Evolutionary Meta Layout of Graphs

Miro Spönemann, Björn Duderstadt, and Reinhard von Hanxleden

Department of Computer Science, Christian-Albrechts-Universität zu Kiel
{msp,bdu,rvh}@informatik.uni-kiel.de

Abstract. A graph drawing library is like a toolbox, allowing experts to select and configure a specialized algorithm in order to meet the requirements of their diagram visualization application. However, without expert knowledge of the algorithms the potential of such a toolbox cannot be fully exploited. This gives rise to the question whether the process of selecting and configuring layout algorithms can be automated such that good layouts are produced. In this paper we call this kind of automation "*meta layout*." We propose a genetic representation that can be used in meta heuristics for meta layout and contribute new metrics for the evaluation of graph drawings. Furthermore, we examine the use of an evolutionary algorithm to search for optimal solutions and evaluate this approach both with automatic experiments and a user study.

Keywords: graph drawing · layout algorithms · evolutionary algorithms · meta layout · readability metrics · user study

1 Introduction

There are many different approaches for drawing graphs, and all have their specific strengths and weaknesses. Therefore successful graph drawing libraries include multiple algorithms, and usually they offer numerous configuration options to allow users to tailor the generated layouts to their needs. However, the proper choice of a layout algorithm as well as its configuration often require detailed knowledge of the background of these algorithms. Acquiring such knowledge or simply testing all available configuration options is not feasible for users who require quick results.

An inspiring idea was communicated by Biedl et al. [1]: by displaying multiple layouts of the same graph, the user may select those that best match her or his expectations. In this paper we build on that idea and apply meta heuristics for generating a variation of layouts using existing layout libraries.

We introduce the notion of *abstract layout*, that is the annotation of graphs with directives for layout algorithm selection and configuration. *Concrete layout* is a synonym for the *drawing* of a graph and is represented by the annotation of graph elements with position and size data. When a layout algorithm is executed on a graph, it transforms its abstract layout into a concrete layout. By *meta layout* we denote an automatic process of generating abstract layouts.

Our contributions are a genetic representation of abstract layouts, metrics for convenient evaluation of *aesthetic criteria* [2], and operations for applying an evolutionary algorithm for meta layout. Furthermore, we propose a simple method for adapting the weights of aesthetic criteria according to the user-selected layouts, supporting the approach of Biedl et al. mentioned above. We performed automated as well as user-based experiments in order to evaluate our proposed method. The results show that the method is well accepted by users and produces at least equally readable drawings compared to a manual configuration approach.

The remainder of this paper is organized as follows. In Sect. 2 we discuss related work on evolutionary graph layout and layout configuration. Sect. 3 introduces the necessary data structures and fitness function that enable the evolutionary process, which is described in Sect. 4. In Sect. 5 we report experimental results on the effectiveness and applicability of these methods. We conclude and outline prospective work in Sect. 6.

2 Related Work

Several authors have proposed evolutionary algorithms where the individuals are represented by lists of coordinates for the positions of the nodes of a graph [3,4,5,6,7,8,9]. Here, in contrast, we do not include any specific graph in our encoding of individuals, hence we can apply the result of our evolutionary algorithm to any graphs, even if they were not considered during the evolutionary process. Furthermore, we benefit from all features that are already supported by the existing algorithms, while previous approaches for evolutionary layout were usually restricted to draw edges as straight lines and did not consider additional features such as edge labels.

Other works have focused on integrating meta heuristics in existing layout methods. De Mendonça Neto and Eades proposed a system for automatic learning of parameters of a simulated annealing algorithm [10]. Utech et al. introduced a genetic representation that combines the layer assignment and node ordering steps of the layer-based drawing approach with an evolutionary algorithm [11]. Such a combination of multiple NP-hard steps is also applied by Neta et al. for the *topology-shape-metrics* approach [12]. They use an evolutionary algorithm to find planar embeddings (*topology* step) for which the other steps (*shape* and *metrics*) are able to create good layouts.

Bertolazzi et al. proposed a system for automatic selection of layout algorithms that best match the user's requirements [13]. The system is initialized by evaluating the available algorithms with respect to a set of aesthetic criteria using randomly generated graphs of different sizes. The user has to provide a ranking of the criteria according to her or his preference. When a layout request is made, the system determines the difference between the user's ranking and the evaluation results of each algorithm for graphs of similar size as the current input graph. The algorithms with the lowest difference are offered to the user.

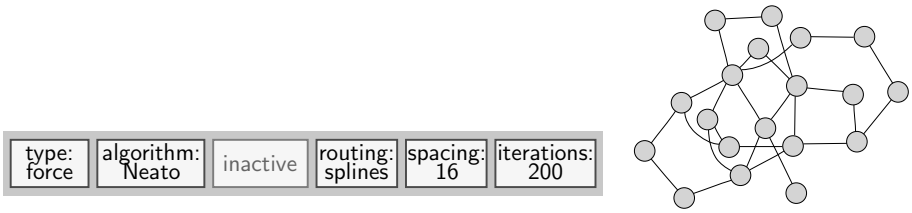
Similarly, Niggemann and Stein proposed to build a database that maps vectors of structural graph features, e. g. the number of nodes and the number of connected components, to the most suitable layout algorithm with respect to some predefined combination of aesthetic criteria [14]. These data are gathered by applying the algorithms to a set of "typical" graphs. A suitable algorithm for a given input graph is chosen by measuring its structural features and comparing them with the entries present in the database. Both the approaches of Bertolazzi et al. and Niggemann and Stein are restricted to selecting layout algorithms. Here, in contrast, we seek to configure arbitrary parameters of algorithms in addition to their selection.

Archambault et al. combined graph clustering with layout algorithm selection in a *multi-level* approach [15]. The clustering process is tightly connected with the algorithm selection, since both aspects are based on topological features of the input graph. When a specific feature is found, e. g. a tree or a clique, it is extracted as a subgraph and processed with a layout algorithm that is especially suited for that feature. This kind of layout configuration depends on detailed knowledge of the behavior of the algorithms, which has to be encoded explicitly in the system, while the solution presented here can be applied to any algorithm independently of their behavior.

3 Genotypes and Phenotypes

The *genotype* of an individual is its genetic code, while the *phenotype* is the total of its observable characteristics. In biology a phenotype is formed from its genotype by growing in a suitable environment. We propose to use abstract layouts (configurations) as genotypes, and concrete layouts (drawings) as phenotypes. The "environment" for this kind of phenotypes is a graph. We generate the concrete layout $L(\lambda)$ that belongs to a given abstract layout λ by applying all parameters encoded in λ to the chosen layout algorithm A, which is also encoded in λ, and executing A on the graph given by the environment. This encoding of parameters and algorithm selection is done with a set of *genes*, which together form a *genome*. A gene consists of a *gene type* with an assigned value. The gene type has an identifier, a data type (integer, floating point, Boolean, or enumeration), optional lower and upper bounds, and an optional parameter controlling the standard deviation of Gaussian distributions.

We assign each layout algorithm to a *layout type* depending on the underlying approach implemented in the algorithm. The main layout types are *layer-based*, *force-based*, *circular*, *orthogonal*, *tree*, and *planar*. Each algorithm A has a set P_A of parameters that control the behavior of A. We consider the union $\mathcal{P} = \bigcup P_A$ of all parameters, which we call the set of *layout options*. Each genome contains a gene g_T for selecting the layout type, a gene g_A for selecting the layout algorithm, and one for each layout option in $\mathcal{P}$. It is also possible to use only a subset of these genes, as long as all generated genomes contain the same subset. Such a restriction can serve to focus on the selected layout options in the optimization process, while other options are kept constant.



(a) Genotype **(b)** Phenotype

Fig. 1. (a) A genome with six genes. The layout type gene is set to force-based algorithms, the layout algorithm gene is set to a specific algorithm named "Neato", and three parameters of that algorithm are set with the remaining genes. One gene is inactive because the corresponding layout option is not supported by Neato. (b) A phenotype of the genome, represented by a layout generated by Neato for an arbitrary graph.

Some genes of a genome are dependent of each other. The gene g_A, for instance, is constrained to a layout algorithm that belongs to the layout type selected in g_T. Furthermore, the layout algorithm A selected in g_A does not support all layout options in $\mathcal{P}$, therefore the options in $\mathcal{P} \setminus P_A$, i. e. those not supported by A, are marked as *inactive*. A genome with six genes and a possible phenotype are shown in Fig. 1.

Inactive genes of a genome X do not contribute to the characteristics of the phenotype of X, i. e. of its drawing, hence two genomes that differ only in their inactive genes may produce the same drawing. On the other hand, some layout algorithms are randomized and produce different drawings when executed twice with the same configuration. However, we assume that drawings that result from the same configuration tend to be similar with respect to our fitness function, hence this ambiguity is probably not noticeable in practice.

3.1 Fitness Function

Our genotypes have a completely different representation compared to previous evolutionary layout algorithms. The phenotypes, in contrast, are commonly represented by graph layouts, hence we can apply the same approach to fitness evaluation as previous solutions, that is the evaluation of aesthetic criteria [2].

Some authors used a linear combination of specific criteria as fitness function [16,4,5]. For instance, given a graph layout L, the number of edge crossings $\kappa(L)$, and the standard deviation of edge lengths $\delta(L)$, the optimization goal could be to minimize the cost function $f(L) = w_\mathrm{c}\kappa(L) + w_\mathrm{d}\delta(L)$, where suitable scaling factors w_c and w_d are usually determined experimentally. The problem of this approach is that the values resulting from $f(L)$ have no inherent meaning apart from the general assumption "the smaller $f(L)$, the better the layout L." As a consequence, the cost function can be used only as a relative measure, but not to determine the absolute quality of layouts.

An improved variant, proposed by several authors, is to normalize the criteria to the range between 0 and 1 [17,2,7,8,9]. However, this is still not sufficient to effectively measure absolute layout quality. For instance, Tettamanzi normalizes the edge crossings $\kappa(L)$ with the formula $\mu_c(L) = \frac{1}{\kappa(L)+1}$ [8]. For the complete graph K_5, which is not planar, even the best layouts yield a result of $\mu_c(L) = 50\%$, suggesting that the layout is only half as good as it could be. Purchase proposed to scale the number of crossings against an upper bound $\kappa_{\max}$ defined as the number that results when all pairs of edges that are not incident to the same node cross each other [2]. Her formula is $\mu_c(L) = 1 - \frac{\kappa(L)}{\kappa_{\max}}$ if $\kappa_{\max} > 0$ and $\mu_c(L) = 1$ otherwise. Purchase herself notes that this definition "is biased towards high values." For instance, the graph N14 used in her evaluations has 24 nodes, 36 edges, and $\kappa_{\max} = 558$. All layouts with up to 56 crossings would result in $\mu_c(L) > 90\%$. When tested with a selection of 28 layout algorithms, all of them resulted in layouts with less that 56 crossings (the best had only 11 crossings), hence the formula of Purchase would assign a very high fitness to all these generated layouts.

We propose new normalization functions that aim at well-balanced distributions of values among typical results of layout algorithms. A *layout metric* is a function μ that maps graph layouts L to values $\mu(L) \in [0, 1]$. Given layout metrics $\mu_1, \ldots, \mu_k$ with weights $w_1, \ldots, w_k \in [0, 1]$, we compute the fitness of a graph layout L by

$$f(L) = \frac{1}{\sum_{i=1}^{k} w_k} \sum_{i=1}^{k} w_k \mu_k(L) \quad . \tag{1}$$

In the following we describe some of the metrics we have used in conjunction with our proposed genotype representation and evolutionary algorithm. The goal of these metrics is to allow an intuitive assessment of the respective criteria, which means that the worst layouts shall have metric values near 0%, the best ones shall have values near 100%, and moderate ones shall score around 50%. The metrics should be parameterized such that this spectrum of values is exhausted for layouts that are generated by typical layout algorithms, allowing to clearly distinguish them from one another. We evaluated to which extent our proposed metrics satisfy these properties based on an experimental analysis. The results confirm that our formulae are very suited to the stated goals. However, we omit the details of these experiments due to space limitations; they are available in a technical report [18], which also contains formulae for more aesthetic criteria.

The basic idea behind each of our formulae is to define a certain *input split value* x_s such that if the value of the respective criterion equals x_s, the metric is set to a defined *output split value* μ^*. Values that differ from x_s are scaled towards 0 or 1, depending on the specific criterion. The advantage of this approach is that different formulae can be applied to the ranges below and above the split value, simplifying the design of metrics that meet the goals stated above. The approach involves several constants, which we determined experimentally.

Let $G = (V, E)$ be a directed graph with a layout L. Let $n = |V|$ and $m = |E|$.

Number of crossings. Similarly to Purchase we define a virtual upper bound $\kappa_{\max} = m(m-1)/2$ on the number of crossings [2]. We call that bound virtual because it is valid only for straight-line layouts, while layouts where edges have bend points can have arbitrarily many crossings. Based on the observation that crossings tend to be more likely when there are many edges and few nodes, we further define an input split value

$$\kappa_{\mathrm{s}} = \min\left\{\frac{m^3}{n^2},\ (1-\mu_{\mathrm{c}}^*)\kappa_{\max}\right\}\ . \tag{2}$$

μ_{c}^* is the corresponding output split value, for which we chose $\mu_{\mathrm{c}}^* = 10\%$. The exponents of m and n are chosen such that the split value becomes larger when the m/n ratio is high. We denote the number of crossings as $\kappa(L)$. Layouts with $\kappa(L) < \kappa_{\mathrm{s}}$ yield metric values above μ_{c}^*, while layouts with $\kappa(L) > \kappa_{\mathrm{s}}$ yield values below μ_{c}^*. This is realized with the formula

$$\mu_{\mathrm{c}}(L) = \begin{cases} 1 & \text{if } \kappa_{\max} = 0, \\ 0 & \text{if } \kappa(L) \geq \kappa_{\max} > 0, \\ 1 - \frac{\kappa(L)}{\kappa_{\mathrm{s}}}(1-\mu_{\mathrm{c}}^*) & \text{if } \kappa(L) \leq \kappa_{\mathrm{s}}, \\ \left(1 - \frac{\kappa(L)-\kappa_{\mathrm{s}}}{\kappa_{\max}-\kappa_{\mathrm{s}}}\right)\mu_{\mathrm{c}}^* & \text{otherwise.} \end{cases} \tag{3}$$

Area. Let $w(L)$ be the width and $h(L)$ be the height of the drawing L. The area required to draw a graph depends on the number of nodes and edges, hence we define a *relative area*

$$\alpha(L) = \frac{w(L)h(L)}{(n+m)^2} \tag{4}$$

that takes into account the number of elements in the graph. We square that number because we observed that many drawings of larger graphs require a disproportionately high area. We split the output values at two points $\mu_{\mathrm{a}}^* = 10\%$ and $\mu_{\mathrm{a}}^{**} = 95\%$, with corresponding input split values $\alpha_{\mathrm{s}1}$ and $\alpha_{\mathrm{s}2}$. Values below μ_{a}^* are met when $\alpha(L) > \alpha_{\mathrm{s}1}$, values above μ_{a}^{**} are met when $\alpha(L) < \alpha_{\mathrm{s}2}$, and values in-between are scaled proportionally. The constants $\alpha_{\mathrm{s}1}$ and $\alpha_{\mathrm{s}2}$ have been determined experimentally as 1000 and 50, respectively. We define the area metric as

$$\mu_{\mathrm{a}}(L) = \begin{cases} \frac{\alpha_{\mathrm{s}1}}{\alpha(L)}\mu_{\mathrm{a}}^* & \text{if } \alpha(L) > \alpha_{\mathrm{s}1}, \\ 1 - \frac{\alpha(L)}{\alpha_{\mathrm{s}2}}(1-\mu_{\mathrm{a}}^{**}) & \text{if } \alpha(L) < \alpha_{\mathrm{s}2}, \\ \left(1 - \frac{\alpha(L)-\alpha_{\mathrm{s}2}}{\alpha_{\mathrm{s}1}-\alpha_{\mathrm{s}2}}\right)(\mu_{\mathrm{a}}^{**}-\mu_{\mathrm{a}}^*) + \mu_{\mathrm{a}}^* & \text{otherwise.} \end{cases} \tag{5}$$

Edge length uniformity. We measure this criterion with the standard deviation $\sigma_\lambda(L)$ of edge lengths and compare it against the average edge length $\bar{\lambda}(L)$, which we use as input split value. We define

$$\mu_{\mathrm{u}}(L) = \begin{cases} \frac{\bar{\lambda}(L)}{\sigma_\lambda(L)}\mu_{\mathrm{u}}^* & \text{if } \sigma_\lambda(L) \geq \bar{\lambda}(L), \\ 1 - \frac{\sigma_\lambda(L)}{\bar{\lambda}(L)}(1-\mu_{\mathrm{u}}^*) & \text{otherwise,} \end{cases} \tag{6}$$

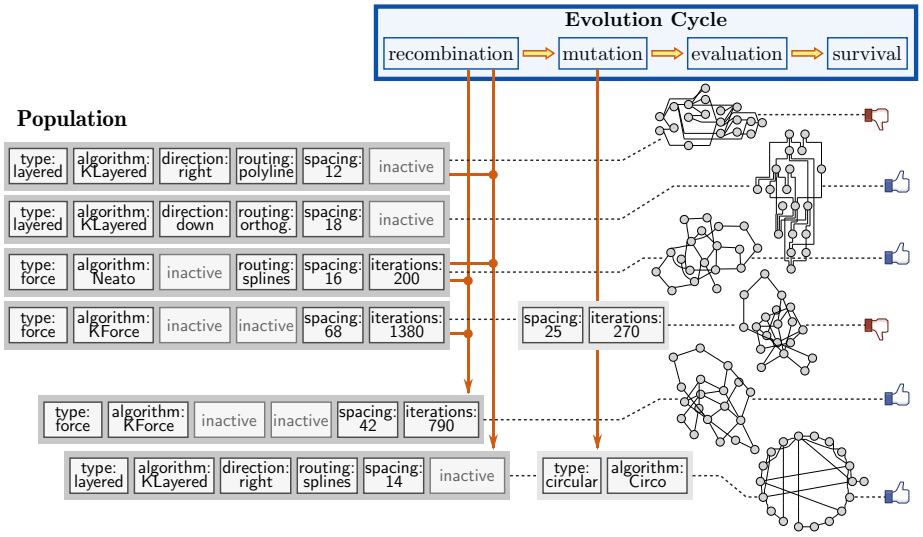


Fig. 2. Evolutionary layout example: starting with a population of four genomes, two new genomes are created through recombination, two genomes are mutated, and four of the resulting genomes survive after their evaluation.

where the output split value $\mu_{\mathrm{u}}^{*} = 20\%$ corresponds to the metric value that results when the standard deviation equals the average.

4 Evolutionary Process

The genetic encoding presented in Sect. 3 can serve as basis for numerous meta heuristics. In this section we discuss one possible heuristic with the goal of creating a starting point of further research, without claiming that this is the ultimate solution. We use an evolutionary algorithm, a popular method for searching large solution spaces.

A *population* is a set of genomes. An *evolution cycle* is a function that modifies a population with four steps, which are explained below. The evolutionary algorithm executes the evolution cycle repeatedly, checking some termination condition after each execution. Simple conditions for fully automatic optimization are to limit the number of iterations and to check whether the fitness of the best individual exceeds a certain threshold. Alternatively, the user can be involved by manually controlling when to execute the next evolution cycle and when to stop the process. The four steps of the evolution cycle are discussed in the following and exemplified in Fig. 2.

1. Recombination. New genomes are created by crossing random pairs of existing genomes. A crossing of two genomes is created by crossing all their genes. Two integer or floating point typed genes are crossed by computing their

average value, while for other data types one of the two values is chosen randomly. Only a selection of the fittest individuals is considered for mating.

When the parent genomes have different values for the layout algorithm gene, the child is randomly assigned one of these algorithms. As a consequence, the *active / inactive* statuses of the other genes of the child must be adapted such that they match the chosen algorithm A: each gene g is made active if and only if A supports the layout option associated to g.

2. Mutation. Genomes have a certain probability of mutating. A mutation is done by randomly modifying its genes, where each gene g has an individual mutation probability p_g depending on its type. We assign the highest p_g values to genes with integer or floating point values, medium values to genes with Boolean or enumeration values, and the lowest values to the layout algorithm and layout type genes. Let g be a gene with value x. If the data type of g is integer or floating point, the new value x' is determined using a Gaussian distribution using x as its average and the standard deviation assigned to the gene type of g. If x' exceeds the upper or lower bound assigned to the gene type of g, it is corrected to a value between x and the respective bound. For genes with other types, which have no specific order, a new value is chosen based on a uniform distribution over the finite set of values, excluding the previous value. When the layout algorithm gene mutates, the *active / inactive* statuses of other genes must be updated as described for the recombination step.

3. Evaluation. A fitness value is assigned to each genome that does not have one yet (see Sect. 3.1), which involves executing the encoded layout algorithm in order to obtain a corresponding phenotype. The population is sorted using these fitness values.

4. Survival. Only the fittest individuals survive. Checking all genomes in order of descending fitness, we include each genome X in the set of survivors if and only if it meets the following requirements: (i) its fitness exceeds a certain minimum, (ii) the maximal number of survivors is not reached yet, and (iii) the distance of X to other individuals is sufficient. The latter requirement serves to support the diversity of the population. Comparing all pairs of individuals would require a quadratic number of distance evaluations, therefore we determine the distance only to some random samples X' from the current set of survivors. We determine the distance $d(X, X')$ of two individuals X, X' by computing the sum of the differences of the gene values. In order to meet the third requirement, $d(X, X') \geq d_{\min}$ must hold for a fixed minimal distance $d_{\min}$.

4.1 Choosing Metric Weights

The fitness function discussed in Sect. 3.1 uses layout metrics $\mu_1, \ldots, \mu_k$ and weights $w_1, \ldots, w_k \in [0, 1]$, where each w_i controls the influence of μ_i on the computed fitness. The question is how to choose suitable weights. Masui proposed to apply genetic programming to find a fitness function that best reflects the user's intention [19]. The computed functions are evolved as Lisp programs

and are evaluated with layout examples, which have to rated as "good" or "bad" by the user. A similar approach is used by Barbosa and Barreto [3], with the main difference that the fitness function is evolved indirectly by modifying a set of weights with an evolutionary algorithm. Additionally, they apply another evolutionary algorithm to create concrete layouts of a given graph. Both algorithms are combined in a process called *co-evolution*: the results of the weights evolution are used for the fitness function of the layout evolution, while the fitness of the weights is determined based on user ratings of sample layouts.

We have experimented with two much simpler methods, both of which involve the user: (a) the user directly manipulates the metric weights with sliders allowing values between 0 and 1, and (b) the user selects good layouts from the current population and the metric weights are automatically adjusted according to the selection. This second method builds on the assumption that the considered layout metrics are able to compute meaningful estimates of the absolute quality of any given layout (see Sect. 3.1). The higher the result of a metric, the higher its weight shall be. Let $\bar{\mu}_1, \ldots, \bar{\mu}_k$ be the average values of the layout metrics $\mu_1, \ldots, \mu_k$ for the selected layouts. Furthermore, let $w_1, \ldots, w_k$ be the current metric weights. For each $i \in \{1, \ldots, k\}$ we determine a *target weight*

$$w_i^* = \begin{cases} 1 - \frac{1}{2}\left(\frac{1-\bar{\mu}_i}{1-\mu_{\mathrm{w}}^*}\right)^2 & \text{if } \bar{\mu}_i \geq \mu_{\mathrm{w}}^*, \\ \frac{1}{2}\left(\frac{\bar{\mu}_i}{\mu_{\mathrm{w}}^*}\right)^2 & \text{otherwise,} \end{cases} \tag{7}$$

where μ_{w}^* is a constant that determines which metric result is required to reach a target weight of 50%. We chose $\mu_{\mathrm{w}}^* = 70\%$, meaning that mediocre metric results are mapped to rather low target weights. The square functions in Equation 7 are used to push extreme results even more towards 0 or 1. The new weight of the layout metric μ_i is $w_i' = \frac{1}{2}(w_i + w_i^*)$, i. e. the mean of the old weight and the target weight.

4.2 User Interface

We have experimented with a user interface that includes both variants for modifying metric weights, shown in Fig. 3. The window visualizes populations by presenting up to 16 small drawings of the evaluation graph, which represent the fittest individuals of the current population. 13 metrics are shown on the side of the window. The user may use the controls in the window to

- view the computed values of the layout metrics for an individual,
- directly set the metric weights,
- select one or more favored individuals for indirect adjustment of weights,
- change the population by executing an evolution cycle ("Evolve" button),
- restart the evolution with a new initial population ("Restart" button), and
- finish the process and select the abstract layout encoded in a selected individual ("Apply" button).

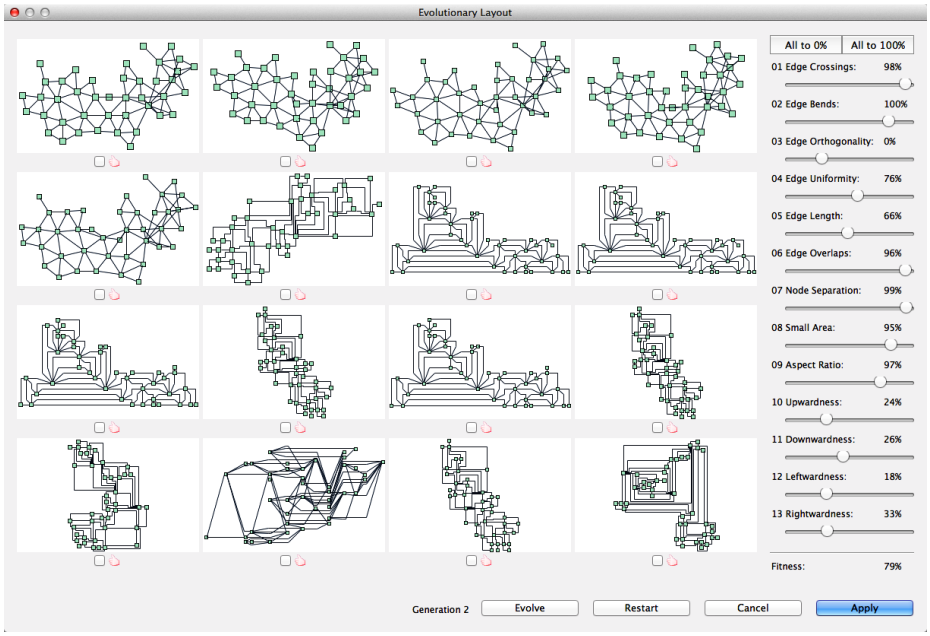


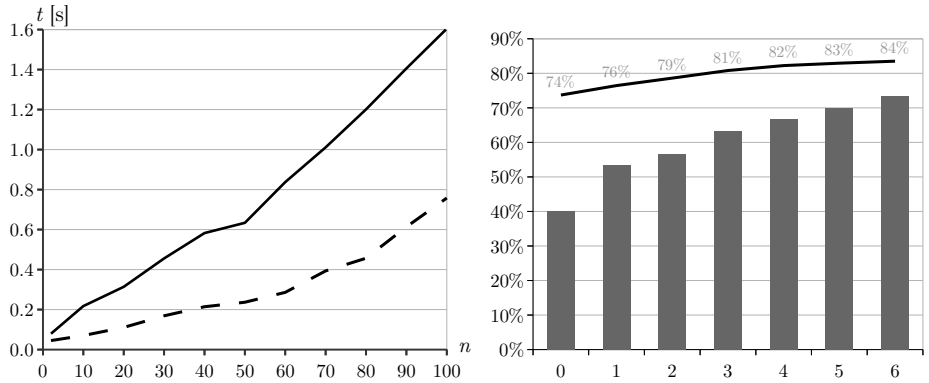
Fig. 3. User interface for evolutionary meta layout, showing drawings for 16 individuals of the current population. The check box below each proposed graph drawing is used to select favored layouts for automatic adaption of metric weights. The sliders on the right offer direct manipulation of the weights.

The indirect method for choosing weights, which adapts them according to the user's selection of favored layouts, is in line with the *multidrawing* approach introduced by Biedl et al. [1]. The main concept of that approach is that the user can select one of multiple offered drawings without the need of defining her or his goals and preferences in the first place. The multidrawing system reacts on the user's selection and generates new layouts that are similar to the selected ones. In our proposed method, this similarity is achieved by adjusting the fitness function such that the selected layouts are assigned a higher fitness, granting them better prospects in the competition against other layouts.

5 Evaluation

The methods presented in this paper have been implemented and evaluated in KIELER, an Eclipse-based open source project.[1] Our experiments included four layout algorithms provided by KIELER as well as five algorithms from the Graphviz library [20] and 22 algorithms from the OGDF library [21]. The total number of genes in each genome was 79.

[1] `http://www.informatik.uni-kiel.de/rtsys/kieler/`



(a) Execution time **(b)** Effectiveness

Fig. 4. (a) Execution time t plotted by number of nodes n with single core execution (solid line) and multicore execution (dashed line). (b) Result of the edge uniformity experiment. The line on top shows the fitness values of the best genomes for iterations 0 to 6 (horizontal axis), while the bars show the fractions of genomes that are set to force-type algorithms.

5.1 Execution Time

We tested the performance of evolutionary meta layout on a set of 100 generated graphs with varying number of nodes $2 \leq n \leq 100$ and $e = 1.5\,n$ edges. The tests have been executed with an Intel Xeon 2.5 GHz CPU. The population contained 16 genomes, the recombination operation bred 13 new genomes, and the mutation operation affected 60% of the whole population, thus 22.6 new genomes were created on average. This means that about 23 layout algorithm executions had to be performed for the evaluation operation of each evolution cycle, and each layout metric has been evaluated just as often. We measured the average execution time of one evolution cycle (i. e. a single iteration), which led to the results shown in Fig. 4a. The vast majority of time is spent in the evaluation step: on average 74% is taken by layout algorithm execution, and 20% is taken by metrics evaluation. The rather high execution time limits the number of evolution cycles that can be performed in an interactive environment. The consequence is that the evolutionary algorithm has to converge to an acceptable solution within few iterations. However, the evaluation step is very suitable for parallelization, since the evaluations are all independent. As seen in Fig. 4a, the total execution time can be reduced by half when run with multiple threads on a multicore machine (eight cores in this example).

5.2 Programmatic Experiments

We carried out three experiments in order to verify the effectiveness of the evolutionary approach. The experiments had different optimization goals: minimal

number of edge crossings, maximal number of edges pointing left, and optimal uniformity of edge lengths. In each experiment the corresponding layout metric was given a weight of 100%, while most other metrics were deactivated (except some basic metrics avoiding graph elements overlapping each other). The optimization goals were chosen such that they can be mapped to certain kinds of layout algorithms, allowing to validate the results according to prior knowledge of their behavior. 30 randomly generated graphs were used as evaluation graphs. In the crossing minimization experiment, for 60% of the graphs a planarization-based algorithm was selected as the genome with highest fitness after three or less iterations. This confirms the intuitive expectation, since planarization methods are most effective in minimizing edge crossings. In the experiment that aimed at edges pointing left, for 90% of the graphs a layer-based algorithm was selected as the genome with highest fitness after three or less iterations. Additionally, the layout option that determines the main direction of edges had to be set to *left*, which was accomplished in 83% of the cases. In the edge uniformity experiment, a force-based algorithm was selected for 63% of the graphs after three iterations, and for 73% of the graphs after six iterations (see Fig. 4b). This result matches the expectation, too, because force-based methods aim at drawing all edges with uniform length. In all experiments it could be observed that the average rating of genomes was consistently increasing after each iteration, but this increase became smaller with each iteration. We conclude that our proposed evolutionary meta layout approach can effectively optimize given aesthetic criteria, and in most cases the kind of layout algorithm that is automatically selected is consistent with the intuition. A very relevant observation is that the process tends to converge very quickly, often yielding good solutions after few iterations, e. g. as illustrated in Fig. 4b. On the other hand, in some cases the computation is trapped in local optima, which could possibly be avoided by improving the parameters of the evolutionary computation.

5.3 User Study

We have conducted a user study to determine the practical usefulness of our approach. The study is based on a set of 8 graphs, inspired by real-world examples that were found on the web, with between 15 and 43 nodes and 18 to 90 edges. 25 persons participated in the study: four members of our research group, 17 computer science students, and four persons who were not involved in computer science. The research group members are experts in graph layout technology and are likely to have predetermined opinions about which layout configurations to use in certain contexts, while the other participants can be regarded as novices for that matter. We expected that novice users would benefit more strongly from the evolutionary meta layout approach compared to the experts.

For each graph, the participants were presented three tasks regarding connectivity, e. g. finding the shortest path between two given nodes. The participants then had to find a layout configuration which they regarded as useful for working on the tasks. The test instructions encouraged the participants to improve the layout configuration until they were sure they had found a well readable layout.

Four of the graphs were treated with the user interface presented in Sect. 4.2, named EVOL in the following, which evolves a population of layout configurations and lets users pick configurations by their previews. They were free to use both the direct and the indirect method for modifying weights (Sect. 4.1). For the other four graphs, the participants were required to find layout configurations manually by choosing from a list of available layout algorithms and modifying parameters of the chosen algorithms. For each participant we determined randomly which graphs to treat with EVOL and which to configure with the manual method, called MANUAL in the following. After the participants had accepted a layout configuration for a graph, they worked on the respective tasks by inspecting the drawing that resulted from the configuration.

After all graphs were done, the participants were asked 6 questions about their subjective impression of the evolutionary approach. The overall response to these questions was very positive: on a scale from -2 (worst rating) to 2 (best rating), the average ratings were 1.0 for the quality of generated layouts, 0.8 for their variety, 1.2 for the time required for finding suitable layouts, 0.6 for the effectiveness of manually setting metric weights, and 1.5 for the effectiveness of adjusting metric weights by favoring individuals. Most notably, the indirect adjustment of metric weights was rated much higher than their direct manipulation. This indicates that most users prefer an intuitive interface based on layout proposals instead of manually setting parameters of the fitness function, since the latter requires to understand the meaning of all layout metrics.

The average rate of correct answers of non-expert users to the tasks was 77.4% for MANUAL and 79.8% for EVOL. The average time used to work on each task was lower by 7.5% with EVOL (131 seconds) compared to MANUAL (142 seconds). These differences are not statistically significant: the p-values resulting from a t-test on the difference of mean values are 29% for the correctness of answers and 23% for the working time. A more significant result ($p = 8.3\%$) is obtained when comparing the differences of EVOL and MANUAL working times between expert users and non-expert users. In contrast to the non-experts, expert users took more time to work on the tasks with EVOL (126 seconds) compared to MANUAL (107 seconds). Furthermore, the average rate of correct answers of expert users was equal for both methods. This confirms the assumption that the method proposed in this paper is more suitable in applications used by persons without expert knowledge on graph drawing.

Many participants commented that they clearly preferred EVOL over MANUAL. It could be observed that novice users were overwhelmed by the number of configuration parameters shown for the manual method. In many cases, they stopped trying to understand the effects of the parameters after some unsuccessful attempts to fine-tune the layout. Therefore the average time taken for finding a layout was lower for MANUAL (129 seconds) compared to EVOL (148 seconds). For the EVOL interface, on the other hand, similarly frustrating experiences were observed in few cases where the evolutionary algorithm apparently ran into local optima that did not satisfy the users' expectations. In these cases users were forced to restart the process with a new population.

The average number of applied evolution cycles was 3.1, which means that in most cases the participants found good solutions after very few iterations of the evolutionary algorithm. Furthermore, we measured the index of the layout chosen for working on the tasks on a scale from 0 to 15. The layout with index 0 has the highest fitness in the current population, while the layout with index 15 is the one with lowest fitness from the 16 fittest individuals. The average selected index was 2.3, a quite low value, suggesting that the computed fitness has a high correlation with the perceived quality of the layouts.

6 Conclusion

We introduced the notion of meta layout, which means creating an abstract layout by choosing and parameterizing a layout algorithm, which in turn generates a concrete layout of a graph. We presented a genetic representation of abstract layouts, layout metrics for building a fitness function, and an evolutionary algorithm for developing a population of abstract layouts. Furthermore, we proposed a simple method for the indirect adjustment of weights of layout metrics. Since the result of the evolutionary computation is not a concrete layout, but a layout configuration, it can be applied to any graph without repeating the process.

Our experiments partially confirmed the usefulness of the presented methods. Participants of the user study clearly preferred the evolutionary approach over the manual setting of parameters for layout algorithms, and they also liked to modify the fitness function indirectly rather than to adjust weights directly. The objective results about the effectivity working on tasks about graph connectivity were not statistically significant with respect to comparing our proposed method with the manual method. However, non-expert users clearly profited from the evolutionary method more than the experts did.

The evolutionary algorithm presented here is not the only heuristic for optimizing abstract layouts. Further work could evaluate other optimization heuristics that build on our genetic representation and compare them to the results of this paper. For instance, using a divide-and-conquer approach one could separately optimize each parameter of the layout algorithms, one after another.

References

1. Biedl, T., Marks, J., Ryall, K., Whitesides, S.: Graph multidrawing: Finding nice drawings without defining nice. In: Whitesides, S. (ed.) Graph Drawing, Lecture Notes in Computer Science, vol. 1547, pp. 347–355. Springer Berlin Heidelberg (1998)
2. Purchase, H.C.: Metrics for graph drawing aesthetics. Journal of Visual Languages and Computing 13(5), 501–516 (2002)
3. Barbosa, H.J.C., Barreto, A.M.S.: An interactive genetic algorithm with co-evolution of weights for multiobjective problems. In: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'01). pp. 203–210 (2001)

4. Branke, J., Bucher, F., Schmeck, H.: Using genetic algorithms for drawing undirected graphs. In: Proceedings of the Third Nordic Workshop on Genetic Algorithms and their Applications. pp. 193–206 (1996)
5. Eloranta, T., Mäkinen, E.: TimGA: A genetic algorithm for drawing undirected graphs. Divulgaciones Matemáticas 9(2), 155–170 (2001)
6. Groves, L.J., Michalewicz, Z., Elia, P.V., Janikow, C.Z.: Genetic algorithms for drawing directed graphs. In: Proceedings of the 5th International Symposium on Methodologies for Intelligent Systems. pp. 268–276 (1990)
7. Rosete-Suarez, A., Ochoa-Rodriguez, A.: Genetic graph drawing. In: Nolan, P., Adey, R.A., Rzevski, G. (eds.) Applications of Artificial Intelligence in Engineering XIII, Software Studies, vol. 1. WIT Press / Computational Mechanics (1998)
8. Tettamanzi, A.G.: Drawing graphs with evolutionary algorithms. In: Parmee, I.C. (ed.) Adaptive Computing in Design and Manufacture, pp. 325–337. Springer London (1998)
9. Vrajitoru, D.: Multiobjective genetic algorithm for a graph drawing problem. In: Proceedings of the Midwest Artificial Intelligence and Cognitive Science Conference. pp. 28–43 (2009)
10. de Mendonça Neto, C.F.X., Eades, P.D.: Learning aesthetics for visualization. In: Anais do XX Seminário Integrado de Software e Hardware. pp. 76–88 (1993)
11. Utech, J., Branke, J., Schmeck, H., Eades, P.: An evolutionary algorithm for drawing directed graphs. In: Proceedings of the International Conference on Imaging Science, Systems, and Technology (CISST'98). pp. 154–160. CSREA Press (1998)
12. Neta, B.M.d.M., Araujo, G.H.D., Guimarães, F.G., Mesquita, R.C., Ekel, P.Y.: A fuzzy genetic algorithm for automatic orthogonal graph drawing. Applied Soft Computing 12(4), 1379–1389 (2012)
13. Bertolazzi, P., Di Battista, G., Liotta, G.: Parametric graph drawing. IEEE Transactions on Software Engineering 21(8), 662–673 (Aug 1995)
14. Niggemann, O., Stein, B.: A meta heuristic for graph drawing: learning the optimal graph-drawing method for clustered graphs. In: Proceedings of the Working Conference on Advanced Visual Interfaces (AVI'00). pp. 286–289. ACM, New York, NY, USA (2000)
15. Archambault, D., Munzner, T., Auber, D.: Topolayout: Multilevel graph layout by topological features. IEEE Transactions on Visualization and Computer Graphics 13(2), 305–317 (2007)
16. Barreto, A.M.S., Barbosa, H.J.C.: Graph layout using a genetic algorithm. In: Proc. of the 6th Brazilian Symposium on Neural Networks. pp. 179–184 (2000)
17. Dunne, C., Shneiderman, B.: Improving graph drawing readability by incorporating readability metrics: A software tool for network analysts. Tech. Rep. HCIL-2009-13, University of Maryland (2009)
18. Spönemann, M., Duderstadt, B., von Hanxleden, R.: Evolutionary meta layout of graphs. Technical Report 1401, Christian-Albrechts-Universität zu Kiel, Department of Computer Science (Jan 2014), ISSN 2192-6247
19. Masui, T.: Evolutionary learning of graph layout constraints from examples. In: Proceedings of the 7th Annual ACM Symposium on User Interface Software and Technology (UIST'94). pp. 103–108. ACM (1994)
20. Gansner, E.R., North, S.C.: An open graph visualization system and its applications to software engineering. Software—Practice and Experience 30(11), 1203–1234 (2000)
21. Chimani, M., Gutwenger, C., Jünger, M., Klau, G.W., Klein, K., Mutzel, P.: The Open Graph Drawing Framework (OGDF). In: Tamassia, R. (ed.) Handbook of Graph Drawing and Visualization, pp. 543–569. CRC Press (2013)