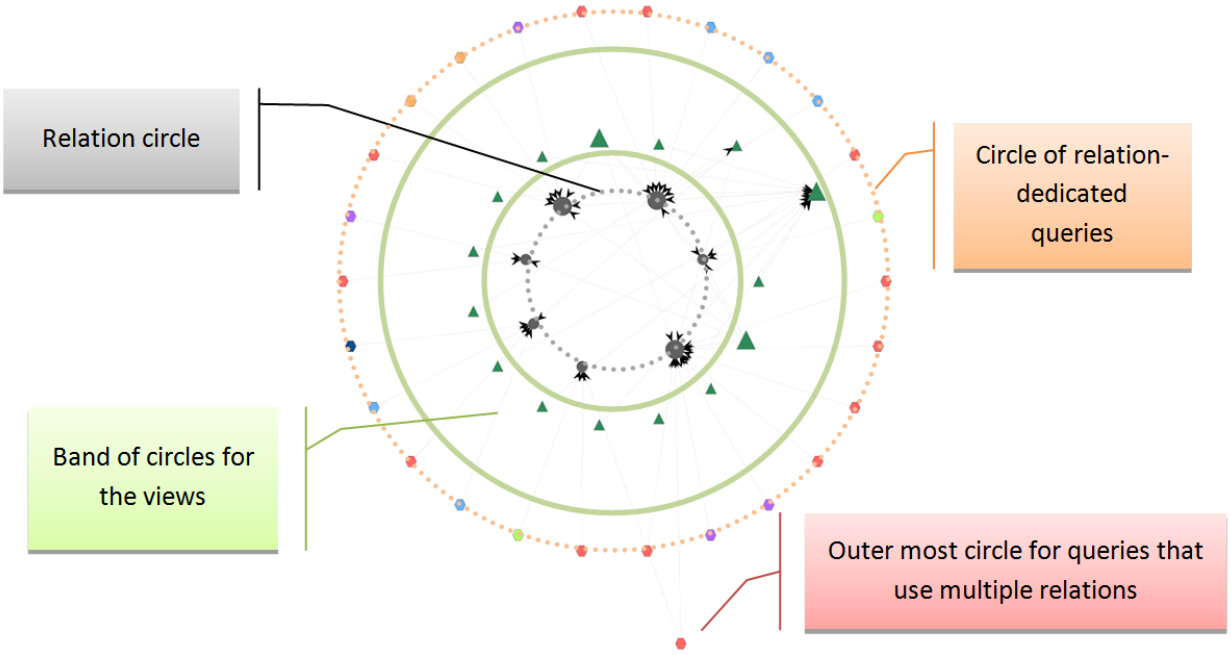


Fig. 6 A cluster from the BioSQL ecosystem

To obtain the bands of views and queries we topologically sort the nodes of the cluster and organize their nodes in strata. For the views, each stratum V_i defines an equivalence class in the graph and includes all the nodes of the graph that depend only from nodes in strata V_j previous to V_i, $j < i$. Of course, relations form the zero-th stratum with no dependencies whatsoever. For queries, we simply split them in two pseudo-strata: (a) relation-dedicated queries and (b) all the rest of the queries. Then, for each stratum, we add a circle, with the radius determined by the formula:

$$R_i = 3 * \log(nodes) + nodes \qquad (2)$$

The rationale for the formula is simple: we need a function that will increase rapidly for a small number of nodes and will not change that much when the number of the nodes gets bigger. This way we make sure that very small clusters will be placed on a visible circle and big clusters will not get enormous. Because of that, we used the logarithmic function described in Formula (2) in which we intentionally added the number of nodes that belong to the circle of the cluster, to ensure that the circles with small number of nodes will obtain a visible area. The scale factor of 3 in the formula was empirically determined by working with the datasets that we have used in our experimentation (see Chapter 4). In Fig. 6 we depict the internal structure of a cluster from the BioSql dataset.

3.4 Layout of Cluster Circle(s)

Once the clusters have been computed and their radius calculated, then it is time to position them on a 2D canvas. We employ a variety of circular layouts for the problem, each with different characteristics. The first layout method places all clusters on a single circle; however, in contrast to existing techniques, we allocate different angles and sectors per cluster. The second layout method places clusters in concentric circles in an attempt to avoid the intermediate empty space of the previous method. Finally, the third method, which is a combination of the previous two methods, utilizes circular arcs instead of circles and assigns different angles for each cluster.

3.4.1 Circular cluster placement with variable angles

In this method, we use a single circle to place circular clusters on. In contrast to the state of the art, we do not use fixed sizes for the sectors of the circle the clusters occupy, but rather, the part of the overall circle that is assigned to each cluster depends on its size (i.e., its number of nodes).

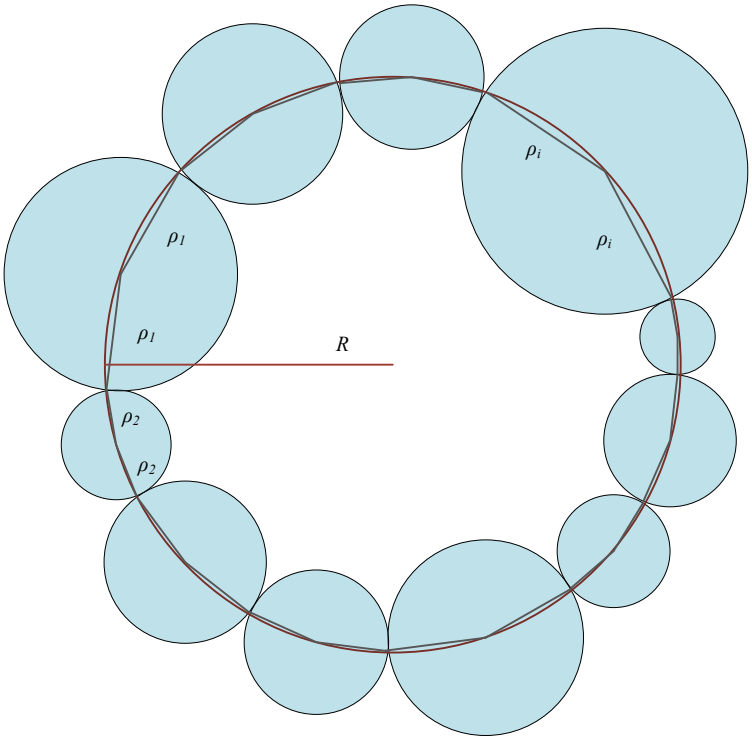


Fig. 7 Calculating the radius R of the embedding circle

As already mentioned, we have already calculated the radius r of each cluster. Given this input, we can also compute *R*, the radius of the embedding circle. We approximate the contour of the inscribed polygon of the circle, computed via the sum of twice the radius of the clusters and then we divide this sum by *2π* to calculate the radius *R* of the embedding circle using equation (3). We approximate the circles periphery, equal to *2πR,* by the sum of edges of the embedded polygon.

$$2\pi R \cong \sum_{i=0}^{|C|} 2\rho_i => R \cong \sum_{i=0}^{|C|} 2 * \rho_i / 2\pi \qquad (3)$$

In Fig. 7 we visually demonstrate how the periphery of a circle is approximated by its embedded polygon, in order to calculate the circle's radius, R.

The next step after calculating the radius of the embedding circle is to assign each cluster to a segment of the circle depending on the cluster's radius (size). Each one of these segments is defined by an angle φ of the embedding circle.

Before calculating the angle φ we have to consider two cases:

- The radius ρ of the cluster we want to place is smaller or equal to the radius of the embedding circle R we calculated above.
- The radius ρ of the cluster we want to place is greater than the radius of the embedding circle R we calculated above.

In the first case, depicted in Fig. 8, we consider the right triangle ABO and based on simple trigonometry we calculate angle $\varphi/2$ as follows

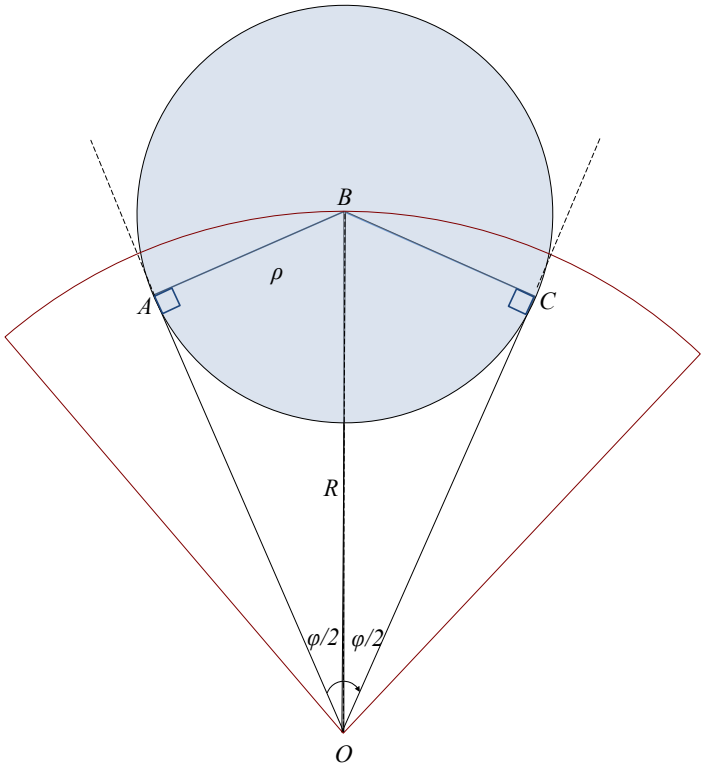


Fig. 8 Calculating angle $\varphi/2$ for the case where ρ is smaller than R.

$$\varphi/2 = \sin^{-1}\left(\frac{\rho}{R}\right) \quad (4)$$

In the second case (see Fig. 10), when ρ is greater than R, Formula 4 is not working correctly since $\sin^{-1}$ does not produce real numbers for values greater than one. To avoid this problem we used the law of cosines. Observe the triangle in Fig. 9. Then, the law of cosines states that $a^2 = b^2 + c^2 - 2bc * cos\varphi$.

Then, the angle φ can be computed as follows

$$cos\varphi = (b^2 + c^2 - a^2)/2bc \quad (5)$$

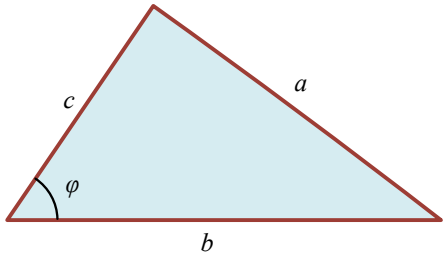


Fig. 9 Calculating φusing the law of cosines

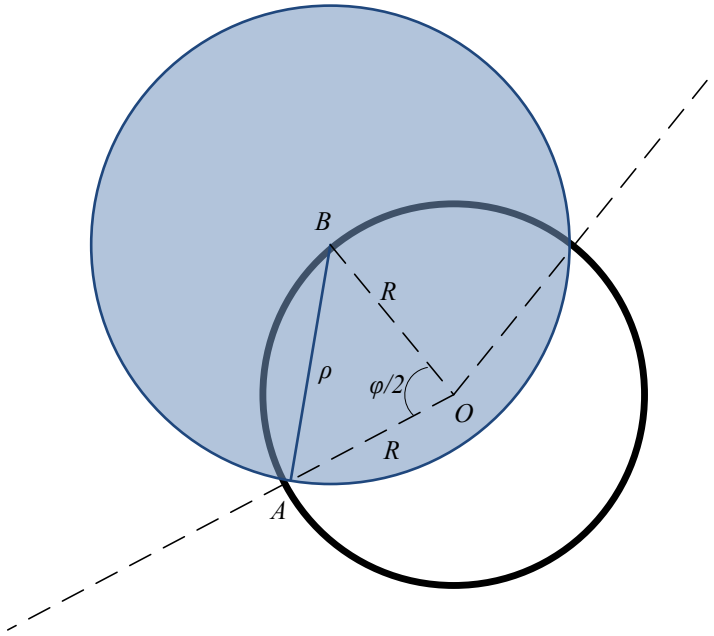


Fig. 10 Calculating φ/2 for the case where ρ is greater than R

To calculate φ in this case, first we need to calculate angle $\varphi/2$. This is formally done in formula (6), which we prove right away. In Fig. 8 we graphically depict the explanation of this calculation.

$$\varphi/2 = \cos^{-1}\left(\frac{2R^2 - \rho^2}{2R^2}\right) \quad (6)$$

Formula (6) is an implication of the law of cosines defined in formula (5). Specifically, let us define the isosceles triangle *ABO* (Fig. 10) with O being the center of the circle and *A*, *C* the limits of the sector on the periphery of the circle. *BO* is the dichotomous of the angle φ of a sector *AOC* and also *BO* is equal to the radius of our circle, so we have constructed the isosceles triangle *AOB* and using the law of cosines (see Fig. 9) with *OA* as *c*, *OB* as *b* and *AB* (which is equal to the radius of the cluster we wish to place) as a, we can calculate $\varphi/2$ via the equation described in (6).

It is noteworthy, that we cannot avoid discriminating the aforementioned two cases: when $\rho < R$, if we use the formula with the law of cosines, the line *OA* of Fig. 8 would not be a tangent line and would actually cut through the disk of the cluster. As already mentioned, in the case of $\rho \geq R$, we cannot produce sin^{-1} for values higher than 1.

We take special care that the layouts of the different clusters do not overlap; to this end, we introduce a white space factor *w* that enlarges the radius *R* of the cluster circle (typically, based on microbenchmarks with the datasets of our experimentation (Chapter 4), we use a fixed value of 1.8 for *w*). To calculate the center coordinates of each cluster we use the radius *R* multiplied by *w*. The physical meaning of *w* is that we enlarge the arc over which we place the cluster via the expansion of the radius we calculated (Fig. 12). Then, *w* is the factor that we multiply the radius with to make sure that clusters will not overlap. Via this expansion of the radius, the arc around which each cluster will be placed is also expanded, thus leaving extra whitespace between the actually exploited parts of the clusters' arcs. This results in producing a *virtual visual border* for the cluster, which is now entirely surrounded by whitespace. We need this visual border for aesthetic criteria as it guarantees that the visual perception of neighboring clusters will not confuse or merge them. Given the above inputs, we can calculate the angle φ that determines the sector of a given cluster for both cases, as well as its center coordinates (*cx*, *cy*) via the following equations:

$$\varphi = 2\cos^{-1}\left(\frac{(2R^2-\rho^2)}{2R^2}\right),$$
$$\varphi = 2\sin^{-1}\left(\frac{\rho}{R}\right) \quad (6)$$
$$c_x = \cos(\varphi/2) * R * w, \quad c_y = \sin(\varphi/2) * R * w$$

To calculate the center coordinates (c_{x_i}, c_{y_i}) of any cluster *i* we use the following method:

$$\mathrm{c_{x_i}} = \cos\left(\sum_{\mathrm{j=0}}^{\mathrm{i}} \varphi_{\mathrm{j}} + \varphi_{\mathrm{i}}/2\right) * \mathrm{R} * \mathrm{w}$$
$$c_{y_i} = \sin\left(\sum_{j=0}^{i} \varphi_j + \varphi_i/2\right) * R * w \quad (7)$$

As we place clusters sequentially over the embedding circle each sector is placed next to the last one and thus, to compute the angle of its dichotomous, over which we place the center of the cluster, we sum the φ_j angles of all the previous clusters.

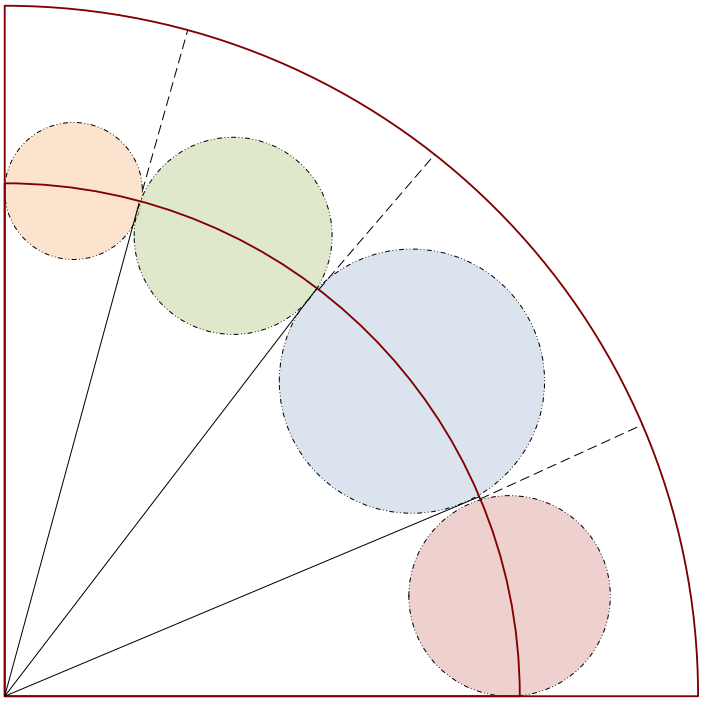


Fig. 11 Original assignment of clusters to segments

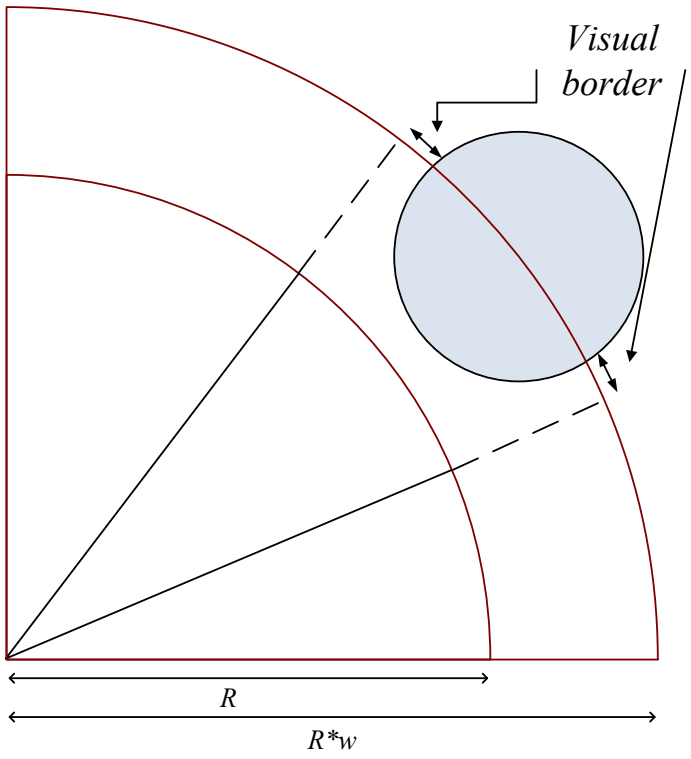


Fig. 12 Enlarging R by *w* to apply visual borders between clusters

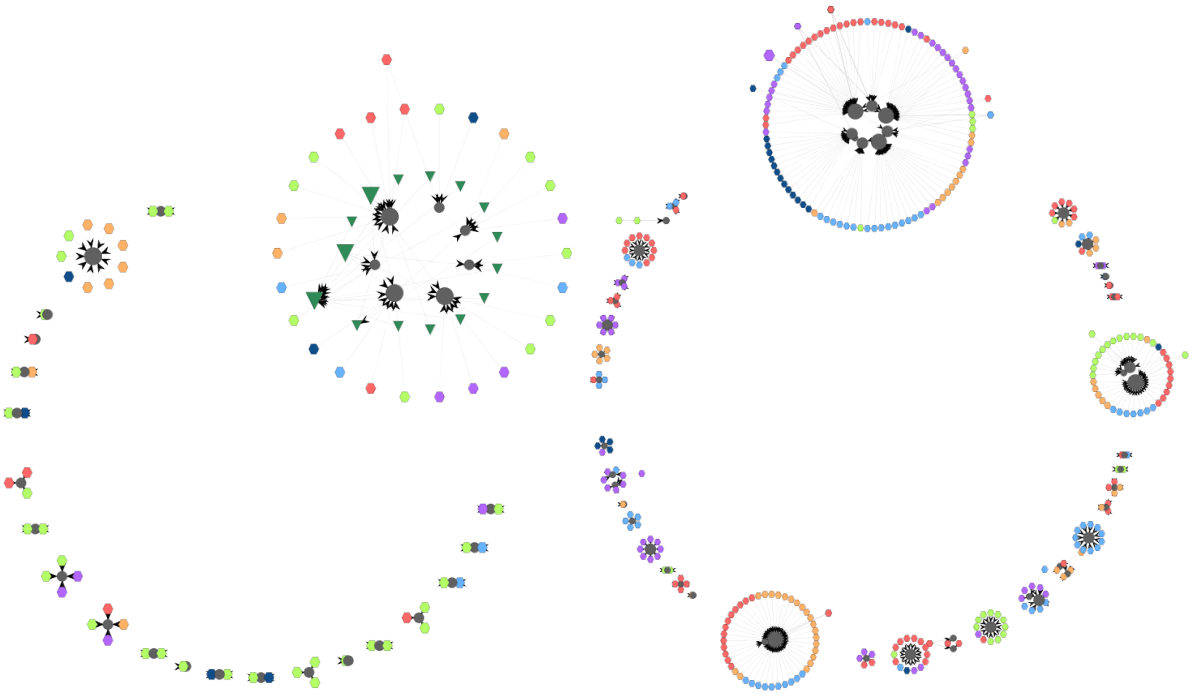


Fig. 13 Visualizations for BioSQL and Drupal with circular cluster placement.

3.4.2 Clusters on Concentric Circles

This method involves the placement of clusters to concentric circles. Each circle includes a different number of segments, each with a dedicated cluster. The proposed method obeys the following sequence of steps:

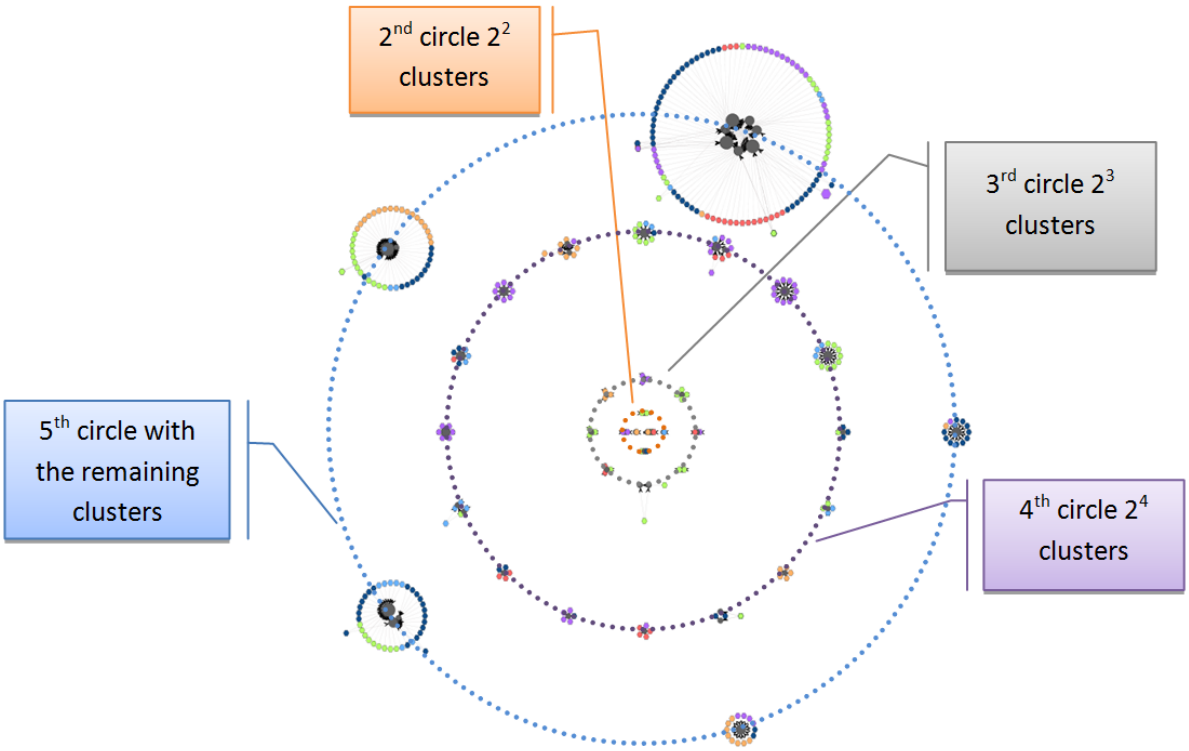


Fig. 14 2^k cluster placement

1. Sort clusters by ascending size in a list L^C
2. While there are clusters not placed in circles
   2.1. Add a new circle and divide it in as many segments as $S = 2^k$ (Fig. 14), with k being the order of the circle (i.e., the first circle has 2^1 segments, the second 2^2 and so on)

2.2. Assign the next S fragments from the list L^C to the current circle and compute its radius according to this assignment

2.3. Add the circle to a list L of circles

3. Draw the circles from the most inward (i.e., from the circle with the least segments) to the outermost by following the list L.

Practically, the algorithm expands a set of concentric circles, split in fragments of powers of 2 (Fig. 14). As the order of the introduced circle increases, the number of fragments increases too ($S = 2^k$), with the exception of the outermost circle, where the segments are equal to the number of the remaining clusters. By assigning the clusters in an ascending order of size, we ensure that the small clusters will be placed on the inner circles, and we place bigger clusters on outer circles since bigger clusters occupy more space.

Again the first step is to calculate the radius for every concentric circle. We also need to guarantee that clusters do not overlap. This can be the result of two problems: (a) clusters of subsequent circles have radii big enough, so that they meet, or, (b) clusters on the same circle are big enough to intersect.

To solve the first problem, we need to make sure that the radius of a circle K_i is larger than the sum of (i) the radius of its previous circle K_{i-1}, (ii) the radius of its larger cluster $R_{max}(C_{ki-1})$ on the previous circle, and (iii) the radius of the larger cluster of the current circle $R_{max}(C_{ki})$. For the second problem, we compute R_i as the encompassing circle's periphery ($2\pi R_i$) that can be approximated as the sum of twice the radii of the circle's clusters (Fig. 7).

Again, to avoid the overlapping of clusters and to apply white space as visual border to distinguish clusters easily, we set the radius of the circle to be the maximum of the two values produced by the aforementioned solutions and we use an additional whitespace factor w to enlarge it slightly (typically, we use a fixed value of 1.2 for w).

In Fig. 15 we depict the first condition we need to be satisfied (i.e. clusters of subsequent circles have radii big enough, so that they do not overlap). In this case we consider the worst scenario: We want to place on circle K_i a cluster C_{ki}, which happens to be the biggest cluster on K_{i} thus it's

radius is $R_{max}(C_{ki})$ and on the previous circle K_{i-1} in layer adjacent position, there is also the biggest cluster of circle K_{i-1} with radius $R_{max}(C_{ki-1})$. In this case to avoid overlaps we need to make sure that the radius R_i of circle K_i is not smaller than the sum of the radii R_{i-1} and $R_{max}(C_{ki})$ and $R_{max}(C_{ki-1})$.

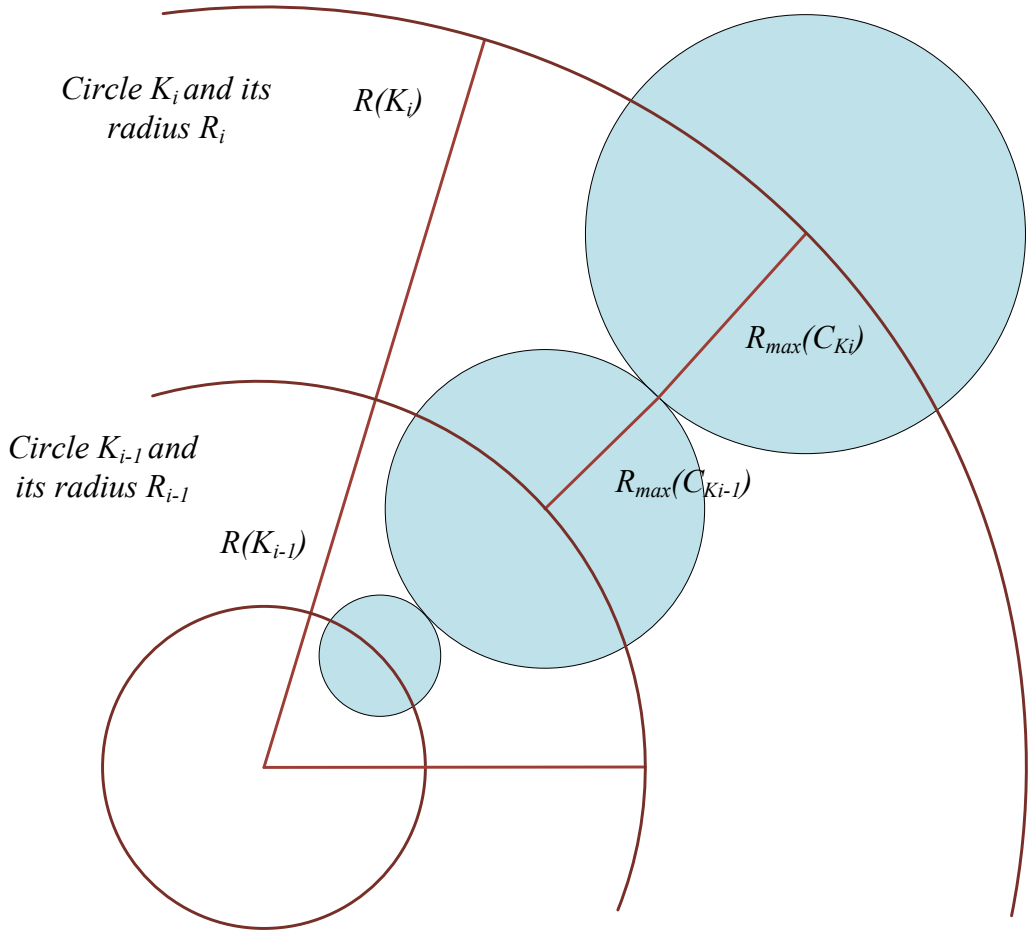


Fig. 15 Avoiding overlaps between clusters on subsequent circles

Before calculating the radius of a circle, we approximate the contour of the inscribed polygon of the circle with the method described in paragraph 3.4.1 and using this contour we calculate minimum value that the radius R_{min} of this circle can take in order to avoid overlaps between clusters on the same circle. Then we calculate the radius of circle K_i with the following method.

$$R(K_i) \quad = w * max \begin{cases} R_{i-1} + R_{max}(C_{K_{i-1}}) + R_{max}(C_{K_i}) \\ \frac{1}{\pi}\sum_{j=1}^{|C|} R(C_{jK_i}), R(C_{jK_i}) : \text{radius of cluster } C_j \text{on circle } K_i \end{cases} \tag{8}$$

In this implementation the angles defining segments of the circle assigned to each cluster are equal for all clusters on the same circle and calculated as follows:

$$\varphi_i = 2\pi / nK_i \tag{9}$$

With φ_i: the angle same for all clusters on circle K_i, n: the number of clusters on circle K_i

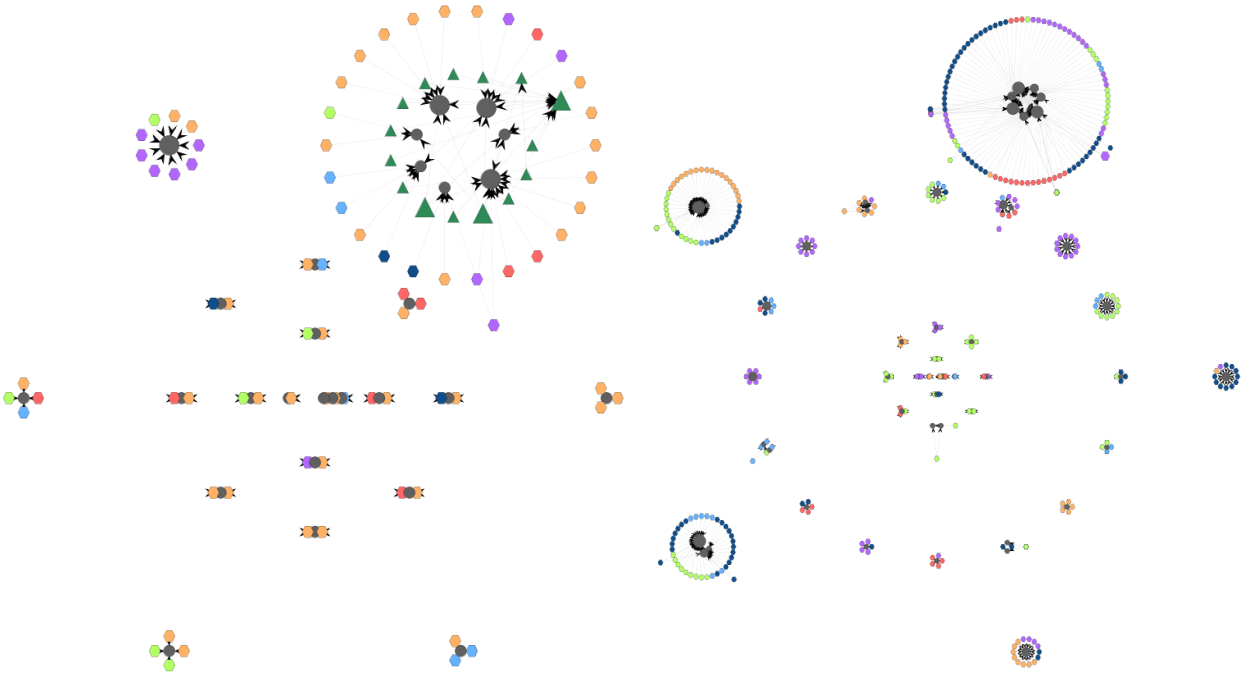


Fig. 16 Visualizations for BioSQL and Drupal with concentric cluster placement method.

3.4.3 Clusters on Concentric Arcs

As an alternative method of laying out the clusters on the canvas, we suggest the concentric arc layout. This method places the clusters in a set of concentric arcs, instead of concentric circles (Fig. 17). This provides better space utilization, as the small clusters are placed in the upper left corner and there is less whitespace devoted to guard against cluster intersection (See Table 6 for

detailed metrics for the area occupied by the graph in all three layouts). Overall, this method is a combination of the previous two methods.

(a) We use the same method for deploying clusters we used in section 3.4.2 with the difference that instead of circles K_i we deploy the clusters on concentric arcs A_i of size $\pi/2$. This way we attempt to reduce white space having a more compact layout. Just like before we place 2^a clusters on the a^{th} arc. In this layout we need to address the same radius calculation problem as in section 3.4.2 (i.e. clusters of subsequent circles have radii big enough, so that they meet, or, clusters on the same circle are big enough to intersect) and for this case we use exactly the same radius optimization technique we used before.

(b) In this layout unlike the 3.4.2Clusters on Concentric layout, the partition assigned to each cluster is proportionate to each size. To calculate the partition size of each cluster, we used the method expressed by equation (6) of section 3.4.1.

Again we use the same way of radius optimization we used in 3.4.2 to avoid cluster overlaps.

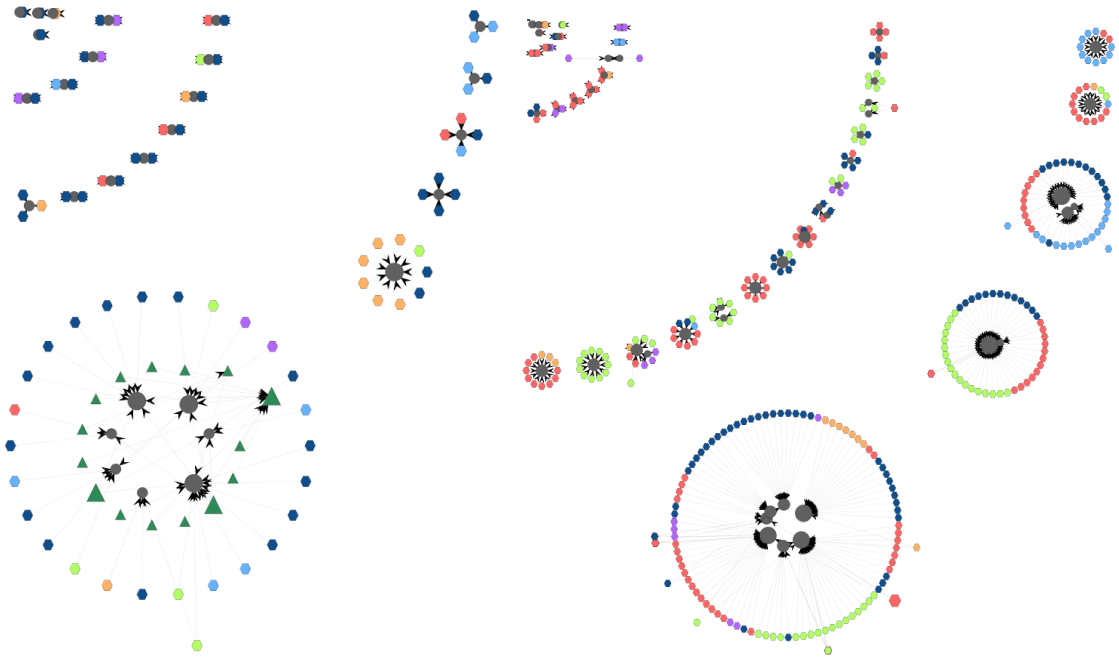


Fig. 17 Visualizations for BioSQL and Drupal with concentric arc placement.

3.5 Layout of Nodes inside a Cluster

The last part of the visualization process involves placing the internals of each cluster within the area designated to the cluster from previous computations. As already mentioned, each cluster is aligned in terms of several concentric circles: an innermost circle for relations, a set of intermediate circles for views and one or more circles for queries, as we previously stated at section 3.3. Now, since the radii of the circles have been computed, what remains to be resolved is the order of nodes on their corresponding circle (see Fig. 18 for the algorithm).

First, we start with relations. We want to place two relations in adjacent positions if they are frequently accessed together by many queries. To this end, we iterate over all the queries of a cluster that access more than one relation and for each combination of relations accessed by a query, we increase a fan-in counter. At the end of this process we sort the combinations with a decreasing order of their fan-in counter in a list. The list is used for sequencing the relations: we start from the most frequent combination and we arrange the relations by the order it identifies; then, we do the same for all the non-placed relations of the following combination, till there are no more combinations. Assume we have the following combinations {*T4*, *T3*}, {*T1*, *T5*} and {*T1*, *T2*, *T3*} in decreasing order of frequency. Then, the final order of the relations will be {*T4*, *T3*, *T1*, *T5*, *T2*}. Exactly the same process is followed to sort the view nodes with respect to the relation nodes they use. As the reader might have noticed, the order in which the queries are placed, with respect to the relation node(s) they use, is not optimal. The nodes are placed in a sequence with a greedy algorithm which has worked well for the experiments we have conducted (Chapter 4). Clearly, a more sophisticated method is part of the future work.

Once the order of the relations is laid out, we compute their coordinates by co-locating them along with their dedicated queries (as already mentioned, these queries come in large numbers practically). We compute the segment that pertains to a relation by assigning a constant $d\varphi$ for each relation-dedicated query. We calculate $d\varphi$ as 2π divided by the number of nodes we want to place on the circle. Then, we place the relation in the middle of this segment and the relation-dedicated queries sequentially in their corresponding circle (Fig. 19).

Algorithm 3. Circle Layout

Input: a cluster *C*

Variables: *T*: the tables of *C*, *V*: the views of *C*, *Q*: the queries of *C*, *S*: a list of (topologically sorted) strata over $T \cup V \cup Q$

Output: an angle φ_i for each node *i* of the cluster *C*

Begin

1. //Order relations
2. Let *L* be a list of pairs $[c,c_n]$, with *c* being a combination of relations hit by queries in the cluster and c_n the frequency of *c*; *L* = ∅
3. Iterate over Q and compute L //for each combination increase a counter
4. Sort *L* in decreasing order
5. Let T* be the list of relations in sorted order; T* = ∅
6. **For** each combination l in L, add its contents to T*, if not already in T*
7. //Order relation-dedicated queries
8. **For** each table ti in T
9. Compute Qd(ti) = the relation-dedicated queries of ti
10. Compute the angle of the segment that pertains to $Qd(t_i)$ $\omega_i = |Qd(t_i)| * d\phi$, $d\phi$ is the angle per node computed as $2\pi / |T|$ (the number of nodes of the circle)
11. Place ti in the middle of its segment, next to the previous: $\phi_i = 0.5*\omega_i + \phi_{i-1}$
12. Place the queries of $Qd(t_i)$ sequentially in their circle
13. //Order views and queries
14. **For** every circle k_i of the cluster's circles (in the order determined by Algorithm 2 in Fig. 5)
15. **For** each node v_j in k_i
16. Sum all the angles of the nodes belonging to the cluster that vj accesses in the previous circles and divide by their number to compute the node's angle ϕj

End

Fig. 18 Algorithm for placing nodes in a cluster.

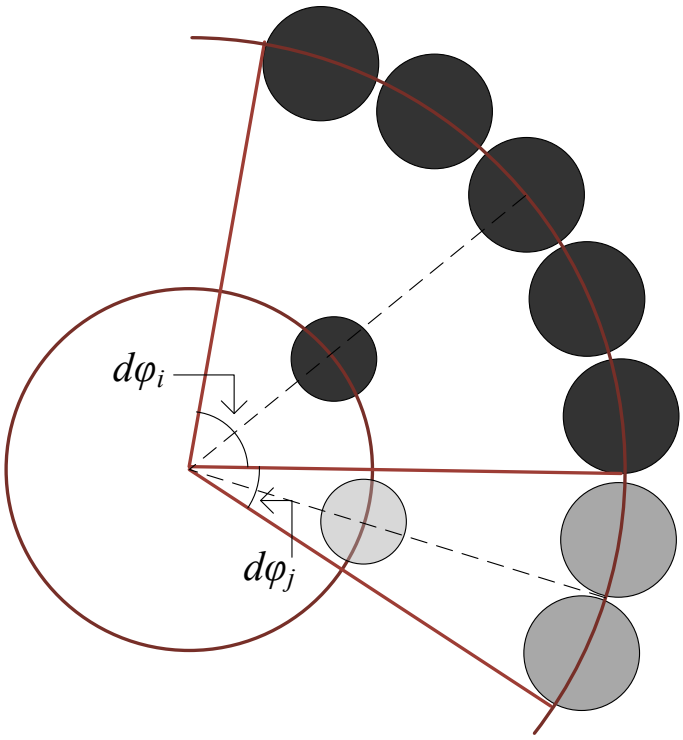


Fig. 19 Layout of query nodes in a cluster with respect to the relation node they reference.

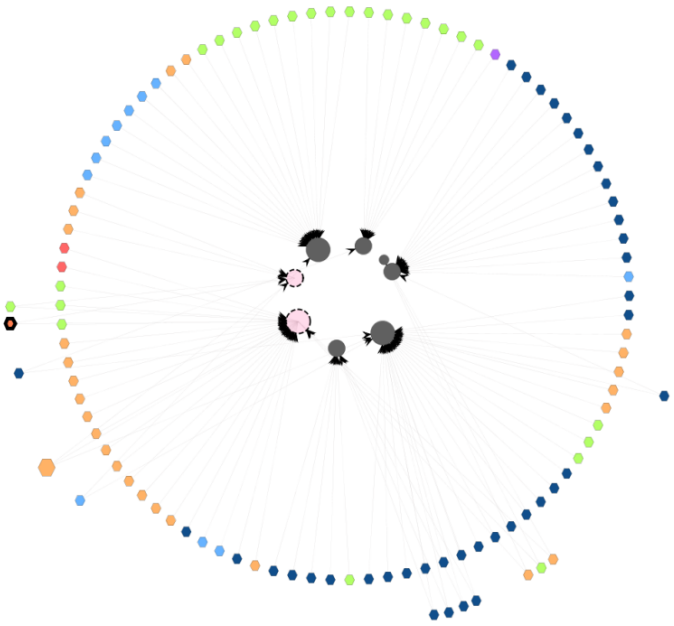


Fig. 20 Layout of nodes in a cluster from Drupal.

Once these nodes have been laid out, we place the rest of the views and queries in their corresponding circle of the cluster. We visit each circle in turn, in the order determined by Algorithm 2 in Fig. 5. This way, we are certain that each node of circle k_i is defined over nodes in the previous circles (and possibly, over nodes in other clusters that do not affect the node's placement). Then, we can follow a traditional barycenter-based method [deTT99] and we place the node in an angle that equals the average value of the sum of the angles of the nodes it accessed (Fig. 21). As it is expected this method assigns to nodes using the same relations the exact same angle causing overlaps of nodes. To avoid this problem every time a node needs to be placed on an occupied position we increase the angle that specifies this position by a very small value δ (in our case δ=0.09 radians).

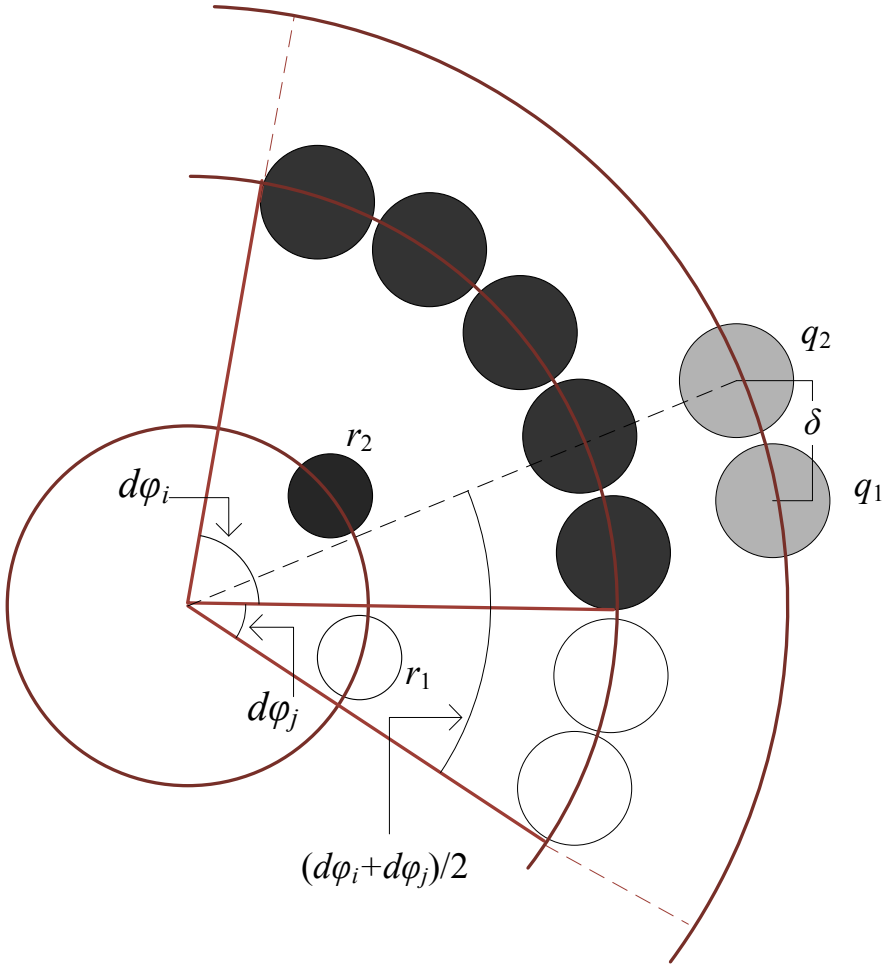


Fig. 21 Layout of nodes in a cluster, the case of queries that reference more than one tables.

Specifically in Fig. 21 we depict the following case: We need to place queries q_1 and q_2 and both of them reference the relation nodes r_1 and r_2. To decide their position we sum the angles $d\varphi_i$ and $d\varphi_j$, r_1 and r_2 use and place q_1 on the corresponding circle with angle $(d\varphi_i + d\varphi_j) / 2$. Then we need to place query q_2 and since it is the second query that references both relations r_1 and r_2, we place it on the corresponding circle with angle $(d\varphi_i + d\varphi_j) / 2 + \delta$ to avoid overlap with query q_1.

As we said before the view nodes are placed on a band of circles or stratums. To decide on which stratum a view should be placed we count the view's incoming edges. View nodes with the same number of incoming edges belong to the same stratum. We compute a stratification of views as follows. In the first stratum we place views with no incoming edges. In the k^{th} stratum we place views that depend on at least one view from stratum k-1 and maybe from smaller strata. To do that we iterate on all view nodes in each cluster and we add them in stratums with respect to their incoming edges, first we start with the view nodes with no incoming edges, we remove them from the list with views and we add them to the corresponding stratum and then, we continue with nodes with more incoming edges until the list with all the views in the cluster is empty (see Algorithm 4 in Fig. 22). In Fig. 23 we depict a simple case of view stratification for six views. S_1, S_2, S_3 represent the different strata where we place view nodes according to Algorithm 4. In the first stratum, S_1 we place the three views that have no incoming edges. Then, we remove their outgoing edges and we search for view nodes with no incoming edges to place in the second stratum S_2, including two views defined over the views of strum S_1. We follow this process until there are no more view nodes to place. In the case of Fig. 23, there is a third stratum, S_3, including a view defined over views belonging to previous strata (notice that the dependency does not need to be confined only to the exactly previous stratum, but it can be defined over a view belonging to any of the previous strata). In the case depicted in Fig. 23, as we have three different strata, we will eventually place the view nodes in three different concentric circles.

Other issues related to the visualization of the clusters are: (a) node shape, (b) node size, (c) node color, (d) edge weight, and (e) edge color. Regarding the node shape, we use different shapes to visually distinguish the different type of nodes. Relation nodes have circular shape, view nodes have triangular shape and query nodes are depicted as hexagons. Moreover, we scale the size of nodes according to their node degree (i.e., their interconnectedness with other nodes); thus, the

most used modules are more conspicuous. In terms of node color, we distinguish node types with different colors. Database-related nodes, placed in the inner part of a cluster have dark tones. Specifically relation nodes are grey and views are dark green. Query nodes have different colors, depending on the folder their embedding script in the applications belongs. Thus, the difference in color provides another way of grouping queries. This way we can draw several conclusions like how likely is it for queries that belong to the same folder, to belong to the same cluster as well.

Algorithm 4 View stratification

Input: A summary of an architecture graph *G (Vs ; Es)* of a cluster with *Vs* the views of the cluster and *Es* the edges between nodes of *Vs*

Variables: *cnt*

Output: *S*: a topologically sorted set of strata $S = \{S_1,\ldots,S_n\}$ each stratum S_i includes a set of views in *Vs*

```
Begin
1.  Copy Vs to V
2.  Stratum S = null
3.  cnt = 0
4.  while V not empty do{
5.      find view with 0 incoming edges from V
6.      remove view from V
7.      remove edges starting from view
8.      Stratum Scnt = new stratum with all nodes of V with 0 incoming edges
9.      cnt = cnt+1
10.     S = S ∪{Scnt}
11. }end while
12. Return S
End
```

Fig. 22 Decide circles for the view nodes.

A matter of particular importance that is not obvious in the first place is the amount of visual clutter introduced by edges. Edges are the main source of visual clutter and to reduce it, we reduced the intensity of the edges' presence of the visual map. To do that, we picked a light gray color for the edges and we made them very thin, in terms of weight, almost invisible. However to

keep the information the edges provide us, every time a particular node is selected by the user its neighboring nodes are highlighted with a blue transparent color so, instead of emphasizing edges, we emphasize neighbors.

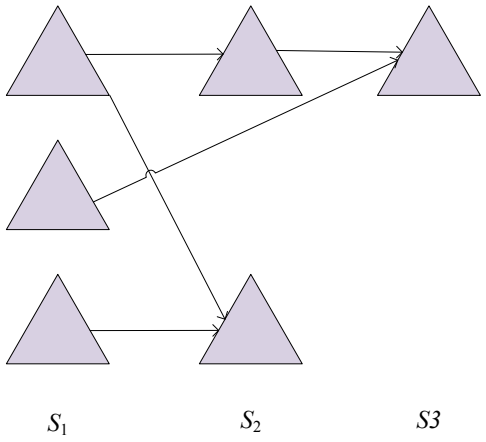


Fig. 23 Strata S1, S2, S3 as a result of view stratification produced from Algorithm 4.

3.6 Refactoring Hecataeus: The Case of Zoom-In for Modules

In the content of this Thesis we had to refactor the visual part of *Hecataeus*. Databases are continuously evolving environments, where design constructs are added, removed or updated rather often. Small changes in the database configurations might impact a large number of applications and data stores around the system: queries and data entry forms can be invalidated, application programs might crash. A data-centric ecosystem comprises a central database and all the software modules and applications that base their operations on it: forms, reports, stored procedures, workflows, etc. *Hecataeus* is a tool that represents the database schema along with its dependent views and queries (acting as an abstraction for all the above software modules) as a uniform directed graph. Hecataeus enables the user to create hypothetical evolution events and examine their impact over the overall graph as well as to define rules so that both syntactical and semantic correctness of the affected workload is retained. In order to visualize the database

modules Hecataeus uses the Jung Visualization viewer as its canvas. A Visualization viewer in Jung is in fact a JPanel on which we draw the graph.

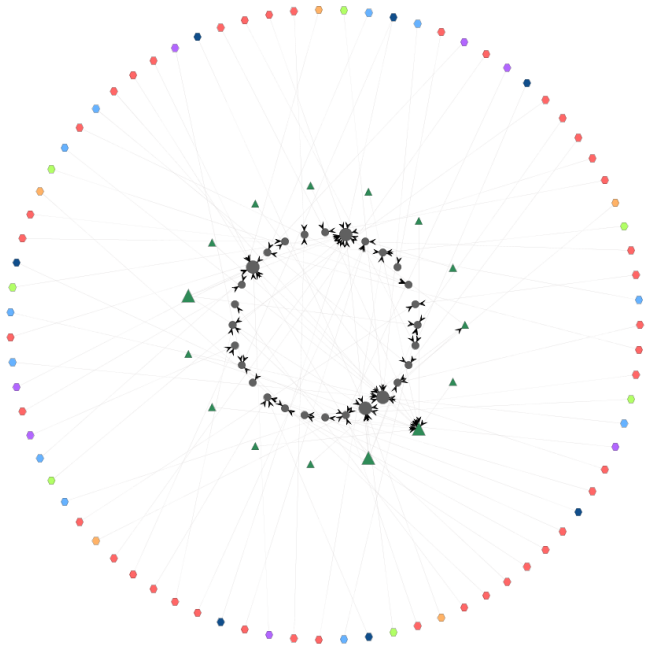


Fig. 24 Initial non clustered layout of Hecataeus

Hecataeus has a graphical user interface environment making its features easier to access. The Hecataeus window consists of two parts. The left part is the visualization viewer, where the graph is displayed and the right part consists of a tabbed pane that enables the user with various features (See Fig. 25).

To use Hecateus there is a simple process to follow.

- The user can create a new project via the menu bar, select the workspace and copy to this location the files that describe the database (i.e. relations, queries, views). Alternatively, if the database schema the user wants to examine had previously been opened with Hecataeus, the user can select to open the already existing project file that was generated from Hecataeus.
- When the user opens a project the graph is displayed on the left part of the window (Fig. 25). By default the initial layout of the graph is a non-clustered concentric layout (See Fig. 24). It consists of at most 3 concentric circles, each one for the three types of nodes

relations, views and queries. In order to decide which circle to assign on each node type we check their number. The type of nodes with the fewer nodes gets in the innermost circle and then for the middle circle we get the node type with the directly less nodes and at the outer circle we place the nodes of the node type with the most nodes. Again the radii of these circles are calculated with the formula (2).

- Next, the user can apply a new layout. All the available layouts can be found by selecting the Algorithms submenu in the Visualize menu. If a clustering layout is selected, before displaying the graph Hecataeus asks the user to provide a value for the clustering parameter *T*. *T* defines how separate the clusters will be. *T* can get values from zero to one. Zero value for *T* means than every node will be considered a cluster and this will result in a simple circular layout. Selecting one for *T* means that the clustering algorithm will not stop until there are no cross edges between clusters (i.e. all clusters are independent). When the user selects a value for *T* the clustering algorithm will start and when it finishes it's time for the layout algorithm to visualize the graph. When the visualization part is over, the user is presented with the result depicted in Fig. 25. On the left part of Hecataeus the user can view the clustered graph and select nodes to open in a different tab for further inspection. In Fig. 25 we see that the nodes with the orange color are selected and the other nodes that use them are highlighted with a blue transparent color. We can also observe that in Fig. 25the user decided to zoom in one node by its self and then the user selected two nodes and zoomed into both of them, thus there are three tabs open on the left part of Hecateus. The last two tabs that where opened by the user to view additional details of the selected nodes can be closed by the user when they are not needed while the first tab which is created by the system and gives a full view of the graph cannot be closed. On the right part of Hecataeus, the user can get a summary of the clusters by checking the overview map in the "Map" tab (Fig. 25). We provide as well the user with a reference for the colors used for the nodes in the "Colors" tab.
- Finally, the user can apply functions to the nodes by clicking the tabs "Event" or "Policy" and filing in the required details.

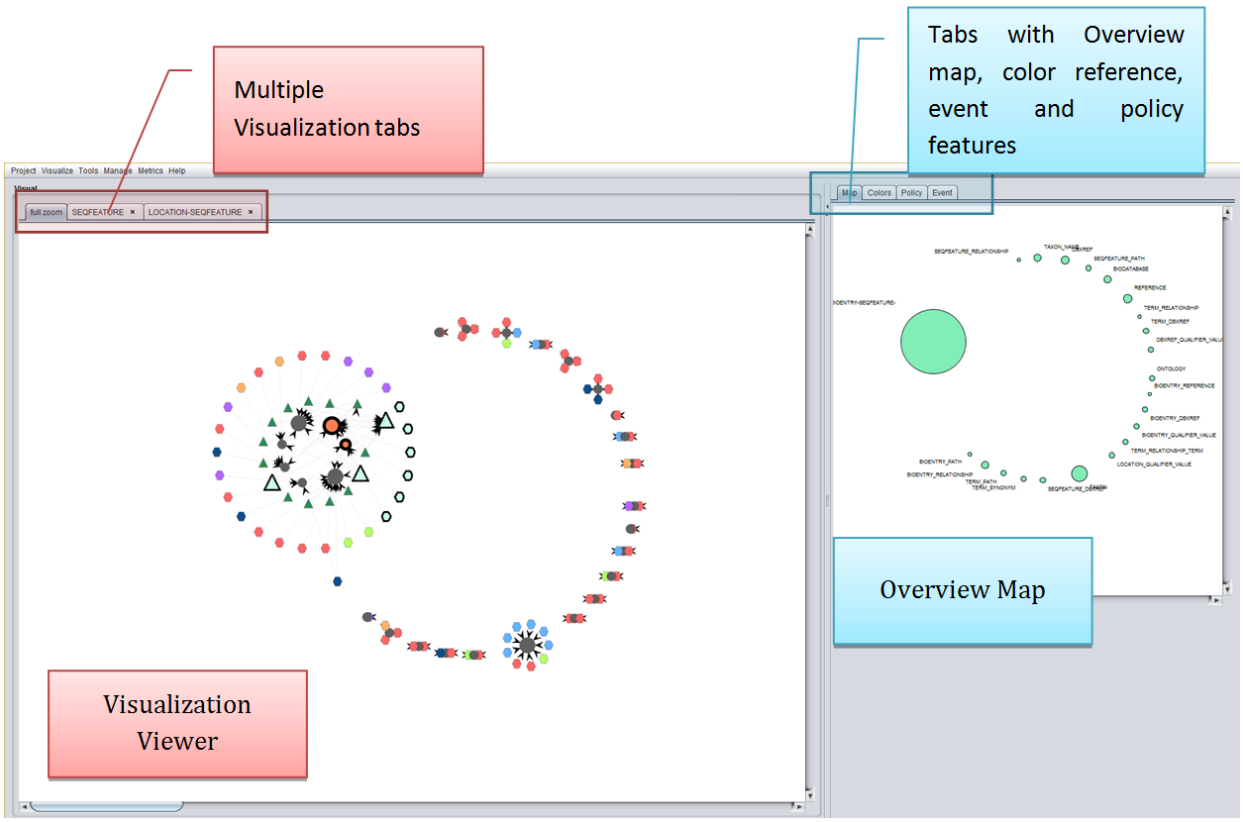


Fig. 25 Hecataeus window.

3.6.1 Tabbed Pane

The old version of Hecataeus featured only one visualization viewer active at a time, so our goal was to refactor Hecataeus to support a new function providing the user with many visualization viewers on demand, with the intent to be used as "zoomed-in" viewers for further detailed information. With multiple viewers the user can perform detailed work simultaneously in different levels of detail without the need to zoom repeatedly in and out [DeRo10]. These viewers can also be used to provide multiple visualizations of the same graph at the same time in a different tab. This feature we added can be used in two ways:

- Zoom-in functionality for modules. This new feature is in fact an application of the "*Visual Information Seeking Mantra*" which suggests the basic guidelines for visual representations: *Overview first, zoom and filter, then details on demand* [Shne96].

In our case, the *overview* of the graph is provided by default on the main tab. This new functionality enables *zooming* in to lower module layers of the user selected node or nodes of interest, which is a way of *filtering* and also providing *details on demand*. With this new feature the user can select a node form the graph and view in another tab its components. This way we reduce the amount of information visualized and as a result visual clutter. In this new "zoomed-in" tab the user can view and apply actions

- View the same graph in a different layout in another tab. Before the addition of this functionality if the user needed to compare two or more layout algorithms for the same graph, the new layout had to be applied and then again the old one. Now the user can apply many layouts, visualized in different tabs, to the graph simultaneously compare them and finally choose one.

To achieve this multiple viewer visualization every time we open a new tab we create a different graph, as there is no way to make visual changes to a graph without making them appear on all viewers that visualize this graph i.e., if we want to see the lower level constructs of a higher level node or apply a different layout on a graph, the result would be applied on all open viewers which visualize this graph.

In our implementation we consider the first tab to be the "main" tab which provides the overview of the graph (main tab cannot be closed) and if any changes are applied to nodes on other open tabs, we copy this changes to the main graph.

3.6.2 Overview Map

To provide a zoomed out and less detailed view of the clusters we visualized with the above methods we implemented an overview map of the clusters. Overviews in general include zoomed out views of certain data types for viewing a summarized view of the graph plus an adjoining detailed view [Shne96]. We placed this map on a tabbed pane on the right side of the application window providing the ability to the user to view both viewers at the same time. This map was created mainly to help the user identify the clusters. To help this cluster identification problem

we place every node in the cluster map in the same position as the cluster it represents so by looking at both visualizations it is straightforward to relate a node on the right (the overview map) with the cluster it represents on the left.

Each cluster is represented by a different node on the overview map. To determine the size of the overview nodes we used formula (2) which in this case means than the size of every cluster overview node is proportionate to the size of the cluster it represents. We do not apply different colors for the map nodes since they are quite distinct by themselves thanks to their positioning.

Another issue that we had to address was the labeling of the map nodes (i.e. how to name them). The main idea was to name each map node with the same name as the relation in the cluster. The problem was that some clusters consist of more than one relation nodes and naming a map node by all the relations it features is not a good idea (huge names).

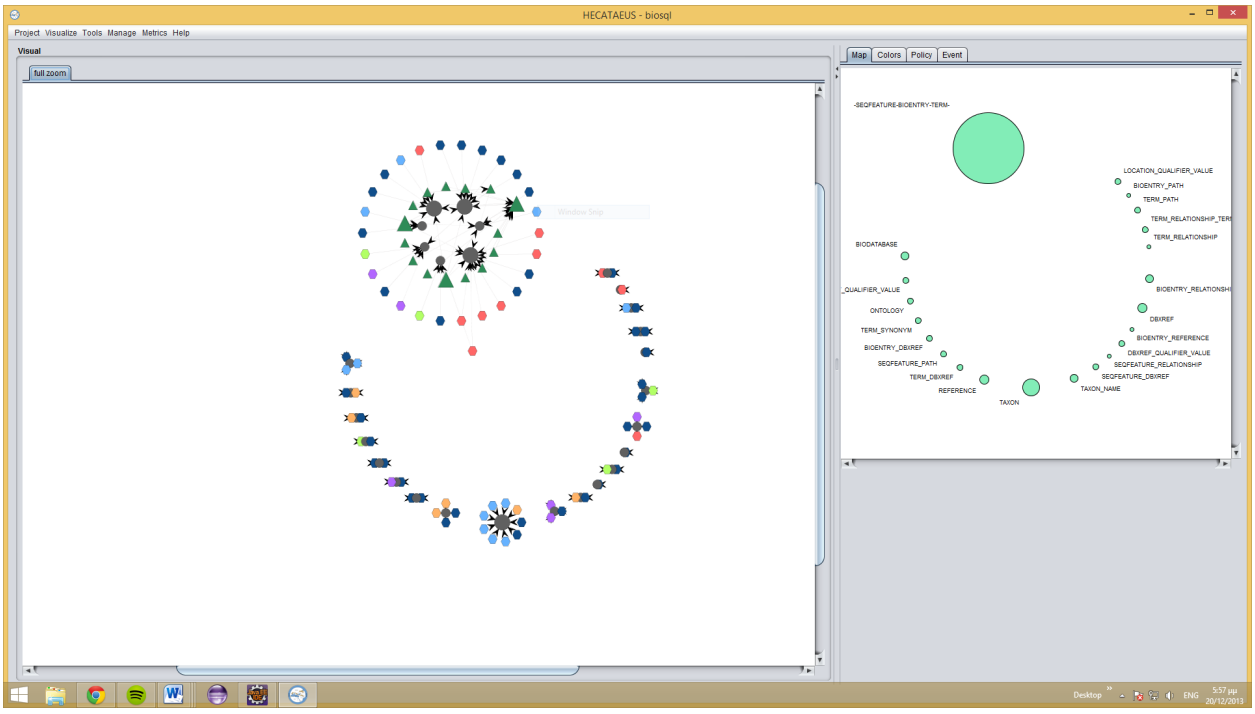


Fig. 26 Application window with two viewers: left the detailed clustered view and right the overview map.

To find a way around this problem, we define the notion of a dominating relation. A relation is dominating if the number of its incoming edges is higher or equal than the average number of incoming edges per relation in the cluster. So, we compute the average number per relation for each cluster by summing the incoming edges for every relation in the cluster and dividing this sum by the total number of relations in the cluster. Then we identify the dominating relations of the cluster on the basis of the above definition (observe that for the majority of clusters, which include a single relation, the definition works fine). Finally the name of the cluster is computed by concatenating the names of the dominating relations, thus the name of the corresponding map node. The overview map is a new visualization viewer just like the main viewer but smaller.

CHAPTER 4. Experiments

4.1 Experimental Method
4.2 Aesthetic Criteria
4.3 Assessment of Objective Criteria
4.4 Comparison to Alternative Methods

In this chapter, we present our experimental assessment of the proposed visualization methods. We start with the discussion of the experimental method and then we assess our method against aesthetic criteria and objective measurements and advantages over opponents.

4.1 Experimental Method

In order to evaluate our work, we have used some well-known open source projects that contain database queries. Table 1 provides a list of the analyzed software projects and gives additional information in regard to the dataset (number of relation nodes, view nodes, query nodes and the total number of edges in each graph). In order to convert the software of the analyzed tools to the graph representation that we use in this Thesis, we performed a sequence of steps. The source code of the last version of each tool was downloaded. We retrieved the database definition from the source code. Also, we grepped the source code for the occurrences of SELECT and FROM terms in it, and out of the resulting text, we isolated lines actually encompassing queries. The actual queries were automatically isolated via a dedicated java application we constructed for this purpose and post-processed in order to be parsable by our tool, Hecataeus that ultimately converts the ecosystem to an architecture graph and visualizes it for the user. For more information about the data preprocessing, please refer to the Appendix.

Table 1. Datasets Used (R: Relations, V: Views, Q: Queries, E: Edges)

Dataset	Description	R	V	Q	E
Biosql	A generic relational schema covering sequences, features, sequence and feature annotation, a reference taxonomy, and ontologies	28	15	79	104
ZenCart	An open source shopping cart software	106	0	149	158
Drupal	An open source content management platform	75	0	355	379
OpenCart	An open source shopping cart software	115	0	650	824

4.2 Aesthetic Criteria

In this subsection, we assess whether our major design choice of using circular layouts as well as the detailed technical choices for each of the proposed methods were appropriate according to the fundamental principles for visual object representations. In our visualizations we employ several of the Gestalt means to provide unity, flow and order.

Good Gestalt – Pragnaz: The individuals tend to simplify the world by removing complexity and focusing on simplicity, orderliness and familiarity. Under this context, objects that form a pattern that is regular, simple and orderly tend to be perceptually grouped together. In accordance with this rule, we place clusters of objects on the periphery of a circle or on concentric circles. Each cluster again is displayed in terms of a set of concentric circles, thus producing a simple, familiar and repetitive pattern. We avoid unnecessary visual elements as much as the requirements of the problem allow us to, in order to retain simplicity and conciseness of the visual representation. How we address the aesthetic criteria for each one of the algorithms is explained in Table 2.
A list of the aesthetic criteria [BRSG07] we use in our visualizations are the following:

- *Proximity* is a fundamental Gestalt principle which states that objects close to each other tend to be perceived as similar. In our implementation, *proximity* is achieved via

clustering: we place nodes with similar semantics closely to each other (in our case, nodes belonging to the same clusters of relations, views and queries).

- *Connectedness* suggests that physically connected elements are usually grouped. In our case, this principle is inherently achieved via the choice of graph representation and clustering.
- *Similarity* and *proportion* state that objects of the same color shape and size are perceived as similar by individuals and they should be placed closely. These criteria are encompassed in our methods via several decisions. We use the same shape for nodes of same type (relations, queries, views), the same color for nodes that belong in similar physical structures (queries that belong to the same files at the source code are colored with the same color; relations and views are always gray and dark green, respectively), and, we scale the size of each node according to its graph degree, so that larger size denotes larger degree of interrelationship.
- *Closure* and *isolation* suggest that the eye tends to create perceptions of closed space, even if they do not exist. Closure is best served when the depicted objects tend to create a "border" around similar objects along with blobs of whitespace for isolation. In our visualizations, closure and isolation are also inherently supported via the idea of circular visualization of each cluster: the outermost circles of each cluster provide a visual border that separates it from other clusters. We take special care for clusters not to intersect and we enhance their surrounding space with intentionally injected whitespace. The same care is paid for individual nodes too.
- *Clutter avoidance* is one of the fundamental problems our methods try to battle. As the main source of visual clutter is the overwhelming presence of edges --especially, when they cross-- we take every possible means to minimize the number of pixels-per-edge, without taking them out of the diagram, at the same time. To this end, we decrease the strength of edge coloring to light grey and highlighting neighbors only when a user interactively picks a node to inspect. The idea of clustering nodes and putting similar nodes closely tries to minimize the span of edges throughout the entire canvas. Within each cluster, we implement an adjustment of the barycenter method to radial layouts to minimize edge crossings. In all our methods, we use straight lines for the edges. Edge

bends make edges more difficult to follow because an edge with a bend is more likely to be perceived as two separate objects. When considering the length of edges, edge length is minimized through clustering. A second source of clutter is the overlapping of clusters or nodes. To avoid such situations, we intentionally tune the methods to avoid such phenomena.

Finally, we avoid any other *emphasis* of individual clusters or nodes and leave this aesthetic tool available for arbitrary usages where there will be a need for emphasizing some parts of the graph. We do not inject visual hierarchies, *contrast*, *asymmetry* or *discontinuity* to emphasize any part of the graph. We also avoid any special purpose focal points to guide the visual flow of the user's optical navigation over the screen. Flow occurs, however, in the concentric methods, as there is a flow from smaller towards larger clusters. This makes these methods more suitable for arrangements where cluster size is also important for the users' work.

Table 2 Aesthetic criteria and how we address them.

Aesthetic Criteria	**Circular Layout**	**Concentric Circle Layout**	**Concentric Arc Layout**
Proximity	Clustering		
Connectedness	graph representation and clustering		
Similarity and Proportion	Same shape and color for same type of nodes, size proportionate to connections		
Closure and Isolation	Circular visualizations, white border between clusters and nodes		
Clutter Avoidance	**In clusters**: different angle for each cluster depending on its size (section 3.4.1)	**In clusters**: concentric circle radius optimization (section 3.4.2)	**In clusters**: different angle for each cluster, radius optimization section 3.4.3)
	In graph: clustering algorithm produces "perfect" clusters – no intra cluster edges **In edges**: barycentric method for node placement, light edge coloring		

Table 3 How the "Visual Information Seeking Mantra" is applied in our implementation.

Visual Information Seeking Mantra principles	**How we address them**
Overview	We give an overview of the graph by visualizing only the higher level nodes. For an overview of the clusters, we created an overview map that represents each cluster as a single node
Zoom	We implemented the zooming feature by providing a more detailed view of the selected parts of the graph in a different tab
Filter	The fact that we support zooming in user specified areas of interest and provide a new tab with only the user selected modules in higher detail is a way of filtering information.
Details-on-Demand	This principle is also available via the zoom in feature we implemented

4.3 Assessment of Objective Criteria

Apart from the aesthetic part, we can also assess the behavior of each of the proposed parts of our approach with specific objective criteria. First, we will discuss what the outcome is and benefit from our clustering and preprocessing steps. Then, we assess the different methods in terms of how efficiently they exploit space and we will also present the time needed for our algorithms for both clustering and visualization parts.

All experiments were made on a workstation with cpu Intel core 2 E7200 @2.53 GHz x 2 and ram 3.9 GiB.

Table 4 Objective measures for all four data sets

Data set	# clusters	# nodes	Avg nodes/ cluster	Avg cluster radius	Avg edge crossings/ cluster	Avg edge length
BioSql	22	112	5,55	12.00	3.50	92.11
ZenCart	41	255	4,8	11.85	0.46	69.21
Drupal	37	429	11	25.01	15.62	497.71
OpenCart	59	765	12,9	28.32	84.08	751.32

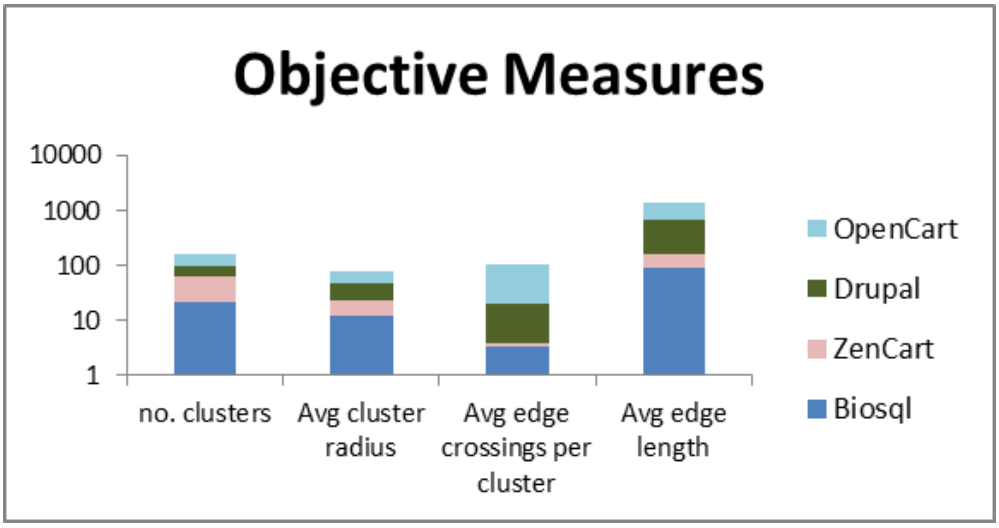


Fig. 27 Objective measures for all datasets.

Clustering Effectiveness: The first important result concerns the effectiveness of the clustering of the graph nodes. In all the studied datasets, our clustering produced a clean separation of the graph in connected components that are completely disjoint and isolated! In other words, clustering produced clusters that have no edges between other clusters (inter-cluster edges). This resulted in the elimination of all the visual clutter that these edges would incur. The second piece of good news is that the number of clusters ranges within 20 – 60 clusters in all four cases, thus it is presentable in a 2D screen canvas (see Table 4 for the exact numbers). We should, however, emphasize that due to the specific nature of the data sets (all are CMS's) the results should by no means be generalized to other types of ecosystems.

In Table 4, we give two objective criteria that are typical metrics used in all the related literature, specifically number of edge crossings and average edge length, as well as a metric that is specific to our method, and concerns the area covered by the clusters, expressed as average cluster radius. It is important to note that, in the absence of inter-cluster edges due to our clustering, all these objective measures are independent from the visualization method that follows the original clustering. To measure the effectiveness of our clustering algorithm, we have measured the time our algorithm needs to cluster the data (See Table 7 for results). As expected, the time needed for the clustering possess depends on the size of the dataset and varies between 33.7 milliseconds to 5785.7 milliseconds.

Method Effectiveness: A second important piece of evidence has to do with the effectiveness of the employed methods in terms of space utilization. For each method, we measure

a. the rectangle produced by the farthest pairs of coordinates,
b. the sum of the areas covered from all the clusters (on the basis of their outermost circle),
c. the resulting percentage of area covered by the visualized clusters.

Remember that the area covered depends on the particularities of the method; then, our system autoscales the result to fit in the screen. Thus, the area of the bounding rectangle is a clear indicator independent of the scale factor of the drawing: between two drawings that will ultimately have to fit in the same screen, the one with the larger area needed, will require more zoom-out but the area it occupies does not depend on the zoom factor. The units we measure the

graph area are pixels. To explain that, we start by defining the difference between user space and device space. In most computer graphics environments, each pixel is numbered. If you draw a square at, say (20, 20), then the top left corner of the square will begin at approximately the twentieth pixel from the left edge, or *axis*, of the drawing space and at the twentieth pixel from the top axis of the drawing space. Coordinates that are offset from axes are called Cartesian coordinates. The location of Java 2D objects is also specified using Cartesian coordinates. The beginning of the space occurs in the upper left side of a hypothetical rectangle that increases to the right and downward. Java 2D, however, defines coordinates in *units* (72 units to an inch), and rendering occurs in a hypothetical plane called the *user space*. Note that we use the term units and not pixels. The latter term implies that the output device is a computer screen. When an object is drawn to the output device, such as a screen or printer, the results may have to be scaled to ensure that the object is the same size. After all, a shape that appears as four inches wide on the screen should also appear four inches wide when it is printed on a piece of paper. A computer monitor typically uses 72 pixels per inch, so each Java 2D unit is conveniently equivalent to a pixel.

Table 5 reports on the area needed for the visualization of the graph. The last column named *Covered area* represents the sum of the area each cluster occupies on the graph, thus it is the same for every visualization method since the cluster sizes do not depend on the layout. The other three columns named after each one of our visualization methods represent the area of the rectangle which embeds the clusters, as it is expected this area is dependent on the visualization method. Underlined blue shows the winner method that requires the least area and bold red highlights the worst performance. We observe that concentric circles always lose and the winner methods are divided, with concentric arches winning for the three smaller cases and circular layout for the largest one (See Table 5 and Table 6). At the same time, the amount of covered area is fixed for all methods (as cluster sizes are fixed before laying them out in the 2D canvas). The percentage of covered area, in all layouts is very small; it ranges between 2% and 25% present. As we see in Table 6 datasets BioSql and Drupal have better space utilization while ZenCart and OpenCart do not take full advantage of the canvas. OpenCart is a large dataset so it is expected to need more space for its visualization. On the other hand, Zencart is a small dataset and its behavior is due to the fact that our clustering algorithm creates many small clusters

(unlike in BioSql and Drupal), which results in a bigger rectangle produced by the farthest pairs of coordinates and smaller covered area by its clusters (see Fig. 29).

We also measured the time needed to visualize the data sets using our algorithms (see Table 7 for average times in five runs). As one would expect the speed of our algorithms depends on the size of the graph we want to visualize. Specifically the fastest visualization is for the BioSql dataset and it takes 6 milliseconds, while the slowest is for the OpenCart dataset and it takes 41 milliseconds. The clustering time depends also on the size of the graph and it varies between 31 for BioSql dataset and 5929 milliseconds for OpenCart. Expectedly, the average clustering times are similar for all of our three algorithms since the clustering procedure is the same (Fig. 30).

Table 5 Area occupied by graph (pixels^2)

Data set	Circular layout	Concentric circles	Concentric arcs	Covered area
BioSql	196243.49	**275943.12**	193640.75	44549.6
ZenCart	2007635.67	**2162419.52**	1238295.66	50739.45
Drupal	2329122.25	**3232092.83**	1612675.06	253172.6
OpenCart	5775976.36	**18392055.26**	9711796.44	461424.53

Table 6 Percentage of occupied area by graph (pixels^2)

Data set	Circular layout	Concentric circles	Concentric arcs	Covered area
BioSql	22%	16%	23%	44549.6
ZenCart	2%	2%	4%	50739.45
Drupal	10%	7%	15%	253172.6
OpenCart	7%	2%	4%	461424.53

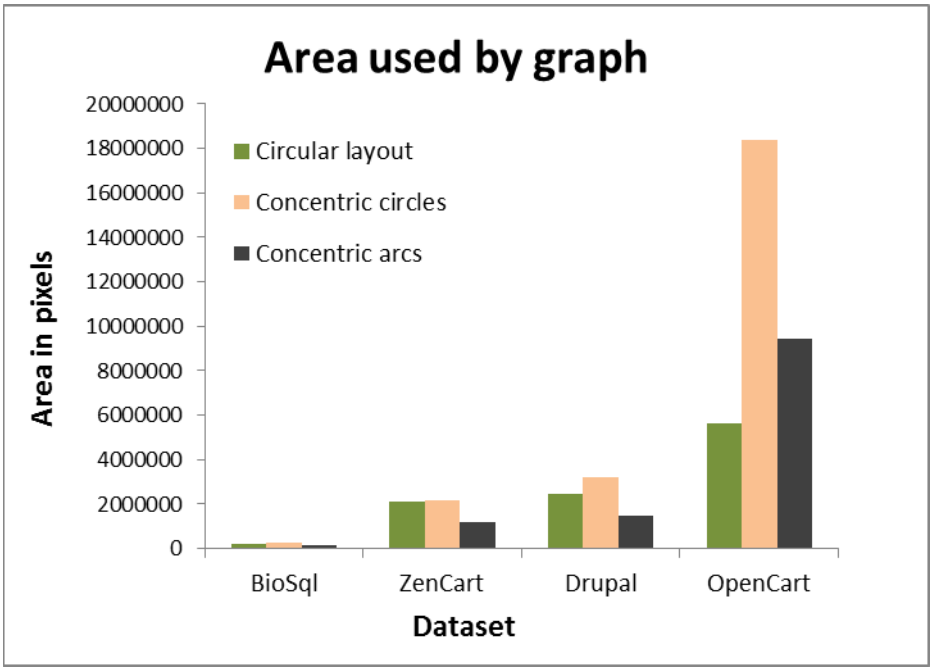


Fig. 28 The area used by the graph for every layout.

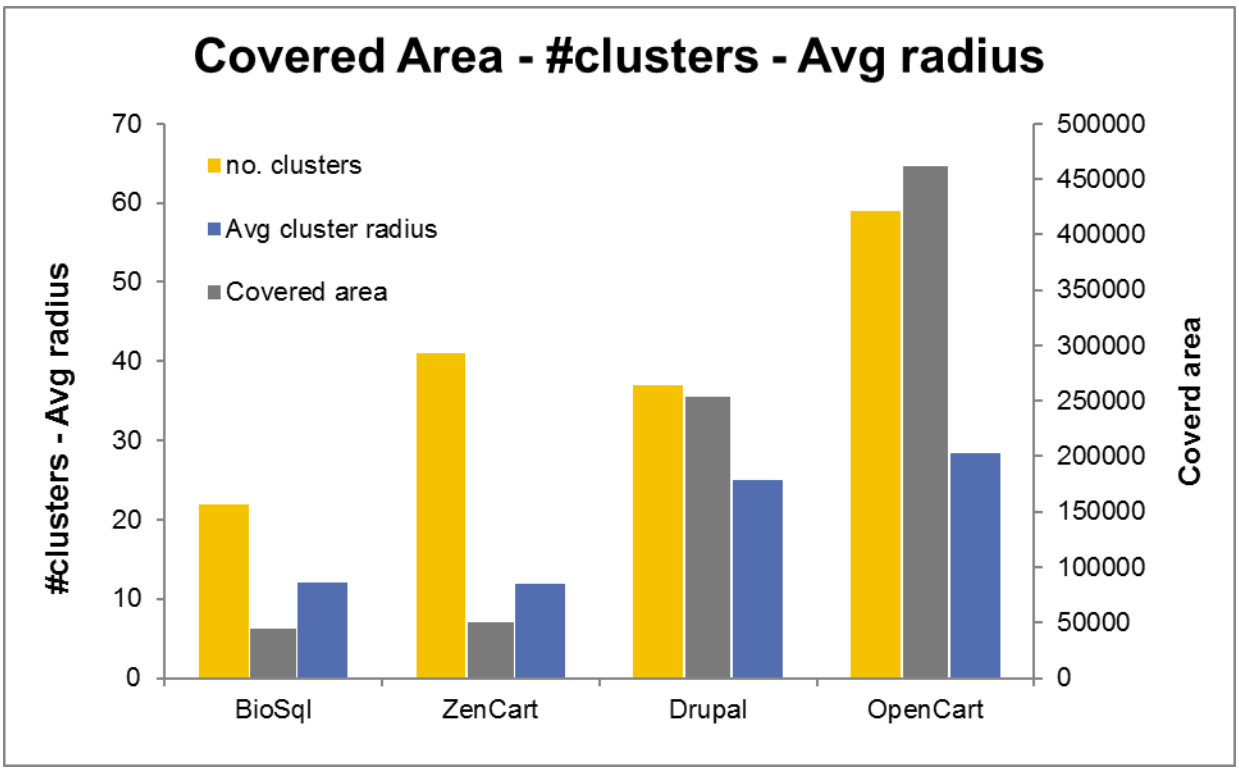


Fig. 29 The connection between number of the clusters and their size with the area they cover.

Table 7 Average time needed for clustering and visualization in milliseconds

Data set	Circular Layout		Concentric Circles		Concentric Arcs	
	Clustering	*Visualization*	*Clustering*	*Visualization*	*Clustering*	*Visualization*
BioSql	36	6	31	6	34	6
ZenCart	118	6	116	12	123	7
Drupal	882	14	758	6	784	5
OpenCart	5.801	41	5.929	13	5.627	8

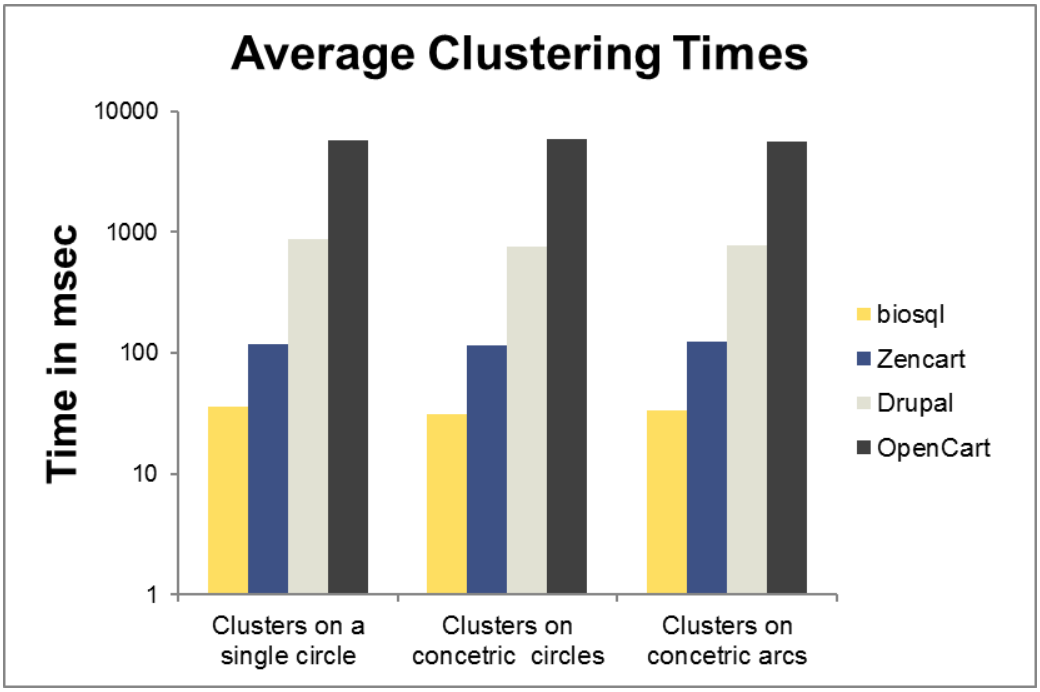


Fig. 30 Time needed for clustering.

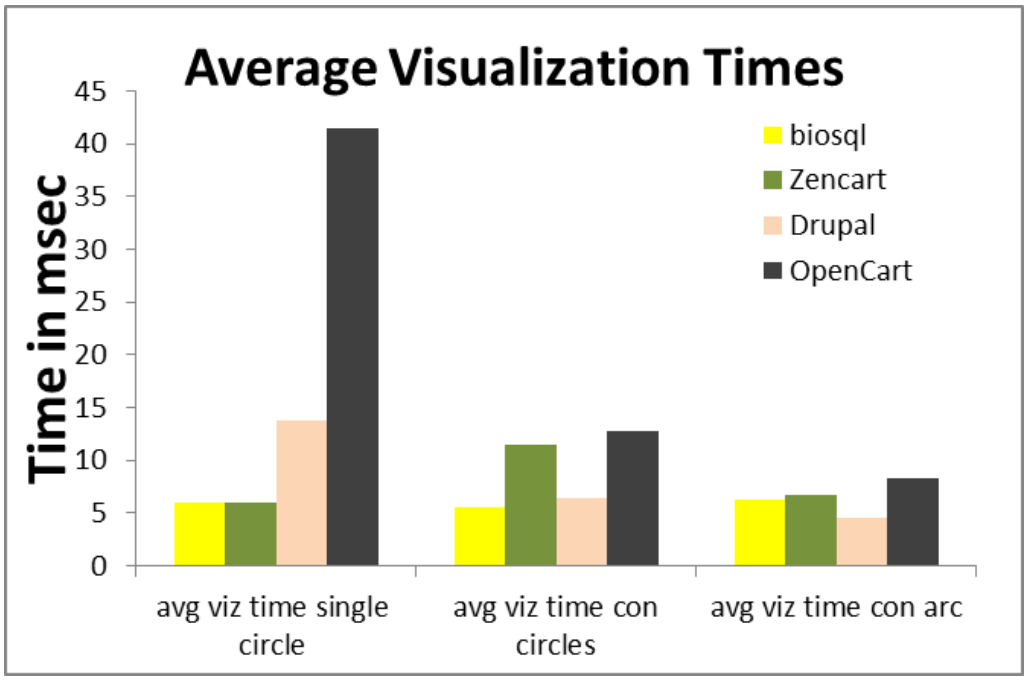


Fig. 31 Average time needed for visualization.

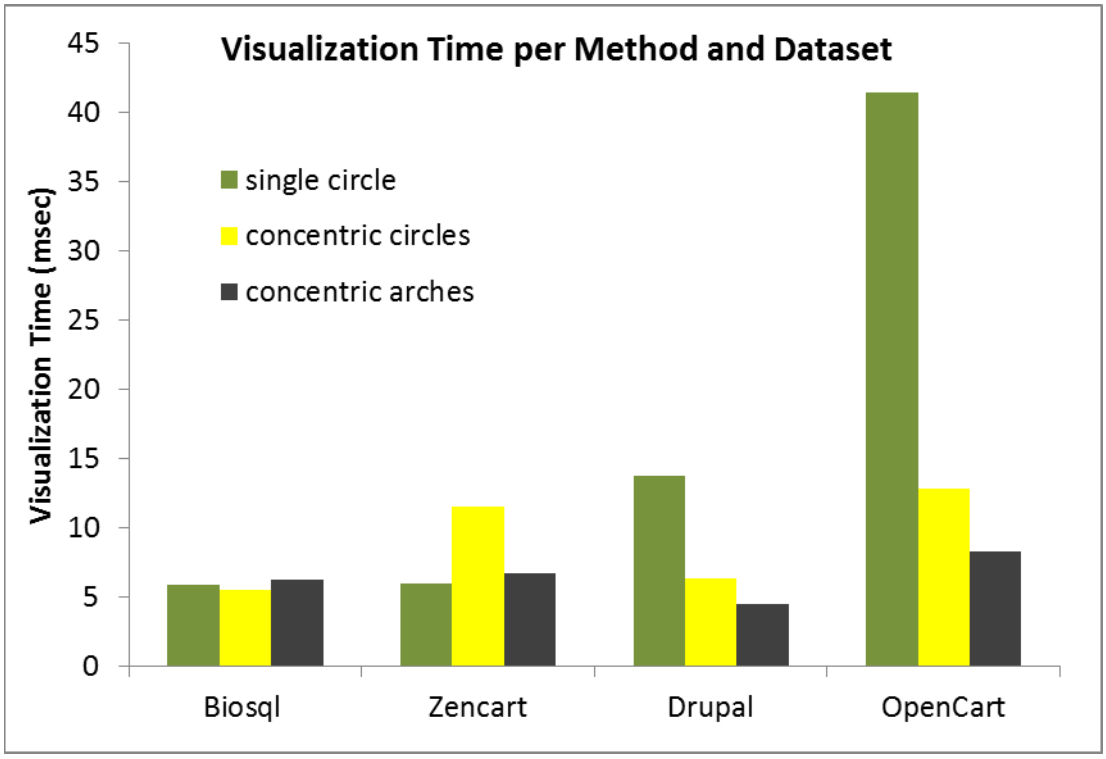


Fig. 32 Visualization time per method and dataset

4.4 Comparison to Alternative Methods

In this section we will discuss alternative visualization methods and we compare them to our approach. Our visualizations were implemented with Jung, a software library that provides a common and extendible language for the modeling, analysis, and visualization of data that can be represented as a graph or network. Jung is written in Java, which allows Jung-based applications to make use of the extensive built-in capabilities of the Java API, as well as those of other existing third-party Java libraries. Jung supports several layouts, out of which we discern the following prominent ones:

- A simple, random Circle Layout (Fig. 33) that places vertices randomly on a circle
- The Fruchterman-Reingold algorithm [FrRe91], denoted as FRLayout (Fig. 34)
- Meyer's "Self-Organizing Map" layout [Meye98] denoted as ISOMLayout (Fig. 35)
- The Kamada-Kawai algorithm [KaKa89], denoted as KKLayout in Jung (Fig. 36),
- The SpringLayout (Fig. 37), which is a simple force-directed spring-embedder[dETT99]

We applied all these layouts on our datasets to compare them with our layouts. The result is shown in the figures bellow. In Jung's circular layout all nodes are randomly placed on a periphery of a circle with radius $R = total\ number\ of\ nodes\ /\ 2\pi$. With this radius there should be no overlaps, however as we see in Fig. 33 nodes do overlap where in some parts of the circumference of the circle there are empty (white) spaces. In the remaining Jung layouts, the nodes of the graph appear to be randomly placed on the canvas.

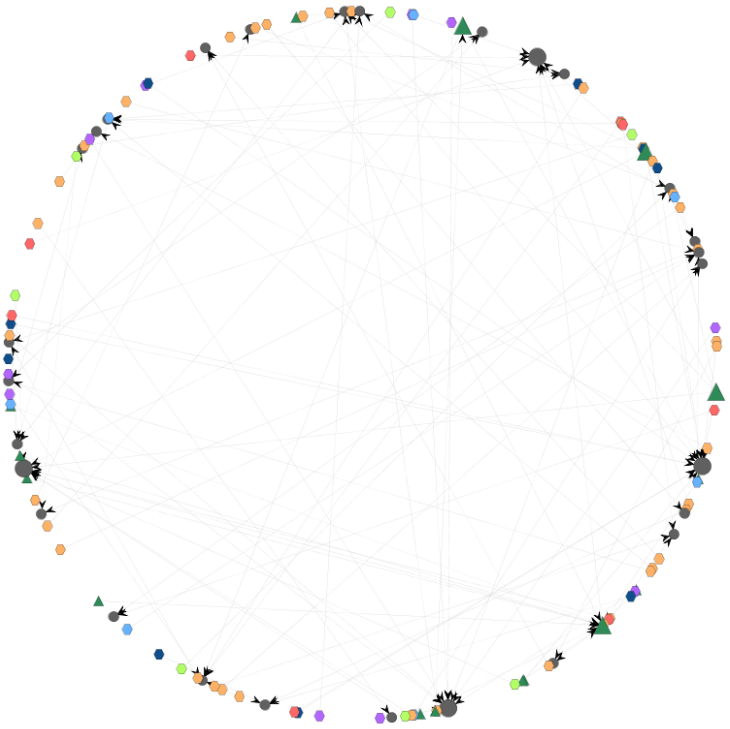


Fig. 33 BioSql visualized via a circular algorithm by Jung.

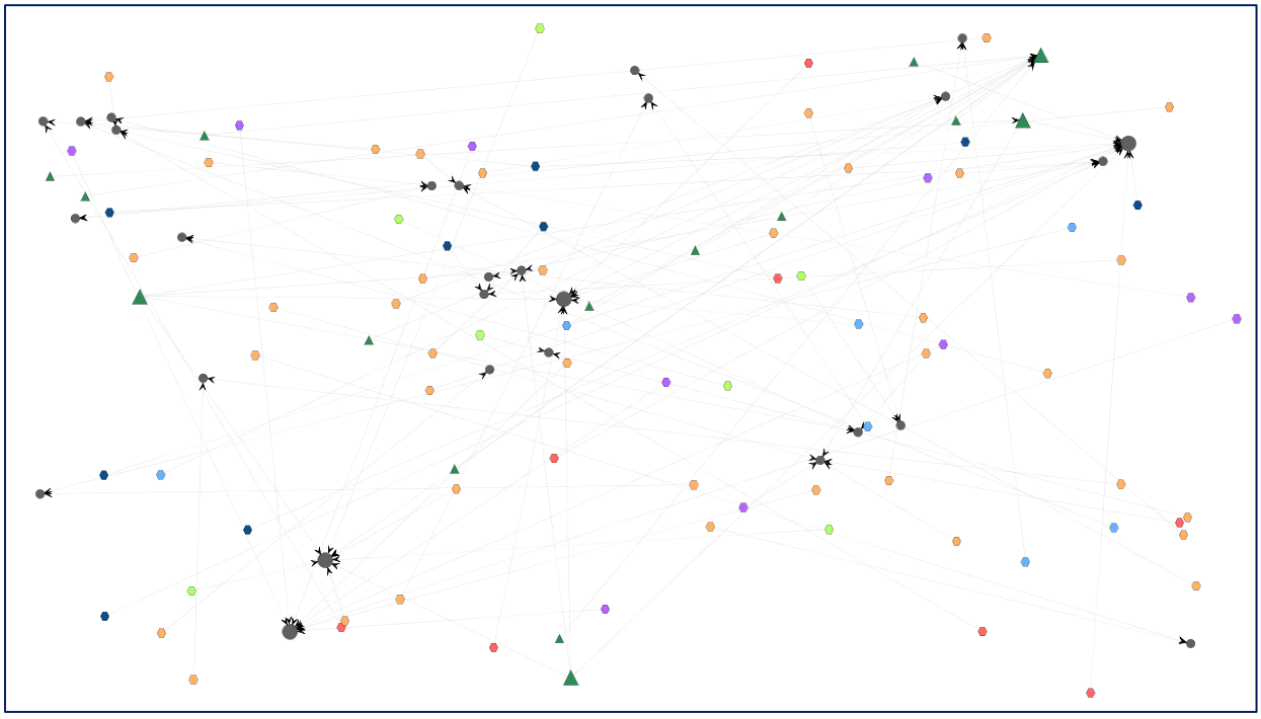


Fig. 34 BioSql visualized with FR algorithm by Jung.

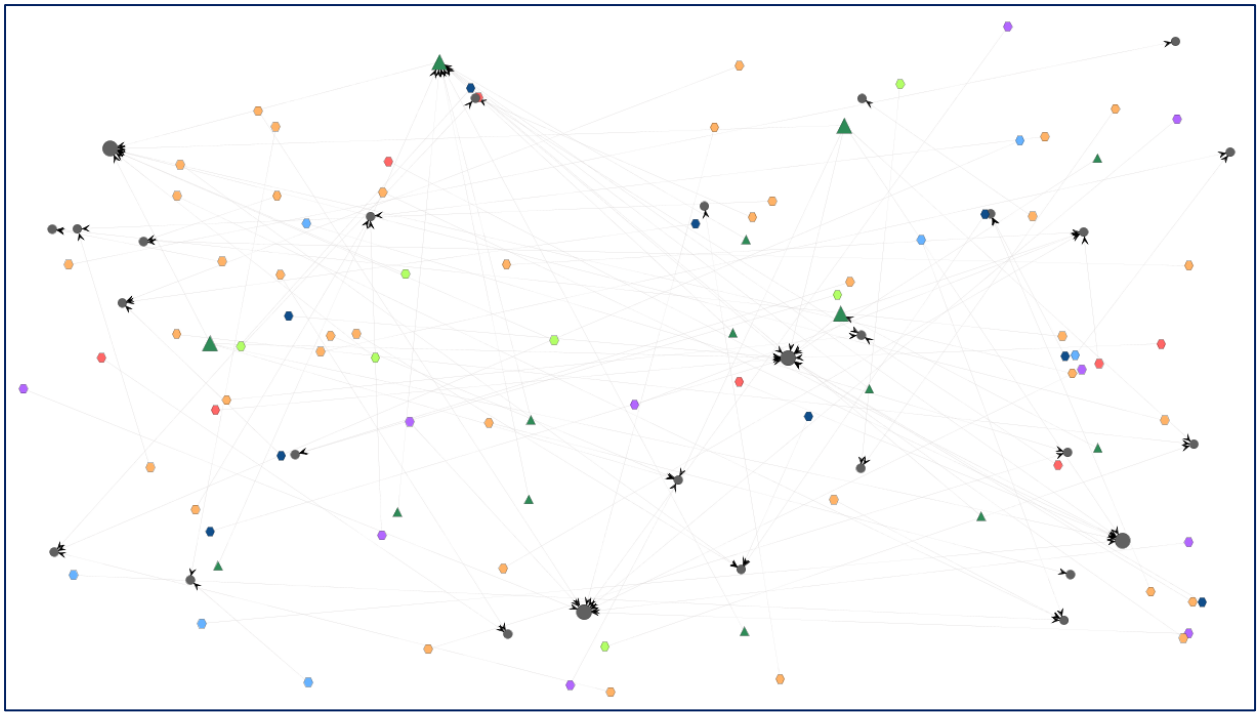


Fig. 35 BioSql visualized with ISOM algorithm by Jung.

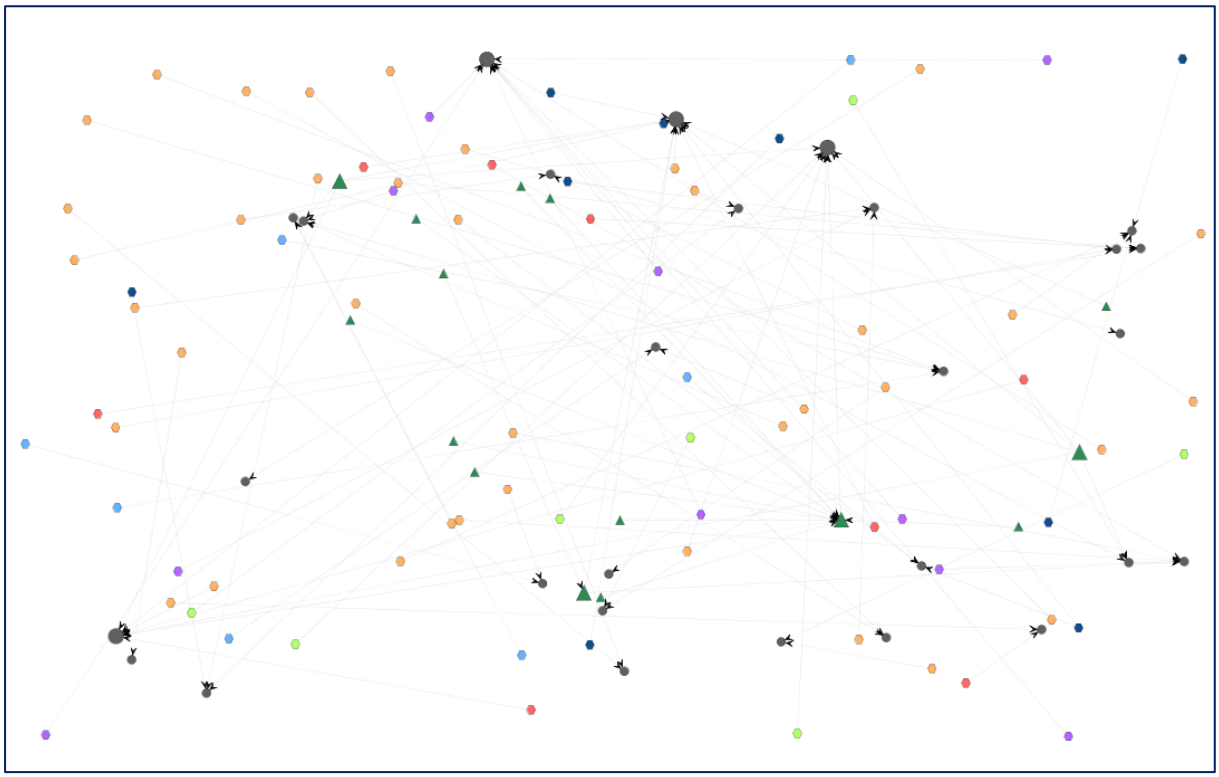


Fig. 36 BioSql visualized with KK algorithm by Jung.

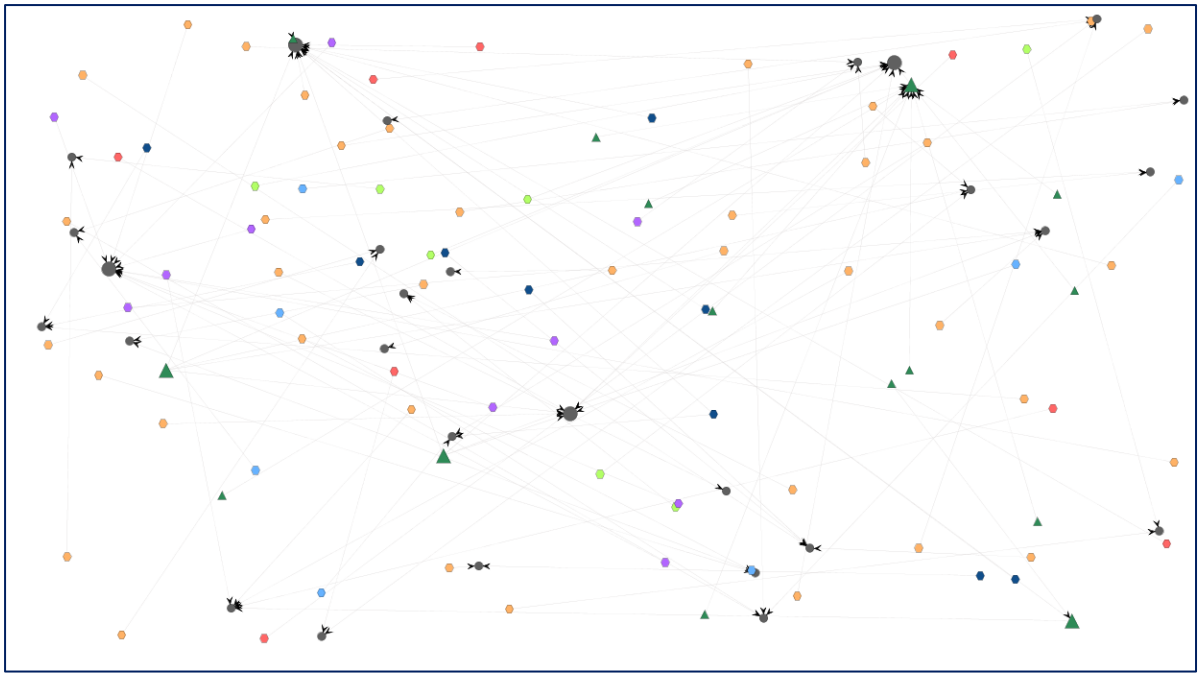


Fig. 37 BioSql visualized with a spring layout algorithm by Jung.

CHAPTER 5. Conclusions and Future Work

5.1 Conclusions
5.2 Future Work

5.1 Conclusions

In this Thesis, we have presented methods for the visualization of data-intensive software ecosystems. We have modeled each such system as a graph. The nodes of the graph represent both database constructs (relations and views) and application-level constructs (queries embedded in the code of the applications) in a uniform manner. Edges denote data provision. To battle visual clutter, we cluster the nodes of the graph in groups; each cluster represents a subset of the entire ecosystem that is related on the grounds of data provision. Given these clusters, we draw a map of the ecosystem in three possible ways, all as variants of circular layouts (to better utilize space) and on the grounds of several aesthetic criteria; at the same time, each cluster is internally visualized as a set of concentric circles, too. Our experimental evidence in all the open-source CMS systems that we have examined, demonstrates that (a) the clusters have produced clean separations of the ecosystem in graph connected components, and, (b) for all the methods, the number of clusters is such that it allows their placement in the screen in a usable way. Concerning space utilization, concentric arches win for small graphs and the circular layout wins for larger graphs.

To the best of our knowledge, this is the first principled step ever in the literature of visualizing data-intensive systems. As such a vast area of research issues remains to be explored, including alternative visualization methods, improved space utilization, and the relationship of graph metrics to source code properties.

5.2 Future Work

Except from the three visualization methods we discussed in previous sections, we have also implemented three other layouts, two concentric and a spiral. We decided however not to present them in the main part of our work because the two concentric were variations of the concentric method we presented in section 3.4.2 and the spiral layout creates visual *flow* something that contradicts the aesthetic criteria we respect.

The first variation of the concentric circles method is depicted in Fig. 38. In this layout we reverse the order we place the clusters. Instead of starting with the smallest clusters and placing them in the innermost concentric circle, in this case we start with the biggest clusters. This method creates a focal point in the center of the concentric circles, thus making the smaller cluster look like outliers.

In the second variation of the concentric circles method we reversed the order we split the circles in 2^k segments. We start with the outermost circle and we divide it in 2 segments and place 2 clusters on the periphery of this circle then we move to the next circle and we divide it to 2^2 and as the algorithm continues, in the end we place the innermost circle the reaming clusters (Fig. 39).

The last layout we implemented is a spiral layout depicted in (Fig. 40). In this method we sort the clusters in ascending order and then we place them on a spiral. To create this spiral, we use a circular layout with the difference that every time we place a cluster we increase the radius so the next one will be placed further form the center of the circle. This method was abandoned because it generates a visual *flow* (i.e. makes the viewer follow the spiral).

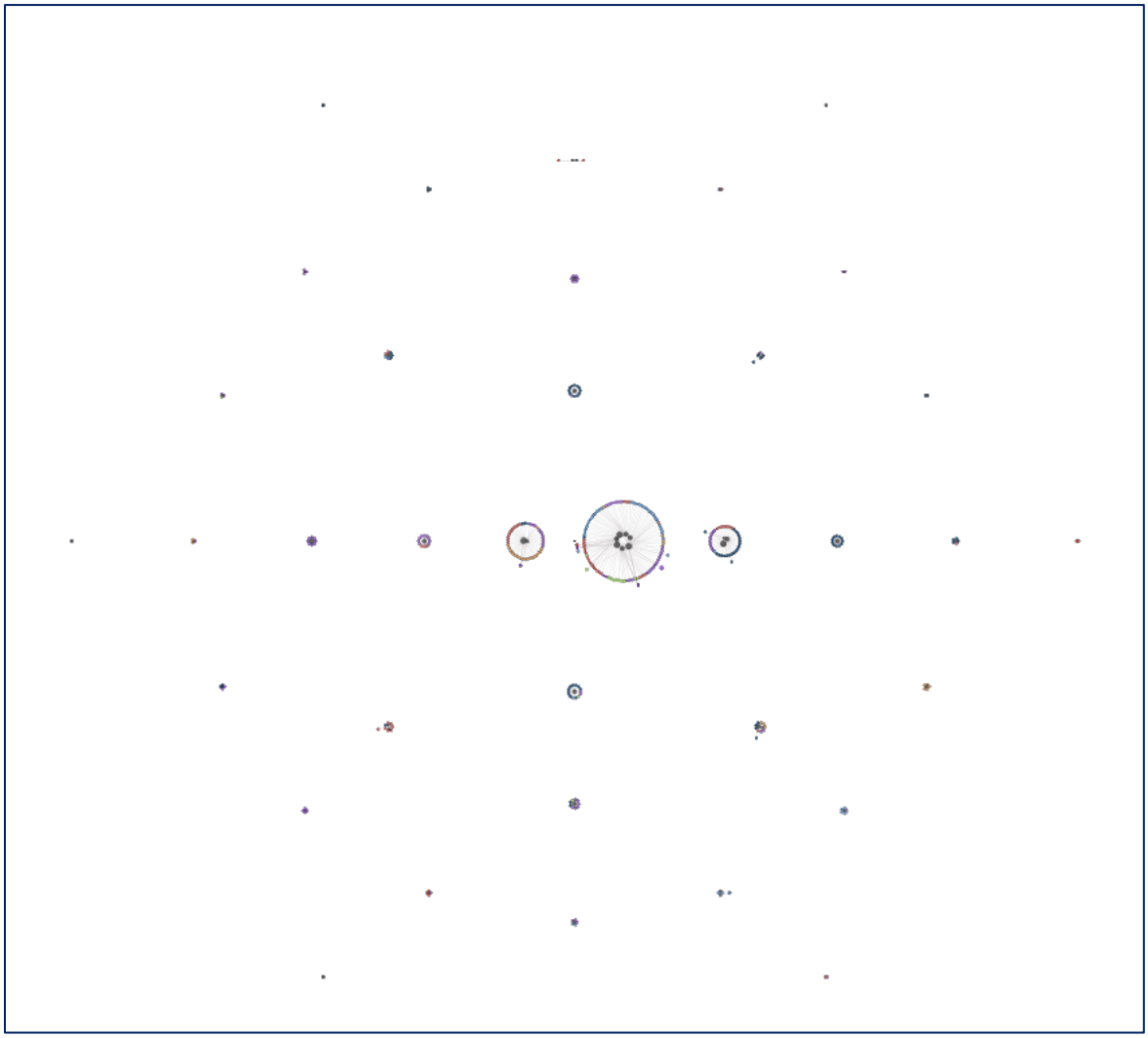


Fig. 38 First variation of concentric circles method for Drupal data set.

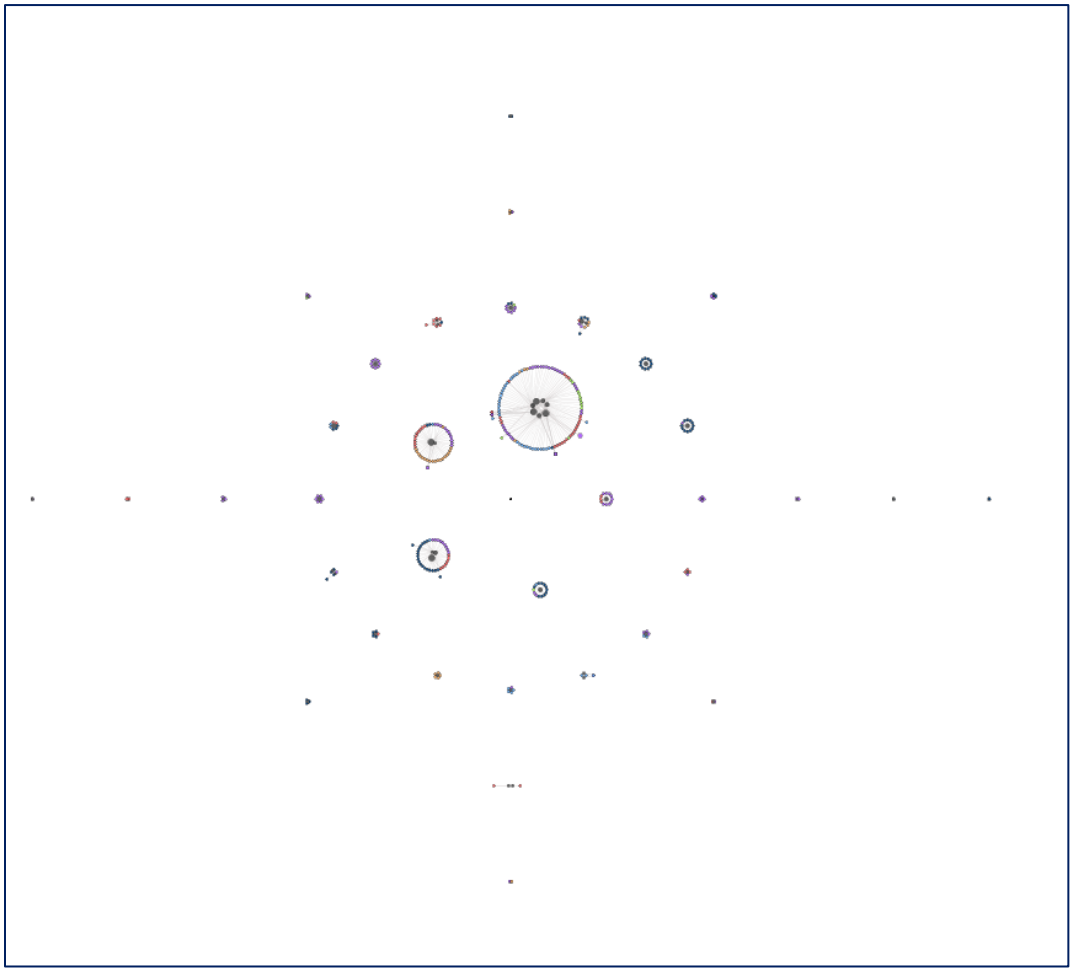


Fig. 39 Second variation of concentric circles method for Drupal dataset

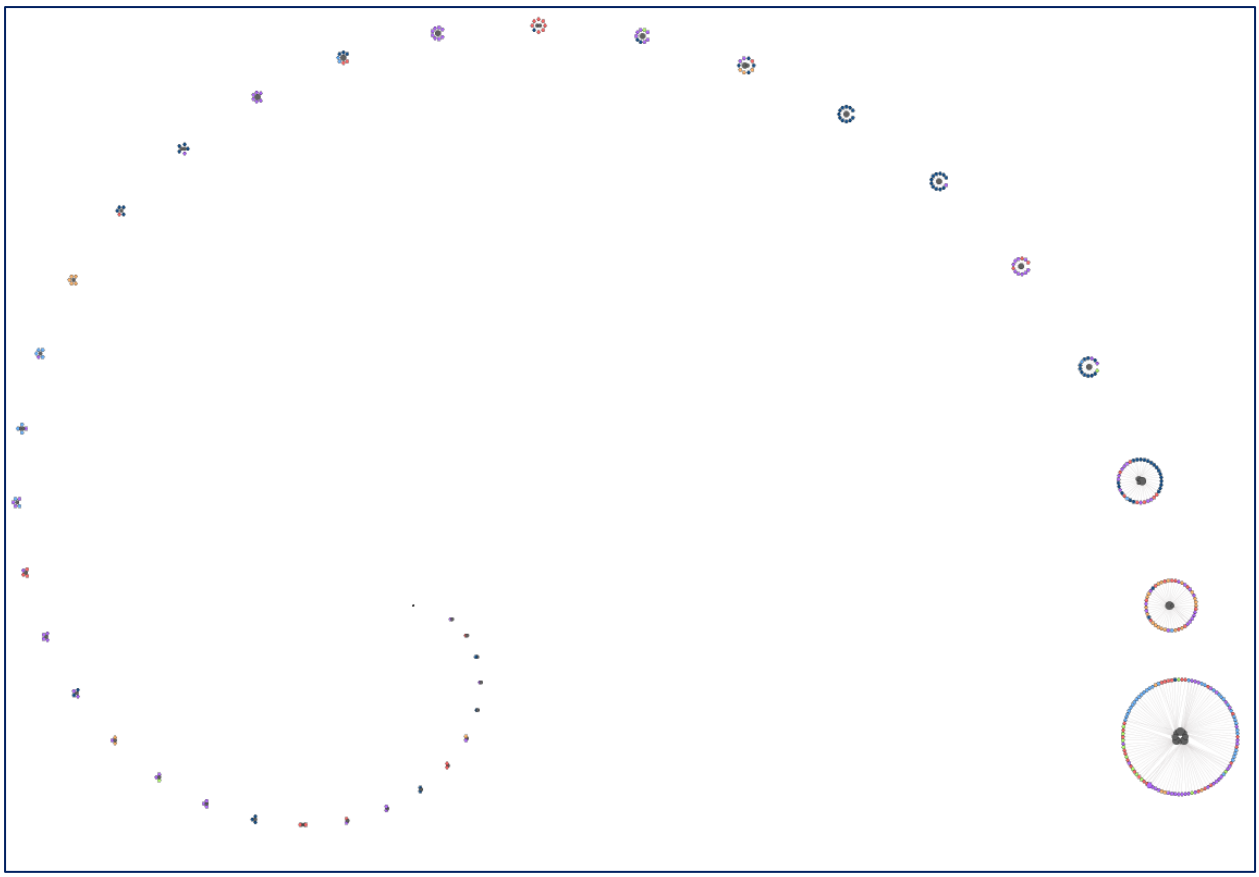


Fig. 40 Spiral layout for Drupal data set.

Naming a few of the things that could get implemented in the near future based on the work we have done are:

- Separate the Hecataeus environment from the Jung visualization part
- Implement other non-circular visualization layouts
- Use another visualization tool besides Jung like D3 or processing
- Find a way to order clusters
- When selecting a file from the color index, highlight the nodes that belong to this file

References

[BeBD09] Fabian Beck, Michael Burch, Stephan Diehl. “Towards an Aesthetic Dimensions Framework for Dynamic Graph Visualisations”, In 13th International Conference on Information Visualisation (IV 2009), pp 592-597, 2009

[BRSG07] Chris Bennett, Jody Ryall, Leo Spalteholz, Amy Gooch. “The Aesthetics of Graph Visualization”, In Eurographics Workshop on Computational Aesthetics in Graphics, Visualization and Imaging (Computational Aesthetics 2007), pp 57-64, 2007

[BZRK10] Andrew Bragdon, Robert C. Zeleznik, Steven P. Reiss, Suman Karumuri, William Cheung, Joshua Kaplan, Christopher Coleman, Ferdi Adeputra,Joseph J. LaViola Jr. “Code bubbles: a working set-based interface for code understanding and maintenance”, Proceedings of the 28th International Conference on Human Factors in Computing Systems (CHI 2010), pp 2503-2512, 2010

[CrCa05] Brock Craft, Paul A. Cairns. “Beyond Guidelines: What Can We Learn from the Visual Information Seeking Mantra?”, 9th International Conference on Information Visualisation, (IV 2005), pp 110-118, 2005

[DeRo10] Robert DeLine, Kael Rowan. “Code canvas: zooming towards better development environments”, Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering (ICSE 2010), pp 207-210, 2010

[dETT99] G. di Battista, P. Eades, R. Tamasia, I. G. Tollis “Graph Drawing: Algorithms for the visualization of graphs”, Prentice-Hall, 1999.

[Dunh02] Margaret H. Dunham “Data Mining: Introductory and Advanced Topics”, Prentice-Hall, 2002.

[FrRe91] T. Fruchterman and E. Reingold. “Graph drawing by force-directed placement“, Software - Practice and Experience (SPE), 21(11), pp 1129–1164, 1991

[HaSk12] Immanuel Halupczok, Andre Schulz “Pinning balloons with perfect angles and optimal area”, Journal of Graph Algorithms and Applications, 16(4), pp 847-870, 2012

[HeMM00] Ivan Herman, Guy Melançon, and M. Scott Marshall. “Graph Visualization and Navigation in Information Visualization: A Survey”, IEEE Transactions on Visualization and Computer Graphics 6, pp 124-43. 2000

[ItKM10] Takao Ito, Kazuo Misue, Jiro Tanaka. “Drawing Clustered Bipartite Graphs in Multi-circular Style”, 14th International Conference on Information Visualisation (IV 2010), pp 23-28, 2010

[KaKa89] T. Kamada and S. Kawai. “An algorithm for drawing general undirected graphs”,

Information Process. Letters, 31(1), pp. 7–15, 1989

[MaVP13] Petros Manousis, Panos Vassiliadis, George Papastefanatos. "Automating the Adaptation of Evolving Data-Intensive Ecosystems", 32th International Conference on Conceptual Modeling (ER 2013), pp. 182-196, 2013.

[MELS95] Kazuo Misue, Peter Eades, Wei Lai, Kozo Sugiyama. "Layout Adjustment and the Mental Map", Journal of Visual Languages and Computing, 6(2) pp 183-210, 1995

[Meye98] Bernd Meyer. "Self-Organizing Graphs A Neural Network Perspective of Graph Layout'', Lecture Notes in Computer Science Volume 1547, pp 246-262, 1998

[Misu06] Kazuo Misue. "Drawing bipartite graphs as anchored maps", Asia-Pacific Symposium on Information Visualisation (APVIS) pp 169-177, 2006

[RoLN07] R. Rosenholtz, Y. Li, L.Nakano. "Measuring visual clutter", Journal of Vision, 7(2) pp 1-22, 2007

[Shne96] Ben Shneiderman. "The Eyes Have It: A Task by Data Type Taxonomy for Information Visualizations", Proceedings of the 1996 IEEE Symposium on Visual Languages, pp 336-343, 1996

[SixT06] Janet M. Six, Ioannis G. Tollis. "A framework and algorithms for circular drawings of graphs", J. Discrete Algorithms, 4(1), pp 25-50, 2006

[Tidw06] Jenifer Tidwell. "Designing interfaces - patterns for effective interaction design", O'Reilly, 2006

[Vass11] Panos Vassiliadis. "RADAR: Radial applications' depiction around relations for data-centric ecosystems", ICDE Workshops 2011, pp 62-67, 2011

[Ware04] C. Ware. "Information Visualization: perception for design", Morgan Kaufmann, 2nd edn., 2004

Appendix

Data preprocessing

In this section we will explain how we got out data (the figures bellow refer to Drupal data set). At first we had to obtain the schema of the database, which looked like this: (Fig. 41)

```
- MySQL dump 10.13  Distrib 5.5.31, for debian-linux-gnu (i686)
--
-- Host: localhost    Database: testdrupal
-- ------------------------------------------------------
-- Server version   5.5.31-0+wheezy1-log
--
-- Table structure for table `actions`
--

DROP TABLE IF EXISTS `actions`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!40101 SET character_set_client = utf8 */;
CREATE TABLE `actions` (
  `aid` varchar(255) NOT NULL DEFAULT '0' COMMENT 'Primary Key: Unique actions ID.',
  `type` varchar(32) NOT NULL DEFAULT '' COMMENT 'The object that that action acts on
(node, user, comment, system or custom types.)',
  `callback` varchar(255) NOT NULL DEFAULT '' COMMENT 'The callback function that
executes when the action runs.',
  `parameters` longblob NOT NULL COMMENT 'Parameters to be passed to the callback
function.',
  `label` varchar(255) NOT NULL DEFAULT '0' COMMENT 'Label of the action.',
  PRIMARY KEY (`aid`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COMMENT='Stores action information.';
/*!40101 SET character_set_client = @saved_cs_client */;
```

```
--
-- Table structure for table `authmap`
--

DROP TABLE IF EXISTS `authmap`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!40101 SET character_set_client = utf8 */;
CREATE TABLE `authmap` (
  `aid` int(10) unsigned NOT NULL AUTO_INCREMENT COMMENT 'Primary Key: Unique authmap
ID.',
  `uid` int(11) NOT NULL DEFAULT '0' COMMENT 'User’s users.uid.',
  `authname` varchar(128) NOT NULL DEFAULT '' COMMENT 'Unique authentication name.',
  `module` varchar(128) NOT NULL DEFAULT '' COMMENT 'Module which is controlling the
authentication.',
  PRIMARY KEY (`aid`),
  UNIQUE KEY `authname` (`authname`)
)  ENGINE=InnoDB  DEFAULT  CHARSET=utf8  COMMENT='Stores  distributed  authentication
mapping.';
/*!40101 SET character_set_client = @saved_cs_client */;
```

Fig. 41 Part of Drupal create table statements.

In order to make this parsable for the parser of Hecataeus to read we had to make some changes. The final form looks like this (Fig. 42):

```
CREATE TABLE  actions  (
   aid  varchar(255) NOT NULL,
   type  varchar(32) NOT NULL,
   callback  varchar(255) NOT NULL,
   parameters   varchar(255) NOT NULL,
   label  varchar(255) NOT NULL,
  PRIMARY KEY ( aid )
);

CREATE TABLE  authmap  (
```

```
   aid  int(10) NOT NULL,
   uid  int(11) NOT NULL,
   authname  varchar(128) NOT NULL,
   module  varchar(128) NOT NULL,
  PRIMARY KEY ( aid )
);
```

Fig. 42 Modified create table statements from Drupal data set to become parsable.

Basically we removed the comments, the DEFAULT property, the longblob data type, the engine and the unique keys, since our parser understands only primary keys. The next step was to get the queries that use those tables. To do that we grepped the Drupal system using the keyword "select" and this is what we got (Fig. 43):

```
./modules/taxonomy/taxonomy.module:        $result = db_query('SELECT t.tid FROM {taxonomy_term_data} t INNER JOIN {taxonomy_term_hierarchy} th ON th.tid = t.tid WHERE t.vid = :vid AND th.parent = 0', array(':vid' => $vid))->fetchCol();
./modules/taxonomy/taxonomy.module:    $names = db_query('SELECT name, machine_name, vid FROM {taxonomy_vocabulary}')->fetchAllAssoc('machine_name');
./modules/taxonomy/taxonomy.install:   $vocabularies = db_query("SELECT machine_name FROM {taxonomy_vocabulary}")->fetchCol();
./modules/taxonomy/taxonomy.install:        return db_query('SELECT v.* FROM {taxonomy_vocabulary} v ORDER BY v.weight, v.name')->fetchAllAssoc('vid', PDO::FETCH_OBJ);
./modules/taxonomy/taxonomy.install:       $vids = db_query('SELECT vid FROM {taxonomy_vocabulary}')->fetchCol();
./modules/taxonomy/taxonomy.install:   $result = db_query('SELECT v.*, n.type FROM {taxonomy_vocabulary} v LEFT JOIN {taxonomy_vocabulary_node_type} n ON v.vid = n.vid ORDER BY v.weight, v.name');
./modules/taxonomy/taxonomy.install:    $sandbox['total'] = db_query('SELECT COUNT(*) FROM {taxonomy_term_data} td INNER JOIN {taxonomy_term_node} tn ON td.tid = tn.tid INNER JOIN {node} n ON tn.nid = n.nid LEFT JOIN {node} n2 ON tn.vid = n2.vid')->fetchField();
```

```
./modules/taxonomy/taxonomy.install:         $result   =   db_query('SELECT   v.vid,
v.machine_name,     n.type     FROM     {taxonomy_vocabulary}     v     INNER     JOIN
{taxonomy_vocabulary_node_type} n ON v.vid = n.vid');
./modules/taxonomy/taxonomy.install:      $result  =  db_query_range('SELECT  vocab_id,
tid, nid, vid, type, created, sticky, is_current FROM {taxonomy_update_7005} ORDER BY
n', $sandbox['last'], $batch);
./modules/taxonomy/taxonomy.install:        $bundles  =  db_query('SELECT  bundle  FROM
{field_conFig._instance}  WHERE  field_name  =  :field_name',  array(':field_name'  =>
'taxonomyextra'))->fetchCol();
./modules/forum/forum.module:               $used   =   db_query_range('SELECT   1   FROM
{taxonomy_term_data} WHERE tid = :tid AND vid = :vid',0 , 1, array(
```

Fig. 43 Sample queries from Drupal data set.

Again we needed to make these parsable by Hecateus parser and the result looks like this (Fig. 44):

```
SELECT format FROM  filter_format ;
SELECT COUNT(*) FROM  taxonomy_term_data ;
SELECT COUNT(*) FROM  taxonomy_term_data ;
SELECT COUNT(*) FROM  taxonomy_term_data ;
SELECT COUNT(*) FROM  taxonomy_index  WHERE nid = 0 AND tid = 0 ;
SELECT COUNT(*) FROM  taxonomy_index  WHERE nid = 0 AND tid = 0 ;
SELECT COUNT(*) FROM  taxonomy_index  WHERE nid = 0 AND tid = 0 ;
SELECT COUNT(*) FROM  taxonomy_index  WHERE nid = 0 AND tid = 0 ;
SELECT COUNT(*) FROM  taxonomy_index  WHERE nid = 0 AND tid = 0 ;
SELECT COUNT(*) FROM  taxonomy_index  WHERE nid = 0 AND tid = 0 ;
SELECT COUNT(*) FROM  taxonomy_index  WHERE nid = 0 AND tid = 0 ;
SELECT COUNT(*) FROM  taxonomy_index  WHERE nid = 0 AND tid = 0 ;
SELECT COUNT(*) FROM  taxonomy_index  WHERE nid = 0 AND tid = 0 ;
SELECT COUNT(*) FROM  taxonomy_index  WHERE nid = 0 AND tid = 0 ;
SELECT COUNT(*) FROM  taxonomy_index  WHERE nid = 0 AND tid = 0 ;
SELECT t.tid FROM  taxonomy_term_data  t ,  taxonomy_term_hierarchy  th WHERE th.tid
```

```
= t.tid AND t.vid = 0 AND th.parent = 0;
SELECT name, machine_name, vid FROM  taxonomy_vocabulary ;
SELECT machine_name FROM  taxonomy_vocabulary ;
SELECT v.* FROM  taxonomy_vocabulary  v ORDER BY v.weight, v.name;
SELECT vid FROM  taxonomy_vocabulary ;
SELECT bundle FROM  field_conFig._instance  WHERE field_name = 0;
```

Fig. 44 A sample of modified queries to be parsable from taxonomy folder.

What we did was to remove the php variables, assign 0 as a value when needed and replaced queries that used joins with the same queries without joins.

Screenshots of all the datasets that we have used

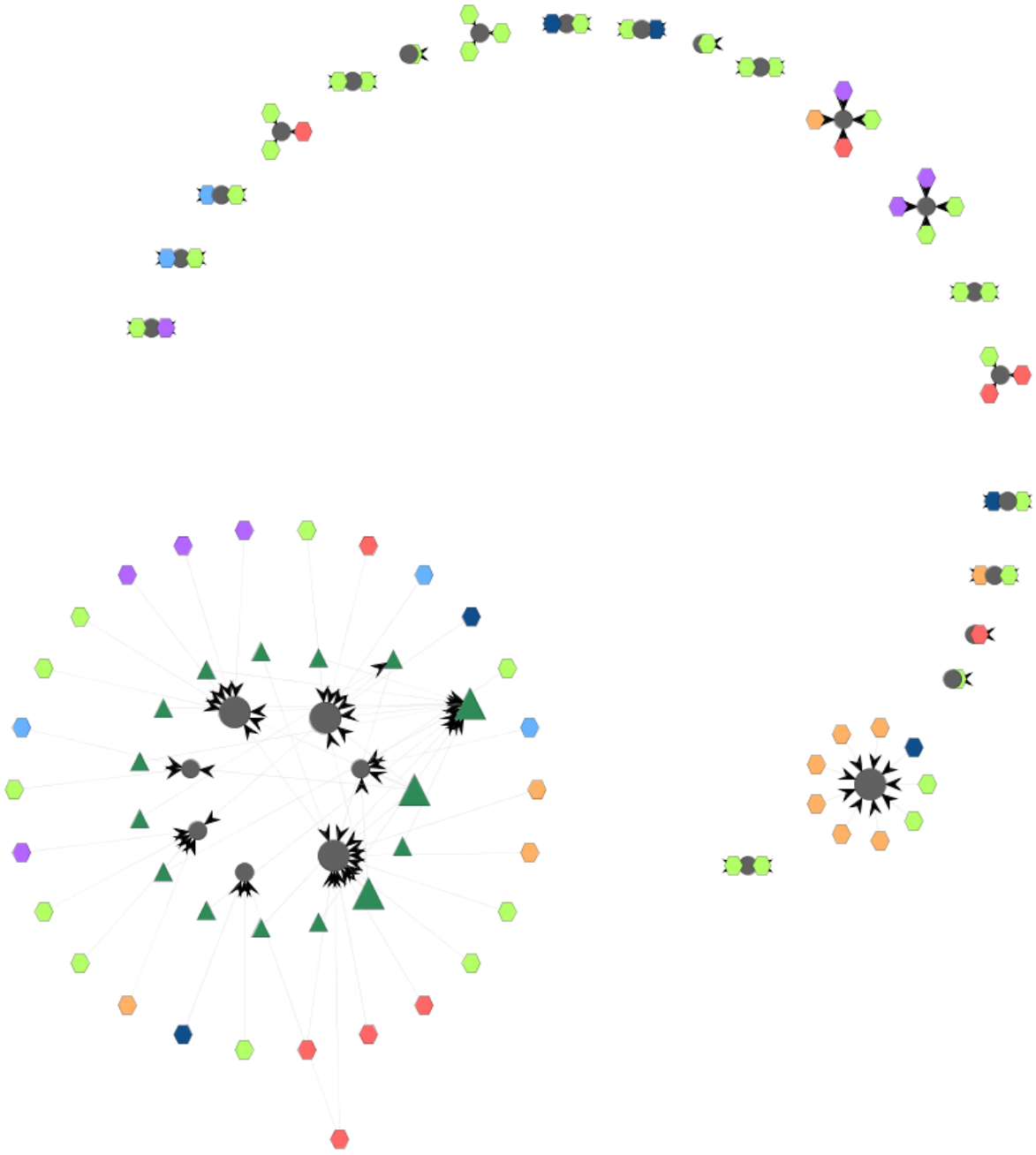


Fig. 45 BioSql data set visualized with the circular method.

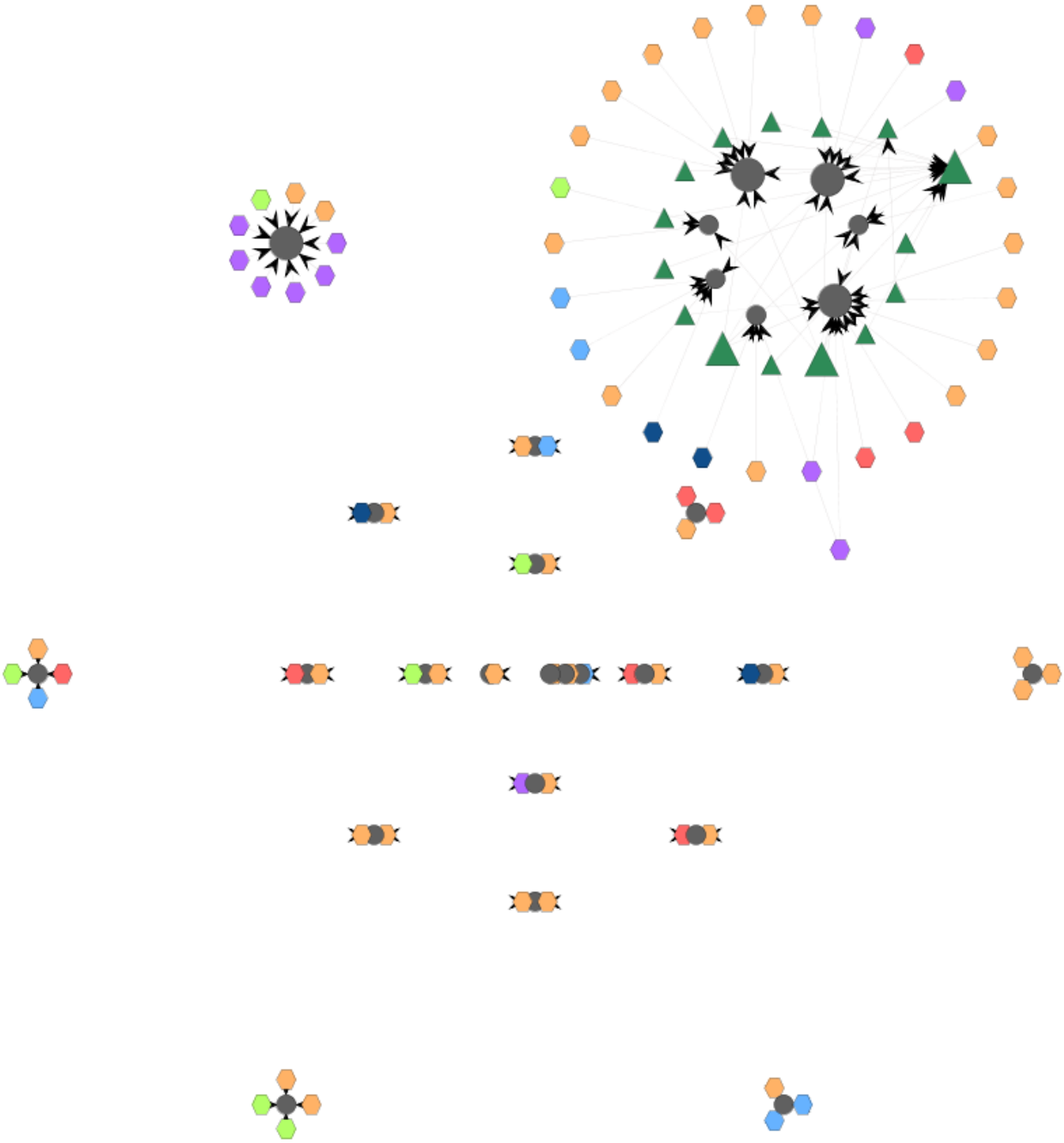


Fig. 46 BioSql data set visualized with the concentric circle method.

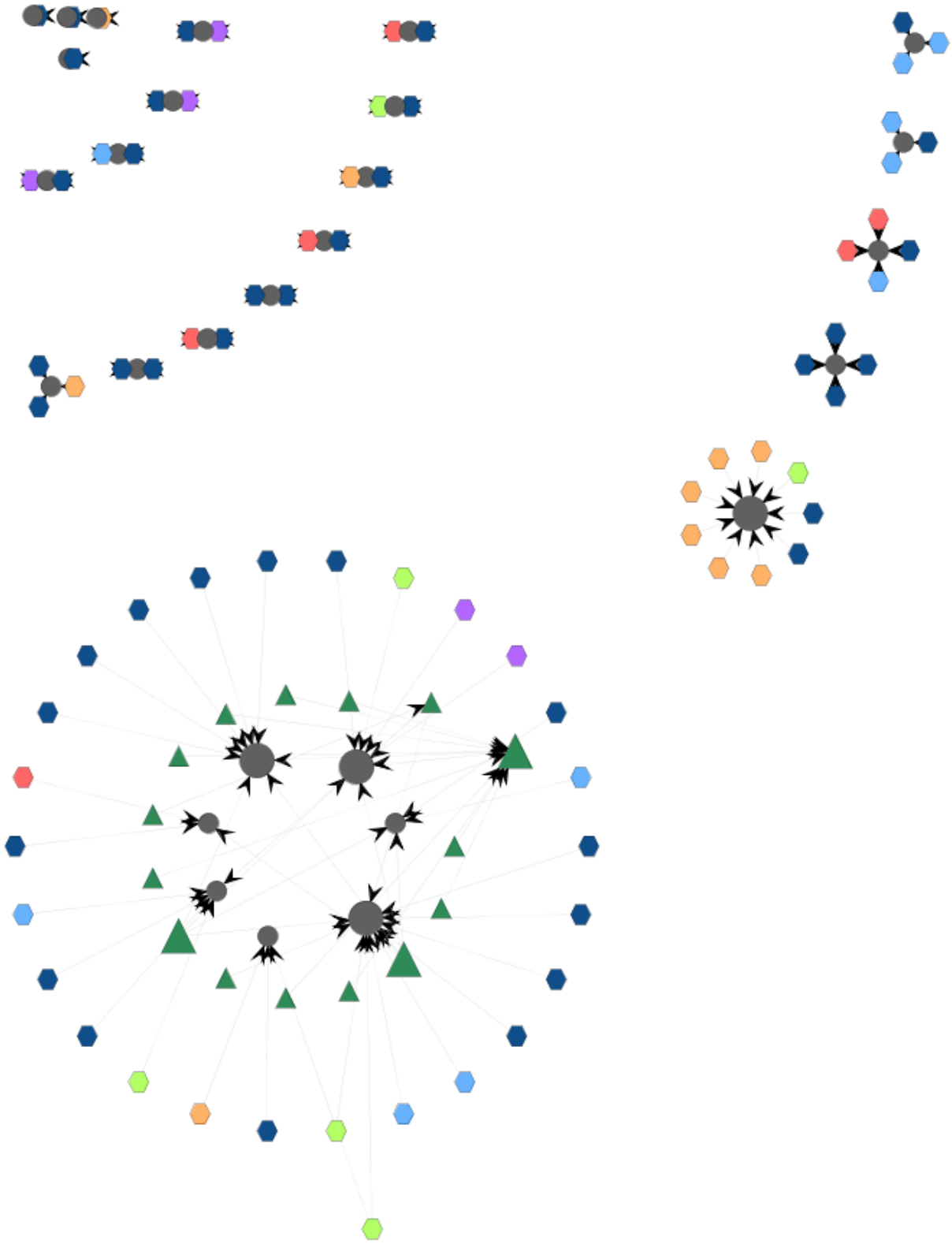


Fig. 47 BioSql data set visualized with the concentric arcs method

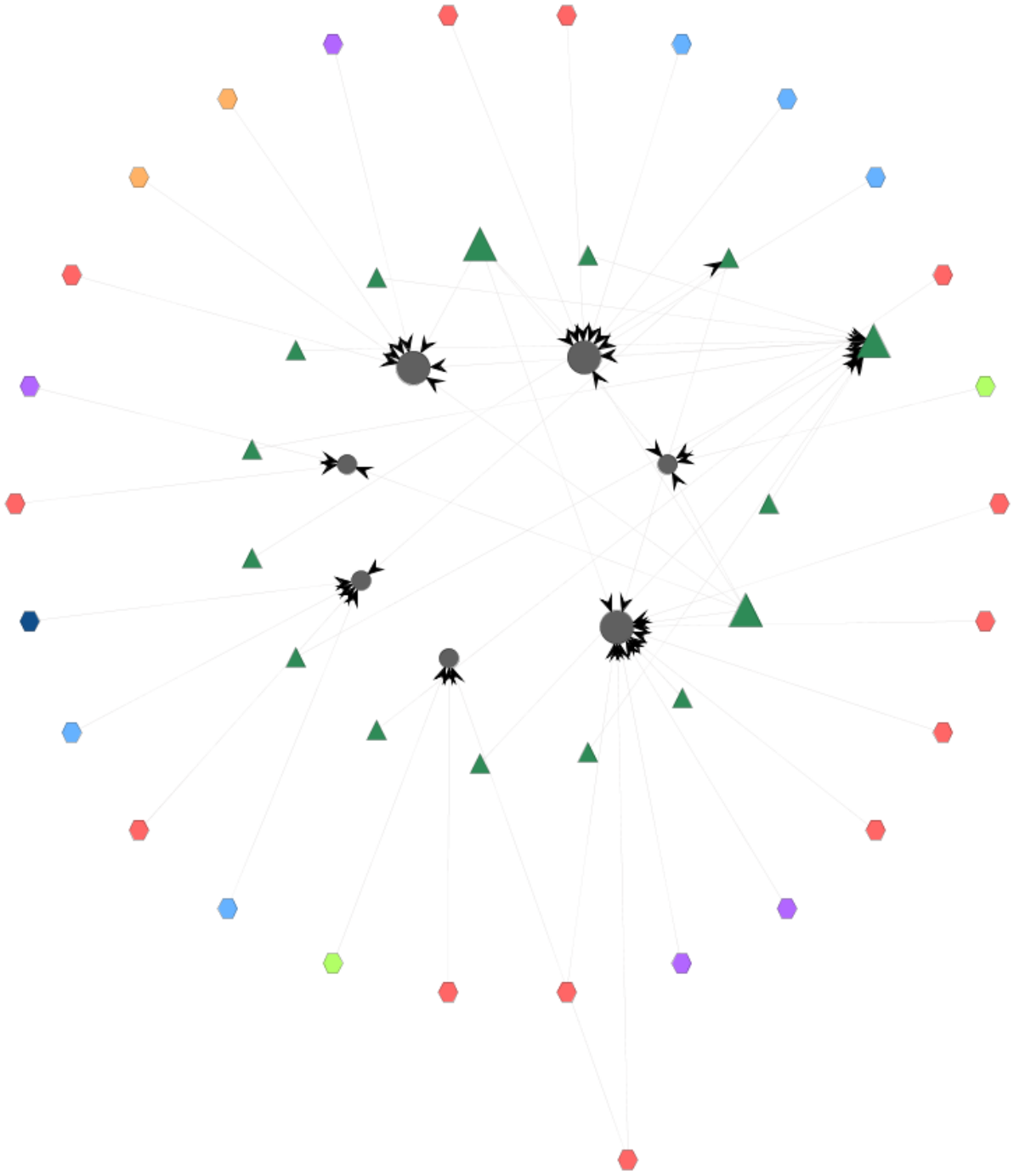


Fig. 48 Biggest cluster of BioSql data set.

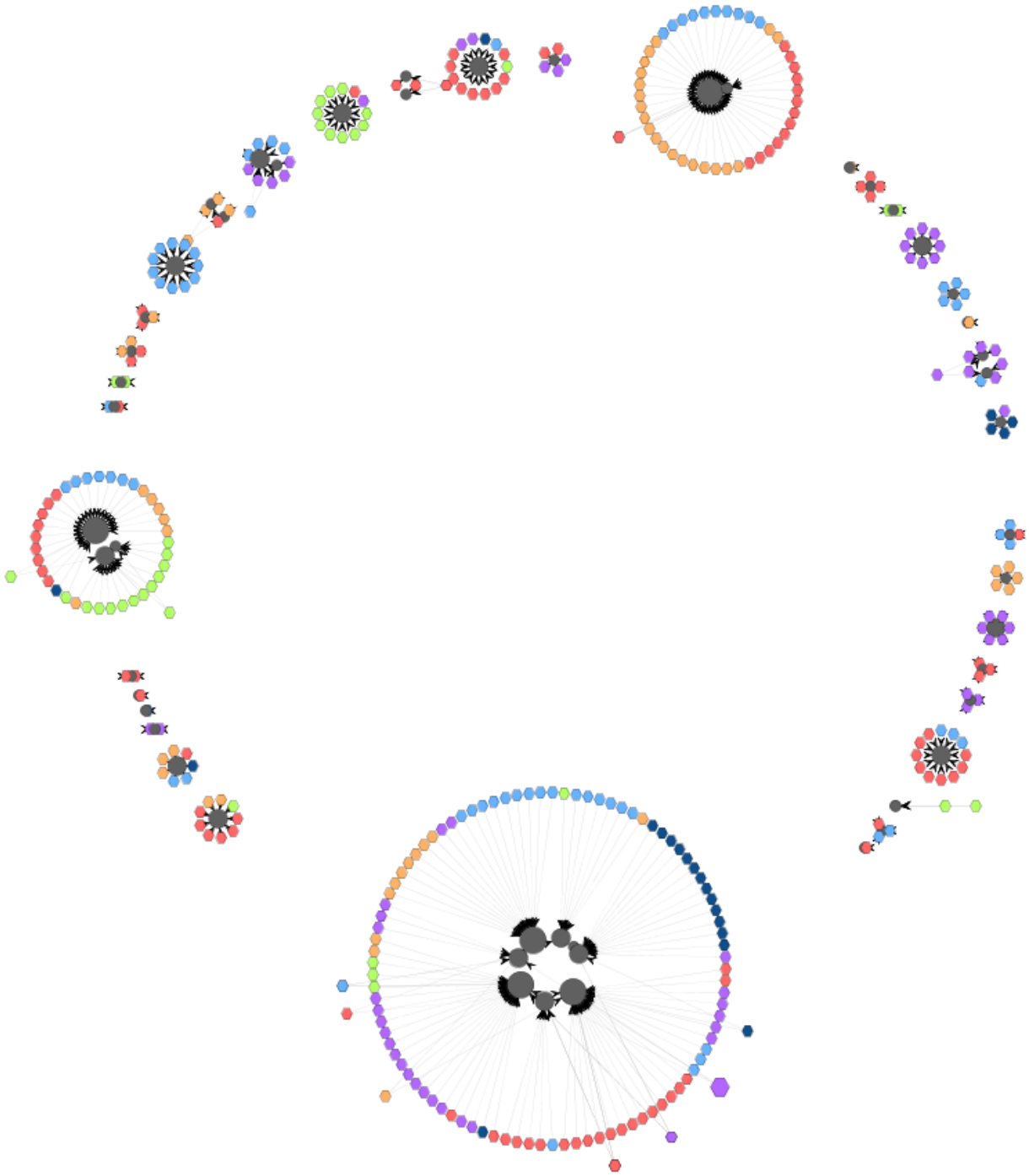


Fig. 49 Drupal data set visualized with the circular method.

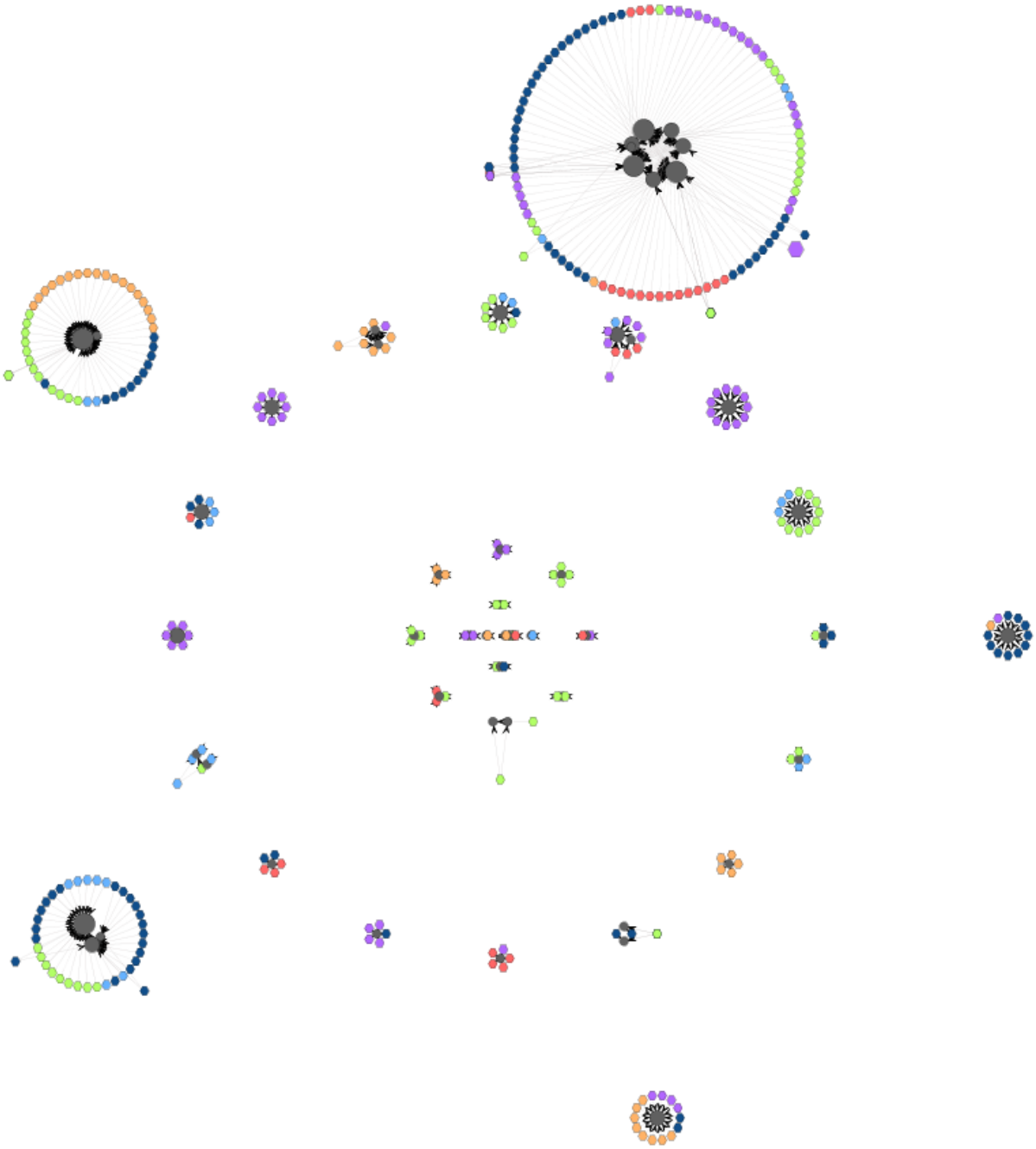


Fig. 50 Drupal data set visualized with the concentriccircles method.

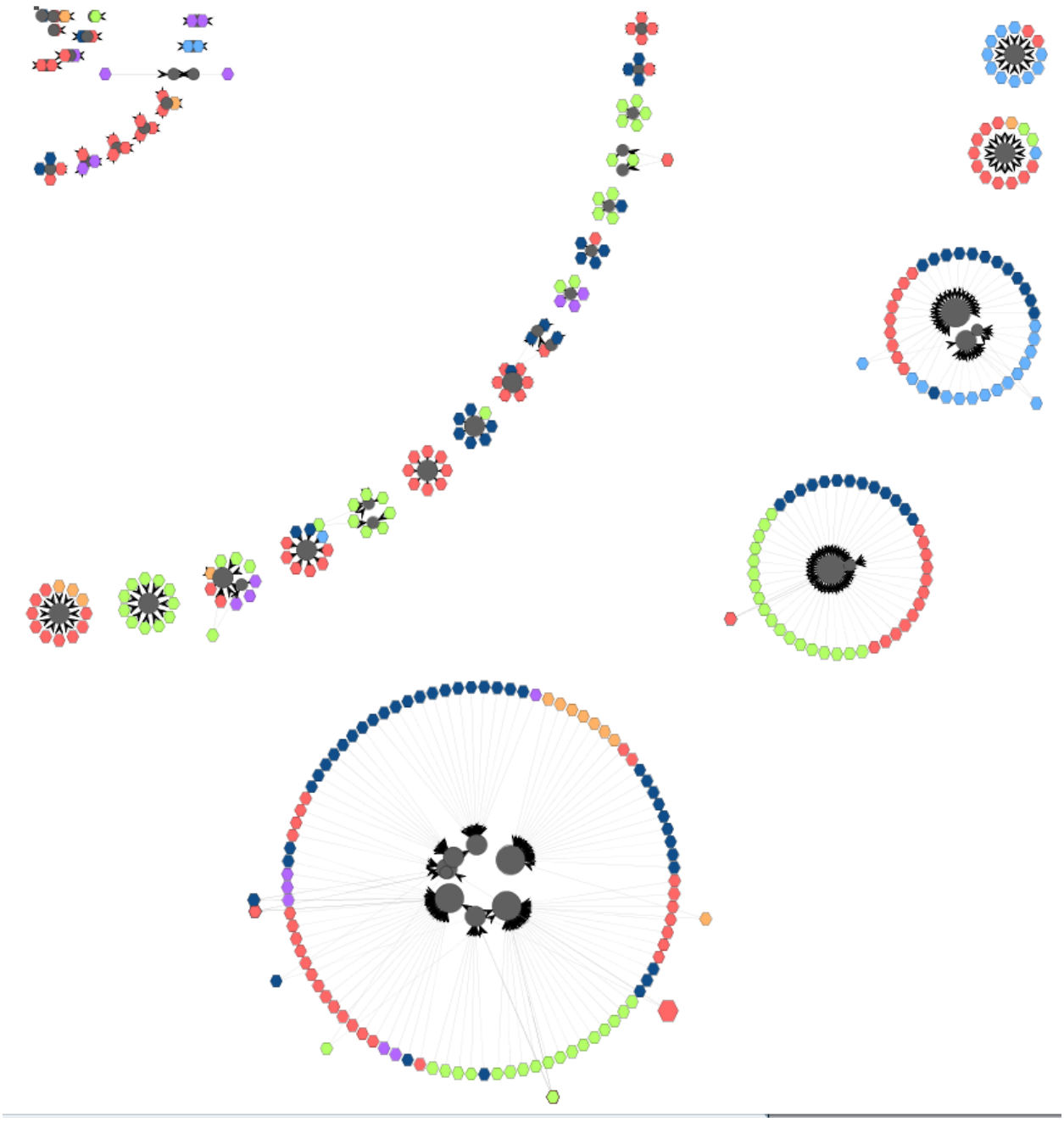


Fig. 51 Drupal data set visualized with the concentric arcs method.

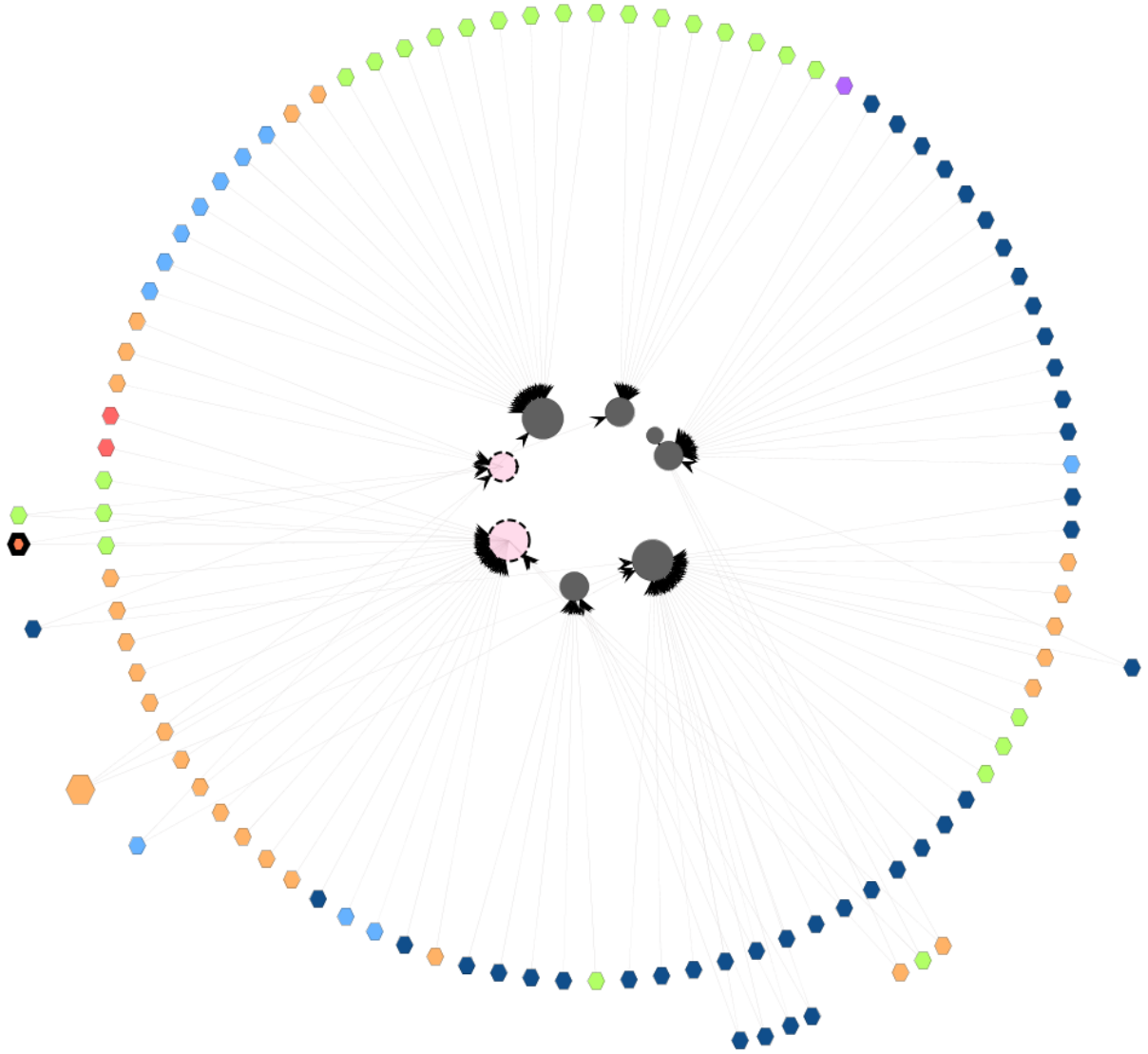


Fig. 52 Biggest cluster form Drupal data set.

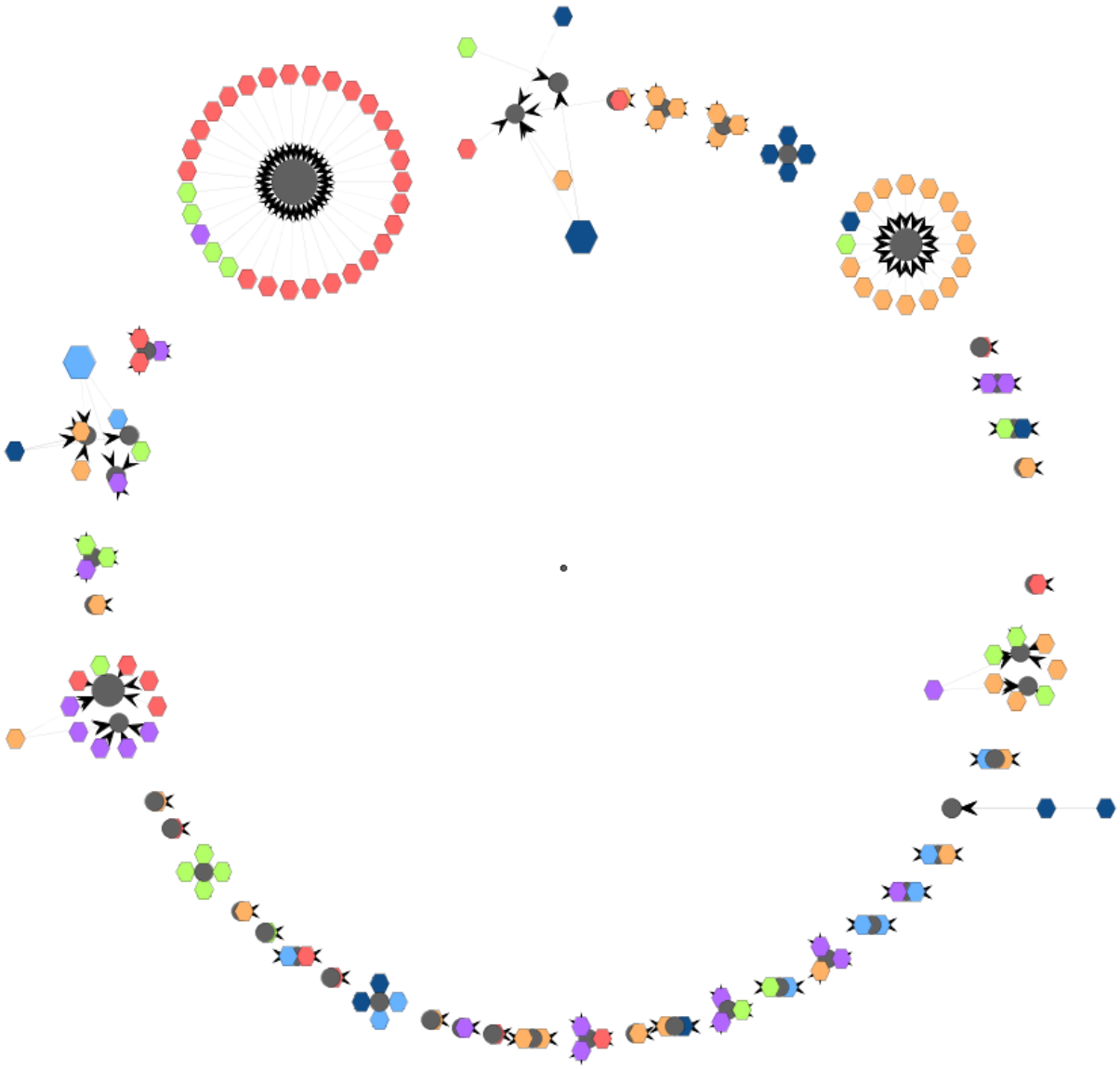


Fig. 53 ZenCart data set visualized with the circular method.

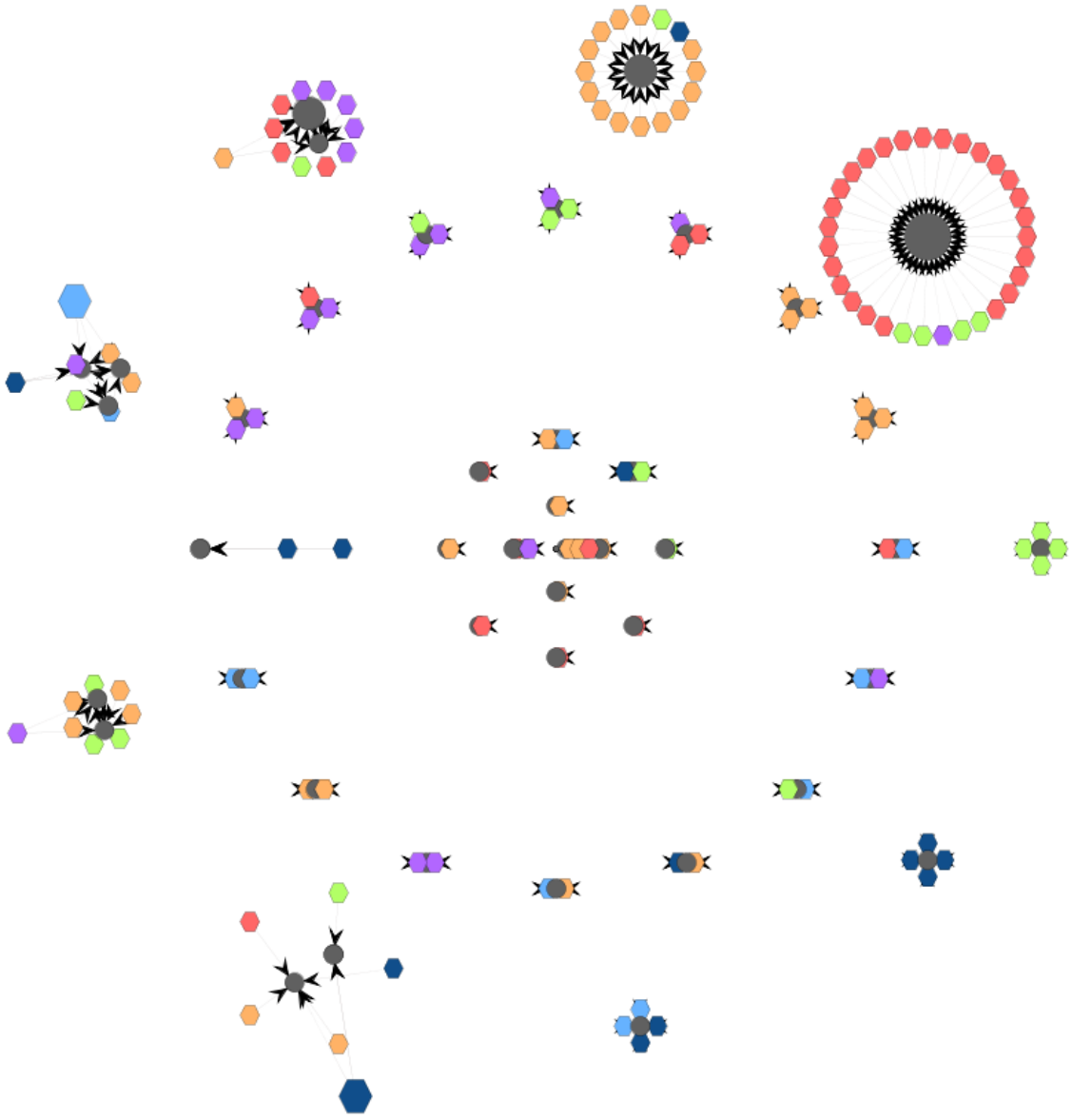


Fig. 54 ZenCart data set visualized with the concentric circles method.

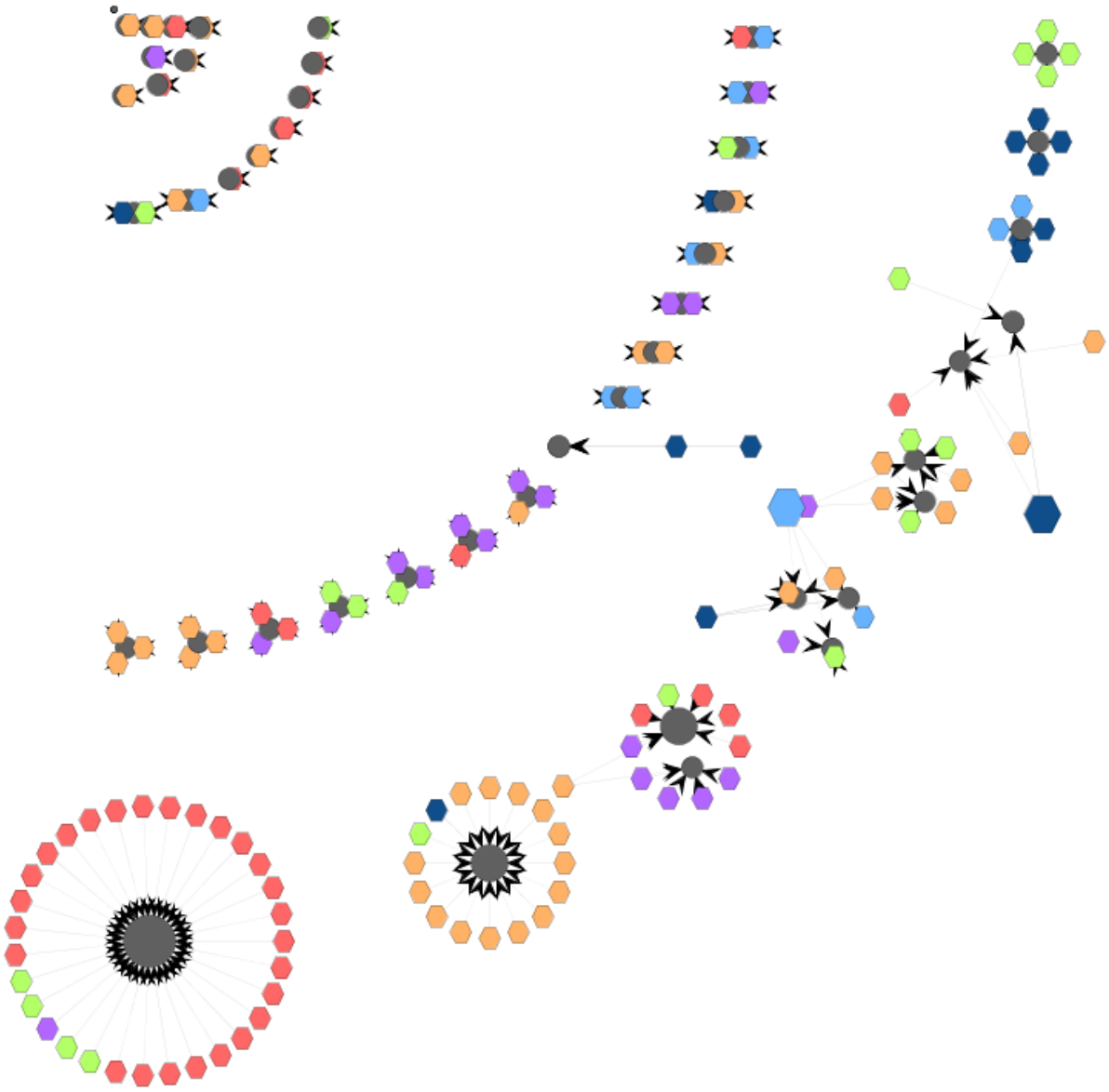


Fig. 55 ZenCart data set visualized with the concentric arcs method.

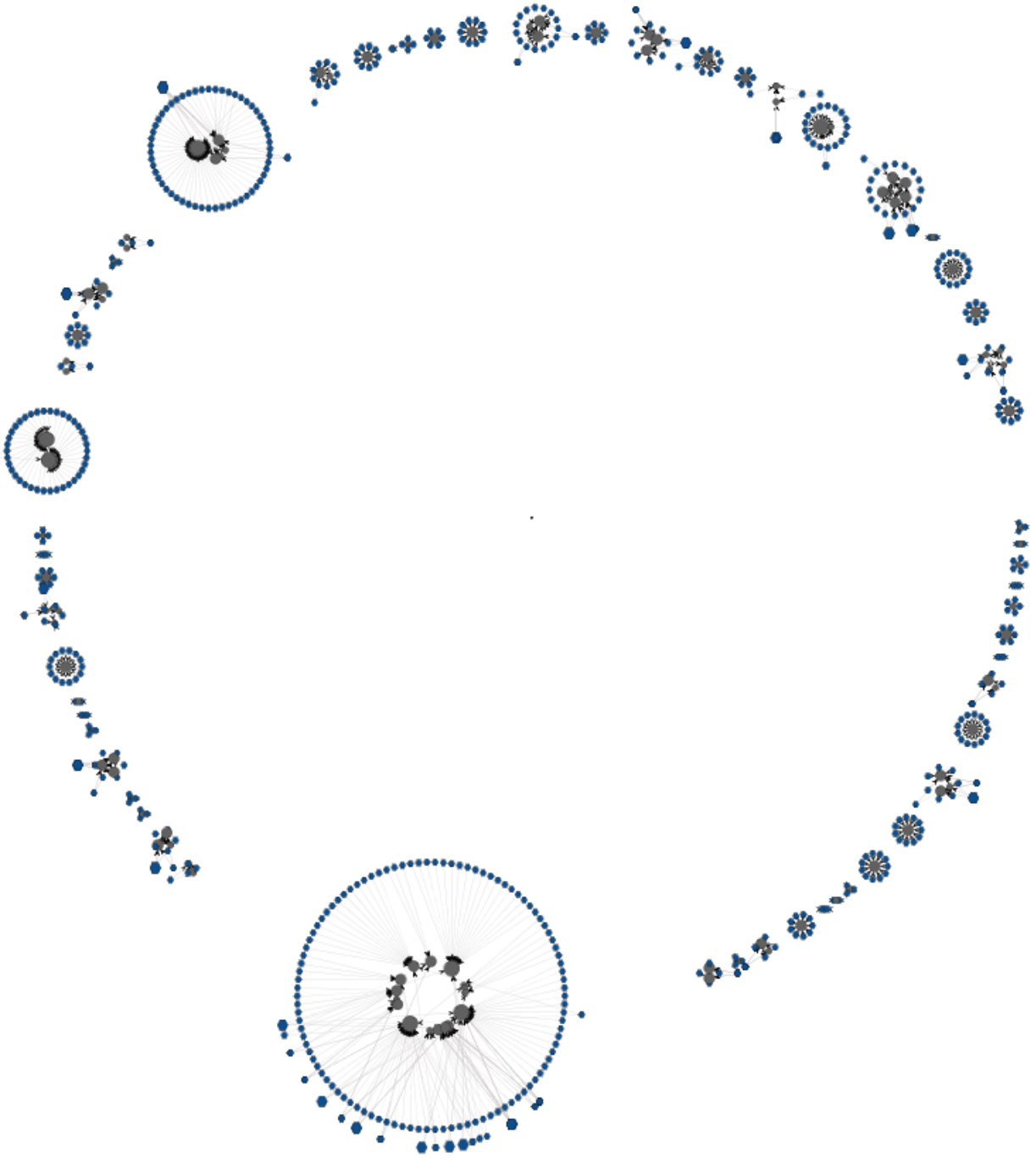


Fig. 56 OpenCart data set visualized with the circular method.

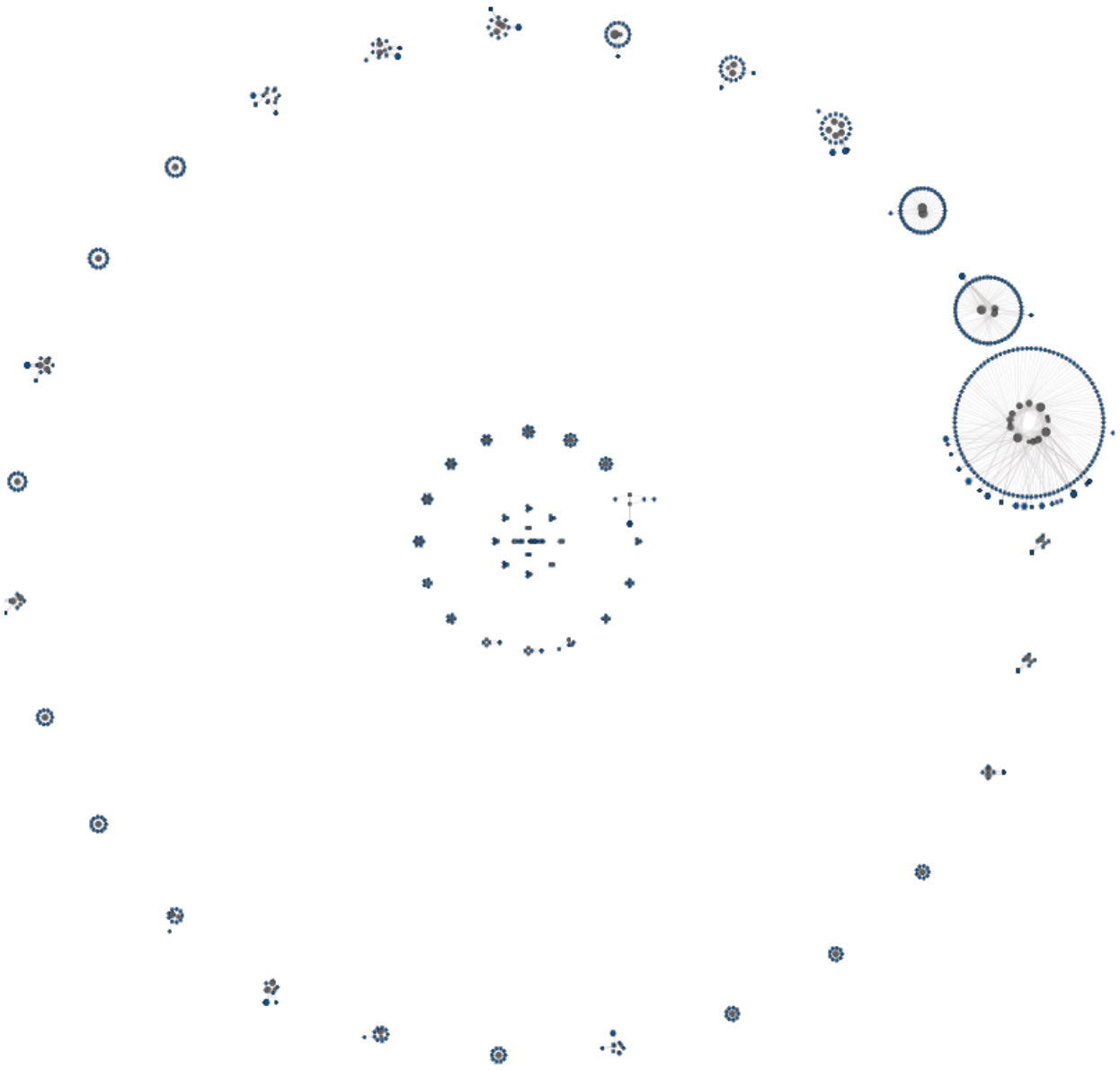


Fig. 57 OpenCart data set visualized with the concentric circles method.

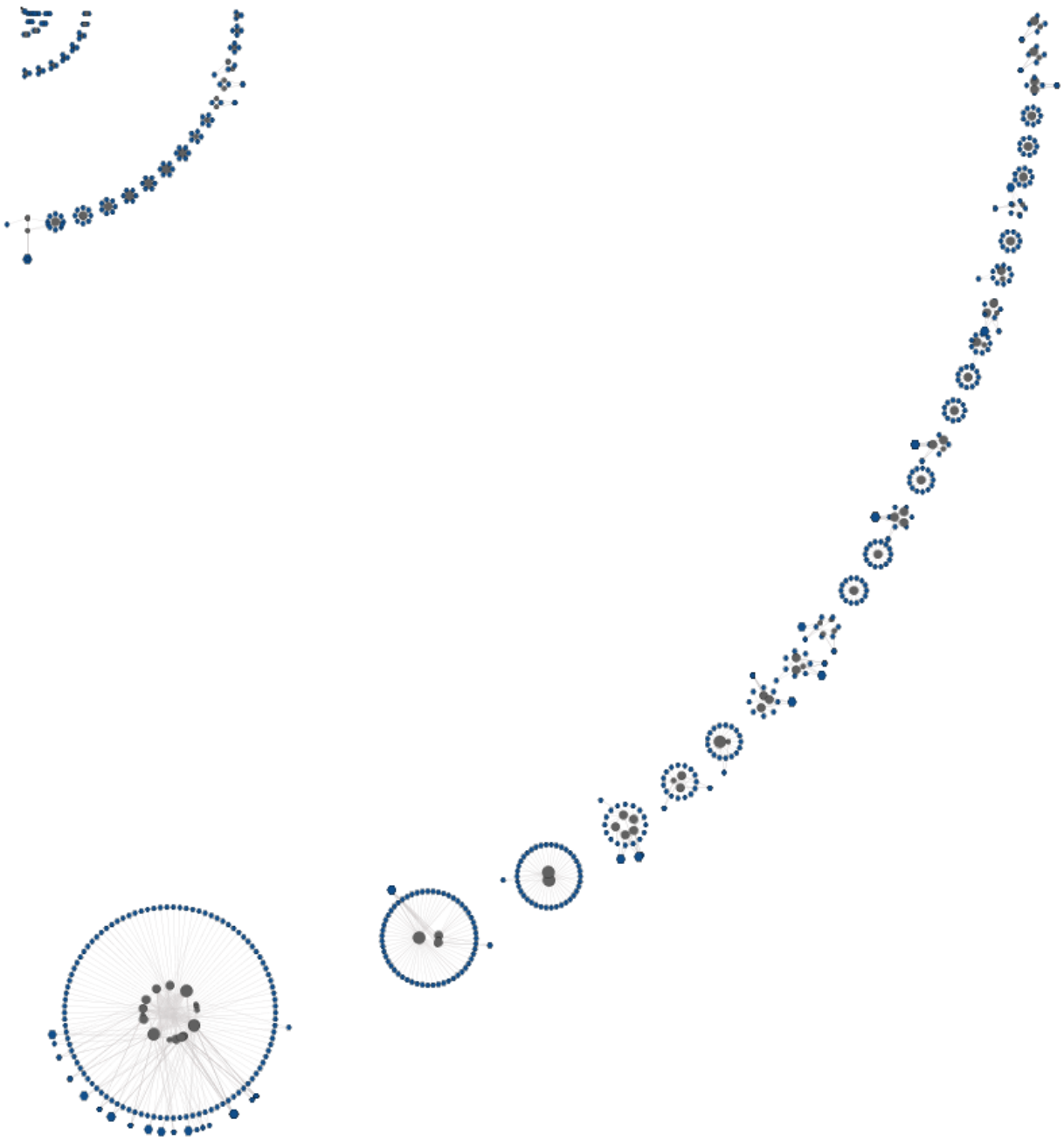


Fig. 58 OpenCart data set visualized with the concentric arcs method.

SHORT CV

Efthimia Kontogiannopoulou was born in Athens in 1989. She was admitted at Computer Science Department of the University of Ioannina in 2007, and she received her BSc degree in computer science in 2011. Later she attended the postgraduate program in University of Ioannina, Computer Science & Engineering Department. Her academic interests lie in the area of software engineering and data visualization.