Overlap Removal by Stochastic Gradient Descent with(out) Shape Awareness

Loann Giovannangeli, Frédéric Lalanne, Romain Giot, Romain Bourqui

Overlap Removal by Stochastic Gradient Descent with(out) Shape Awareness

Loann Giovannangeli, Frederic Lalanne, Romain Giot and Romain Bourqui

Abstract—In many 2D visualizations, data points are projected without considering their surface area, although they are often represented as shapes in visualization tools. These shapes support the display of information such as labels or encode data with size or color. However, inappropriate shape and size selections can lead to overlaps that obscure information and hinder the visualization's exploration. Overlap Removal (OR) algorithms have been developed as a layout post-processing solution to ensure that the visible graphical elements accurately represent the underlying data. As the original data layout contains vital information about its topology, it is essential for OR algorithms to preserve it as much as possible.
This article presents an extension of the previously published FORBID algorithm by introducing a new approach that models OR as a joint stress and scaling optimization problem, utilizing efficient stochastic gradient descent. The goal is to produce an overlap-free layout that proposes a compromise between compactness (to ensure the encoded data is still readable) and preservation of the original layout (to preserve the structures that convey information about the data). Additionally, this article proposes SORDID, a shape-aware adaptation of FORBID that can handle the OR task on data points having any polygonal shape. Our approaches are compared against state-of-the-art algorithms, and several quality metrics demonstrate their effectiveness in removing overlaps while retaining the compactness and structures of the input layouts.

Index Terms—Layout adjustment, Overlap removal, Stress optimization, Stochastic gradient descent

1 Introduction

In most 2D visualizations, data are projected from an initial high-dimensional space and represented as shapes, either to encode some of its properties or to fulfill some aesthetic criteria. However, most layout algorithms (*e.g.*, Multi Dimensional Scaling [49], [50], Graph Layout [27], [35], [53]) process data points as elements without shape or size. If not chosen carefully, these shapes and sizes can create *overlaps* in the visualization, and severely hinder its exploration by hiding information. Resolving these overlaps is still an open problem referred to as the *Overlap Removal* (OR) task [7], [8].

If not directly handled by the dimension reduction algorithm (*e.g.*, [28], [36]), it is then the responsibility of a post-processing OR algorithm to solve the task. Such algorithms take a set of data points with properties such as position, shape and size as input and move them to remove *all* the existing overlaps. The produced layout is called *overlap-free*; and is supposed to improve the readability of the initial layout. As described by Chen *et al.* [7], [8] survey, we identify two main criteria an OR algorithm should optimize to achieve satisfactory overlap-free layout: (i) the preservation of the compactness, $\mathbf{C_{comp.}}$; and (ii) the preservation of the initial layout $\mathbf{C_{lay.}}$, described in the following.

The $\mathbf{C_{comp.}}$ criterion states that an overlap-free layout should limit the increase in surface of the initial layout. Indeed, an algorithm that uniformly upscales the coordinates of the initial layout until there is no overlap solves the task (*e.g.*, uniform Scaling [8]) as long as there is no *perfect overlap* (*i.e.*, data with the exact same position). However, upscaling the initial layout implies to reduce the graphic element sizes when rendered in a viewbox of fixed size (*e.g.*, viewport on a monitor). Such upscaling should remain limited, otherwise the data encoded within the shape representations becomes too small and unreadable.

The other criterion, $\mathbf{C_{lay.}}$, defines the necessity to preserve the initial layout structures. As post-process OR algorithms are applied on already designed layouts, they must expect that these layouts were already generated by some algorithm, or even manually built to encode some task-relevant information. Preserving the user *mental map* [17] is mandatory to ensure they can relate the data points in the adjusted layout to those in the initial one. Hence, input layouts must be considered as ground truths from which the OR algorithm should not deviate too much.

While overlap can exist in any 2D visualization, we consider the OR task in a graph layout context in this article. The Graph Drawing community has always been involved in research on the OR task and we leverage its literature to extend the research (see Section 2). Thus, the data points in this article are *nodes* positioned by a graph layout algorithm. Overlaps between the nodes are considered in the geometric space, such that two nodes overlap if the intersection of their shape representation is not null.

This article proposes an extended version of FORBID [24] (Fast Overlap Removal by Stochastic Gradient Descent) and also presents SORDID, its shape-aware generalization. Both are Overlap Removal algorithms that aim at optimizing the $\mathbf{C_{lay.}}$ and $\mathbf{C_{comp.}}$ criteria explicitly. Their idea is to combine a search for an optimal scale of the initial layout with some node movements to remove the overlaps. The optimal scale is found by binary search. At each step in the binary search, node movements are computed to remove overlaps modeling a *stress optimization* problem. The stress optimization is realized with a simulated Stochastic Gradient Descent algorithm from the literature [53] with a fixed number of iterations. Fixing the number of iterations comes down to give the

All authors are with the Univ. Bordeaux, CNRS, Bordeaux INP, LaBRI, UMR 5800, F-33400 Talence, France.
E-mail: {firstname}.{lastname}@u-bordeaux.fr
This paper is an extended version of [24] appeared in GD'22.

(a) Slicing into super-pixels. (b) Alteration to create overlaps. (c) Bounding box approach. (d) Shape-aware approach.

Fig. 1: This figure illustrates our motivation for designing a shape-aware algorithm for Overlap Removal (OR). (a) The famous painting "*La Joconde*" (1506) by Leonardo da Vinci sliced into non-convex super-pixels using SLIC [1]. (b) Altered version of (a) to create overlaps by doubling the super-pixel size. (c) Result of FORBID' on (b) with a bounding box approach. (d) Result of the shape-aware OR adaptation, *i.e.*, SORDID' on (b).

algorithm a limited *budget* of node movements. Thus, it looks for the minimum scale (*i.e.*, optimizing $\mathbf{C_{comp.}}$) at which it can remove overlaps by using at most *budget* movements (*i.e.*, optimizing $\mathbf{C_{lay.}}$). As far as we know, FORBID is the first OR algorithm to explicitly include the initial layout upscaling as a property to optimize. Most algorithms from the literature measure the upscaling as a side-effect of the node movements but do not explicitly control it.

The first contribution of this article is an extensive presentation of FORBID [24] algorithm. Being designed to remove overlaps in layouts where all the nodes have a common shape (*e.g.*, rectangle, circle), it can't handle nodes with any shape efficiently. The common way to adapt standard OR algorithms to any shape is by approximating the node shapes by their bounding box, either circular or rectangular (see Figure 1c). However, such approximation is not efficient as many empty spaces remain unused. Hence, the second contribution of this article is the generalization of FORBID to polygonal nodes. This shape-aware extension is referred to as SORDID and can remove overlaps in layouts where nodes have any polygonal shape by decomposing them into triangles with a Delaunay triangulation [46]. The overlap removal task is then considered between sets of triangles composing the nodes, and it is able to produce overlap-free layouts with fine adjustments (including imbrications) of the polygonal nodes, as presented in Figure 1d. Finally, both approaches are evaluated and compared with state-of-the-art algorithms on a set of quality metrics specifically selected for this purpose. The evaluation includes various node shapes and demonstrates our algorithms capabilities to optimize the $\mathbf{C_{comp.}}$ and $\mathbf{C_{lay.}}$ criteria. FORBID[1] and SORDID[2] implementations are available online.

The remainder of this paper is organized as follows. Section 2 presents related works principally centered around the description of OR algorithms and their evaluation as proposed in [7], [8]. Section 3 describes FORBID algorithm and SORDID, its shape-aware adaptation. Section 4 reports their evaluation and comparison with state-of-the-art algorithms, while Section 5 discusses visual examples of overlap-free layouts from several OR algorithms, and our algorithms convergence. Finally, Section 6 concludes the article and presents leads for future works.

Notations: Let $G = (V, E)$ be an undirected graph with $V = \{v_1, v_2, ..., v_N\}$ its set of $N = |V|$ nodes and $E \subseteq V \times V$ its set of edges. A graph layout is defined as a tensor $X \in \mathbb{R}^{N\times 2}$ where X_i is the 2D projection of the node v_i. An initial (*i.e.*, input) layout is denoted by X^0 while a corresponding overlap-free layout is denoted by X'. For convenience, we define the set of overlapping pairs of nodes in a graph by $O \subseteq V \times V$. The Euclidean distance between v_i and v_j is denoted $||X_i - X_j||$. The minimal rectangle in which all node representations of a layout fit in is called the *Bounding box* of the layout and is denoted *BB*.

2 Related Works

This section presents related works on the Overlap Removal task, and describes some of its main algorithms. Moreover, some methods tackling Overlap Removal with a different approach or in different contexts (*e.g.*, scatterplots) are discussed.

2.1 Motivation for *Post-process* Overlap Removal

The main goal of Overlap Removal (OR) algorithms is to produce drawings where all the information can be visualized. Hence, some works [28], [36] proposed graph layout algorithms that directly handled nodes with a shape and area, and guaranteed that the graph layout would not contain overlap. Harel and Koren [28] proposed an adaptation of the seminal Kamada-Kawai [35] stress layout algorithm to produce graph drawings with elliptical nodes where there is neither node-node nor edge-node overlap. Handling the node sizes while laying them out is theoretically the ideal approach. However, in practice, many layout algorithms are designed without considering the node shapes (and thus overlap) constraints, and it is not possible to overload each and every layout technique to handle overlaps. The overlaps appear during the graph layout rendering, and the node shapes and sizes are design choices either set arbitrarily (*i.e.*, it is more aesthetically pleasing to visualize *shapes* than *dots*) or to map some node properties on their

[1]FORBID: github.com/labribkb/FORBID
[2]SORDID: github.com/labribkb/sordid

representation. Hence, generic post-process OR algorithms were designed to satisfy the overlap-free constraint of any layout. The main difference between them is their input. *Pre-process* algorithms being coupled with the layout process, they can leverage the graph structure itself, while *post-process* algorithms usually assume they only have access to some minimalist geometric data (*i.e.*, the node positions, shapes and sizes in 2D). In fact, most post-process Overlap Removal algorithms do not use the input graph edges at all. The main criterion for these algorithms is to minimize the deformation of their input layout, since it already conveys some semantics of the data and must be considered as ground-truth.

2.2 Post-Process Overlap Removal Algorithms

To remove all the node-node overlaps in a layout, the first step is to identify them. The naive way to find all overlaps (or detect if there is at least one) is to search exhaustively between all pairs of nodes, leading to a complexity in $\mathcal{O}(N^2)$. Dwyer *et al.* [15] proposed *scan-line* to search for overlaps by sorting the nodes horizontally and vertically at first. It enables to skip many comparisons and significantly accelerates the search. It achieves a complexity of $\mathcal{O}(N \log N)$ to detect if a layout contains overlaps. The algorithm remains quadratic on the number of nodes to identify all the overlaps in a layout, but in practice the *skipped* comparisons make it significantly faster than a full pairwise search. Many OR algorithms (*e.g.*, [29], [33], [43]), including the method presented in the article, rely on this scan process to detect overlaps.

The state-of-the-art of OR algorithms is well described in Chen *et al.* [7], [8] surveys, and these algorithms focus on the main criteria $\mathbf{C_{comp.}}$ and $\mathbf{C_{lay.}}$ (see Section 1). PFS [43], PFS' [29], FTA [33], RWordle-L [48] and uniform scaling [7], [8] rely on the *scan line* algorithm to identify overlaps and remove them sequentially. PFS, PFS' and FTA are made of two passes handling horizontal and vertical movements separately, while RWordle-L moves nodes on both axes simultaneously. All these algorithms have a quadratic complexity as they process all pairs of nodes at some point (*e.g.*, when moving them).

Mariott *et al.* [40] proposed several approaches to remove overlaps in a layout based on constrained optimization to maximize the $\mathbf{C_{lay.}}$ criterion. The main goal of these approaches is to preserve the user *mental map* [17] of the graph so that he can understand how the nodes/edges in the adjusted layout relate to those in the initial layout. VPSC [15], [16] also models OR as a set of constraints to relax but tends to highly deform the initial layout. Its complexity is $\mathcal{O}(CN \log C)$ where C is the number of constraints in $\mathcal{O}(N)$; leading to a final complexity in $\mathcal{O}(N^2 \log N)$. Finally, Diamond [42] is another constraint programming-based OR algorithm in $\mathcal{O}(N^2)$ that optimizes orthogonal order preservation. Its originality is to temporarily rotate nodes by 45°, representing them as *diamonds* to ease the constraint relaxations.

Gansner and North [21] introduced the idea of a proximity model to remove overlaps. Their algorithm computes a Voronoi diagram using the center of polygonal nodes. Then, every overlapping node is moved to the centroid of its Voronoi cell. If this step does not resolve all the overlaps, the layout coordinates are upscaled to expand the drawing, and the process is restarted. In addition to node-node overlaps, they propose a method to remove node-edge overlaps, allowing edge bends when necessary. While this algorithm is claimed to handle nodes represented by any polygonal shape, it only makes use of the node shapes to identify the overlaps. By design, this approach is not prone to enable node shapes imbrication to minimize node movements, since the nodes are always moved from their position to the centroid of their Voronoi cell.

Finally, PRISM [19], [20] is, as far as we know, the first stress-based OR algorithm. It models OR as a stress optimization problem in a *proximity graph* defined as the Delaunay triangulation of the initial layout. The proximity graph models the shortest distances in the initial layout, and PRISM optimizes distances alongside its edges assuming that these are where the overlaps are most likely to occur. As this assumption is often not exact, they iteratively augment the proximity graph's edges and re-compute forces until there is no overlap. In the end, PRISM is in $\mathcal{O}(t(mkN + N \log N))$ where m and k are optimization hyper-parameters, and t depends on the number of overlaps (that can be greater than N). GTREE [44] is another stress-based algorithm that leverages PRISM proximity graph to remove overlaps, building a minimum spanning tree upon it to reduce the number of forces to compute. Both GTREE and PRISM results showed that stress-based approaches to OR could be very efficient. These methods are the closest ones to FORBID and SORDID, since they also model the node movements as a stress optimization task.

Most of the OR algorithms presented in this section are designed to solve the task on nodes with the same shape. In practice, the node shapes could be different as long as it is possible (i) to check if two nodes overlap, and (ii) to compute the distance between two nodes such that they do not overlap. Nonetheless, their design and implementation focus on the context where nodes have a common shape and they are evaluated as such. The exception is the work of Gansner and North [21] that is adaptable to layouts with varying node shapes to a certain extent (*e.g.*, as long as the node shapes are not significantly larger than their Voronoi cell). However, none of these algorithms are designed to leverage the node shapes to improve the adjustments brought to the layout. In SORDID, our goal is to make use of the node shapes to detect overlaps *and* compute node displacements in order to produce finer adjustments of the layout, making use of empty spaces that exist within the bounding box of node representations.

2.3 Overlap Removal for other Criteria

Other techniques have been designed to achieve OR task, but with different focuses. In the following, we present two works that are worth mentioning, although they are not part of our benchmark as they do not tackle the same problem. More specifically, they are dedicated to scatterplots with a focus on scalability, while most of the methods presented above focus on quality and do not scale well due to their quadratic complexity.

DGRID was originally designed to produce compact visualizations [31], [39], where every datum corresponds to exactly one pixel on screen, and they adapted it to OR in more generic scatterplots [30]. The algorithm's idea is to create a grid over the initial layout, with dummy points representing locations where actual nodes could be moved minimizing displacement. Then, the nodes are assigned to grid cells using a KD-tree [4] as the spatial partitioning strategy. DGRID is in $\mathcal{O}(N \log N)$ and among the fastest OR methods because it does not need to identify overlaps as the position gridifications intrinsically make them impossible. However it suffers from local distortions due to the grid boundaries that limit the node displacements.

ScatterplotUnfold [38] is a recent *overdraw* removal technique. As opposed to all the algorithms presented above, it focuses on the

Visual Space (which is why they handle *overdraw* and not *overlap*). The algorithm does not necessarily aim at removing overlaps, but rather focuses on the preservation of the data representation *on screen*. Their goal is to preserve the visualization of clusters and the density distribution of the initial drawing. To achieve this, they manipulate parameters relative to the layout rendering (*e.g.*, node color, transparency, radius on screen). FORBID and SORDID fit in the category they call *Node dispersion* and from which they aim at overcoming the intrinsic flaws (*e.g.*, density distribution preservation, computational efficiency).

GIST [25] is another Visual Space algorithm that uses stochastic gradient descent to resolve overdraws, and aims at guaranteeing the nodes visibility in the final image that represents the layout. The originality of this algorithm is to mix the optimization of the node movements in the Geometric Space, with constraints on the node sizes in the Visual Space to make sure that they remain visible in the desired image resolution. It also uses a *bounded tolerance* to soften the overlap constraints and ease its optimization process.

Other approaches such as Data Abstraction [9], [41] or Sampling [10], [11] have been designed to process 2D scatterplots and ensure that they faithfully represent the original data. However, these methods do explicitly focus on reducing the number of represented elements to avoid overdraws. This strategy is not the scope of this article which is why we do not discuss them further.

3 FORBID AND SORDID ALGORITHMS

This section presents FORBID, a post-process Overlap Removal (OR) algorithm, and SORDID, its generalization to generic node shapes. Both algorithms follow the same structure, and aim at optimizing a *stress* function by simulating stochastic gradient descent while looking for the optimal upscaling ratio of the initial layout. Figure 2 presents a high-level abstraction view of the building blocks the algorithms are made of.

3.1 FORBID

This section presents FORBID (for Fast Overlap Removal by Stochastic Gradient Descent), and focuses on rectangular nodes.

3.1.1 Stress Model for Overlap Removal

Stress for graphs. Inherited from the MultiDimensional Scaling (MDS) [37], the *stress* energy function has demonstrated great performance as an objective function in the graph context, and especially in graph layout algorithms [6], [35], [45], [53]. The *stress* function can be expressed as:

$$\sigma(X) = \sum_{v_i,v_j \in V} W_{ij}(||X_i - X_j|| - \delta_{ij})^2 \quad (1)$$

where δ_{ij} represents the ideal distance between nodes v_i and v_j, and W_{ij} is a weighting factor for the pair (v_i, v_j). W_{ij} is usually set to δ_{ij}^{-2} such that the higher the ideal distance between two nodes is, the less they produce *energy*. Optimizing *stress* then comes down to fit a distribution of distances in a low-dimensional space (*e.g.*, 2D) to the distribution of ideal distances taken from a high-dimensional space too complex to be represented.

The goal of graph layout algorithms is to map every node from the theoretical space to a 2D position. Thus, *stress* was intuitively adapted in this context by setting the ideal distance between pair of nodes to their shortest path length in the theoretical object.

A similar reasoning was proposed by Gansner and Hu [19] to adapt the *stress* to the Overlap Removal task in PRISM. The idea

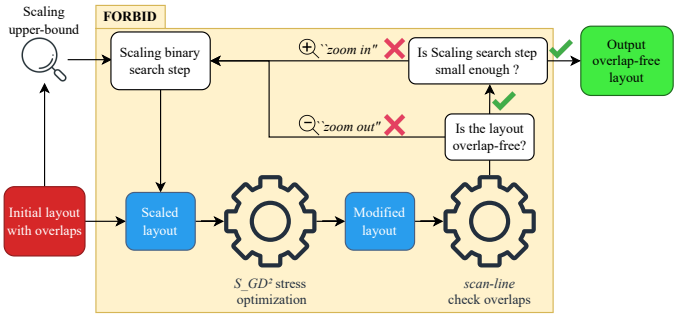


Fig. 2: Simplified workflow of FORBID algorithm representing how the stress and upscaling criteria are organized together. Gears represent the algorithms FORBID relies on (*i.e.*, *S_GD²* [53] and *scan line* [15]), and colored boxes represent layouts while white boxes represent the search for the optimal scaling.

is to tailor the notion of *ideal distance* such that optimizing the function will tend to remove overlaps while also preserving the initial layout. This is done by defining a two folded ideal distance. If a pair of nodes $p_{ij} = (v_i, v_j)$ does not overlap (*i.e.*, $p_{ij} \notin O$), the ideal distance is set to their distance in the initial layout $\delta_{ij} = ||X_i^0 - X_j^0||$. This states that if two nodes do not overlap, the distance between them should be preserved. On the other hand, if $p_{ij} \in O$, the ideal distance is set to some value such that the two nodes do not overlap anymore. That way, fitting the ideal distances distribution will intrinsically remove overlaps. The actual ideal distance value between two nodes is a design choice that must be high enough so that they do not overlap anymore. Nevertheless, it must also remain small enough since two nodes that overlap in the initial layout should remain close in the overlap-free layout. For example, PRISM [19] defined their ideal distance as $\delta_{ij} = r_{ij}||X_i - X_j||$ where r_{ij} is the minimal expansion factor of the vector $\overrightarrow{X_iX_j}$ such that the two nodes are moved following the direction between their centers until they do not overlap anymore.

This model of ideal distances comes down to create a space in which the distances between all pairs of nodes in the initial layout are preserved, and there is no longer any overlap. Such a space cannot be in 2D as preserving the distances between all pairs of nodes is not possible when we move some of them to remove overlaps. This ideal space is necessarily of higher dimension. The *stress* optimization then tries to fit the distribution of distances in the 2D plan to this ideal high dimensional space.

FORBID stress implementation. In contrast to PRISM, the ideal distance between two nodes in FORBID is set according to their size. More formally, it is the distance between v_i and v_j centers if they were tangent in their corner as shown in Figure 3:

$$\delta_{ij} = \begin{cases} ||X_i^0 - X_j^0||, \text{if } (v_i, v_j) \notin O \\ \sqrt{\left(\frac{w_i+w_j}{2}\right)^2 + \left(\frac{h_i+h_j}{2}\right)^2}, \text{if } (v_i, v_j) \in O \end{cases} \quad (2)$$

where X^0 is the initial layout, while w_k and h_k are the width and height of the node v_k. Having too tight a definition of the ideal distance such as PRISM's one means that there is only one direction alongside which the nodes can be moved to satisfy that distance. On the other hand, our model of the ideal distances makes sure that two nodes cannot overlap anymore no matter what their relative position is. We expect that this constraint will be easier to relax

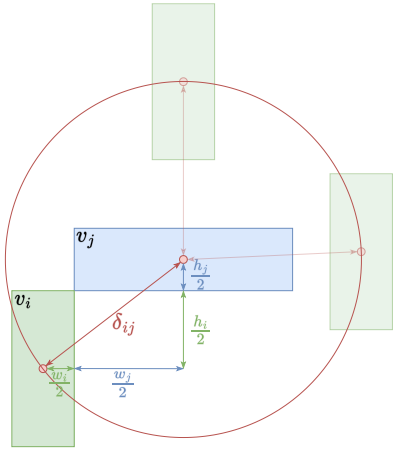


Fig. 3: FORBID ideal distance for an overlapping pair of nodes as defined in Equation 2. This is the smallest distance such that the nodes cannot overlap, no matter what their relative position is.

overall and ease the algorithm's convergence; though it might come at the cost of some inversions in the nodes orthogonal order.

Finally, the weight factor W_{ij} is an adjustable power of δ_{ij} based on whether two nodes v_i and v_j overlap:

$$W_{ij} = \begin{cases} \delta_{ij}^{\alpha}, \text{if } (v_i, v_j) \notin O \\ \delta_{ij}^{\alpha * K}, \text{if } (v_i, v_j) \in O \end{cases} \quad (3)$$

where $\alpha \in \mathbb{R}$ is a factor related to the ideal distance between the nodes (typically set to $\alpha = -2$ such that the closer two nodes are, the higher their weight is), and $K \in \mathbb{R}$ is a weight to tune the importance given to overlapping pairs of nodes.

3.1.2 Stress Optimization

Among the existing approaches to optimize a *stress* function [2], [3], [13], [52], FORBID leverages that of *S_GD²* [53]. The originality of *S_GD²* is to propose a constraint relaxation framework that mimics a stochastic gradient descent to optimize the objective function. In standard stress optimization algorithms, every node movement is computed by an aggregation of attraction and repulsion forces with every other node in the graph. To overcome the cost of such methods, *S_GD²* uses a constrained graph layout [5], [14] approach to model the stress optimization as a set of constraints between every pair of nodes that are relaxed individually [34]. Every pair is moved in direction of the gradient of the stress between its nodes. More precisely, two nodes v_i and v_j are moved in opposite direction based on the divergence between the euclidean distance $||X_i - X_j||$ and the ideal distance δ_{ij}, following the direction between their centers. The movement can be negative to move the nodes closer. The algorithm optimizes the distances between all pairs of nodes for a given number of iterations, using an annealing step size η to reduce the movements amplitude and converge toward a solution as the optimization progresses.

Stress optimization is meant to fit a distance distribution in a low dimensional space to that of a higher dimensional one (*i.e.*, δ). However, with our stress model of the OR task, modifications in the low dimensional space (*i.e.*, node movements) can affect the high dimensional space. More precisely, the distribution of ideal distances depends on the overlaps in the layout (see Equation 2). By moving the nodes during an iteration, some overlaps will be removed but some new ones can be created. Thus, it becomes necessary to re-evaluate the ideal distances such that the algorithm will always optimize the node movements to fit a distribution of distances that corresponds to the current layout with its overlaps.

3.1.3 Optimal Scaling

We define *upscaling the layout* as multiplying all its node coordinates by a fixed factor. This comes down to uniformly expand (or compress) the drawing without changing the relative distances between the nodes. For any Overlap Removal (OR) algorithm, the initial layout upscaling is a major concern. OR algorithms are required when the information encoded within the nodes must remain visible. When the graph representations are rendered in a viewbox of fixed resolution, upscaling the layout results in reduction of the node sizes on screen, which should therefore be minimized. Yet, it is not always possible to remove all the overlaps in a layout without upscaling, for example when the sum of the node areas is higher than its bounding box. Moreover, this lower bound may not be satisfactory either, especially if it requires severe distortion of the initial layout.

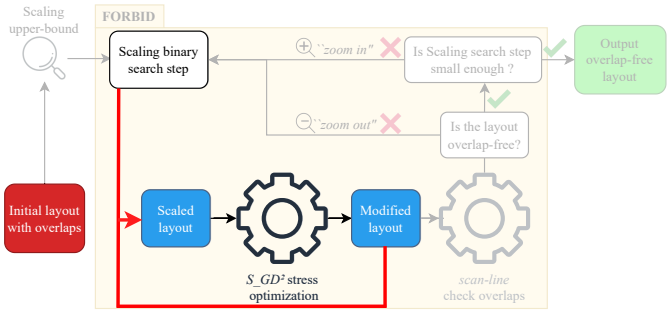
The upscaling of the initial layout should then (i) remain limited such that the node sizes do not get too small; and (ii) should be sufficient such that there is enough space to draw an overlap-free layout that preserves the initial layout structures. In most algorithms in the literature, the initial layout upscaling is a side-effect induced by the node movements, and is an evaluation criterion to compare with other algorithms.

In FORBID, finding the optimal *scaling ratio* is part of the algorithm. It is not optimized alongside the node movements with *S_GD²* since it must be set to compute the ideal distances necessary to the node movements optimization. It is found by binary search between 1 (*i.e.*, the scale of the initial layout) and an upper bound s_{max}. To ensure that FORBID converges toward an overlap-free layout, s_{max} is set to the scaling ratio of the Scaling [8] algorithm which guarantees that there is no overlap anymore (if there is no *perfect* overlap). The search is ended when the difference between two consecutive scale factors is lower than a precision threshold s_p. The depth of the binary search is $s \leq \log\left(\frac{s_{max}-1}{s_p}\right)$.

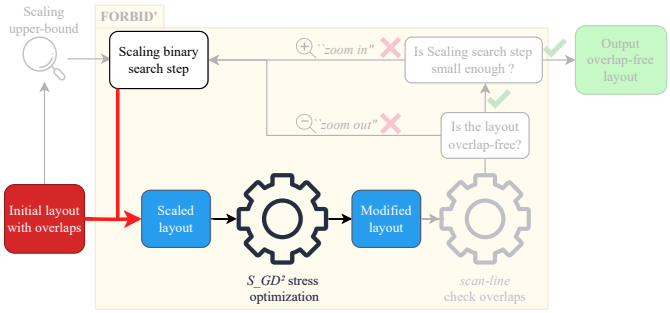
3.1.4 FORBID Algorithm and FORBID' Variant

As described in Figure 2, the first step in FORBID is to define a *scaling ratio* using the binary search (see Section 3.1.3). The binary search builds what we call *passes* in the algorithm. In every *pass*, FORBID tries to remove all overlaps at the current scale by moving the nodes with the stress optimization algorithm (see Section 3.1.2) for a fixed number of *iterations*. We also introduced a *minimum movement* condition that works as an *early stopping* for a pass if the node movements are not significant anymore (*e.g.*, less than 10^{-7}). If there remain overlaps after a pass P, the next *pass* $P+1$ will be given a higher upscaling ratio (*i.e.*, more space is needed to produce an overlap-free layout). On the other hand, there are two possibilities if there is no overlap after moving the nodes. If the difference between the scaling ratio of the previous *pass* $P-1$ and that of the current *pass* P is higher than a threshold s_p (for *scale precision*), we consider that the search is not refined enough and we start a new *pass* $P+1$ with a smaller scaling ratio (*i.e.*, we could find a more compact overlap-free layout). However, if the difference of scaling ratio is lower than s_p, we stop the optimization and return the overlap-free layout.

To summarize, the binary search looks for the optimal scaling ratio that enables the stress optimization algorithm to remove



(a) FORBID, every new pass $P+1$ starts from the state of the previous pass *Modified layout* X^P.



(b) FORBID', a pass P starts from the state of the *Initial layout* X^0.

Fig. 4: Differences between FORBID (a) and FORBID' (b) variants.

overlaps without too much node movements. This enable to limit both the upscaling effect and the distortion of the initial layout.

The number of *passes* is given by the depth of the binary search $s \leq \log\left(\frac{s_{max}-1}{s_p}\right)$, the detection of overlaps with *scan-line* [15] is in $\mathcal{O}(N\log(N))$ and *S_GD²* stress optimization algorithm is in $\mathcal{O}(N^2)$ such that FORBID complexity is $\mathcal{O}(sN^2)$.

FORBID and FORBID' variants. There remain one design choice to make in order to have an exhaustive description of FORBID. In fact, since every pass P takes a layout X as input and produces a new layout X^P, there are two choices to start the next step $P+1$. The first possibility is to start $P+1$ from the output layout of P (*i.e.*, X^P), as shown in Figure 4a. This means that the algorithm will be allowed to move the nodes further and further as P increases. We expect that allowing more movement in this way will enable the model to converge faster, at the cost of a larger deformation of the initial layout. On the other hand, the second possibility is to start $P+1$ from the (scaled) initial layout X^0 (see Figure 4b). This will induce less deformation of the initial layout, but will most likely slow the convergence down.

The first variant (starting $P+1$ from X^P) is called **FORBID** while the second (starting $P+1$ from X^0) is referred to as **FORBID'**. The difference between both is highlighted by the red arrows in their respective illustration in Figure 4.

3.2 SORDID

This section presents SORDID (for Shape-aware Overlap Removal by Stochastic Gradient Descent), the shape-aware adaptation of FORBID. While FORBID optimizes one distance for every pair of nodes, SORDID decomposes every node's polygon into triangles, and optimizes the distances between the triangles of every pair of nodes. As far as we know, SORDID is the first method to handle OR with nodes of various shapes by decomposing them, as opposed to most methods from the literature that handle arbitrary node shapes by considering their minimum bounding circle or rectangle.

Where FORBID complexity is $\mathcal{O}(sN^2)$, SORDID one is higher due to the decomposition of the nodes into smaller elements, leading to a significant increase in the number of distances to optimize. The decomposition being bounded by the number of vertices of the polygons, the worst case complexity of SORDID is $\mathcal{O}(sN^2t)$ where t is the maximum number of vertices of the polygons in a layout.

3.2.1 Node Shape Representation

SORDID handles various shapes by decomposing them into triangles with a Constrained Delaunay Triangulation [46]. It means that in reality, any *shape* that is not a polygon must be approximated with a polygon. Another limitation that we didn't overcome is the handling of hollow shapes, such as donuts. Instead, SORDID triangulates polygons using their outer border only.

In the next sections, we consider that a *node* can either be considered by its actual shape, or by the set of triangles that compose its shape. In addition, the minimum bounding circle of every node's shape is computed and will be used later.

3.2.2 Overlap Detection Between Polygonal Nodes

The test for overlap between two nodes v_i and v_j follows a hierarchical process. We first verify if the minimum bounding circles of their shape are overlapping. If not, then the two nodes cannot overlap. But if the bounding circles do overlap, it does not mean that the node shapes overlap either. In this case, SORDID will go a step deeper in the node representations by checking if the minimum bounding circle of any triangle composing v_i does overlap with that of any triangle composing v_j. If not, then there is no overlap between v_i and v_j. But for each overlapping bounding circle between pairs of triangles composing v_i and v_j, SORDID finally checks whether the triangles actually overlap.

SORDID also leverages *scan-line* [15] (see Section 2) accelerated detection of overlaps using the stored leftmost and rightmost points of every node to detect skippable tests.

3.2.3 Ideal Distance and Movements

In SORDID, the ideal distances are also conditioned by the node overlaps. For a pair of nodes $p_{ij} = (v_i, v_j) \notin O$ (*i.e.*, that does not overlap), the ideal distance is set to the distance between the node centers in the initial layout, as described in Equation 2. In this case, only one movement will be computed between the two node centers to optimize that distance.

$$\delta_{ij} = ||X_i^0 - X_j^0||, \forall (v_i, v_j) \notin O \tag{4}$$

On the other hand, when $p_{ij} \in O$, there are as many ideal distances as there are pairs of triangles between the triangulated representation of the two nodes. For every pair of triangle, if they do not overlap, then the ideal distance is the euclidean distance between the center of their minimum bounding circles. If they do overlap, the ideal distance is the sum of their minimum bounding circle radiuses.

$$\Delta_{ij,kl} = \begin{cases} ||\mathrm{c}(T_k) - \mathrm{c}(T_l)||, \text{if } (T_k, T_l) \notin \Omega^{ij} \\ \mathrm{r}(T_k) + \mathrm{r}(T_l), \text{if } (T_k, T_l) \in \Omega^{ij} \end{cases} \tag{5}$$
$$, \forall (v_i, v_j) \in O, \forall (T_k, T_l) \in \mathrm{DT}^i \times \mathrm{DT}^j$$

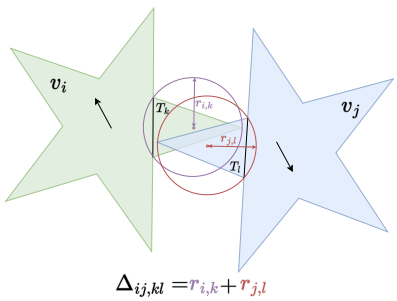


Fig. 5: SORDID ideal distance for an overlapping pair of nodes, as defined in Equation 5. This figure only illustrates the distance between the overlapping triangles, but the distance between all the pairs of triangles composing v_i and v_j are computed.

where the set of triangles composing a node v_i is defined as DT^i (standing for Delaunay Triangulation) and $(T_k, T_l) \in \mathrm{DT}^i \times \mathrm{DT}^j$ are the pairs of triangles between the triangulation of v_i and v_j. The ideal distance for one pair of triangles (T_k, T_l) composing v_i and v_j is defined by $\Delta_{ij,kl}$. The set of overlapping pairs of triangles in $\mathrm{DT}^i \times \mathrm{DT}^j$ is represented by Ω^{ij}. Finally, $\mathrm{c}(T)$ and $\mathrm{r}(T)$ give the center and radius of the minimum bounding circle of a triangle T. An illustration of an ideal distance between two overlapping triangles composing polygonal nodes is presented in Figure 5.

It is important to note that when a movement is computed between two triangles, its direction follows the vector between their minimum bounding circle centers, and the whole nodes (*i.e.*, all the triangles composing the two nodes) are moved. In practice, it means that there are much more distances to optimize in SORDID than in FORBID, and we expect it to be slower. Nevertheless, this fine-grained definition of ideal distances enables tight adjustments (and even imbrications) of polygonal nodes.

The weight factor also adapts FORBID's definition (see Equation 3) using the triangles from Delaunay Triangulation of the node shapes:

$$W_{ij} = ||X_i^0 - X_j^0||^{\alpha}, \forall (v_i, v_j) \notin O \tag{6}$$

$$\omega_{ij,kl} = \begin{cases} (||\mathrm{c}(T_k) - \mathrm{c}(T_l)||)^{\alpha}, \text{if } (T_k, T_l) \notin \Omega^{ij} \\ (\mathrm{r}(T_k) + \mathrm{r}(T_l))^{\alpha * K}, \text{if } (T_k, T_l) \in \Omega^{ij} \end{cases} \tag{7}$$
$$, \forall (v_i, v_j) \in O, \forall (T_k, T_l) \in \mathrm{DT}^i \times \mathrm{DT}^j$$

where $\alpha \in \mathbb{R}$ is a factor related to the ideal distance between the nodes, and $K \in \mathbb{R}$ is a weight to configure the importance given to overlapping pairs of nodes/triangles.

Finally, the stress optimized by SORDID, considering the triangulation of polygonal nodes, is expressed as:

$$\sigma(X) = \sum_{(v_i, v_j) \notin O} W_{ij} (||X_i - X_j|| - \delta_{ij})^2 + \sum_{(v_i, v_j) \in O} \sum_{(T_k, T_l) \in \mathrm{DT}^i \times \mathrm{DT}^j} \omega_{ij,kl} (||c(T_k) - c(T_l)|| - \Delta_{ij,kl})^2 \tag{8}$$

where δ_{ij} and W_{ij} are defined in Equation 4 and Equation 6.

To sum up most of the framework design, we remind that Figure 4 presents the arrangement of its building blocks. In addition, two behaviors were omitted from the framework description. The first is to conduct a pass of stress optimization without changing the scale factor of the initial layout, if the area of the initial layout's bounding box is higher than the sum of the node areas. Doing so prevents the algorithm from going through a complete optimization when the task can be solved in the initial layout's scale. The second omitted behavior is responsible for guaranteeing that FORBID generates an overlap-free layout. Hence, during the whole optimization, we recall a *best* version of the layout we found to return it in the end, should the last pass in the algorithm not be an overlap-free layout. If no overlap-free layout is ever found, the algorithm returns the initial layout upscaled by the scaling ratio that guarantees that there is no overlap anymore (see Section 3.1.3).

4 Evaluation

This section presents the results of FORBID and SORDID evaluation as well as their comparison with state-of-the-art Overlap Removal (OR) algorithms. The section is split in two evaluations: (i) on rectangular nodes, and (ii) on polygonal nodes. Our algorithms are compared with the state-of-the-art baseline algorithms (see Section 4.1.1) on a selection of metrics described in Section 4.1.3.

Some results presented in this section differ from those in the printed original study [24] as there was an error on the *el_rsdd* metric reports. This error was made to the disadvantage of FORBID and FORBID' and has been corrected in the latest online version.

4.1 Evaluation protocol

This section presents the design choices made to evaluate FORBID and SORDID. We follow the protocol defined by Chen *et al.* [7], [8] for reproducibility purposes. That includes quality metrics, datasets and baseline algorithms to compare with.

4.1.1 Baseline Algorithms

Chen *et al.* [7], [8] presented a complete survey comparing many Overlap Removal (OR) algorithms from the literature. Following their results, the baseline algorithms we selected are PFS' [29], PRISM [19], and Diamond [42], each of them being state-of-the-art in its approach to the OR task. They are respectively based on scan-line, stress optimization and constraint relaxation. In addition, since FORBID belongs to the stress optimization approach, we also included GTree [44] as it is supposed to be an improvement of PRISM and it will enable more comparison within the family. Please refer to Section 2 for a more detailed description of these algorithms.

4.1.2 Datasets

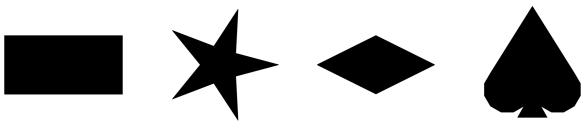
Following Chen *et al.* [7], [8] evaluation protocol, we use the Generated and Graphviz datasets available online[1].

Generated is a set of 840 synthetic graphs specifically created for the benchmark of Chen *et al.* [7], [8]. It is made of 120 graphs of size $10, 20, 50, 100, 200, 500, 1000$, laid out with the FM^3 algorithm [27].

Graphviz is a set of 14 real-world graphs from the Graphviz suite [22]. They have between 36 and 1463 nodes and are laid out with *SFDP* algorithm [32].

These two datasets are made of nodes with a rectangular shape. Since SORDID works on polygonal nodes, we created modified versions of the two datasets to evaluate it on additional shapes.

[1] Generated and Graphviz graphs: https://github.com/agorajs/agora-dataset



Rectangle Star Diamond Spade

Fig. 6: Polygonal shapes considered in the evaluation. Stars can have from 3 to 6 arms.

TABLE 1: Number of graphs, nodes, and overlaps for each dataset in the experiment. For the number of overlaps, the second row of each dataset presents the *average* $\pm$ *standard deviation*.

		Generated	Graphviz
# Graphs		840	14
# Nodes		[10; 1000]	[36; 1463]
# Overlaps	**Rectangle**	[0; 36537] 3196 $\pm$ 8717	[4; 11582] 2119 $\pm$ 4080
	Star	[0; 30920] 2687 $\pm$ 7338	[4; 21921] 3746 $\pm$ 7106
	Diamond	[0; 18350] 1575 $\pm$ 4325	[0; 8111] 1239 $\pm$ 2506
	Spade	[0; 37615] 3277 $\pm$ 8935	[5; 25370] 4432 $\pm$ 8352

More precisely, we preserve the node positions (*i.e.*, their center), and their rectangle's dimension in the original datasets is used as basis for the polygons. The polygons will be defined in the next, and Figure 6 presents an example for each of them. Assuming that a node was defined with a center (x,y) and a rectangular shape (w,h), all the following shapes are built to have the maximal size that fit in a circle of center (x,y) with a radius $r = \max(w,h)/2$.

Star. A star has a random number of arms between 3 and 6. The arm lengths is based on the original rectangular node and is defined as $\max(w,h)/2$. The hollow parts of a star is $\max(w,h)/6$ away from its center.

Diamond. A diamond is defined as the maximum regular diamond that fits in the original rectangle of the node.

Spade. A spade is made by connecting two discretized "ears" (*i.e.*, circular arcs) with three points: one for the top and two defining the "tail" of the shape.

These transformations are applied for all nodes in all graphs of both **Graphviz** and **Generated** datasets. We did not consider polygon rotations in this evaluation. Information on the initial layout of every dataset are reported in Table 1.

4.1.3 Metrics

With Post-process Overlap Removal (OR) algorithms, we assume that the provided initial layouts were computed to optimize some criteria, producing a meaningful representation for the end-user. Hence, a good OR algorithm should be able to remove all the overlaps in the initial layout while also preserving its aspect.

As the OR algorithms that will be compared remove all overlaps in the layouts, the evaluation of their results is based on their preservation of the initial layout. Since initial layouts can be deteriorated in different ways, we leverage five quality metrics from Chen *et al.* [8] survey to capture various kinds of deterioration. Other metrics can measure the compliance with $\mathbf{C_{comp.}}$ and $\mathbf{C_{lay.}}$ criteria (*e.g.*, [26], [38]). We chose to use the metrics defined by Chen *et al.* [8] to leverage their results and extend the study. All these metrics are oriented *lower is better*.

Nodes orthogonal order. *oo_nni* counts the number of times the orthogonal order of the nodes has been violated.

$$oo\_nni = \frac{\begin{array}{r}\sum_{(v_i,v_j)\in V^2, i\neq j} \mathbb{1}(x_i^0 > x_j^0 \wedge x_i' < x_j'),\\ +\mathbb{1}(y_i^0 > y_j^0 \wedge y_i' < y_j')\end{array}}{N(N-1)} \tag{9}$$

where $X_i^0 = (x_i^0, y_i^0)$ (resp. $X_i' = (x_i', y_i')$) are the 2D coordinates of v_i in the initial layout X^0 (resp. overlap-free layout X'). The metric is normalized to enable the comparison of layouts with different number of nodes.

Convex hull area. *sp_ch_a* is the ratio of the overlap-free layout's convex hull's area over that of the initial layout, the optimal value being 1. This metric comes down to evaluate the *expansion* of the layout and thus captures its *upscaling*. It is the main metric to evaluate the criterion $\mathbf{C_{comp.}}$.

$$sp\_ch\_a = \frac{\mathcal{A}(\text{ConvexHull}(X'))}{\mathcal{A}(\text{ConvexHull}(X^0))} \tag{10}$$

where *ConvexHull* computes the smallest convex region containing all the nodes of a layout. *ConvexHull* is computed taking into consideration the node shapes and sizes, and not only their center.

Aspect ratio. *gs_bb_iar* measures the deviation of the overlap-free layout's aspect ratio compared to that of the initial layout.

$$gs\_bb\_iar = \max\left(\frac{W' * H^0}{H' * W^0}, \frac{H' * W^0}{W' * H^0}\right) \tag{11}$$

where W^0 and H^0 (resp. W' and H') are the initial layout X^0 (resp. overlap-free layout X') rectangular bounding box width and height.

Node movements. *nm_dm_imse* measures the average deformation of the initial layout by quantifying node movements from their position in the initial layout to their position in the overlap-free layout. Since the layout upscaling is already captured by another metric (*i.e.*, *Convex Hull Area*), Node movements is made unsensitive to the scale and range of the layouts. Doing so enables this metric to capture the raw deformation of the initial layout and is especially useful to evaluate the criterion $\mathbf{C_{lay.}}$.

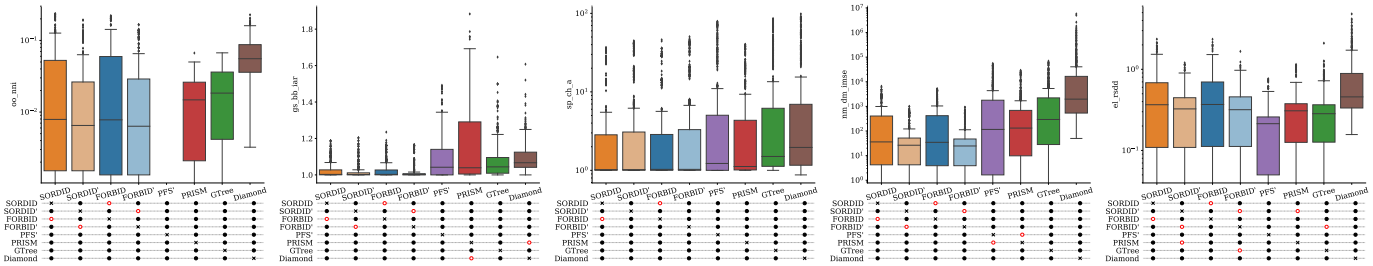
$$nm\_dm\_imse = \frac{1}{N}\sum_{v_i\in V} ||X_i' - \text{shift}(\text{scale}(X_i^0))||^2 \tag{12}$$

where *shift* and *scale* functions project the positions of the initial layout in the value domain of the overlap-free layout. These transformations are linear based on the layout bounding box sizes.

Edge length preservation. *el_rsdd* captures the uniformity of distance preservation. As opposed to *Node Movements* that penalizes any relative node movement, *Edge Length Preserv.* measures the average deviation of distances alongside the edges of the Delaunay Triangulation of the initial layout. The intuition is that if some quantity of movement is required to solve the task, all the nodes should move a little (as opposed to a few nodes moving a lot). Formally, the metric is defined as:

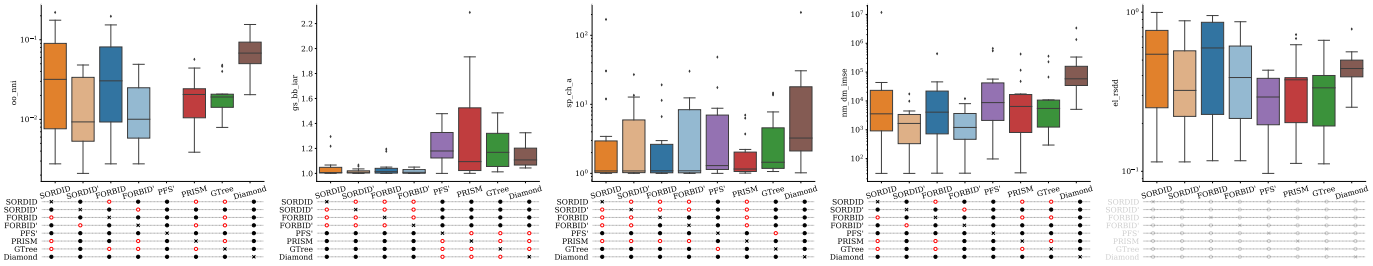
$$\begin{aligned}\rho_{uv} &= \frac{||X_u' - X_v'||}{||X_u^0 - X_v^0||}, \forall (u,v) \in E_{DT},\\ \overline{\rho} &= \frac{1}{|E_{DT}|}\sum_{(u,v)\in E_{DT}} \rho_{uv},\\ el\_rsdd &= \frac{\sqrt{\frac{1}{|E_{DT}|}\sum_{(u,v)\in E_{DT}}(\rho_{uv} - \overline{\rho})^2}}{\overline{\rho}}\end{aligned} \tag{13}$$

where E_{DT} is the set of edges of the Delaunay Triangulation of the initial layout X^0, and ρ_{uv} represents (u,v) edge length's deviation ratio in the overlap-free layout.



(a) *Nodes Ortogh. Order* (b) *Aspect Ratio* (c) *Convex Hull Area* (d) *Node Movements* (e) *Edge Length Preserv.*

Fig. 7: SORDID, FORBID and some selected state-of-the-art OR algorithms performance according to the aesthetic metrics described in Section 4.1.3 on the **Generated** dataset. All plots have a log scale on the Y-axis. All metrics are oriented *lower is better*. The complete description of these representations is given in Section 4.1.4. As expected, our algorithms can produce inversions in the nodes orthogonal order. Nevertheless, they demonstrate great capability on compactness (*sp_ch_a*) and distortion minimization (*nm_dm_imse*).



(a) *Nodes Ortogh. Order* (b) *Aspect Ratio* (c) *Convex Hull Area* (d) *Node Movements* (e) *Edge Length Preserv.*

Fig. 8: SORDID, FORBID and some selected state-of-the-art OR algorithms performance according to the aesthetic metrics described in Section 4.1.3 on the **Graphviz** dataset. All plots have a log scale on the Y-axis. All metrics are oriented *lower is better*. The complete description of these representations is given in Section 4.1.4. Trends are close to that on the **Generated** dataset (see Figure 7).

4.1.4 Statistical Validation and Results Presentation

Statistical validation. In the following, we will compare the performance of various algorithms on relatively large sets of data. To improve the faithfulness of the evaluation, we support the performance analysis with statistical tests following the protocol of Purchase [47]. For every metric, an ANOVA test is first conducted to assert whether the differences between the algorithms is due to chance or not. To do so, we use the repeated measures non-parametric test of Friedman [18] with a confidence threshold $\alpha = 0.05$. If this test passes, post-hoc pairwise comparisons are tested with Conover [12] at confidence $\alpha = 0.05$.

Results report. The metric score distribution of the algorithms are reported in box-plots (*e.g.*, Figure 7a). All them have Y-axis log scales. Below each plot is represented a matrix of pairwise significance. If the Friedman test didn't pass (*e.g.*, Figure 8e) the matrix is faded as pairwise comparisons should not be interpreted. Otherwise, black circles in the matrix encode pairwise significance between the corresponding line/column algorithms, while red circles encode the non-significance. Since all the metrics can be read as *lower is better*, an algorithm is better than another if its distribution (*i.e.*, *box*) is more dense on a smaller value domain, **and** if the difference between them is statistically significant.

4.2 Evaluation on Rectangular Nodes

This section focuses on the evaluation and comparison of FORBID, SORDID, their variant and state-of-the-art algorithms on rectangular nodes. Figure 7 reports the performance of these algorithms on the **Generated** dataset, while Figure 8 reports them on the **Graphviz** dataset. Please refer to Section 4.1.4 for more information on the results presentation.

4.2.1 Performance on Generated

Figure 7a shows that all the algorithms succeed in minimizing *Nodes Ortogh. Order* according to the order of magnitude of their performance (overall $< 10^{-1}$). By design, PFS' has a score of 0 on that metric since it cannot violate the nodes orthogonal order. FORBID' and SORDID' have similar performance to PRISM and are a slightly better than GTree. The difference is not significant between neither FORBID and SORDID, nor FORBID' and SORDID'. Finally, Diamond performance is significantly worse than other algorithms on that metric as it consistently deteriorates it more than them.

On *Aspect Ratio* (see Figure 7b), we can see that our four methods are significantly better than all the others. We assume this score is related to the explicit handling of the upscaling factor in our algorithms. Indeed, the initial layout positions are *uniformly* upscaled alongside the X and Y dimensions to create more empty spaces. This uniform upscaling does not deteriorate the layout bounding box's aspect ratio. Only the node movements that come afterward can modify that ratio, and we can see that the deviation remains minimal.

Figure 7c presents *Convex Hull Area* performance, a measurement of the overlap-free layout's compactness. The distributions show that our methods have better results than state-of-the-art algorithms overall. As expected (see Section 3.1.4), the FORBID' and SORDID' variants of the algorithm produce less compact

layouts than their standard counterparts, although the difference between FORBID and SORDID is not significant.

Node Movements is the main metric to measure the initial layout preservation (*i.e.*, $\mathbf{C_{lay.}}$ criterion), for which Figure 7d presents all algorithm results on the Generated dataset. We can see that our algorithms have significantly less node movements than other state-of-the-art-methods. The difference is not significant between FORBID (resp. FORBID') and SORDID (resp. SORDID'). We can also note that the FORBID' and SORDID' variants have consistently less node movements than their standard counterparts. This confirms the assumption (see Section 3.1.4) that starting every new *pass* from the initial layout, rather than from the output of the previous pass, leads the model towards an overlap-free layout with less relative movements.

The last metric to compare the selected algorithms on, *Edge Length Preserv.*, measures how uniform the deformation of the closest nodes in the initial layout is. The performance on that metric are reported in Figure 7e. We can see that PFS' is the best on that metric, followed by PRISM and GTREE, then SORDID' and FORBID'. SORDID and FORBID have slightly worse results, and Diamond have the worst among the selected algorithms. Again, the *prime* variants have better metric values than the standard ones. This result could be expected as they produce less node relative movements, but it also means that the produced movements were more uniformly distributed across the nodes.

4.2.2 Performance on Graphviz

Overall, the results on the Graphviz dataset (see Figure 8), corroborate those observed on the Generated. There are less significant pairwise differences on Graphviz than Generated, mainly because of Graphviz's size of 14 graphs. With such a small volume of data, statistical tests cannot validate the significance of pairwise differences if the results are not *strongly* correlated. On *Nodes Ortogh. Order*, PFS' is still the best while FORBID' and SORDID' compete with GTree and PRISM. On the other hand, FORBID, SORDID are worse, and Diamond is the worst on that metric. *Aspect Ratio* is still highly in favor of FORBID, SORDID and their variants. On *Convex Hull Area* and *Node Movements*, we can also observe that our algorithms are overall better than state-of-the-art methods (except for PRISM that overperforms on *Convex Hull Area*) with SORDID suffering from one problematic case on both metrics (see the outlier in Figure 8c and 8d). Finally, the ANOVA test did not pass on the *Edge Length Preserv.* metric. It means that, according to the statistical test, the variation of performance according to the algorithms used for each graph could be due to chance. Pairwise comparison should not be interpreted in this case. Eventually, the trends between the boxes are close to those observed on the Generated dataset.

4.2.3 Synthesis on Rectangular Nodes

In this section, we have seen that FORBID and FORBID' competed well with state-of-the-art algorithms. On some metrics, they even significantly outperform them (*e.g.*, *Aspect Ratio*, *Node Movements* and *Convex Hull Area* to a lesser extent).

As expected, we observed that the *prime* variant of FORBID and SORDID consistently led to overlap-free layouts that better preserve the initial layout (*i.e.*, optimizing $\mathbf{C_{lay.}}$), at the cost of some upscaling. On the other hand, the standard variant tends to produce compact layouts optimizing $\mathbf{C_{comp.}}$.

SORDID being a generalization of FORBID, we assumed that the difference between them was not significant on rectangular nodes as this shape is strictly equivalent to its rectangular bounding box. The evaluation demonstrates that it is indeed the case as the difference between them is *never* significant, except between SORDID' and FORBID' on *Convex Hull Area* with **Generated**.

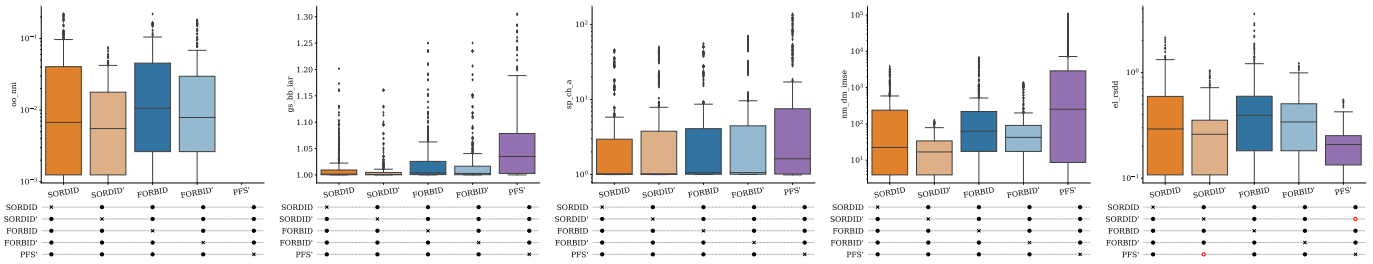
4.3 Evaluation on Polygonal Nodes

This section presents the results of SORDID, SORDID', FORBID, FORBID' and PFS' on the **Generated** dataset with polygonal node representations as described in Section 4.1.2. We only compare the Overlap Removal (OR) algorithms proposed in this article with PFS' as it is the best performing state-of-the-art algorithm according to the evaluation on rectangular nodes (see Section 4.2). In addition, we only compare the algorithms on the **Generated** dataset as it is the largest considered in this evaluation, and thus the one on which results are the most significant. The **Graphviz** dataset will be later used for visual evaluation (see Section 5.2). To handle polygonal nodes with standard OR algorithms such as FORBID and PFS', the common way is to provide them a bounding box of the node shapes (here, the rectangular bounding box). One can note that the rectangular bounding boxes of the Diamond shapes are equal to to the rectangular nodes that were evaluated in the previous Section 4.2. Hence, the metric scores of FORBID, FORBID' and PFS' on the Diamond shape are the same as in the evaluation on Rectangular nodes (up to some randomness in the optimization's initialization).

As the trends between the methods' performance are similar across various *shapes* on some metrics, the results are discussed *by metric* (rather than *by dataset* of shape) in the following. The main assumption we can make in this evaluation is that a method that handles polygonal nodes directly (*i.e.*, SORDID), should lead to more compact overlap-free layouts. Indeed, as opposed to a bounding box approach, the fine-grained awareness that the algorithm has of the node shape representations should enable it to make more use of the empty spaces in the layout. In general, we expect a shape-aware algorithm to provide better results on polygonal nodes than standard algorithms that must approximate the node shapes. The results of these evaluations are presented in Figures 9, 10 and 11 for the **Spade**, **Diamond**, and **Star** shapes respectively.

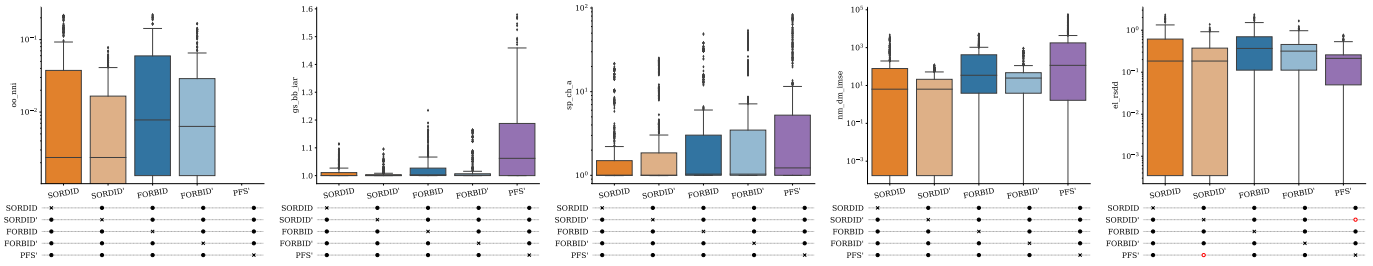
On the *Nodes Ortogh. Order* metric, the methods performance is the same across the three shapes. PFS' has never produced any inversion of the nodes orthogonal order, as its design does not allow it. We can observe that SORDID and SORDID' are slightly better than their FORBID and FORBID' counterparts on this metric. We think this result is due to the shape-aware approach's capability to have finer adjustments between the nodes to relax the overlap constraints. These fine adjustments can lead to significant benefits, especially in denser regions of the layouts, where an approach on bounding boxes would have no other choice but to move some nodes out of the dense region to be able to relax the overlap constraints, which is more prone to inversions in the nodes orthogonal order.

FORBID and SORDID are still significantly better than PFS' on *Aspect Ratio*, as shown in Figure 9b, 10b and 11b. The difference is significant between SORDID and FORBID (and their *prime* counterparts), meaning that the shape-aware approach leads to better preservation of the initial layout's aspect ratio. Since the aspect ratio can only be deteriorated by node movements in our methods, this result was expected in view of our assumption that a shape-aware algorithm should be able to use more empty spaces



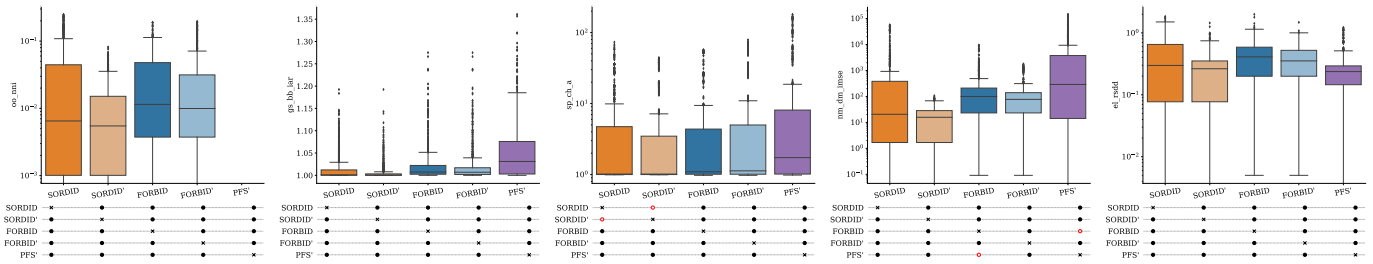
(a) *Nodes Ortogh. Order* (b) *Aspect Ratio* (c) *Convex Hull Area* (d) *Node Movements* (e) *Edge Length Preserv.*

Fig. 9: SORDID, FORBID and some selected state-of-the-art OR algorithms performance according to the aesthetic metrics described in Section 4.1.3 on the **Generated** dataset with a **Spade** shape (see Section 4.1.2). All plots have a log scale on the Y-axis. All metrics are oriented *lower is better*. The complete description of these representations is given in Section 4.1.4.



(a) *Nodes Ortogh. Order* (b) *Aspect Ratio* (c) *Convex Hull Area* (d) *Node Movements* (e) *Edge Length Preserv.*

Fig. 10: SORDID, FORBID and some selected state-of-the-art OR algorithms performance according to the aesthetic metrics described in Section 4.1.3 on the **Generated** dataset with a **Diamond** shape (see Section 4.1.2). All plots have a log scale on the Y-axis. All metrics are oriented *lower is better*. The complete description of these representations is given in Section 4.1.4.



(a) *Nodes Ortogh. Order* (b) *Aspect Ratio* (c) *Convex Hull Area* (d) *Node Movements* (e) *Edge Length Preserv.*

Fig. 11: SORDID, FORBID and some selected state-of-the-art OR algorithms performance according to the aesthetic metrics described in Section 4.1.3 on the **Generated** dataset with a **Star** shape (see Section 4.1.2). All plots have a log scale on the Y-axis. All metrics are oriented *lower is better*. The complete description of these representations is given in Section 4.1.4.

to resolve overlaps, making the task easier to solve with less movement.

Convex Hull Area is the main metric to evaluate how compact the overlap-free layout produced by an algorithm is, and we expect SORDID shape-aware approach to achieve better results. As presented in Figure 9c, 10c and 11c, it is the case with the three shapes, but in different proportions. SORDID and SORDID' provide the most benefits compared to FORBID and PFS' (*i.e.*, bounding box approaches) on the Diamond shape. We argue that this is because this shape is the most prone to fine adjustments and *imbrications*, especially compared to the Spade shape. On Stars, it is noteworthy that the difference between SORDID and SORDID' is not significant. We could expect better results for SORDID and SORDID' on the Star shape as we can easily imagine stars to be imbricated alongside their arms. However, this is rarely possible in practice as the star arms orientation is fixed and they cannot be rotated. With our shapes generation, having neighboring stars with arms sharing the same orientation on a common side is very rare, especially since the arm orientation of a star depends on its number of arms (which is random).

Figure 9d, 10d and 11d present the algorithm results on the *Node Movements* metric. As we can see, SORDID and SORDID' are consistently better on that metric. Although we assume that a shape-aware approach should be able to remove overlaps using less node movements by using more available empty spaces than an approach with bounding boxes, it was not evident that this behavior could be observed on this metric. Indeed, since *Node Movements* measurement ignores the overlap-free layout's upscaling, it can favor a method that uniformly upscales the initial layout and does not move any node, compared to another method that moves some

nodes to limit the upscaling.

Finally, the results on *Edge Length Preserv.* are presented in Figure 9e, 10e and 11e. On Spade and Star shapes, PFS' is better than both SORDID and FORBID, while SORDID is better on the Diamond shape. Overall, the shape-aware approach of SORDID always shows better results than FORBID. As a reminder, this metric captures the uniformity of short distances degradation. Hence, we can conclude that the fine adjustments that SORDID makes are significantly beneficial to the layouts it produce.

Overall, we observed that the shape-aware approach of SORDID consistently led to better results than the adaptation of standard algorithms that uses the bounding box of the node shapes. We also found evidence supporting the assumption that a shape-aware approach would lead to more compact overlap-free layout by achieving more fine-grained adjustments between the node shapes. The evaluation demonstrates the good capacity of this approach to minimize the node movements and upscaling necessary to remove all overlaps.

4.4 Execution Times

This section discusses the execution times of FORBID, SORDID and the state-of-the-art Overlap Removal (OR) algorithms of the evaluation. Diamond algorithm's execution times are not reported as they were dramatically higher than other methods (as presented in Chen *et al.* [7], [8]). All the algorithm execution times were measured on the same i9-12900KF CPU. It is important to note that the decomposition of the node shapes with the Delaunay Triangulation, and the computation of the pairwise ideal distances that come with it, are parallelized in SORDID to alleviate the additional cost induced by the decomposition. This parallel execution makes it about twice as fast, and makes its execution time acceptable despite the additional cost of the node shape decompositions, and considering the increases in quality observed in Section 4.3. In both FORBID and SORDID, we also tried to parallelize the stochastic optimization of pairwise node movements, but found that it led to results of lower quality. We think that processing the node movements sequentially improves the optimization as any node coordinate update takes into consideration the movements that were already applied to the node during the iteration.

We only report the results on Graphviz with Rectangular nodes, since they were consistently similar with other shapes and on Generated. Reporting on Graphviz enables comparison of the algorithms' execution times in regard of the input graph properties. Thus, we identify three groups of difficulty, as presented in Figure 12.

On *easy* graphs, FORBID, FORBID' and PFS' are instantaneous (*i.e.*, less than 5*ms*). SORDID and SORDID' are instantaneous as well on most cases, but take more time on *b143* (14*ms* and 51*ms* respectively). On the other hand, GTree and PRISM are consistently slower on these graphs. Even though these examples are *easy* (*i.e.*, the number of nodes and overlaps is small), we can already observe some difference between the algorithms.

On *medium* graphs, FORBID, FORBID' are among the fastest, followed by SORDID GTree and SORDID', while PRISM is the slowest by a significant margin.

FORBID and SORDID are overall the fastest on *hard* graphs, SORDID results being surprising as its complexity is higher than that of FORBID, but its execution time is lower. This improvement

	|V|	|E|	|O|	Graph	SORDID	SORDID'	FORBID	FORBID'	PFS'	GTREE	PRISM
Easy	36	107	4	dpd	1	1	0	0	0	3	5
	41	49	20	unix	1	3	0	0	0	6	7
	43	64	9	rowe	1	1	0	0	0	5	7
	47	55	33	size	1	1	0	0	0	7	19
	50	99	13	ngk104	2	1	0	0	0	5	7
	76	104	19	nan	2	2	1	1	0	21	64
	79	181	33	b124	4	4	2	2	0	16	81
	135	366	53	b143	14	51	5	5	0	35	107
Medium	213	269	1105	mode	101	183	73	107	1	120	521
	302	611	268	xx	276	334	143	169	1	250	725
	302	611	282	b102	211	274	127	176	1	280	948
Hard	1054	1083	11582	root	1324	5241	2226	5783	13	4861	27945
	1235	1616	10540	badvoro	1900	5046	3258	6239	23	6646	17652
	1463	5806	5691	b100	4789	10350	11158	10571	30	10569	54634

Fig. 12: Execution time (in milliseconds) of the algorithms on **Graphviz** with rectangular nodes. Rows are sorted by *graph complexity* measured in number of nodes, edges and overlaps.

is mostly due to some difference in their optimization. SORDID', FORBID' and GTree are overall slower and have relatively close execution times. Finally, PRISM is still slower than the others.

On *medium* and *hard* graphs, PFS' is still almost instantaneous; making it the fastest of the considered algorithms in this evaluation. It is also interesting to note that the graph on which the algorithms were the slowest is not the one that has the most overlaps, as *b100* has only half the number of overlaps of *root* or *badvoro*. All these methods are more sensitive to increases in the number of nodes than in overlaps, as expected in view of their complexities (*i.e.*, quadratic on the number of nodes).

Overall, we can conclude that FORBID scales well according to the task complexity. As expected, FORBID' is slower than FORBID, mostly because of its constraint to start each stress optimization from the upscaled initial layout. We also observe that SORDID and SORDID' shape-aware approaches tend to be slower, though SORDID can be faster thanks to its different model of ideal distances. Since SORDID' has the constraint of starting each optimization from the scaled initial layout of a *prime* variant, and the necessity to optimize more distances with the decomposition of the node shapes into triangles, it is not surprising to see that it is slower than the other variants.

5 Discussion

This section discusses some behaviors of FORBID and SORDID observed during the evaluation, and some of their limitations. It also discusses their convergence and some visual examples.

5.1 Convergence Analysis

The convergence plots of FORBID, SORDID and their *prime* variant are presented in Figure 13. Each plot reports the evolution of the *stress*, number of overlaps and upscaling against the number of iterations in the stress optimization algorithm. Vertical dashed lines represent beginning of new *passes* (see Section 3.1.4).

In all these plots, we can see that the stress curves follow that of the number of overlaps. This confirms that optimizing stress to remove overlaps is efficient. The difference with *prime* variants is also well represented. In FORBID and SORDID, most overlaps are removed in the first few passes and the last ones are dedicated to the search for the optimal upscaling ratio while preserving the overlap-free layout. On the other hand, FORBID' and SORDID' have to restart their optimization from the scaled initial layouts every time a new pass begins, the problem being made easier or harder with a different upscaling ratio. In the end, we can also observe that the *prime* variants need more iterations to converge toward an overlap-free layout that satisfies the search for a compromise

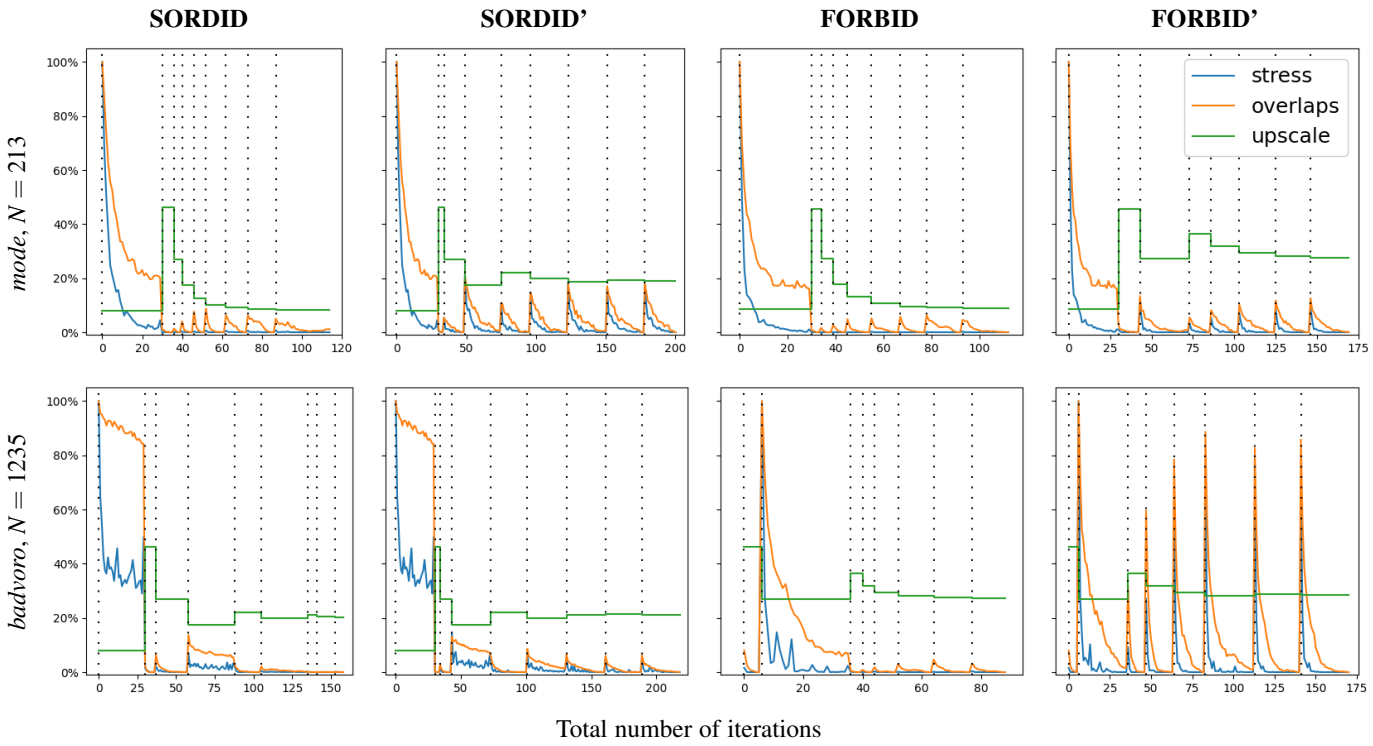


Fig. 13: Convergence plots of SORDID, FORBID and their *prime* variant on two **Graphviz** graphs with **Diamond** shape. They report the evolution of stress, number of overlaps and scaling ratio against the total number of iterations. Vertical dashed lines represent beginning of new *passes*. Stress and number of overlaps are normalized by their respective maximum value, while upscaling ratio is normalized by its binary search maximum bound (see Section 3.1.3).

between $\mathbf{C_{lay.}}$ and $\mathbf{C_{comp.}}$. This explains why *prime* variants are consistently slower.

Comparing SORDID and FORBID, we can see that the stress, number of overlaps and upscaling follow the same trend on the *mode* graph. On the other hand, the results are different on *badvoro*, which is a more complicated sample. These differences mostly come from the two approaches difference when it comes to overlap identification. For instance, we can see on *badvoro* that both SORDID and SORDID' began the first *pass* with a scaling ratio of 1. It means that according to their more fine-grained definition of the node shape representations, there was enough empty space in the initial layout to try to solve the Overlap Removal task with movements only. On the other hand, the bounding box approach of FORBID and FORBID' did not identify that. In the end, SORDID requires slightly more iterations to converge than FORBID.

It is also interesting to observe that despite having a large definition of the upper bound s_{max} for the binary search for the optimal scaling ratio, the algorithms are able to find a solution at a lower scale. This upper bound is set to guarantee that the algorithms converge toward an overlap-free layout, but is most of the time unnecessarily high. Other upper bounds or optimization strategies could be used to find the optimal scale, but are not investigated in this article and are left for future works.

5.2 Visual Evaluation

This section discusses some visual examples of the overlap-free layouts produced by most of the algorithms selected for the evaluation. Visual examples of **Graphviz** graphs with rectangular nodes are presented in Figure 14.

On very small graphs such as *unix*, no significant difference can be observed between the algorithms. On *b124*, all the algorithms also have similar results, with the exception of SORDID and SORDID'. Indeed, we can see in their layouts that the "wide and flat" node was pushed far from its initial position. This behavior is probably induced by the design of ideal distances in SORDID which leads to high repulsive forces in this case. Indeed, the wide flat node is most likely decomposed into two triangles which boundig circle radiuses are relatively high. Hence, the movements involving that node have high amplitudes, and their result are clearly visible on *b124*. Such an example shows that there is a limit to which SORDID can preserve the initial layout when the node shapes have a high aspect ratio. SORDID and SORDID' drawings of *b124* look similar in regard to that wide flat node because the amplitude of that node's movements are so high that it is moved to its final position in very few iterations. Hence, even though SORDID' begins every new pass from the initial layout node coordinates, its first few iterations consistently move the node to this location. Then, it does not find a way to put it back closer to its initial position because of the high amplitudes of the movements it involves.

On *mode*, FORBID, SORDID and PRISM damaged the initial layout to produce more compact embeddings. The other algorithms preserved the initial layout structure. PFS' and GTree layouts are made of some orthogonally unpleasing parts as opposed to FORBID' and SORDID' layouts that very well preserved the initial layout structure. The same behavior can be observed on *xx*, where FORBID, SORDID and PRISM produced compact layouts, and GTree created some unnecessarily long edges. Finally, on *badvoro*, we can clearly see the downside of producing very

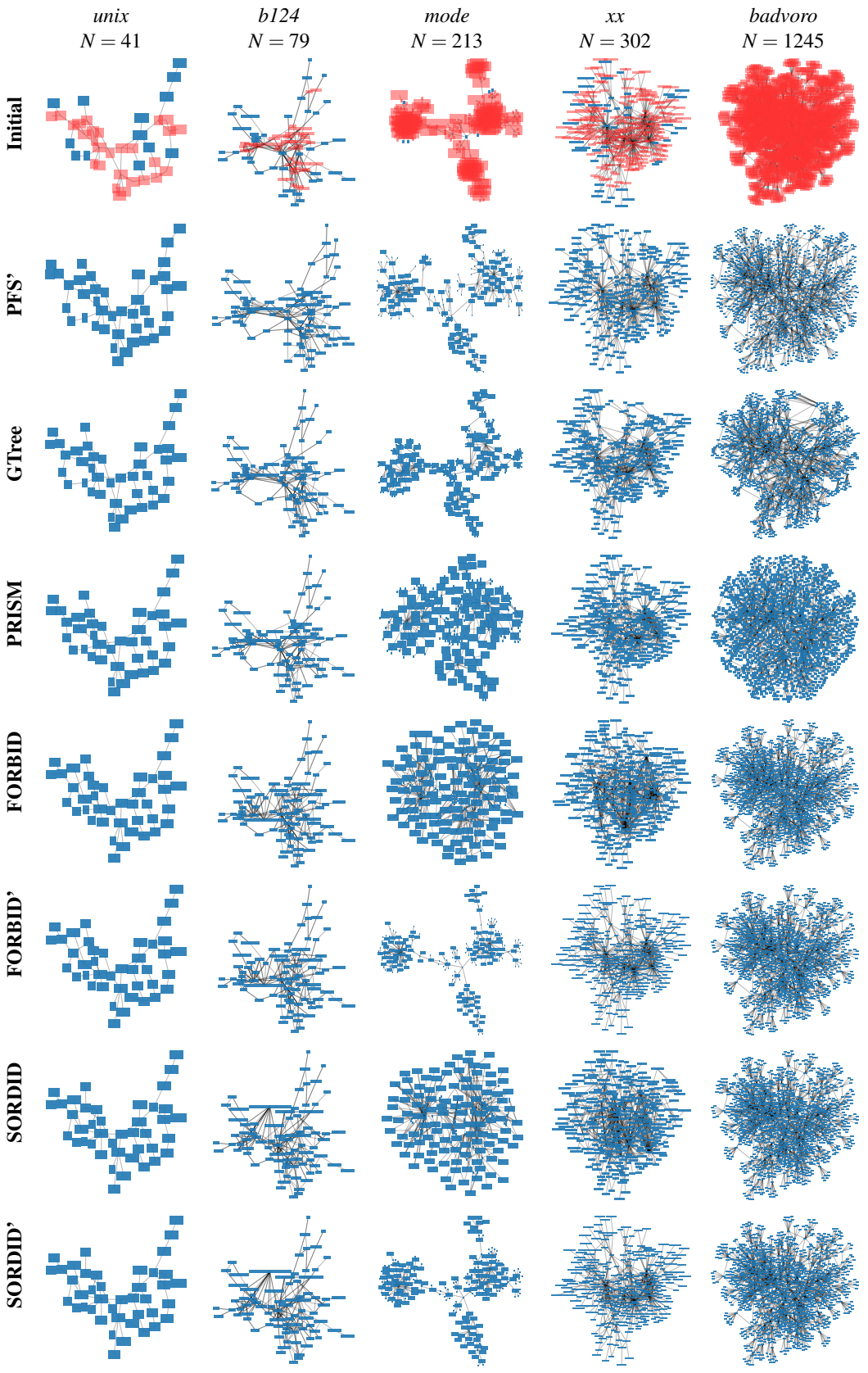


Fig. 14: Graph visualization of examples from the **Graphviz** dataset with rectangular node shapes. The images present the overlap-free layouts of SORDID, FORBID and their *prime* variant, as well as PFS', GTREE and PRISM. Nodes are colored in transparent red if they overlap each other, and opaque blue otherwise.

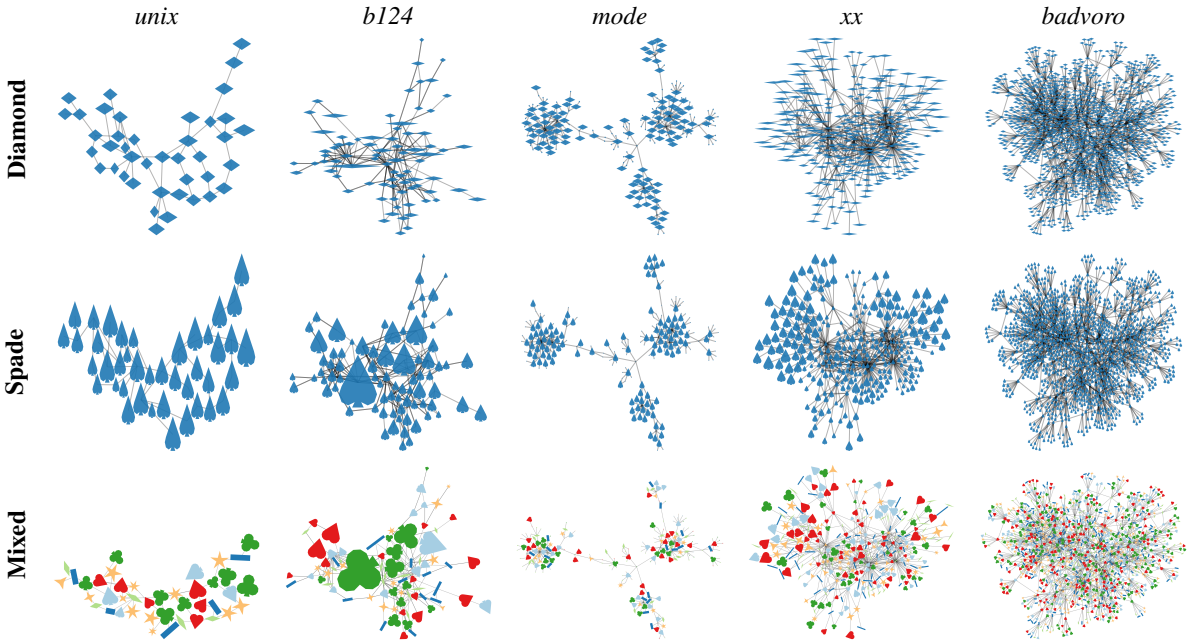


Fig. 15: SORDID' overlap-free layout examples from the **Graphviz** dataset with Diamond, Spade and Mixed shapes. In Mixed shapes visualizations, the shapes used are Heart and Clover in addition to those presented in Figure 6.

compact layouts. PRISM overlap-free layout is so compact that it becomes impossible to recognize the initial layout structures and visualize the edges. GTree overlap-free layout has also severely deteriorated the initial layout. The remaining algorithms produced satisfactory results where the initial layout structures are preserved, while maintaining some compactness.

In addition to graph visualization with nodes having a rectangular shape, Figure 15 presents some layouts of SORDID' with Diamond, Spade and Mixed node shapes. In Mixed shape visualizations, the shapes used are those presented in Figure 6 and two others: Heart and Clover. These new shapes are built in the same way as Spade, by connecting some points with discretized circular arcs. These representations demonstrate the great capacity of the shape-aware approach to leverage empty spaces that would not be used by bounding-box approaches. We can even observe the *imbrications* of nodes representations, especially on *b124*.

5.3 Limitations

Throughout the evaluation, we measured how the algorithms presented in this article performed. However, some of their behaviors could not be captured and are discussed in this section.

The first limitation concerns FORBID and SORDID algorithms. In these variants, the algorithm begins every stress optimization from the output of the previous pass. It means that the ideal distances of a pass P are based on the node positions at the end of the pass $P-1$. This design allows the algorithm to move the nodes more, and helps it to converge faster. On the other hand, it means that the longer the algorithm takes to converge, the more it will forget about the initial layout. At some point, the structures of the initial layout could theoretically be completely lost. In addition, even if a pass P in the stress optimization does not lead to an overlap-free layout, the next pass $P+1$ will take the node positions of that pass X^P as the reference to compute the ideal distances. This behavior could be prevented to improve the algorithm capability to preserve the initial layout, but this is left for future works.

Another limitation concerning both FORBID and SORDID comes from the stress optimization algorithm used [53]. What makes this algorithm efficient is its principle of applying forces independently between every pair of nodes. However, this principle can create problematic behaviors. We believe that it is the source of the bad placement of the "wide flat" node of SORDID and SORDID' observed on *b124* in Figure 14, alongside their model of ideal distances. This behavior is quite common with stress optimization algorithms: an element is pushed far away from its initial position because of a high repulsing force with a neighbor, and it will never come back to its original position because of other nodes' repulsive forces. This pitfall is most likely accentuated by handling pairs of nodes individually. To try another efficient optimization algorithm is left for future works.

The last limitation we discuss concerns the heuristic encoded within FORBID to obtain a compromise between $\mathbf{C_{comp.}}$ and $\mathbf{C_{lay.}}$, and that can be expressed as follows. Every pass in the stress optimization algorithm has a budget of node movements: it can move all pairs of nodes *n_iter* times. If all overlaps are not removed after this budget is consumed, we assume that too much movement would be necessary to remove all overlaps and it would not comply with $\mathbf{C_{lay.}}$ criterion. Hence, a new pass is started in the algorithm, changing the scale of the input layout. The issue is that this heuristic is very dependant on some hyper-parameters such as the number of iterations *n_iter*, the weight factors of the distances to optimizes (*i.e.*, α and K, see Section 3), and even the upper bound and the optimization strategy to find the optimal scale (here, binary search). That is to say, the optimal compromise between $\mathbf{C_{comp.}}$ and $\mathbf{C_{lay.}}$ criteria is only possible if the hyper-parameters provided permit them; the algorithm itself hardly provides this guarantee. Nevertheless, the parameters were fixed for all the samples in the evaluations we conducted, and the algorithms demonstrated good capabilities to produce satisfying overlap-free layouts for a large variety of graphs.

6 Conclusion

This paper has presented FORBID, an Overlap Removal (OR) algorithm that leverages upscaling and stress optimization by simulated stochastic gradient descent to minimize deformations of the initial layout. In addition, the article proposed SORDID, an adaptation of FORBID to handle any polygonal node representation. Both are based on the preservation of the scale and the node pairwise distances to produce an overlap-free layout where the the initial graph layout structures are preserved and the surface used limited. The adaptation to polygonal nodes is achieved by decomposing every polygon into triangles and processing nodes as sets of triangles.

They were compared to a selection of state-of-the-art OR algorithms. The evaluation demonstrates that FORBID is among the best techniques to forbid overlaps while preserving the initial layout and limiting the upscaling effect.

Despite their quadratic complexity, FORBID and SORDID compete with the fastests methods of the literature, handling graphs with up to a thousand nodes and/or ten thousands of overlaps in less than 10 seconds. Nevertheless the PFS' algorithm remains the fastest by a significant margin compared to any other technique.

The main lead for future works is to improve the algorithm's complexity to better handle large graphs (*e.g.*, up to tens of thousands of nodes). The first idea to achieve this is to sub-sample the pairs of nodes that must be processed to remove overlaps. A multi-scale approach could enable to optimize the preservation of the initial layout structures while sampling the distances to preserve (i.e., preserve distances between-clusters and within-cluster; ignore between nodes of different clusters). Finally, with the recent advances in Deep Learning for graph drawing [23], [51], training a Deep Learning model to solve the OR task could be an interesting lead to investigate. By design, these models can scale to large graphs as they are capable of solving the task they have learned in almost constant time.

Another lead for future work is to optimize the optimal scaling ratio alongside the node positions by stochastic gradient descent. In our algorithms, the scaling ratio is set before the optimization of the node movements because we build the ideal distances distribution on the upscaled layout. Yet, it would be interesting to see if we can reach better compromises between compactness and the preservation of the initial layout by optimizing both the node movements and the scaling ratio at the same time.

Acknowledgments

This research was founded by the french ANR project InvolvD OPE 2020-0425.

References

[1] R. Achanta, A. Shaji, K. Smith, A. Lucchi, P. Fua, and S. Süsstrunk. SLIC superpixels compared to state-of-the-art superpixel methods. *IEEE transactions on pattern analysis and machine intelligence*, 34(11):2274–2282, 2012.

[2] R. Ahmed, F. De Luca, S. Devkota, S. Kobourov, and M. Li. Multicriteria Scalable Graph Drawing via Stochastic Gradient Descent, $(SGD)^2$. *IEEE Transactions on Visualization and Computer Graphics*, 28(6):2388–2399, 2022.

[3] R. Ahmed, F. D. Luca, S. Devkota, S. Kobourov, and M. Li. Graph Drawing via Gradient Descent, $(GD)^2$. In *International Symposium on Graph Drawing and Network Visualization*, pp. 3–17. Springer, 2020.

[4] J. L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

[5] M. Bostock, V. Ogievetsky, and J. Heer. D^3 data-driven documents. *IEEE transactions on visualization and computer graphics*, 17(12):2301–2309, 2011.

[6] U. Brandes and C. Pich. An experimental study on distance-based graph drawing. In *International Symposium on Graph Drawing*, pp. 218–229. Springer, 2008.

[7] F. Chen, L. Piccinini, P. Poncelet, and A. Sallaberry. Node overlap removal algorithms: A comparative study. In *International Symposium on Graph Drawing and Network Visualization*, pp. 179–192. Springer, 2019.

[8] F. Chen, L. Piccinini, P. Poncelet, and A. Sallaberry. Node Overlap Removal Algorithms: an Extended Comparative Study. *Journal of Graph Algorithms and Applications*, 24(4):683–706, 2020. doi: 10.7155/jgaa.00532

[9] H. Chen, W. Chen, H. Mei, Z. Liu, K. Zhou, W. Chen, W. Gu, and K.-L. Ma. Visual abstraction and exploration of multi-class scatterplots. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):1683–1692, 2014. doi: 10.1109/TVCG.2014.2346594

[10] X. Chen, T. Ge, J. Zhang, B. Chen, C.-W. Fu, O. Deussen, and Y. Wang. A recursive subdivision technique for sampling multi-class scatterplots. *IEEE transactions on visualization and computer graphics*, 26(1):729–738, 2019.

[11] X. Chen, J. Zhang, C.-W. Fu, J.-D. Fekete, and Y. Wang. Pyramid-based Scatterplots Sampling for Progressive and Streaming Data Visualization. *IEEE Transactions on Visualization and Computer Graphics*, 28(1):593–603, 2022. doi: 10.1109/TVCG.2021.3114880

[12] W. J. Conover. *Practical nonparametric statistics*, vol. 350. john wiley & sons, 1999.

[13] S. Devkota, R. Ahmed, F. De Luca, K. E. Isaacs, and S. Kobourov. Stress-Plus-X (SPX) Graph Layout. In *International Symposium on Graph Drawing and Network Visualization*, pp. 291–304. Springer, 2019. doi: 10.1007/978-3-030-35802-0_23

[14] T. Dwyer. Scalable, versatile and simple constrained graph layout. In *Computer graphics forum*, vol. 28, pp. 991–998. Wiley Online Library, 2009.

[15] T. Dwyer, K. Marriott, and P. J. Stuckey. Fast node overlap removal. In *International Symposium on Graph Drawing*, pp. 153–164. Springer, 2005.

[16] T. Dwyer, K. Marriott, and P. J. Stuckey. Fast node overlap removal—correction. In *International Symposium on Graph Drawing*, pp. 446–447. Springer, 2006.

[17] P. Eades, W. Lai, K. Misue, and K. Sugiyama. Preserving the mental map of a diagram. Technical report, Technical Report IIAS-RR-91-16E, Fujitsu Laboratories, 1991.

[18] M. Friedman. The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the american statistical association*, 32(200):675–701, 1937.

[19] E. Gansner and Y. Hu. Efficient, proximity-preserving node overlap removal. *Journal of Graph Algorithms and Applications*, 14(1):53–74, 2010.

[20] E. R. Gansner and Y. Hu. Efficient node overlap removal using a proximity stress model. In *International Symposium on Graph Drawing*, pp. 206–217. Springer, 2008.

[21] E. R. Gansner and S. C. North. Improved Force-Directed Layouts. In *International Symposium on Graph Drawing*, pp. 364–373. Springer, 1998. doi: 10.1007/3-540-37623-2_28

[22] E. R. Gansner and S. C. North. An open graph visualization system and its applications to software engineering. *Software: practice and experience*, 30(11):1203–1233, 2000.

[23] L. Giovannangeli, F. Lalanne, D. Auber, R. Giot, and R. Bourqui. Toward Efficient Deep Learning for Graph Drawing (DL4GD). *IEEE Transactions on Visualization and Computer Graphics*, 2022.

[24] L. Giovannangeli, F. Lalanne, R. Giot, and R. Bourqui. FORBID: Fast Overlap Removal By stochastic gradIent Descent for Graph Drawing. In *International Symposium on Graph Drawing and Network Visualization*, pp. 61–76. Springer, 2022.

[25] L. Giovannangeli, F. Lalanne, R. Giot, and R. Bourqui. Guaranteed Visibility in Scatterplots with Tolerance. *IEEE Transactions on Visualization and Computer Graphics*, 30(1):792–802, 2024. doi: 10.1109/TVCG.2023.3326596

[26] K. Gray, M. Li, R. Ahmed, M. K. Rahman, A. Azad, S. Kobourov, and K. Börner. A Scalable Method for Readable Tree Layouts. *IEEE Transactions on Visualization and Computer Graphics*, 2023. doi: 10.1109/TVCG.2023.3274572

[27] S. Hachul and M. Jünger. Drawing large graphs with a potential-field-based multilevel algorithm. In *International Symposium on Graph Drawing*, pp. 285–295. Springer, 2004.

[28] D. Harel and Y. Koren. Drawing graphs with non-uniform vertices. In *Proceedings of the Working Conference on Advanced Visual Interfaces*, pp. 157–166, 2002.
[29] K. Hayashi, M. Inoue, T. Masuzawa, and H. Fujiwara. A layout adjustment problem for disjoint rectangles preserving orthogonal order. In *International Symposium on Graph Drawing*, pp. 183–197. Springer, 1998.
[30] G. M. Hilasaca, W. E. Marcílio-Jr, D. M. Eler, R. M. Martins, and F. V. Paulovich. Overlap Removal of Dimensionality Reduction Scatterplot Layouts. *arXiv preprint arXiv:1903.06262*, 2019.
[31] G. M. Hilasaca and F. V. Paulovich. A visual approach for user-guided feature fusion. In *Anais Estendidos da XXXII Conference on Graphics, Patterns and Images*, pp. 133–139. SBC, 2019.
[32] Y. Hu. Efficient, high-quality force-directed graph drawing. *Mathematica journal*, 10(1):37–71, 2005.
[33] X. Huang, W. Lai, A. Sajeev, and J. Gao. A new algorithm for removing node overlapping in graph visualization. *Information Sciences*, 177(14):2821–2844, 2007.
[34] T. Jakobsen. Advanced character physics. In *Game developers conference*, vol. 3, pp. 383–401. IO Interactive, Copenhagen Denmark, 2001.
[35] T. Kamada, S. Kawai, et al. An algorithm for drawing general undirected graphs. *Information processing letters*, 31(1):7–15, 1989.
[36] T. Kamps, J. Kleinz, and J. Read. Constraint-based spring-model algorithm for graph layout. In *International Symposium on Graph Drawing*, pp. 349–360. Springer, 1995.
[37] J. B. Kruskal and M. Wish. *Multidimensional scaling*, vol. 11. Sage, 1978.
[38] Z. Li, R. Shi, Y. Liu, S. Long, Z. Guo, S. Jia, and J. Zhang. Dual Space Coupling Model Guided Overlap-Free Scatterplot. *IEEE Transactions on Visualization and Computer Graphics*, 29(1):657–667, 2022.
[39] G. M. H. Mamani. *A visual approach for user-guided feature fusion*. PhD thesis, Universidade de São Paulo.
[40] K. Marriott, P. Stuckey, V. Tam, and W. He. Removing Node Overlapping in Graph Layout Using Constrained Optimization. *Constraints*, 8:143–171, 2003. doi: 10.1023/A:1022371615202
[41] A. Mayorga and M. Gleicher. Splatterplots: Overcoming overdraw in scatter plots. *IEEE transactions on visualization and computer graphics*, 19(9):1526–1538, 2013.
[42] W. Meulemans. Efficient optimal overlap removal: Algorithms and experiments. In *Computer Graphics Forum*, vol. 38, pp. 713–723. Wiley Online Library, 2019.
[43] K. Misue, P. Eades, W. Lai, and K. Sugiyama. Layout adjustment and the mental map. *Journal of Visual Languages & Computing*, 6(2):183–210, 1995.
[44] L. Nachmanson, A. Nocaj, S. Bereg, L. Zhang, and A. Holroyd. Node overlap removal by growing a tree. In *International Symposium on Graph Drawing and Network Visualization*, pp. 33–43. Springer, 2016.
[45] M. Ortmann, M. Klimenta, and U. Brandes. A sparse stress model. In *International Symposium on Graph Drawing and Network Visualization*, pp. 18–32. Springer, 2016.
[46] L. Paul Chew. Constrained delaunay triangulations. *Algorithmica*, 4(1):97–108, 1989.
[47] H. C. Purchase. *Experimental human-computer interaction: a practical guide with visual examples*. Cambridge University Press, 2012.
[48] H. Strobelt, M. Spicker, A. Stoffel, D. Keim, and O. Deussen. Rolled-out Wordles: A Heuristic Method for Overlap Removal of 2D Data Representatives. *Computer Graphics Forum*, 31(3):1135–1144, 2012. doi: 10.1111/j.1467-8659.2012.03106.x
[49] M. E. Tipping and C. M. Bishop. Probabilistic principal component analysis. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 61(3):611–622, 1999.
[50] L. Van der Maaten and G. Hinton. Visualizing data using t-SNE. *Journal of machine learning research*, 9(11), 2008.
[51] X. Wang, K. Yen, Y. Hu, and H.-W. Shen. DeepGD: A Deep Learning Framework for Graph Drawing Using GNN. *IEEE Computer Graphics and Applications*, 41(5):32–44, 2021.
[52] Y. Wang, Y. Wang, Y. Sun, L. Zhu, K. Lu, C.-W. Fu, M. Sedlmair, O. Deussen, and B. Chen. Revisiting stress majorization as a unified framework for interactive constrained graph visualization. *IEEE transactions on visualization and computer graphics*, 24(1):489–499, 2017.
[53] J. X. Zheng, S. Pawar, and D. F. Goodman. Graph drawing by stochastic gradient descent. *IEEE transactions on visualization and computer graphics*, 25(9):2738–2748, 2018.

Loann Giovannangeli is a PhD. student at the LaBRI, University of Bordeaux, France. He worked one year as a research engineer in the LaBRI. He obtained his Master of Science degree in 2019 from the University of Bordeaux. His research interest include Information Visualization, Machine Learning and especially the applications of Artificial Intelligence for visualizations generation and evaluation.

Frederic Lalanne was graduated from Enseirb-Matmeca in 2013 and joined the LaBRI, University of Bordeaux in 2014 where he has been working mostly on large data analysis, visualization and LaBRI's large data platform.

Romain Giot received his Ph.D. degree in biometric authentication at the University of Caen in 2012 and is an associate professor at the University of Bordeaux in a big-data visualization team since 2013. His researches are dedicated to visualization, machine learning and their junction in eXplainable AI (XAI). He co-authored dozens of peer-reviewed papers and is involved in several Program Committees.

Romain Bourqui received his Master and PhD degrees in Computer Science from the University Bordeaux I in 2005 and 2008. He has been an associate professor at the University of Bordeaux since 2009. His research interests include Information Visualization, Large Data Visualization, Explainable Machine Learning.