J Vis (2025) 28:181–204
https://doi.org/10.1007/s12650-024-01018-5

REGULAR PAPER

Ayana Murakami · Takayuki Itoh

Flexible optimization of hierarchical graph layout by genetic algorithm with various conditions

Received: 27 February 2024 / Revised: 19 June 2024 / Accepted: 3 July 2024 / Published online: 14 August 2024

Abstract Graph layouts visualize relationships among data entities, where nodes represent individual entities and edges represent their relationships. Hierarchical graph layouts efficiently provide an overview of large-scale graphs, where nodes form clusters (called metanodes in this paper) based on their properties. Here, it is challenging to determine layouts for large-scale graphs, particularly hierarchical ones. Although various graph layout drawing methods, such as force-directed layout, have been discussed so far, the quality of a layout heavily relies on the initial positions of nodes or metanodes. Furthermore, it is more challenging to obtain layouts where specific desired metanodes stand out. This paper presents a layout optimization method for hierarchical graphs using a genetic algorithm (GA). Our method allows for the consistent improvement of layouts compared to relying solely on an existing algorithm for generating hierarchical graph layouts. In our implementation, first, hierarchical graph layouts are generated by applying an existing algorithm multiple times. Then, they are evaluated by specific metrics for hierarchical graph layouts and optimized using GA. Consequently, optimal layouts for these metrics are obtained. The paper also presents particular examples of layouts optimized under different conditions using a co-authorship graph dataset.
Keywords Hierarchical graph layout · Genetic algorithm · Optimization

1 Introduction

Graphs represent data elements as nodes and their relationships as edges. In recent years, graph visualization methods have been applied in various fields. (Aslam et al. (2020), Valeri and Baggio (2021), Fregnan et al. (2023)) The readability of a graph is significantly influenced by its layout (Yoghourdjian et al. 2018), which denotes the positioning of nodes and the design of edges. In particular, hierarchical graphs consisting of multiple clusters of nodes (called metanodes in the paper) are known as an effective representation to visualize the overview of large graphs (Didimo and Montecchiani 2014). However, it is extremely challenging to generate a highly readable layout due to the complexity of connections.

Methods to generate graph layouts have been actively discussed (Gansner et al. 2010; Itoh and Klein 2015; Suh et al. 2019). However, each method has its distinctive characteristics, and it can be difficult to adjust the output layouts flexibly based on the purpose of visualization. In recent years, techniques for generating graph layouts through the application of deep learning have been actively discussed. As an example, methods that treat graphs as sequences and use RNN (recurrent neural network) (You et al. 2018)

Ayana Murakami and Takayuki Itoh have contributed equally to this work.

A. Murakami (✉) · T. Itoh
Ochanomizu University, Otsuka, Bunkyo-ku, Tokyo 1128610, Japan
E-mail: murakami.ayana@is.ocha.ac.jp

T. Itoh
E-mail: itot@is.ocha.ac.jp

or LSTM (long short-term memory) (Wang et al. 2019) as learning models have been proposed. However, these studies have yet to be applied to hierarchical graph layouts. These methods are still challenging to achieve the desired layouts. Another disadvantage is the extended training time for the model. As the size of the graphs increases, it takes more time to train the model and generate output layouts.

Evaluation for graph layouts is another issue that has been discussed for a long time. Various studies have been published on evaluations that consider both graph structure and the appearance of overall graph layouts. Evaluation metrics considering graph structure have been actively discussed for many years. Examples of the metrics include the number of edge crossings, the angle of edge crossings, and graph layout symmetry (Purchase et al. 1996; Purchase 1997, 2002; Ware et al. 2002; Huang et al. 2016). On the other hand, evaluation metrics considering the appearance of graph layouts are less discussed compared to the other (Biedl et al. 1998; Battista et al. 1998; Eades et al. 2017). These metrics assess, for example, the spread of overall nodes (Taylor and Rodgers 2005). However, there were very few studies on evaluation methods specialized for visualization results of hierarchical graphs. Recently, Liu et al. (2020) proposed Sprawlter, a new numerical evaluation metric specialized for hierarchical graphs. Sprawlter consists of two metrics: Clutter, evaluating the overlaps of graph elements, and Sprawl, assessing the waste of space. In the study of Liu et al. (2020), Sprawlter has been used only as a metric for evaluating the layout results of hierarchical graphs, without being applied for the optimization of graph layouts.

In this paper, we propose a method to optimize hierarchical graph layout using mathematical optimization techniques to solve the above problems. Our approach first applies an existing method to generate multiple distinct hierarchical graph layouts to the provided dataset. Subsequently, the optimization process is executed using these layouts as an initial generation. The Sprawl and Clutter metrics are applied as fitness functions of the optimization. These metrics were proposed in Liu et al. (2020), and the Sprawlter combines them. The problem is treated as a multi-objective optimization problem since there are two fitness functions. A genetic algorithm (GA) is adopted to optimize the layouts in our study. The advantages of GA are that they can search a wide solution space and are unlikely to fall into local solutions.

In this paper, six experiments were conducted using a co-authorship dataset, involving two types of hierarchical graph layouts and three different conditions. Our method can support multi-level graphs, but this paper just shows one-level clustering algorithms and results. As a result, the experiments produced optimized layouts. The evaluation results for these layouts are visualized in a scatterplot. This scatterplot provides insights into the distinctive features of each layout based on the evaluation results. Our contribution is optimizing hierarchical graph layouts generated by an existing algorithm through the application of specialized metrics designed for hierarchical graphs.

The results of these experiments unveiled the efficacy of a specific condition for constraining the optimization target. Furthermore, this condition allows us to emphasize desired metanodes. This finding is one of our primary contributions. This implies that our approach extends the expressiveness of graph layouts generated by existing algorithms.

2 Related work

Our approach mainly consists of three steps: (1) hierarchical graph layout generating, (2) layouts evaluating, and (3) layouts being optimized based on the evaluation. We discuss related work at each step.

2.1 Layout of large graphs

The readability of graph layouts is strongly affected by the positions of nodes, particularly when dealing with graphs with a large number of nodes or edges. After a lot of papers addressing graph layout methods have been published, a majority of the papers used small graphs with less than one hundred nodes for their experiments (Yoghourdjian et al. 2018). Humans may have difficulties recognizing an entire graph when dealing with large-scale graphs. One approach to overcome these limitations is clustering nodes and representing a graph as a hierarchical structure. In this paper, a graph containing 1538 nodes and 8040 edges is used as an example. However, the scalability of our approach depends on the number of node clusters rather than the total number of nodes. Given that clustering algorithms allow for adjustment of the number of clusters, our approach demonstrates high scalability.

There have been several methods to compute layouts of hierarchical graphs. For instance, Koala algorithm (Itoh and Klein 2015), a method to form clusters based on the feature vectors and connectivity of

nodes, and then nodes belonging to the same cluster (called metanode) are placed close to each other. In the algorithm, the initial positions of metanodes are first given, and then, these positions are updated by applying force-directed layout and Laplacian smoothing. Here, output layouts are unpredictable when initial positions of metanodes are randomly given, and therefore, it is often difficult to obtain desired layouts. Suh et al. (2019) proposed a method to control the layouts by adding specific forces to the Fruchterman–Reingold force-directed layout (Fruchterman and Reingold 1991). Their method provides an interactive mechanism for selecting layouts that focus on features of the graph. However, users may miss important features, leading to worse final layouts. To overcome such problems, this paper proposes a method to optimize layouts generated by an existing algorithm and easily obtain desired graph layouts.

2.2 Evaluation for graph layouts

Evaluation for graph layouts is an important issue and has been actively discussed in recent years. Layouts strongly affect the readability of the graph (Bennett et al. 2007). There have been two main perspectives of metrics for the evaluation of graph layouts: the former focuses on the structure of graphs, while the latter centers on the overall layout. Typical examples for the former metrics have been presented by Purchase et al. (1996); Purchase (1997, 2002), and Ware et al. (2002). There have been not only single-purpose metrics but also metrics that consist of multiple metrics (Huang et al. 2016). Several metrics evaluate the readability directly, including visualization coverage (Dunne et al. 2015). However, most studies deal with small graphs.

The latter metrics include, for example, the symmetry of a graph (Biedl et al. 1998), the aspect ratio of the drawing canvas (Battista et al. 1998), the distribution of nodes within the area the graph occupies, and the distribution of graph elements in a drawing space (Taylor and Rodgers 2005). In the case of large graphs, some metrics categorized the latter tend to scale easily, like shape-based metrics (Eades et al. 2017).

It is difficult to define a single best metric for graph layouts because preferable metrics differ depending on the task or the type of graph layouts. In the case of tasks searching the shortest path, the number or angle of edge crossings has the most significant impact on cognitive load and visualization efficiency (Huang and Huang 2010). In the case of edge bundling (Nguyen et al. 2013, 2017), mean edge length difference and edge density distribution have impacts (Saga 2018). Symmetries, the distance among clusters, reduces the length between elements or improves compactness making a significant difference when drawing orthogonal layouts (Kieffer et al. 2015).

These metrics are basically for non-hierarchical graphs and mainly for small graphs; in other words, there are few metrics for evaluating graph layouts specific to hierarchical large graphs. To address this issue, Liu et al. (2020) proposed Sprawlter, the specific metrics for hierarchical graph layouts. Sprawlter consists of two metrics: Sprawl, which evaluates space waste, and Clutter, which evaluates cluttering among nodes and edges. The strength of Sprawlter lies in its ability to handle metanodes. Existing metrics designed for non-hierarchical graphs do not adequately address the impact of metanodes. As mentioned above, it is preferable to search for a set of solutions that optimize multiple fitness functions. Therefore, we adopt the Sprawl and Clutter metrics as individual fitness functions.

2.3 Graph layouts as an optimization problem

There have been various evaluation metrics for graph layouts. No metric is universally considered the most important, while these metrics often have trade-off relationships. In addition, there is no singular optimal layout for a graph. In other words, there are several “preferable” layouts of a certain quality level among the countless layouts generated by adjusting parameters and initial configurations. Therefore, graph layout problems can be interpreted as multi-objective optimization problems within a high-dimensional solution space.

Genetic algorithms (GA) (Fonseca and Fleming 1993; Goldberg 1989) are applied to generate various graph layouts, including undirected graphs (Eloranta and Mäkinen 2001; Wu et al. 2019; Zhang et al. 2005; Barreto and Barbosa 2000), direct graphs (Groves et al. 1990; Utech et al. 1998), digraphs (Laguna et al. 1997), hierarchical dynamic digraphs (Pinaud et al. 2004), orthogonal graphs (Neta et al. 2012) and bipartite graphs (Khan et al. 2011). However, the above studies do not deal with hierarchical undirected graph layouts with straight edges. Our study presented in this paper applies GA with large hierarchical graphs.

These approaches used small graphs in the paper. On the other hand, Ferreira et al. (2018) adopted GA to address edge bundling problems of larger graphs. Although the problem slightly differs from node layouts,

they demonstrated two key findings: First, GA proves effective for larger graphs when proper fitness functions are defined. Second, the methods using GA can generate various layouts depending on its parameters.

Among these studies, genetic operations and fitness functions differ from each other. The method to encode graph layouts to genes for GA is essential to solving graph layout problems with GA. TimGA (Eloranta and Mäkinen 2001) draws graphs in the $N \times N$ matrix. Each node is located in a square of the matrix, and all edges are drawn as straight lines. In this representation, the size of the matrix becomes extremely large while dealing with a large graph. In the other studies (Wu et al. 2019; Zhang et al. 2005; Barreto and Barbosa 2000), the coordinates of each node are represented in a one-dimensional real number vector. This method is simpler than the use of the matrix. However, the length of the vector becomes long if a graph is large-scale. In most cases, the computational cost of fitness functions depends on the number of nodes and edges, which correlates with the length of vectors. As a result, the computational cost of the overall optimization process increases when handling long vectors during iterations. In our approach, the center coordinates of each metanodes, not each individual node, are assigned to a one-dimensional real number vector. Therefore, even graphs with many nodes, the length of the vectors becomes shorter. This allows us to represent information in a large graph using a simpler expression.

One more key factor of GA is fitness functions. As mentioned in the last section, there are multiple metrics for evaluating graph layouts. In most studies, a fitness function is expressed as one single formula combining several metrics. For example, Zhang et al. (2005) combine seven esthetics criteria, including the distance of nodes, the number of edge crossings, the existence of symmetric features, and so on. Barreto and Barbosa (2000) used the function composed by a weighted sum of the several criteria and energy functions. However, they mentioned that optimizing through a single function aims to find "the best" ranking solution within the population and is not suitable for graph layout problems. Graph layout problems are one of multi-objective problems with several metrics. The method to explore the Pareto set of non-dominated solutions is more efficient in a high-dimensional solution space, rather than using a single function combining these metrics (Gunantara 2018). In this paper, we consider the graph layout problem as a multi-objective optimization task to acquire optimal layouts. As a result, the entire population evolves to approximate the Pareto set of non-dominated solutions, leading to the acquisition of the overall optimal layouts. Consequently, these layouts are optimized entirely and exhibit various features reflecting different metrics.

Several algorithms for multi-objective optimization problems have been studied (Zitzler and Thiele 1998). Each method has both advantages and challenges (Tian et al. 2021). It is crucial to carefully choose the most suitable algorithm based on the characteristics of the problems. Among various algorithms, NSGA-II (non-dominated sorting genetic algorithm) (Deb et al. 2002) and SPEA2 (strength Pareto evolutionary algorithm) (Zitzler et al. 2001) are widely utilized in many optimization problems (Hiroyasu et al. 2002). NSGA-II (Deb et al. 2002) significantly improves the computational complexity of NSGA (Srinivas and Deb 1994). It introduces elite selection for selecting the next generation. This method excels PAES (Pareto Archived Evolution Strategy) (Knowles and Corne 1999) and SPEA (Strength Pareto Evolutionary Algorithm) (Zitzler 1999) in finding diverse solutions and converging near the Pareto-Optimal set. Zitzler et al. (2001) have shown that NSGA-II and SPEA2, an improved version of SPEA, perform best across various test problems. This study handles only two fitness functions, so NSGA-II will suffice. Although Multi-Objective Evolutionary Algorithms like NSGA-III (Deb and Jain 2013) and SPEA2 (Zitzler et al. 2001) find better solutions in higher-dimensional objective spaces, they are not necessary in our study. Therefore, we chose NSGA-II for our optimization process.

In short, it is an important issue to obtain desired layouts of large graphs with high readability. Hierarchical graph visualization is one of the approaches to draw large graphs. Here, it is challenging to achieve better layouts of hierarchical graphs flexibly. Therefore, our study presented in this paper optimizes layouts of hierarchical graphs computed by an existing algorithm by applying NSGA-II.

3 Graph drawing by a hierarchical graph layout algorithm

This section supposes that hierarchical graph layouts have two layers. We describe a graph as $G = (V, E)$, where V is the set of nodes, and E is the set of edges and a subset of $V \times V$. Meanwhile, there are metanodes M in a hierarchical graph. A metanode is a cluster of nodes, where each metanode $m \in M$ is the subset of nodes V. Each node in V belongs to exactly one metanode in M ($V = \bigcup_{m \in M} m$).

We generate hierarchical graph layouts by applying Koala algorithm proposed by Itoh and Klein (2015). The algorithm places metanodes based on the pre-calculated ideal distances between them. Here, random values can be applied as initial positions of metanodes. In this study, multiple different hierarchical graph layouts are generated by performing Koala algorithm multiple times.

3.1 Node clustering

Our implementation performs node clustering and displays the graph as a set of metanodes to simplify the appearance of visualization results. Suppose each node has feature vectors. Initially, the algorithm performs clustering based on the similarity of feature vectors and the commonality of connected nodes. We applied the clustering algorithm described in Itoh and Klein (2015) that performs an agglomerative-clustering algorithm with the furthest-neighbor method. The algorithm starts generating clusters consisting of one node and repeats the merge of clusters until the minimum distance between two clusters exceeds the user-defined threshold calculated from one of the parameters, ClustersizeRatio. When ClustersizeRatio is small, clusters are easily formed, resulting in small numbers of nodes within each cluster (metanode) and a high overall count of metanodes in the graph.

3.2 Graph layout

A metanode is a cluster of nodes, consisting of one or more nodes. When using Koala algorithm, the nodes within a metanode are arranged in a circular formation, particularly when the metanode consists of two or more nodes. For example, in Fig. 1, twenty light blue nodes form one metanode, while one light blue node and several blue nodes form another metanode nearby. Additionally, each gray node also represents a metanode consisting of a single node.

In the original Koala algorithm, the initial positions of metanodes are first given, and then these positions are improved by applying force-directed layout and Laplacian smoothing methods. Then, nodes in a metanode are swapped with each other to reduce the sum of edge lengths. Here, by assigning random values as initial positions of metanodes, different layouts are obtained by performing the algorithm multiple times. In other words, an identical layout is obtained if the same initial positions are given.

4 Optimization of hierarchical graph layouts

We adopt a genetic algorithm (GA) (Fonseca and Fleming 1993; Goldberg 1989) to optimize the initial positions of metanodes. In this approach, we encode the center coordinates of metanodes as chromosomes in the GA and optimize their positions accordingly. The overall workflow is illustrated in Fig. 2. The steps outlined in the blue area are iterated until the termination conditions are satisfied. In this paper, "Termination Condition" refers to the number of iterations, as explained in Sect. 5.3. Further details on "Genetic Operations" are provided in the following sections.

4.1 Encoding

This study supposes that the graph G is a hierarchical graph, especially two layers. A graph layout needs to be represented as a set of genes so that we can solve graph drawing problems with GA. Suppose a graph G is composed of n metanodes ($m \in M$), and the center coordinates of them are defined as $\{(x_i, y_i)gt|gt1 \leq i \leq n, m_i \in M\}$. Suppose a metanode m_i is composed by the set of nodes $\{v_j gt|gt0 \leq j < size(m_i)\}$, where (x_j, y_j) denotes the coordinate of node v_j and $size(m_i)$ means the number of the nodes. The x-coordinate of the center of the metanode m_i is defined as $x_i = (x_{max} - x_{min})/2$, with $x_{max} = max(\{x_j gt|gt0 \leq j < size(m_i)\})$ and $x_{min} = min(\{x_j gt|gt0 \leq j < size(m_i)\})$. Similarly, the y-coordinate y_i is determined using the same process. Finally, a gene of GA is defined as a real number vector $[x_1, y_1, ..., x_i, y_i, ..., x_n, y_n]$, with a length of $2n$. The encoded coordinates represent the positions of the metanodes in Koala algorithm and define a particular graph layout.

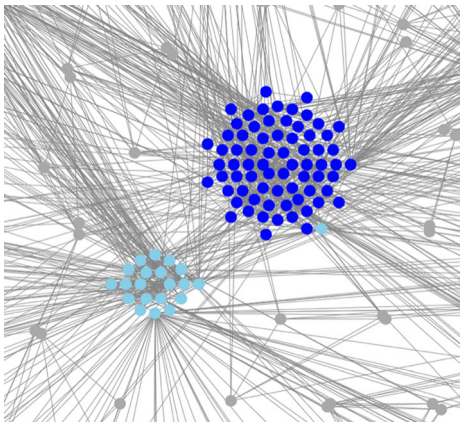


Fig. 1 Part of a Graph Layout generated by Koala algorithm. Twenty light blue nodes form one metanode, while one light blue node and several blue nodes form another metanode nearby. Each gray node represents a metanode consisting of a single node as well

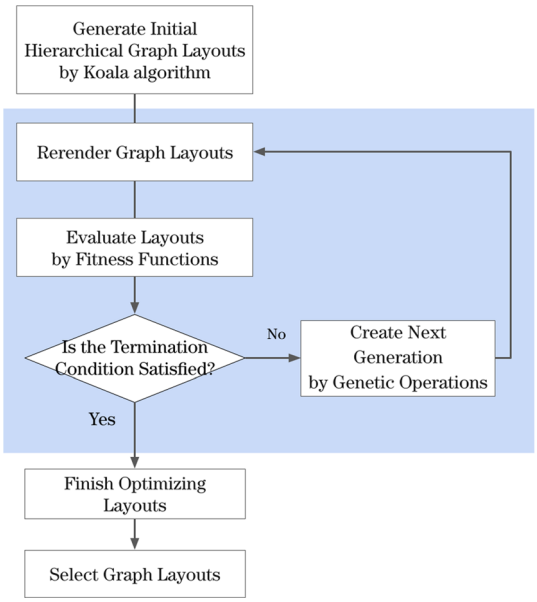


Fig. 2 The workflow chart of the introduced algorithm. Details of the 'Genetic Operations' are provided in Sect. 4.4. 'Termination Condition' is elaborated on in Sect. 5.3

4.2 Fitness functions

The quality of each individual layout is evaluated by fitness functions. We basically apply Sprawl and Clutter metrics (Liu et al. 2020) as fitness functions. Here, the smaller the penalty is, the better the individual layout is. Since these metrics are specific for hierarchical graph layout, GA can properly optimize individuals corresponding to the metanodes of a hierarchical graph.

4.2.1 Sprawl metric

The Sprawl metric evaluates space waste. The larger the Sprawl penalty is, the larger space on a canvas is wasted.

Sprawl metric is represented by the following formula (1):

$$S(G) = \frac{area(G)}{\sum_{v \in V} area(v)} \tag{1}$$

Here, $S(G)$ represents the sprawl, and $area(G)$ is the total drawing area of the graph layout G, including all the empty space between the nodes. $\sum_{v \in V} area(v)$ is the sum of areas of all individual node shapes.

4.2.2 Clutter metric

The Clutter metric captures cluttering among nodes and edges. The larger the Clutter penalty is, the more overlaps of nodes and the more crossings of edges the layout has.

The definition of Clutter is provided by the following formula (2):

$$Clutter = \alpha\, NNpen + \beta\, NEpen + \gamma\, EEpen \tag{2}$$

Here, *NNpen* denotes Node–Node penalty, *NEpen* denotes Node–Edge penalty, and *EEpen* denotes Edge–Edge penalty. In the study by Liu et al. (2020), the weights of the three penalties are adjusted by normalizing them and modifying parameters α, β, and γ. Layouts are then evaluated relatively within specific groups in the approach. In our implementation, layouts are evaluated in relation to the set of layouts from all current and previous generations.

Here, g_i represents the set of genes in the i-th generation ($0 \leq i$), where g_0 denotes the initial generation. g_{ik} denotes the k-th gene in g_i ($0 \leq k$). $normedNN_{ki}$ denotes Node–Node penalty of k-th gene in the i-th generation ($0 \leq i$).

$$maxNN_i = max(\{calcNN(g) \mid g \in \cup_{0 \leq j \leq i} g_j\}) \tag{3}$$

$$minNN_i = min(\{calcNN(g) \mid g \in \cup_{0 \leq j \leq i} g_j\}) \tag{4}$$

$$normedNN_{ik} = \frac{g_{ik} - maxNN_i}{maxNN_i - minNN_i} \; (g_{ik} \in g_i) \tag{5}$$

Other two penalties, Node–Edge penalty and Edge-Edge penalty, are also normalized by this process. Finally, the Clutter penalty of the k-th gene in the i-th generation is calculated by Eq. 6.

$$Clutter_{ik} = \alpha\, normedNN_{ik} + \beta\, normedNE_{ik} + \gamma\, normedEE_{ik} \tag{6}$$

4.3 Initialization

Our implementation, first cluster nodes depend on their connectivity and feature vectors by applying the clustering algorithm by Itoh and Klein (2015). Individuals in the first generation are randomly generated as real number vectors. The layout algorithm of Koala (Itoh and Klein 2015) can generate a hierarchical graph layout with the clustering result and a set of individuals.

4.4 Genetic operations

It is essential to perform genetic operations to improve the quality of the individuals. Our implementation of the genetic operation is mainly based on NSGA-II (Deb et al. 2002), described as the following pseudo-code.

```
Initialize
for generation ≦ 20 do
    Crossover generate IndividualsC
    Mutation generate IndividualsM
    NewIndividuals ⇐ IndividualsC + IndividualsM
    Evaluate NewIndividuals
    NextIndividuals ⇐ Selection(Individuals + NewIndividuals)
end for
```

Algorithm 1 Optimization layout

Each of the genetic operations in our implementation is introduced in the following sections.

4.4.1 Crossover

We introduce a simulated binary crossover to generate children individuals. We can choose how much children resemble their parents by this operation. The crossover operation of NSGA-II is adopted in this study.

4.4.2 Mutation

We introduce polynomial mutation to create individuals that differ from their parent individuals to avoid getting trapped in local optima. The mutation operation of NSGA-II is adopted.

4.4.3 Selection

A distinctive algorithm within the genetic operations of NSGA-II is applied to select the individuals passed to the next generation. This operation is mainly based on elite selection with the ranking algorithm.

NSGA-II excels at selecting individuals with lower penalty while maintaining diversity among solutions. This feature is particularly advantageous in avoiding getting trapped in local optima. Graph layout problems, in particular, do not have a single clear solution but offer numerous better solutions. Therefore, maintaining high diversity among solutions is well-suited for our specific problem.

4.5 Focusing on specific metanodes

We consider focusing on metanodes that satisfy specific conditions. Large graphs with more than a hundred nodes exceed the limitations of human cognition (Yoghourdjian et al. 2018). Large graphs with more than a hundred nodes surpass the limits of human cognitive capacity, making it nearly impossible to grasp the positions of all nodes at once. In these situations, focused metanodes should be prioritized in areas with high readability. Moreover, different metanodes may be focused on depending on the dataset, the goal of visualization, or the features of layouts.

To be more specific, when we evaluate individuals, we apply the fitness function only to specific metanodes that meet certain conditions, while we do not optimize the positions of other metanodes. To implement this approach, we extend both the original Clutter and Sprawl metrics.

4.5.1 Sprawl metric for focused metanodes

Sprawl metric is computed by the formula (1), where we extend the method to calculate $area(G)$ and $\sum_{v \in V} area(v)$. Only the focused metanodes are recognized as targets of a graph layout in this process, while the other metanodes are skipped. An example is illustrated in Fig. 3, where $area(G)$ is the area of the blue rectangle and $\sum_{v \in V} area(v)$ is determined from the sum of the areas of the nine red nodes in this figure.

4.5.2 Clutter metric for focused metanodes

Clutter consists of three penalties, Node–Node penalty, Node–Edge penalty, and Edge–Edge penalty in the following formula (2). When the focus condition is set, the formula (7) is applied.

$$Clutter = \alpha NNpen + \beta NEpen \tag{7}$$

We extended the calculation of Node–Node and Node–Edge penalties as follows. Suppose that red metanodes satisfy the focus condition as shown in Fig. 4. In this case, only red metanodes are targets to calculate the Node–Node penalty, which means the overlap between two gray metanodes is ignored. Similarly, only blue edges which are pass-through focused metanodes are targets to calculate the Node–Edge penalty. In Formula (7), the Edge-Edge penalty is excluded because it concerns edges connected to individual nodes rather than metanodes.

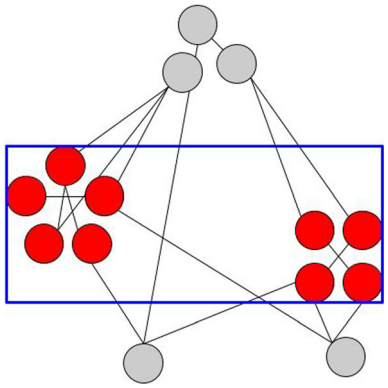


Fig. 3 An example of a hierarchical graph layout. Red nodes represent focused nodes, and gray nodes represent not-focused nodes

5 Preparations for experiments

5.1 Dataset

We visualize a paper co-authorship dataset where the papers were published at the NERC Biomolecular Analysis Facility (NBAF) from 1998 to 2013. The dataset consists of 1,821 author nodes and 11,097 co-authorship edges, where each node has a 12-dimensional feature vector. Each feature vector represents a term that is frequently used in all the paper titles and strongly related to all paper authors. Itoh and Klein (2015) described how to organize the dataset in detail.

We extracted the largest connected subgraph from our dataset, ensuring that all nodes are interconnected. Any disconnected smaller subgraphs were removed, resulting in a graph where all nodes have one or more edges. By the process, the final size of the dataset became smaller as shown in Table 1.

5.2 Graph layouts

We generated graph layouts from the processed dataset using Koala algorithm (Itoh and Klein 2015) with varying values of the ClustersizeRatio parameter. The features of these layouts are summarized in Table 2. When the ClustersizeRatio is higher, each metanode tends to include a larger number of nodes. Consequently, this leads to fewer metanodes overall in the graph.

5.3 Specification of parameters by preliminary experiments

We performed preliminary experiments with the aforementioned dataset. It is necessary to specify both the proper population size, the number of individuals in a generation, and the termination condition depending on the dataset. To find proper values, we performed two preliminary experiments with the conditions shown in Table 3. The crossover probability is set to 0.90, and the mutation probability is set to 0.10. The specifications of the computer used in these experiments are as follows: MacBook Air with a 1.6GHz dual-core Intel processor, 8GB of memory, and running macOS Ventura 13.2.1.

Here, we used Hypervolume (Knowles et al. 2006) to evaluate the progress of optimization. Hypervolume calculates an area enclosed by the Pareto solution set and an arbitrary reference point, as illustrated in Fig. 5. A larger Hypervolume value indicates a wider range covered by the Pareto solution set, which means a better set of solutions. In our experiments, Clutter and Sprawl metrics are used as fitness functions. We set the maximum Clutter and Sprawl values in the initial generation as the reference point.

Figure 6 shows the transition of Hypervolume of each experiment. Due to variations in the initial layouts generated by Koala algorithm across experiments, the reference point for calculating Hypervolume differs. As a result, the y-axis of Fig. 6 illustrates the relative increase in Hypervolume. Pre-Experiment 1 required 324 min to complete 100 generations of iterations, whereas Pre-Experiment 2 completed the same number of iterations in 173 min.

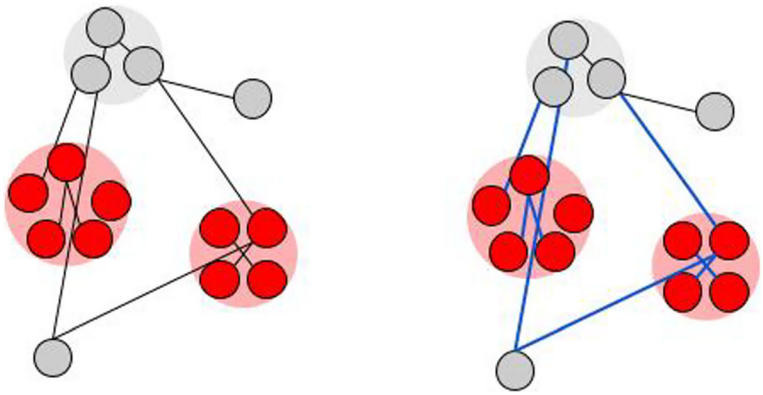


Fig. 4 (Left) An example of a hierarchical graph layout. Red nodes represent focused nodes, and gray nodes represent not-focused nodes. (Right) Highlight edges under consideration for computation

Table 1 Results of data preprocessing

	Before preprocessing	After preprocessing
Nodes	1821	1538
Edges	9692	8040

Table 2 Graph layouts drawn by Koala algorithm

	ClustersizeRatio= 0.5	ClustersizeRatio= 0.7
Nodes	1538	1538
Edges	8040	8040
Metanodes	769	461

Table 3 Conditions of preliminary experiments

	Pre-experiment 1	Pre-experiment 2
The population size	40	20
ClustersizeRatio	0.7	0.7
Iterations	100	100

We first discuss the population size. During the initial 10 generations of optimization, in Fig. 6, the pace of improvement of Hypervolume was rapid. However, at a specific point in the process, the optimization rate transitioned to a more moderate pace.

We focused on the difference in Hypervolume between the initial and final generation to discuss a range of improvements. Compared to Pre-Experiment 1 (the red line), Pre-Experiment 2 (the blue line) has greater improvement in the same iterations. This means population size affects the speed to optimize the whole population. Furthermore, a shorter computation time is required when the population size is small. Based on this result, we have specified a population size of 20 for the main experiments described in Sect. 6.

Secondly, we discuss the termination condition of this GA algorithm. Upon analyzing the Hypervolume transition in Pre-Experiment 2 (depicted by the blue line in Fig. 6), convergence is observed around the 20th generation. Increasing the iterations to 40 results in a doubling of computational time. However, we find that 20 iterations are sufficient for convergence. Therefore, based on these results, we have set the termination condition of the genetic algorithm with the dataset to 20 iterations.

6 Experiments

Experiments for optimizing graph layouts are conducted with two types of hierarchical graphs and three conditions. The types of hierarchies are determined by ClustersizeRatio mentioned in Sect. 3.1. The conditions determine metanodes which are focused in the process of optimization. Only metanodes that satisfy

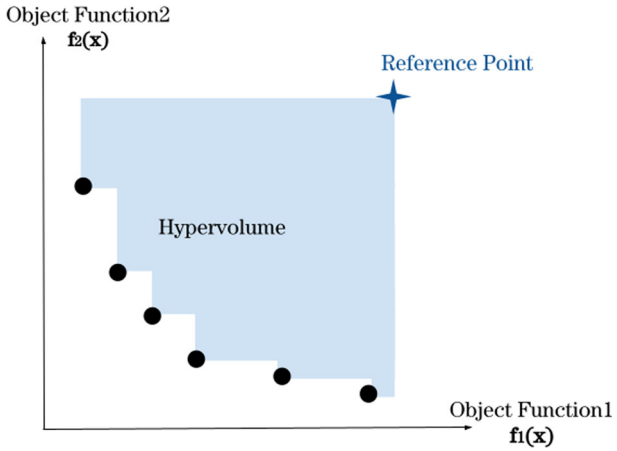


Fig. 5 The area of the blue region in this figure represents Hypervolume

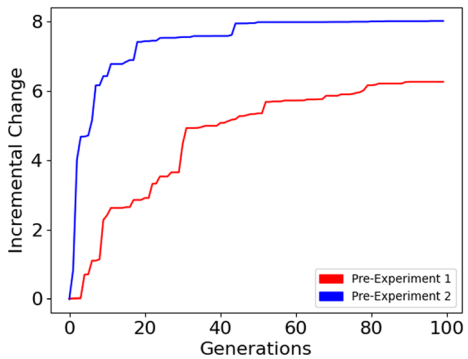


Fig. 6 The transition of Hypervolume across experiments. The y-axis depicts the relative increase from the initial generation

the specific condition are optimized when the condition is set. The fitness function follows the metrics mentioned in Sect. 4.5 in the experiments. The crossover probability is set to 0.90, and the mutation probability is set to 0.10.

Table 4 shows all combinations of types and conditions in the experiments. The following sections describe the results and details of the experiments.

The evaluation results of individuals in a generation are represented in a scatterplot in the following sections. The horizontal axis represents the Sprawl penalty, while the vertical axis represents the Clutter penalty. Each point represents a single individual corresponding to a single hierarchical graph layout. For the Sprawl penalty (the x-axis), the penalty of an individual in the initial generation and the final generation are comparable. On the other hand, it is meaningless to compare them for the Clutter penalty (the y-axis), due to the normalization method, where the maximum and minimum values used in the Clutter metric are slightly different in generations. Annotations near dots in a scatterplot are sequential numbers. The coincidence of labels in different scatterplots does not imply the same graph layout. Genetic operations may generate entirely new layouts, rather than adjusting an original graph layout.

6.1 Condition 1: optimization of entire graph layouts

In the following two experiments 1-S and 1-L, the positions of all metanodes are the targets of optimization. Consequently, all metanodes and edges are evaluated by the fitness functions.

6.1.1 Experiment 1-S

Figure 7 shows the distribution of evaluations for individuals in both the initial and final generations. There are two scatterplots, each corresponding to the initial and final generations.

Table 4 Types of experiments

Focus condition	ClustersizeRatio	
	0.5	0.7
Condition 1 (no additional information)	Experiment 1-Small (1-S)	Experiment 1-Large (1-L)
Condition 2 (with feature vector)	Experiment 2-Small (2-S)	Experiment 2-Large (2-L)
Condition 3 (with node degree)	Experiment 3-Small (3-S)	Experiment 3-Large (1-L)

The individuals in the final generation shown in Fig. 7b spread moderately across the solution space. Several individuals achieve low Clutter penalties and high Sprawl penalties, while others exhibit the opposite pattern. Other individuals achieve both moderate Clutter penalties and moderate Sprawl penalties. In any case, individuals obtain overall low penalties in the final generation. Each dot in the scatterplot represents a graph layout. For example, the graph layout corresponding to the blue dot annotated with No. 7 is visualized in Fig. 8a, while the graph layout corresponding to the red dot annotated with No. 12 is visualized in Fig. 8b.

The layout shown in Fig. 8a achieves the highest Clutter penalty. As a result, the metanodes overlap each other, which is shown in red rectangles of Fig. 9. Although the Clutter penalty of the layout No. 12 is highest among layouts in the final generation, there are few overlaps of metanodes observed. In other words, overlaps are eased thanks to the optimization process. On the other hand, the difference of Sprawl penalty is difficult to be recognized.

6.1.2 Experiment 1-L

Figure 10 shows the distribution of evaluations for individuals in both the initial and final generations. Compared to scatterplots in Fig. 7, individuals overall achieve low Clutter penalties in the final generation. This is because when the ClustersizeRatio is small, the number of metanodes decreases, and overlaps of elements occur less frequently. Therefore, the Node–Node penalty becomes much smaller, and the Node–Node penalty of some layouts is 0.0.

Layout No. 6 achieves the highest Clutter value in the initial generation as visualized in Fig. 11a. The Node–Node penalty of its layout tends to be small in the case of small ClustersizeRatio. Therefore, the Clutter penalty is influenced by the Node–Edge penalty and Edge-Edge penalty. Figure 11b shows an enlarged view, and edges connecting to the right orange node in the red circle are highlighted. The edges cross metanodes with a gray color, which is shown in the blue circles in Fig. 11b. This makes the Clutter value worse.

Figure 12 shows several layouts in the final generation. Although layout No. 12 achieves the worst Clutter value among the final generation, it appears to be no different from layout No. 0, which achieves a better Clutter value. In terms of the Sprawl values, there are no obvious differences, even if the Sprawl values are improved.

6.1.3 Brief summary of experiments 1-S and 1-L

When ClustersizeRatio is large, the difference between before and after optimization is particularly obvious, especially in terms of Clutter metrics. On the other hand, when ClustersizeRatio is small, layouts already tend to be preferable, making it difficult to recognize differences. However, the layouts after optimization in both experiments show improvements, including a reduction in the overlaps of elements.

6.2 Condition 2: metanodes with feature-vectors of nodes

In the following experiments 2-S and 2-L, the positions of only metanodes containing one or more distinctive nodes are optimized. In our implementation, a distinctive node is defined as a node with a nonzero feature vector. In graph layouts of Sect. 6.1, these distinctive nodes are represented with a color corresponding to their feature vector, and the others are represented in gray. The fitness function follows the metrics mentioned in Sect. 4.5 since the conditions are applied in these experiments.

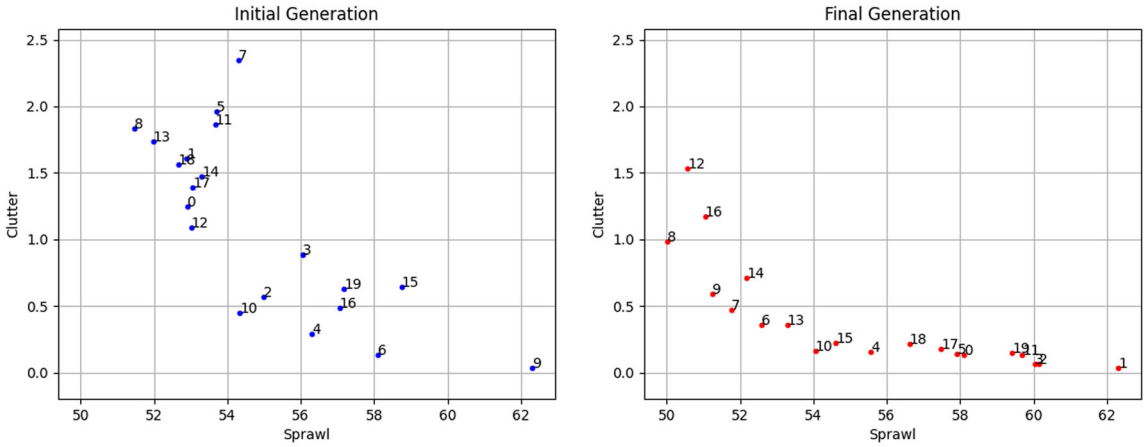


Fig. 7 Evaluation results of population. (Experiment 1-S: smaller ClustersizeRatio, none focus condition)

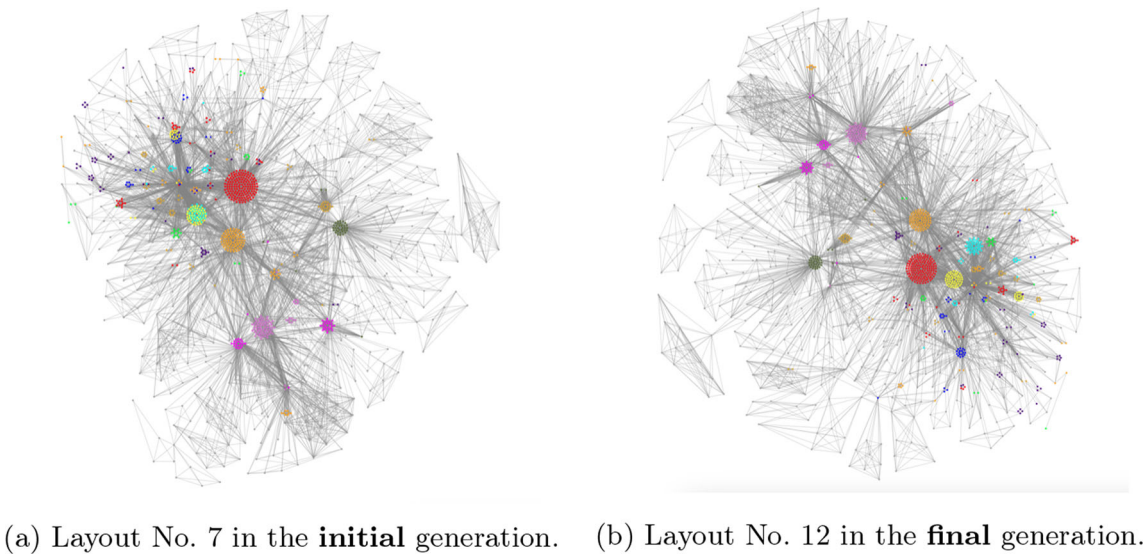


Fig. 8 Examples of layouts before optimization

6.2.1 Experiment 2-S

The number of metanodes that satisfies the condition is 23, out of a total of 461 metanodes in Experiment 2-S. Figure 13 shows the distribution of evaluations for individuals.

According to Fig. 13, layout No. 11 achieves notably the largest Clutter penalty but low Sprawl penalty. On the other hand, layout No. 10 achieves the low Clutter penalty and notably the highest Sprawl penalty. Since the positions of only colored-metanodes are optimized in the results of Experiments 2-S and 2-L, we only focus on these positions. We compared the two layouts in terms of both overlaps of elements and the waste of the drawing space (Fig. 15).

In terms of the Clutter metric, layout No. 11 has many overlaps of metanodes, which leads to making the Node–Node penalty worse. The high density of metanodes is likely responsible for an increase in edges crossing over colored-metanodes. On the other hand, there are few overlaps in layout No. 10.

In terms of the Sprawl metric, since layout No. 11 is highly evaluated, colored-metanodes are placed in a small area. This means the space is effectively used in layout No. 11. Figure 16 shows the target area of the Sprawl metric in each layout. The layouts No. 10 and 11 shown in Figs. 16a and b are layouts after optimization; the layout No. 17 shown in Fig. 16c is the layout before optimization.

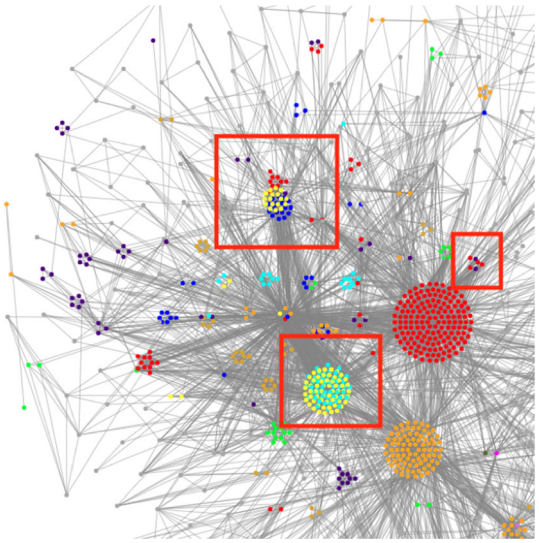


Fig. 9 An enlarged view of the layout shown in Fig. 8a

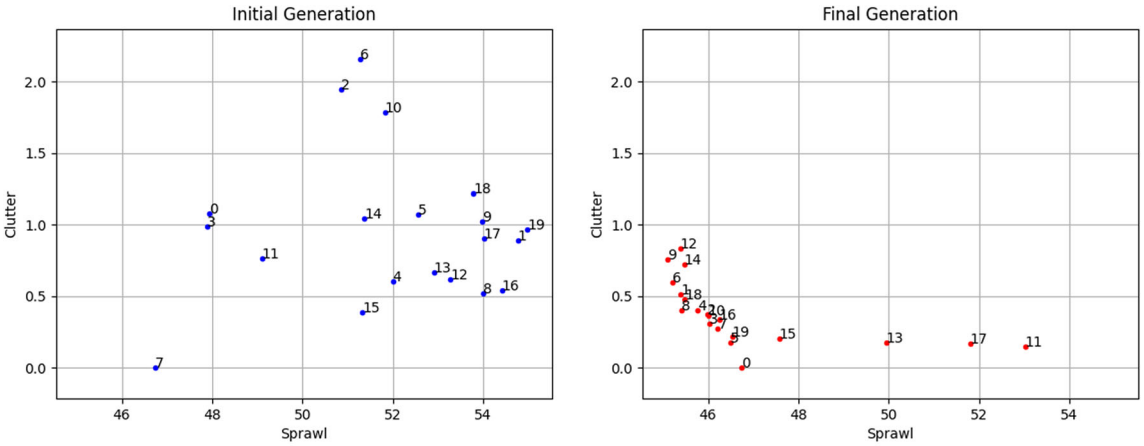


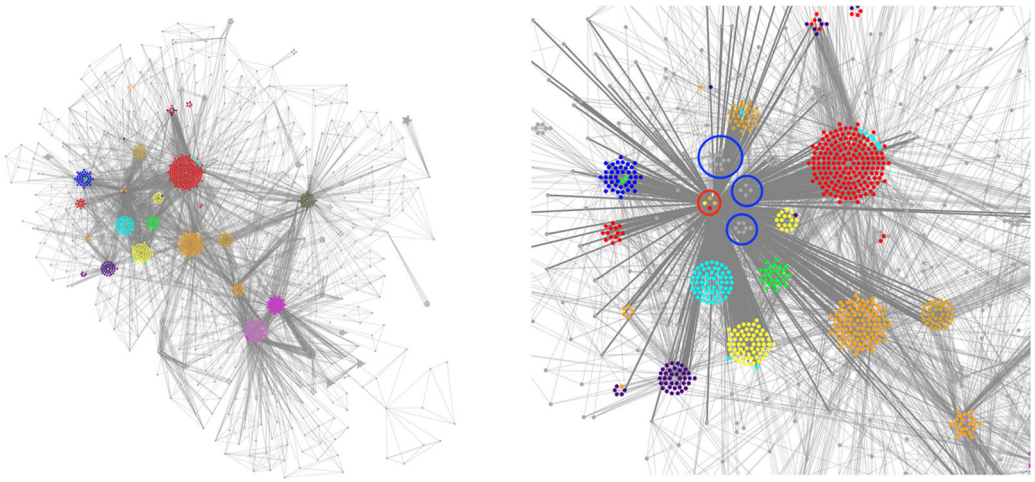
Fig. 10 Evaluation results of population. (Experiment 1-L: larger ClustersizeRatio, none focus condition)

Although the difference between layouts No. 10 and 11 looks small, their target area shown in the red rectangle is obviously smaller than Fig. 16c.

6.2.2 Experiment 2-L

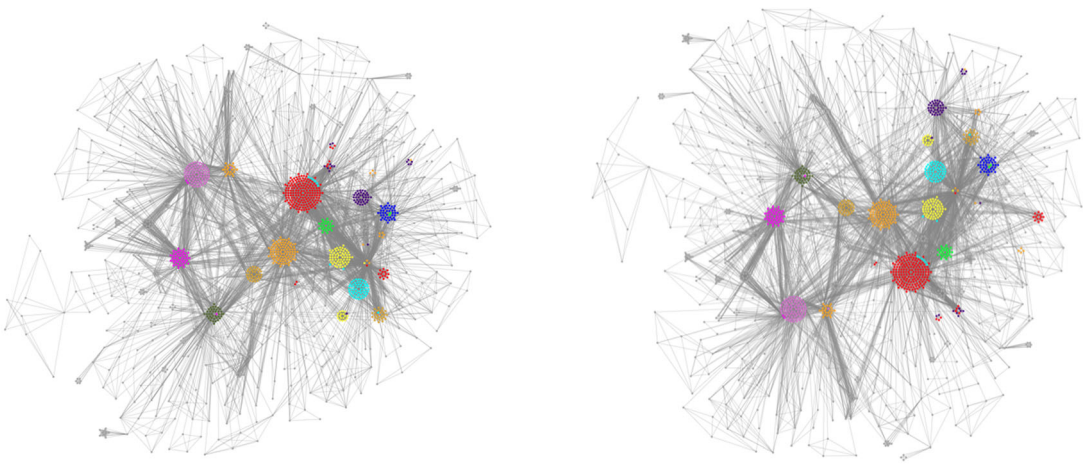
The number of metanodes that satisfy the condition is 95, out of a total of 769 metanodes in Experiment 2-L. Figure 17 shows the distribution of evaluations for individuals.

The Clutter value of layouts No. 16, 8, and 0 is 0.0. There are two potential factors contributing to this result. One is that the Node–Node penalty is 0.0, indicating no overlap of metanodes. In addition, another factor is that the penalty NE can be 0.0. The minimum Node–Edge penalty for each generation will be 0.0. Even if an individual whose Node–Edge penalty is smaller, the penalty of past individuals is not updated, so multiple individuals may have the Node–Edge penalty of 0.0. Therefore, the Node–Edge penalty of multiple individuals can be 0.0 as well in the final generation. As a result, some Clutter values in the final generation can be 0.0.



(a) Layout No. 6 in the initial generation. (b) An enlarged view of layout No. 6.

Fig. 11 The example of layouts in the initial generation



(a) Layout No. 12 in the final generation. (b) Layout No. 0 in the final generation.

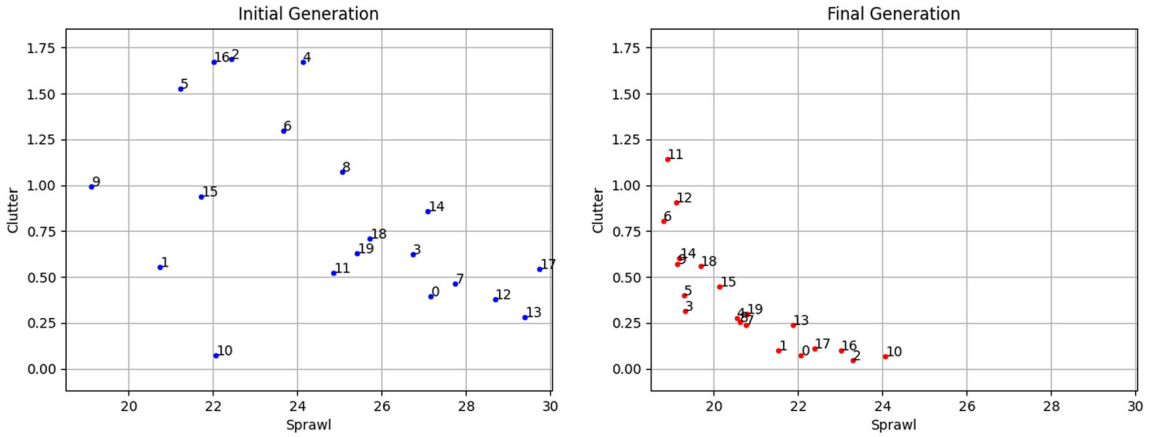
Fig. 12 The example of layouts in the final generation

6.2.3 Brief summary of experiments 2-S and 2-L

Layouts are optimized even when the target of optimization is limited. Moreover, compared to optimization of the entire graph layout, such as Experiments 1-S and 1-L, the appearance of layouts has significantly improved. In Particular, the Sprawl metric works better.

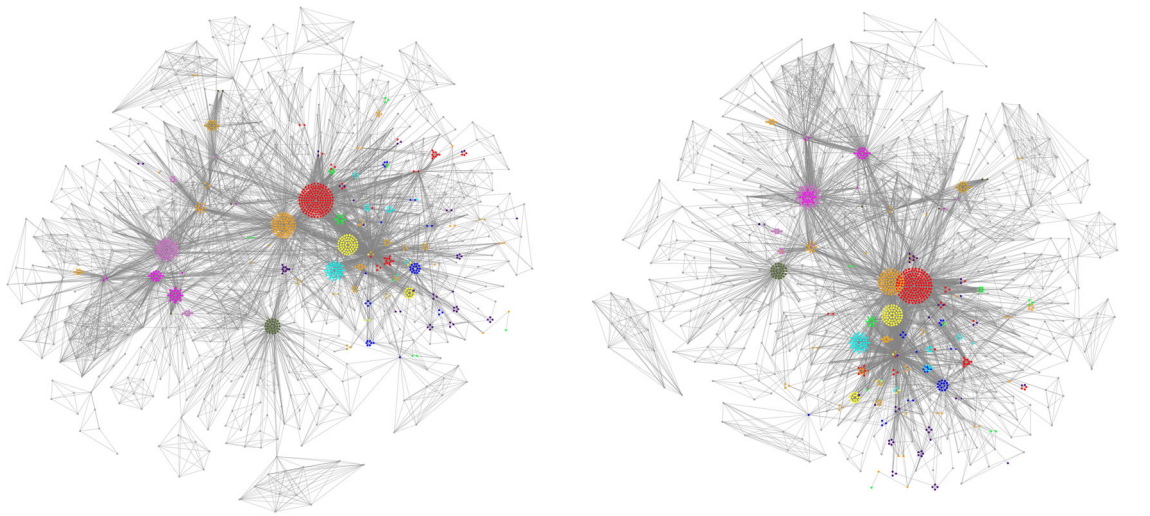
6.3 Condition 3: metanodes consisting of high-degree nodes

In the following experiments 3-S and 3-L, the positions of only metanodes containing high-degree nodes are optimized. Node degree is calculated by counting nodes connected to the node. A high-degree node is defined as a node with a degree greater than five in our implementation. Also, metanodes containing more than three high-degree nodes are optimized in our implementation. The other parameters of the fitness functions remain the same as those of Experiments 2-S and 2-L.



(a) Initial Generation (b) Final Generation

Fig. 13 Evaluation results of population. (Experiment 2-S: smaller ClustersizeRatio, focus on metanodes with feature vectors)



(a) Layout No. 10 in final generation (b) Layout No. 11 in final generation

Fig. 14 Examples of layouts in the final generation

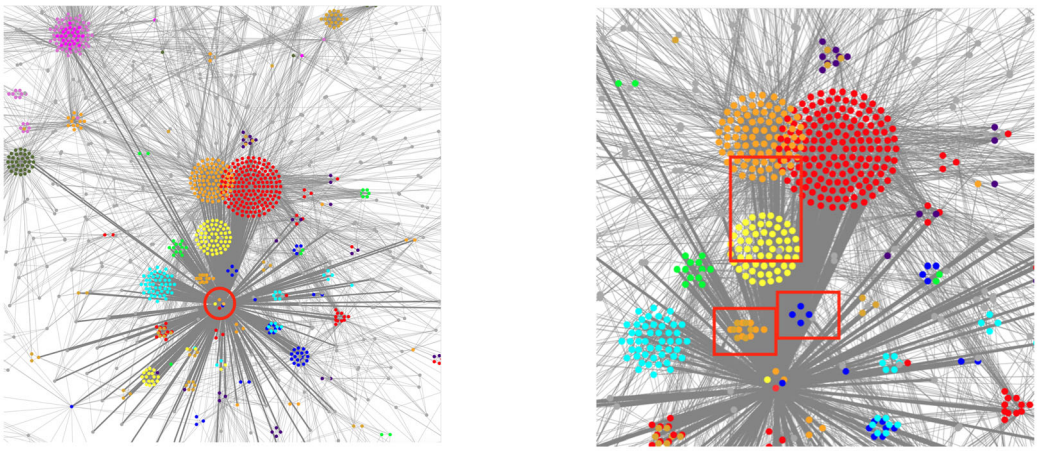
6.3.1 Experiment 3-S

The number of metanodes that satisfy the condition is 33, out of a total of 769 metanodes in Experiment 3-S. Figure 18 shows the distribution of evaluations for individuals.

First of all, Fig. 19 shows optimal graph layouts for Sprawl and Clutter. Each node in Figs. 19a and 19b is colored based on its specific feature vector. However, it is difficult to identify metanodes that satisfy the condition and observe the effect of optimization. Therefore, in Figs. 19c and 19d, nodes belonging to metanodes that satisfy the condition are colored in red, while the rest are colored in gray.

According to Fig. 20, the difference in the target area of Sprawl between layout No. 6 and 16 is noticeable. This implies that the condition related to node degree is effective in reflecting the impact of Sprawl.

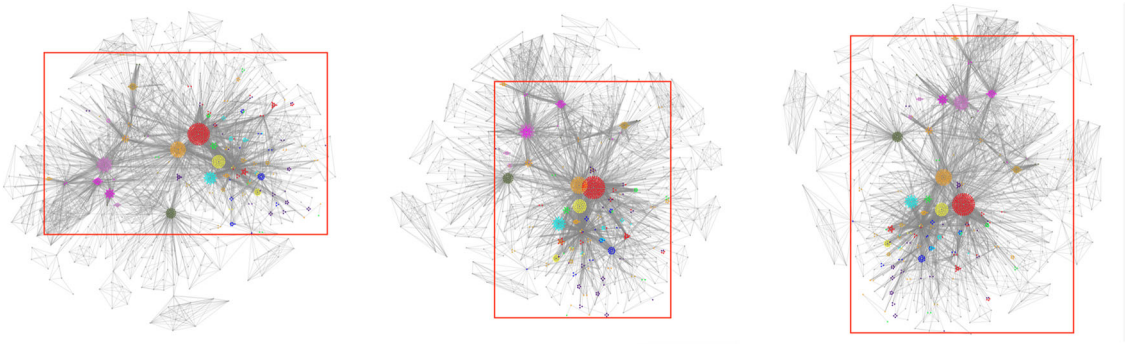
Second, Fig. 21 displays moderately favorable graph layouts from the aspects of both Clutter and Sprawl. There are no significant overlaps of red metanodes, while the metanodes are spread moderately. However, there are overlaps between several red and gray metanodes in Experiment 3-S as illustrated in Fig. 22, where only metanodes that satisfy the condition of high-degree nodes are optimized. This implies



(a) Highlighting edges which connect to the red node in red circle in the figure.

(b) Overlaps of colored metanode and edges.

Fig. 15 The reason for the high Node–Edge penalty of the layout No. 11 in final generation. (Fig. 14b)

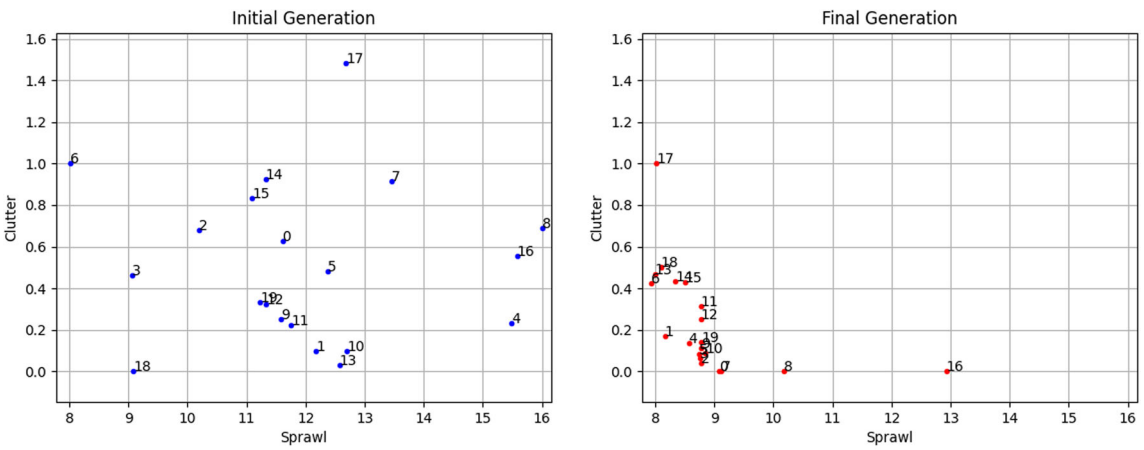


(a) Layout No. 10 in the final generation.

(b) Layout No. 11 in the final generation

(c) Layout No. 17 in the initial generation

Fig. 16 The red rectangle in each figures represents the target area of Sprawl metric



(a) Initial Generation

(b) Final Generation

Fig. 17 Evaluation results of population. (Experiment 2-L: larger ClustersizeRatio, focus on metanodes with feature vectors)

that the optimization is successfully limited to specific metanodes. The results of the optimization vary depending on the applied condition; thus, it is important to set the proper condition to obtain desired layouts.

6.3.2 Experiment 3-L

The number of metanodes that satisfy the condition is 49, out of a total of 461 metanodes in Experiment 3-L. Figure 23 shows the distribution of evaluations for individuals.

Figure 24 shows optimal graph layouts from the aspects of both Sprawl and Clutter.

6.3.3 Brief summary of experiment 3-S and 3-L

The results of these experiments show that the condition to limit the target metanodes of optimization is effective. The variation in evaluation by Clutter and Sprawl metrics more clearly depends on the layout. This finding implies that our method can extend the original Koala algorithm based on the target of visualization by setting the optimization condition.

6.4 Performance

In this section, we discuss the computational cost and response time associated with our approach.

The process of drawing the graph with Koala Algorithm has a $O(N \log N)$ computational complexity, where N is the number of metanodes (Itoh and Klein 2015). For the optimization process with NSGA-II, the computational complexity is $O(MK^2)$, where M represents the number of objectives, and K represents the population size (Deb et al. 2002). The Clutter penalty is calculated in $O(K^2) + O(KE) + O(E^2)$, and the Sprawl penalty is calculated in $O(1)$, where K is the number of metanodes, and E is the number of edges (Liu et al. 2020). When the condition to focus on the target metanodes of optimization is set, the Clutter penalty is calculated in $O(K^2) + O(KE)$. To summarize, the computational cost of our approach is $O(MN^2) + O(KlogK + K^2 + KE + EE)$.

Table 5 shows the response times for each experiment. These times represent the duration required to obtain optimized layouts after 20 iterations, which is the termination condition for each experiment. They also include the time needed to output the graph in both the first and last generations. The computer with the same specifications as those in the preliminary experiments was used for these six experiments.

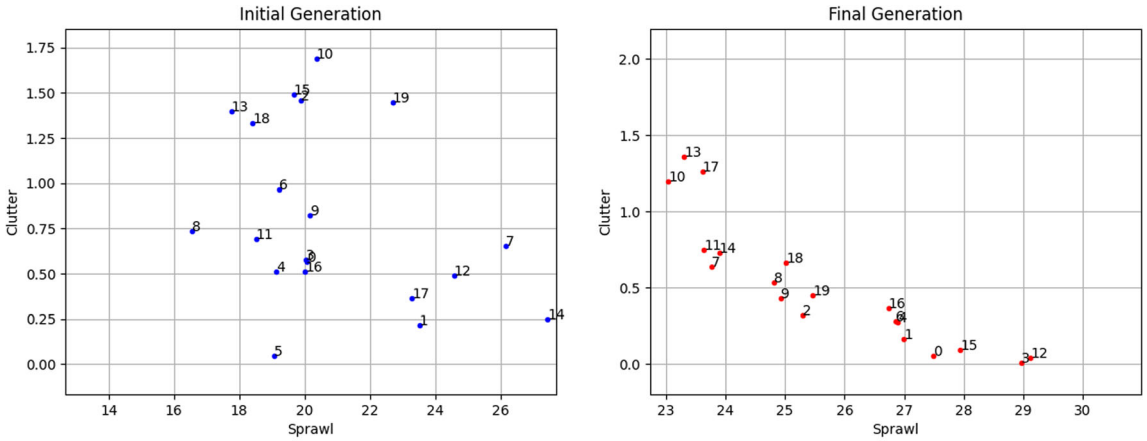
6.5 Summary of experiments

In this section, six experiments are conducted with two types of graph clustering and three conditions. In all experiments, the improvement of the evaluation of graph layouts is evident.

Especially, the difference in the Sprawl penalty becomes obvious by focusing on the specific metanodes. Each box plot in Fig. 25 shows the comparison of the Sprawl penalties of individuals before and after optimization. The median of Sprawl penalties is improved after optimization in almost all experiments. Furthermore, the range of improvement is observable, especially in the latter six experiments, where only the specific metanodes are optimized.

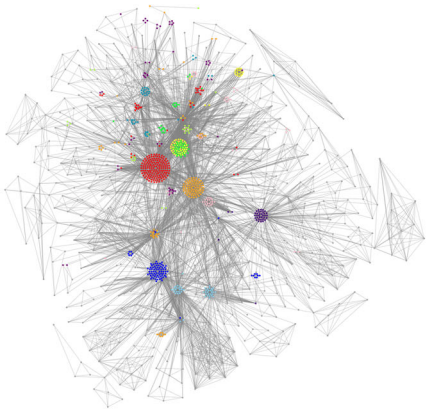
Among the box plots in Fig. 25, only in the case of Experiment 3-S, the Sprawl penalty gets worse after optimization. The likely reason for the observed result lies in the trade-off relationship between Sprawl and Clutter metrics. Figure 18b in Sect. 6.3.1 shows the distribution of evaluations for all individuals in the final generation. One of the individuals achieves the worst Sprawl penalty, but the best Clutter penalty. The feature of NSGA-II adopted in our implementation is the preservation of diversity among individuals. Therefore, we suppose that individuals are retained even if they receive a poor evaluation in one metric, as long as their evaluation in the other metric is excellent.

Hierarchical graph layouts usually tend to be complex, because they consist of a lot of nodes and edges. Although Koala algorithm can generate layouts of moderate quality, it is difficult to make specific metanodes outstanding. However, if there is a specific target for visualization, it is essential to represent features relevant to that target. Our optimization approach contributes to ensuring that features associated with the specified target become more apparent. Consequently, it is conducive to gaining valuable insights from complex and large hierarchical graph layouts.

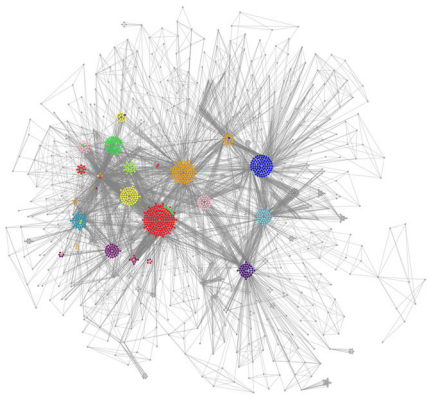


(a) Initial Generation (b) Final Generation

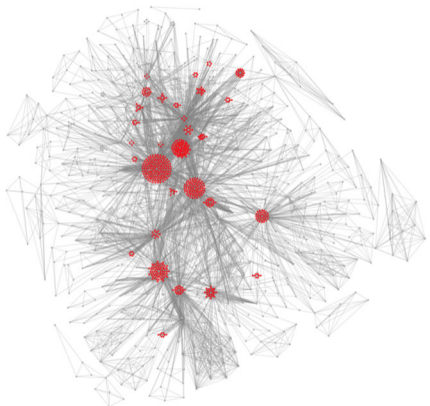
Fig. 18 Evaluation results of population. (Experiment 3-S: smaller ClustersizeRatio, focus on metanodes consisting of high-degree nodes)



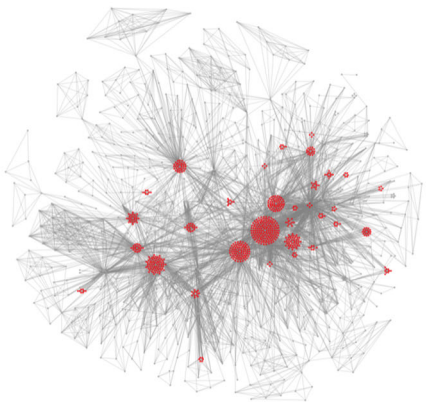
(a) Layout No. 6 with nodes colored based on feature vectors.



(b) Layout No. 16 with nodes colored based on feature vectors.

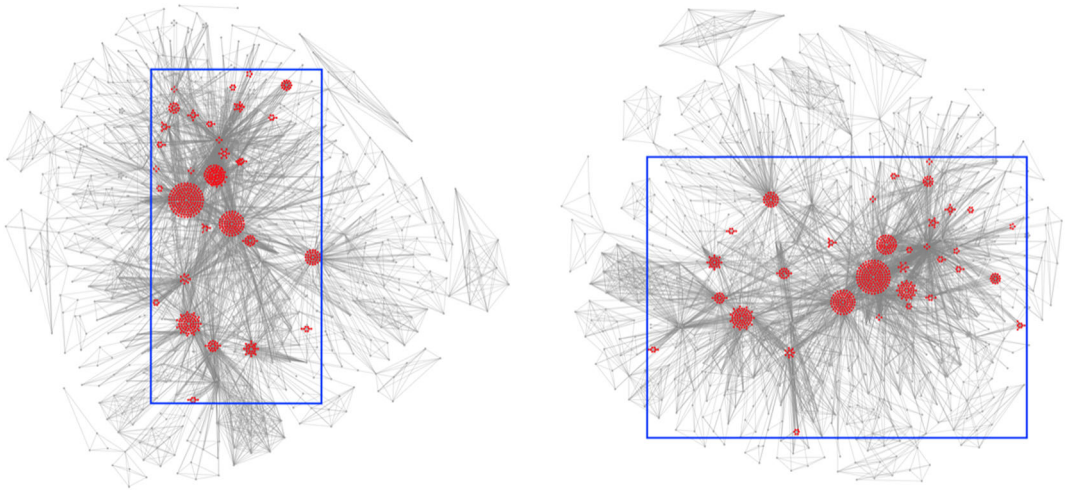


(c) Red metanodes satisfy the condition. Layout No. 6. Optimal layouts for Sprawl.



(d) Red metanodes satisfy the condition. Layout No. 16. optimal layouts for Clutter.

Fig. 19 Examples of layouts in the final generation



(a) Layout No. 6. (b) Layout No. 16.

Fig. 20 The difference of the target area for Sprawl between layout No. 6 and No. 16

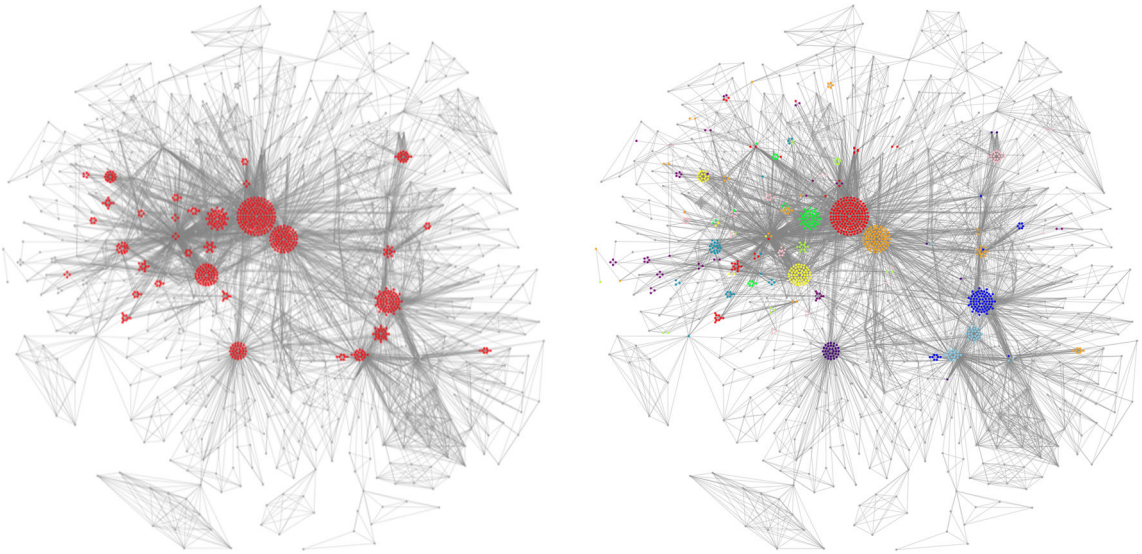


Fig. 21 Layout No. 4 in the final generation

7 Conclusions and future work

This paper presented an optimization technique for hierarchical graph layout by adjusting the position of metanodes, which enables to achieve highly evaluated graph layout constantly. Furthermore, our approach can enhance the variability of graph layouts generated by existing graph drawing algorithms by setting the condition to focus on the target metanodes of optimization. The formulation of conditions is inherently intuitive, making it easier to obtain layouts that align with the target of visualization. The paper showed the experimental results that improved hierarchical graph layouts generated by an existing algorithm by optimizing the position of its metanodes, compared to layouts generated by the original algorithm. The paper also introduced the transformed layouts depending on the given conditions.

Our method is theoretically applicable to other graph layout algorithms that consume random initial positions. Furthermore, it is not limited by the size of the graph. It means that this method is applicable to large graphs. The computational cost of our approach depends on the number of metanodes and edges. In other words, reducing the number of metanodes during clustering can help control the computational cost,

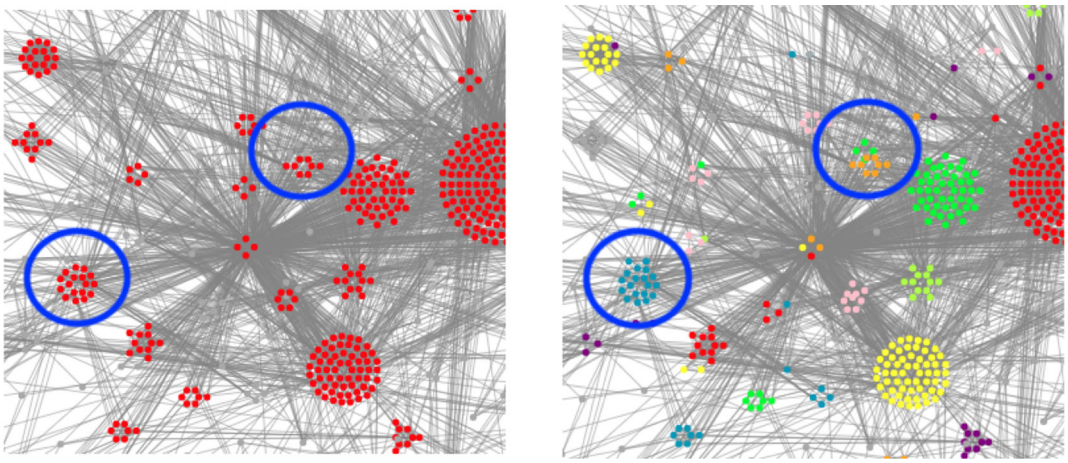
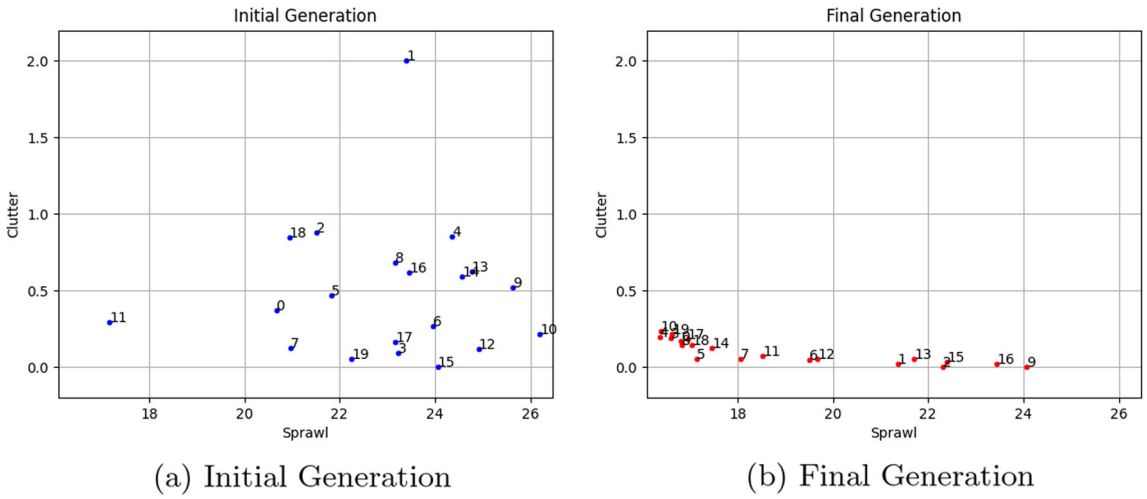
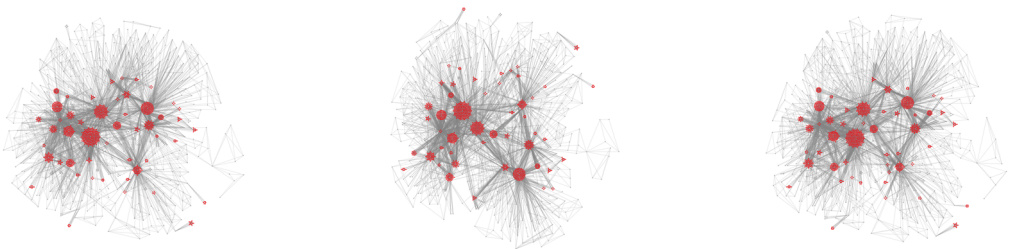


Fig. 22 An enlarged view of layout No. 4 in the final generation



(a) Initial Generation (b) Final Generation

Fig. 23 Evaluation results of population. (Experiment 3-L: larger ClustersizeRatio, focus on metanodes consisting of high-degree nodes)



(a) Layout No. 4, the optimal layout for Sprawl. (b) Layout No. 9, the optimal layout for Clutter. (c) Layout No. 5, the optimal layout for both.

Fig. 24 Examples of layouts in the final generation

Table 5 The number of metanodes and the response time for each experiments

	All metanodes	The target metanodes of optimization	Response time(s)
Experiment 1-S	769	769	2470.47
Experiment 1-L	461	461	1483.59
Experiment 2-S	769	95	2426.05
Experiment 2-L	461	23	1455.81
Experiment 3-S	769	33	2431.09
Experiment 3-L	461	49	1435.82

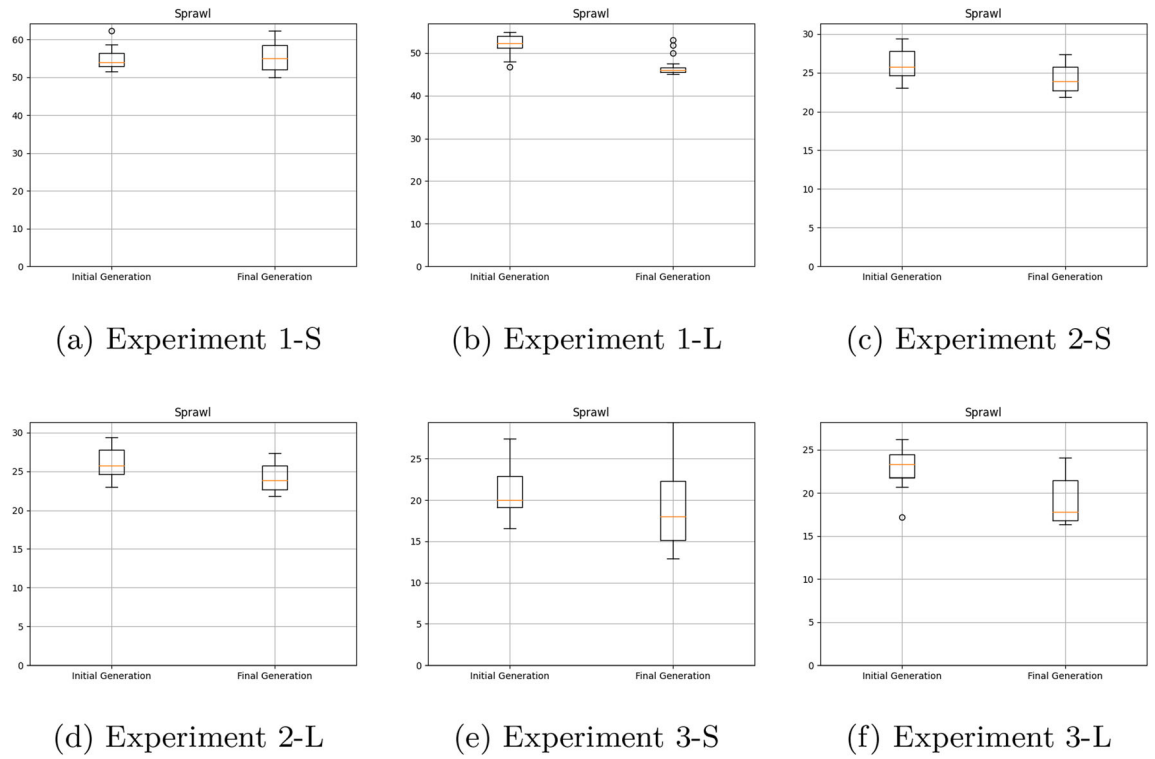


Fig. 25 Comparison of the Sprawl penalties for all six experiments

even for large graphs. Furthermore, the increase in the computational cost for large graphs is not a significant concern. Layouts may improve even after several iterations from the initial position. As the number of metanodes increases, more iterations may be required for complete convergence, but this is not a substantial issue. We aim to explore the potential applications of our optimization method to layouts generated by different algorithms or datasets of various sizes.

References

Aslam F, Mohmand YT, Ferreira P, Memon BA, Khan M, Khan M (2020) Network analysis of global stock markets at the beginning of the coronavirus disease (covid-19) outbreak. Borsa Istanbu Rev 20:49–61

Barreto AdMS, Barbosa HJ (2000) Graph layout using a genetic algorithm. In: Proceedings. Vol 1. Sixth Brazilian symposium on neural networks, IEEE, pp 179–184

Battista GD, Eades P, Tamassia R, Tollis IG (1998) Graph drawing: algorithms for the visualization of graphs. Prentice Hall, New Jersey

Bennett C, Ryall J, Spalteholz L, Gooch A (2007) The aesthetics of graph visualization. In: CAe, pp 57–64

Biedl T, Marks J, Ryall K, Whitesides S (1998) Graph multidrawing: finding nice drawings without defining nice. In: International symposium on graph drawing, Springer, pp 347–355

Deb K, Jain H (2013) An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part i: solving problems with box constraints. IEEE Trans Evol Comput 18(4):577–601

Deb K, Pratap A, Agarwal S, Meyarivan T (2002) A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Trans Evol Comput 6(2):182–197
Didimo W, Montecchiani F (2014) Fast layout computation of clustered networks: algorithmic advances and experimental analysis. Inf Sci 260:185–199
Dunne C, Ross SI, Shneiderman B, Martino M (2015) Readability metric feedback for aiding node-link visualization designers. IBM J Res Dev 59(2/3):14:1-14:16
Eades P, Hong S-H, Nguyen A, Klein K (2017) Shape-based quality metrics for large graph visualization. J Graph Algorithms Appl 21(1):29–53
Eloranta T, Mäkinen E (2001) TimGA: a genetic algorithm for drawing undirected graphs. Divulgaciones Mat 2:155–171
Ferreira J d M, Do Nascimento HA, Foulds L R (2018) An evolutionary algorithm for an optimization model of edge bundling. Information 9(7):154
Fonseca CM, Fleming PJ (1993) Genetic algorithms for multiobjective optimization: formulation, discussion and generalization. In: Proceedings of the 5th international conference on genetic algorithms, vol 93, pp 416–423
Fregnan E, Fröhlich J, Spadini D, Bacchelli A (2023) Graph-based visualization of merge requests for code review. J Syst Softw 195:111506
Fruchterman TM, Reingold E M (1991) Graph drawing by force-directed placement. Softw Pract Exp 21(11):1129–1164
Gansner ER, Hu Y, Kobourov S (2010) Gmap: visualizing graphs and clusters as maps. In: 2010 IEEE pacific visualization symposium (PacificVis), IEEE, pp 201–208
Goldberg DE (1989) Genetic algorithms in search, optimization and machine learning. Addison-Wesley, Reading, MA
Groves LJ, Michalewicz Z, Elia PV, Janikow CZ (1990) Genetic algorithms for drawing directed graphs. In: Methodologies for intelligent systems, 5th proceedings of the fifth international symposium, pp 268–276
Gunantara N (2018) A review of multi-objective optimization: methods and its applications. Cogent Eng 5(1):1502242
Hiroyasu T, Miki M, Watanabe S, Sakoda T, Kamiura J (2002) Evaluation of genetic algorithm for objective computation methods. Sci Eng Rev Doshisha Univ 43(1):41–52
Huang W, Huang ML, Lin C-C (2016) Evaluating overall quality of graph visualizations based on aesthetics aggregation. Inf Sci 330:444–454
Huang W, Huang M (2010) Exploring the relative importance of crossing number and crossing angle. In: Proceedings of the 3rd international symposium on visual information communication, pp 1–8
Itoh T, Klein K (2015) Key-node-separated graph clustering and layouts for human relationship graph visualization. IEEE Comput Graphics Appl 35(6):30–40
Khan S, Bilal M, Sharif M, Khan FA (2011) A solution to bipartite drawing problem using genetic algorithm. In: Advances in swarm intelligence: second international conference, ICSI 2011, Chongqing, China, June 12-15, 2011, proceedings, Part I 2, Springer, pp 530–538
Kieffer S, Dwyer T, Marriott K, Wybrow M (2015) Hola: Human-like orthogonal network layout. IEEE Trans Visual Comput Graphics 22(1):349–358
Knowles JD, Thiele L, Zitzler E (2006) A tutorial on the performance assessment of stochastic multiobjective optimizers. Technical report 214, computer engineering and networks laboratory
Knowles JD, Thiele L, Zitzler E (2006) A tutorial on the performance assessment of stochastic multiobjective optimizers. Technical report 214, computer engineering and networks laboratory
Knowles J, Corne D (1999) The pareto archived evolution strategy: a new baseline algorithm for pareto multiobjective optimisation. In: Proceedings of the 1999 congress on evolutionary computation-CEC99 (Cat. No. 99TH8406), vol 1, IEEE, pp 98–105
Laguna M, Mart'ı R, Valls V (1997) Arc crossing minimization in hierarchical digraphs with tabu search. Comput Oper Res 24(12):1175–1186
Liu Z, Itoh T, Dawson JQ, Munzner T (2020) The sprawlter graph readability metric: combining sprawl and area-aware clutter. IEEE Trans Visual Comput Graphics 26(6):2180–2191
Neta BM, Ara'ujo GH, Guimarães FG, Mesquita RC, Ekel PY (2012) A fuzzy genetic algorithm for automatic orthogonal graph drawing. Appl Soft Comput 12(4):1379–1389
Nguyen QH, Eades P, Hong S-H (2017) Towards faithful graph visualizations. arXiv preprint arXiv:1701.00921
Nguyen Q, Eades P, Hong S-H (2013) On the faithfulness of graph visualizations. In: visualization symposium (PacificVis), 2013 IEEE pacific, IEEE, pp 209–216
Pinaud B, Kuntz P, Lehn R (2004) Dynamic graph drawing with a hybridized genetic algorithm. In: Adaptive computing in design and manufacture VI, pp 365–375
Purchase H (1997) Which aesthetic has the greatest effect on human understanding? In: International symposium on graph drawing, pp 248–261
Purchase HC, Cohen RF, James M (1996) Validating graph drawing aesthetics. In: Graph drawing: symposium on graph drawing, GD'95 Passau, Germany, September 20–22, 1995 Proceedings 3, Springer, pp 435–446
Purchase HC (2002) Metrics for graph drawing aesthetics. J Vis Lang Comput 13(5):501–516
Saga R (2018) Validation of quantitative measures for edge bundling by comparing with human feeling. In: EuroVis (Posters), pp 25–27
Srinivas N, Deb K (1994) Multiobjective optimization using nondominated sorting in genetic algorithms. Evol Comput 2(3):221–248
Suh A, Hajij M, Wang B, Scheidegger C, Rosen P (2019) Persistent homology guided force-directed graph layouts. IEEE Trans Visual Comput Graphics 26(1):697–707
Taylor M, Rodgers P (2005) Applying graphical design techniques to graph visualisation. In: Ninth international conference on information visualisation (IV'05), IEEE, pp 651–656
Tian Y, Si L, Zhang X, Cheng R, He C, Tan KC, Jin Y (2021) Evolutionary large-scale multi-objective optimization: a survey. ACM Comput Surv (CSUR) 54(8):1–34

Utech J, Branke J, Schmeck H, Eades P (1998) An evolutionary algorithm for drawing directed graphs. In: Proceedings of the international conference on imaging science, systems and technology, pp 154–160
Valeri M, Baggio R (2021) Italian tourism intermediaries: a social network analysis exploration. Curr Issue Tour 24(9):1270–1283
Wang Y, Jin Z, Wang Q, Cui W, Ma T, Qu H (2019) Deepdrawing: a deep learning approach to graph drawing. IEEE Trans Visual Comput Graphics 26(1):676–686
Ware C, Purchase H, Colpoys L, McGill M (2002) Cognitive measurements of graph aesthetics. Inf Vis 1(2):103–110
Wu HT, Guo LH, Wang MJ, Yang J (2019) Optimization algorithms study and implementation on graph drawing based on xml document. Procedia Comput Sci 154:33–39
Yoghourdjian V, Archambault D, Diehl S, Dwyer T, Klein K, Purchase HC, Wu H-Y (2018) Exploring the limits of complexity: a survey of empirical studies on graph visualisation. Vis Inform 2(4):264–282
You J, Ying R, Ren X, Hamilton W, Leskovec J (2018) Graphrnn: generating realistic graphs with deep auto-regressive models. In: International conference on machine learning, PMLR, pp 5708–5717
Zhang Q-G, Liu H-Y, Zhang W, Guo Y-J (2005) Drawing undirected graphs with genetic algorithms. In: Advances in natural computation: first international conference, ICNC 2005, Changsha, China, August 27-29, 2005, proceedings, Part III 1, pp 28–36 Springer
Zitzler E (1999) Evolutionary algorithms for multiobjective optimization: methods and applications, vol 63. Shaker, Ithaca
Knowles JD, Thiele L, Zitzler E (2006) A tutorial on the performance assessment of stochastic multiobjective optimizers. Technical report 214, computer engineering and networks laboratory