Multicriteria Scalable Graph Drawing via Stochastic Gradient Descent, $(SGD)^2$

Reyan Ahmed, Felice De Luca, Sabin Devkota, Stephen Kobourov, and Mingwei Li

Abstract—Readability criteria, such as distance or neighborhood preservation, are often used to optimize node-link representations of graphs to enable the comprehension of the underlying data. With few exceptions, graph drawing algorithms typically optimize one such criterion, usually at the expense of others. We propose a layout approach, Multicriteria Scalable Graph Drawing via Stochastic Gradient Descent, $(SGD)^2$, that can handle multiple readability criteria. $(SGD)^2$ can optimize any criterion that can be described by a differentiable function. Our approach is flexible and can be used to optimize several criteria that have already been considered earlier (e.g., obtaining ideal edge lengths, stress, neighborhood preservation) as well as other criteria which have not yet been explicitly optimized in such fashion (e.g., node resolution, angular resolution, aspect ratio). The approach is scalable and can handle large graphs. A variation of the underlying approach can also be used to optimize many desirable properties in planar graphs, while maintaining planarity. Finally, we provide quantitative and qualitative evidence of the effectiveness of $(SGD)^2$: we analyze the interactions between criteria, measure the quality of layouts generated from $(SGD)^2$ as well as the runtime behavior, and analyze the impact of sample sizes. The source code is available on github and we also provide an interactive demo for small graphs.

Index Terms—Graph drawing, gradient descent, quality metrics

1 Introduction

Graphs represent relationships between entities and visualization of this information is relevant in many domains. Several criteria have been proposed to evaluate the readability of graph drawings, including the number of edge crossings, distance preservation, and neighborhood preservation. Such criteria evaluate different aspects of the drawing and different layout algorithms optimize different criteria. It is challenging to optimize multiple readability criteria at once and there are few approaches that can support this. Examples of approaches that can handle a small number of related criteria include the stress majorization framework of Wang *et al.* [46], which optimizes distance preservation via stress as well as ideal edge length preservation. The Stress Plus X (SPX) framework of Devkota *et al.* [15] can minimize the number of crossings, or maximize the minimum angle of edge crossings. While these frameworks can handle a limited set of related criteria, it is not clear how to extend them to arbitrary optimization goals, as such frameworks are dependent on particular mathematical formulations. For example, the SPX framework was designed for crossing minimization, which can be easily modified to handle crossing angle maximization (by adding a cosine factor to the optimization function). This idea can be applied only to a limited set of criteria but the majority of other criteria are incompatible with the basic formulation.

In this paper, we propose a general approach, Multicriteria Scalable Graph Drawing via Stochastic Gradient Descent, $(SGD)^2$, that can optimize a large set of drawing criteria, provided that the corresponding metrics that evaluate the criteria are differentiable functions. If the criterion is not naturally differentiable, we design a differentiable surrogate function to approximate and optimize the original criterion. In $(SGD)^2$, auto-differentiation tools are used for the gradient-based optimization. To demonstrate the flexibility of the approach, we consider a set of nine criteria: minimizing stress, maximizing node resolution, obtaining ideal edge lengths, maximizing neighborhood preservation, maximizing crossing angle, optimizing total angular resolution, minimizing aspect ratio, optimizing the Gabriel graph property, and minimizing edge crossings. We evaluate the effectiveness of our approach quantitatively and qualitatively with evidence drawn from a set of experiments. To illustrate the effectiveness and efficiency of multicriteria optimization, we evaluate the compatibility of every pair of criteria, measure the quality of each criterion, and demonstrate the distinctive looks of graph layouts under different drawing objectives. We also evaluate the runtime performance and the impact of sample sizes used in the optimization, and compare our methods with existing ones. We implemented our method with PyTorch. The code is available at: https://github.com/tiga1231/graph-drawing/tree/sgd. For demonstration purposes, we also built an interactive prototype (that implements full-batch gradient descent on small graphs) in JavaScript using tensorflow.js and D3.js, which is available on http://hdc.cs.arizona.edu/~mwli/graph-drawing/. This interactive prototype allows nodes to be moved manually and combinations of criteria can be optimized by selecting a weight for each; see Fig. 1.

• *The authors are with the Department of Computer Science, University of Arizona, Tucson, AZ 85721 USA. E-mail: {abureyanahmed, devkotasabin, mwli}@email.arizona.edu, felicedeluca@me.com, kobourov@cs.arizona.edu.*

Manuscript received 14 June 2021; revised 31 Jan. 2022; accepted 15 Feb. 2022. Date of publication 1 Mar. 2022; date of current version 2 May 2022.
This work was supported in part by NSF under Grants CCF-1740858, CCF-1712119, and DMS-1839274.
(Corresponding author: Reyan Ahmed.)
Recommended for acceptance by T. Dwyer.
Digital Object Identifier no. 10.1109/TVCG.2022.3155564

2 Related Work

Many criteria for the readability of graph drawings have been proposed [48], but graph layout algorithms are designed to (explicitly or implicitly) optimize a single one. For instance, a classic layout criterion is stress minimization [31], where stress is defined by $\sum_{i<j} w_{ij}(|X_i - X_j| - d_{ij})^2$. Here, X is a $n \times 2$ matrix containing coordinates for the n nodes, d_{ij} is typically the graph-theoretical distance between two nodes i and j and $w_{ij} = d_{ij}^{-\alpha}$ is a normalization factor with α equal to $0, 1$ or 2. Thus reducing the stress in a layout corresponds to computing node positions so that the actual distance between pairs of nodes is proportional to the graph theoretic distance between them. Optimizing stress can be accomplished by stress minimization, or stress majorization, which can speed up the computation [24].

Stress minimization corresponds to optimizing the global structure of the layout, as the stress metric takes into account all pairwise distances in the graph. The t-SNET algorithm of Kruiger *et al.* [34] optimizes neighborhood preservation, which captures the local structure of a graph, as the neighborhood preservation metric only considers distances between pairs of nodes that are close to each other. Optimizing local or global distance preservation can be seen as special cases of the more general dimensionality reduction approaches such as multi-dimensional scaling [35].

Purchase *et al.* [39] showed that the readability of graphs increases if a layout has fewer edge crossings. The underlying optimization problem is NP-hard and several graph drawing contests have been organized with the objective of minimizing the number of crossings in the graph drawings [1], [9]. Recently several algorithms that directly minimize crossings have been proposed [40], [42].

The negative impact on graph readability due to edge crossings can be mitigated if crossing pairs of edges have a large crossings angle [28]. Formally, the crossing angle of a straight-line drawing of a graph is the minimum angle between two crossing edges in the layout, and optimizing this property is also NP-hard. Recent graph drawing contests have been organized with the objective of maximizing the crossings angle in graph drawings and this has led to several heuristics for this problem [4], [13].

The algorithms above are very effective at optimizing the specific readability criterion they are designed for, but they cannot be directly used to optimize additional criteria. This is a desirable goal, since optimizing one criterion often leads to poor layouts with respect to one or more other criteria: for example, algorithms that optimize the crossing angle tend to create drawings with high stress and no neighborhood preservation [15].

Davidson and Harel [11] used simulated annealing to optimize different graph readability criteria (keeping nodes away from other nodes and edges, uniform edge lengths, minimizing edge crossings). Huang *et al.* [28] extended a force-directed algorithm to optimize crossing angle and angular resolution by incorporating two additional angle forces. The authors show that in addition to optimizing crossing angle and angular resolution, the algorithm also improves other desirable properties (average size of crossing angles, standard deviation of crossing angles, standard deviations of angular resolution, etc.). In a force-directed method

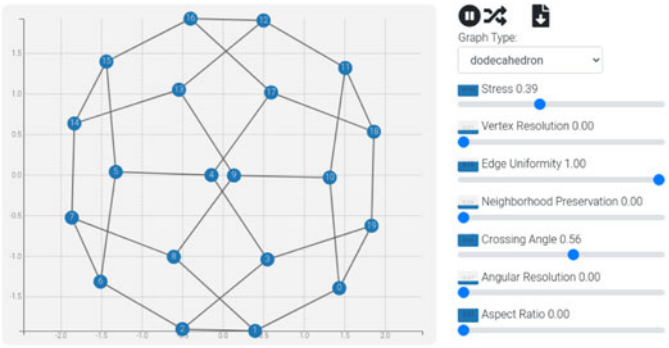

Fig. 1. An interactive prototype of $(SGD)^2$ simutaneously optimizing stress, edge uniformity and crossing angles on a dodecahedron.

similar to the algorithm proposed by Huang *et al.* [28], to optimize each criterion one needs to design a new force. The new force can be considered as a gradient update by hand, whereas $(SGD)^2$ is a gradient descent based algorithm where the gradients are computed automatically using auto-differentiation tools. Recently, several approaches have been proposed to simultaneously improve multiple layout criteria. Wang *et al.* [46] propose a revised formulation of stress that can be used to specify ideal edge direction in addition to ideal edge lengths in a graph drawing. Wang *et al.* [47] extended that stress formulation to produce structure-aware and smooth fish-eye views of graphs. Devkota *et al.* [15] also use a stress-based approach to minimize edge crossings and maximize crossing angles. Eades *et al.* [20] provided a technique to draw large graphs while optimizing different geometric criteria, including the Gabriel graph property. Although the approaches above are designed to optimize multiple criteria, they cannot be naturally extended to handle other optimization goals.

Constraint-based layout algorithms such as COLA [17], can be used to enforce separation constraints on pairs of nodes to support properties such as customized node ordering or downward pointing edges. The coordinates of two nodes are related by inequalities in the form of $x_i \geq x_j + gap$ for a node pair (i, j). Dwyer *et al.* [18] use gradient projection to handle these constraints, be moving nodes as little as needed to satisfy the inequalities/ equalities after each iteration of the layout method. The gradient projection method has been extended to also handle non-linear constraints [19]. These hard constraints are powerful but a bit restrictive and are different from the soft constraints in our $(SGD)^2$ framework.

Our earlier approach, $(GD)^2$ [2] optimizes several criteria already considered in the literature (e.g., stress, neighborhood preservation) as well as other criteria which had not yet been explicitly optimized in such fashion (e.g., vertex resolution, angular resolution, aspect ratio). While the full-batch gradient descent works for small graphs it does not scale well. The $(SGD)^2$ framework discussed in this paper reformulates the optimizations using stochastic gradient descent, resulting in much faster running times and making it possible to work with larger graphs.

3 The $(SGD)^2$ Framework

The $(SGD)^2$ framework is a general optimization approach to generate a layout with any desired set of aesthetic

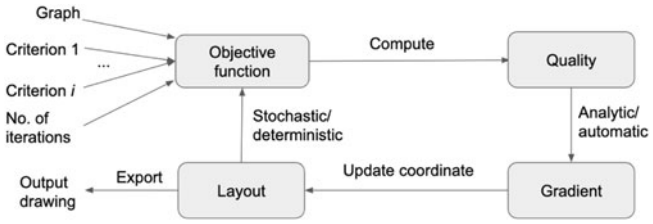

Fig. 2. The $(SGD)^2$ framework: formulate an objective function based on the given a graph and criteria set. Then compute the quality (value) of the objective function for the current layout, generate the gradient, update the coordinates of the layout, and update the objective function. This process is repeated for a fixed number of iterations.

metrics, provided that they can be expressed by a smooth function. The basic principles underlying this framework are simple. The first step is to select a set of layout readability criteria and loss functions that measure each of them. Then we define the function to optimize as a linear combination of the loss functions for each individual criterion. Finally, we iterate the gradient descent steps, from which we obtain a slightly better drawing at each iteration. Fig. 2 depicts the framework of $(SGD)^2$: Given any graph with n nodes and a readability criterion $c \in C$, we design a loss function $L_c : \mathbb{R}^{n\times 2} \to \mathbb{R}$ that maps the current layout $X \in \mathbb{R}^{n\times 2}$ to a measure $L_c(X)$ with respect to the readability criterion. Then we combine multiple loss functions from different criteria into a single one by taking a weighted sum, $L(X) = \Sigma_c w_c L_c(X)$, where a lower value is always desirable. At each iteration, a slightly better layout can be found by taking a small (η) step, often refer to as learning rate, along the (negative) gradient direction: $X^{(new)} = X - \eta \cdot \nabla L(X)$. The learning rate and all weights vary according to the current stage t of the optimization. In $(SGD)^2$, we decrease the learning rate $\eta(t)$ adaptively based on the recent changes in the loss function; see Section 3.1 and supplementary materials for details, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TVCG.2022.3155564. Given a specific criteria set C, we design a weight schedule $w_c(t)$ for each $c \in C$; see Section 5.2. Algorithm 1 summarizes the $(SGD)^2$ optimization procedure.

3.1 Gradient Descent Optimization

There are different kinds of gradient descent algorithms. The standard method considers all nodes, computes the gradient of the objective function, and updates node coordinates based on the gradient. Some objectives may consider all the nodes in every step. For example, the basic stress formulation [31] falls in this category. To compute the gradient for optimization, one has to iterate through all the nodes which makes it not scalable to very large graphs. Fortunately, most of these objectives can be decomposed into optimization over only subsets of nodes. Consider stress minimization again, if we sample a set of node pairs randomly and minimize the stress between the nodes in each pair, the stress of the whole graph is also minimized [50]. This approach is known as stochastic gradient descent (SGD) and we use this idea extensively. In Section 4, we specify the objective loss functions and sampling methods we used for each readability criterion we consider.

Not all readability criteria come naturally in the form of differentiable functions. We cannot compute the gradient of or apply SGD on non-differentiable functions. In cases that the original objective is continuous but not everywhere differentiable e.g., a 'hinge' function f(x)=max(0,x), we can compute the subgradient and update the objective based on the subgradient. Hence, as long as the function is continuously defined on a connected component in the domain, we can apply the subgradient descent algorithm.

Algorithm 1. The $(SGD)^2$ Algorithm

Input:
$G = (V, E), C = \{\dots c \dots\}$ // `graph, criteria`
$S : c \mapsto s_c$ // `sample sizes for each` c
$L_c : \mathbb{R}^{s_c\times 2} \to \mathbb{R}_+$ // `loss functions`
$W : c \mapsto w_c, \eta$ // `weights & learning rate`
q, Q_q // `safe update criterion & quality`
Output: X // `layout`
Function *Layout*$(G; C, S, W, maxiter, \eta)$:
  $X \leftarrow$ InitializeLayout(G)
  if *'crossings'* $\in C$ **then**
    $cd \leftarrow$ InitializeCrossingDetector()
  for $t = 1, \dots, maxiter$ **do**
    $l \leftarrow 0$
    for $c \in C$ s.t. $w_c(t) > 0$ **do**
      $sample \leftarrow$ Sample(c, s_c)
      if c == *'crossings'* **then**
        UpdateCrossingDetector$(cd, sample)$
        $l_c \leftarrow L_c(sample; G, cd)$
      else
        $l_c \leftarrow L_c(sample; G)$
      $l \leftarrow l + w_c(t) \cdot l_c$
    if *'Safe update' is enabled* **then**
      $X_{prev} \leftarrow X$
      $X_{new} \leftarrow X - \eta(t) \cdot \nabla_X l$
      $X \leftarrow$ SafeUpdate$(X_{prev}, X_{new}; G, Q_q)$ // `Algorithm 2`
    else
      $X \leftarrow X - \eta(t) \cdot \nabla_X l$
Return X

When a function is not defined in a connected domain, we use surrogate loss functions to 'connect the pieces'. For example, maximizing the Jaccard similarity between graph neighbors and the nearest neighbors in the layout optimizes neighborhood preservation. However, as Jaccard similarity is defined between two binary vectors, we use Lovász extension [6] to all real vectors and apply that to optimize neighborhood preservation. An essential part of gradient descent based algorithms is to compute the gradient/subgradient of the objective function. In practice, it is not necessary to write down the gradient analytically as it can be computed automatically via (reverse-mode) automatic differentiation [26]. Deep learning packages such as Tensorflow.js [43] and PyTorch [38] apply automatic differentiation to compute the gradient of complicated functions.

Most of the objective functions that we consider here are not convex and do not have unique global minimizers. Therefore, even though SGD is known to converge (to at least a local optimum) in relatively relaxed settings [3], [25], few optimization objectives are guaranteed to find the global optimum. In particular, unlike methods such as

stress majorization [24], most of our optimization objectives are not guaranteed to converge to the global optimum. Meanwhile, most of the objective functions for which SGD works well in practice (e.g., in deep learning) are neither convex nor have unique global minimizers [36]. With this in mind, we follow the common practice of applying an annealing process, if necessary, to ensure convergence (to a possibly local minimum).

When optimizing multiple criteria simultaneously, we combine them via a weighted sum. However, choosing a proper weight for each criterion can be tricky. Consider, for example, maximizing crossing angles and minimizing stress simultaneously with a fixed pair of weights. At the very early stage, the initial drawing may have many crossings and stress minimization often removes most of the early crossings. As a result, maximizing crossing angles in the early stage can be harmful as it moves nodes in directions that contradict those that come from stress minimization. Therefore, a well-tailored *weight scheduling* is needed for a successful outcome. Continuing with the same example, a better outcome can be achieved by first optimizing stress until it converges, and later adding weights for the crossing angle maximization. In order to measure the compatibility of a combination of criteria, in Section 5.1 we analyze the interactions between any pair of criteria using constant weights. We provide an interface that allows manual tuning of the weights and discuss weight schedules for different criteria sets in Section 5.2.

To encourage convergence, we decrease the SGD's learning rate when the loss does not improve after a certain number of iterations, often referred to as "patience". In machine learning frameworks such as PyTorch [38] or Keras [10], this learning rate schedule is called ReduceLROnPlateau

$$\eta(t) = ReduceLROnPlateau(L_t; factor, patience), \tag{1}$$

where L_t is the loss used to determine the timing of the learning rate deduction. To further improve the robustness of the learning schedule, we use the exponential moving average of sample loss as L_t; see the supplementary material, available online, (Section A) for the definition of L_t, factor and patience.

3.2 Implementation

The $(SGD)^2$ framework is implemented in Python and relies on PyTorch [38] for automatic differentiation, NetworkX [27] for processing graphs, and matplotlib [29] for drawing. To demonstrate our method we provide an interactive JavaScript prototype which uses the automatic differentiation tools of tensorflow.js [43] and the D3.js library [8].

4 Properties and Measures

In this section we specify the aesthetic goals, definitions, quality measures and loss functions for each of the 9 graph drawing properties we optimized: stress, ideal edge lengths, neighborhood preservation, crossing number, crossing angle, aspect ratio, angular resolution, node resolution and Gabriel graph property. In each subsection, we first define our loss function for the entire graph. For small graphs, one could apply (full-batch) gradient descent directly on this loss, which corresponds to the method we used in our early work [2]. To speed up the method for larger graphs, we sample portions of our loss functions at each iteration and apply (mini-batch) stochastic gradient descent on them. Even though full-batch gradient descent works reasonably well for small graphs, sampling still provides some speedup while maintaining comparable quality. Therefore, in $(SGD)^2$ we always use sampling, independent of graph size. The definition of a sample can be different for each criterion. For example, for stress minimization we sample pairs of nodes; for ideal edge length, we sample edges. Hence, the sample sizes of different criteria can be set independently. Moreover, for a given objective function, when the sample size for a certain criterion exceeds the number of possible samples, our method defaults to (full-batch) gradient descent. In Section 5.3, we discuss the effect of the sample sizes on the convergence rates. The analysis has helped us set the default values for each readability criterion. In general, for each criterion we sample mini-batches from a pool of all sample points (e.g., all pairs of nodes for stress, all edges for ideal edge length) without replacement, and 'refill the pool' when all sample points are drawn. In practice, we shuffle the list of data points, draw mini-batches from the list in consecutive order, and re-shuffle the list once every data point is drawn.

4.1 Stress

We minimize stress, L_{ST}, to draw a graph that matches the euclidean distances between pairs of nodes in the drawing to their graph theoretic distances. Following the original definition of stress [31], we minimize

$$L_{ST} = \sum_{i<j} w_{ij}(||X_i - X_j||_2 - d_{ij})^2. \tag{2}$$

Where d_{ij} is the graph-theoretical distance between nodes i and j, X_i and X_j are the coordinates of nodes i and j in the layout. The normalization factor $w_{ij} = d_{ij}^{-2}$ balances the influence of short and long distances: the longer the graph theoretic distance, the more tolerance we give to the discrepancy between two distances. When comparing two drawings of the same graph with respect to stress, a smaller value (lower bounded by 0) corresponds to a better drawing. To work with large graphs, we take the mean loss for any pairs of nodes to turn it into the expectation of stress

$$\hat{L}_{ST} = \mathbb{E}_{i\neq j}\,[w_{ij}(||X_i - X_j||_2 - d_{ij})^2]. \tag{3}$$

The quality measure for stress, Q_{ST}, is equal to the loss $\hat{L}_{ST}$ over all pairs of nodes. In each SGD iteration we minimize the loss by sampling a number of node pairs. Since the expectation of the gradient of the sample loss equals the true loss, we can use the gradient of the sample loss as an estimate of the true gradient and update the drawing through SGD accordingly. In each SGD iteration, we sample m pairs of nodes. By default, we set $m = 32$ based on our experiments with different sample sizes in Section 5.3. Before a round that goes over all pairs of nodes, we shuffle a list of node-pairs and take mini-batches from the shuffled list. This guarantees that we process every pair of nodes exactly once per round.

4.2 Ideal Edge Length

Given a set of ideal edge lengths $\{l_{ij} : (i,j) \in E\}$ we minimize the variance from the ideal lengths

$$L_{IL} = \sum_{(i,j)\in E} \left(\frac{||X_i - X_j|| - l_{ij}}{l_{ij}} \right)^2. \tag{4}$$

For unweighted graphs, by default we use 1 as the ideal edge length for all edges $(i,j) \in E$. As with stress minimization, for large graphs we replace the summation by the expectation and estimate it through sampling the edges.

$$\hat{L}_{IL} = \mathbb{E}_{(i,j)\in E}\left[\left(\frac{||X_i - X_j|| - l_{ij}}{l_{ij}} \right)^2 \right]. \tag{5}$$

The quality measure $Q_{IL} = \hat{L}_{IL}$ is lower bounded by 0 and a lower score yields a better layout. Similar to the sampling strategy for stress, here we keep a list of all edges in random order, draw mini-batches (by default, of size $m = 32$) from it, and re-shuffle the list after all edges are processed once.

4.3 Neighborhood Preservation

Neighborhood preservation aims to keep adjacent nodes close to each other in the layout. Similar to Kruiger *et al.* [34], the idea is to have the k-nearest (euclidean) neighbors (k-NN) of node i in the drawing to match the k nearest nodes (in terms of graph distance from i). Here we set k to be the degree of node. A natural quality measure for the match is the Jaccard index between the two pieces of information. Let, $Q_{NP} = JaccardIndex(K, Adj) = \frac{|\{(i,j):K_{ij}=1 \text{ and } A_{ij}=1\}|}{|\{(i,j):K_{ij}=1 \text{ or } A_{ij}=1\}|}$, where Adj is the adjacency matrix and the ith row in K denotes the k-nearest neighborhood information of i: $K_{ij} = 1$ if j is one of the k-nearest neighbors of i and 0 otherwise. To express the Jaccard index as a differentiable minimization problem, we first express the neighborhood information in the drawing as a smooth function of node positions X_i and store it in a matrix $\hat{K}$. In $\hat{K}$, a positive entry $\hat{K}_{i,j}$ means node j is one of the k-nearest neighbors of i, otherwise the entry is negative. Next, we take a differentiable surrogate function of the Jaccard index, the Lovász hinge loss (LHL) [6], to make the Jaccard loss optimizable via gradient descent. We minimize

$$L_{NP} = LHL(\hat{K}, Adj), \tag{6}$$

where $\hat{K}$ denotes the k-nearest neighbor estimation. For simplicity, let $d_{i,j} = ||X_i - X_j||$ denote the euclidean distance between node i and j, then we design $\hat{K}$ as

$$\hat{K}_{i,j} = \begin{cases} -(d_{i,j} - \frac{d_{i,\pi_k} + d_{i,\pi_{k+1}}}{2}) & \text{if } i \neq j \\ 0 & \text{if } i = j \end{cases}, \tag{7}$$

where π_k denotes the k^{th} nearest neighbor of node i. That is, for every node i, we treat the average distance to its k^{th} and $(k+1)^{th}$ nearest neighbor as a threshold, and use it to measure whether node j is in the neighbor or not. Note that $d_{i,j}$, d_{i,π_k} and $d_{i,\pi_{k+1}}$ are all smooth functions of node positions in the layout, so $\hat{K}_{i,j}$ is also a smooth function of node positions X. $\hat{K}_{i,j}$ is positive if node j is a k-NN of node i, otherwise it is negative, as is required by LHL [6].

We then sample nodes for stochastic gradient descent. Note that the nearest neighbors π_k and π_{k+1} in $\hat{K}_{i,j}$ depend on distances from all nodes. To derive a reliable estimate of the Jaccard index, instead of letting k equal to the degree of node i in the full graph, we let k equal to the degree of i in the subgraph that we sample. In other words, in every gradient descent iteration we sample a subgraph from the full graph and compute LHL of the subgraph. In practice, we randomly select a small set of m nodes (by default, $m = 16$), along with nodes that are 1 or 2 hops away from any of them. We also include a fraction of nodes that are not already in the sample. We extract the subgraph induced by this set of nodes and apply stochastic gradient descent.

4.4 Crossing Number

Reducing the number of edge crossings is one of the classic optimization goals in graph drawing, known to affect readability [39]. Shabbeer *et al.* [42], employed an expectation-maximization-like algorithm. Since two edges do not cross if and only if there exists a line that separates their extreme points, they trained support vector machine (SVM) classifiers to separate crossing pairs and use the classifiers as a guide to eliminate crossings. Since one has to train as many SVM classifiers as the number of crossings in the graph and knowledge learned by one SVM does not naturally transfer to another, this approach does not work well for large graphs. With this in mind we modified our initial approach to that of Tiezzi *et al.*[45], which uses Graph Neural Networks to reduce the number of crossings in two steps. First, a generic neural network is trained to predict if any two edges cross. Since neural networks are differentiable, the well-trained edge crossing predictor from this step serves as a guide to gradient descent steps later on. Second, a Graph Neural Network is trained to use the edge crossing predictor as a guide to improve the layout. Our method uses only the first step above and utilizes a different training strategy. Instead of training the edge crossing predictor using a synthetic dataset before the layout optimization, we train the crossing predictor directly on the current graph layout while simultaneously updating the node coordinates, using the crossing predictor as a guide. Formally, let f_β denote a neural network with trainable parameters β that takes the coordinates of the four nodes of any two edges $X^{(i)} \in \mathbb{R}^{4\times 2}$ and outputs a scalar from the (0,1) interval. An output close to 0 means "no crossing" and one close to 1 means "crossing". In practice, f_β is a simple multi-layer perceptron (MLP) with batch normalization [30] and LeakyReLU activation. To train a neural crossing detector f_β, we feed different edge pairs $X^{(i)} \in \mathbb{R}^{4\times 2}$ to approximate the ground truth $t^{(i)} \in \{0,1\}$ where 0 means "no crossing" and 1 means "crossing". We optimize the parameters β to minimize the cross entropy (CE) loss L_β between the prediction $f_\beta(X^{(i)})$ and the ground truth $t^{(i)}$, averaging over a sample of n instances of edge pairs

$$L_\beta(\beta; X^{(1)} \ldots X^{(n)}) = \frac{1}{n}\sum_{i=1}^{n} CE(f_\beta(X^{(i)}), t^{(i)}),$$

where

$$CE(y,t) := -t \cdot log(y) - (1-t) \cdot log(1-y). \tag{8}$$

We use the neural crossing detector f_β to construct a differentiable surrogate loss function for crossing minimization. Specifically, given a well-trained f_β, we can reduce the number of crossings in a layout by minimizing the cross entropy between the prediction of edge pairs $f_\beta(X^{(i)})$ and the desired target (i.e., no crossing $t = 0$)

$$L_{CR}(X;\beta) = \frac{1}{n}\sum_{i=1}^{n} CE(f_\beta(X^{(i)}), 0). \tag{9}$$

In practice, we minimize L_β and L_{CR} simultaneously in each $(SGD)^2$ iteration. We first improve the neural crossing predictor using a sample of edge pairs from the graph. For simplicity, we describe the training by SGD, although in practice one can utilize any SGD variants (e.g., SGD with momentum, ADAM [33] or RMSProp [44]) to train the predictor more efficiently.

$$\beta^{(new)} = \beta - \epsilon' \cdot \nabla L_\beta. \tag{10}$$

In the meantime we update the layout in a similar manner

$$X^{(new)} = X - \epsilon \cdot \nabla L_{CR}. \tag{11}$$

Although one could improve the neural crossing predictor by multiple steps in every $(SGD)^2$ iteration, we found little difference when varying the number of steps. Therefore, we only take one step to improve the neural crossing predictor in every $(SGD)^2$ iteration. As with other criteria, we randomly draw mini-batches (by default, of size $m = 128$) and iterate through all edge pairs over the course of the SGD iterations.

When a graph layout does not have many crossings (e.g., a stress-minimized layout of a near-planar graph), this sampling strategy is not efficient. In that case, we use an efficient algorithm (Bentley-Ottmann [5]) to find all crossing edges in a graph, and sample a mini-batch of crossings. Since finding all crossings can be slow for large graphs, we only do this once every few iterations and reuse the finding across a few iterations. Specifically, when we draw mini-batches from the pool of all crossings, we recompute all crossings again once the pool is drained. To evaluate the quality we simply count the number of crossings.

4.5 Crossing Angle Maximization

When edge crossings are unavoidable, the graph drawing can still be easier to read when edges cross at angles close to 90 degrees [48]. Heuristics such as those by Demel *et al.* [13] and Bekos *et al.* [4] have been proposed and have been successful in graph drawing challenges [14]. We use an approach similar to the force-directed algorithm given by Eades *et al.* [21] and minimize the squared cosine of crossing angles

$$L_{CAM} = \sum_{\text{all crossed edge pairs } (i,j),(k,l)\in E} \left(\frac{\langle X_i - X_j, X_k - X_l\rangle}{|X_i - X_j| \cdot |X_k - X_l|}\right)^2. \tag{12}$$

We evaluate quality by measuring the worst (normalized) absolute discrepancy between each crossing angle θ and the target crossing angle (i.e., 90 degrees): $Q_{CAM} = \max_\theta |\theta - \frac{\pi}{2}| / \frac{\pi}{2}$. As with crossing numbers, for large graphs we sample a subset (by default, of size $m = 16$) of edge pairs and consider their crossing angles if the edge pair cross each other. Again, if there are not many crossing pairs we use an efficient algorithm to find all crossings. When optimizing the number of crossings and crossing angles simultaneously, we sample from the same pool of crossings formed via the Bentley-Ottmann algorithm.

4.6 Aspect Ratio

Good use of drawing area is often measured by the aspect ratio [16] of the bounding box of the drawing, with 1:1 as the optimum. The idea here is to consider different rotations of the current layout and try to "squarify" the corresponding bounding boxes. In practice, we rely on the singular values of the matrix of node coordinates to approximate the worst aspect ratio. Formally, assume vertex coordinates are centered with zero mean and let X denote the collection of (centered) vertex coordinates as rows in a matrix. Since the coordinates are two dimensional, X has only two (non-zero) singular values, denoted by σ_1 and σ_2 and each measures the standard deviation of the layout along with two orthogonal directions. Then we approximate the aspect ratio using the quotient of the two singular values of X and encourage the ratio to be close to the target ratio $r = 1$ using the cross entropy (CE) in Eq. (8)

$$L_{AR} = CE\Big(\frac{\sigma_2}{\sigma_1}, r\Big).$$

Note that although we only consider 1:1 ratios, the formulation of cross entropy let us consider arbitrary ratios. During mini-batch SGD, we simply sample a subset of nodes (by default, of size $m = 128$) and use the singular values of the matrix formed by the subset to optimize the aspect ratio.

Finally, we evaluate the drawing quality by measuring the worst aspect ratio on a finite set of rotations. The quality score ranges from 0 to 1. In our case, 1 is optimal and the minimal ratio among different rotations is the worst: $Q_{AR} = \min_{\theta\in\{\frac{2\pi k}{N}, \text{ for } k=0,\ldots(N-1)\}} \frac{\min(w_\theta, h_\theta)}{\max(w_\theta, h_\theta)}$, where N is the number of rotations sampled (e.g., $N = 7$), and w_θ, h_θ are the width and height of the bounding box when rotating the drawing around its center by an angle θ.

4.7 Angular Resolution

Distributing edges adjacent to a node makes it easier to perceive the information presented in a node-link diagram [28]. Angular resolution [28], defined as the minimum angle between incident edges, is one way to quantify this goal. Formally, $ANR = \min_{j\in V} \min_{(i,j),(j,k)\in E} \varphi_{ijk}$, where φ_{ijk} is the angle formed by between edges (i, j) and (j, k). Note that for any given graph, an upper bound of this quantity is $\frac{2\pi}{d_{max}}$ where d_{max} is the maximum degree of nodes in the graph. We use this upper bound to normalize the quality measure to [0,1], i.e., $Q_{ANR} = \frac{ANR}{2\pi/d_{max}}$. To achieve a better drawing quality via gradient descent, we define the angular energy of an angle φ to be $e^{-s\cdot\varphi}$, where s is a constant controlling the sensitivity of angular energy with respect to the angle (by default $s = 1$), and minimize the total angular energy over all incident edges

$$L_{ANR} = \sum_{(i,j),(j,k)\in E} e^{-s\cdot\varphi_{ijk}}. \tag{13}$$

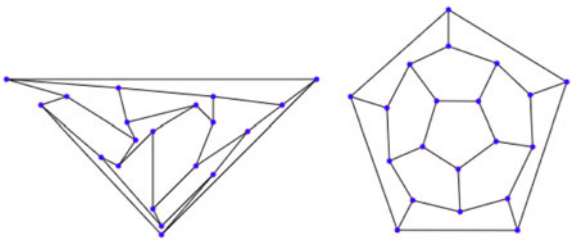

Fig. 3. Optimizing Planar Graphs: (a) An initial layout without crossings, (b) A layout after optimizing stress while maintaining planarity.

When the graph is large, it is expensive to compute the energy of all pairs of incident edges. Therefore, in $(SGD)^2$ we randomly sample a minibatch of pairs of incident edges (by default, of size $m = 128$) and minimize the energy of the sample accordingly.

4.8 Node Resolution

Good node resolution is associated with the ability to distinguish different nodes by preventing nodes from occluding each other. Node resolution is typically defined as the minimum euclidean distance between two nodes in the drawing [41]. However, in order to align with the units in other objectives such as stress, we normalize the minimum euclidean distance with respect to a reference value. Hence we define the node resolution to be the ratio between the shortest and longest distances between pairs of nodes in the drawing, $VR = \frac{\min_{i \neq j} ||X_i - X_j||}{r \cdot d_{max}}$, where $d_{max} = \max_{k,l} ||X_k - X_l||$. To achieve a certain target resolution $r \in [0, 1]$ by minimizing a loss function, we minimize

$$L_{VR} = \sum_{i,j \in V, i \neq j} max\left(0, \left(1 - \frac{||X_i - X_j||}{r \cdot d_{max}}\right)^2\right). \tag{14}$$

In practice, we set the target resolution to be $r = \frac{1}{\sqrt{|V|}}$, where $|V|$ is the number of nodes in the graph. In this way, an optimal drawing will distribute nodes uniformly in the drawing area. Each term in the summation vanishes when the distance between two nodes meets the required resolution r, otherwise it is greater than zero. In the evaluation, we report, as a quality measure, the ratio between the actual and target resolution and cap its value between 0 (worst) and 1 (best). $Q_{VR} = \min(1.0, \frac{\min_{i,j} ||X_i - X_j||}{r \cdot d_{max}})$

For large graphs, we sample a subset of nodes (by default, of size $m = 256$) and compute the approximate loss of node resolution on the sample.

4.9 Gabriel Graph Property

A graph is a Gabriel graph if it can be drawn in such a way that any disk formed by using an edge in the graph as its diameter contains no other nodes. Not all graphs are Gabriel graphs, but drawing a graph so that as many of these edge-based disks are empty of other nodes has been associated with good readability [20]. This property can be enforced by a repulsive force around the midpoints of edges. Formally, we establish a repulsive field with radius r_{ij} equal to half of the edge length, around the midpoint c_{ij} of each edge $(i, j) \in E$, and we minimize the total potential energy

$$L_{GA} = \sum_{(i,j) \in E, k \in V \setminus \{i,j\}} max(0, r_{ij} - |X_k - c_{ij}|)^2, \tag{15}$$

where $c_{ij} = \frac{X_i + X_j}{2}$ and $r_{ij} = \frac{|X_i - X_j|}{2}$. We use the (normalized) minimum distance from nodes to centers to characterize the quality of a drawing with respect to Gabriel graph property: $Q_{GA} = \min(1, \min_{(i,j) \in E, k \in V} \frac{|X_k - c_{ij}|}{r_{ij}})$.

For large graphs, we sample a mini-batch of node-edge pairs (by default, of size $m = 64$) and compute the approximate loss from the sample.

Algorithm 2. Update Coordinates Without Quality Decline

Function *SafeUpdate*($X_{prev}, X_{new}; G, Q_q$):
  $X \leftarrow X_{prev}$
  $q_0 \leftarrow Q_q(X; G)$
  for *each node* $u \in V$ **do**
    $X[u] \leftarrow X_{new}[u]$
    $q_{tmp} \leftarrow Q_q(X)$
    if *QualityDeclines*(q_0, q_{tmp}) **then**
      $X[u] \leftarrow X_{prev}[u]$
  return X

4.10 Optimizing Layouts Without Quality Decline

Many optimization criteria (e.g., stress minimization) tend to reduce edge crossings but cannot guarantee a crossing-free drawing even when the graph is planar. One of the reasons for the popularity of stress-based layout methods is that they capture the underlying graph topology well. On the other hand, algorithms that are guaranteed to produce planar drawing (for planar graphs) are well known to distort the graph topology.

Our crossing minimization optimization is a soft constraint and does not guarantee a planar drawing when graph is planar. Hence, we provide an extra feature in our system that if we start with a planar drawing we can optimize any of the criteria above, while guaranteeing that no edge crossing will arise at any time. To do this, we add one additional test for every gradient descent step: for each proposed coordinate, we first check whether moving a node to this coordinate will introduce a crossing and if so, we do not update the coordinate. Using this method starting with an initial planar drawing, we can improve it, and provide a layout that is also planar and preserves the topology; see Fig. 3. This technique can be useful in other force-directed algorithms that do not guarantee crossing-free drawing when the initial layout is without crossings [23].

We can generalize this idea for any graphs while optimizing any criterion. In the above scenario, we maintained the number of crossings of the layouts equal to zero since we started with a planar layout. If the graph is non-planar then we can update the coordinates in a similar way and the number of crossings in the progressive layouts will be decreasing. Similarly, we can consider other criteria, for example, the edge uniformity loss to decrease in the progressive layouts. If we are optimizing another criterion, for example, stress, there is no guarantee that edge uniformity will improve; see Fig. 4. The general algorithm for safely update the layout with respect to any quality measure is described in Algorithm 2. Note that Algorithm 2 is applied in each SGD iteration of Algorithm 1. Furthermore, it goes

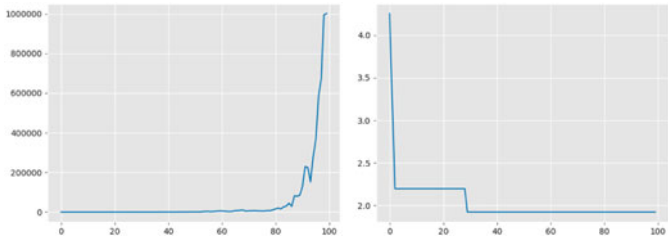

Fig. 4. (a) The edge uniformity loss is increasing when we optimize the stress of a nested triangular graph, (b) The loss is decreasing when we update the coordinates carefully.

through all nodes in the graph and the quality measure is evaluated every time an intermediate layout is generated by a single node update. Hence, this approach does not scale well to large graphs when the quality measure requires high computational overhead.

5 Experimental Evaluation

In this section, we assess the effectiveness and limitations of our approach. Since multiple criteria are not necessarily compatible with each other during optimization, we first identify all compatible pairs of criteria. After identifying all compatible pairs, we hand-craft weight schedules to optimize multiple criteria using $(SGD)^2$ and compare our layouts from multicriteria optimization with classic drawing algorithms. Finally, we analyze the runtime behavior and the impact of sample size in our approach.

5.1 Interactions Between Criteria

We test the interactions between every pair of criteria using two regular graphs, a 6x10 grid (60 nodes) and a balanced binary tree with depth 5 (63 nodes). Before dealing with pairs of criteria, we first test every single criterion to establish a lower bound of the quality measure. In this section, we invert some quality measures (e.g., neighborhood preservation, and angular resolution) such that lower values are always better in all quality measures. Then, we optimize every pair of criteria, monitor the quality measures of the pair over the course of training, and compare them with the corresponding lower bound found when optimizing each single criterion.

As expected, we observe that all but one criterion, when optimized on their own, improve or maintain high quality during improvement iterations. The exception is crossing angle maximization, with a quality measure that depends on the worst crossing in the graph. The initial random layout usually has many crossings and maximizing crossing angles on its own (e.g., without also minimizing the number of crossings) does not necessarily lead to high-quality results. Later we will see that optimizing other criteria together with crossing angle maximization helps. Further, when weight factors can be adjusted with a schedule, we recommend assigning positive weight to crossing angle maximization only at the later iterations.

When optimizing pairs of aesthetic criteria, we see three types of pairs: compatible pairs, better pairs and worse pairs. Fig. 5 shows an example for each of the three cases. Most pairs are compatible pairs. For example, stress minimization is compatible with most other drawing aesthetics, as the qualities of both optimization goals can improve over time and they both achieve their lower bounds. Some pairs of criteria even do better together than alone. For example, crossing angle maximization together with stress minimization leads to better results than just crossing angle maximization, confirming the results of Huang *et al.* [28]. A few pairs of criteria are not fully compatible, leading to worse joint optimization. For example, when simultaneously optimizing vertex resolution and angular resolution, neither value can reach their corresponding lower bound.

Out of all 36 pairs, we find 20 compatible pairs, 9 better pairs and 7 worse pairs for the 6×10 grid; for the binary tree with depth 5, we find 13 compatible pairs, 9 better pairs and 14 worse pairs. Comparing the compatibility between the two graphs, we note that all worse pairs in the grid are also worse pairs in the tree, and most of the better pairs and compatible pairs are shared between two graphs. See the supplementary materials, available online, for the drawings and quality curves of all criteria pairs and singletons. It is worth noting that the compatibility of criteria also depends on the specific choice of weight factors. For example, having a dominating criterion by assigning a large weight to it can

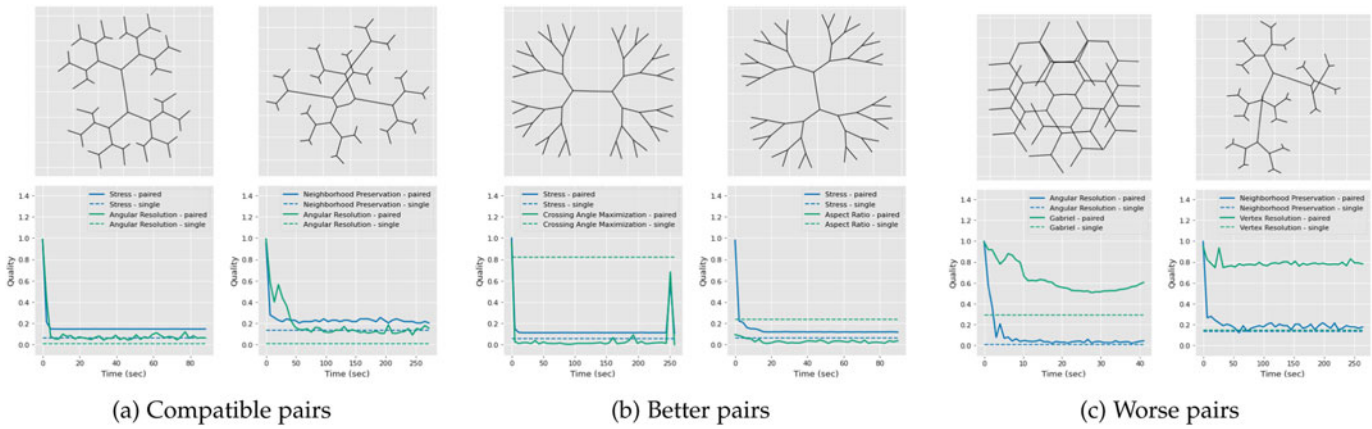

Fig. 5. We observed three types of interactions between criteria pairs: (a) compatible pairs can be optimized together; (b) better pairs do even better together than alone; (c) worse pairs are not fully compatible with each other. In the second row, stress is normalized to [0,1] based on its maximum value in each chart; angular resolution and vertex resolution are inverted $Q \mapsto 1 - Q$ so that for all criteria smaller values are always better and the worst value is always 1.

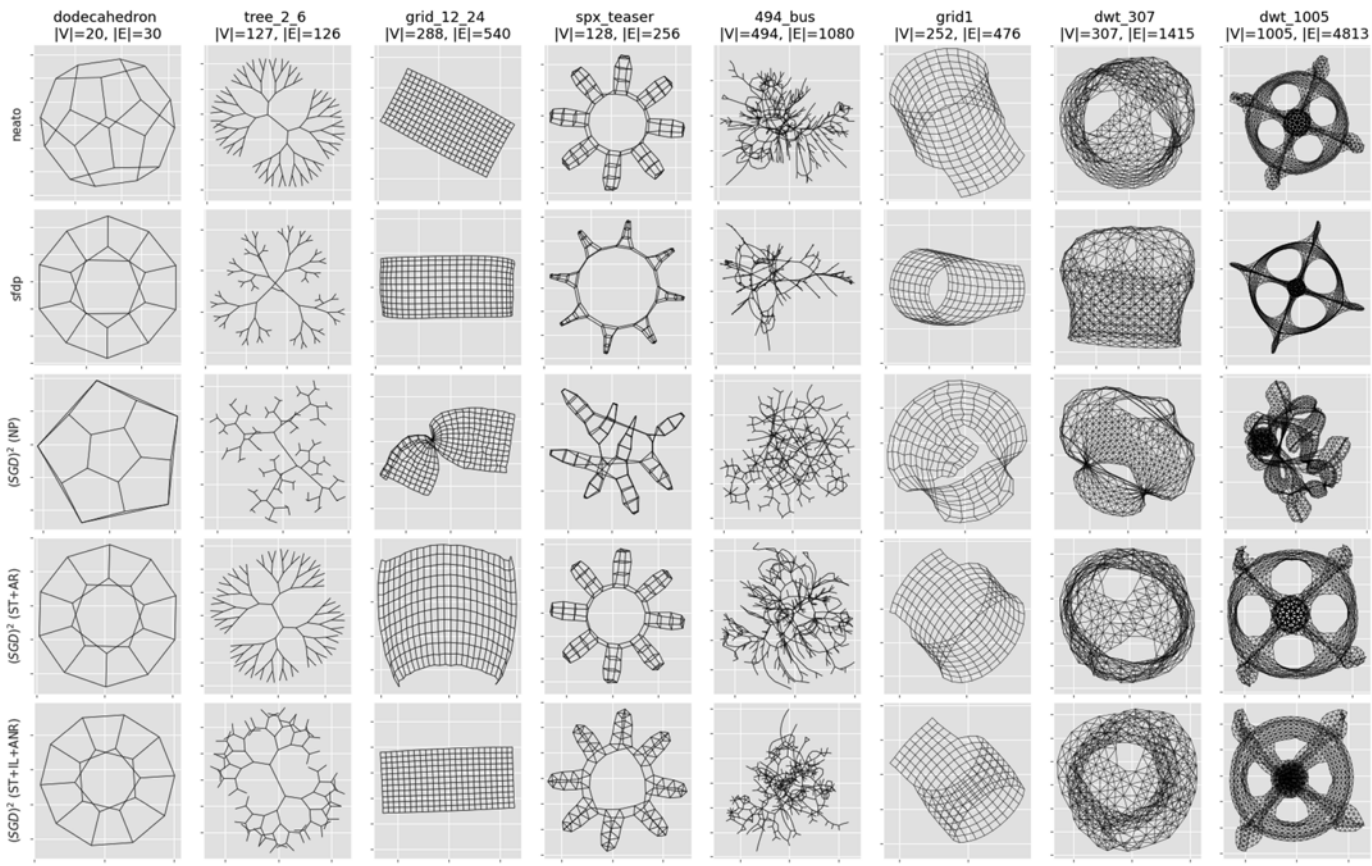

Fig. 6. Distinctive layouts of 8 graphs obtained with neato, sfdp and $(SGD)^2$ optimizing selected combinations of the 9 criteria: stress (ST), ideal edge length (IL), neighborhood preservation (NP), crossings (CR), crossing angle maximization (CAM), aspect ratio (AR), angular resolution (ANR), vertex resolution (VR), and Gabriel (GB). The supplementary materials, available online, include more layouts for more such combinations.

deteriorate the quality of the other. In this analysis, we assign a fixed weight factor (and sample size) to each criterion so that every pair yields a reasonable outcome.

5.2 Quality Analysis

We compare layouts obtained with $(SGD)^2$ when optimizing different aesthetic goals to layouts obtained by neato [22] and sfdp [22], which are classic implementations of stress-majorization and scalable force-directed methods. Fig. 6 shows the layouts along with information about each graph. The graphs are chosen to represent a variety of classes such as trees, grids, regular shapes, and to also include real-world examples. In particular, the last four graphs in Fig. 6 are from the Sparse Matrix Collection [12] and are also used to evaluate stress minimization via SGD in [50]; see the supplementary materials for more layouts, available online.

Next, we evaluate each layout on 9 readability criteria: stress (ST), node resolution (VR), ideal edge lengths (IL), neighbor preservation (NP), crossings (CR), crossing angle (CA), angular resolution (ANR), aspect ratio (AR), and Gabriel graph property (GB). Our experiment utilizes 8 graphs and layouts computed by neato, sfdp, and 7 runs of $(SGD)^2$ using various combinations of objectives. Tables 1 and 2 summarize 2 of the 9 quality measures for the layouts in Fig. 6. More combinations of criteria used for $(SGD)^2$ and the remaining quality measures are included in the supplementary materials, available online. The quality measure for crossings is the actual number of edge crossings in the layout. For all other criteria, we use the formulas defined in Section 4. All quality measures produce values greater than or equal to zero: the lower the value the better the measure. In each column, the best score is bold. When optimizing via multicriteria $(SGD)^2$, we choose compatible pairs, better pairs, or compatible triples among the 9 criteria. When optimizing incompatible pairs or triples, we fix the number of iterations in $(SGD)^2$, select and prioritize one criterion (or compatible pair) in an early stage of the training and postpone the others to the later stage. For example, when simultaneously optimizing ideal edge length (IL), neighborhood preservation (NP) and vertex resolution (VR), we assign zero weight to VR and positive weights to IL and NP in the first half of the iterations. Then we gradually decrease the weights of IL and NP to 0 (by a smooth function that interpolates the highest and lowest weights) and increase the weight of VR in the second half of the iterations with a similar smooth growth function. At each stage, we interpolate the two weight levels of each criterion w_{start} and w_{stop} between the start and stopping iterations t_{start} to t_{stop} by a scaled and translated smooth-step function $w(t)$

$$w(t) = (w_{stop} - w_{start}) \cdot f\left(\frac{t - t_{start}}{t_{stop} - t_{start}}\right) + w_{start}, \tag{16}$$

where $f(x) = 3x^2 - 2x^3$ for $x \in [0, 1]$ is typically called the (standard) smooth-step function.

The experimental results confirm that $(SGD)^2$ yields better or comparable results for most quality measures and on

TABLE 1
Quality Measures of Neighborhood Preservation

methods \ graphs	dodecahedron	tree-2-6	grid-12-24	spx-teaser	494-bus	grid1	dwt-307	dwt-1005
neato	0.723	0.718	**0.000**	**0.474**	0.846	0.558	0.699	0.545
sfdp	0.571	0.592	0.063	0.533	0.750	0.651	0.584	0.653
$(SGD)^2$ (NP)	**0.400**	**0.225**	0.276	0.487	**0.659**	**0.480**	**0.428**	**0.516**
$(SGD)^2$ (ST+AR)	0.500	0.727	0.418	0.492	0.842	0.560	0.757	0.584
$(SGD)^2$ (ST+IL+ANR)	0.500	0.749	**0.000**	0.539	0.823	0.622	0.705	0.520

TABLE 2
Quality Measures of Aspect Ratio

methods \ graphs	dodecahedron	tree-2-6	grid-12-24	spx-teaser	494-bus	grid1	dwt-307	dwt-1005
neato	0.062	0.145	0.483	0.010	0.143	0.314	0.065	**0.012**
sfdp	0.068	0.084	0.536	0.010	0.192	0.452	0.095	0.018
$(SGD)^2$ (NP)	**0.047**	0.124	0.481	0.176	0.162	**0.049**	0.109	0.282
$(SGD)^2$ (ST+AR)	**0.047**	**0.017**	**0.048**	**0.008**	**0.118**	0.154	**0.043**	0.057
$(SGD)^2$ (ST+IL+ANR)	0.048	0.197	0.508	0.045	0.178	0.269	0.058	0.084

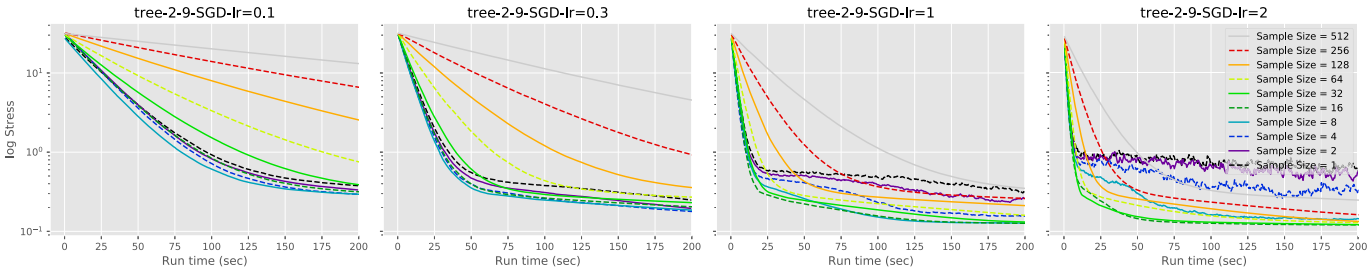

Fig. 7. Optimal sample size for stress minimization depends on the learning rate. (*two figures on the left*) a smaller sample size (e.g., 4 or 8) gives faster convergence when the learning rate is relatively small; (*two figures on the right*) a medium sample size (e.g., 16 or 32) benefits from larger learning rate when training with smaller sample sizes becomes less stable and may give even faster convergence rate than using smaller sample size in the low-learning-rate cases.

most graphs. We do note that some criteria (e.g., CR and GB) are harder to optimize on real-world large graphs; improving the performance on such tough criteria is natural direction for future work.

5.3 Analysis of Sample Size

Here we analyze the impact of sample size on the convergence rate of $(SGD)^2$. In deep learning, models trained with different sample sizes can converge to different types of minima; e.g., smaller samples tend to lead to a better generalization[32]. In $(SGD)^2$, smaller samples usually results in faster run time *per iteration* but does not necessarily yield faster *per-second* convergence. As described in Section 4, we use different sampling strategies and sample sizes for each readability criterion. Consider, for example, stress minimization and how optimal sample size closely depends on other factors in the optimization. In other words, there is no "one size fits all" sample size. In particular, for any given graph, the optimal sample size depends on the learning rate of the SGD algorithm. Fig. 7 shows the quality (i.e., stress) of layouts for a binary tree with 9 levels (1,023 nodes) as a function of total run time of the $(SGD)^2$ algorithm. In each plot, we visualize the convergence of the algorithm under a fixed learning rate with various sample sizes. When the learning rate is small we observe that smaller sample sizes (e.g., 4 or 8) converge faster, while medium sample sizes (e.g., 16 or 32) can benefit from larger learning rates and converge faster than any cases that use a smaller learning rate. Moreover, when using a large learning rate, the training with a smaller sample size becomes less stable (due to the high variance of gradients). We illustrate this observation on the binary tree with 9 levels (1,023 nodes). This impact of the interplay between sample size and learning rate on convergence rate is less obvious in variants of the SGD algorithm. When replacing SGD with some of its variants (e.g., AdaDelta [49], RMSProp [44] or ADAM [33]) that takes adaptive step size based on the gradient of previous steps, the convergence rate of stress minimization becomes less sensitive to sample size or learning rate.

5.4 Analysis of Run Time

To test the scalability of our method, we test the runtime of our method on larger graphs. We tested our code on a MacBook Pro with a 2.9 GHz Dual-Core Intel Core i5 CPU and 16GB of memory. We picked two families of graphs: balanced binary trees and 2D grids, and measured the convergence time as the size of the graph grows. For balanced binary trees, we start with a tree with 4 levels (15 nodes) and gradually increase the depth to 12 levels (4095 nodes). For grids, we start with a grid of size 16×2 (32 nodes) and

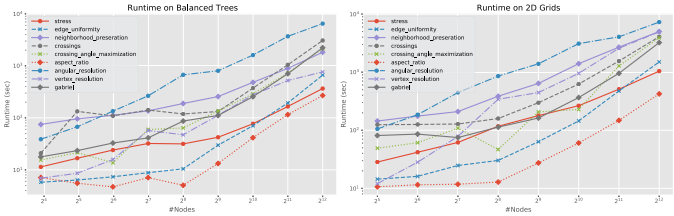

Fig. 8. Runtime of balanced trees (top) and 2D grids (bottom). The plots have log scales on both x and y axes.

double the number of columns until we have a grid of size 16×256 (4096 nodes). For each criterion, we randomly initialize nodes in the layout from standard Gaussian, optimize the layout with respect to only one criterion using SGD and stop as soon as the layout converges.

Fig. 8 summarizes the runtime analysis for the two families of graphs (trees and grids) for all 9 criteria. Note that we are using log-log plots (log scales for both the x and y axes). This experimental analysis shows linear or near-linear time for the underlying algorithms. This is shown as steeper slopes in the log-log plots.

6 Conclusion, Limitations, Future Work

We introduced the graph drawing framework, $(SGD)^2$, for multicriteria graph drawing and showed how this approach can be used to optimize different graph drawing criteria and combinations thereof. We showed that multiple readability criteria can be optimized jointly via SGD if each of them can be expressed as a differentiable function. In cases that some readability criteria are not naturally differentiable (e.g., neighborhood preservation or crossing number), one can find differentiable surrogate functions and optimize the criteria indirectly. We measured the quality of generated layouts and analyzed interactions between criteria, the runtime behavior, and the impact of sample sizes; all of which provide evidence of the effectiveness of $(SGD)^2$.

Support for More Constraint Types. Although $(SGD)^2$ is a flexible framework that optimizes a wide range of criteria, we did not consider constraints where node coordinates are related by some inequalities (i.e., hard constraints). Similarly, in the $(SGD)^2$ framework we do not naturally support shape-based drawing constraints such as those in [17], [46]. Incorporating a wider range of constraint types and studying the interactions between them in the multi-objective setting are natural directions for future work.

Weight Balancing for Multicriteria Objectives. The $(SGD)^2$ framework is flexible and natural directions for future work include adding further drawing criteria and better ways to combine them. An appropriate balance between weights for the different criteria can be crucial as more and more criteria are incorporated into the optimization. Currently, we manually choose appropriate weight schedules based on specific combinations of criteria. In the future, we would like to explore ways to automatically design and balance weight schedules in multicriteria graph drawing.

Applications of Different Techniques and Frameworks. Besides gradient descent, there are other optimization techniques that could be deployed to multi-objective problems [37]. Similarly, while we used Tensorflow.js and PyTorch to implement $(SGD)^2$, there are other frameworks (e.g., pymoo [7]) with support for multi-objective optimization. The application of different optimization techniques and frameworks to multicriteria network visualization seems like an interesting direction for future work.

Scalability for Larger Graphs. Currently, not all criteria are fully optimized for speed. Alternative objective functions, for example tsNET by Kruiger *et al.* [34] for neighborhood preservation, could be considered in the $(SGD)^2$ framework as further runtime scalability and quality improvements are needed for graphs with millions of nodes and edges. One possible direction for improving scalability is to employ a multi-level algorithmic framework.

Acknowledgments

This is an extended version of the work that appeared in GD'20 [2].

References

[1] B. M. Ábrego, S. Fernández-Merchant, and G. Salazar, "The rectilinear crossing number of k_n: Closing in (or are we?)," in *Proc. 30th Essays Geometric Graph Theory*, 2013, pp. 5–18.
[2] R. Ahmed, F. D. Luca, S. Devkota, S. Kobourov, and M. Li, "Graph drawing via gradient descent, $(GD)^2$," in *Proc. 28th Int. Symp. Graph Drawing Netw. Vis.*, 2020, pp. 3–17.
[3] R. Bassily, M. Belkin, and S. Ma, "On exponential convergence of SGD in non-convex over-parametrized learning," 2018, *arXiv: 1811.02564*.
[4] M. A. Bekos *et al.*, "A heuristic approach towards drawings of graphs with high crossing resolution," in *Proc. 26th Int. Symp. Graph Drawing Netw. Vis.*, 2018, pp. 271–285.
[5] J. L. Bentley and T. A. Ottmann, "Algorithms for reporting and counting geometric intersections," *IEEE Comput. Archit. Lett.*, vol. 28, no. 09, pp. 643–647, Sep. 1979.
[6] M. Berman, A. Rannen Triki, and M. B. Blaschko, "The lovász-softmax loss: A tractable surrogate for the optimization of the intersection-over-union measure in neural networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 4413–4421.
[7] J. Blank and K. Deb, "Pymoo: Multi-objective optimization in python," *IEEE Access*, vol. 8, pp. 89 497–89 509, 2020.
[8] M. Bostock, V. Ogievetsky, and J. Heer, "D3: Data-driven documents," *IEEE Trans. Vis. Comput. Graphics*, vol. 17, no. 12, pp. 2301–2309, Dec. 2011.
[9] C. Buchheim, M. Chimani, C. Gutwenger, M. Jünger, and P. Mutzel, "Crossings and planarization," in *Handbook of Graph Drawing and Visualization*, Boca Raton, FL, USA: CRC Press, 2013, pp. 43–85.
[10] F. Chollet *et al.*, "Keras," 2015. [Online]. Available: https://keras.io
[11] R. Davidson and D. Harel, "Drawing graphs nicely using simulated annealing," *ACM Trans. Graph.*, vol. 15, no. 4, pp. 301–331, 1996.
[12] T. A. Davis and Y. Hu, "The University of Florida sparse matrix collection," *ACM Trans. Math. Softw.*, vol. 38, no. 1, pp. 1–25, 2011.
[13] A. Demel, D. Dürrschnabel, T. Mchedlidze, M. Radermacher, and L. Wulf, "A greedy heuristic for crossing-angle maximization," in *Proc. 26th Int. Symp. Graph Drawing Netw. Vis.*, 2018, pp. 286–299.
[14] W. Devanny, P. Kindermann, M. Löffler, and I. Rutter, "Graph drawing contest report," in *Proc. 25th Int. Symp. Graph Drawing Netw. Vis.*, 2017, pp. 575–582.
[15] S. Devkota, R. Ahmed, F. De Luca, K. E. Isaacs, and S. Kobourov, "Stress-plus-x (SPX) graph layout," in *Proc. 27th Int. Symp. Graph Drawing Netw. Vis.*, 2019, pp. 291–304.
[16] C. A. Duncan, M. T. Goodrich, and S. G. Kobourov, "Balanced aspect ratio trees and their use for drawing very large graphs," in *Proc. 6th Int. Symp. Graph Drawing*, 1998, pp. 111–124.
[17] T. Dwyer, "Scalable, versatile and simple constrained graph layout," *Comput. Graph. Forum*, vol. 28, pp. 991–998, 2009.
[18] T. Dwyer, Y. Koren, and K. Marriott, "Constrained graph layout by stress majorization and gradient projection," *Discrete Math.*, vol. 309, no. 7, pp. 1895–1908, 2009.

[19] T. Dwyer and G. Robertson, "Layout with circular and other nonlinear constraints using procrustes projection," in *Proc. Int. Symp. Graph Drawing*, 2009, pp. 393–404.
[20] P. Eades, S.-H. Hong, K. Klein, and A. Nguyen, "Shape-based quality metrics for large graph visualization," in *Proc. 23rd Int. Conf. Graph Drawing Netw. Vis.*, 2015, pp. 502–514.
[21] P. Eades, W. Huang, and S.-H. Hong, "A force-directed method for large crossing angle graph drawing," 2010, *arXiv:1012.4559*.
[22] J. Ellson, E. Gansner, L. Koutsofios, S. C. North, and G. Woodhull, "Graphviz–open source graph drawing tools," in *Proc. 9th Int. Symp. Graph Drawing*, 2001, pp. 483–484.
[23] J. J. Fowler and S. G. Kobourov, "Planar preprocessing for spring embedders," in *Proc. Int. Symp. Graph Drawing*, 2012, pp. 388–399.
[24] E. R. Gansner, Y. Koren, and S. North, "Graph drawing by stress majorization," in *Proc. Int. Symp. Graph Drawing*, 2004, pp. 239–250.
[25] R. M. Gower, N. Loizou, X. Qian, A. Sailanbayev, E. Shulgin, and P. Richtárik, "SGD: General analysis and improved rates," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 5200–5209.
[26] A. Griewank and A. Walther, *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*, vol. 105. Philadelphia, PA, USA: SIAM, 2008.
[27] A. A. Hagberg, D. A. Schult, and P. J. Swart, "Exploring network structure, dynamics, and function using networkx," in *Proc. 7th Python Sci. Conf.*, 2008, pp. 11–15.
[28] W. Huang, P. Eades, S.-H. Hong, and C.-C. Lin, "Improving multiple aesthetics produces better graph drawings," *J. Vis. Lang. Comput.*, vol. 24, no. 4, pp. 262–272, 2013.
[29] J. D. Hunter, "Matplotlib: A 2D graphics environment," *Comput. Sci. Eng.*, vol. 9, no. 3, pp. 90–95, 2007.
[30] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proc. Int. Conf. Mach. Learn.*, 2015, pp. 448–456.
[31] T. Kamada and S. Kawai, "An algorithm for drawing general undirected graphs," *Inf. Process. Lett.*, vol. 31, no. 1, pp. 7–15, 1989.
[32] N. S. Keskar, D. Mudigere, J. Nocedal, M. Smelyanskiy, and P. T. P. Tang, "On large-batch training for deep learning: Generalization gap and sharp minima," 2016, *arXiv:1609.04836*.
[33] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014, *arXiv:1412.6980*.
[34] J. F. Kruiger, P. E. Rauber, R. M. Martins, A. Kerren, S. Kobourov, and A. C. Telea, "Graph layouts by t-SNE," *Comput. Graph. Forum*, vol. 36, no. 3, pp. 283–294, 2017.
[35] J. B. Kruskal, "Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis," *Psychometrika*, vol. 29, no. 1, pp. 1–27, 1964.
[36] H. Li, Z. Xu, G. Taylor, C. Studer, and T. Goldstein, "Visualizing the loss landscape of neural nets," 2017, *arXiv: 1712.09913*.
[37] T. Orosz *et al.*, "Robust design optimization and emerging technologies for electrical machines: Challenges and open problems," *Appl. Sci.*, vol. 10, no. 19, 2020, Art. no. 6653.
[38] A. Paszke *et al.*, "Pytorch: An imperative style, high-performance deep learning library," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2019, pp. 8024–8035.
[39] H. Purchase, "Which aesthetic has the greatest effect on human understanding?," in *Proc. 5th Int. Symp. Graph Drawing*, 1997, pp. 248–261.
[40] M. Radermacher, K. Reichard, I. Rutter, and D. Wagner, "A geometric heuristic for rectilinear crossing minimization," in *Proc. 20th Workshop Algorithm Eng. Exp.*, 2018, pp. 129–138.
[41] A. Schulz, "Drawing 3-polytopes with good vertex resolution," *J. Graph Algorithms Appl.*, vol. 15, no. 1, pp. 33–52, 2011.
[42] A. Shabbeer, C. Ozcaglar, M. Gonzalez, and K. P. Bennett, "Optimal embedding of heterogeneous graph data with edge crossing constraints," in *Proc. Int. Conf. Neural Inf. Process. Syst. Workshop Challenges Data Vis.*, 2010, p. 1.
[43] D. Smilkov *et al.*, "Tensorflow.js: Machine learning for the web and beyond," in *Proc. Mach. Learn. Syst.*, 2019, pp. 309–321.
[44] T. Tieleman and G. Hinton, "Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude," COURSERA: Neural Networks for Machine Learning, 2012.
[45] M. Tiezzi, G. Ciravegna, and M. Gori, "Graph neural networks for graph drawing," 2021, *arXiv:2109.10061*.
[46] Y. Wang *et al.*, "Revisiting stress majorization as a unified framework for interactive constrained graph visualization," *IEEE Trans. Vis. Comput. Graphics*, vol. 24, no. 1, pp. 489–499, Jan. 2018.
[47] Y. Wang *et al.*, "Structure-aware fisheye views for efficient large graph exploration," *IEEE Trans. Vis. Computer Graphics*, vol. 25, no. 1, pp. 566–575, 2018.
[48] C. Ware, H. Purchase, L. Colpoys, and M. McGill, "Cognitive measurements of graph aesthetics," *Inf. Vis.*, vol. 1, no. 2, pp. 103–110, 2002.
[49] M. D. Zeiler, "ADADELTA: An adaptive learning rate method," 2012, *arXiv:1212.5701*.
[50] J. X. Zheng, S. Pawar, and D. F. Goodman, "Graph drawing by stochastic gradient descent," *IEEE Trans. Vis. Comput. Graphics*, vol. 25, no. 9, pp. 2738–2748, Sep. 2019.

Reyan Ahmed received the BS and MS degrees in computer science and engineering from the Bangladesh University of Engineering and Technology, Bangladesh. He is currently working toward the PhD degree with the Department of Computer Science, University of Arizona, Tucson, Arizona. His research interests include graph algorithms, network visualization, and data science.

Felice De Luca received the MS and PhD degrees in computer and automation engineering from the University of Perugia, Italy. He is currently a postdoctoral researcher with the Department of Computer Science, University of Arizona, Tucson, Arizona. His research interests include graph drawing, information visualization, algorithm engineering, and computational geometry.

Sabin Devkota received the BE degree in electronics and communication engineering from Tribhuvan University, Nepal. He is currently working toward the PhD degree with the Department of Computer Science, University of Arizona, Tucson, Arizona. His research is in data visualization.

Stephen Kobourov received the BS degree in mathematics and computer science from Dartmouth College, Hanover, New Hampshire, and the MS and PhD degrees from Johns Hopkins University, Baltimore, Maryland. He is currently a professor with the Department of Computer Science, University of Arizona, Tucson, Arizona. His research interests include information visualisation, graph theory, and geometric algorithms.

Mingwei Li received the BE degree in electronics engineering from the Hong Kong University of Science and Technology, Hong Kong. He is currently working toward the PhD degree with the Department of Computer Science, University of Arizona, Tucson, Arizona. His research interests include data visualization and machine learning.