DOI: 10.1111/cgf.70139
Eurographics Conference on Visualization (EuroVis) 2025
W. Aigner, N. Andrienko, and B. Wang
(Guest Editors)

COMPUTER GRAPHICS *forum*
Volume 44 (2025), Number 3

Optimizing Staircase Motifs in Biofabric Network Layouts

Sara Di Bartolomeo[†1], Markus Wallinger[2] and Martin Nöllenburg[1]

[1]TU Wien, Vienna, Austria
[2]Technical University of Munich, Germany

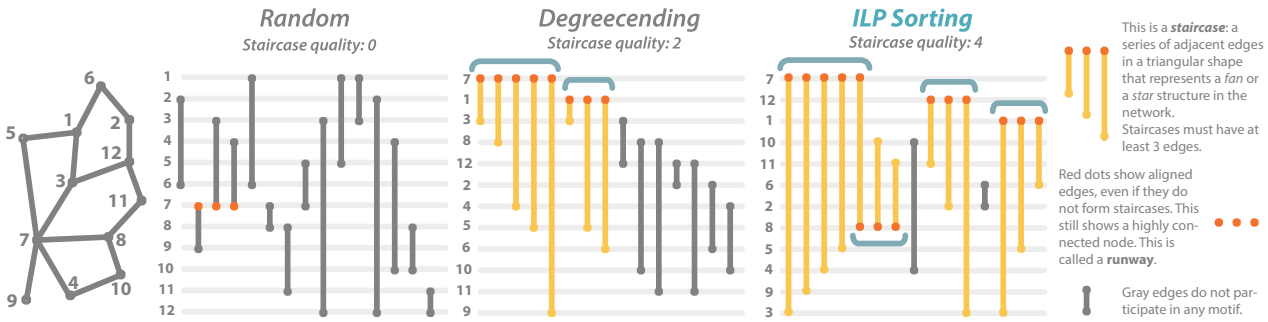


Figure 1: *An example of the same graph (`grafo1943.12`) from Rome-Lib [BGL*97]) rendered using three different methods of node and edge sorting: the first just uses the same ordering in which edges and nodes are read, and shows how a* ***random*** *ordering of edges and nodes does not help to spot insights about the structure of the graph at a glance. The second uses the* ***Degreecending*** *method [FFH*25] and clearly reveals a couple of relevant motifs: two staircases, which correspond to two star motifs in node-link visualization, highlighting highly connected nodes and their connections. However, being a heuristic, it does not produce the best possible sorting. The third is* ***our proposed, exact method****: with a better graph sorting of nodes and edges, it is immediately apparent that the graph contains more staircase motifs. The ILP solution contains 221 variables and 3479 constraints, and it is solved in 21 seconds.*

Abstract

Biofabric is a novel method for network visualization, with promising potential to highlight specific network features. Recent studies emphasize the importance of staircase motifs — equivalent to fans or stars in node-link diagrams — within Biofabric. However, to effectively showcase these motifs, we need to formulate specialized layout algorithms. This paper introduces a method to compute optimal layouts for Biofabric, focusing on maximizing staircase formation. We present an Integer Linear Programming (ILP) model for this task and evaluate its performance in terms of scalability and output quality against a leading heuristic method, Degreecending. Our results demonstrate that the ILP approach identifies significantly more, and often longer, staircases compared to Degreecending, albeit with the trade-off of higher computation times.
Our supplemental material, including a full copy of the paper, code, and results, is available on osf.io.

CCS Concepts

• ***Human-centered computing*** → ***Graph drawings;***

1. Introduction

Biofabric is a relatively recent network visualization technique introduced by Longabaugh in 2012 [Lon12]. In Biofabric, nodes are represented as horizontal lines, while edges appear as vertical lines connecting the respective node lines, creating a structured grid where the endpoints of the vertical edges correspond to the non-zero entries of the incidence matrix of the underlying graph. Biofabric found use in the visualization of biological pathways [PF15], social networks [CPLBO*20], and more. Biofabric offers an alternative approach to network visualization, similar to node-link and adjacency matrix methods. Like these methods, Biofabric relies heavily on an optimized layout to maximize its effectiveness. Indeed, a well-optimized layout is essential for Biofabric to effectively convey the properties of the data and enhance interpretability.

† sdb@ac.at.tuwien.at

As shown in Figure 1, a carefully designed layout can highlight key patterns and structures, such as motifs or relationships, that might remain obscured in other visualization techniques.

Biofabric offers several advantages over traditional node-link diagrams and adjacency matrices. Unlike node-link diagrams, it eliminates the issue of edge crossings, which often obscure dense networks. Compared to adjacency matrices, Biofabric provides a more intuitive mapping of node-to-node relationships, making it easier to trace connections. These benefits make Biofabric suited for visualizing densely connected networks or hierarchical structures, where clarity and interpretability are critical [VBP*21b].

In this regard, this visualization technique constitutes an interesting layout problem: although it might be considered a hybrid between an adjacency matrix and a node-link diagram, algorithms developed for these traditional methods cannot be directly applied to Biofabric without significant adaptation and reformulation. Indeed, creating a good layout for Biofabric involves establishing both a node ordering and an edge ordering – as opposed to adjacency matrices, where only node ordering is typically considered, and node-link diagrams, which offer far greater degrees of freedom in positioning nodes and edges. These constraints make Biofabric layout optimization a unique problem.

Previous research on Biofabric has attempted to create layouts that optimize for the formation of *staircases*: visual motifs in the network that are particularly easy to interpret when represented using Biofabric. These staircases correspond to *fans* or *stars* in the underlying graph structure – subgraphs where a high-degree node connects to many others. Staircase patterns are of interest because they align with Gestalt principles such as *proximity*, *continuity*, and *similarity*, which aid in human pattern recognition and improve the readability of complex networks. For these motifs to stand out, nodes and edges must be arranged so that the edgecgfs constituting the star form triangular structures. Achieving this requires a careful arrangement of nodes and edges, with a layout algorithm identifying the optimal configuration to maximize the number and length of staircase structures. To the best of our knowledge, previous attempts at developing layout algorithms for Biofabric have relied exclusively on heuristic approaches [FFH*25], leaving room for more systematic and robust methods to improve these layouts further.

The purpose of this paper is to provide **an optimal solution for the arrangement of nodes and edges in Biofabric to maximize the creation of staircases**. Based on a quality function for staircases (Section 3.1), the solution we provide creates an arrangement of nodes and edges that results in the highest overall staircase quality. The method we propose uses Integer Linear Programming (ILP), a common choice in the formulation of optimal solutions for combinatorial layout problems. The challenge in applying ILP lies in designing effective linear constraints that accurately capture the problem's requirements. Detailed descriptions and explanations of this process are provided in Section 3. Even though the method we provide is more computationally expensive than a heuristic (solving an ILP problem is, indeed, NP-complete), the relevance of having a method to provide optimal solutions relies on its important role in the future creation and validation of more refined, faster heuristics, as discussed later in this same section.

Overall, our contributions can be summarized as:

- An optimal formulation for arranging nodes and edges in Biofabric to maximize the number and quality of staircase motifs.
- The results of a computational evaluation for exploring the scalability and quality improvement of our proposed formulation, as compared to another state-of-the-art heuristic method.

We believe that developing exact layout optimization algorithms is essential for unlocking the full potential of this relatively new network visualization technique. A poorly organized or unsorted visualization obscures the strengths of Biofabric, while a well-structured layout allows its capabilities to shine. Our primary **objective** is not only to provide an **optimal arrangement** for Biofabric but also to **facilitate the development of new heuristics** that can extend its applicability to larger and more complex networks. Additionally, Biofabric layout algorithms encapsulate an exciting problem that we hope will speak to the curiosity of the graph drawing field: the simultaneous need to arrange both edges and nodes is a fascinating challenge that doesn't appear in other forms of network visualization.

All of our supplemental materials, including code to test the ILP formulation and results, are distributed on osf.io.

> *Motivation:* Optimal results are really important for graph layout algorithms: not only do they produce the best-quality graph drawings as defined by a specific quality function, but they are also very important to determine *baselines* in order to evaluate other heuristics (perhaps more scalable than exact methods). Indeed, albeit slower and computationally more expensive, exact methods produce a solution whose results can't be improved in terms of the chosen quality function. Heuristics, instead, focus more on scalability and try to attain the same objectives without striving for complete optimality. Thus, the choice between an exact method or a heuristic is a question of which tradeoff to pick. In order to evaluate how far away the solutions given by a heuristic fall from optimality – thus, in order to evaluate how big the tradeoff is – it is useful to compare it against a baseline of previously computed optimal results. This is why collections of optimal results for popular benchmark datasets are proposed – an example is Chimani and Wiedera [CW16] computing the exact crossing number for node-link visualization for Rome-Lib, or Clancy et al. [CHN19] publishing a survey of graphs with known crossing numbers. We maintain that layout is extremely important to highlight the strengths of a network visualization method and make its potential shine, and – being a relatively new and underexplored technique – Biofabric definitely needs research to advance in terms of layout methods. Publishing a method to create optimal solutions for Biofabric not only produces the best results layout-wise (as measured by the objective function), but it also fosters the development of other methods, providing a baseline to compare them against.

2. Background

In recent years, there has been a growing interest in exploring Biofabric as an alternative to traditional visualization methods like node-link diagrams and adjacency matrices. Each of these techniques has its own strengths and limitations, making it important to determine which tasks are best suited for each visualization style.

For instance, adjacency matrices are effective for quickly identifying the direct neighbors of a node but can make it challenging to trace paths through a network [GFC04, HF06]. In contrast, Biofabric employs a unique grid-like structure that addresses specific issues found in node-link visualizations. As Longabaugh notes in his original paper, Biofabric is resistant to the formation of *hairballs* [Lon12] — a common problem in densely connected networks where edges become so entangled that individual connections are nearly impossible to distinguish.

The design space of Biofabric introduces new and intriguing possibilities. While Nobre et al. [NMSL19] proposed Biofabric as a platform for multivariate data representation, Fuchs et al. [FDH*24] expanded on this by demonstrating how multivariate data can be encoded in numerous ways, either along the sides or embedded directly within Biofabric. Additionally, the work of Valdivia et al. [VBP*21a, DBPB*22] on PAOHvis, a system similar to Biofabric for hypergraph representation, highlights the adaptability and versatility of this design approach.

Beyond its design potential, Biofabric presents a fresh challenge in the development of layout algorithms: the simultaneous sorting of both nodes and edges. To address this challenge, we can draw inspiration from layouts traditionally designed for node-link diagrams and adjacency matrices. The process of sorting nodes along a single axis in Biofabric is reminiscent of similar problems in other visualization techniques, such as arc diagrams [Wat02], circular layouts [GK07], or book embeddings [GWYM22, KMN18], or in general any kind of orderable layout [ANMMG24]. These algorithms focus primarily on reducing edge crossings, which significantly impact the readability of node-link diagrams [Pur02, DRSM15]. However, Biofabric eliminates the issue of edge crossings entirely — edges do not intersect in this system. As a result, while existing methods can provide inspiration, they require adaptation to address new objectives and metrics that align with the unique structure and requirements of Biofabric.

To define our optimization objective for a Biofabric layout algorithm, we draw from previous research — a field still in its early stages but steadily gaining attention. In Longabaugh's original paper [Lon12], layout design is not explicitly discussed — however, the preferred ordering displayed in the sample figures suggests sorting nodes by their degree and edges by the maximum degree of their endpoints. This approach naturally produces triangular structures in the embedding, which highlight high-degree nodes and their connections. These triangular patterns can be likened to *motifs*, a key focus in node-link visualizations [DS13]. The perception of such visual structures is heavily influenced by the Gestalt principles of *proximity*, *similarity*, and *continuity*. Building on the significance of motifs, Fuchs et al. [FFH*25] have recently advanced this topic by developing heuristic methods aimed at producing *staircases* — triangular arrangements that correspond to *fans* or *stars* in node-link visualizations, as illustrated on the right of Figure 2. Additionally, it is worth considering techniques used for adjacency matrices [BBHR*16, BBK*18] and Massive Sequence Views [vdEHBvW13], albeit the motifs in these structures differ from those in Biofabric.

Differently from the methods described above, the method presented in this paper uses Integer Linear Programming (ILP). Mathematical programming is commonly used to solve combinatorial problems in the context of graph drawing and network visualization. ILP and Integer Quadratic Programming (IQP) models have been used to tackle layout problems such as circular vertex orders [NW24], bend minimization [NR20], or computing metro map-like layouts [JWKN21]. Specifically interesting in the context of this work are linear ordering problems. In two-layer drawings of networks, each layer is represented as a linear order of assigned vertices. Models for crossing minimization [JM95, JM97] represent the order as a binary vector of variables for each vertex. If the order in one layer is fixed, the objective function consists of linear terms but is quadratic otherwise. Experimental evaluations have demonstrated that modern solvers can solve moderately-sized instances (300 nodes) to optimality within seconds. However, the solution presented by Wilson et al. [WPC*25] uses layered graphs, where the number of combinations is distributed across multiple layers. This approach makes it difficult to compare their results directly to the performance achievable with Biofabric, which does not use such layering. Similarly, the ordering of rows and columns of matrices can be considered a linear ordering problem. For example, column ordering of incidence matrices has been optimized via ILP formulations [JWKN21]. However, if only neighboring columns have been considered during ordering, then this problem can often be reformulated as a traveling-salesperson problem (TSP) [DN22, WDN23]. Even though TSP instances can be formulated as ILPs, much more efficient solvers such as Concorde [con20] exist. However, it is important to note that Concorde is specifically designed to solve TSPs and cannot currently be applied to our case, as our problem does not reduce to a TSP (though it is an exciting prospect for future exploration if a reduction can be formulated). The formulation that we present in this paper builds upon earlier works on linear ordering problems in network visualization. In contrast to matrix ordering problems, we cannot base our formulation on only considering row (vertex) or column (edge) order.

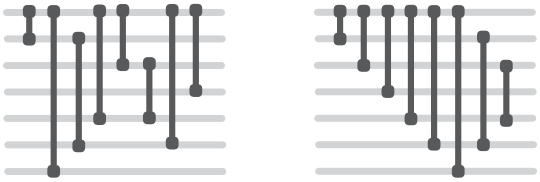


Figure 2: *Left: A Biofabric visualization with no particular layout applied to it. Right: The same graph, where edges have been ordered based on the maximum degree of their endpoints. The right visualization offers more immediate insights into the structure of the graph: one central node (the first row) is highly connected to all the other nodes of the graph, in a fan motif, also called a* ***staircase*** *in Biofabric. While this is a simple case with a very small network only drawn for the purpose of this explanation, more complicated graphs require appropriate layout algorithms.*

> *Degreecending:* A significant portion of this paper evaluates the performance of our proposed method against a leading heuristic known as **Degreecending**, recently introduced in the work of Fuchs et al. [FFH*25]. Given its central role in our comparisons, it is important to describe how it operates. In

their study, the authors explore several approaches for ordering nodes and edges, drawing inspiration from techniques used in linear graph layouts and adjacency matrices. They evaluate these methods on their ability to produce staircases. The node-ordering methods tested include force-directed layouts [FR91], reverse Cuthill-McKee (RCM) [CM69], barycentric ordering [MS05, GKNV93], simulated annealing [BKS08], and degree-based sorting. The edge-ordering methods include sorting by length, by maximum degree, or by Degreecending. In Degreecending, edges are grouped according to their higher endpoint — thus, grouped by the maximum degree of their endpoints. Within the same group, they are sorted according to the degree of their other endpoints. Ultimately, the authors conclude that sorting nodes by degree as a first step, and then sorting edges through Degreecending is the most effective method for generating staircases. In the rest of the paper, whenever we mention the use of Degreecending, we are always going to assume a node ordering based on degree.

3. Formulation

This section provides a detailed explanation of the components that make up the ILP formulation. Together, these components work to determine the optimal configuration of nodes and edges for maximizing the number and quality of staircases in Biofabric. A summary table explaining the notation used throughout this section is provided in Table 1.

*Nodes are named with latin letters (**a**, **b**, **c**, ...),*
Edges are named with greek letters (α, β, γ, ...)

N	The set of all nodes
E	The set of all edges
$x_{\alpha,\beta} \in \{0,1\}$	The relative position of edges α and β. $x_{\alpha,\beta} = 1$ if α comes before (to the left of) β, 0 otherwise.
$y_{a,b} \in \{0,1\}$	The relative position of nodes a and b. $y_{a,b} = 1$ if node a is above node b, 0 otherwise.
$z_{\alpha,\beta} \in \{0,1\}$	Indicates if two edges are consecutive. $z_{\alpha,\beta} = 1$ if edges α and β are consecutive, 0 otherwise.
$c_{\alpha,\beta,\gamma} \in \{0,1\}$	Indicates if the three edges α, β and γ form a staircase.
$p_\alpha \in \mathbb{N}$	Indicates the integer position of edge α from the left of the visualization. This is used to determine if two edges are consecutive.
$I(a)$	Indicates the set of edges incident to node a.

Table 1: *Definition of variables used in the formulation.*

3.1. Quality of staircases

To determine what constitutes a good Biofabric drawing of a network, we must define a quality function. Our approach builds upon the quality function introduced by Fuchs et al. [FFH*25], which was previously used to evaluate the effectiveness of their heuristic-based layout algorithm. Let us start with some definitions:

- Two edges are **consecutive** if they lay next to each other on a Biofabric visualization.
- A **staircase** is defined as a sequence of consecutive edges, sharing one of the endpoints while their other endpoints are aligned vertically, increasing or decreasing in a monotonic fashion. We will call the shared endpoint the *center* of the staircase. Previous research [FFH*25] has allowed for staircases constituted by, at minimum, three edges, a constraint we retain in this work. There is no upper limit on the number of edges that can form a staircase.
- The **length** of a staircase is defined as the number of edges that constitute it.
- A **complete staircase** is one that has a length equal to the degree of its center. In this way, all the connections for a specific node are displayed clearly.
- **Density** is the ratio of existing edges to the total possible edges if the graph was fully connected, defined as $2|E|/(|V|^2 - |V|)$.

Our objective is to maximize the number of staircases while ensuring they are as close to complete staircases as possible. Fuchs et al. [FFH*25] define their quality function as a product of the *completeness* of a staircase (or, how close to a complete staircase it is, defined as the ratio between the number of edges making up the staircase and the degree of the node at its center) and the *smoothness* of a staircase (an inverse metric aimed at minimizing the vertical distance between the endpoints of consecutive edges in the staircase, ensuring a more visually coherent structure).

Fuchs et al. use the following formula as a quality metric for evaluating staircase completeness, which we report verbatim:

$$q_{staircases} = \sum_{\text{staircases}} \left(\frac{\#steps + 1}{degree} \right)^2 \tag{1}$$

The $\#steps + 1$ element corresponds to the length of a staircase. This means that the best quality embedding is the one that has the highest number of staircases, and it is as close to being complete staircases as possible.

A note about the difference in quality functions: The quadratic nature of Equation (1) makes it so that we cannot use it in our linear programming formulation. Instead, we replace it with a similar — but not equal — function (Equation (2)), used as an optimization objective for the ILP. Both functions ultimately optimize for the same general objective: obtaining the highest number of staircases, as complete as possible. However, Equation (1) does it by elevating completeness to the power of 2, while Equation (2) does it by summing over triplets of edges constituting a staircase. To avoid confusion and misleading reporting, we used both functions to evaluate the final quality of the results, which can be seen in the last two rows of Table 2. Both rows ultimately end up containing similar results, indicating they are comparable.

It is not always possible to create an embedding that results in all staircases being complete, as the ordering of nodes and edges needed to create one staircase might prevent the creation of another one. This happens especially if staircases share several member nodes. Thus, even in an optimal solution, it might be that not all staircases can be drawn as complete staircases. When this is the case, longer staircases should be preferred over short ones. On the upside, though, this information can be used for our own benefit to reduce the cost of computing the layout by excluding from the computation conflicting staircases (see Section 3.6).

3.2. Objective function

The objective is to maximize the number of edges that participate in staircases. To this objective, we define a variable $c_{\alpha,\beta,\gamma}$ that indicates if edges α, β, and γ participate together as a consecutive triplet in a staircase. The objective function is written as such:

$$\text{Maximize} \sum_{\substack{\forall \alpha,\beta,\gamma \in E \\ \alpha \neq \beta \neq \gamma \\ \alpha \cap \beta \cap \gamma \neq \varnothing}} c_{\alpha,\beta,\gamma} \tag{2}$$

Note that this creates $O(|E|^3)$ variables if applied naively. There are many cases in which it does not make sense to define such a variable for a triplet of edges because they can never form a staircase — for instance, if there is no shared endpoint between all three of them. Several of these cases are explained in Section 3.6. Still, even after optimization, the number of variables and constraints of our model are dependent on the number of edges in the network with $O(|E|^3)$.

3.3. Edges forming a staircase

We need to define how the position of nodes and edges is tied to the existence of staircases. We rely on two basic ideas that are the fundamental requirements for a staircase among a triplet of edges:

- They need to be consecutive.
- The endpoints they don't share need to be positioned in a downward or upward sequence in a monotonic fashion.

For the first point, we rely on a variable, $z_{\alpha,\beta}$, which is equal to 1 if α and β are consecutive, 0 otherwise. We will better discuss this variable later, but in the meantime, we can define that $c_{\alpha,\beta,\gamma}$ can only be 1 if two pairs of edges in the triplet are consecutive to each other, forming a consecutive sequence α, β, γ:

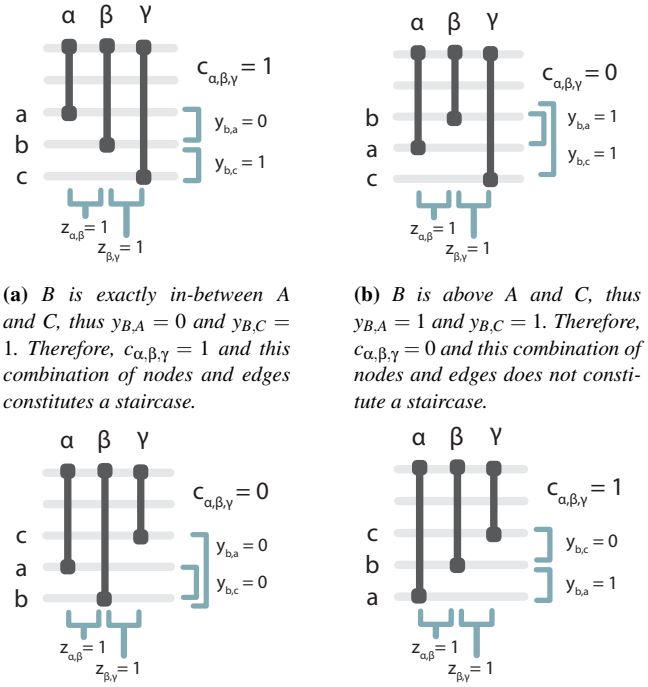
$$\begin{aligned} c_{\alpha,\beta,\gamma} - z_{\alpha,\beta} &\leq 0 \\ c_{\alpha,\beta,\gamma} - z_{\beta,\gamma} &\leq 0 \end{aligned} \qquad \forall \alpha,\beta,\gamma \in E \tag{3}$$

The next step we need is to define how the relationships among the vertical positioning of the nodes constitutes a staircase. We rely on variables $y_{a,b}$, indicating the relative position of nodes in a staircase: $y_{a,b} = 1$ if node a is positioned above node b, 0 otherwise. In the equation below, a, b, and c are used to indicate the endpoints of the edges in a triplet of edges *except* the center (see Figure 3). In order to obtain a monotonic order, it is sufficient to say that the sum of relative positions of the middle edge has to be equal to 1. We do it as such:

$$\begin{aligned} c_{\alpha,\beta,\gamma} - y_{b,a} - y_{b,c} &\leq 0 \\ c_{\alpha,\beta,\gamma} + y_{b,a} + y_{b,c} &\leq 2 \end{aligned} \qquad \begin{array}{c} \forall \alpha,\beta,\gamma \in E, \\ \alpha \in I(a), \beta \in I(b), \gamma \in I(c) \end{array} \tag{4}$$

These two sets of constraints ensure that $c_{\alpha,\beta,\gamma}$ is only equal to 1 if edges α,β,γ and their endpoints respect the previously stated conditions for the formation of a staircase. The fact that the objective function maximizes the sum of c variables, and the fact that they are defined as Boolean variables, allows us to only have to define constraints where c has to be 0 if some conditions are not respected — in all other cases, c will be set to 1 by the solver.

Longer staircases are encouraged by overlapping triplets. The score given to every full staircase is basically its length $\ell - 2$, encouraging the construction of longer staircases.



(a) *B is exactly in-between A and C, thus* $y_{B,A} = 0$ *and* $y_{B,C} = 1$. *Therefore,* $c_{\alpha,\beta,\gamma} = 1$ *and this combination of nodes and edges constitutes a staircase.*

(b) *B is above A and C, thus* $y_{B,A} = 1$ *and* $y_{B,C} = 1$. *Therefore,* $c_{\alpha,\beta,\gamma} = 0$ *and this combination of nodes and edges does not constitute a staircase.*

(c) *B is below A and C, thus* $y_{B,A} = 0$ *and* $y_{B,C} = 0$. *Therefore,* $c_{\alpha,\beta,\gamma} = 0$ *and this combination of nodes and edges does not constitute a staircase.*

(d) *B is between A and C, thus* $y_{B,A} = 1$ *and* $y_{B,C} = 0$. *Therefore,* $c_{\alpha,\beta,\gamma} = 1$ *and this combination of nodes and edges constitutes a staircase.*

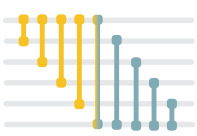
Figure 3: *How do Equation (3) and Equation (4) apply to different cases? The figures above maintain the same order of edges but show how different relative order of nodes affect the objective. It is important that if B is the endpoint corresponding to the center of a possible staircase, then its relative positions with the other endpoints sum up to 1.*

3.4. Defining consecutiveness

One of the most important variables in our formulation, z, defines consecutiveness between two edges. Consecutiveness can be defined and constrained in multiple ways. In our case, we use the relative positions of the edges through Boolean variables x to establish if two edges are consecutive and an auxiliary integer variable p that indicates the index of the position of a given edge. x variables work in a similar way to y variables: $x_{\alpha,\beta} = 1$ if α comes before β, 0 otherwise. The intuition here is that the index p_α of an edge α can be calculated as the sum of the relative positions of all the edges leading up to α, as such:

$$p_\alpha = \sum_{\beta \in E} x_{\beta,\alpha} \qquad \forall \alpha \in E \tag{5}$$

The second intuition, tying together z and p, is that two edges are consecutive if their positions differ by 1. Intuitively, we would write $z_{\alpha,\beta} = 1$ if $|p_\alpha - p_\beta| = 1$. However, we cannot calculate an absolute value in linear programming, as it would not be a linear constraint.



(a) *On the left, the edge in the middle participates in two staircases. If we wanted to avoid this configuration, we could add the constraint* $\sum_{\alpha,\beta,\gamma \in E, \alpha \neq \beta \neq \gamma} c_{\alpha,\beta,\gamma} = 1$*, declaring that every edge can only be in one staircase at once.*

(b) *On the left, one of the edges of the staircase is longer than the others.*

(c) *On the left, the staircase crosses over its own central node.*

Figure 4: *Admissible staircases*

Fortunately, we can linearize the equation using big-M constraints:

$$\begin{aligned} p_\alpha - p_\beta - M(1 - z_{\alpha,\beta}) &\leq 1 \\ p_\alpha - p_\beta + M(1 - z_{\alpha,\beta}) &\geq -1 \end{aligned} \qquad \forall \alpha, \beta \in E \tag{6}$$

M is an arbitrarily large number, larger than the maximum value that any of the variables involved could have. In our case, we define M as $|E| + 1$, as no edge can ever have a position that has an index beyond the total number of edges. Also, in order to reduce the number of variables, we can keep in mind that $z_{\alpha,\beta} = z_{\beta,\alpha}$.

3.5. Transitivity constraints

Transitivity constraints are fundamental in order to make relative positions work: we need to explicitly define that, for instance, if a comes before b, and b comes before c, then a must come before c. This allows us to establish an ordering among elements in a row. We establish transitivity constraints among all triplets of nodes:

$$\begin{aligned} x_{a,b} \quad + x_{b,c} \quad - x_{a,c} \quad &\geq 0 \\ -x_{a,b} \quad - x_{b,c} \quad + x_{a,c} \quad &\geq -1 \end{aligned} \qquad \forall a, b, c \in N \tag{7}$$

And all triplets of edges:

$$\begin{aligned} y_{\alpha,\beta} \quad + y_{\beta,\gamma} \quad - x_{\alpha,\gamma} \quad &\geq 0 \\ -y_{\alpha,\beta} \quad - y_{\beta,\gamma} \quad + y_{\alpha,\gamma} \quad &\geq -1 \end{aligned} \qquad \forall \alpha, \beta, \gamma \in E \tag{8}$$

3.6. Improving efficiency and scalability

The number of variables can quickly grow, causing the solution time to increase exponentially with the graph's size and structure. Fortunately, we can reduce the number of paths the solver must explore by removing elements or adding constraints that simplify trivial cases. Well-chosen constraints shrink the feasible solution space, thus speeding up computation—provided they are not redundant. All the following tricks are *not necessary* to obtain an optimal solution but speed up the computation time.

Exclude edges that have both endpoints with a degree less than the minimum staircase length. Certain edges can never be part of a staircase (Figure 5). This is the case for edges whose both endpoints have a degree that is less than the minimum allowed staircase length (in our case, 3). These edges can be excluded from the computation, as including them cannot change the amount or length

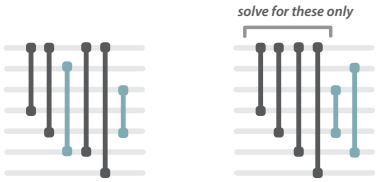


Figure 5: *The two edges highlighted in blue have both their endpoints of degree less than 3. Both these edges will never participate in a staircase, regardless of their position, thus we may completely* ***exclude*** *them from the problem, and add them back to one of the sides of the visualization after computing the solution with a reduced set of edges.*

Figure 6: *The edge in red is contested among different staircases.*

of staircases in the final result. Note that this is not the same for nodes: even though some nodes can never be the center of staircases, they might contribute to one or more.

One edge can participate in a maximum of two triplet-forming staircases, three in case they all share the same center node. Some edges can be contested among different staircases. If the unit staircase is a triplet of edges, then one edge can only belong to a maximum of three triplets.

$$\forall a \in N, \alpha \in I(a) \sum_{\substack{\beta,\gamma \in I(a) \\ \alpha \neq \beta \neq \gamma}} c_{\alpha,\beta,\gamma} \leq 3 \tag{9}$$

Thus, this says that for each edge α incident to a node a, there can be a maximum of three triplets that include that edge and form a staircase, provided that the two other edges in the staircase are incident to a as well. See the rightmost example in Figure 6.

If the nodes at the center of the triplets are not the same, an edge can belong to a maximum of two triplets instead, as the only position where it would be able to form two staircases simultaneously is if it is positioned at one of the sides of both of them. (see the leftmost and center images in Figure 6 for an example).

$$\forall a, b \in N, a \neq b, \quad \alpha \in I(a) \cap I(b) \qquad \sum_{\substack{\beta,\gamma \in I(a) \\ \delta,\varepsilon \in I(b)}} c_{\alpha,\beta,\gamma} + c_{\alpha,\delta,\varepsilon} \leq 2 \tag{10}$$

The equation above says that for every edge α that has endpoints a and b, there can be a maximum of two staircases involving any two other edges incident to a and any two other edges incident to b.

In some cases, we can assume that a staircase can always be formed. In certain conditions, it is trivial to form a staircase, as the edges that would create it have no conflicts with other elements of the networks. Figure 7 illustrates one such instance. This can happen when: Three edges share an endpoint, their other endpoints

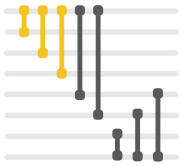


Figure 7: *The yellow edges form a section of a staircase that does not create any other conflict in the graph: moving the nodes so that they form a staircase for those edges is not going to prevent the formation of any other staircase. Thus, we can just force into the model that that one staircase can be created.*

have degree 1 *or* the other endpoints can never form a staircase. **Relative positions can be defined only once.** If node a comes before node b, then automatically node b comes after node a. In ILP constraints, this translates to saying $x_{a,b} = 1 - x_{b,a}$. We can do this replacement directly in the constraints we write without ever defining more than one of the two variables. In our formulation, this can be used for x, y, and z variables and allows us to define half the variables for every one of these groups.

4. Experimental setup

We test the quality of our solution against the degreecending method, a heuristic method previously proposed by Fuchs et al. [FFH*25]. The method used for our evaluation is a computational evaluation, as described in Di Bartolomeo et al.'s survey of methods and best practices to evaluate graph layout algorithms [DBCS*24]. We tested the quality of the results on graphs from Rome-Lib [BGL*97], one of the most popular datasets used for benchmarking graph layout algorithms. To generate the results, we used Gurobi [Gur24] as a linear programming solver, a standard tool to use in such evaluations, on a 2020 MacBook with 32 GB RAM and an Intel Core i5-1038NG7 CPU. In total, we tested 4766 graphs from Rome-Lib, all under 50 nodes and all with a maximum degree of 7 or less. These limits were determined based on the scalability of the method. Result statistics from the experiment are presented in Tables 2 and 3, with additional charts in the appendix. We used a timeout of 50 seconds for each run, further discussed in Consideration 3 in the next section. The timeout allowed us to be able to process the entire dataset in reasonable time.

5. Discussion

We reported the results from our experiments in Table 2, regarding both the timing and quality of the results. An additional table, Table 3, reports results, including statistics about runs that timed out. Finally, we include in the appendix (available on osf.io) additional reports on result quality.

In terms of **quality**, our method consistently outperforms Degreecending in every completed computation. This is expected, as the nature of the method used wouldn't allow otherwise. In Table 2, quality is represented in the two bottom rows (and), with the first representing quality according to Equation (1), and the second representing quality according to Equation (2). In both cases, the results obtained through our method outperform Degreecending, with an improvement in quality of 1.72 times with Equation (1) and of 1.56 times with Equation (2). In both rows, we can see that in no instance do we have a ratio under 1, which means that in no case does Degreecending obtain a better solution than our method. In the most fortunate cases for Degreecending, the ratio is 1, meaning that the heuristic used in Degreecending actually managed to obtain a solution that was optimal. However, while our method confirms that the solution found by it is the best possible layout for obtaining staircases, there is no way with a heuristic to confirm that the solution reaches the optimum.

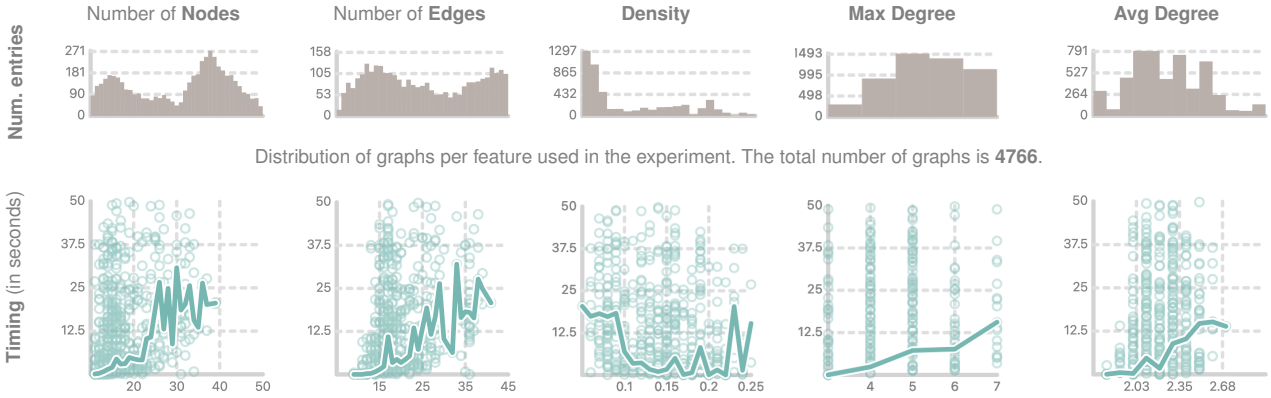
A by-product of computing layouts for staircases is also the creation of runways — another motif that can be formed on Biofabric [FFH*25]. As we can evaluate the quality of runways as well, even though they're not as relevant for the paper, we included the results for runways in osf.io.

The results obtained through this method can be used as a baseline for heuristic benchmarks, as explained in the motivation section at the beginning of the paper.

Although the point of the approach is *optimality*, and not *scalability*, we were still concerned with the size of a network we could handle. In Table 2, timing for each completed run is displayed in cyan. The maximum node degree in a graph has a particularly strong effect on performance: while in nodes with a degree of 3, we only need to take into account one triplet of nodes, there are $O(n^3)$ ways to form staircases when a node has degree n, rapidly expanding the number of variables and constraints in the formulation. Aside from this, growing numbers of edges, nodes, and average degrees also negatively affect performance. In the time limit we established, 50 seconds, we can expect to solve relatively small graphs quickly: up to 20 nodes and 25 edges, with a maximum degree of 4. Larger graphs will require longer times. Similarly, we represent the percentage of timed out graphs in red.

Although they might seem like small numbers, these performances are fairly in line with other ILP problems. Even in recent publications, the graph problems solved through ILP often do not process large graphs in a short time. While our problem might look comparable to a linear ordering of nodes, we have to keep in mind that the number of possible combinations with organizing nodes and edges in the two dimensions that we have greatly expands the difficulty of solving the problem. While it is tempting to compare the problem at hand directly to a linear ordering of nodes, or sorting of nodes in an adjacency matrix, the layout for Biofabric adds an entirely new dimension.

We analyzed the asymptotic complexity of our ILP formulation and report the following dependencies: Equations 3, 4, and 8 have a computational complexity of $O(|E|^3)$, as they depend on the number of edges. Similarly, Equations 5 and 6 exhibit a complexity of $O(|E|^2)$. In contrast, Equation 7 depends on the number of vertices with $O(|N|^3)$ complexity, while Equations 9 and 10 scale with $O(|N|)$ and $O(|N|^2)$, respectively. In summary, both the number of constraints and variables are asymptotically upper bounded by $O(|E|^3)$. Notably, it is sufficient to consider only the number of edges in a single connected component of the graph, as computing a biofabric layout is independent for each component. For graphs with multiple components, the resulting layouts can be juxtaposed. Additional results regarding the number of variables and constraints generated in our experiments can be found in the appendix, available on osf.io.



Distribution of graphs per feature used in the experiment. The total number of graphs is **4766**.

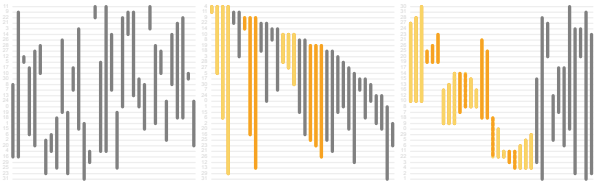
What element influences computation the most, and what kind of graphs can we hope to solve in a reasonable timeframe? The charts above show how long it took for graphs to be computed. A combination of maximum degree, average degree, and number of nodes and edges influence the computation, making graphs progressively harder to solve. Any graphs whose solution hits the timeout boundary we set for the computation are excluded from the charts above (a choice that is further discussed in Section 5) but presented in the next row of charts.

*How many graphs per feature could not complete the computation in the allotted timeframe of **50** seconds?* The charts show the percentage of instances that could not complete the computation, with 1 being 100% of instances going unfinished. The results show similar trends to the chart above: an increment in **maximum degree** and/or **average** degree is correlated to a high timeout rate. Although the computation of these graphs cannot be completed, we can still obtain a partial solution which still constitutes a good solution — albeit non-optimal. See Consideration 4 for further discussion.

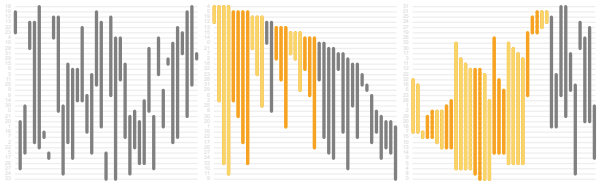
How much improvement in quality does our method provide over Degreecending? The charts above represent the ratio in quality of the results using our method and Degreecending. A ratio of 2 means that the quality produced using our method was twice as much as the one using Degreecending (using Equation (1) as a quality metric). Overall, on average, our method creates results with **1.72 times** the quality of Degreecending.

The charts above present very similar results to the previous row, but instead use Equation (2) as a way to measure quality of the results. Measuring quality this way, the quality produced with our method is overall, on average, **1.56 times** the one produced by Degreecending.

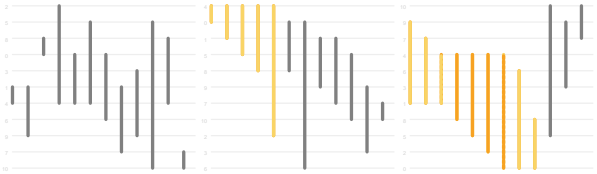
Table 2: *The chart above represents results for various metrics on the rows (Number of Entries, Timing, Percentage of Timeouts, Quality ratio according to Equation (1) and Quality ratio according to Equation (2)). The different columns divide the charts, organizing results by number of nodes, number of edges, density, maximum degree of the graphs, and average degree. While a circle on a chart represent an individual result, a line represents the median of the results obtained by binning results by their value on the x-axis (e.g., in the leftmost row, the line represents the median for all the graphs with 10 nodes, 11 nodes, 12 nodes...).*



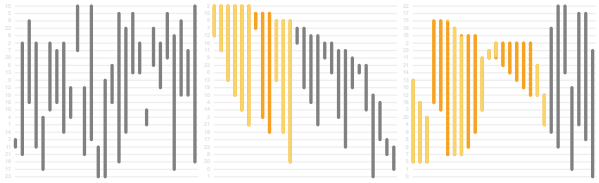
(a) `grafo1953.32`. *While Degreecending finds 4 complete staircases (quality: 4), our method actually reveals 9 complete staircases (quality: 9).*



(b) `grafo5663.34`. *While Degreecending finds 6 complete staircases (quality: 6), our method reveals 11 complete staircases (quality: 11).*



(c) `grafo1131.11`. *While Degreecending finds 1 complete staircase (quality: 1), our method actually reveals 3 complete staircases (quality: 3).*



(d) `grafo1862.24`. *While Degreecending finds 3 complete staircases (quality: 3), our method actually reveals 7 complete staircases (quality: 7).*

Figure 8: *Examples of graphs from the Rome-Lib dataset illustrate a selection of graphs with diverse features and varying numbers of nodes and edges. In each subfigure, the leftmost image shows the original graph without any specific ordering, the center image shows the graph ordered using the Degreecending heuristic, and the rightmost image shows the results obtained with the ILP method proposed in this paper. Different colors (,) represent individual staircases, while dashed, multicolored lines ( ) indicate edges shared among multiple staircases.*

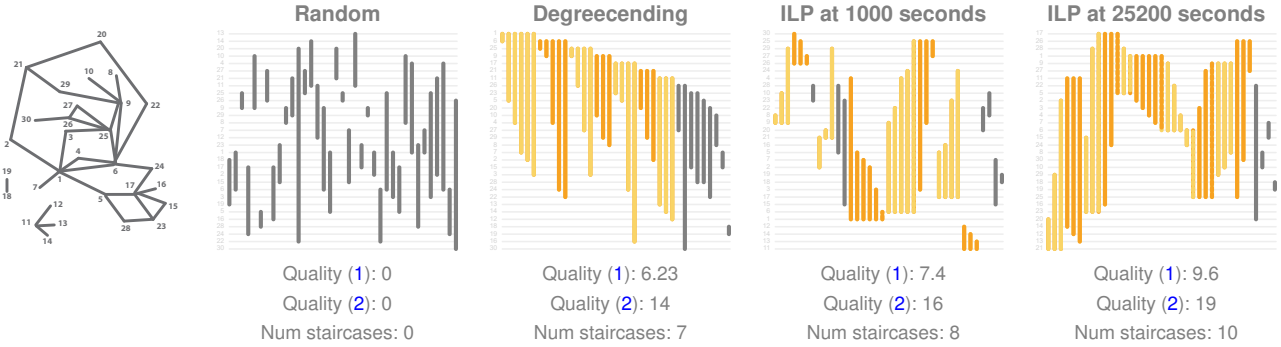


Figure 9: *This case study is based on a real-world dataset from the Network Repository [RA15], specifically, a dataset representing interactions within a colony of ants [QHBH15]. First, the example highlights how the quality of partial solutions evolves over time, even before the computation is fully complete. The figure compares results obtained at 1,000 seconds and after 25,200 seconds (equivalent to 7 hours). Second, it illustrates the limitations of Degreecending. While the repeating patterns created by Degreecending may appear visually orderly, they obscure important structures within the graph. In particular,* ***Degreecending leaves some edges outside any motif, giving the impression that these edges are more disconnected than they actually are****. In contrast, the ILP approach reveals more staircases, showing that far fewer edges remain unconnected to motifs, conveying that the graph is more densely connected than the Degreecending output implies.*

Consideration 1 — Differences with Degreecending Obtaining an optimal solution makes evident some issues that can be created by Degreecending. While it is true that Degreecending is much more scalable and can solve larger graphs in much less time (Degreecending finishes its computation for each one of the graphs in the dataset in under 10 milliseconds), the insights revealed by the optimal solutions we provide can help further develop scalable algorithms that address such issues. For instance, if we look at the example presented in Figure 9, we can see how Degreecending shows a number of edges that appear fairly disconnected from the rest of the graph on the right and do not participate in any motif. In the result of the ILP on the right, instead, it is immediately apparent that much fewer edges are not participating in motifs, and more internal structure in the graph is revealed. The different staircases also appear more disconnected from each other. While the repeating pattern created by Degreecending might look visually appealing, it ends up hiding some motifs. In Figure 8, it is apparent how Degreecending struggles, particularly when staircases are densely connected between themselves. Two densely connected staircases would correspond to two high-degree nodes that share a large number of neighbors. In particular, Figure 8a even shows that this causes some staircases in the middle of the visualization to appear completely disconnected because their centers share most

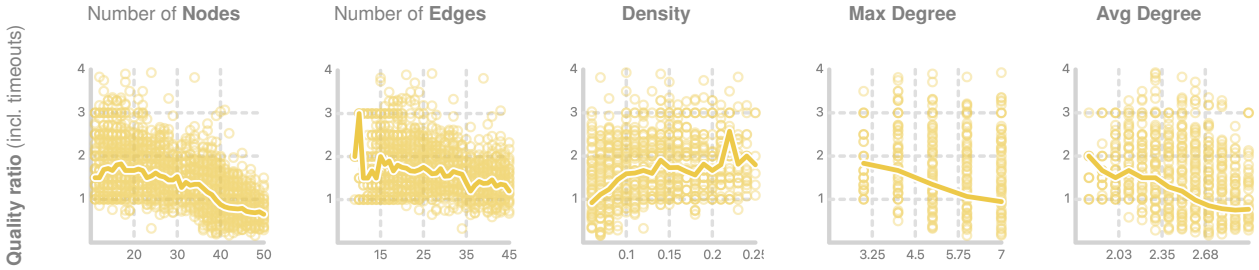


Table 3: *These charts illustrate the quality of the computation results* ***including those that timed out****. Essentially, this represents how our method performs, setting aside the "optimality" criterion and viewing it as a heuristic for producing "good enough" results. The charts display the ratio of the quality of results obtained by our method compared to those from Degreecending. Like in Table 2, each dot represents a single graph in the dataset, while the line shows the median results, binned by the x-axis attribute. More complex graphs tend to yield lower-quality results—for instance, graphs with more than 38 nodes, as shown by the median line going below 1 for those instances. Allocating more time per graph will improve the quality of these outcomes.*

neighbors with higher degree nodes. This is not an issue with the ILP-produced layout.

It can be argued that shared edges ( ) between different staircases appear in the ILP solution, but never in the Degreecending solution. However, sharing an edge is not prohibited in a staircase — they are just rare in Degreecending due to its sorting strategy.

Consideration 2 — Why do graphs with lower density take longer? In Table 2, most of the charts show a generally monotonic increase as the x-axis variable grows, but density behaves differently. Interestingly, graphs with lower density (as defined in Section 3.1) tend to produce the most timeouts and require the longest processing times. At first glance, this might seem counterintuitive — after all, given an equal number of nodes, a denser graph contains more edges. However, in the Rome-Lib dataset, larger graphs are more likely to have lower density. This uneven distribution introduces a bias, making it more common for computationally expensive cases to be associated with sparse graphs. Consequently, the results reflect this underlying peculiarity, making it more likely that the most computationally expensive cases correspond to graphs with lower density.

Consideration 3 — A discussion about timeouts. In the reporting of the results in Table 2, we omit the results that took over 50 seconds, which was the total timeout we established for each individual computation. This is due to multiple reasons: when trying to make considerations about timing, a computation that timed out after x seconds cannot be considered as having taken x seconds — as it would clearly have taken more — and it cannot be considered as having taken infinite time. This prevents from being able to compute any kind of summary statistic over it, as the only fact we know about a timed out solution is that it took more than the allotted time. The quality of the final results, as well, cannot be determined for instances that timed out, as a final optimal solution could not be reached. They are therefore excluded from Table 2, but we further discuss what to do with the partial results in Consideration 4. This way of reporting results is the same used in other studies involving ILP approaches (such as [CGMW10, dBRGD22, CMB08]).

Consideration 4 — Analyzing partial results. Even when a solver times out, there's still value: modern ILP solvers like Gurobi automatically return the best solution found so far. Although optimality is no longer guaranteed, these partial solutions remain valid for Biofabric layout and can be meaningfully compared to our baseline, Degreecending. This allows us to ask: *What is the best solution achievable within a reasonable time?* Table 3 shows that even with a 50-second timeout, ILP often outperforms Degreecending——up to a certain graph size. For larger or more complex instances, Degreecending may outperform the ILP's intermediate results, but given more time, ILP solutions continue to improve (Figure 9).

6. Limitations and Future Work

One limitation of our formulation is that it doesn't explicitly minimize the vertical length of staircases. Shorter, more uniform edges improve visual clarity, and adding length minimization as a secondary ILP objective could enhance layout quality.

While staircases are the primary focus of our optimization, they are not the only motifs of interest in Biofabric layouts. Fuchs et al. [FFH*25] also discuss motifs such as cliques, paths, and connectors, though their approach does not optimize for them. Expanding the current formulation to include these additional motifs represents a possible direction for future research. Important future work could be establishing a hierarchy of importance among different motifs and developing a strategy to incorporate them into the existing optimization framework.

Scalability remains a key challenge. Our ILP is effective for moderately sized graphs, but larger instances require more efficient formulations. One possible direction is a TSP-based reformulation, which could better handle the ordering constraints and improve performance——though this remains to be explored.

7. Conclusion

We presented an Integer Linear Programming (ILP) formulation for computing Biofabric layouts optimized for the creation of staircase motifs. The approach provides a structured foundation for producing high-quality layouts. While limited in scalability, we hope that the formulation will aid in the development of new and improved layout algorithms for Biofabric.

Acknowledgements

This research was funded in whole or in part by the Austrian Science Fund (FWF) [10.55776/ESP513]. Open access funding provided by Technische Universitat Wien/KEMÖ.

References

[ANMMG24] AL-NAAMI N., MÉDOC N., MAGNANI M., GHONIEM M.: Improved visual saliency of graph clusters with orderable node-link layouts. *IEEE Transactions on Visualization and Computer Graphics 31*, 1 (Sept. 2024), 1028–1038. doi:10.1109/TVCG.2024.3456167. 3

[BBHR*16] BEHRISCH M., BACH B., HENRY RICHE N., SCHRECK T., FEKETE J.-D.: Matrix reordering methods for table and network visualization. *Computer Graphics Forum 35*, 3 (2016), 693–716. doi:10.1111/cgf.12935. 3

[BBK*18] BEHRISCH M., BLUMENSCHEIN M., KIM N. W., SHAO L., EL-ASSADY M., FUCHS J., SEEBACHER D., DIEHL A., BRANDES U., PFISTER H., SCHRECK T., WEISKOPF D., KEIM D. A.: Quality metrics for information visualization. *Computer Graphics Forum 37*, 3 (2018), 625–662. doi:10.1111/cgf.13446. 3

[BGL*97] BATTISTA G. D., GARG A., LIOTTA G., TAMASSIA R., TASSINARI E., VARGIU F.: An experimental comparison of four graph drawing algorithms. *Computational Geometry 7*, 5-6 (Apr. 1997), 303–325. doi:10.1016/s0925-7721(96)00005-3. 1, 7

[BKS08] BRUSCO M. J., KÖHN H.-F., STAHL S.: Heuristic implementation of dynamic programming for matrix permutation problems in combinatorial data analysis. *Psychometrika 73*, 3 (2008), 503–522. doi:10.1007/s11336-007-9049-5. 4

[CGMW10] CHIMANI M., GUTWENGER C., MUTZEL P., WONG H.-M.: Layer-free upward crossing minimization. *ACM J. Exp. Algorithmics 15* (mar 2010). doi:10.1145/1671970.1671975. 10

[CHN19] CLANCY K., HAYTHORPE M., NEWCOMBE A.: A survey of graphs with known or bounded crossing numbers. *Australas. J Comb. 78* (2019), 209–296. doi:10.48550/arXiv.1901.05155. 2

[CM69] CUTHILL E., MCKEE J.: Reducing the bandwidth of sparse symmetric matrices. In *Proceedings of the 1969 24th National Conference* (New York, NY, USA, 1969), ACM '69, Association for Computing Machinery, p. 157–172. doi:10.1145/800195.805928. 4

[CMB08] CHIMANI M., MUTZEL P., BOMZE I.: A new approach to exact crossing minimization. In *Algorithms - ESA 2008* (Berlin, Heidelberg, 2008), Halperin D., Mehlhorn K., (Eds.), Springer Berlin Heidelberg, pp. 284–296. doi:10.1007/978-3-540-87744-8_24. 10

[con20] Concorde Home, Oct. 2020. [Online; accessed 27. Nov. 2024]. URL: https://www.math.uwaterloo.ca/tsp/concorde.html. 3

[CPLBO*20] CAMACHO D., PANIZO-LLEDOT A., BELLO-ORGAZ G., GONZALEZ-PARDO A., CAMBRIA E.: The four dimensions of social network analysis: An overview of research methods, applications, and software tools. *Information Fusion 63* (2020), 88–120. doi:10.1016/j.inffus.2020.05.009. 1

[CW16] CHIMANI M., WIEDERA T.: An ILP-based Proof System for the Crossing Number Problem. In *24th Annual European Symposium on Algorithms (ESA 2016)* (Dagstuhl, Germany, 2016), Sankowski P., Zaroliagis C., (Eds.), vol. 57 of *Leibniz International Proceedings in Informatics (LIPIcs)*, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, pp. 29:1–29:13. doi:10.4230/LIPIcs.ESA.2016.29. 2

[DBCS*24] DI BARTOLOMEO S., CRNOVRSANIN T., SAFFO D., PUERTA E., WILSON C., DUNNE C.: Evaluating graph layout algorithms: A systematic review of methods and best practices. *Computer Graphics Forum n/a*, n/a (2024), e15073. doi:10.1111/cgf.15073. 7

[DBPB*22] DI BARTOLOMEO S., PISTER A., BUONO P., PLAISANT C., DUNNE C., FEKETE J.-D.: Six methods for transforming layered hypergraphs to apply layered graph layout algorithms. *Computer Graphics Forum 41*, 3 (2022), 259–270. doi:10.1111/cgf.14538. 3

[dBRGD22] DI BARTOLOMEO S., RIEDEWALD M., GATTERBAUER W., DUNNE C.: Stratisfimal layout: A modular optimization model for laying out layered node-link network visualizations. *IEEE Transactions on Visualization and Computer Graphics 28*, 1 (2022), 324–334. doi:10.1109/TVCG.2021.3114756. 10

[DN22] DOBLER A., NÖLLENBURG M.: On computing optimal linear diagrams. In *Diagrammatic Representation and Inference - 13th International Conference, Diagrams 2022, Rome, Italy, September 14-16, 2022, Proceedings* (2022), pp. 20–36. doi:10.1007/978-3-031-15146-0_2. 3

[DRSM15] DUNNE C., ROSS S. I., SHNEIDERMAN B., MARTINO M.: Readability metric feedback for aiding node-link visualization designers. *IBM Journal of Research and Development 59*, 2/3 (2015), 14:1–14:16. doi:10.1147/JRD.2015.2411412. 3

[DS13] DUNNE C., SHNEIDERMAN B.: Motif simplification: improving network visualization readability with fan, connector, and clique glyphs. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (New York, NY, USA, 2013), CHI '13, Association for Computing Machinery, p. 3247–3256. doi:10.1145/2470654.2466444. 3

[FDH*24] FUCHS J., DENNIG F. L., HEINLE M.-V., KEIM D. A., DI BARTOLOMEO S.: Exploring the design space of biofabric visualization for multivariate network analysis. *Computer Graphics Forum 43*, 3 (2024), e15079. doi:10.1111/cgf.15079. 3

[FFH*25] FUCHS J., FRINGS A., HEINLE M.-V., KEIM D. A., DI BARTOLOMEO S.: Quality metrics and reordering strategies for revealing patterns in biofabric visualizations. *IEEE Transactions on Visualization and Computer Graphics 31*, 1 (2025), 1039–1049. doi:10.1109/TVCG.2024.3456312. 1, 2, 3, 4, 7, 10

[FR91] FRUCHTERMAN T. M. J., REINGOLD E. M.: Graph drawing by force-directed placement. *Software: Practice and Experience 21*, 11 (1991), 1129–1164. doi:10.1002/spe.4380211102. 4

[GFC04] GHONIEM M., FEKETE J.-D., CASTAGLIOLA P.: A comparison of the readability of graphs using node-link and matrix-based representations. In *IEEE Symposium on Information Visualization* (2004), pp. 17–24. doi:10.1109/INFVIS.2004.1. 3

[GK07] GANSNER E. R., KOREN Y.: Improved circular layouts. In *Graph Drawing* (Berlin, Heidelberg, 2007), Kaufmann M., Wagner D., (Eds.), Springer Berlin Heidelberg, pp. 386–398. doi:10.1007/978-3-540-70904-6_37. 3

[GKNV93] GANSNER E., KOUTSOFIOS E., NORTH S., VO K.-P.: A technique for drawing directed graphs. *IEEE Transactions on Software Engineering 19*, 3 (1993), 214–230. doi:10.1109/32.221135. 4

[Gur24] GUROBI OPTIMIZATION, LLC: Gurobi optimizer reference manual, 2024. URL: http://www.gurobi.com. 7

[GWYM22] GUAN X., WU C., YANG W., MENG J.: A survey on book-embedding of planar graphs. *Frontiers of Mathematics in China 17*, 2 (2022), 255–273. doi:10.1007/s11464-022-1010-5. 3

[HF06] HENRY N., FEKETE J.-D.: Matrixexplorer: a dual-representation system to explore social networks. *IEEE Transactions on Visualization and Computer Graphics 12*, 5 (2006), 677–684. doi:10.1109/TVCG.2006.160. 3

[JM95] JÜNGER M., MUTZEL P.: Exact and heuristic algorithms for 2-layer straightline crossing minimization. In *Graph Drawing, Symposium on Graph Drawing, GD '95, Passau, Germany, September 20-22, 1995, Proceedings* (1995), Brandenburg F., (Ed.), vol. 1027 of *Lecture Notes in Computer Science*, Springer, pp. 337–348. doi:10.1007/BFB0021817. 3

[JM97] JÜNGER M., MUTZEL P.: 2-layer straightline crossing minimization: Performance of exact and heuristic algorithms. *J. Graph Algorithms Appl. 1*, 1 (1997), 1–25. doi:10.7155/JGAA.00001. 3

[JWKN21] JACOBSEN B., WALLINGER M., KOBOUROV S. G., NÖLLENBURG M.: Metrosets: Visualizing sets as metro maps. *IEEE Trans. Vis. Comput. Graph. 27*, 2 (2021), 1257–1267. doi:10.1109/TVCG.2020.3030475. 3

[KMN18] KLAWITTER J., MCHEDLIDZE T., NÖLLENBURG M.: Experimental evaluation of book drawing algorithms. In *Graph Drawing and Network Visualization (GD'17)* (2018), Frati F., Ma K.-L., (Eds.), vol. 10692 of *LNCS*, Springer, pp. 224–238. doi:10.1007/978-3-319-73915-1_19. 3

[Lon12] LONGABAUGH W. J.: Combing the hairball with BioFabric: a new approach for visualization of large networks. *BMC Bioinformatics 13*, 1 (Oct. 2012), 275. doi:10.1186/1471-2105-13-275. 1, 3

[MS05] MÄKINEN E., SIIRTOLA H.: The barycenter heuristic and the reorderable matrix. *Informatica (Slovenia) 29*, 3 (2005), 357–364. 4

[NMSL19] NOBRE C., MEYER M., STREIT M., LEX A.: The state of the art in visualizing multivariate networks. *Computer Graphics Forum 38*, 3 (2019), 807–832. doi:10.1111/cgf.13728. 3

[NR20] NIEDERMANN B., RUTTER I.: An integer-linear program for bend-minimization in ortho-radial drawings. In *Graph Drawing and Network Visualization - 28th International Symposium, GD 2020, Vancouver, BC, Canada, September 16-18, 2020, Revised Selected Papers* (2020), pp. 235–249. doi:10.1007/978-3-030-68766-3_19. 3

[NW24] NÖLLENBURG M., WALLINGER M.: Computing hive plots: A combinatorial framework. *J. Graph Algorithms Appl. 28*, 2 (2024), 101–129. doi:10.7155/JGAA.V28I2.2990. 3

[PF15] PADUANO F., FORBES A. G.: Extended linesets: a visualization technique for the interactive inspection of biological pathways. *BMC Proceedings 9*, 6 (2015), S4. doi:10.1186/1753-6561-9-S6-S4. 1

[Pur02] PURCHASE H. C.: Metrics for graph drawing aesthetics. *Journal of Visual Languages & Computing 13*, 5 (2002), 501–516. doi:10.1006/jvlc.2002.0232. 3

[QHBH15] QUEVILLON L. E., HANKS E. M., BANSAL S., HUGHES D. P.: Social, spatial and temporal organization in a complex insect society. *Scientific Reports 5*, 1 (2015), 13393. doi:10.1038/srep13393. 9

[RA15] ROSSI R. A., AHMED N. K.: The network data repository with interactive graph analytics and visualization. In *AAAI* (2015). URL: https://networkrepository.com. 9

[VBP*21a] VALDIVIA P., BUONO P., PLAISANT C., DUFOURNAUD N., FEKETE J.-D.: Analyzing dynamic hypergraphs with parallel aggregated ordered hypergraph visualization. *IEEE Transactions on Visualization and Computer Graphics 27*, 1 (2021), 1–13. doi:10.1109/TVCG.2019.2933196. 3

[VBP*21b] VALDIVIA P. R., BUONO P., PLAISANT C., DUFOURNAUD N., FEKETE J.-D.: Analyzing Dynamic Hypergraphs with Parallel Aggregated Ordered Hypergraph Visualization. *IEEE Transactions on Visualization and Computer Graphics 27*, 1 (Jan. 2021), 1–13. doi:10.1109/TVCG.2019.2933196. 2

[vdEHBvW13] VAN DEN ELZEN S., HOLTEN D., BLAAS J., VAN WIJK J. J.: Reordering massive sequence views: Enabling temporal and structural analysis of dynamic networks. In *2013 IEEE Pacific Visualization Symposium (PacificVis)* (2013), pp. 33–40. doi:10.1109/PacificVis.2013.6596125. 3

[Wat02] WATTENBERG M.: Arc diagrams: visualizing structure in strings. In *IEEE Symposium on Information Visualization, 2002. INFOVIS 2002.* (2002), pp. 110–116. doi:10.1109/INFVIS.2002.1173155. 3

[WDN23] WALLINGER M., DOBLER A., NÖLLENBURG M.: Linsets.zip: Compressing linear set diagrams. *IEEE Trans. Vis. Comput. Graph. 29*, 6 (2023), 2875–2887. doi:10.1109/TVCG.2023.3261934. 3

[WPC*25] WILSON C., PUERTA E., CRNOVRSANIN T., DI BARTOLOMEO S., DUNNE C.: Evaluating and extending speedup techniques for optimal crossing minimization in layered graph drawings. *IEEE Transactions on Visualization and Computer Graphics 31*, 1 (2025), 1061–1071. doi:10.1109/TVCG.2024.3456349. 3