STRATISFIMAL LAYOUT: A modular optimization model for laying out layered node-link network visualizations

Sara Di Bartolomeo, Mirek Riedewald, Wolfgang Gatterbauer, and Cody Dunne

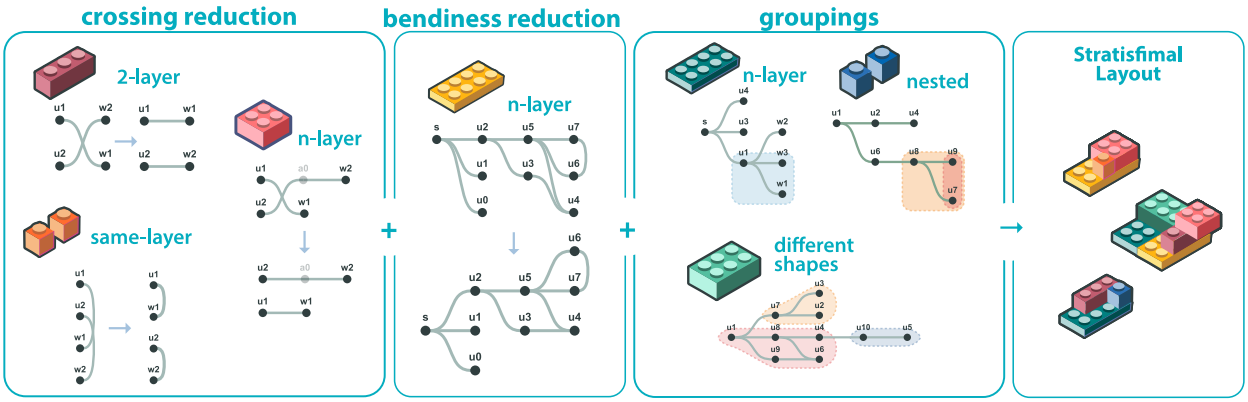


Fig. 1: Our goal is to make complicated networks easier to read. We achieve this by formulating readability criteria as a constrained optimization problem. Our optimization model for laying out layered node-link visualizations includes several modular, customizable components. Each addresses a different readability criteria or network feature — hence the building blocks metaphor. This modularity allows the layout to be tailored for diverse use cases. Here we illustrate some of the features of our Integer Linear Programming (ILP) formulation, which we call STRATISFIMAL LAYOUT— combining the words *stratified* (arranged in layers) and *optimal*.

Abstract—Node-link visualizations are a familiar and powerful tool for displaying the relationships in a network. The readability of these visualizations highly depends on the spatial layout used for the nodes. In this paper, we focus on computing *layered* layouts, in which nodes are aligned on a set of parallel axes to better expose hierarchical or sequential relationships. Heuristic-based layouts are widely used as they scale well to larger networks and usually create readable, albeit sub-optimal, visualizations. We instead use a *layout optimization model* that prioritizes *optimality* — as compared to *scalability* — because an optimal solution not only represents the best attainable result, but can also serve as a baseline to evaluate the effectiveness of layout heuristics. We take an important step towards powerful and flexible network visualization by proposing STRATISFIMAL LAYOUT, a *modular integer-linear-programming formulation* that can consider several important readability criteria *simultaneously* — crossing reduction, edge bendiness, and nested and multi-layer groups. The layout can be adapted to diverse use cases through its modularity. Individual features can be enabled and customized depending on the application. We provide open-source and documented implementations of the layout, both for web-based and desktop visualizations. As a proof-of-concept, we apply it to the problem of visualizing complicated SQL queries, which have features that we believe cannot be addressed by existing layout optimization models. We also include a benchmark network generator and the results of an empirical evaluation to assess the performance trade-offs of our design choices. A full version of this paper with all appendices, data, and source code is available at `osf.io/qdyt9` with live examples at `https://visdunneright.github.io/stratisfimal/`.

Index Terms—Layered node-link visualization, integer linear programming, crossing reduction, bendiness reduction, nested groups.

1 INTRODUCTION

Networks are widely used across many disciplines to model entities and the relationships between them [37]. Specifically, a network consists of a finite set of nodes and a finite set of edges, each connecting two nodes. **Node-link visualizations** are a familiar and powerful tool for exposing the network topology (e.g., paths between nodes and highly-connected clusters) by drawing nodes as point marks and edges as connecting line marks [58]. To better show the topology, nodes can be assigned spatial coordinates based on the relationship structure.

We may wish to expose hierarchical or sequential relationships in a network. These can be displayed in a **layered node-link visualization**, where each node is assigned to exactly one layer (rank) [73]. This layer assignment can be given a priori or obtained algorithmically. The visualization arranges the nodes in a layer along a linear axis and the axes in parallel. Here we use vertical lines for axes, but our methods work horizontally as well. The edges then fall into three categories: **same-layer edges** between nodes that share a layer, **2-layer edges** that connect nodes in adjacent layers, and **3^+-layer edges** which cross at least one intermediate layer between the incident nodes. In this context, we define **groups** as sets of nodes that must be placed adjacent to each other, both within the same layer and across consecutive layers. Each pair of groups must be either disjoint or one is a subset of the other.

For both general and layered node-link visualizations, assigning spatial coordinates to nodes is key for creating a readable visualization [4, 22, 73]. Optimally creating this assignment, called a visualization **layout**, based on readability criteria remains an open challenge. Just determining the minimal number of edge crossings is NP-complete [29]. The field of *graph drawing* focuses on this and other network visualization problems. Researchers have proposed many layout approaches — see overviews in [3, 4, 21, 31, 73] — but most are heuristics that prioritize *scalability* over *optimality*.

• *The authors are all with Northeastern University. E-mail: [dibartolomeo.s | m.riedewald | w.gatterbauer | c.dunne]@northeastern.edu*

Manuscript received xx xxx. 201x; accepted xx xxx. 201x. Date of Publication xx xxx. 201x; date of current version xx xxx. 201x. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org. Digital Object Identifier: xx.xxxx/TVCG.201x.xxxxxxx

There are two main reasons, though, to search for an **optimal solution**: (1) An optimal layout can be used as a *baseline* against which heuristics can be compared, making it possible to quantify the trade-off between increased scalability and worse readability. (2) There are numerous "small" networks in practice where maximal readability is paramount. For such applications, the extra compute time to create an optimal layout to improve readability is worth the wait.

One such example is a metro map, which has few nodes and edges, but needs to be easily and quickly readable. The up-front layout computation time is negligible compared to the possible human and capital expense to deploy a new map.

Another example, which we use as a motivating case study, is helping users interpret database queries. SQL has been the standard language for writing database queries for decades [11], but it has been repeatedly criticized for its complexity and for the time it takes to read a query. Indeed, a query can have a number of nested subqueries, involve many tables in the database, or contain complex joins — all of which make keeping a mental map of the intermediate results of the operations contained in it a very difficult task [9]. Query visualizations, such as the recently-proposed QueryVis diagrams [17, 48], expose the logical structure of SQL queries using layered node-link visualizations. The size of the network QueryVis produces to model a complicated query is small, generally consisting of fewer than 30 nodes. Since one of the uses of QueryVis is to help students learn SQL, unnecessary readability issues should be eliminated as much as possible. Moreover, as a complicated query can take several minutes to understand, the layout computation time is negligible if it stays within a few seconds.

This paper focuses on the use of **Integer Linear Programming (ILP)** to create optimal layouts for layered node-link visualizations. ILP is a method to achieve the optimal solution for a mathematical model within the boundaries established by linear constraints. An ILP formulation is comprised by an objective function, which describes the goal, and a set of constraints. The fundamental challenge of solving a problem through ILP is figuring out a smart way to define these two components, which, together, describe a problem. While ILP is NP-complete, solvers have made tremendous advances over the years to a degree where many practical problems can be solved efficiently. Recent progress on solvers, together with a natural problem-size limit established by human ability to "consume" a network visualization, have renewed interest in finding exact criteria for optimal layout algorithms.

ILP formulations have already been explored for creating optimal layouts for node-link visualizations based on readability criteria [22,71], including minimizing edge crossings [34, 35, 60, 82], minimizing edge bendiness [28], and contiguously grouping nodes within a single layer [82]. However, existing approaches do not handle same-layer edges, groups that span many layers, or nested groups — nor can these criteria be solved simultaneously.

Our goal is to provide a set of more expressive and modular formulas for readability optimizations that can be "cherry-picked" and flexibly composed like building blocks, depending on the target application and the reader's needs. Figure 1 illustrates several of our modules and how they can be composed into an overall layout optimization model.

Contributions and Supplemental Materials

In this paper, we contribute:

1. **StratisfimalLayout, an optimization model for laying out layered node-link visualizations**. The model translates important readability criteria into a modular ILP formulation and layout algorithm. It includes:
   (a) **Edge crossing reduction** for 2-layer and 3^+-layer edges as per Zarate et al. [82] and, for the first time, for same-layer edges.
   (b) **Edge bendiness reduction**, extending Gansner et al.'s formulation [28] to support modular composition with other criteria.
   (c) **Contiguous grouping of nodes** enclosed by concave or convex shapes. We improve efficiency vs. Zarate et al. [82] for groups within a single layer and, novelly, support groups that span many layers and nested groups.

   Our **modular, customizable ILP formulation** enables tailoring the layout to diverse use cases. Each module is presented with an in-depth explanation and application example to illustrate the reasoning and trade-offs for including each feature.
2. **Open-source and documented layout implementations**, both as a web-based JavaScript library and for desktop computation via Gurobi [33]. Our model can serve as a layout algorithm directly or an optimal baseline for evaluating fast layout heuristics.
3. **A case study demonstrating the utility of the layout** for improving the readability of logical diagrams of SQL queries.
4. **A benchmark network generator** with parameters to control crucial network properties, including groups of nodes, and **the results of an empirical evaluation** on these networks that illustrate the performance trade-offs of our design choices.

A full version of this paper with all appendices, data, and source code is available at `osf.io/qdyt9` with live examples at `https://visdunneright.github.io/stratisfimal/`.

2 Motivating Case study: Interpreting SQL Queries

We introduce StratisfimalLayout in the context of a motivating case study on visualizing the logical structure of SQL queries. The goal is to improve the layout of QueryVis diagrams [48], which incorporate layered node-link visualizations and hierarchical grouping of nodes within and across layers. This application was our original motivation, as we know of no existing heuristic or optimization-based layout algorithm that can address all the necessary constraints simultaneously.

SQL (Structured Query Language) has been the standard query language for relational databases for decades [11]. Formulating or interpreting a non-trivial query can be challenging and time consuming, even for experts. SQL queries can be verbose, deeply nested, and involve complex logical constructs. Several attempts have been made to improve query readability, either by mapping SQL to natural language [43] or through visual query languages [10, 30, 38]. QueryVis [17, 48], the subject of our case study, focuses on displaying the underlying logic behind an SQL query. A controlled experiment found that existing SQL users were faster at correctly interpreting queries with QueryVis than with SQL alone, even after only a brief exposure to the visual language.

QueryVis models an SQL query as a network (see Figure 2, *Middle*). Nodes represent attributes, and edges represent relationships between them (e.g., joins). Each node belongs to a group for its database table; these groups are disjoint sets. Additional groups are used to show any hierarchical nesting of subqueries. This enables modeling query nesting and logical quantifiers (e.g., NOT EXISTS). This network is then displayed using a layered node-link visualization.

Optimal layouts for readability. The visualizations produced by QueryVis may still be challenging to interpret, even for well-trained users, and despite being comparatively easier to read than the original SQL queries. This is a consequence of the logic behind SQL, which may take long to understand for complicated queries (e.g., see the unique beer taste query in Figure 2, *Left*). In many cases, running a misinterpreted query can incur hours or even days of delay for the user. Making interpretation of a visualization as easy as possible is imperative in this situation.

For that reason, we would like to find an optimal layout (subject to the constraints imposed by the query logic). QueryVis originally used a heuristic-based layout: GraphViz's *dot* [27], which is based on Gansner et al.'s edge crossing reduction algorithm for layered networks [28]. The latter is fast but does not guarantee minimal edge crossings, a widely accepted readability criteria. (See, e.g., Figure 5 and Appendix E). It also does not support groups, a key aspect of QueryVis, but *dot* can lay out groups by treating them as subproblems. This further compromises readability, especially as it does not reduce crossings of edges within a layer. The network for a complicated SQL query generally consists of fewer than 30 nodes. In this case, spending the time to compute an optimal layout for the visualization is clearly warranted.

Another case study based on StoryLines [32, 50] is in Appendix F.

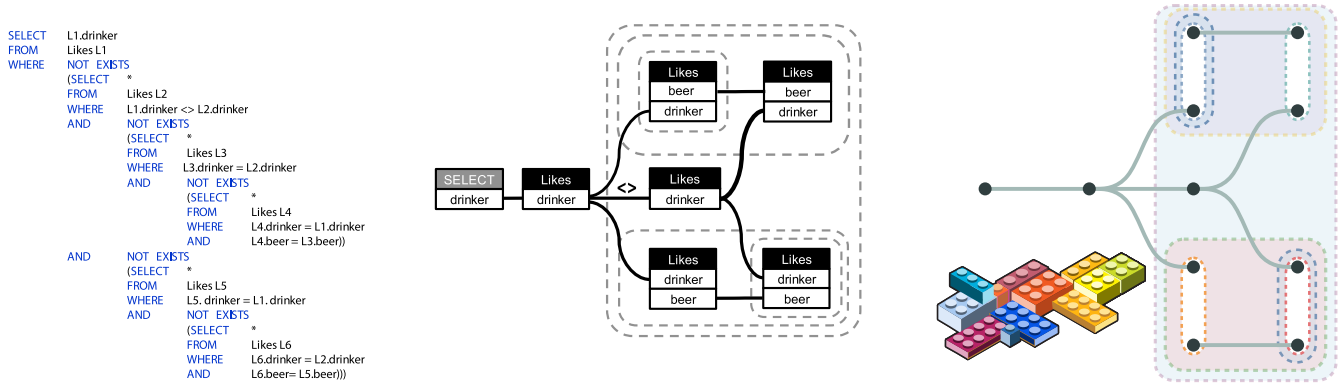


Fig. 2: Visualizing an SQL query (adapted from QueryVis, Fig. 1 [48]). *Left*: The SQL query for "find drinkers that like a unique set of beers." *Middle*: The same query shown using QueryVis. Each table used is represented by a rectangle, subdivided vertically to show the title (black background) and relevant attributes (white background). The SELECT statement is similarly displayed, just with a gray background for the title. Curves show relationships between attributes (e.g., joins) and may be labeled with comparison operators (e.g., $<>$). NOT EXISTS statements are shown as dashed lines surrounding hierarchically-nested groups for subqueries. Layers are arranged left-to-right to show depth from the SELECT statement, and the contained nodes are aligned to a vertical line. *Right*: An abstract representation of the QueryVis network, simplified by omitting any layout-irrelevant information (e.g., title nodes). The layout is optimized for readability using our method, with circles for nodes/attributes and nested convex hulls for groups. We illustrate which building blocks are used in the layout formulation (their arrangement is arbitrary).

3 Background and Related Work

Node-link visualization of networks. Networks are used to represent data in an innumerable amount of contexts, among which linguistics [12], cartography [2,54,80], computer networks [5], and social network analysis [16,56,70] are just a few examples [37]. Effectively visualizing a network can help a reader better understand relationships between elements in the data, find insights [36], and discover patterns and outliers [63]. Research on how to best represent a network as a node-link visualization started as early as 1934 with Moreno [56]. The discipline of representing networks visually is called *graph drawing* [4], though in this context we use the terminology of networks, node-link visualizations, and layouts of them.

Scientists have developed several readability criteria for visualizations [22, 66], and algorithms to implement those criteria in practice [22, 45]. Computational layout approaches began appearing in the 1960s [41, 76, 77]. Though most layout algorithm research has focused on heuristics (relevant examples: [28, 72]), there have been efforts at using neural networks [14, 46, 53] and Integer Linear Programming (ILP). The problem is still open and the general agreement is that there is no universal layout solution that embraces all the possible applications and use cases [6, 31].

Integer Linear Programming (ILP). Our StratisfimalLayout approach uses ILP to find an optimal layout based on readability criteria, rather than layout heuristics. ILP [18] is a generic optimization technique that can solve a wide range of combinatorial optimization problems. In order to use ILP for laying out a node-link visualization, a composition of a set of readability criteria needs to be translated into a quantitative objective function, such as minimizing the number of edge crossings, minimizing edge bendiness, and creating uniform edge length. Additional constraints may be specified in the formulation. The formulation can then be fed to an ILP solver, e.g., GLPK.js[1] or Gurobi [33], in order to find the optimal solution.

We integrate a discussion of the literature most relevant to our optimization model formulation as part of Section 5, but here mention a closely-related problem: metro map layouts. Nöllenburg and Wolff [62] used a mixed-integer linear program to lay out a metro map, optimizing for uniform edge length and avoiding bends. They proposed constraints that enforce that the solution keeps the original topology intact, so that readers can easily trace back the network topology to the actual geography of the represented region. Metro maps have been a popular subject for this kind of approach, e.g., [2, 54, 80].

Crossing reduction: Minimizing edge crossings is one of the best known and important optimization criteria for node-link visualization layout algorithms, as crossings clearly affect network readability for path-following tasks [66]. Buchheim et al. first proposed a branch-and-cut approach [8] to reduce the number of variables involved in an integer-linear-programming problem for crossing-restricted networks (networks that allow at most one intersection between two edges) by dividing the network into smaller subproblems. A similar approach is also used in [40]. Chimani et al. later worked on improving the efficiency of a similar method based on column generation [13]. The crossing minimization problem has also been reduced to a planarization problem [59], at first with two layers [60], then on k-level networks [34, 35]. Gange et al. [26] use ILP to obtain crossing reduction and minimal edge deletion for planarization including an optimization based on $K_{2,2}$ graphs [44]. Alternatively, we consider the entire graph for the layout as well as edge bendiness and groups. An extensive survey on crossing minimization in node-link visualizations can be found in [73].

The most relevant approach to discuss is Zarate et al.'s optimization model for layered node-link visualizations [82], which can minimize crossings of 2-layer and 3^+-layer edges. Similar to Zarate et al. and without loss of generality, we assume that all edges are same-layer or 2-layer — we show in Section 5.1.6 how to "break up" 3^+-layer edges using dummy anchor nodes to satisfy this condition. We based our initial formulation for minimizing edge crossings on Zarate et al.'s, but further extend it to support same-layer edges.

Bendiness reduction: *Edge bendiness* is another important layout optimization criteria for node-link visualizations. That is to say, having predominantly straight edges versus edges that have bends, curves, or have to be drawn diagonally. Low edge bendiness is important for users performing path-following tasks [66]. Consistent with previous work on layered networks with vertically aligned layer axes, we define an edge to be "bent" if and only if it is not horizontal, i.e., its ends have different y coordinates. The first optimization model solution by Sugiyama et al. [72] formalized the optimization objective using the squared difference in vertical coordinates between the two end nodes of an edge: $(y_{u_1} - y_{w_1})^2$. This quadratic formulation makes the optimization expensive. Gansner et al. [28] simplified the optimization objective into a linear function, though they also added a new variable and two constraints for each edge. Both Sugiyama et al. and Gansner et al. counterbalance the costliness of their solutions by additionally providing a heuristic-based layout algorithm as an alternative. There are also several other heuristic-based approaches for optimizing edge bendiness [7,24,69]. Our formulation here extends Gansner et al.'s [28] approach to support modular composition with other readability criteria. Optimizing for horizontal edges requires modifications to the objective function and introduction of variables that capture the vertical position (i.e., the y coordinate) of a node and the resulting bendiness of an edge.

[1] https://github.com/hgourvest/glpk.js

This paper generalizes several previous optimization model approaches by defining a modular mathematical framework that allows for combining and optimizing several readability criteria *simultaneously* — minimizing edge crossings, minimizing edge bendiness, and contiguously grouping nodes. These criteria can be mixed and matched to adapt to different use cases. Moreover, we contribute extensions to previous ILP-based approaches [28, 82] to novelly support minimizing edge crossings for same-layer edges, add support for groups that span multiple layers, and add support for nested groups.

4 Readability Objectives and Constraints

Based on an analysis of previous work (e.g., [22, 66, 67, 71]) and our motivating case study on QueryVis (Section 2), we identified the following objectives and constraints necessary for creating an optimal, readable layout for a layered node-link visualization.

Optimization objectives: These are the metrics we aim to optimize.

- **Minimize edge crossings [22, 66]:** In a 1997 study [66], Purchase found that crossing edges are the main characteristic negatively affecting network readability. Placing nodes so that the number of crossings is minimized is thus given the highest priority
- **Minimize edge bendiness [22, 66]:** Edge bendiness is considered a criterion negatively affecting readability as well. In a layered node-link visualization with parallel layers, we define an edge to be "bent" if its points do not lie on a line perpendicular to the layers.

Hard constraints: These conditions must be valid in any context.

- **No node-node or node-edge overlaps [22]:** Nodes cannot be drawn on top of each other and no edge can lie over a node.
- **Groups cannot enclose non-member nodes:** Only nodes that are members of a group can be drawn within the area enclosed by the group's mark.

Soft constraints: Nice-to-have features, but not strictly necessary.

- **Rectangular groups [78, 81]:** Groups can be constrained to fit in a rectangular area.
- **Minimize group area:** The area enclosed by a group should be as small as possible, given any necessary whitespace padding.

5 The Stratisfimal Layout Optimization Model

We now discuss how to formalize all these varied optimization objectives and constraints for laying out a layered node-link visualization. Our optimization model formulation uses Integer Linear Programming (ILP) and is modular — depending on the desired visualization properties, the reader can combine the corresponding features. An extensive demonstration of our approach on real networks can be found at `https://visdunneright.github.io/stratisfimal/proofs.html`.

How to read this section: Each subsection will detail a feature using a modular ILP formulation, which the reader can simply add to the overall layout optimization model. The actual formulation is marked by an image of a building block. Relevant information about the formulation is shown in an accompanying gray box. The *Intuition* behind the constraints explains the idea more plainly (but imprecisely). The *Number of constraints generated* provides some indication of how computationally expensive it would be to add the feature to the optimization model. Finally, for cases where constraints build atop each other, all the *Prerequisites* from other subsections are shown. Please refer to Figure 3 for an explanation of the notation we use.

Preprocessing: Our method assumes an already-established and immutable layer assignment for the nodes of the input network. Layer assignment may be given a priori, such as the query depth in our motivating case study (Section 2). Alternatively, layers can be assigned algorithmically based on network topology, e.g., using approach detailed by Gansner et al. [28], Rüegg et al.'s extension of it [68], or the ILP formulation by Tang & Hu [75].

5.1 Crossing reduction

5.1.1 Objective function for crossing reduction

Crossings can only occur between edges in the set E_k. The following formula minimizes the number edge crossings:

Definitions:

$G = \{N, E\}$	The network (graph) consists of a set of nodes N and edges E.
N_k	The nodes in layer k.
$E_k^=$	Same-layer edges with both ends in layer k.
$E_k^<$	2-layer edges with one end in layer k and the other in layer $k+1$.
$E_k = E_k^= \cup E_k^<$	2-layer edges (layer k to $k+1$) and same-layer edges (only layer k).
$L = \{1, 2, \ldots, \ell\}$	The set of ℓ layers in G.
$u_1 w_1$	An edge between nodes u_1 and w_1.
Γ	The set of groups in G.
g_1	A group; $g_1 \in \Gamma$.
L_{g_1}	Set of layers in which g_1 has at least one node.

Decision variables:

x_{u_1,u_2}	The relative vertical order of nodes. Boolean equal to 1 if u_1 is above u_2, 0 otherwise.
$c_{u_1w_1,u_2w_2}$	Indicates if edges u_1w_1 and u_2w_2 cross. Boolean equal to 1 if they cross, 0 otherwise.
y_{u_1}	Vertical coordinate of node u_1.
$b_{u_1w_1}$	Bendiness of edge u_1w_1, defined as $\lvert y_{u_1} - y_{w_1} \rvert$.
$y^T_{g_1}$	Topmost boundary of group g_1
$y^B_{g_1}$	Bottommost boundary of group g_1

Parameters:

m	Greatest allowed absolute vertical distance between the topmost and bottommost nodes in a layer. Default: 50, but depends on implementation details.
γ_1, γ_2	Weights for the two parts of the objective function. Default: γ_1 (crossings) $= 10, \gamma_2$ (bendiness) $= 1$.

Fig. 3: The notation used in this paper.

$$\text{Minimize}: \sum_{k \in L} \sum_{\substack{u_1w_1, u_2w_2 \in E_k \\ u_1w_1 \neq u_2w_2}} c_{u_1w_1,u_2w_2} \tag{1}$$

5.1.2 Transitivity constraints

Laying out nodes in a layer establishes a total order of nodes within it. We enforce that the ILP solution is consistent with such per-layer orders by defining transitivity constraints for each triplet of nodes in a layer. The idea is that if node u_1 is above node u_2 and node u_2 is above node u_3, then node u_1 must be above node u_3. The opposite must also be true. We can write this as $x_{u_1,u_2} \wedge x_{u_2,u_3} \Rightarrow x_{u_1,u_3}$ and $\neg x_{u_1,u_2} \wedge \neg x_{u_2,u_3} \Rightarrow \neg x_{u_1,u_3}$. These formulas, translated to ILP constraints, are written thus:

$$\begin{aligned} x_{u_1,u_2} + x_{u_2,u_3} - x_{u_1,u_3} &\geq 0 \\ -x_{u_1,u_2} - x_{u_2,u_3} + x_{u_1,u_3} &\geq -1 \end{aligned} \tag{2}$$

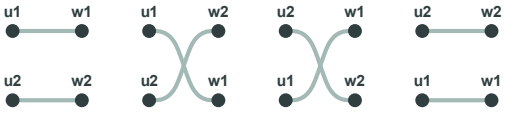
$$(\forall k \in L : \forall u_1, u_2, u_3 \in N_k, \text{ where } u_1 \neq u_2 \neq u_3 \neq u_1)$$

Intuition: If u_1 is above u_2, and u_2 is above u_3, then u_1 must be above u_3.
Number of constraints generated: $O(|N|^3)$

Note that, in any one of the constraints, x_{u_1,u_2} can always be written as $1 - x_{u_2,u_1}$. This reduces the number of variables needed in a model.

5.1.3 2-layer edge crossings

Crossing indicators c are computed by comparing position indicators x for each pair of 2-layer edges in a layer. There are 4 possible cases, captured by the visualization and formulas below.



The visualization shows that there is a crossing if the two ends of the edges are in opposite order on the first and on the second layer. This can be written as $x_{u_1,u_2} \wedge \neg x_{w_1,w_2} \Rightarrow c_{u_1w_1,u_2w_2}$ and $\neg x_{u_1,u_2} \wedge x_{w_1,w_2} \Rightarrow c_{u_1w_1,u_2w_2}$, which in turn translates to:

$$\begin{aligned} c_{u_1w_1,u_2w_2} + x_{u_2,u_1} + x_{w_1,w_2} &\geq 1 \\ c_{u_1w_1,u_2w_2} + x_{u_1,u_2} + x_{w_2,w_1} &\geq 1 \end{aligned} \tag{3}$$

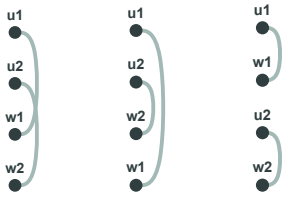
$$(\forall k \in L : \forall u_1w_1, u_2w_2 \in E_k^<, \text{ where } u_1w_1 \neq u_2w_2)$$

Intuition: There is a crossing ($c_{u_1w_1,u_2w_2} = 1$) if the ends of a 2-layer edge have opposite positions in the two adjacent layers: either u_1 is above u_2 but w_1 is below w_2 or vice versa.
Number of constraints generated: $O(|E|^2)$
Prerequisites: 5.1.2 Transitivity constraints

We provide a more detailed explanation of this constraint, which can be found in Appendix A.1.

5.1.4 Same-layer edge crossings

We need additional constraints to detect crossings of same-layer edges. There are $4! = 24$ permutations of 4 nodes in a layer; 8 of these permutations can produce a crossing between two same-layer edges. The 8 formulas of this module, illustrated and detailed below, each encode one such permutation:



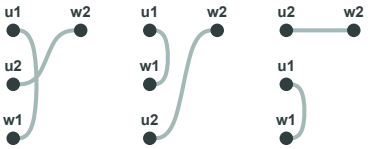
$$\begin{aligned}
c_{u_1w_1,u_2w_2} + x_{u_2,u_1} + x_{w_1,u_2} + x_{w_2,w_1} &\geq 1 \\
c_{u_1w_1,u_2w_2} + x_{w_2,u_1} + x_{w_1,w_2} + x_{u_2,w_1} &\geq 1 \\
c_{u_1w_1,u_2w_2} + x_{u_1,u_2} + x_{w_2,u_1} + x_{w_1,w_2} &\geq 1 \\
c_{u_1w_1,u_2w_2} + x_{w_1,u_2} + x_{w_2,w_1} + x_{u_1,w_2} &\geq 1 \\
c_{u_1w_1,u_2w_2} + x_{u_2,w_1} + x_{u_1,u_2} + x_{w_2,u_1} &\geq 1 \\
c_{u_1w_1,u_2w_2} + x_{w_2,w_1} + x_{u_1,w_2} + x_{u_2,u_1} &\geq 1 \\
c_{u_1w_1,u_2w_2} + x_{w_1,w_2} + x_{u_2,w_1} + x_{u_1,u_2} &\geq 1 \\
c_{u_1w_1,u_2w_2} + x_{u_1,w_2} + x_{u_2,u_1} + x_{w_1,u_2} &\geq 1
\end{aligned} \tag{4}$$

$$(\forall k \in L : \forall u_1w_1, u_2w_2 \in E_k^=, \text{ where } u_1w_1 \neq u_2w_2)$$

Intuition: There is a crossing between two same-layer edges if nodes are in one of 8 specific orderings. An example: there is a crossing if u_1 is above u_2, u_2 is above w_1, and w_1 is above w_2.
Number of constraints generated: $O(|E|^2)$
Prerequisites: 5.1.2 Transitivity constraints

5.1.5 Crossings between same-layer and 2-layer edges

Detecting crossings between same-layer and 2-layer edges requires yet another type of constraint:



$$\begin{aligned} c_{u_1w_1,u_2w_2} - x_{u_1,u_2} + x_{w_1,u_2} &\geq 0 \\ c_{u_1w_1,u_2w_2} + x_{u_1,u_2} - x_{w_1,u_2} &\geq 0 \end{aligned} \tag{5}$$

$$(\forall k \in L : \forall u_1w_1 \in E_k^= : \forall u_2w_2 \in E_k^<)$$

Intuition: There is a crossing between a same-layer and a 2-layer edge if an end of the 2-layer edge is in-between the two ends of the same-layer edge.
Number of constraints generated: $O(|E|^2)$
Prerequisites: 5.1.2 Transitivity constraints

5.1.6 3^+-layer edges

Similar to Chimani et al. [13] and Zarate et al. [82], we split any 3^+-layer edge, i.e., an edge whose endpoints are in layers i and $i+n-1$, for $n > 2$, into a path consisting of $n+1$ 2-layer edges between adjacent layers. This is done during a pre-processing step, so that the ILP formulation only has to deal with 1- and 2-layer edges, eliminating the need for more complex constraint types for 3^+-layer edges. (The pseudocode for this step can be found in Appendix C.1.) The added nodes in the intermediate layers are called *anchors*, and they will not appear in the visualization. Breaking up 3^+-layer edges increases the cardinality of E and N, thus increasing the number of constraints accordingly. The picture below shows an example effect of anchors used in combination with crossing reduction. Anchors are represented with semi-transparent nodes that are added here only for illustration.

5.2 Bendiness reduction

5.2.1 Objective function for bendiness reduction

For bendiness reduction, we add a term to the objective function to account for bent edges. Parameters γ_1 and γ_2 are set by the user to weight the relative importance of minimizing edge crossings vs. bendiness. Note that in our formulation, the bendiness $b_{u_1w_1}$ of an edge u_1w_1 is defined as the absolute difference $|y_{u_1} - y_{w_1}|$ between the vertical coordinates of its endpoints. Other formulations with more complex distance measures (e.g., squared difference) are possible, but may fall outside the capabilities of ILP and hence are left for future work.

$$\text{Minimize} : \gamma_1 \sum_{k \in L} \sum_{\substack{u_1w_1, u_2w_2 \in E_k \\ u_1w_1 \neq u_2w_2}} c_{u_1w_1,u_2w_2} + \gamma_2 \sum_{u_1w_1 \in E} b_{u_1w_1}$$

5.2.2 Vertical position constraints

For nodes in the same layer, we must ensure that their vertical positions (y variables) are consistent with the ordering established by the x variables: if $x_{u_1,u_2} = 1$ then $y_{u1} < y_{u2}$; and if $x_{u_1,u_2} = 0$ then $y_{u1} > y_{u2}$. These implications can be expressed as $y_{u_1} > y_{u_2} x_{u_1,u_2}$ and $y_{u_2} > y_{u_1} x_{u_1,u_2}$, respectively. (Note that $x_{u_1,u_2} = 1 - x_{u_2,u_1}$.) Since ILP does not allow non-linear constraints — like those involving the product of 2 variables — we need to convert them to equivalent linear constraints. For the specific structure of our problem, we identified an effective conversion detailed by Coelho [15]. The conversion introduces an auxiliary variable z_i for each node u_i and requires a predefined constant, m, that limits the maximal vertical span (i.e., the greatest absolute difference between any pair of y coordinates) of the layer:

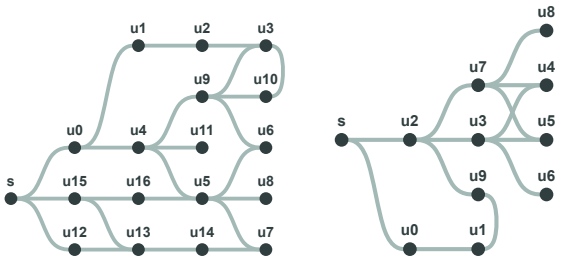
$$
\begin{aligned}
z_i - m \cdot x_{u_2,u_1} &\leq 0 \\
z_i - y_{u_2} + m \cdot x_{u_2,u_1} &\geq -m \\
y_{u_1} - z_i - 1 \cdot x_{u_2 u_1} &\geq 0 \qquad (\forall k \in L : \forall u_1, u_2 \in N_k) \qquad (6) \\
z_i &\leq y_{u_2} \\
z_i &\geq 0
\end{aligned}
$$

Intuition: Vertical node coordinates must be consistent with their vertical position ordering.
Number of constraints generated: $O(|N|^2)$
Prerequisites: 5.1.2 Transitivity constraints,

Equation (6) contains a term $-1 \cdot x_{u_2 u_1}$, which defines the distance between two nodes and can be adjusted by increasing the constant factor.

5.2.3 Computing edge bendiness

We define the bendiness of an edge as the absolute difference between the vertical coordinates of its endpoints. Unfortunately the absolute function is not allowed in ILP constraints as the resulting function would be non-linear. We therefore convert each constraint $b_{u_1 w_1} \geq |y_{u_1} - y_{w_1}|$ to the following equivalent linear constraints:



$$
\begin{aligned}
y_{u_1} - y_{w_1} &\leq b_{u_1 w_1} \qquad (\forall u_1 w_1 \in E) \qquad (7) \\
y_{w_1} - y_{u_1} &\leq b_{u_1 w_1}
\end{aligned}
$$

Intuition: The bendiness of every edge is the difference between the y variables of the two ends of the edge.
Number of constraints generated: $O(|E|)$
Prerequisites: 5.2.2 Vertical position constraints

5.3 Grouping

We now discuss how to lay out enclosed and contiguous groups of nodes and the associated constraints we developed.

Showing contiguous groups of nodes within a single layer is useful in many contexts. E.g., the proportionally sized groups in each layer in Sankey diagrams and Di Bartolomeo et al.'s Sequence Braiding visualization for temporal event sequence data — both showing an overview of flow patterns. In our motivating case study on QueryVis (Section 2, Figure 2), groups are used to display tables of attributes contiguously within a layer, akin to an entity relationship (ER) diagram. Thus, Zarate et al.'s ILP approach for laying out Sankey diagrams [82] includes a simple constraint to support such single-layer groups.

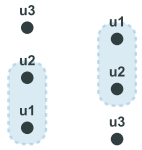
However, showing any hierarchical subqueries in an SQL query additionally requires support for groups that (1) **span multiple layers** and (2) **can be nested**. To the best of our knowledge, there are no optimization models for laying out layered node-link visualizations that have multi-level or nested groups. Providing this support in StratisfimalLayout is one of the main technical innovations of this work.

For ease of presentation and without loss of generality, the discussion here focuses on rectangular group shapes. Our framework supports more general shapes by removing the corresponding "rectangularity" constraints below, as illustrated near the end of Section 5.3.4.

5.3.1 Single-layer groups

We are not aware of any prior framework that could seamlessly deal with groups, especially in combination with crossings and bendiness. Importantly, as the experiments will confirm, in our framework groups do *not* fundamentally increase the asymptotic complexity of the optimization problem.

To enforce adjacent placement of group members in a single layer, it is sufficient to specify that if two nodes u_1 and u_2 are in the same group, then no node u_3 *not* belonging to that group can be placed between them. The constraint below equivalently expresses that u_3 has to be either above both u_1 and u_2, or below both.



$$
x_{u_1,u_3} - x_{u_2,u_3} = 0 \qquad (8)
$$

$$
(\forall g \in \Gamma, \text{ where } \Gamma \text{ is a single-layer group} : \forall u_1, u_2 \in g : \forall u_3 \notin g)
$$

Intuition: If nodes u_1 and u_2 are in group g_1, and node u_3 is not in g_1, then either u_3 is above both u_1 and u_2, or below both of them.
Number of constraints generated: Reduces the number of constraints. For every single-layer group constraint we add, 2 transitivity constraints are removed.
Prerequisites: 5.1.2 Transitivity constraints

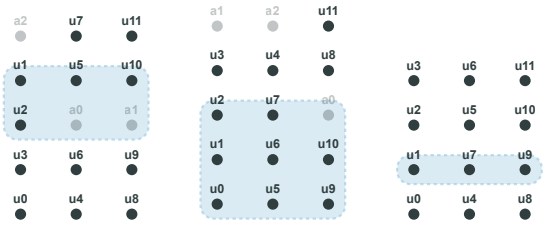
Notice that Equation (8) implies the 2 transitivity constraints in Equation (2), meaning that after adding Equation (8), *both implied constraints can be removed without changing the solution*. We show the implication for $x_{u_1,u_2} + x_{u_2,u_3} - x_{u_1,u_3} \geq 0$; the other one is analogous: From $x_{u_1,u_3} - x_{u_2,u_3} = 0$ follows $x_{u_2,u_3} - x_{u_1,u_3} = 0$, which together with $x_{u_1,u_2} \geq 0$ yields the desired $x_{u_1,u_2} + x_{u_2,u_3} - x_{u_1,u_3} \geq 0$.

5.3.2 Multi-layer groups

When a group spans multiple layers, Equation (8) is not sufficient. In addition to adjacent node placement within each layer, the vertical positions across layers needs to be taken into account. To this end, we could follow an approach similar to edge bendiness by penalizing differences in vertical coordinates between group members in adjacent layers. Since that is straightforward, we instead demonstrate our framework's flexibility by enforcing **rectangular multi-layer groups**. A rectangle is a simple easy-to-read convex hull [57], popular for enabling easy identification of group boundaries and group membership via enclosure [78, 81].

To achieve rectangular groups, we use a trick: for a given group, we determine the maximum number of group members in each layer and then add "filler nodes" to ensure the same number of group members across all layers it spans. We apply this procedure to all groups, then analogously ensure that all layers have the same total number of nodes. The pseudocode for this step is in Appendix C.2.

With the filler nodes in place, we only need to ensure that the topmost node of the group in each layer has the same y coordinate. Our ILP framework does this by "counting" the nodes above a group through the x variables. Consider a group g_1 with nodes u_1 and u_2 in some layer. A non-group node u_3 is above the group if and only if $x_{u_1,u_3} = 1$, or, equivalently, $x_{u_2,u_3} = 1$. In general, since Equation (8) enforces that all group members in a layer must be adjacent to each other, we only need to select one group member node per layer to count the number of non-group nodes above it in that layer. This count must be consistent across all layers the group spans:



$$\sum_{u_2 \notin g, u_2 \in L_k} x_{u_1 u_2} = \sum_{u_4 \notin g, u_4 \in L_{k+1}} x_{u_3 u_4} \tag{9}$$

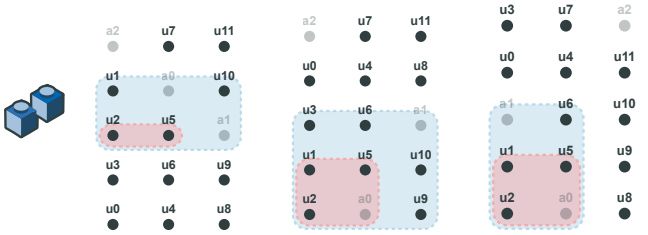
$$(\forall g \in \Gamma : \forall k \in L_g, \text{where } k+1 \in L_g)$$

In the above formula, u_1 is an arbitrary member of group g in layer k and u_3 is an arbitrary member of group g in the next layer $k+1$, i.e., $u_1 \in g \wedge u_1 \in L_k \wedge u_3 \in g \wedge u_3 \in L_{k+1}$.

Intuition: The number of non-group nodes above the group must be the same across all layers the group spans.
Number of constraints generated: $O(\ell \cdot |\Gamma|) = O(\ell \cdot |N|)$
Prerequisites: 5.1.2 Transitivity constraints, 5.3.1 Single-layer groups

5.3.3 Nested multi-layer groups

Nested groups play an important role in hierarchical networks, including our motivating case study of visualizing SQL queries. There, the relative scopes of tables are nested according to the nesting structure of the query (see Section 2). Interestingly, the ILP constraints introduced so far can, by design, already handle nested groups. We simply apply the group constraints discussed above (Equations (8) and (9)) recursively such that the subgroup (red box in the diagram below) takes on the role of the group and the enclosing group (blue box) takes on the role of the entire network. This recursive construction is applied by starting with the innermost group. The pseudocode for this procedure is described in Appendix C.2.

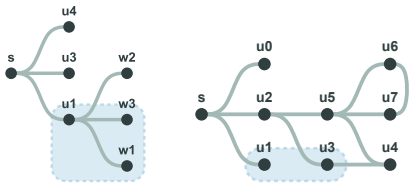


5.3.4 Multi-layer groups with vertical coordinates

Notice that the ILP constraints for multi-layer groups in Equations (8) and (9) can be used regardless of whether the problem otherwise requires y coordinates (for bendiness reduction). We now show an alternative formulation that relies on the y coordinates and has the major benefits that no filler nodes are needed and that additional requirements, e.g., regarding the shape and area of non-rectangular groups, can be expressed easily. To this end, we introduce two new variables, y_g^T and y_g^B, for every group $g \in \Gamma$ to represent the top and bottom boundary of the group, respectively. For a rectangular group, all group members must be within those boundaries:

$$\begin{aligned} y_{u_1} &\geq y_g^T \\ y_{u_1} &\leq y_g^B \end{aligned} \qquad (\forall g \in \Gamma : \forall u_1 \in g) \tag{10}$$

Intuition: All nodes in a group must fall between top and bottom boundary of the group.
Number of constraints generated: $O(|N|)$
Prerequisites: 5.2.2 Vertical position constraints, 5.3.1 Single-layer groups



Similarly, all nodes not in the group must fall outside the group boundaries in all layers spanned by the group. The direct encoding — enforcing for a node u_2 outside the group that either $y_{u_2} < y_g^T$ or $y_{u_2} > y_g^B$ — requires disjunction ("OR"), which is not supported by ILP. We therefore convert the disjunction to:

$$y_{u_2} - m \cdot x_{u_1,u_2} < y_g^T \tag{11}$$

$$-y_{u_2} + m \cdot x_{u_1,u_2} < m - y_g^B \tag{12}$$

$$(\forall g \in \Gamma : \forall k \in L_g : \forall u_2 \in L_k, \text{where } u_2 \notin g)$$

Recall that m denotes the largest vertical coordinate difference allowed within a layer. Node u_1 is an arbitrary node in layer k that belongs to group g. (Since group nodes in the same layer are adjacent to each other due to Equation (8), any one of them being above u_2 implies that all of them are.)

Intuition: In a layer spanned by the group, nodes outside the group should be either above or below the group.
Number of constraints generated: $O(|N| \cdot |\Gamma|)$
Prerequisites: 5.2.2 Vertical position constraints, 5.3.1 Single-layer groups

To prevent the solver from setting all top and bottom group boundaries to 0 (the lowest possible y value) and m (the highest possible y value), respectively, we need to add constraints that force the group boundaries to not overlap. This again introduces disjunctions (to not overlap, in each layer the top boundary of one group must be below the bottom boundary of the other, *or* vice versa), which we transform to ILP-supported linear constraints:

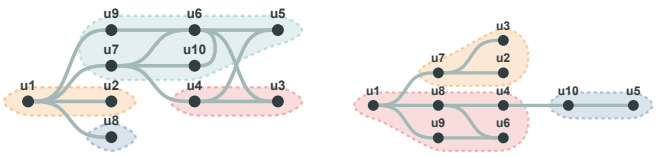
$$\begin{aligned} y_{g_2}^B - m \cdot x_{u_2,u_1} - y_{g_1}^T &< 0 \\ -y_{g_2}^T + m \cdot x_{u_2,u_1} + y_{g_1}^B &< m \end{aligned} \tag{13}$$

$$(\forall g_1, g_2 \in \Gamma, \text{where } g_1 \neq g_2 \wedge L_{g_1} \cap L_{g_2} \neq \emptyset)$$

Notice that the constraints are only introduced for groups that contain nodes in at least one common layer. Here $u_1 \in g_1$ and $u_2 \in g_2$ refer to arbirtrary group members from one of the common layers.

Intuition: For each two groups that have in common at least one layer, either the top boundary of one is below the bottom boundary of the other, or vice versa.
Number of constraints generated: $O(|\Gamma|^2)$
Prerequisites: 5.2.2 Vertical position constraints, 5.3.1 Single-layer groups

Non-rectangular groups. The y-coordinate variables enable more flexible group-shape constraints. For non-rectangular groups, one can replace variables y_g^T and y_g^B with per-layer boundaries like $y_{g_k}^T$ for each layer k spanned by group g. Or the user may decide for a group to have a common bottom boundary y_g^B, but flexible per-layer top boundaries $y_{g_k}^T$. In particular cases like the ones below, in which all the nodes in the network are enclosed in completely distinct groups, flexible group shapes can simply be attained by leaving out constraints 11 and 12.



Constraining the area of a group. So far, none of the constraints forces a group to be "compact," e.g., there could be a large gap between adjacent group members in the same layer. The introduction of variables for vertical coordinates offers an easy fix for this issue through constraints on the difference between top and bottom boundary of a group. For example, let *groupsize* be a user-controlled parameter, then we can constrain group area as follows:

$$y_g^B - y_g^T \leq groupsize \qquad (\forall g \in \Gamma) \tag{14}$$

Intuition: For each group, the difference between top and bottom boundary must not exceed a given limit.
Number of constraints generated: $O(|\Gamma|)$
Prerequisites: 5.2.2 Vertical position constraints, 5.3.4 Multi-layer groups with vertical coordinates

Note that one could similarly specify a different limit for each group. In all cases, the limit must be at least as high as the number of group members in a single layer. If it is set too low, the ILP solver will not find a solution and needs to be re-started with a higher limit.

5.4 Optimizing the number of variables and constraints

Complement variables. We cut the number of order-indicator variables in half by removing complement variables: instead of using both x_{u_1,u_2} and x_{u_2,u_1}, we use only x_{u_1,u_2} and replace x_{u_2,u_1} with $1 - x_{u_1,u_2}$.

Collapsing groups. We propose to exploit groups to reduce the *effective* network size, and thus optimization cost, for all types of constraints by essentially collapsing groups into single nodes. A special case of this idea is in Section 5.3.1, where each group-related constraint enabled the removal of 2 transitivity constraints. For the general approach, consider Figure 4. The "inclusion network" on the right encodes the hierarchy of groups and subgroups from the network on the left, making each subgroup a child of the directly enclosing group.

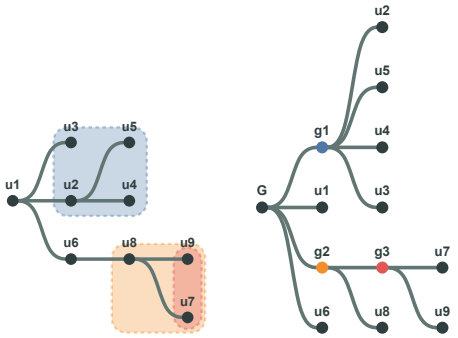


Fig. 4: A network with nested groups and its corresponding inclusion network. Colored nodes on the right represent groups on the left.

In the inclusion network, comparable elements are children of the same node. It is easy to show that every constraint involving inclusion-network nodes that are not siblings can be removed without affecting the ILP solution. E.g., in Figure 4, we can replace $x_{u_6u_2}$ and $x_{u_6u_3}$ with $x_{u_6g_1}$, as node u_6 can only be above or below both nodes in g_1. For networks with large nested groups, this can vastly reduce the number of variables and constraints, resulting in lower optimization cost.

6 Implementation and Evaluation

We implemented two versions of StratisfimalLayout: (1) a web-based JavaScript library using GLPK.js[2] as solver and (2) a formulation for desktop computation via Gurobi [33]. Our implementations, website, and supplemental material can be found at `osf.io/qdyt9` with a live demo also available online at `https://visdunneright.github.io/stratisfimal/`.

[2] `https://github.com/hgourvest/glpk.js`

The JavaScript implementation allows for easy distribution of web-based visualizations built upon StratisfimalLayout, requiring no server-side computation or local installation. The formulation of the problem, the solving process, and the visualization can all be done seamlessly client-side in the browser. The downside of using a JavaScript solver is reduced performance. But most of the networks in our target size (a few dozen nodes) can still be solved quickly.

The website also contains a network generator with parameters that can be tweaked and processed on the spot, a page[3] containing more extensive explanations and examples of the application of the modules to the network, comparisons of the application of our method compared to other layout algorithms, and several examples and benchmarks.

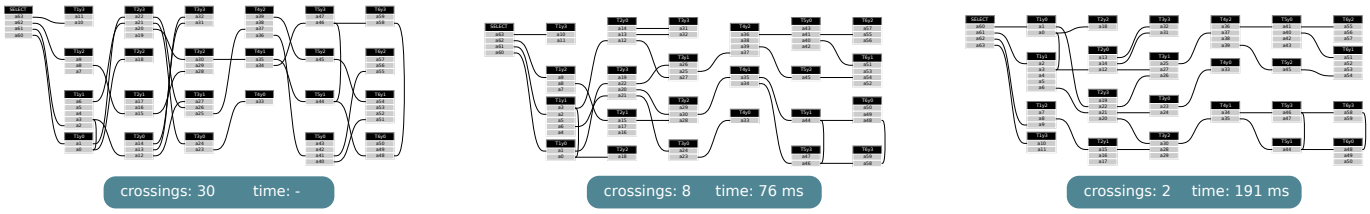
Applying the method to QueryVis on the web: We provide initial validation of our approach using of our motivating case study on QueryVis (Section 2). We conducted an experimental evaluation which compared StratisfimalLayout with Gansner et al.'s [28] heuristic-based layout based on computation time and the number of crossings. Bendiness reduction was omitted other than a very simple postprocessing step which did not impact our measurements. We used the constraints from sections 5.1.2–5.1.5 and 5.3.1. At first, a network is randomly generated (Figure 5a), then the results from the application of Gansner et al.'s method (Figure 5b) and our method (Figure 5c). While the original, randomly-generated network had 30 crossings, Gansner is able to reduce the number of crossings to 8 in 76 ms, while our method reduces the number of crossings to 2 in 191 ms. Comparisons on a number of different networks and network sizes are available on our website and appendices. As expected, our method takes more time but finds an optimal solution — demonstrating the utility of StratisfimalLayout for improving QueryVis visualizations.

The values reported in Figure 5 are obtained by running the comparison in Firefox 86, on a 2020 MacBook with 32 GB RAM and an Intel Core i5-1038NG7 CPU. We wrote both layouts in JavaScript and run entirely on the client, including the ILP solver GLPK.js.

Rome-Lib benchmark on the desktop: We also experimented with a commercial desktop-based solver, Gurobi [33] (version 9.1). In Figure 6, we report results obtained using the Rome-Lib benchmark dataset [20] — comparing the application of just crossing reduction (CM) to both crossing and bendiness reduction (CM + BR). All the tests are run using the same subset of problems from Rome-Lib — the first 26 networks with a number of nodes that is a multiple of 5, starting from 10. The problems were all run on a computer with an Intel i7-7700K processor, 32 GB RAM, and running Ubuntu 20.04. Note that Rome-Lib contains undirected, non-layered networks with no groups. In order to obtain these results, we preprocessed the Rome-Lib dataset, performing a layer-assignment step (Appendix C) to obtain layered networks. We used Navlakha et al.'s [61] network summarization algorithm to obtain groups of nodes. In Figure 6, we report the time spent solving problems of increasing size. The results are split in three figures: the first compares CM vs. CM + BR on Rome-Lib with no groups, the second includes groups but does not apply group collapsing, the third has groups and group collapsing. The visualizations show the median time to solve problems with increasing number of nodes, with boundaries at the 25% and 75% quartiles. We set a timeout of 1000 seconds for each problem, and the lines in the visualization are cut wherever the success rate of solving problems within the timeout became less than 75%, which would affect the shown data. A more extensive report can be found in Appendix C.

A word on scalability. While our experiments indicate that StratisfimalLayout could easily scale to networks with 50 to 100 nodes, it generally is difficult to predict ILP optimization time from network size alone. Other factors such as the connection pattern and the selection of constraints play an important role as well. In theory, the complexity of an ILP problem is determined by the number of variables, which define the space of possible combinations. Here Integer variables are more expensive than Boolean variables. However, in practice, an ILP solver can exploit special properties of a given problem to find the optimal solution much more rapidly. In some cases, adding

[3] `https://visdunneright.github.io/stratisfimal/proofs.html`



(a) A randomly-generated QueryVis [48] visualization.

(b) Gansner et al. [28] applied to the network.

(c) StratisfimalLayout applied to the network.

Fig. 5: Comparison of different layouts applied to a randomly generated QueryVis example.

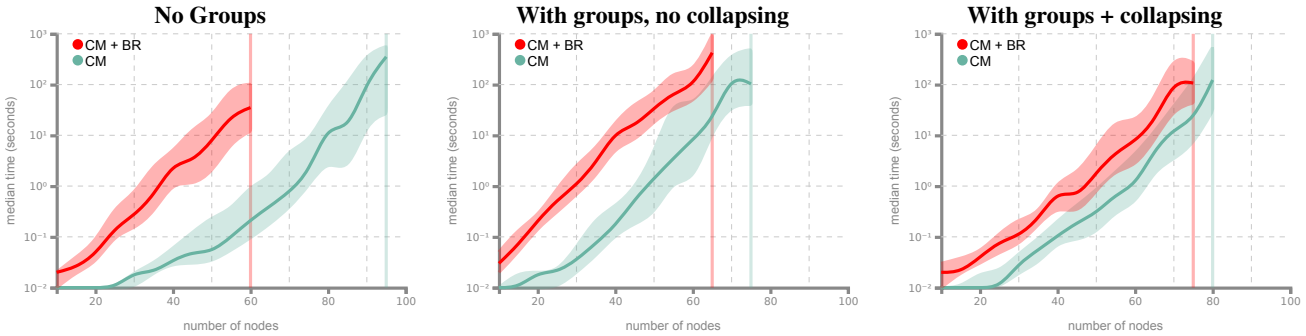


Fig. 6: The median time spent solving a problem from the Rome-Lib [20] network benchmark. The interval between the first and fourth percentiles is shown in lighter colors Each line represents the execution on the first 26 networks in Rome-Lib with n nodes, where n is a multiple of 5 between 10 and 100, for a total of 468 networks per line. The timeout was set to 1000 seconds per individual network, and the lines are cut whenever less than 75% of problems with the same number of nodes could be completed within the timeout. Crossing reduction (CM) + bendiness reduction (BR) is always several times costlier than CM only, and groups improve performance and the number of problems solvable within the timeout. Some oscillations in the lines are only due to varying difficulty in the set of problems.

redundant constraints that are already implied by existing ones can improve the solver's ability to home in on the feasibility region [47, 55]. Another approach is the introduction of variance in performance via *erraticism* [25, 51], due to imperfect tie-breaking by the ILP solver's search algorithm [42] and the choice of initial conditions for the search algorithm, such as the seed for the random-number generator [47]. Lastly, we have to take into account how performance changes with different solvers and machines [51].

For our problem, the number of variables grows more rapidly with the number of nodes in a layer than with the number of layers, because the number of $x_{u,w}$ order indicator variables is quadratic in the number of nodes in a layer. Our group-collapsing optimization (Section 5.4) ultimately reduces constraint redundancy, yet it also succeeds in reducing the number of variables. It thus effectively lowers problem complexity — in the best case, halving the number of variables. Thus, an ILP solver will perform better on a network with pre-established groups than on a network of a similar size without groups. Specific examples for this effect can be found in Appendix B.

Heuristics vs. ILP. Our method complements work on heuristic solutions by establishing a baseline against which the solutions found by the faster techniques can be compared, as done for network layout algorithms and other problems in [23, 26, 39, 52, 64]. Furthermore, ILP solvers, together with other optimization tools, have been improving at a staggering rate in terms of optimization speed and features supported [1]. Both Gurobi[4] and CPLEX's[5] most recent technical reports claim substantial improvements in processing times and the number of benchmark problems solved, including a 3× improvement over the last five years in processing time. This means that StratisfimalLayout will become faster with each improvement in ILP solver technology, allowing it to scale to larger problems even as new visualization constraints may be added.

[4] https://www.gurobi.com/wp-content/uploads/2020/02/Performance-Gurobi-9.0-1.pdf

[5] https://www.ibm.com/downloads/cas/KEVDB4NZ

7 Conclusion and Future Work

Algorithms that lay out node-link network visualizations based on human factors can help users understand complicated networks faster. Motivated by the real-world application of helping users understand complicated SQL queries faster, we devised a modular layout optimization framework that supports a combination of existing and novel readability criteria and features that together had not been addressed by any previous optimization model. Novel features include crossing minimization for same-layer edges and multi-layer and nested groups. Our approach finds optimal layouts for networks with 30 variables and complicated groupings within a second, which is a reasonable time, given that a diagram can take minutes to understand, and then be reused multiple times. The framework is versatile and allows the user to cherry-pick modules according to the features they require for their visualizations. Moreover, we provided our solution as open-source code together with a benchmark to measure optimization time for problems of various sizes and features.

Future work. Many more network features could be integrated into our modular formulation, e.g., to optimize edge-crossing angles [79], to improve the symmetry of the visualization [49, 66, 67], to bundle groups of edges [19, 65], or to minimize edge length [81]. A feature that is not necessary for our motivating example (but may be relevant for other applications) is to handle groups with overlapping intersections such as in Venn diagrams.

Acknowledgements. This work was supported in part by the National Science Foundation (NSF) under award numbers CAREER IIS-1762268 and IIS-1956096.

References

[1] R. Anand, D. Aggarwal, and V. Kumar. A comparative analysis of optimization solvers. *Journal of Statistics and Management Systems*, 20(4):623–635, 2017. doi: 10.1080/09720510.2017.1395182

[2] M. Asquith, J. Gudmundsson, and D. Merrick. An ILP for the metro-line crossing problem. In *Proceedings of the Fourteenth Symposium on Computing: The Australasian Theory - Volume 77*, CATS '08, p. 49–56. Australian Computer Society, Inc., 2008.

[3] G. D. Battista, P. Eades, R. Tamassia, and I. G. Tollis. Algorithms for drawing graphs: an annotated bibliography. *Computational Geometry*, 4(5):235 – 282, 1994. doi: 10.1016/0925-7721(94)00014-X

[4] G. D. Battista, P. Eades, R. Tamassia, and I. G. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice Hall PTR, USA, 1st ed., 1998.

[5] R. Blue, C. Dunne, A. Fuchs, K. King, and A. Schulman. Visualizing real-time network resource usage. In J. R. Goodall, G. Conti, and K.-L. Ma, eds., *Visualization for Computer Security*, pp. 119–135, 2008. doi: 10.1007/978-3-540-85933-8_12

[6] J. Blythe, C. McGrath, and D. Krackhardt. The effect of graph layout on inference from social network data. In F. J. Brandenburg, ed., *Graph Drawing*, pp. 40–51, 1996. doi: 10.1007/BFb0021789

[7] U. Brandes and B. Köpf. Fast and simple horizontal coordinate assignment. In P. Mutzel, M. Jünger, and S. Leipert, eds., *Graph Drawing*, pp. 31–44, 2002. doi: 10.1007/3-540-45848-4_3

[8] C. Buchheim, D. Ebner, M. Jünger, G. W. Klau, P. Mutzel, and R. Weiskircher. Exact crossing minimization. In P. Healy and N. S. Nikolov, eds., *Graph Drawing*, pp. 37–48, 2006. doi: 10.1007/11618058_4

[9] M. Cembalo, A. De Santis, and U. Ferraro Petrillo. SAVI: A new system for advanced SQL visualization. In *Proceedings of the 2011 Conference on Information Technology Education*, SIGITE '11, p. 165–170, 2011. doi: 10.1145/2047594.2047641

[10] C. Cerullo and M. Porta. A system for database visual querying and query visualization: Complementing text and graphics to increase expressiveness. In *18th International Workshop on Database and Expert Systems Applications (DEXA 2007)*, pp. 109–113, 2007. doi: 10.1109/DEXA.2007.91

[11] D. D. Chamberlin and R. F. Boyce. SEQUEL: A structured english query language. In *Proceedings of the 1974 ACM SIGFIDET (Now SIGMOD) Workshop on Data Description, Access and Control*, SIGFIDET '74, p. 249–264, 1974. doi: 10.1145/800296.811515

[12] Z. Chen and H. Ji. Graph-based clustering for computational linguistics: A survey. In *Proceedings of the 2010 Workshop on Graph-Based Methods for Natural Language Processing*, TextGraphs-5, p. 1–9. Association for Computational Linguistics, USA, 2010.

[13] M. Chimani, C. Gutwenger, and P. Mutzel. Experiments on exact crossing minimization using column generation. In C. Àlvarez and M. Serna, eds., *Experimental Algorithms*, pp. 303–315, 2006. doi: 10.1145/1498698.1564504

[14] A. Cimikowski and P. Shope. A neural-network algorithm for a graph layout problem. *IEEE Transactions on Neural Networks*, 7(2):341–345, 1996. doi: 10.1109/72.485670

[15] L. C. Coelho. Linearization of the product of two variables. https://www.leandro-coelho.com/linearization-product-variables. Last accessed on March 31, 2021.

[16] T. Crnovrsanin, C. W. Muelder, R. Faris, D. Felmlee, and K.-L. Ma. Visualization techniques for categorical analysis of social networks with multiple edge sets. *Social Networks*, 37:56–64, 2014. doi: 10.1016/j.socnet.2013.12.002

[17] J. Danaparamita and W. Gatterbauer. QueryViz: Helping users understand SQL queries and their patterns. In *Proceedings of the 14th International Conference on Extending Database Technology (EDBT)*, EDBT '11, pp. 558–561, 2011. doi: 10.1145/1951365.1951440

[18] G. B. Dantzig and M. N. Thapa. *Linear Programming 1: Introduction*. Springer-Verlag, Berlin, Heidelberg, 1997.

[19] S. Di Bartolomeo, Y. Zhang, F. Sheng, and C. Dunne. Sequence braiding: Visual overviews of temporal event sequences and attributes. *IEEE transactions on visualization and computer graphics*, 27(2):1353—1363, February 2021. doi: 10.1109/tvcg.2020.3030442

[20] G. Di Battista, A. Garg, G. Liotta, R. Tamassia, E. Tassinari, and F. Vargiu. An experimental comparison of four graph drawing algorithms. *Computational Geometry*, 7(5):303 – 325, 1997. 11th ACM Symposium on Computational Geometry. doi: 10.1016/S0925-7721(96)00005-3

[21] W. Didimo, G. Liotta, and F. Montecchiani. A survey on graph drawing beyond planarity. *ACM Computing Surveys*, 52(1), 2019. doi: 10.1145/3301281

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

[23] T. Dwyer, N. Henry Riche, K. Marriott, and C. Mears. Edge compression techniques for visualization of dense directed graphs. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2596–2605, 2013. doi: 10.1109/TVCG.2013.151

[24] P. Eades, X. Lin, and R. Tamassia. An algorithm for drawing a hierarchical graph. *International Journal of Computational Geometry & Applications*, 06(02):145–155, 1996. doi: 10.1142/S0218195996000101

[25] M. Fischetti and M. Monaci. Exploiting erraticism in search. *Operations Research*, 62(1):114–122, 2014. doi: 10.1287/opre.2013.1231

[26] G. Gange, P. J. Stuckey, and K. Marriott. Optimal k-level planarization and crossing minimization. In U. Brandes and S. Cornelsen, eds., *Graph Drawing*, pp. 238–249, 2011. doi: 10.1007/978-3-642-18469-7_22

[27] E. R. Gansner, E. Koutsofios, and S. C. North. Drawing graphs with dot, 1996.

[28] E. R. Gansner, E. Koutsofios, S. C. North, and K.-P. Vo. A technique for drawing directed graphs. *IEEE Transactions on Software Engineering*, 19(3):214–230, 1993. doi: 10.1109/32.221135

[29] M. R. Garey and D. S. Johnson. Crossing number is NP-complete. *SIAM Journal on Algebraic Discrete Methods*, 4(3):312–316, 1983. doi: 10.1137/0604033

[30] W. Gatterbauer. Databases will visualize queries too. *PVLDB*, 4(12):1498–1501, 2011. doi: 10.14778/3402755.3402805

[31] H. Gibson, J. Faith, and P. Vickers. A survey of two-dimensional graph layout techniques for information visualisation. *Information Visualization*, 12(3-4):324–357, 2013. doi: 10.1177/1473871612455749

[32] M. Gronemann, M. Jünger, F. Liers, and F. Mambelli. Crossing minimization in Storyline visualization. In Y. Hu and M. Nöllenburg, eds., *Graph Drawing and Network Visualization*, pp. 367–381, 2016. doi: 10.1007/978-3-319-50106-2_29

[33] Gurobi Optimization, LLC. Gurobi optimizer reference manual, 2021.

[34] P. Healy and A. Kuusik. The vertex-exchange graph: A new concept for multi-level crossing minimisation. In J. Kratochvíyl, ed., *Graph Drawing*, pp. 205–216, 1999. doi: 10.1007/3-540-46648-7_21

[35] P. Healy and A. Kuusik. Algorithms for multi-level graph planarity testing and layout. *Theoretical Computer Science*, 320(2):331–344, 2004. doi: 10.1016/j.tcs.2004.02.033

[36] N. Henry and J. Fekete. MatrixExplorer: a dual-representation system to explore social networks. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):677–684, 2006. doi: 10.1109/TVCG.2006.160

[37] I. Herman, G. Melancon, and M. S. Marshall. Graph visualization and navigation in information visualization: A survey. *IEEE Transactions on Visualization and Computer Graphics*, 6(1):24–43, 2000. doi: 10.1109/2945.841119

[38] H. Jaakkola and B. Thalheim. Visual SQL – high-quality ER-based query treatment. In M. A. Jeusfeld and Ó. Pastor, eds., *Conceptual Modeling for Novel Application Domains*, pp. 129–139, 2003. doi: 10.1007/978-3-540-39597-3_13

[39] M. Jünger, E. K. Lee, P. Mutzel, and T. Odenthal. A polyhedral approach to the multi-layer crossing minimization problem. In G. DiBattista, ed., *Graph Drawing*, pp. 13–24, 1997. doi: 10.1007/3-540-63938-1_46

[40] M. Jünger and P. Mutzel. Exact and heuristic algorithms for 2-layer straightline crossing minimization. In F. J. Brandenburg, ed., *Graph Drawing*, pp. 337–348, 1996. doi: 10.1007/BFb0021817

[41] D. E. Knuth. Computer-drawn flowcharts. *Communications of the ACM*, 6(9):555–563, 1963. doi: 10.1145/367593.367620

[42] T. Koch, T. Achterberg, E. Andersen, O. Bastert, T. Berthold, R. Bixby, E. Danna, G. Gamrath, A. Gleixner, S. Heinz, A. Lodi, H. Mittelmann, T. Ralphs, D. Salvagnin, D. Steffy, and K. Wolter. MIPLIB 2010: Mixed integer programming library version 5. *Mathematical Programming Computation*, 3(2):103–163, 2011. doi: 10.1007/s12532-011-0025-9

[43] G. Koutrika, A. Simitsis, and Y. E. Ioannidis. Explaining structured queries in natural language. In *2010 IEEE 26th International Conference on Data Engineering (ICDE 2010)*, pp. 333–344, 2010. doi: 10.1109/ICDE.2010.5447824

[44] C. Kuratowski. Sur le problème des courbes gauches en topologie. *Fundamenta Mathematicae*, 15(1):271–283, 1930.

[45] O.-H. Kwon, T. Crnovrsanin, and K.-L. Ma. What would a graph look like in this layout? a machine learning approach to large graph visualization. *IEEE Transactions on Visualization and Computer Graphics*, 24(1):478–488, Jan 2018. doi: 10.1109/tvcg.2017.2743858

[46] O. H. Kwon and K. L. Ma. A deep generative model for graph layout. *IEEE Transactions on Visualization and Computer Graphics*, 26(1):665–675, 2020. doi: 10.1109/TVCG.2019.2934396

[47] E. Lalla-Ruiz and S. Voss. Improving solver performance through redun-

dancy. *Journal of Systems Science and Systems Engineering*, 25(3):303–325, 2016. doi: 10.1007/s11518-016-5301-9

[48] A. Leventidis, J. Zhang, C. Dunne, W. Gatterbauer, H. Jagadish, and M. Riedewald. QueryVis: Logic-based diagrams help users understand complicated SQL queries faster. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, SIGMOD '20, p. 2303–2318, 2020. doi: 10.1145/3318464.3389767

[49] R. J. Lipton, S. C. North, and J. S. Sandberg. A method for drawing graphs. In *Proceedings of the First Annual Symposium on Computational Geometry*, SCG '85, p. 153–160, 1985. doi: 10.1145/323233.323254

[50] Y. T. K. liu Ma. Design considerations for optimizing StoryLine visualizations. *IEEE Transactions on Visualization and Computer Graphics*, 18(12):2679–2688, 2012. doi: 10.1109/TVCG.2012.212

[51] A. Lodi and A. Tramontani. *Performance Variability in Mixed-Integer Programming*, chap. Chapter 1, pp. 1–12. INFORMS, 2013. doi: 10.1287/educ.2013.0112

[52] R. Martí, G. Reinelt, and A. Duarte. A benchmark library and a comparison of heuristic methods for the linear ordering problem. *Computational Optimization and Applications*, 51(3):1297–1317, Apr. 2012. doi: 10.1007/s10589-010-9384-9

[53] B. Meyer. Self-organizing graphs — a neural network perspective of graph layout. In S. H. Whitesides, ed., *Graph Drawing*, pp. 246–262, 1998. doi: 10.1007/3-540-37623-2_19

[54] T. Milea, O. Schrijvers, K. Buchin, and H. Haverkort. Shortest-paths preserving metro maps. In M. van Kreveld and B. Speckmann, eds., *Graph Drawing*, pp. 445–446, 2012. doi: 10.1007/978-3-642-25878-7_45

[55] A. Miyauchi and N. Sukegawa. Redundant constraints in the standard formulation for the clique partitioning problem. *Optimization Letters*, 9:199–207, 2015. doi: 10.1007/s11590-014-0754-6

[56] J. L. Moreno. *Who shall survive?: A new approach to the problem of human interrelations.* Nervous and mental disease publishing co, 1934.

[57] D. M. Mount. Geometric intersection. In *Handbook of Discrete and Computational Geometry, chapter 33*, pp. 615–632. CRC Press LLC, Boca, 1997.

[58] T. Munzner. *Visualization analysis and design*. CRC press, 2014.

[59] P. Mutzel. An alternative method to crossing minimization on hierarchical graphs. In S. North, ed., *Graph Drawing*, pp. 318–333, 1997. doi: 10.1007/3-540-62495-3_57

[60] P. Mutzel and R. Weiskircher. Two-layer planarization in graph drawing. In K.-Y. Chwa and O. H. Ibarra, eds., *Algorithms and Computation*, pp. 72–79, 1998. doi: 10.1007/3-540-49381-6_9

[61] S. Navlakha, R. Rastogi, and N. Shrivastava. Graph summarization with bounded error. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, SIGMOD '08, p. 419–432, 2008. doi: 10.1145/1376616.1376661

[62] M. Nollenburg and A. Wolff. Drawing and labeling high-quality metro maps by mixed-integer programming. *IEEE Transactions on Visualization and Computer Graphics*, 17(5):626–641, 2011. doi: 10.1109/TVCG.2010.81

[63] A. Perer and B. Shneiderman. Integrating statistics and visualization: Case studies of gaining clarity during exploratory data analysis. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '08, p. 265–274, 2008. doi: 10.1145/1357054.1357101

[64] S. Polyakovskiy, M. R. Bonyadi, M. Wagner, Z. Michalewicz, and F. Neumann. A comprehensive benchmark set and heuristics for the traveling thief problem. In *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*, GECCO '14, p. 477–484, 2014. doi: 10.1145/2576768.2598249

[65] S. Pupyrev, L. Nachmanson, and M. Kaufmann. Improving layered graph layouts with edge bundling. In U. Brandes and S. Cornelsen, eds., *Graph Drawing*, pp. 329–340, 2011. doi: 10.1007/978-3-642-18469-7_30

[66] H. Purchase. Which aesthetic has the greatest effect on human understanding? In G. DiBattista, ed., *Graph Drawing*, pp. 248–261, 1997. doi: 10.1007/3-540-63938-1_67

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

[68] U. Rüegg, T. Ehlers, M. Spönemann, and R. von Hanxleden. A generalization of the directed graph layering problem. In Y. Hu and M. Nöllenburg, eds., *Graph Drawing and Network Visualization*, pp. 196–208, 2016. doi: 10.1007/978-3-319-50106-2_16

[69] G. Sander. A fast heuristic for hierarchical Manhattan layout. In F. J. Brandenburg, ed., *Graph Drawing*, pp. 447–458, 1996. doi: 10.1007/BFb0021828

[70] M. A. Smith, B. Shneiderman, N. Milic-Frayling, E. Mendes Rodrigues, V. Barash, C. Dunne, T. Capone, A. Perer, and E. Gleave. Analyzing (social media) networks with NodeXL. In *Proceedings of the Fourth International Conference on Communities and Technologies*, C&T '09, p. 255–264, 2009. doi: 10.1145/1556460.1556497

[71] K. Sugiyama. *Graph Drawing and Applications for Software and Knowledge Engineers*. World Scientific, 2002. doi: 10.1142/4902

[72] K. Sugiyama, S. Tagawa, and M. Toda. Methods for visual understanding of hierarchical system structures. *IEEE Transactions on Systems, Man, and Cybernetics*, 11(2):109–125, 1981. doi: 10.1109/TSMC.1981.4308636

[73] R. Tamassia. *Handbook of Graph Drawing and Visualization*. Chapman & Hall/CRC, 1st ed., 2016.

[74] Y. Tanahashi, C.-H. Hsueh, and K.-L. Ma. An efficient framework for generating storyline visualizations from streaming data. *IEEE Transactions on Visualization and Computer Graphics*, 21(6):730–742, 2015. doi: 10.1109/TVCG.2015.2392771

[75] H. Tang and Z. Hu. Network simplex algorithm for DAG layering. In *2013 International Conference on Computational and Information Sciences*, pp. 1525–1528, 2013. doi: 10.1109/ICCIS.2013.401

[76] W. T. Tutte. Convex representations of graphs. *Proceedings of the London Mathematical Society*, s3-10(1):304–320, 1960. doi: 10.1112/plms/s3-10.1.304

[77] W. T. Tutte. How to draw a graph. *Proceedings of the London Mathematical Society*, s3-13(1):743–767, 1963. doi: 10.1112/plms/s3-13.1.743

[78] C. Vehlow, F. Beck, and D. Weiskopf. Visualizing group structures in graphs: A survey. *Computer Graphics Forum*, 36(6):201–225, 2017. doi: 10.1111/cgf.12872

[79] C. Ware, H. Purchase, L. Colpoys, and M. McGill. Cognitive measurements of graph aesthetics. *Information Visualization*, 1(2):103–110, 2002. doi: 10.1057/palgrave.ivs.9500013

[80] H.-Y. Wu, S. Takahashi, D. Hirono, M. Arikawa, C.-C. Lin, and H.-C. Yen. Spatially efficient design of annotated metro maps. *Computer Graphics Forum*, 32(3pt3):261–270, 2013. doi: 10.1111/cgf.12113

[81] V. Yoghourdjian, T. Dwyer, G. Gange, S. Kieffer, K. Klein, and K. Marriott. High-quality ultra-compact grid layout of grouped networks. *IEEE Transactions on Visualization and Computer Graphics*, 22(1):339–348, 2016. doi: 10.1109/TVCG.2015.2467251

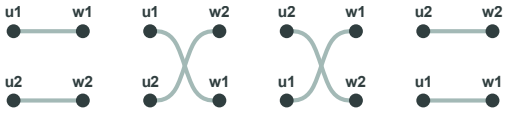
[82] D. C. Zarate, P. L. Bodic, T. Dwyer, G. Gange, and P. Stuckey. Optimal Sankey diagrams via integer programming. In *2018 IEEE Pacific Visualization Symposium (PacificVis)*, pp. 135–139, 2018. doi: 10.1109/PacificVis.2018.00025

A GUIDED UNDERSTANDING OF A CONSTRAINT

Understanding why and how the constraint work is not straightforward. In this section, we include a short tutorial on understanding a few of the constraints. We will not be doing the same for every constraint because it would take too much time and space, but we care about giving a hint to how to think about these rules.

A.1 *2-layer edge crossings*

This discussion compliments Section 5.1.

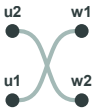


The figure above shows all the possible permutations of two nodes on two adjacent layers. The ones with a crossing are the second and third ones. Based on this picture we ask ourselves: what criteria determine if there is a crossing? And the answer that we find is: there is a crossing if the position of the ends of the edges are inverted in the two ranks. Indeed, the second picture has u_1 above u_2, but w_2 above w_1. The third one has the exact opposite, which still results in a crossing: u_2 is above u_1, but w_1 is above w_2.

Now, the constraints tie together the existence of a crossing to the relative position of the nodes. Here are the two formulas corresponding to this constraint:

$$c_{u_1w_1,u_2w_2} + x_{u_2,u_1} + x_{w_1,w_2} \geq 1 \tag{15}$$
$$c_{u_1w_1,u_2w_2} + x_{u_1,u_2} + x_{w_2,w_1} \geq 1 \tag{16}$$

As mentioned in the paper, the x variables indicate the relative position of two nodes. x_{u_1,u_2} is equal to 1 if and only if u_1 is above u_2, and it is 0 otherwise. The c variables indicate the existence of a crossing, thus $c_{u_1w_1,u_2w_2} = 1$ if there is a crossing, 0 otherwise. Note that these variables don't have any assigned value — they are just an instruction for the solver to tell to it that, if it assigns the node positions in some particular ways, there will be a crossing. Let's see it applied in practice on one of the visualization above, the third one:



This is the case in which u_1 is below u_2, thus $x_{u_1,u_2} = 0$, and w_2 is below w_1, thus $x_{w_2,w_1} = 0$. If we assign these values in Equation (16), we obtain:

$$c_{u_1w_1,u_2w_2} + 0 + 0 \geq 1$$

As the objective function given to the solver is trying to minimize the sum of all the c variables, the values assigned to $c_{u_1w_1,u_2w_2}$ will be the minimum possible value that still makes Equation (16) hold, thus $c_{u_1w_1,u_2w_2} = 1$, meaning that, with this combination of relative positions of the nodes, there will necessarily be a crossing.

There is another formula in the same constraint, Equation (15), which, if we replace the variables, gives us $c_{u_1w_1,u_2w_2} + 1 + 1 \geq 1$, which doesn't give us the lower bound for $c_{u_1w_1,u_2w_2}$ that we want. All these constraints are evaluated together though, and the values assigned to the variables must respect all equations. Thus, the final value assigned to $c_{u_1w_1,u_2w_2}$ will be 1.

All the other constraints present in the paper follow a similar thought process.

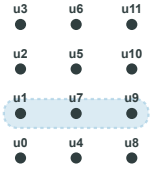
A.2 *Multi-layer groups*

To keep multi-layer groups rectangular, we presented the following formula in the body of the paper:

$$\sum_{u_2 \notin g, u_2 \in L_k} x_{u_1u_2} = \sum_{u_4 \notin g, u_4 \in L_{k+1}} x_{u_3u_4} \tag{17}$$

$$(\forall g \in \Gamma : \forall k \in L_g, \text{ where } k+1 \in L_g)$$

In practice, the way in which we impose the node to be all in a rectangle is by forcing the sum of the distance from the top of the network for any node in the group to be the same. Let's see it expanded on one example:



In this example, expanding those constraints would give us

$$\begin{aligned} x_{u_1,u_3} + x_{u_1,u_2} + x_{u_1,u_0} &= x_{u_7,u_6} + x_{u_7,u_5} + x_{u_7,u_4} \\ x_{u_7,u_6} + x_{u_7,u_5} + x_{u_7,u_4} &= x_{u_9,u_{11}} + x_{u_9,u_{10}} + x_{u_9,u_8} \\ x_{u_1,u_3} + x_{u_1,u_2} + x_{u_1,u_0} &= x_{u_9,u_{11}} + x_{u_9,u_{10}} + x_{u_9,u_8} \end{aligned} \tag{18}$$

Those constraints force all three nodes in the group to be at the same height. This can only work if all the layers in the rank contain the same number of elements and the group contains the same number of nodes on every layer it spans across, thus we first need a preprocessing step described in Appendix C.2.

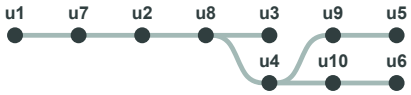
B EXAMPLE APPLICATIONS OF OUR METHOD

In this section, we show how a full model looks like and how quickly can the number of variables and constraints scale.

The first example, shown below, with only crossing reduction active on it and no groups, represents a very simple network with only two edges which can collide (u_9u_5 and $u_{10}u_6$), resulting in a very short model. Since there is a maximum of two nodes per rank, there is no need to have transitivity constraints specified. Also, this example is made especially simple by the facts that:

- In a rank with a single node, there is no need for any variable or constraint
- Two edges which have the same source or the same target can't intersect (u_4u_9 and u_4u_{10}).
- Crossing reduction alone requires only binary variables, as shown in the "Binaries" section of the model.

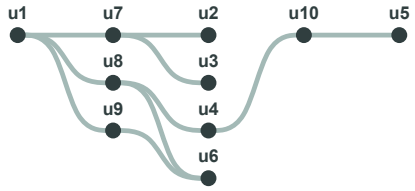
The first gray box in the text below reports a few statistics about the model and the result obtained from executing it, while the second reports the full model.




```
Constraints:  2
Variables:  4
Crossings:  0
Time to solve: 3 ms
```

```
Minimize
1 c_u9u5_u10u6

Subject To
c_u9u5_u10u6 + x_u10_u9 + x_u5_u6 >= 1
c_u9u5_u10u6 - x_u10_u9 - x_u5_u6 >= -1

Binaries
c_u9u5_u10u6
x_u10_u9
x_u5_u6
x_u3_u4
```

The next example is just a little more complicated, although it has the same number of nodes as the previous one and, like the previous one, only deals with crossing minimization. Still, more nodes in the same rank make transitivity constraints necessary and introduces the need to check more possible intersections between pairs of edges. This example shows how it's not the sheer number of nodes that influences the number of variables and constraints needed, but their positioning in the ranks.



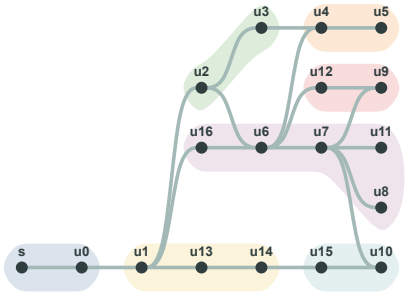

```
Constraints:  24
Variables:  16
Crossings:  0
Time to solve: 1 ms
```

```
Minimize
1 c_u7u2_u9u6 + 1 c_u7u2_u8u6 + 1 c_u7u2_u8u4 + 1 c_u7u3_u9u6
+ 1 c_u7u3_u8u6 + 1 c_u7u3_u8u4 + 1 c_u9u6_u8u4

Subject To
 + x_u7_u8 + x_u8_u9 - x_u7_u9 >= 0
 - x_u7_u8 - x_u8_u9 + x_u7_u9 >= -1
 + x_u2_u3 + x_u3_u4 - x_u2_u4 >= 0
 - x_u2_u3 - x_u3_u4 + x_u2_u4 >= -1
 + x_u2_u3 + x_u3_u6 - x_u2_u6 >= 0
 - x_u2_u3 - x_u3_u6 + x_u2_u6 >= -1
 + x_u2_u4 + x_u4_u6 - x_u2_u6 >= 0
 - x_u2_u4 - x_u4_u6 + x_u2_u6 >= -1
 + x_u3_u4 + x_u4_u6 - x_u3_u6 >= 0
 - x_u3_u4 - x_u4_u6 + x_u3_u6 >= -1
c_u7u2_u9u6 - x_u7_u9 + x_u2_u6 >= 0
c_u7u2_u9u6 + x_u7_u9 - x_u2_u6 >= 0
c_u7u2_u8u6 - x_u7_u8 + x_u2_u6 >= 0
c_u7u2_u8u6 + x_u7_u8 - x_u2_u6 >= 0
c_u7u2_u8u4 - x_u7_u8 + x_u2_u4 >= 0
c_u7u2_u8u4 + x_u7_u8 - x_u2_u4 >= 0
c_u7u3_u9u6 - x_u7_u9 + x_u3_u6 >= 0
c_u7u3_u9u6 + x_u7_u9 - x_u3_u6 >= 0
c_u7u3_u8u6 - x_u7_u8 + x_u3_u6 >= 0
c_u7u3_u8u6 + x_u7_u8 - x_u3_u6 >= 0
c_u7u3_u8u4 - x_u7_u8 + x_u3_u4 >= 0
c_u7u3_u8u4 + x_u7_u8 - x_u3_u4 >= 0
c_u9u6_u8u4 + x_u8_u9 - x_u4_u6 >= 0
c_u9u6_u8u4 - x_u8_u9 + x_u4_u6 >= 0

Binaries
c_u7u2_u9u6
c_u7u2_u8u6
c_u7u2_u8u4
c_u7u3_u9u6
c_u7u3_u8u6
c_u7u3_u8u4
c_u9u6_u8u4
x_u7_u8
x_u8_u9
x_u7_u9
x_u2_u3
x_u3_u4
x_u2_u4
x_u3_u6
x_u2_u6
x_u4_u6
```

An extensive number of examples of various sizes and complexities can be found at `https://visdunneright.github.io/stratisfimal/`, as well as a random network generator with various settings to let the user see the model applied to many different cases.

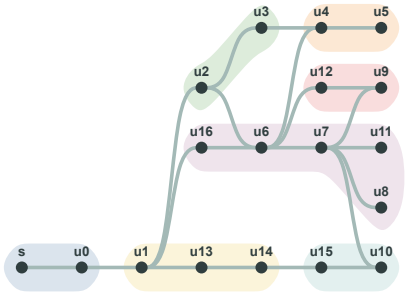
The next two examples show instead the benefit of applying our group collapsing optimization method explained in Section 5.4. The network is a little more complicated and features groups, and optimizes for both crossing and bendiness reduction, resulting in a considerable number of variables and constraints. We will not report the extended model this time, as we care more about showing the differences in sheer number of constraints and variables.

Both executions are done on the same network, and result in the same output, as shown in the two pictures, but the first one has group collapsing disabled, while the second one has it enabled. This results in the number of variables needed to describe the same problem to differ vastly, as can be seen in the gray boxes below the networks: while the first one requires 93 variables and 479 constraints, the second execution only needs 24 variables and 172 constraints. The time required to solve the problem is also greatly affected, going from 835 to just 36 ms.



Group collapsing disabled:

```
Constraints:  479
Variables:  93
Crossings:  0
Bendiness:  14
Time to solve: 835 ms
```



Group collapsing enabled:

```
Constraints:  172
Variables:  24
Crossings:  0
Bendiness:  14
Time to solve: 36 ms
```

C Pseudocode for some operations mentioned in the paper

In this section, we report the pseudocode for some procedures used in preprocessing in our method.

C.1 Adding anchors to edges

The first one deals with the introduction of anchors to split 3^+-layer edges into smaller edges (Section 5.1.6).

Algorithm 1 Adding anchors to edges

```
for edge in E do
    sr = edge.source.layer
    tr = edge.target.layer
    if abs(sr - tr) > 1 then
        anchors = [edge.source]
        minr = min(sr, tr)
        maxr = max(sr, tr)
        for i from minr + 1 to maxr do
            nweNode = new Node()
            newNode.layer = i
            anchors.push(newNode)
        anchors.push(edge.target)
        for i from 1 to anchors.length do
            newEdge = new Edge()
            newEdge.source = anchors[i-1]
            newEdge.target = anchors[i]
        network.edges.remove(edge)
```

C.2 Adding filler nodes to groups

This pseudocode is used to add filler nodes when using crossing reduction only, in order to keep groups in networks enclosed in a rectangular shape. It is mentioned in Section 5.3.2 and Section 5.3.3.

Algorithm 2 Adding filler nodes to groups

```
sortGroupsBySize()
for g of network.groups do
    minLayer ← min(group.layers)
    maxLayer ← max(group.layers)
    maxNodesPerLayer ← max(numNodesAtLayer(g, [minLayer, ..., maxLayer))
    for k from minLayer to maxLayer do
        while numNodesAtLayer(g, k) < maxNodesPerLayer do
            n ← new Node()
            newNode.layer ← k
            newNode.type ← filler
            g.nodes.push(newNode)
maxNodesInLayer = max(numNodesAtLayer(network, [minLayer, ..., maxLayer))
for ℓ of network.layers do
    while numNodesAtLayer(network, ℓ) < maxNodesInLayer do
        newNode ← new Node()
        newNode.layer ← k
        newNode.type ← fake
        g.nodes.push(newNode)
```

C.3 Simple layer assignment for non-layered networks

This snippet of code serves to assign nodes to layers in non-layered networks, so that we can obtain a layered network to process with our method. This procedure is in no way optimal, and just pushes the nodes to the n-th rank, where n is the length of the shortest path from the node to any other node in the network we design as source. This is not intended to be part of our method, but we found ourselves needing to perform this operation a number of times through the development — this is the method we use to assign ranks when generating random networks from scratch, for example. Another issue we had is that we weren't able to find a viable benchmark dataset containing layered networks, and we resorted to using Rome-Lib anyways — whose networks are not layered, thus we needed to find a way to treat them as layered. Reinstating that it is in no way optimal and should not be considered part of the project, we provide this snippet for replicability purposes, in case other researchers wanted to compare their methods against ours using networks with the same features.

Algorithm 3 Simple rank assignment

```
procedure moveToDepth(node, newDepth)
    g.nodeIndex[node.depth].remove(node)
    node.depth ← newDepth
    g.nodeIndex[node.depth].push(node)

procedure getCurEdges(r)
    curEdges ← []
    for e of g.edges do
        if e has an end in r-1 and the other in r then
            curEdges.push(edge)
    return curEdges

procedure getCurNodes(r, curEdges)
    curNodes ← []
    rankNodes ← getNodesInRank(g, r)
    for node of rankNodes do
        if no edge in curEdges has node as one of its ends then
            curNodes.push(node)
    return curNodes

g ← the network
startnode ← source node for the network
g.nodes ← list of all nodes in g
g.edges ← list of all edges in g
curRank ← 0

while g.nodeIndex[curRank] ≠ undefined do
    if curRank == 0 then
        for node of g.nodes do
            if node ≠ startnode then
                moveToDepth(node, node.depth + 1)
    curEdges ← getCurEdges(curRank)
    curNodes ← getCurNodes(curRank, curEdges)
    for node of curNodes do
        moveToDepth(node, node.depth + 1)
    curRank++
```

D Provided dataset

As mentioned in the previous section, we used Rome-Lib as a base for doing our tests, but had to introduce a couple of features into it: namely, we assigned ranks to the nodes in it and added groups through Navlakha et al.'s graph summarization algorithm [61]. For replicability purposes, we include in our supplemental material a version of Rome-Lib with our preprocessing steps already done, so that other researchers can take it and compare their methods against our own from the same starting point. It can be found on `osf.io/qdyt9`.

The dataset is provided in json format, and it is split in two folders: "paper set" contains only the networks used to run the benchmarks included in this paper, a subsection of Rome-Lib comprised of 260 networks. "Full set" instead, contains the entirety of Rome-Lib with our preprocessing step applied.

Each json contains a list of nodes with an *id* and a *depth*, which is the rank assigned to them, a list of edges with the source and the target, and a list of groups, each group having the set of nodes included in it. The group entry can be ignored when running experiments which do not consider groups.

E GraphViz issues

This section serves as a support to our claim that GraphViz is sub-optimal for network layouts with certain features. Indeed, it uses a heuristic-based approach, which does not guarantee optimality, and it does not take into account some specific features, such as same-rank edges.

This is a description of a simple planar network containing same-rank edges in the DOT language, used by GraphViz.

```
digraph G {
    s -> u3;
    s -> u0;
    u0 -> u1;
    u3 -> u1;
    u3 -> u7;
    u3 -> u8;
    u3 -> u4;
    u7 -> u8;
    u1 -> u2;
    u8 -> u9;
    u4 -> u5;
    u4 -> u6;
    u2 -> u5;

    {rank="same"; u3; u0;}
    {rank="same"; u7; u8; u1; u4}
    {rank="same"; u2; u9; u5; u6}
}
```

This is GraphViz's rendering of the network above. Even if we take into account that the curve of edge u_2u_5 could be turned towards the bottom to solve the intersection with u_4u_6, the intersections between u_7u_8 and other edges cannot be solved as easily.

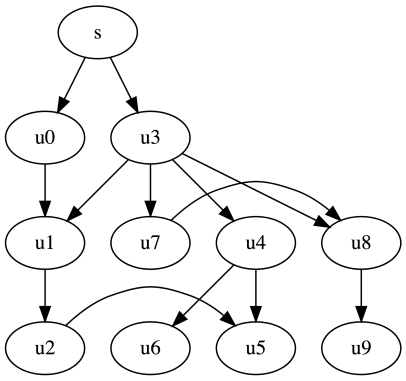


Fig. 7: GraphViz dot layout for a planar network with same-rank edges, which creates unnecessary edge crossings.

This is, instead, the rendering of the same network obtained through our own method. It can easily be seen that it found a solution with no intersections.

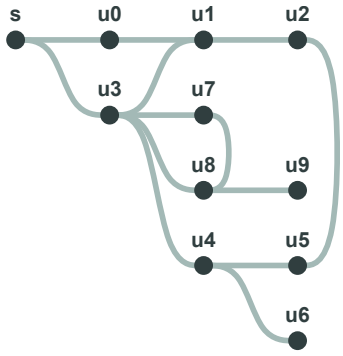


Fig. 8: The StratisFimal Layout layout for the same network as Figure 7 finds the planar solution.

F Case study: StoryLine visualization of the original Star Wars trilogy

Figure 9 shows a comparison of three different layout methods applied to creating a StoryLine visualization [50] for the original trilogy of Star Wars movies. This idea was originally featured in a xkcd comic,[6] then made into a dataset for Tanahashi and Ma's research on StoryLine visualizations [50] as part of a larger set of movie interaction datasets, and has since been used for comparative assessments of various StoryLine layout algorithms [32, 74]. At each timestep, characters are grouped together if they appear together in a scene.

The first picture, Figure 9a, is the application of Tanahashi and Ma's own method [50] on the Star Wars dataset, directly taken from their paper. They use a genetic layout algorithm which tries to approximate the optimal solution through iterations (sub-optimal). Their paper states that it took 12 minutes to generate this visualization and it has 51 crossings. In a later work, Tanahashi et al. [74], using a similar algorithm, were able to get the number of crossings down to 41, but removed a fair amount of group constraints from the previous iteration.

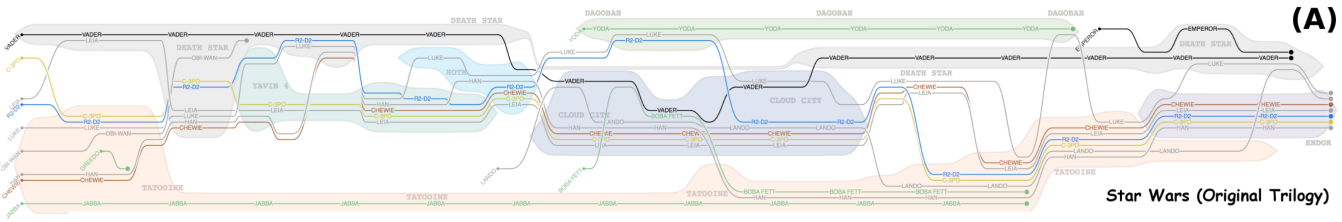
The second picture, Figure 9b, presents the same dataset using Gronemann et al.'s method [32], which is instead based on linear programming and claims optimality for crossing reduction. They claim that generating the visualization takes 1 second and has 39 crossings (although from the image we were only able to find 38 crossings). They used best-of-breed C++ desktop solvers on a supercomputing cluster.

Interestingly, our formulation (shown in Figure 9c) is actually able to obtain fewer crossings than Gronemann et al.'s [32], despite their optimality claims. Our solution has one fewer: 37 crossings. The constraints should be exactly the same: characters which appear in the same scene should be kept adjacent, while characters that do not appear together must appear separate. Note that Gronemann et al. includes twice as many layers, but in their dataset every other layer is just an exact copy of the previous one, and removing all the duplicate layers has no effect on the number of crossings.

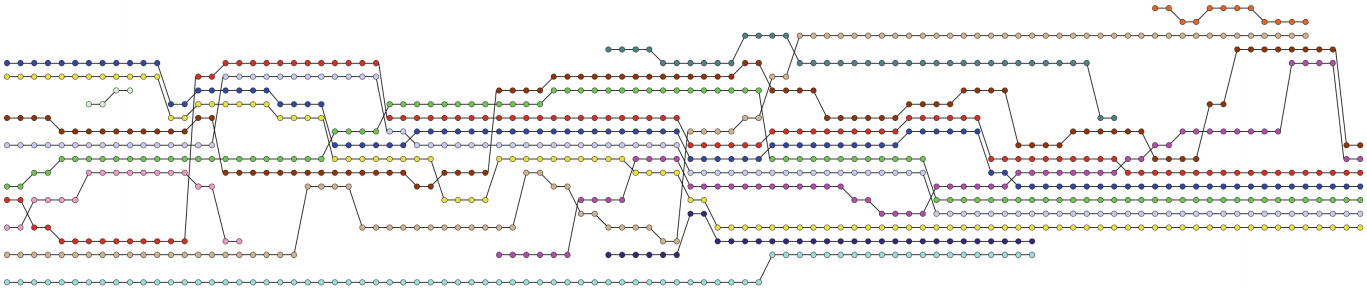
Regarding the application of StratisFimal Layout to the dataset, we used single-layer groups to constrain characters that need to be adjacent. Since the dataset is larger than what we were able to process in a single run, we used a sliding-window approach: we consider 15 layers at a time, solve the problem for those 15 layers, then slide the window to the next 15 layers. The windows always overlap by 1 layer. Thus the solution of the first problem on that shared layer is passed on to the next window, so that the ordering of the nodes can be kept consistent when passing from a window to the next. Crossing reduction is computed at first, then bendiness reduction is applied as a post-processing step so as to focus on the comparing the number of crossings. The whole dataset is processed in under 5 seconds in Chrome 91 on a 2020 MacBook Pro with 32 GB RAM and an Intel Core i5 CPU.

It's important to note that the use of this sliding-window method makes us unable to guarantee global optimality for our solution. Each solution is only optimal locally, in its own window. Indeed, if we

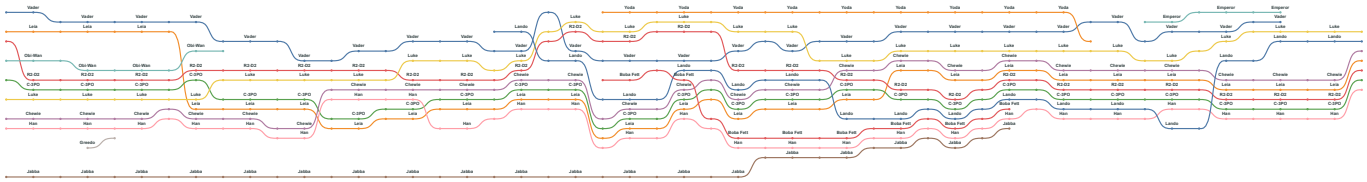
[6] https://xkcd.com/657/



(a) Tanahashi et al.'s original StoryLines visualization [50], which has 51 crossings.



(b) Gronemann et al.'s layout for the Star Wars trilogy [32]. Although the paper claims that their optimal result contains 39 crossings, we count only 38 crossings in this picture.



(c) StratisfimalLayout applied to the original Star Wars trilogy. This visualization has 37 crossings, better than Gronemann et al.'s optimal solution [32].

Fig. 9: Comparison of three layout approaches for creating a StoryLine visualization [50] for the Star Wars trilogy.

solve the window for the first set of layers, there's no way to know if the ordering of the nodes in the first window is going to cause a crossing that would have been solvable in another window. But despite this potential limitation, we had one fewer crossing than the optimal solution of Gronemann et al. [32].

G Crossing reduction: StratisfimalLayout vs. Gansner et al.

We include here a comparison of StratisfimalLayout and [28] using a randomly-generated QueryVis-style network. In these examples, StratisfimalLayout is set to compute crossing reduction only, without taking into account edge bendiness. All tests were run in Firefox 86, on a 2020 MacBook Pro with 32 GB RAM and an Intel Core i5 CPU.

The results from these tests show, predictably, that the heuristic-based method is much more scalable than the optimal method, but clearly does not achieve optimality. This confirms that StratisfimalLayout works best on smaller networks where the difference in time between the heuristic and the optimal method would not be noticeable anyways. The decision on which method is best heavily depends on the context: if the visualization is going to be static, we can afford long processing times. Cases with 45–50 edges can be processed in a very short time even using ILP, thus in these cases StratisfimalLayout can be used even in visualizations that feature interaction.

H Benchmark times

Table 2 reports in full the times briefly summarized in Figure 6 in the paper. The benchmarks were all run on a machine with an Intel i7-7700K processor, 32 gb RAM, running Ubuntu 20.04. Each test set contains a total of 364 networks, the first 26 networks with n nodes in Rome-Lib where n is 5, 10, 15... up to 80, with a timeout of 1000 seconds per problem. The line in the visualization at the top of each column represents the median time, with the confidence boundaries representing the interval within the first and fourth quartile. Just below the times, another line chart shows the timeout rate. The visualizations are cut wherever less than 75% of networks in the same category could not be processed within the timeout. The first column reports results with no groups at all — in this case, running the code with crossing reduction *and* bendiness reduction severely affects the performance, in many cases taking 10 or even 100 times the amount of time required to solve the same problems with crossing reduction only. This difference is lessened in the presence of groups — shown on the second and third columns. Note how collapsing groups (third column) improves the performances compared to not collapsing them (second column) allowing for a larger percentage of problems to be processed within the same timeout. This comparison between collapse/no collapse is analyzed further in Figure 11a and Figure 11b, which compare results within the same setting (either CM, or CM+BR) directly juxtaposing the results of applying group collapse or not applying it. From the charts, we can see that collapsing groups in most cases improves performance by 10×.

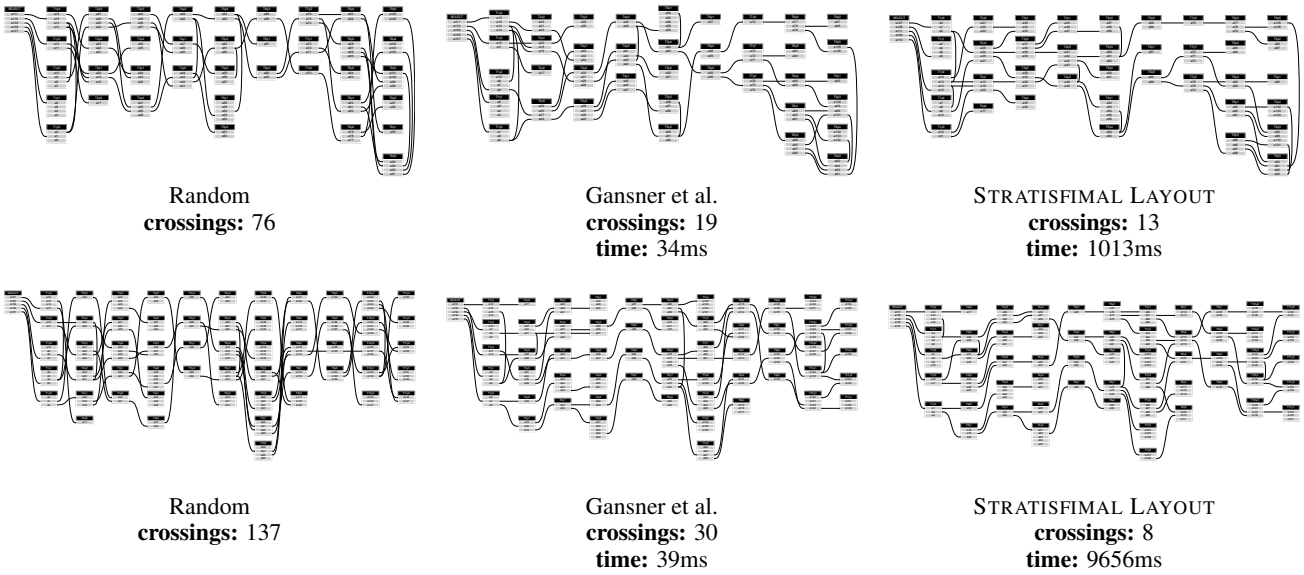
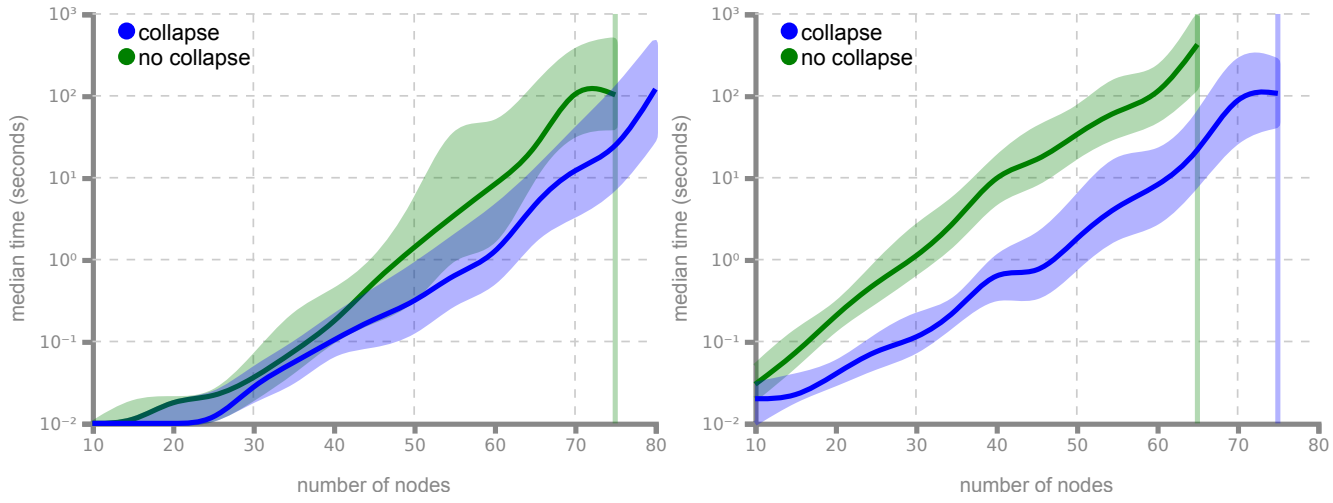


Fig. 10: Comparison of the visual result between two randomly-generated QueryVis-style networks, the same networks with Gansner et al.'s layout [28] applied to it, and the same networks with STRATISFIMAL LAYOUT (crossing reduction only) applied to it.

	Random	Gansner et al.		Stratisfimal Layout			
Edges	Crossings	Time (ms)	Crossings	Time (ms)	Crossings	Constraints	Variables
8	1	10	0	75	0	49	50
10	4	14	0	97	0	110	92
12	6	3	0	190	0	118	82
18	27	31	1	175	0	485	375
26	12	18	0	183	0	414	318
27	21	29	6	164	1	349	252
41	36	37	7	289	1	707	532
44	38	29	3	362	1	947	698
49	29	40	7	2128	4	844	567
68	62	67	18	9,586	5	1803	1285
71	89	54	9	1,033	3	1654	1095
72	91	54	12	1,468	5	1882	1692
94	111	115	31	33,362	6	2772	1794
109	200	141	35	112,446	17	3173	1912

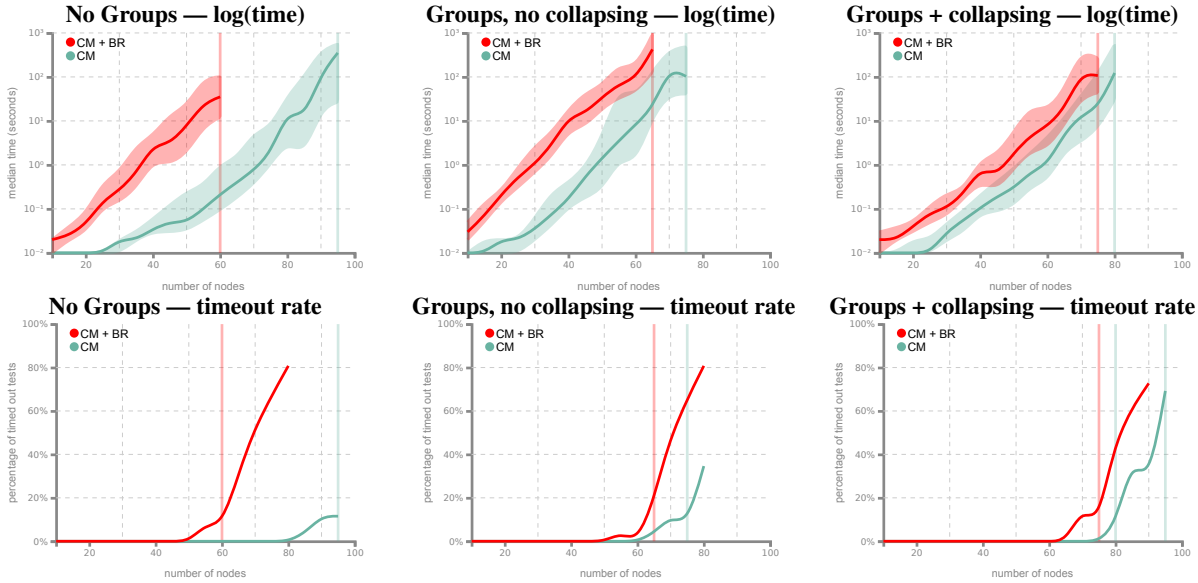
Table 1: More timing, crossings, and constraint/variable count comparisons between Gansner et al.'s layout [28] and STRATISFIMAL LAYOUT (crossing reduction only) for several random networks.



(a) Comparison for crossing minimization (CM) only.

(b) Comparison for crossing minimization with bendiness reduction (CM + BR).

Fig. 11: Comparison of the time spent solving problems when using Section 5.4 **Optimizing the number of variables and constraints** and when not applying it.



Number of nodes	CM		CM + BR		CM		CM + BR		CM		CM + BR	
	median time (s)	timeout rate	median time (s)	timeout rate	median time (s)	timeout rate	median time (s)	timeout rate	median time (s)	timeout rate	median time (s)	timeout rate
10	0		0.01		0		0.02		0		0.01	
15	0		0.02		0		0.06		0		0.01	
20	0		0.04		0.01		0.2		0		0.03	
25	0		0. 14		0.01		0.52		0		0.07	
30	0.01		0.26		0.03		1.05		0.02		0.1	
35	0.01		0.64		0.07		2.83		0.05		0.21	
40	0.03		2.8		0.16		11.91		0.1		0.83	
45	0.04		3.01		0.54		15.18		0.18		0.57	
50	0.04		7.66		1.41		34.37		0.29		1.87	
55	0.09		21.03	8%	3.51		58.54	4%	0.68		4.63	
60	0.21		28.76	8%	8.3		91.61		1.02		7.67	
65	0.39		69.25	31%	19.21	4%	329.04	19%	5.19		18.56	
70	0.75		229.24	53%	136.91	12%	281.17	50%	13.31		69.63	15%
75	1.85				88.74	8%			19.56		94.61	8%
80	16.72		620.08	81%	168.63	35%	517.54	81%	91.12	8%	164.03	50%
85	10.98	4%							102.87	38%		
90	94.63	12%							59.06	26%	440.86	73%
95	120.57	12%							74.66	69%		
100												

Table 2: The table reports the results from our method on the Rome-Lib [20] benchmark dataset of networks. Each test is run on the first 26 networks included in Rome-Lib containing the number of nodes reported on the left column. The first column reports results with no preprocessing to introduce groups, the second column introduces groups, the third one has groups and makes use of group collapsing to optimize the number of variables and constraints. For each one of these cases, we tested the method with just crossing minimization in the objective function and constraints (CM) and with both crossing minimization and bendiness reduction (CM + BR). For each test, we set a timeout of 1000 seconds. The times reported are averages, in seconds, excluding the tests that timed out. The percentage of timed out tests for each test is reported in the "fail ratio" column.