Rank-Directed Layout of UML Class Diagrams
Hao Hu, Jun Fang, Zhengcai Lu, Fengfei Zhao, Zheng Qin
1School of Software, Tsinghua University, Beijing, China
2Department of Computer Science and Technology, Tsinghua University, Beijing, China
{haohu.th, zhaofengfei}@gmail.com, {fangjun06, luzc09}@mails.tsinghua.edu.cn, qingzh@tsinghua.edu.cn

ABSTRACT
UML class diagram layout is an important task in software visualization to enhance people’s comprehension about the systems. In this paper, we describe a novel UML class diagram layout algorithm, called rank-directed method, which captures the difference in relationships among classes and stresses significant classes. As a layout algorithm, rank-directed method supports the clustering of classes according to the inherent characteristics of classes. To recognize the significance of classes, we applied PageRank algorithms through abstracting relationships among different classes as the link among web pages. We assume that important classes have more relationships with other classes. To emphasize the important classes, rank-directed method adopts a sub graph layout method based on clustering of classes. We have developed a UML class diagram layout platform to evaluate our method. Our evaluation shows that rank-directed method could effectively recognize the important classes and layout the class diagram with higher readability than traditional layout methods do.

Categories and Subject Descriptors
D.2.7 [Software Engineering]: Distribution, Maintenance, and Enhancement—Restructuring, reverse engineering, and reengineering; D.1.7 [Programming Techniques]: Visual Programming

General Terms
Algorithms, Design, Measurement, Experimentation

Keywords
Class Diagram; Automatic Layout; PageRank; Forced-Directed.

1. INTRODUCTION
UML class diagram layout is an important task in software visualization to enhance people’s comprehension about the systems. It played as a key feature to the software modeling system in the practice of software engineering, especially in reverse engineering. An efficient class diagram layout could not only help engineers quickly understand the overview design of systems, but also assistant them to identify the most important classes when they are reading source code or trying to extend the existing software systems. In practice, the class diagram layout algorithm has been implemented as an independent module which is integrated into a software modeling system.

Intuitively, experienced software engineers are able to understand the existing systems without detailed documents based on their empirical knowledge about the relationship among classes, class types, class names and other features extracted from source code. These features can be learnt by experienced engineers but are too subtle and difficult to incorporate into existing layout engines. Even for experienced engineers, the process of manually identifying the important classes and relationships costs too much time. Therefore, leveraging this information, which implies the difference of importance among classes, into the class diagram layout algorithms may greatly help engineers understand the existing systems.

In this paper, we propose a method to layout class diagrams using the mined information from the structure of the diagrams. As shown in Figure 1, we mined the underlying importance of the classes according to relationships related to them and the inherent characters of the classes. Given a class diagram, our method could layout the class diagram with high readability by recognizing the important classes and the tight relationship among classes, which will help engineers quickly understand the system.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

SoftwareMining ’12, August 12, Beijing, China
Copyright © 2002 ACM 978-1-4503-1560-9/12/08... $10.00.
• We proposed a rank-based layout method to stress the classes with more significance as well as minimize crossing edges through classes ranking algorithm, which is similar to PageRank.

• We build our modeling system based on GMF framework and evaluate our methods by conducting in-the-field evaluation (extract from open source projects). The results show that our method could efficiently recognize and the important classes and emphasize them during the layout process, making the diagram more readable than traditional hierarchical layout method could do.

In the rest of the paper, we first formally define our problem in Section 2 and give an overview of our approach in Section 3. Then we elaborate on the weighted class direct graph construction, class importance calculation, sub-graph layout and combination in Section 4. After that, we reported on the evaluation in Section 5. Finally, we discuss related work in Section 6 and conclude this paper in Section 7.

2. PROBLEM STATEMENT

In this section, we first introduce some terms used in this paper then define our problem.

Definition 1. (Class Rank): A Class Rank is a double value ranges [0, 1], and it represents the probability that the class node be valued.

Definition 2. (Relation Rank): A Relation Rank is a double value ranges [0, 1], and it represents the importance weight of the relation edge.

Definition 3. (Weighted Class Directed Graph, WCDG): A Weighted Class Directed Graph $G_w$ is a directed graph, $G_w=(V_w, E_w)$, where $V_w$ is a set of nodes with class rank, and $E_w$ is a set of directed edges with relation rank.

Definition 4. (Sub Weighted Class Directed Graph, SWCDG): A Sub Weighted Class Directed Graph is a sub graph of WCDG.

Problem Definition: Given a class diagram $G(V, E)$, where $V$ represents the nodes of all the classes with position and size in two-dimension coordination and $E$ represent all directed relationships among these classes nodes $V$, calculate a mapped graph $G'(V', E')$, where $V'$ represents nodes $V$ with new position information.

3. OVERVIEW

As shown in figure 2, our rank-directed layout method consists of five major steps: Static Ranking Modeling, Dynamic Ranking Calculation, Sub Graph Clustering, Sub Graph Layout and Sub Graphs Combination. The first step is performed offline, and the other steps are performed online.

Static Ranking Modeling: This step will assign a static rank to each class node and relation edge among class nodes. We assume that the abstract class, interfaces and other classes with many public methods are more important comparatively. As to relationships, we assume that relations of different kinds are different in their importance. For example, aggregation relation may present the two classes related more tightly than other relations. We differentiate the importance of classes and relations empirically and then build a WCDG for each class diagram.

Dynamic Ranking Calculation: We assume that the classes linked from important classes are also important as people are more likely to be guided from an important class to its linked classes. Therefore, we could also build a random walk model for our scenario as PageRank algorithm did. We could abstract each weighted node as a web page and each weighted edge between nodes as a link between web pages. The differences between our random walk model and PageRank model lie in that our model take the inherent difference of classes and relations into consideration. We iteratively calculate new class rank for each node until the class ranks become stable. Then we could rank the importance of each class node.

Sub Graph Clustering: In order to emphasize the important node, it will be efficient if we put the node in the center of the graph which is composed of nodes related tightly with the central node. With the recognized important class nodes, we first separate the WCDG into many SWCDGs according to the important class nodes. We selected top k nodes according to the rank of their importance degree and perform clustering on these k central nodes. Each cluster represents all nodes in a SWCDG.

Sub Graph Layout: Until now, we have obtained k SWCDGs, and each graph has one central node. In this step, we layout the graph using magnetic spring model with a pillar magnetic field. In this magnetic field, we initiate the central node as the pillar of the magnetic field with the purpose to stress the important class node.

Sub Graphs Combination: Given k layout Sub Weighted Class Directed Graphs, we carry out our Rank-Directed method again on a new graph $G''(V'', E'')$, where each node in $V''$ represents a sub graph, each edge in $E''$ represents a relation edge between class nodes in different sub graphs. Then all the sub graphs are combined into a complete graph without destroying the cohesion within each sub graph.

Figure 2. Overview of Rank-directed layout method

4. RANK-DIRECTED LAYOUT

This section first describes the construction of static ranking model and the calculation of dynamic ranking, and then details the sub graph clustering and layout, finally illustrates the combination process.

4.1 Static Ranking Modeling

Definition 5. (importance degree): The importance degree of a node or edge is a predefined integer value denotes the importance of this node or edge.

In this paper, we divided all nodes into three categories: interface node, implementation node and utility node. Interface nodes mean the class nodes that played the role as API, such as public interfaces and public (abstract) classes. Implementation nodes represent the classes that are not released to users but hold the implementation of the public interfaces. Most of them contain the controlling functions [11]. Utility nodes represent the class nodes played as utilities to assistant implementation node; most of these classes are public static classes.

As our goal is to recognize the high level structure of the system, interface nodes are more important than the other kind of nodes as they represent the final goal of the system. Implementation nodes take the second place on importance degree. And the utilities nodes are the most unimportant nodes, because most of these classes only implemented functions surrounding the controlling functions.
Besides the classification of nodes, we also differentiate the importance degree of edges, because we assume that different relations have posed different influences on nodes. This is the key difference between our static ranking model and PageRank model.

In our method, the aggregation/combinations relation is the most important relation, followed by the inherit relation and the incidence relation.

4.2 Dynamic Ranking Calculation

Definition 6. (emphasis probability): The emphasis probability of a class node denotes the probability that this node would be emphasized.

We are trying to understand the importance degree of a class node $A$ from two aspects: first, the class node $A$ is important as its own characters; second, the class node $A$ is influenced by other important class nodes through the relationship between them. Therefore, we could explain the importance degree of a class node as the probability that this node is emphasized.

The Equation 1 shows the initial emphasis probability of a class node $C_i$:

$$P_0(C_i) = \frac{W(C_i)}{\sum_{j \in I} W(C_j)}$$

where $W(C_i)$ is the importance degree of node $C_i$ and $P_0(C_i)$ is the initial emphasis probability of node $C_i$. The influence from other class node on class node $C_i$ could be denoted as Equation 2:

$$PR(j,i) = P(C_j) \frac{R(j,i)}{\sum_{j \in I} R(j,k)}$$

where $R(j,i)$ is the relation rank of relation from class node $C_j$ to class node $C_i$. $PR(j,i)$ is the total influence from class node $C_j$ to class node $C_i$.

Assume a class node’s probability of being emphasized due to its inherent characters is $d$, then $1-d$ could represents the probability of being emphasized due to its relations. Then the emphasis probability of class node $C_i$ in the $k$th iteration could then be represented as Equation 3:

$$P_k(C_i) = d \times P_{k-1}(C_i) + (1-d) \times \sum_{j \in I} PR(j,i)$$

The above mentioned dynamic ranking calculation is an iterative process and in most cases it’ll stable after about $Iter_{m} = 200$ iterations when $d$ is set to 0.15. After all the dynamic ranking value has been calculated, we could easily recognize the top $K$ significant class nodes. We set a threshold value $\sigma = 0.20$, any nodes with rank value larger than $\sigma$ will be regarded as important class node besides the top 1 node.

4.3 Sub Graph Clustering

After recognizing the most important class nodes, we need to do effective layout to emphasize these class nodes. In this paper, we employed a central layout strategy in which the important classes are surrounded by their most related class nodes. In this step, we are going to cluster non-important class nodes to their most related important class nodes.

In order to cluster the class nodes, we first need to define the “distance” between nodes.

Definition 7. (node distance): The node distance between two class nodes is a measure denoted the closeness between them. It could be denoted as Equation 4:

$$\text{Dis}(n_u, n_v) = \max_{(i, j, k, l) \in \text{path}(n_u, n_v)} \left\{ \prod_{(i', j', k', l') \in (i, j, k, l)} R(i', i+1) \right\}$$

where $\text{path}(n_u, n_v)$ is the minimization collection of nodes that connected node $n_u$ and $n_v$. $\text{Dis}(n_u, n_v)$ is the node distance between class node $n_u$ and $n_v$. The node distance between nodes without any direct or indirect relations is 0.

Algorithm 1 sub graph clustering algorithm

1. FOREACH non-central node $v_i$ IN graph G
2. DO initialize the distance of $v_i$ to all centric nodes ($d_v < 1, d_v < 0$)
3. Initialize the cluster $S_v < \emptyset$
4. Build a priority queue $Q < V[G]$ according to their distances to $V_i (d_v)$
5. WHILE $Q != \emptyset$
6. DO $v_u < \text{extract} \text{the minimum node from} \text{Q}$
7. $S_v < S_v \cup v_u$
8. FOREACH vertex $v_k \in$ the adjacent vertexes of $v_u$
9. DO IF $d_{v_k} * \text{Dis(u, k)} > d_{v_k}$
10. $d_{v_k} < d_{v_k} * \text{Dis(u, k)}$
11. ENDIF
12. ENDFOREACH
13. ENDFOREACH
14. $v_c < \text{find a centric node in the set of centric nodes which d_v is larger than others}$
15. add $v_c$ to the sub graph of $v_c$
16. ENDFOREACH

Here we are going to demonstrate that Equation 4 will correctly cluster all nodes in a graph according to the important class nodes (central nodes of the sub graphs).

Lemma 1: Given a path $P$ connected a non-important class node $C_j$ with the central class node $U$ of the sub graph $S$ where $C_j$ lies, any class node $C_j'$ on the path $P$ also belongs to the sub graph $S_{C_j}$.

Proof: As it shown in Figure 3, U and V are the central class nodes of two sub graphs $S_u$ and $S_v$. $C_j$ belongs to $S_u$. We are going to prove that $C_j$ also belongs to $S_v$. Figure 3. Proving of Lemma 1
Assume \( C_j \notin S_u \) and \( C_j \in S_i \), then \( \text{Dist}(C_i, V) > \text{Dist}(C_j, U) \): so \( \text{Dist}(C_i, V) = \text{Dist}(C_i, C_j) \times \text{Dist}(C_j, V) > \text{Dist}(C_j, C_j) \times \text{Dist}(C_j, U) = \text{Dist}(C_j, U) \), then \( C_i \) also belongs to \( S_u \), which is contrary to our assumption.

**Theorem 1:** According to Equation 4, the graph could be clustered to \( k \) sub graphs, where either one of the following conditions will be satisfied: all class nodes in a sub graph are connected; the sub graph contains only one node.

Proof: Given any class node \( C_i \), if \( C_i \) belongs to \( S_u \), there are two conditions: first, node \( C_i \) is an isolated node, and it doesn’t have relations with other nodes; second, there exists a path between node \( C_i \) and the central node \( U \). According to Lemma 1, in the second condition, the nodes in the path from node \( C_i \) to \( U \) belong to \( S_u \). Then they are connected.

The Algorithm 1 shows the whole algorithm to do sub graph clustering.

### 4.4 Sub Graph Layout

In a sub graph, we have exactly one important class node and several other non-important nodes which connected with the important class node. We take the important class node as central node and arrange other node around the central node. In order to disperse other nodes, we employed the polar magnetic field model. The magnetic field is shown as figure 4:

![Polar magnetic field](image)

Figure 4. Polar magnetic field

where the magnetic force in position \((x, y)\) equals to

\[
m(x, y) = (x, y) / |(x, y)|
\]

(5)

The three forces in this model are as follows:

- **Spring Force** \((F_s)\): used to limit the distance between the two linked nodes, ensure that they could neither be too far or too close. Equation 6 shows the spring force:

\[
F_s = C_s \log(d / k)
\]

(6)

where \( d \) is the idea distance between two nodes, \( k \) is the real distance, \( C_s \) is a constant parameter.

- **Repulsive Force** \((F_r)\): used to limit the distance between non-neighborhood nodes. We take the class rank value of each node into consideration when measure \( F_r \), because we assume the distances between important classes should be more obvious and the non-important class nodes should be more close to the important class nodes. Equation 7 reflects this consideration:

\[
F_r = C_r P_i P_j / d^2
\]

(7)

where \( P_i, P_j \) are the class rank value of non-neighborhood class node \( i \) and \( j \), \( C_r \) is a constant parameter.

- **Rotation Force** \((F_{rot})\): used to stretch edges in magnetic field and consider much more about the angle between the magnetic line of force and the magnetic needle (edge), the length of the edge and the degree of the magnetic force. Equation 8 shows \( F_{rot} \):

\[
F_{rot} = C_{rot} \log(\alpha, \beta, \theta)
\]

(8)

where \( b \) is the constant force of magnetic field, \( d \) is the distance of the edge, \( \theta \) is the small angle between magnetic needle and magnetic line of force, \( \alpha, \beta \) and \( C_{rot} \) are constant parameters.

Before we use the model mentioned above to layout the graph, we need to adjust the orientation of edges from these near central nodes to these far from central nodes temporarily. Because the magnetic forces on source nodes and target nodes have different orientation.

When all nodes are placed in a right position, we employed the Shortest Path Connection Router (SPCR) method to get the orthogonal line.

### 4.5 Sub Graphs Combination

This is the last step of our rank-directed method. We first build a new graph where each node represents a sub graph, each edge represents an edge cross two sub graphs, and then we layout the new graph using the method mentioned in section 4.4.

### 5. EVALUATION

#### 5.1 Evaluation Indicators

In this paper, we not only care about the aesthetic criteria, such as crossing minimization and so on, but also care the semantic readability of the class diagram. So besides traditional indicators, we proposed two new indicators to evaluate the performance of our method and make comparison with popular hierarchical method and magnetic field method.

- **Cohesion factor**: used to indicate the degree of cohesion in a set of classes. It could be represented as follows:

\[
F_{cohesion} = \frac{\text{avg}(d)}{\text{avg}(\text{dis}(u, v))}
\]

(9)

where \( d \) is the predefined idea distance, \( \text{dis}(u, v) \) is the real distance between class node \( u \) and \( v \). In theory, the larger the cohesion factor, the better the readability.

- **Coupling factor**: used to indicate the degree of coupling among different class sets. Its definition is shown as Equation 10:

\[
F_{coupling} = \frac{d}{\text{avg}(\sum \text{dis}(u, v))}
\]

(10)

where \( d \) and \( \text{dis}(u, v) \) are with the same meaning with the above. In theory, the less the coupled factor, the better the readability of the class diagram.

#### 5.2 Experimental setting

We developed a UML class diagram layout platform based on Graphical Modeling Framework (GMF), and implemented our algorithms as a plugin. In the evaluation section, we are going to use in-the-field class diagrams to demonstrate the effect of our method.
We extract a class diagram of JUnit project to show the result of our method. The figure 5 shows the result of the final layout result of hierarchical method (a) and our rank-directed method (b) of JUnit class diagram.

In this diagram, we have the parameters values as follows:
- the dynamic value iteration count Iter_d: 200
- the magnetic force iteration count Iter_m: 400–1600
- other parameters: $\alpha = -0.5$, $\beta = 1.0$, $C_s = 5.0$, $C_r = 1.0$, $C_m = 1.0$, $k = 150$, $b \in (0, 12)$

In figure 5, the rank-directed method recognized two important classes (interfaces) – TestDecorator and Test, other classes are layout around them, making the whole diagram more readable.

5.3 In-the-Field Evaluation
In our evaluation environment, we make comparison of hierarchical method, traditional magnetic force method and our rank-directed method on the indicators of cohesion and coupling. Figure 6 shows how the two indicators mentioned above changed as the magnetic field force changes.

From figure 6, we could conclude that in our test scenario, the rank directed method owns higher cohesion than hierarchical method and magnetic directed method, and in coupling factor, it also performs better than magnetic-directed method.

In order to continue compare our method with traditional method, we choose an aesthetic criteria, edge ratio, to do comparison. Edge ratio means the ratio of the average distances to the predefined idea distance. Figure 7 shows the comparison result.

From figure 7, it’s obvious that rank directed method has an average longer distance than hierarchical method. This is cause by the aggregation of classes with tight semantic relation. The aggregation will enlarge the distance between classes separated in different sub graphs.

In a word, we sacrifice a little readability in aesthetic criteria, but we gained much more readability in semantic.

6. RELATED WORK
Rank-Directed layout algorithm builds on a large body of existing literature in programming techniques, information retrieval and software visualization. We focus on four most related areas: graph drawing, software visualization, software analysis and graph mining.

Graph Drawing (GD). Graph drawing has always been an active research field, and it focus on finding a drawing of a given graph that is visually pleasant, based on some aesthetic criteria [1]. Research topic about aesthetic criteria has changed from cross edge minimization, bends minimization several years ago to edge
bundling in recent years [4]. Graph layout and edge router composed of major content of graph drawing. There exist many layout tools which support user-defined drawing convention and exterior appearance. GraphViz [5] is the most popular one.

The most popular algorithms in GD are Hierarchical Method [2], Force-Directed Method [3]. Hierarchical method performs well on cross minimization. Force-Directed method, such as spring model and magnetic field model, trends to add physical force models to the graph. And they are easy to implemented and extended to other constraints. In contrast to these traditional methods which aim at aesthetic criteria only, our method put more emphasis on the recognition of the important nodes and the graph layout by considering the semantic characters.

**Software Visualization.** Software visualization is a major branch of information visualization as there are many kinds of data and views (class diagram, activity diagram, data flow graph and so on) related with software.

In class diagram layout, Seemann [6] was the first one who introduce Hierarchical method into class diagram layout. The recent aesthetic criteria research on class diagram is [7]. The researches on class diagram layout have been switching from simple aesthetic criteria to rules containing domain knowledge. Many researchers now are considering add design pattern knowledge into layout and doing test on the effects of layout with design patterns [8]. In this paper, we are on the way of adding more domain knowledge into layout to help engineers understand the design of the system more efficiently.

**Software Analysis.** Software analysis is a major branch in Software Engineering. In Software analysis, one of the most important goals is to identify the critical software components, using both static analysis and dynamic analysis.

In recent years, researchers are using clustering technology to identify the key classes/components in software systems. Andy et al. proposed a detection approach that based on dynamic coupling and web-mining [11]. Gholam R. S. et al. proposed a method based on clustering of use cases to identify software components and their responsibilities [12]. Y. Chiricota et al. proposed a metric based clustering method to capture software component [13]. Most of these methods are focus on identifying the key component in software system using only clustering method. In addition, they mined little information from the static relationship among classes. In contrast to these methods, we focus on identifying the key classes by ranking the classes metrics which are mined from static relationships among classes.

**Graph Mining.** In recent years there has been a lot of interest in the research and industry towards graph mining, which has been driven by increasing data from social networks. Ranking, derived from information retrieval, plays an important role in graph mining.

PageRank [9] algorithm, invented by google, is one of the most popular algorithms in ranking due to its good performance. In recent years, many extensions based on PageRank has been proposed to mining knowledge in graphs, such as TunkRank algorithm [10] used to compute user influences on social network. Fan Chung et al. proposed a method to find smaller local communities within a larger graph by leveraging graph’s structure information mined from PageRank [14]. Yizhou Sun et al. conducted a research on heterogeneous graphs by Integraing clustering with ranking [15]. In contrast to their work, we conduct clustering based on ranking, instead of ranking in each cluster.

**7. CONCLUSIONS**

This paper presents an approach of layouting the UML class diagram to help engineers quickly identify the key classes as well as the most important relations among classes. In our method, we first construct a weighted class directed graph, and then rank the class nodes according to their class rank and cluster all class nodes into sub weighted class directed graphs. After layout each sub graph with enhanced magnetic layout method, we combined all sub graphs into a complete graph and then mapped the graph to class diagram. We build a layout platform based on GMF framework, and evaluate our method using in-the-field diagrams extracted from open source code projects. The results show that our method significantly improves the cohesion of the diagram and effectively identifies the important classes. Besides, our method is not only suitable for UML class diagram, but also effective for other domain specific diagrams.

In the future, we are going to do more experiment on the performance evaluation, e.g. extract larger and more open source projects to validate the correctness of our semantic aggregation and the effectiveness of readability. Besides, we’re going to discuss how the different values of all parameters could influence the final results. In addition, we will also take more consideration on how to deal with the class nodes with obviously variant sizes.

**8. ACKNOWLEDGMENTS**

The authors would like to Hongyu Zhang for his encouragement and suggestions on an early draft, and the anonymous reviewers for their insightful comments and advices.

**9. REFERENCES**


