Journal of Graph Algorithms and Applications
http://jgaa.info/ vol. 28, no. 2, pp. 101–129 (2024)
DOI: 10.7155/jgaa.v28i2.2990

Computing Hive Plots: A Combinatorial Framework

Martin Nöllenburg[1] *Markus Wallinger*[1]

[1]Algorithms and Complexity Group, TU Wien, Vienna, Austria

Submitted: November 2023 Accepted: October 2024 Published: October 2024
Article type: Regular Paper Communicated by: Michael A. Bekos and Markus Chimani

Abstract. Hive plots are a graph visualization style placing vertices on a set of radial axes emanating from a common center and drawing edges as smooth curves connecting their respective endpoints. In previous work on hive plots, assignment to an axis and vertex positions on each axis were determined based on selected vertex attributes and the order of axes was prespecified. Here, we present a new framework focusing on combinatorial aspects of these drawings to extend the original hive plot idea and optimize visual properties such as the total edge length and the number of edge crossings in the resulting hive plots. Our framework comprises three steps: (1) partition the vertices into multiple groups, each corresponding to an axis of the hive plot; (2) optimize the cyclic axis order to bring more strongly connected groups near each other; (3) optimize the vertex ordering on each axis to minimize edge crossings. Each of the three steps is related to a well-studied, but NP-complete computational problem. We combine and adapt suitable exact and heuristic algorithmic approaches, implement them as an instantiation of our framework, and show in a case study how it can be applied in a practical setting. Furthermore, we conduct computational experiments to gain further insights regarding the algorithmic choices of our framework. The code of the implementation and a prototype web application can be found on OSF[1].

1 Introduction

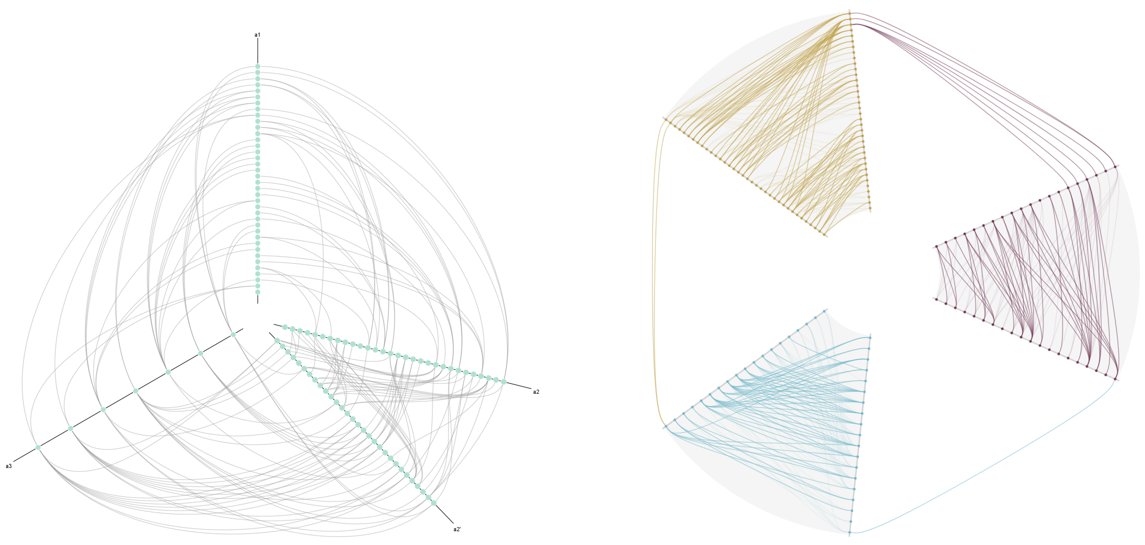
Hive plots [16] is a visualization style for network data, where vertices of a graph are mapped to positions on a predefined number of radial axes emanating from a common center, like spokes

Special issue on Selected papers from the Thirty-first International Symposium on Graph Drawing and Network Visualization, GD 2023

A preliminary version of this paper appeared in the proceedings of the 31st International Symposium on Graph Drawing and Network Visualization (GD 2023) [21]. This research has been funded by the Vienna Science and Technology Fund (WWTF) [10.47379/ICT19035].

E-mail addresses: noellenburg@ac.tuwien.ac.at (Martin Nöllenburg) mwallinger@ac.tuwien.ac.at (Markus Wallinger)

[1]https://osf.io/6zqx9/ (10.17605/OSF.IO/6ZQX9)



(a) (b)

Figure 1: (a) A small social network (see Section 5) visualized with jhive [16]. Vertices are mapped to axes according to their degree. The position on each axis is determined by a rank order over vertex degrees. (b) A hive plot created by our framework shows the same graph, where vertices are assigned to $k = 3$ axes by modularity maximization. Additionally, crossings of inter- and intra-axis edges are minimized.

of a wheel. Mapping and positioning are usually done based on vertex attributes and not to optimize layout aesthetics. Due to this strict, rule-based definition, hive plots are a deterministic network visualization style; see Fig. 1a for an example. Similarly to parallel coordinate plots [28], the original idea behind hive plots is to create a unique fingerprint-like visualization for quantitatively understanding and comparing network structures. This task can quickly get very difficult with force-based layouts due to the 'hairball' effect for large and dense graphs and their often unpredictable behavior when optimizing for conflicting aesthetic criteria.

Usually, edges are drawn as Bézier curves connecting their respective endpoints while being restricted to three axes to avoid the problem of routing longer edges around axes; this is considered beneficial for visual clarity. In the case of edges between vertices on the same axis, the axis, and its associated vertices are cloned and positioned closely to each other such that edges are either drawn twice (symmetrically) or once (asymmetrically). The latter case reduces visual complexity but increases ambiguity as an edge is only explicitly connected to one copy of each vertex; see Fig. 2 for a sketch of the different concepts.

Multiple hive plots can also be arranged in a matrix, called a hive panel [16], where columns and rows represent different axis mapping functions. Differential hive plots visualize networks changing over time [17]. Since their inception a decade ago, hive plots have been utilized in various applications and use-cases, e.g., cyber security [13], machine learning of visual patterns [24], life sciences [23], biological data [32], or sports data [22]. Although various use cases exist, hive plots have not yet been investigated from a formal graph drawing perspective, which systematically defines degrees of freedom and layout optimization criteria.

This is a rather unexpected observation, especially as hive plots have some inherent properties that make them an interesting layout style. For example, by placing vertices on axes, the layout is predictable and usually has a good aspect ratio. Similarly, edges can be routed strictly between or around an axis. Thus, edges overlapping with unrelated vertices is not an issue in hive plot layouts and increases the overall faithfulness of the drawing. Furthermore, it is also relatively straightforward to position labels and label-vertex overlaps. Lastly, edges between vertices on the same axis can be hidden or shown on demand, thus, reducing unnecessary information and decreasing the cognitive load.

Contributions and Related Work: In this paper, we present the first formal model of hive plots and identify their associated computational optimization problems from a combinatorial point of view. Based on this model, we investigate several degrees of freedom that can be utilized for optimizing hive plot layouts on typical graph drawing quality metrics for arbitrary undirected graphs.

First, in our investigation, we expand on an idea that was already mentioned in the original hive plot publication [16]. We partition the graph into some number k of densely connected clusters, where each cluster is assigned to exactly one axis. In terms of visual design, this allows us to show or hide intra-cluster edges on demand and focus on representing the sparse connectivity between clusters. We find such clusters by applying techniques from the area of community detection in networks [10]. Even though a similar assignment strategy is presented in the original hive plot publication [16], the focus here is on visually clustering vertices according to their community membership and assigning vertex clusters to segments on a subdivided axis. Vertex assignment to k layers has also been investigated in the context of cyclic level drawings of directed graphs [2]. The main difference to our model is that cyclic level drawings consider edges between vertices on the same layer to wrap around the cycle, thus, having a length of k. It has been shown to be NP-hard [2] to minimize the total edge length by assigning vertices to k given layers. However, several heuristics have been proposed and computationally evaluated [2] for this problem. In contrast, we assign vertices to axes before we optimize the total edge length by reordering axes.

Second, we are free to assign any cyclic order over the k different axes. Here we minimize the total length of inter-axis edges by placing pairs of axes with many edges between them close to each other. This is essentially equivalent to the circular arrangement problem, in which vertices are positioned evenly spaced on a circle such that the total weighted length of edges is minimized. Finding the minimum circular arrangement of undirected and directed graphs is NP-complete [11, 18]. However, a polynomial-time $O(\log n)$-approximation for undirected graphs exists [18]. Similarly, the problem of minimizing the crossings in a circular arrangement of a graph is NP-complete [19]. The concept of circular arrangements has been applied to circular drawings [12] where a subset of edges is drawn outside of the circle to reduce edge crossings. Here, we present an exact approach based on *integer quadratic programming* (IQP) as well as a heuristic approach based on simulated annealing to compute an axis order.

Lastly, once the order of axes is fixed, we want to minimize the number of inter- and intra-axis edge crossings. Here, the problem is similar to multi-layer *crossing minimization* (CRM), which has been studied in the context of the Sugiyama framework [26] for hierarchical level drawings of directed graphs. In this type of drawing, vertices are assigned to horizontal layers with edges either drawn in an upward or downward direction. In the case of cycles in the graph, some edges need to be reversed in the drawing. Cyclic level drawings have already been mentioned by Sugiyama et al. as an alternative to reversing edges, and they have been thoroughly investigated in more recent years [1, 2, 3, 12]. Algorithms for crossing minimization in cyclic level drawings generalize to our hive plot model.

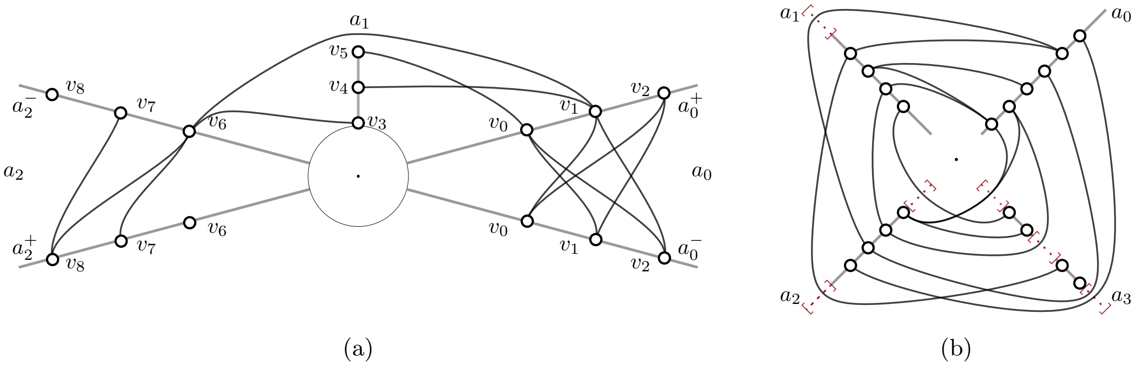


Figure 2: In (a), a schematized hive plot with three axes shows different concepts. Axis a_1 is collapsed. Axis a_0 and a_2 are expanded with edges in a_0 being drawn symmetrically. A long edge between v_1 and v_6 is bypassing a_1. In (b), the concept of gaps and how long edges are routed is shown. Here, axis a_0 has no gaps, axis a_1 has one gap, a_2 has two gaps, and a_3 has three gaps. The gaps are indicated with red dotted lines.

Crossing minimization in cyclic level drawings and layered drawings commonly performs repeated 2-layer crossing minimization steps. The 2-layer crossing minimization problem is already NP-hard [8], even if one layer is fixed. As 2-layer CRM is a prominent problem, several exact approaches have been proposed over the years. The most prominent is the linear and quadratic integer programming formulation of Jünger and Mutzel [14, 15]. We adapt their formulation to our problem and show that we can compute exact solutions for small instances.

Additionally, we present several variants based on heuristic algorithms, such as the barycenter algorithm [26] or the sifting algorithm [20]. The sifting algorithm has been specifically investigated in the context of cyclic level drawings before [1]. Our IP and heuristic CRM variants require additional constraints such that long edges are routed through gaps (see Section 2). Adding constraints to 2-layer crossing minimization heuristics has been applied previously, e.g., for fixing the relative positions of a subset of vertex pairs [9].

Based on our framework, in Section 3, we present a 3-step pipeline that computes a Hive plot layout. For each step, we introduce several integer programming (IP) and heuristic variants to compute an exact or heuristic solution for the associated problem statement. In Section 4, we show how we visually encode the computed Hive plot layouts and present some simple interaction techniques. We conduct a small case study and explain in Section 5 how hive plots generated by our framework can be applied in a practical context of co-authorship networks. Finally, in Section 6, we investigate the computational boundaries of the different pipeline variants in an extensive computational experiment based on synthetic graphs.

2 Formal Model

A *hive plot layout* $H(G) = (A, \alpha, \phi, \Pi)$ of an undirected graph $G = (V, E)$ is a tuple consisting of a set $A = \{a_0, \ldots, a_{k-1}\}$ of k axes, a surjective function $\alpha : V \rightarrow A$ mapping vertices to axes, a bijective function $\phi : A \rightarrow \{0, ..., |A| - 1\}$ representing a cyclic ordering of axes and a set $\Pi = \{\pi_0, \ldots, \pi_{|A|-1}\}$ of orderings over the vertices assigned to each axis. As each vertex is assigned to one axis $a_i \in A$ this imposes a disjoint grouping $V_i := \alpha^{-1}(a_i)$ of V such that $V_i \cap V_j = \emptyset$ for each $i \neq j$ with $i, j \in \{0, \ldots, |A-1|\}$. Each π_i is a bijective function $\pi_i : V_i \rightarrow \{0, \ldots, |V_i| - 1\}$.

We use the shorthand notation $\phi(u) = \phi(\alpha(u))$ whenever we refer to the order of the axis $\alpha(u)$ to which a vertex u is mapped. The *span* of two axes a_i, a_j or two vertices u, v is defined as $\text{span}(a_i, a_j) = \min\{\phi(a_i) - \phi(a_j) \pmod k, \phi(a_j) - \phi(a_i) \pmod k\}$ or $\text{span}(u, v) = \text{span}(\alpha(u), \alpha(v))$. Based on the span, we can classify edges into three different categories. An edge $e = (u, v)$ is called *proper* if $\text{span}(u, v) = 1$. Otherwise, an edge is considered *long* if $\text{span}(u, v) > 1$ or *intra-axis* if $\text{span}(u, v) = 0$. *Inter-axis* edges are all edges that are either proper or long. A long edge (u, v) can be subdivided and replaced by $\text{span}(u, v) - 1$ virtual vertices assigned to the appropriate axes between $\alpha(u)$ and $\alpha(v)$. A long edge in a hive plot layout needs to *bypass* axes to connect source and target vertices without creating axis-edge overlaps. We assume that edges are routed along the shorter distance according to their span. In the case of an even number of axes, we assume that all edges connecting two vertices on opposite axes are routed clockwise. Combinatorially this can be realized by enforcing virtual vertices to appear at certain positions in each axis order. In our model, a hive plot layout can have up to g gaps per axis; see Fig. 2b where the concept is shown. If $g = 1$, then all virtual vertices have to be at the end of each order. If $g = 2$, then virtual vertices have to be at either the beginning or end of each order. In cases where $g > 2$ all virtual vertices form a partition into up to g groups, where they must appear consecutively within each group.

To consider intra-axis edges during optimization an adaption is necessary. Basically, all axes and their associated vertices are duplicated such that for each axis a_i, there are two copies a_i^+ and a_i^-, respectively, and vertex sets V_i^+ and V_i^-; see Fig. 2. The vertex order on duplicate axes remains the same, i.e., $\pi_i^+ = \pi_i^- = \pi_i$.

We consider two proper edges (u, v) and (x, y) to be *crossing* if $u, x \in V_i$ and $v, y \in V_j$ such that $\pi_i(u) < \pi_i(x)$ and $\pi_j(y) < \pi_j(v)$. Similarly, if the end points of two long edges (u, v) and (x, y) are on four different axes such that, w.l.o.g, $\phi(u) < \phi(x) < \phi(v) < \phi(y) \pmod k$ or on three different axes such that, w.l.o.g., $\phi(u) = \phi(x) = i$, $\pi_i(x) < \pi_i(u)$, and $\phi(x) < \phi(v) < \phi(y) \pmod k$ a crossing is unavoidable if $g = 1$. We note that our assumption of routing edges along the shorter cyclic distance prevents avoiding such a crossing by routing one of the involved edges along the longer cyclic distance. The proper *neighborhood* of a vertex u is defined as $N(u) = \{v \mid (u, v) \in E, \text{span}(u, v) = 1\}$.

3 Framework for Computing Hive Plots

Next, we present our framework for creating a hive plot layout $H(G) = (A, \alpha, \phi, \Pi)$ of an undirected simple graph $G = (V, E)$. The framework itself, as seen in Fig. 3, is modeled as a pipeline consisting of three stages where each stage has multiple variants. In stage (1), we partition the vertices into multiple groups, each corresponding to an axis of the hive plot. Next, in stage (2), we optimize the cyclic axis in order to bring strongly connected groups near each other. Finally, in stage (3), we optimize the vertex ordering on each axis to minimize edge crossings. Furthermore, stage (3) is a two-step process where first inter-axis edge crossings are optimized (3a), then the relative order of vertices with inter-axis connections are fixed, and intra-axis edge crossings are optimized (3b). Moreover, edge crossing minimization is performed under the constraint that long edges need to be routed through gaps in the axes that they pass.

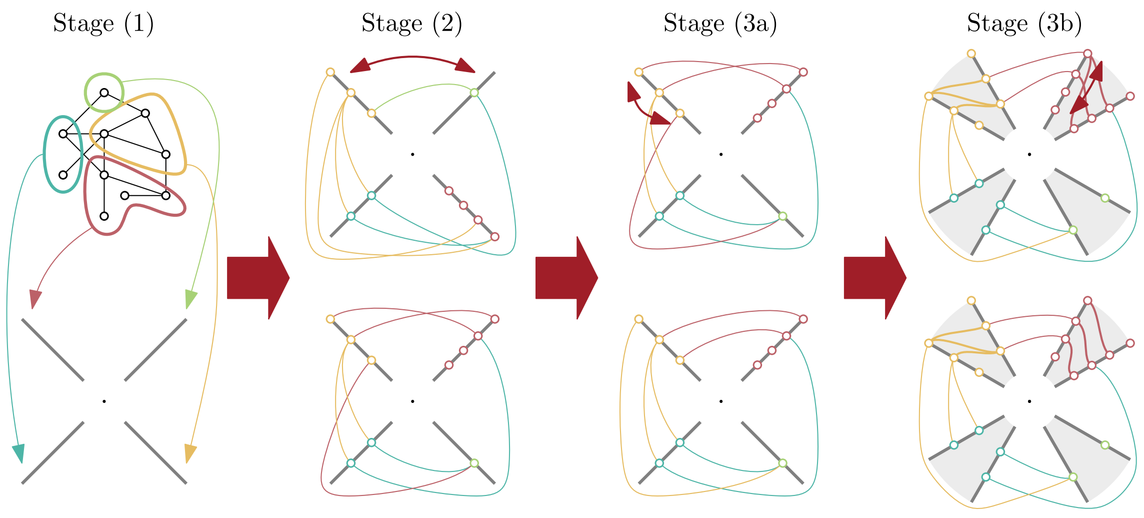


Figure 3: Conceptual overview of our Hive plot framework. In stage (1), we partition the graph. Then, in stage (2), we optimize the axis order before inter-axis edge crossings are minimized in stage (3a). Lastly, in stage (3b), we fix the partial order of vertices with inter-axis connections and minimize the intra-axis edge crossings. It is also feasible to perform (3a) and (3b) at once.

3.1 Vertex Partitioning (1)

In the first stage, we partition the vertex set V into k subsets $\{V_0, \ldots, V_{k-1}\}$ such that each subset maps to exactly one axis a_i in the hive plot. The core idea is that the subsets of the partition represent dense induced subgraphs. In our implementation, we provide three different strategies to compute a partition, depending on the properties of the input. First, if we consider the parameter k as an additional input we use the Clauset-Newman-Moore greedy modularity maximization [5] to compute a partition of size k. Second, if k is not specified, we apply the Louvain [4] community detection algorithm instead. Here, the size of the partition is determined by how many communities are detected. Lastly, this step of the framework is not necessary if a partition is already given in the input, e.g., via the home institutions of authors in a co-authorship network. We note that any other algorithm to partition the graph into meaningful groups can be used.

3.2 Axis Ordering (2)

The second stage orders the axes such that the total *span* of edges is minimized. Our approach assumes that edges are always drawn along the shortest path around the circle between endpoints, either clockwise or counterclockwise. Basically, we want to maximize the number of proper edges while minimizing the number and total length of long edges. We do not consider the individual position of vertices on their respective axes but rather look at the aggregated edges incident to the subsets of the axis partition.

Let $w_{i,j}$ be the number of edges between V_i and V_j for $i < j$. Recall that the definition of span covers edges spanning over the last axis. The cost function of an axis order ϕ is defined as follows:

$$\mathrm{cost}(\phi) = \sum_{i=0}^{k-1} \sum_{j=i+1}^{k-1} w_{ij}\ \mathrm{span}(a_i, a_j). \tag{1}$$

Integer Quadratic Programming Model: We can model the problem as a simple integer quadratic program

$$\begin{aligned}
\text{minimize} \quad & \sum_{i=0}^{k-1}\sum_{j=0}^{k-1}\sum_{l=0}^{k-1}\sum_{h=0}^{k-1} x_{i,l}x_{j,h}\,\mathrm{span}(a_l, a_h) w_{i,j} && \qquad (2)\\
\text{s.t.} \quad & \sum_{j=0}^{k-1} x_{i,j} = 1 & \forall i \in \{0,\dots,k-1\} & \qquad (3)\\
& \sum_{i=0}^{k-1} x_{i,j} = 1 & \forall j \in \{0,\dots,k-1\} & \qquad (4)\\
& x_{i,j} \in \{0,1\} & \forall i,j \in \{0,\dots,k-1\}. & \qquad (5)
\end{aligned}$$

Each $x_{i,j}$ represents the assignment of vertex partition V_i to axis a_j. Naturally, each vertex partition can only be assigned to one axis (Eq. (3)), and each axis can only be assigned one vertex partition (Eq. (4)). As the weight of two vertex partitions and the span between two axes is known in advance, we model the objective function in Eq. (2) such that each quadratic term is only active if both vertex sets V_i and V_j are assigned to the respective axes a_l and a_h.

Simulated Annealing Heuristic: As an alternative to the exact computation above, we describe an approach based on simulated annealing [25] to optimize the axis order. Conceptually, simulated annealing is a probabilistic optimization technique that tries to compute the global optimum by transitioning through a sequence of states into the optimal state. To overcome local optima simulated annealing allows transitions into worse states if the remaining energy is sufficient. At the start, an initial amount of energy is assumed and gradually decreased until no more state transitions can occur and the procedure terminates. In the case of our approach, this means that we consider an ordering of axes as a state, switching two axes in the order as the transition between two states, and $\mathrm{cost}(\phi)$ is used to determine the current energy.

We initialize the approach with a random order of axes and iteratively transition into new states. Let ϕ_t be the state representing the current order of axes at step t. We transition into a neighboring state ϕ_{t+1} by exchanging two random axes in the order. Equation (1) is used to compute the cost of a state and let $cost_\Delta(\phi_t, \phi_{t+1}) = cost(\phi_{t+1}) - cost(\phi_t)$ be the absolute cost of a transition. If $cost_\Delta(\phi_t, \phi_{t+1}) < 0$ we immediately accept the new order as the current state. Otherwise, we draw p from a random uniform distribution with range $[0, 1]$ and use the following inequality to decide if the new order should become the current state even though it doesn't improve the global cost.

$$p < e^{-\dfrac{cost_\Delta(\phi_t, \phi_{t+1})}{\tau_t}}$$

Here, τ describes the remaining energy of the system. In practice, the initial energy should be chosen, such that 80% of all transitions are accepted. As the cost of transitions can initially only be estimated, we initialize the energy of the system to be $\tau_0 = \sum_{i=0}^{k-1} \sum_{j=i+1}^{k-1} w_{i,j}$. Our experiments have shown that this is a sufficient assumption. In each iteration, we decrease the remaining energy in the system with $\tau_{t+1} = \alpha\tau_t$ with $0 < \alpha < 1$ being the parameter controlling the cooling schedule. Lastly, we stop the approach if $\tau_t < 0.01$ and accept the order with the least cost as our solution.

As simulated annealing is sensitive to the initial energy, which is initialized by a rough estimate as stated above, we restart our simulated annealing approach with slightly different parameters and perform dual-phase simulated annealing [27]. In the second run, we initialize the remaining energy as the mean of all cost-positive state transitions and start with the best order from the first run as the initial order.

3.3 Crossing Minimization (3)

In the third stage of our framework, we are concerned with minimizing edge crossings under the assumption that the assignment to axes and the cyclic axis order is already fixed. We present multiple variants that can roughly be categorized as either using exact integer programming models or building on other well-known heuristics from the domain of layered crossing minimization.

All variants except one are envisioned as a two-step approach, where we first optimize inter-axis crossings (3a) before intra-axis crossings are minimized (3b). This results in solutions with the primary goal of minimizing the number of inter-axis edge crossings, while minimizing intra-axis crossings, which only become visible upon explicit axis expansion, is a secondary goal. In the first step, isolated vertices which are not an endpoint of at least one inter-axis edge can be ignored during edge crossing minimization and such vertices are removed and re-inserted after the optimization. For brevity we assume that V_i only contains non-isolated vertices in this section, otherwise, we will indicate it in the text. In the second step, intra-axis edge crossings are optimized under the constraint that the partial order of vertices computed in step one must be adhered to.

A commonality for all variants is that we subdivide all long edges to turn them into sequences of proper edges. Each edge $e = (u, v)$ with $\text{span}(u, v) > 1$ is subdivided by inserting $\text{span}(u, v) - 1$ virtual vertices assigned to the appropriate axes between $\alpha(u)$ and $\alpha(v)$. Whenever we differentiate between the two types of vertices, let V_i^r be the set of real vertices and let V_i^v be the set of virtual vertices with $V_i = V_i^r \cup V_i^v$. Additionally, we assume that a global parameter $g \geq 1$, which represents the maximal number of gaps per axis, is specified. If $g = 1$ we assume that all long edges are routed on the outside. In the case of $g = 2$, we assume that gaps are on the outside and inside of each axis. If $g > 2$, we evenly distribute the gaps along each axis; see Fig. 2b.

3.3.1 Integer Programming Models

We first describe several variants that use *integer programming* (IP) models. All models are based on the formulation by Jünger and Mutzel [14, 15], which we will briefly describe below. Here, we state the quadratic formulation while details on implementation, efficiency, and linearization of the model can be found in [31]. Let π_i and π_{i+1} be the vertex orders of two consecutive axes a_i and a_{i+1} with vertex partition V_i and V_{i+1}. Let $\delta^i_{u,v} \in \{0, 1\}$ be a binary variable describing the pair-wise order of vertices $u, v \in V_i$ in π_i, where $\delta^i_{u,v} = 1$ iff $\pi_i(u) < \pi_i(v)$ and 0 otherwise. Therefore, the vector $\delta^i \in \{0, 1\}^{\binom{|V_i|}{2}}$ fully describes the total order of vertices in π_i.

With the above, we can define the induced crossings of two neighboring axes:

$$C(\pi_i, \pi_{i+1}) = C(\delta^i, \delta^{i+1}) = \sum_{(u,v)\in\binom{V_i}{2}} \sum_{s\in N^{i+1}(u)} \sum_{t\in N^{i+1}(v)} (\delta^i_{u,v}\delta^{i+1}_{t,s} + \delta^i_{v,u}\delta^{i+1}_{s,t}) \tag{6}$$

Here, $N^{i+1}(u)$ describes the neighbors of u in V_{i+1}. Whenever u is before v in order π_i and t before s in order π_{i+1} a crossing is induced.

For the model to work correctly, the following transitivity constraints are necessary. For all triples $(u, v, w) \in V_i$ we have to add the following constraints:

$$0 \leq \delta^i_{u,v} + \delta^i_{v,w} - \delta^i_{u,w} \leq 1 \qquad \forall (u, v, w) \in \binom{V_i}{3} \tag{7}$$

This constraint implies that whenever $\pi_i(u) < \pi_i(v)$ and $\pi_i(v) < \pi_i(w)$ we also require $\pi_i(u) < \pi_i(w)$. Thus, preventing the contradiction that u is before v, v is before w and w is before u.

Gaps: To model gaps, we have to add additional constraints for $g > 1$. We explain the differences for $g = 1$ at the end of this paragraph. For each virtual vertex $u \in V_i$ we introduce variables $\beta^\kappa_u \in \{0, 1\}$ with $\kappa \in \{1, \ldots, g\}$ and the following constraints:

$$\kappa \frac{|V_i|}{g-1} \leq \sum_{j \in V_i^r} \delta^i_{j,u} + M_1(1 - \beta^\kappa_u) \qquad \forall u \in V_i^v \tag{8}$$

$$\kappa \frac{|V_i|}{g-1} \geq \sum_{j \in V_i^r} \delta^i_{j,u} - M_1(1 - \beta^\kappa_u) \qquad \forall u \in V_i^v \tag{9}$$

$$\sum_{\kappa=1}^{g} \beta^\kappa_u = 1 \qquad \forall u \in V_i^v \tag{10}$$

As stated earlier, we want an equal amount of vertices between potential gaps. Equations (8) and (9) model that a virtual vertex should be after a $\frac{\kappa}{g-1}$ fraction of the vertices in V_i^r. Thus, each virtual vertex is constrained to a position in a gap by requiring a predefined amount of vertices before it in the respective order. M_1 is a constant that must be initialized larger than the maximum number of vertices per axis. Now, if $\beta^\kappa_u = 1$ then Eqs. (8) and (9) express a tight bound. Otherwise, if $\beta^\kappa_u = 0$, then both inequalities are trivially fulfilled. Equation (10) states that only one β^κ_u can be 1. Thus, in combination Eqs. (8) to (10) state that a vertex must be in a gap position in the order π_i in exactly one gap. Now, for $g = 1$ we can drop β^κ_u and Eqs. (9) and (10), then, we adapt Eq. (8) such that each virtual vertex must be after $|V_i^r|$ vertices.

One-layer Two-sided Crossing Minimization ILP (1L2S ILP): In the first variant, we iterate a fixed number of times in clockwise or counter-clockwise order over all axes, minimizing inter-axis edge crossings. At each iteration step, we assume that the vertex order π^i of axis a_i is variable while the order π^{i+1} of the next axis a_{i+1} and the order π^{i-1} of the previous axis a_{i-1} are fixed. Now, at each iteration, we optimize the following objective function:

$$\text{minimize} \quad C(\pi^{i-1}, \pi^i) + C(\pi^i, \pi^{i+1}) \tag{11}$$

As π^{i-1} and π^{i+1} are considered as an additional input here, the quadratic terms in Eq. (6) simplify to linear terms in Eq. (11). The previously mentioned constraints of Eqs. (7) to (10) only have to be added for vertices in V_i.

Two-layer Two-sided Crossing Minimization IQP (2L2S IQP): In the second variant, we apply a similar iterative approach as for the one-layer two-sided variant. At each iteration, we optimize the order of two consecutive axes a_i and a_{i+1} and assume that π^i and π^{i+1} are variable. The order of vertices π^{i+2} and π^{i-1} are considered fixed. Now, at each iteration, we optimize the following objective function:

$$\text{minimize} \quad C(\pi^{i-1}, \pi^i) + C(\pi^i, \pi^{i+1}) + C(\pi^{i+1}, \pi^{i+2}) \tag{12}$$

The part $C(\pi^i, \pi^{i+1})$ includes quadratic terms in the objective function, while the other two terms remain linear. Thus, it is an integer quadratic program. Lastly, constraints of Eqs. (7) to (10) have to be added for vertices in V_i and V_{i+1}.

Full Crossing Minimization IQP (Full IQP): In this variant we optimize all inter-axis edge crossings at once instead of iteratively applying a crossing minimization model to individual axes. Thus, the objective function is as follows:

$$\text{minimize} \quad \sum_{i=1}^{k-1} C(\pi^{i-1}, \pi^i) + C(\pi^{k-1}, \pi^0) \tag{13}$$

Here, each term in the objective function is a quadratic term. Each δ^i is considered variable and we have to add the constraints of Eqs. (7) to (10) for all V_i.

Intra-axis Edge Crossing Minimization: All above models have in common that they only optimize inter-axis edge crossings of the proper edges (after subdividing the long edges). Moreover, they specifically address the intent of stage (3a) of the pipeline. Now, as the second stage (3b) of the crossing minimization step we will minimize intra-axis edge crossings using the base IP model described above while adding additional constraints to disallow changing the previously computed order. Here, vertex partition V_i represents all vertices instead of the subset with inter-axis connections. We can re-introduce the previously removed vertices and then apply the one- or two-sided IP model to individual axes. The gap constraints of Eqs. (8) to (10) are still used but require virtual vertices to be placed in the appropriate gaps dependent on the full vertex sets. Remember, intra-axis crossings are induced by the duplicated vertices; however, the order of vertices is identical, i.e., $\pi_i^+ = \pi_i^- = \pi_i$. Thus, the induced crossings are $C(\pi_i^-, \pi_i^+) = C(\pi_i, \pi_i) = C(\delta^i, \delta^i)$ and our objective function is as follows:

$$\text{minimize} \quad C(\pi_i, \pi_i) \tag{14}$$

Naturally, we have to add the constraints of Eqs. (7) to (10). However, we have to introduce additional constraints that fix $\delta^i_{u,v}$ to the previously computed value iff u and v have been optimized in stage (3a).

Lastly, this model can be run as either a two-sided crossing minimization IQP or as a one-sided crossing minimization ILP. In the case of the two-sided IQP, we initialize all constraints and consider $\pi_i^+ = \pi_i^- = \pi_i$; thus, the terms in Eq. (14) imply a quadratic program. In the case of the one-sided ILP, we iteratively improve the order by considering π_i^- to be fixed by the previous iteration while π_i^+ is variable. After each iteration, we set $\pi_i^+ = \pi_i^-$.

Full-weighted Integer Quadratic Programm (Full-weighted IQP): The last variant we want to describe optimizes both inter- and intra-axis crossings at the same time, therefore combining stage (3a) and (3b), with the following objective function:

$$\text{minimize} \quad \sum_{i=1}^{k-1} C(\pi^{i-1}, \pi^{i}) + C(\pi^{k-1}, \pi^{0}) + \omega \sum_{i=0}^{k-1} C(\pi^{i}, \pi^{i}) \tag{15}$$

The model uses the same building blocks as previously seen. The main difference is that we weigh the terms in the objective function Eq. (15). The weight factor ω can be used to balance between inter- and intra-axis crossings. As we prefer to optimize inter-axis crossings, we set it to $\omega = \frac{1}{|E|^2}$, thus using an upper bound on the number of crossings. Still, we have to add the constraints of Eqs. (7) to (10) for each vertex set V_i, and the objective function remains a quadratic function. Informally, this means that we always find a solution with the least number of inter-axis edge crossings, but, if there exist multiple such solutions, we prefer the one with the least intra-axis edge crossings.

3.3.2 Heuristics

In this section, we describe several heuristic algorithms that are based on the sifting [20] and barycenter heuristic [26]. Again, we subdivide all long edges to turn them into sequences of proper edges and remove isolated vertices. While the variants differ drastically in how the orders are computed, they share the commonality of how gaps are handled during the respective procedures.

Therefore, we will first describe a greedy procedure to handle gaps. We assume that for each vertex, a position in the order π_i has already been computed, but virtual vertices can still be in non-gap positions in the order. If $g = 1$ we simply want virtual vertices on the outside to route the long edges around axes. Here, we sort vertices by their computed position but constrain the sorting algorithm to put all non-virtual vertices before all virtual vertices in the respective orders. Afterward, we assign the index of a vertex as the new position. For $g > 1$ we perform the following procedure. We split the order π_i of axis a_i into $g - 1$ suborders such that the number of real vertices V^r is approximately equal in each. This places the gaps equidistantly between the real vertices. Now, we iterate over all virtual vertices in each split order and compute the induced crossings if we move the vertex to either the beginning or the end of the split order. Then, we greedily pick the choice that induces fewer crossings. Note, that we do not change the relative order of virtual vertices to not introduce crossings between them. Figure 4 illustrates how the greedy gap assignment works. Once we completed the above procedure and processed all split orders, we merged the split orders again. We assign the index of a vertex in this order to its new position.

Barycenter Heuristic: As the first variant we adapt the barycenter heuristic [26] for crossing minimization. Our approach works by iterating several times in clockwise or counter-clockwise order over all axes while performing a layer-by-layer crossing minimization sweep. At each iteration, we process all vertices V_i of an axis by computing a new barycenter position as follows:

$$\text{pos}(u) = \frac{1}{|N(u)|} \sum_{v \in N(u)} \frac{\pi_{\alpha(v)}(v)}{|\pi_{\alpha(v)}|}.$$

As it is necessary to avoid cases where axes are imbalanced, we normalize both axes and consider the neighborhood $N(u)$ of vertex u in the next and previous axes a_{i+1} and a_{i-1}. The reason for

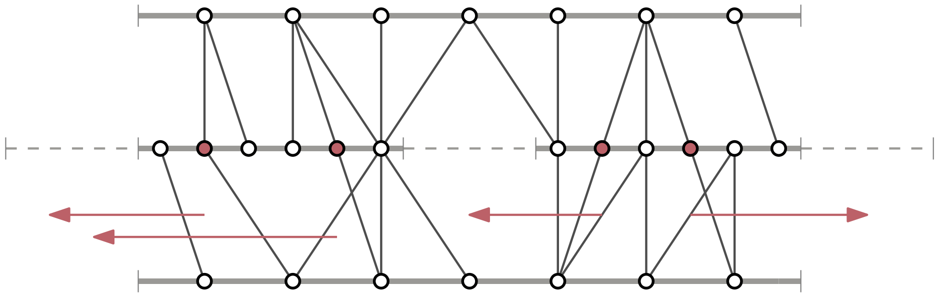


Figure 4: An axis with $g = 3$ gaps. Gaps are indicated by dashed lines, while axis segments are solid lines. Vertices colored red are virtual vertices. After computing new positions we move virtual vertices into either a gap to the left or right. The side is determined by counting the crossings and greedily picking the better option. The order of virtual vertices remains unchanged after the procedure.

considering both axes simultaneously is that otherwise, when only considering the previous axis, crossings might be introduced that are overall worse for the next axis.

Once barycenter positions are calculated, we sort all vertices $v \in V_i$ of axis a_i by their positions $\text{pos}(v)$ and apply the gap handling procedure. After each iteration, we batch update the vertex positions with their respective new positions. We terminate the overall process after either no change is detected for one cycle or an iteration threshold is reached.

Layer-by-layer Sifting: The next variant we want to describe is built on the sifting heuristic. See [20] for a more detailed overview of the general sifting algorithm. Similar to the barycenter heuristic, we process axes by iterating several times in clockwise or counter-clockwise order over all axes while performing a layer-by-layer crossing minimization sweep. Each sweep starts with computing the number of crossings that are induced with the current order of an axis. Then we process all vertices in order of their previously assigned position. For each vertex, we perform the following: we iteratively move the vertex from its current position to the first position in the order while computing the number of induced crossings for each intermediate position. The same procedure is repeated but the vertex is moved to the last position in the order. Then, we move the vertex to the position where we encountered the least number of overall crossings. We repeat this process for the remaining vertices on the updated order. Once all vertices are processed, we perform the gap handling procedure and assign the position in the order as the new vertex position.

Note, that [20] investigated several variants of orders in which the vertices are processed but didn't find a statistically significant difference in the resulting number of crossings.

Global Sifting: The last variant we want to describe is based on global sifting [20]. Here, instead of iteratively processing the vertices layer-by-layer we sort all vertices in all layers by the respective degree. Similarly to layer-by-layer sifting, we iteratively process each vertex by moving it to the optimal position in the order of its axis. Once all vertices are processed, we perform the gap handling procedure and assign the position in the order as the new vertex position.

Heuristics for Intra-Crossing Minimization: In the second phase of the crossing minimization we aim to further reduce intra-axis crossings by applying the barycenter or sifting heuristic.

As the focus of the hive plot layout is on proper and long edges, we introduce the additional constraint that the relative order of vertices incident to inter-axis edges are not allowed to change in this phase anymore. Basically, this is again a classic 2-layer crossing minimization for a vertex subset, however both layers have the same order. Moreover, we apply the same procedure as described above to constrain virtual vertices to gap positions. We process each axis individually and terminate the processing of an axis once no change is detected, or the iteration threshold is reached.

4 Implementation and Hive Plot Rendering

In this section, we will briefly explain the design decisions regarding the visualization; see Fig. 5 or Fig. 14 for examples. The implementation code and a prototype web application can be found on OSF. Axes are drawn as straight lines emanating from a common center with equal angular distribution. Axes can be expanded or collapsed on demand.

When an axis is expanded, the background is colored in a light grey color with low opacity. When expanded, the available space is distributed 40 : 60 between intra-axis and inter-axis edges. Vertices are drawn as small circles, and their positions on their respective axis a_i are computed based on their positions in π_i. Labels are placed in a clockwise direction next to an axis horizontally. If a vertex' assigned axis differs by less than $\pm$ 25° to the horizontal reference direction, the label is rotated by 45°. The color of vertices is computed by mapping the angle of the assigned axis to a radial color map [6]. For edges, we assign the color of the first endpoint in the counter-clockwise direction. The ideas behind coloring edges are that it becomes easier to follow individual edges and that it is, for half of the vertices, immediately clear to which axis the edge connects.

Edge Routing: Edges between neighboring axes are drawn as cubic Bézier curves with one additional control point for each endpoint. Those two extra control points are positioned perpendicular 0.15 of the straight-line distance between the associated endpoint and a mirrored copy on the other axis. For long edges which are routed through gaps we use B-splines. Here, the control points are computed by chaining the control points of proper edges with only one control point for each virtual vertex.

Interactivity: Figure 5d shows an example of how interactivity was realized in our visualization. First, when hovering a vertex, the vertex itself, all neighbors, and incident edges are highlighted by a color contrasting the color scheme. Initially, each axis in the visualization is collapsed. By clicking on a single axis it is expanded to show more details on demand. Furthermore, it is also possible to expand all axes with a button click. Naturally, it is also possible to contract individual pairs of expanded axes. Similarly, by clicking a button in the interface, vertices can be scaled to represent their respective degree; see Fig. 5b. Lastly, labeling can be toggled on or off.

5 Case Study

We evaluate our framework by a case study using the *citation* dataset [7] from the creative contest at the 2017 Graph Drawing conference. This dataset contains all papers published at GD from 1994 to 2015. We created co-authorship graphs for different years by extracting researchers from papers and connecting them with edges whenever they co-authored a paper. Full-sized images of the case study can be found on OSF.

In Fig. 5, we show a hive plot of the co-author network of 2015 and three alternative renderings computed with our framework in less than 10ms. We used the rendering style described in Section 4. We did not specify the number of axes k in the input and computed a clustering with the Louvain method, which yielded seven clusters. We specified the number of gaps as $g = 1$, i.e., all long edges are routed around the axes. The network has a total of 75 vertices and 190 edges, which are split into 172 intra-axis edges, 15 proper edges, and 6 long edges.

Authors mapped to individual axes seem to represent mainly clusters of geographic proximity of researchers' institutions or established close collaborations. Inter-axis edges are emphasized in our hive plots and indicate collaborations between clusters in the form of researchers bridging institutions and forming new connections, e.g., via papers originating from research visits or recent changes in affiliation. Another possible interpretation can be seen when vertices are scaled by degree. Researchers with connections to other axes are often also highly connected inside their own cluster. This could mean that they are well-connected and prolific and use existing connections to start new collaborations.

In contrast to a force-based layout, see Fig. 6a, several observations can be made. While cliques are very prominent in the force-based layout, the macrostructure of the graph is less clear. The hive plot layout, on the other hand, focuses more on the macro structure of the graph, with the intra-axis structure only shown on demand. However, with two copies per vertex, cliques are harder to identify. Still, the hive plot layout requires no additional cue, such as color, in this case, to highlight the community structure. Furthermore, in the hive plot layout, individual vertices are easier to identify, labels are more uniform, and edges are routed in a predictable manner, which is similar to schematic diagrams. Due to the possibility of expanding axes on demand in the hive plot layout, individual communities can be easily explored without being overwhelming. While it is possible to represent communities in the force-based layout by meta vertices, it is not straightforward to encode the relationship of each single vertex to the rest of the network. Both layouts show some label-edge overlap. Finally, the hive plot layout has a more balanced space utilization.

Furthermore, we also compared against a hierarchical layout; see Fig. 6b. Due to the absence of edge directions in the underlying data, we derived edge directions from the hive plot layout by directing all edges clockwise. Naturally, the hierarchical layout emphasizes the imposed hierarchy while the communities are dispersed over the layout. The communities are visible, although this requires the use of vertex coloring. This layout gives the layer assignment a quite different meaning compared to the axis assignment in our approach. The orthogonal layout of edges initially simplifies the process of following an edge, but it becomes progressively more challenging with an increase in bends and crossings. In the hive plot layout this is less of an issue, especially for edges between communities. Lastly, the label placement in the hierarchical layout is optimized and avoids label-edge and label-vertex overlaps. However, this optimization comes at the cost of requiring more space. In contrast, the hive plot layout has a few label-edge overlaps but utilizes space more efficiently.

Summary: Hive plots are an interesting alternative for investigating social networks. Especially, as the complexity of dense clusters can be hidden and shown on demand. Furthermore, a Hive plot layout gives prominence to edges connecting different clusters. However, investigating the intra-axis edges of a Hive plot layout can be overwhelming. This is mainly caused by many edges in a small visual area. Similarly, the visual scalability of a Hive plot layout is probably worse than other approaches. Perceiving edges in Hive plots with more than 8-12 axes is already hard. Moreover, too many long edges cause a Hive plot layout to become quickly unreadable and the constraint that edges are routed through gaps can cause unnecessary crossings.

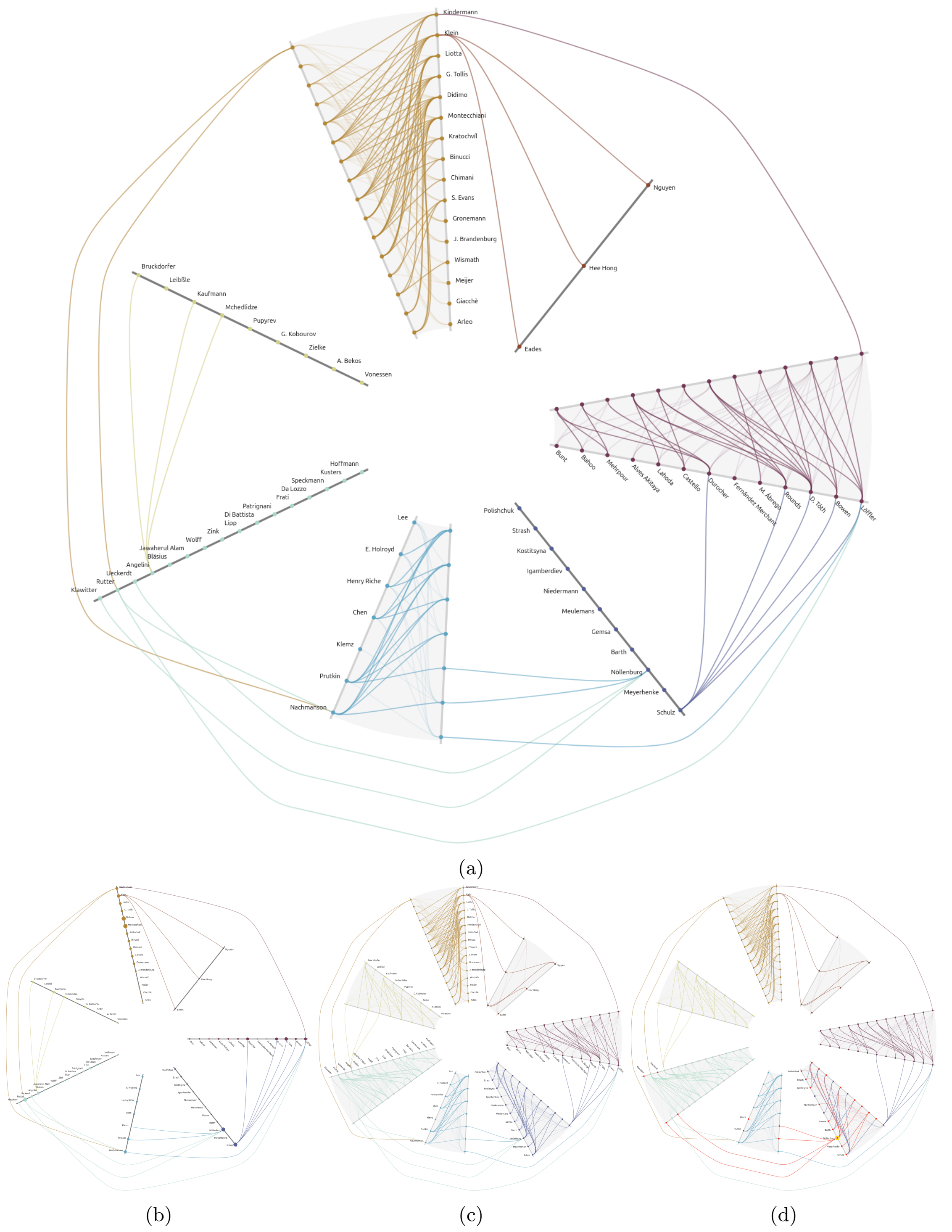
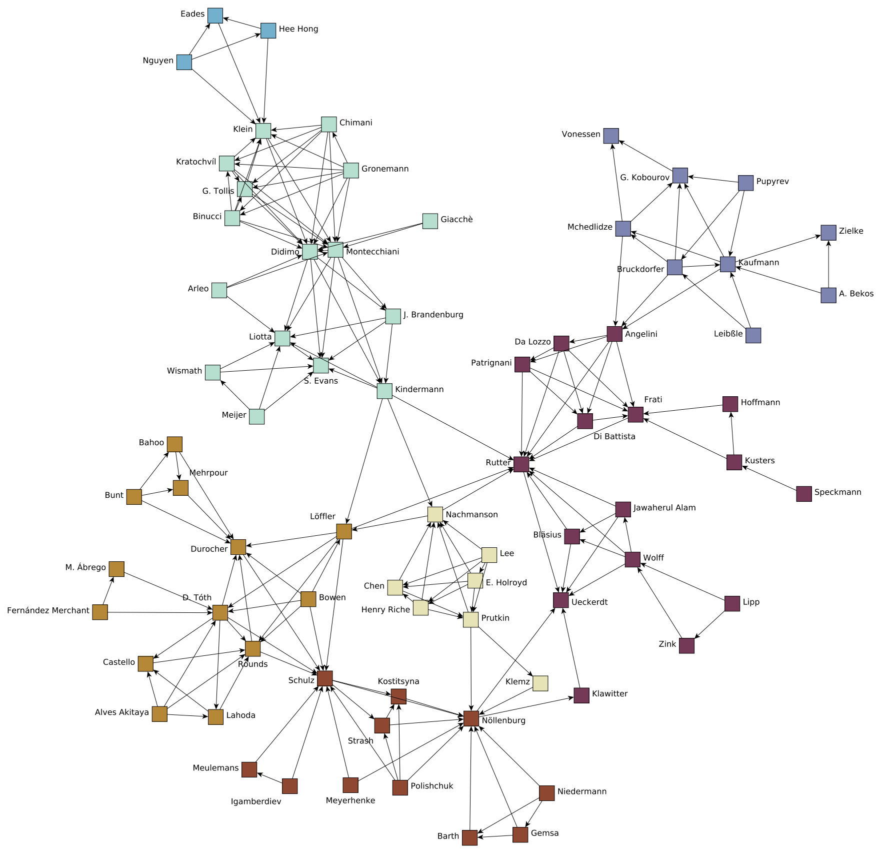
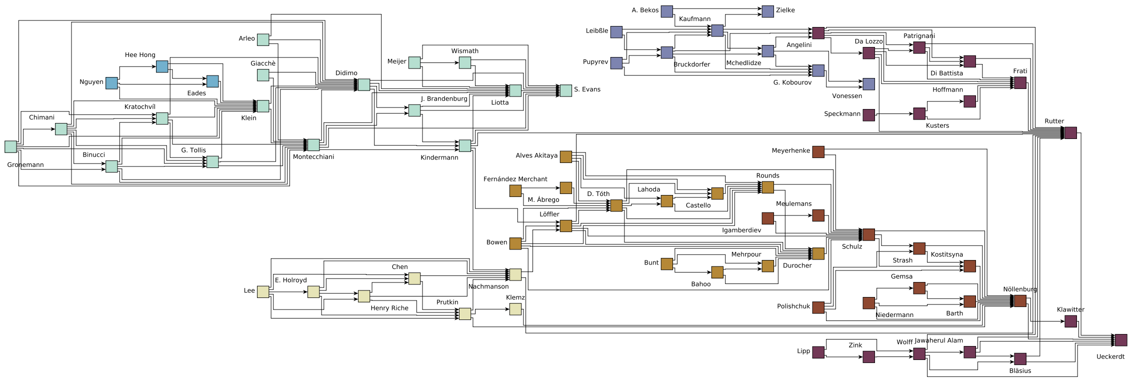


Figure 5: Variations of the co-author graph of GD 2015. In (a), some axes of interest are expanded, while (c) shows all axes expanded. In (b) vertices are scaled by degree and all axes are collapsed. In (d) interactive highlighting is obtained by hovering the vertex “Nöllenburg” marked with a yellow disk.



(a) force-based



(b) hierarchical

Figure 6: Force-based layout (a) of the co-authorship graph of Section 5 created with yEd [30]. The “smart organic layout” functionality was used. The preferred edge length was set to 100, while the minimal vertex distance was set to 20. The option of avoiding vertex/edge overlaps was active with a value of 0.8. Also, the labeling was optimized, and the graph was colored according to the partitions from the case study. (b) shows a hierarchical layout created with yEd. Similar layout optimization steps were applied.

6 Computational Experiments

We conducted computational experiments to test different combinations of the pipeline. First, we conducted a set of experiments to tune the parameter settings of the simulated annealing heuristic and the iterative crossing minimization heuristics and integer programs. Second, we compared different combinations of the pipeline in regards to the optimization criteria, number of crossings, and number of long edges, as well as in regards to the runtime. The datasets, the evaluation code, and additional plots can again be found in the supplemental material on OSF.

6.1 Dataset and Setup

All graphs used in the experiment were generated with the following approach where we state the parameters first and then the intention of each parameter: k, $\overline{n}$, σ_n, δ_{intra} and δ_{inter}.

To circumnavigate the problem of having an uncontrolled distribution of long edges we already assume that the order of axes is known. For each of the k axes, we generate a vertex set of mean size $\overline{n}$ and standard deviation of σ_n. Sampling the number of vertices from a normal distribution with a given mean and standard deviation gives us control over the approximate number of vertices while the size of the graph remains to some degree random. Afterwards, we have k vertex partitions V_i where V_i is on axis a_i next to V_{i+1} on a_{i+1}.

Then, to add edges, we fix the average degree of vertices. For intra-axis edges, we fix the average degree δ_{intra} and perform a random experiment between all vertex pairs $u, v \in V_i$. We draw a random value p from a uniform distribution with range $[0, 1]$ and check if $p < \frac{\delta_{\text{intra}}}{|V_i|}$. Whenever the inequality holds we add an edge between u, v.

For inter-axis edges, we assume that the average degree is δ_{inter} but use a slightly more complex procedure than for the intra-axis edges. To avoid having a very uniform distribution of inter-axis edges, we prefer having more proper edges than long edges, which models the intended use-case of hive plots better, as too many long edges decrease the readability. Now, for each pair of vertices $u \in V_i$ and $v \in V \setminus V_i$ We draw a random value p from a uniform distribution with range $[0, 1]$ and check the following inequality:

$$p < p_{\text{inter}} p_{\text{long}}^{\text{span}(u,v)-1}$$

The span of two vertices is known as we fixed the axis each vertex set is assigned to. The input parameter $p_{\text{long}} = 0.2$ is fixed and models that the further two vertices are apart, regarding their assigned axes, the less likely it is that an edge exists between them. Moreover, if two vertices are on neighboring axes (span $= 1$), the term becomes inconsequential. $p_{\text{inter}} = \frac{\delta_{\text{inter}}}{|V\ V_i|}$ is derived from the input parameter of δ_{inter}. If the sampled p satisfies the inequality, we add the edge (u, v) to the graph.

Lastly, to have a less uniform distribution of edges on each axis, we create hub vertices. For each axis, we select 1 to σ_n vertices uniformly at random. For each edge that has an endpoint in the remaining vertices of the axis, we perform a random experiment. With probability 0.1, we disconnect the edge from its endpoint on the axis and connect it uniformly at random to one of the selected hub vertices instead.

We varied the number of axes k from 3 to 14 and the vertex number per axis via the parameters $(\overline{n}, \sigma_n)$ from the set $\{(10, 2), (20, 4), (30, 6), (40, 8), (50, 10)\}$. We introduced three densities: *low* ($\text{deg}_{\text{inter}} = 0.5$, $\text{deg}_{\text{intra}} = 4.0$), *medium* ($\text{deg}_{\text{inter}} = 1.0$, $\text{deg}_{\text{intra}} = 5.0$) and *high* ($\text{deg}_{\text{inter}} = 1.5$, $\text{deg}_{\text{intra}} = 6.0$). We generated five graphs for each density and each combination of k and n. In total, our experimental synthetic dataset consists of $5 \times 3 \times 12 \times 5 = 900$ instances. In summary, we

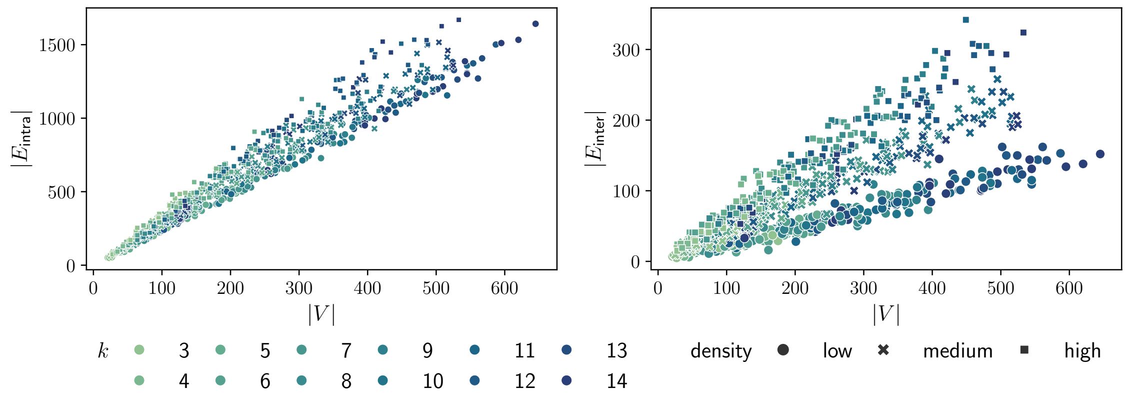


Figure 7: Distribution of number of inter- and intra-axis edges in our experimental dataset by number of vertices in a generated instance. Our density parameter mostly affects inter-axis edges.

generated instances that represent graphs that exhibit dense community structure of k communities with a mean of $\overline{n}$ vertices. Furthermore, communities are connected to each other with decreasing likelihood correlating with how distant two communities are assumed during the generation process. The distribution of the numbers of inter- and intra-axis edges can be seen in Fig. 7. As expected, the density of intra-axis edges is much higher than the density of inter-axis edges. We chose this approach over existing community graph generators, for example, LFR [29] or random partition graphs [10], as we have better control over the number of axes, the number of vertices per axis and the distribution of proper and long edges, especially, as we want more proper edges than long edges. Furthermore, we tried to cover the parameter space of where we think a hive plot layout is sensible, while also testing the computational boundaries when scaling to large instances.

We implemented all algorithms in Python 3.11 and used Gurobi 10.0.3 to optimize our IP models. We ran the experiments on a compute cluster using Ubuntu 22.10 with an Intel Xeon E5-2640 2.40GHz. However, we restricted Gurobi to only use one core and restricted the memory to 32 GB. We used the Gurobi model tuning tool to find appropriate solver parameters for our models, which resulted in a speed-up of up to one order of magnitude compared to Gurobi's default settings. Also, we tried to linearize the IQP models but observed degraded performance compared to Gurobi's automatic linearization.

6.2 Tuning the Pipeline

In the first set of experiments, we investigated tuning the simulated annealing heuristic and the iterative crossing minimization variants of our pipeline. First, we looked at simulated annealing for the circular arrangement problem of computing an optimal axis order. Here, we used the medium density instances with $\overline{n} = 50$ and additionally added instances with up to $k = 26$ axes. We compared linear ($\tau_{t+1} = \tau_t - \alpha$) and geometric ($\tau_{t+1} = \alpha\tau_t$) cooling strategies with different values of α. Additionally, we tested each cooling strategy with and without dual-phase annealing. For each instance we computed the wall-clock runtime of the algorithm and recorded the total number of iterations. As simulated annealing is sensitive to random effects, we performed five runs for each cooling strategy and averaged the results. Lastly, we also recorded the time and optimal cost of the IQP model of each instance.

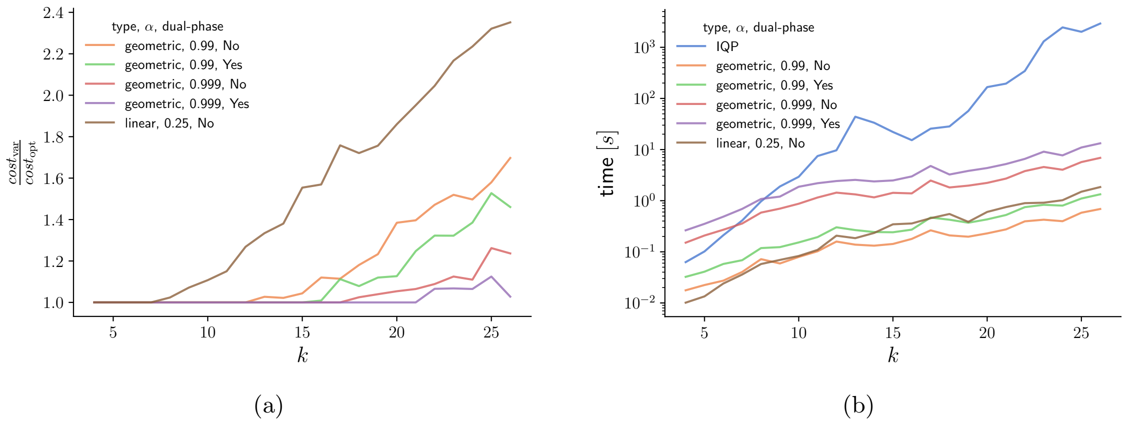


Figure 8: Plots of exact-to-heuristic ratio (a) and runtime (b) of the best performing cooling strategies when the number of axes is increased. Note that (b) has a logarithmically scaled y-axis. Each tuple represents the cooling schedule, the value of α, and if we used dual-phase annealing.

Figure 8a shows the best-performing cooling strategies compared to the optimal solution. All linear cooling strategies performed considerably worse than the geometric cooling strategies. With the geometric cooling strategy and $\alpha = 0.999$ we were able to compute the optimal solution in all instances up to $k = 16$. The double start improved the overall result in most cases while not drastically increasing the runtime.

When looking at the runtimes in Fig. 8b, we can see that the exact IQP model is able to solve all instances to optimality, but, only small instances with $k < 10$ can be solved in less than a second. The geometric cooling strategy with $\alpha = 0.999$ terminates in less than 10 seconds for all k while the geometric strategy with $\alpha = 0.99$ terminates in less than one second for all k. Considering that hive plot layouts become increasingly difficult to read when k increases beyond 10 (see Fig. 14), it is reasonable to assume that a geometric cooling strategy with $\alpha = 0.99$ and double start is sufficient and will find an optimal or near-optimal solution with high probability quickly.

Lastly, we looked at tuning the maximum number of iterations for the iterative heuristics and IP model. We randomly sampled one of the five instances per parameter setting of the medium-density dataset. Then, we computed the optimal axis order for each instance and performed inter- and intra-axis crossing minimization. For each iterative variant, we doubled the maximum number of iterations and recorded the number of crossings. Then, we computed the iteration number where no change in crossings occurred for each variant. The plots can be found in the supplemental material. For the barycenter, global sifting, and layer-by-layer sifting heuristics, we could not detect a change in the number of crossings after 16 iterations for most instances. For the 1L2S ILP, we could not detect a change after 8 iterations, while the 2L2S IQP converged already after four iterations.

6.3 Evaluation of the Pipeline

To compare the different proposed variants of our pipeline, we evaluated all instances of our dataset with several pipeline configurations. See Table 1 for an overview of the eight different combinations. Generally, we can distinguish between combinations that use an IP model and combinations that

	axis ordering	**inter-axis CRM**	**intra-axis CRM**
Heu.	simulated annealing	barycenter	barycenter
	simulated annealing	layer-by-layer sifting	sifting
	simulated annealing	global sifting	sifting
IP	IQP	1L2S IQP	one-sided ILP
	IQP	2L2S IQP	two-sided IQP
	IQP	full-model IQP	one-sided ILP
	IQP	full-model IQP	two-sided IQP
	IQP	full-weighted IQP	

Table 1: Variants of the pipeline used in the experiment.

don't use an IP model. In the case of variants that used an IP model, we set a timeout of 120 seconds for iterative per-axis models and 1800 seconds for full models. We measured the wall-clock time for each pipeline step. We also measured the number of inter- and intra-axis edge crossings. Lastly, we measured the total edge length but as explained in Section 6.2 we could only observe minimal differences between the solutions of the optimal IQP and simulated annealing of stage (2). Thus, we will not investigate it further in this section.

Figure 9 shows the percentage of instances our IP models were able to solve to optimality before a timeout occurred. We aggregated data for the number of gaps $g = \{1, 2, 3\}$; the individual plots can be found in the supplemental material on OSF. There is a slight indication that with three gaps and a small number k of axes, timeouts occurred. Unsurprisingly, the full-weighted IQP model is only able to solve rather small instances before running into timeouts, while the 1L2S and one-sided model for inter- and intra-axis CRM can compute a solution for almost all instances. The full model IQP variant is still able to find an exact solution even if the number of vertices increases. However, it starts to struggle with an increasing number of axes and edges. Interestingly, when combined with the two-sided IQP model for intra-axis CRM, it seems that timeouts are mostly caused by the two-sided IQP.

Next, Fig. 10 shows the $\log_2(\frac{\mathrm{cr}_{\mathrm{alg}}}{\mathrm{cr}_{\mathrm{opt}}})$ optimality ratio of inter-axis edge crossings of different pipeline combinations against the runtime each pipeline combination required to perform inter-axis CRM. This plot only shows those instances, regardless of the number of gaps or density, that were solved optimally with the full IQP model, i.e., for which we can compute the actual optimality ratios. Regarding the runtime, the heuristic pipeline with the barycenter heuristic can compute a solution in less than $100ms$ while sifting and global-sifting can take up to $10s$. The variants based on IP models take considerably longer to compute a solution and only a fraction of the instances can be solved in less than $10s$. Interestingly, if it is possible to compute a solution with the full IQP model, then the runtime behavior is similar to the iterative IP models (see the rug plot at the bottom of the figure).

Regarding solution quality, all three iterative heuristics perform equally well, while the 1L2S ILP model is slightly better. On average, the three heuristics achieve 2.88 (barycenter), 2.77 (layer-by-layer sifting), and 2.82 (global sifting) optimality ratio. However, in some instances a ratio of up to 128 is observed. The 1L2S ILP model achieves a 2.11 ratio. The 2L2S IQP performs best of all iterative models with 1.28 ratio and can even find optimal solutions for many instances. However, of all approaches besides the full-weighted model, the most timeouts occurred.

If we look at the instances in Figure 11, where we were not able to compute an optimal solution before a timeout occurred, the distribution of runtimes of the iterative algorithms is

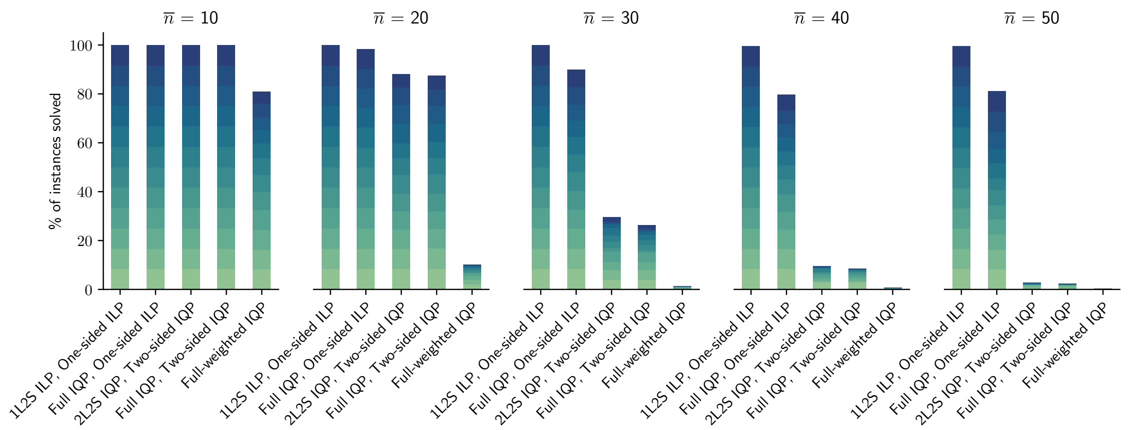


Figure 9: Percentage of instances an IP-based pipeline combination was able to solve before a timeout occurred. Each bar represents a combination of variants for a given number of $\overline{n}$. The color gradient indicates an increasing number of axes, with light green representing $k = 3$ and dark blue $k = 14$.

similar. However, the iterative 1L2S IQP model outperforms other variants regarding the number of crossings. If we compare the performance of the best variant for an instance with all other variants, then the performance ratio is at most 8. As the worst ratio for the 1L2S IQP model is already 40 for instances where we were able to compute the optimal solution, we can assume that the overall ratio further deteriorated.

Lastly, we also compared solution quality to a number of inter-axis edges of an instance. Here, we can report that with an increasing number of edges, the variance of solution quality becomes smaller for all variants. A plot can be found in the supplemental material.

Effect of Gaps: Now, if we consider the influence of the number of gaps on the behavior of the inter-axis crossing minimization in Fig. 12, two general trends are noticeable. Here, we only show the exact full IQP model and the fast barycenter heuristic. The behavior for other variants is similar and can be found in the supplemental material.

First, in Fig. 12a, adding gaps increases the runtime as we add more variables in the IP models. Similarly, the runtime for the barycenter heuristic increases. However, the driving factor here is the gap-handling procedure as the procedure does not increase in complexity with the number of gaps. The runtime remains similar for $g = 2$ and $g = 3$. When looking at the distribution of points in the scatterplot, it can be observed that fewer instances are solved optimally when the number of gaps increases.

Second, in Fig. 12b, the number of crossings drastically decreases in the full model when gaps are introduced, which can be expected in an optimal model when introducing an additional degree of freedom and enlarging the solution space. Interestingly, though, in the barycenter variant, the number of crossings generally decreases from $g = 1$ to $g = 2$ but increases again from $g = 2$ to $g = 3$. To further investigate we calculated the number of gaps associated with the best solutions observed for each pipeline variant in each instance. The table of the absolute and relative values can be found in the supplemental material. All three iterative heuristics had the best solution with two gaps in 75% and 83% of all instances. The iterative 1L2S and 2L2S IP models mostly found

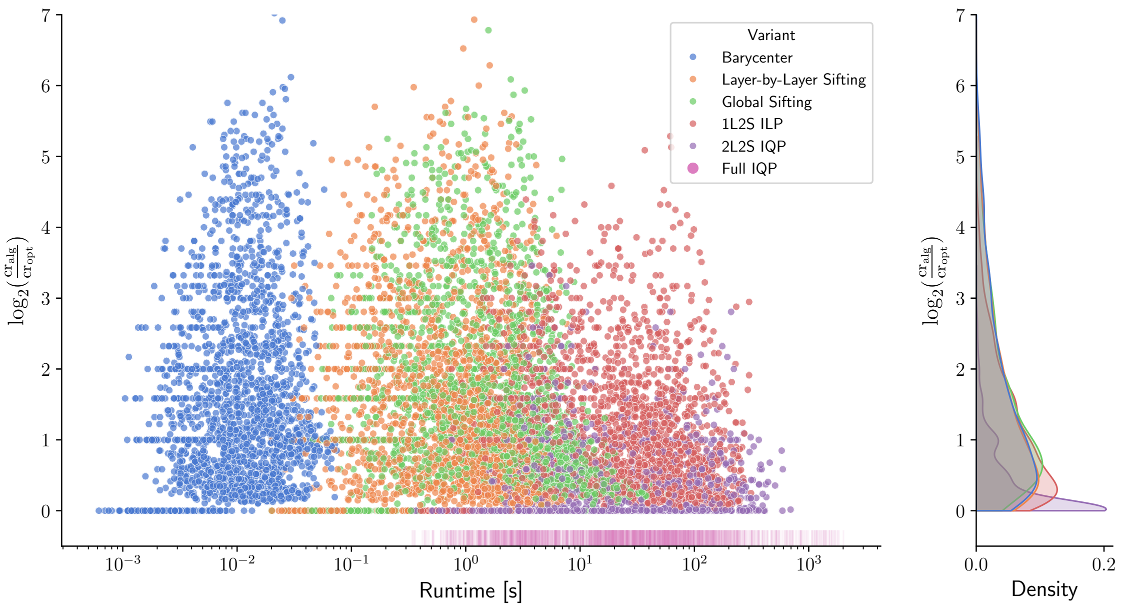


Figure 10: $\log_2$ optimality ratio of inter-axis crossings in regards to the runtime of five different heuristics. Only instances are shown where the full IQP model computed an optimal solution within the time limit. The runtime distribution of the full IQP model is shown as a rug plot on the bottom. A density plot of the $\log_2$ optimality ratio is shown on the right.

the best solution with three gaps, however, in less than 10% of all instances the best solution was found with one or two gaps. One explanation for this could be that gap handling, in combination with iterative approaches, restricts the solution space and leaves the optimization stuck in a local minimum.

Intra-axis Edge Crossings: Figure 13 shows the number of intra-axis crossings and runtime of the different pipeline variants. In this plot, we compared against the optimal solution found with the full-weighted IQP model. The behavior of each approach is similar to what can be observed for inter-axis CRM regarding the runtime. Again, the barycenter heuristic is much faster than all other approaches. However, the solution quality drastically differs. For both pipeline combinations using the two-sided IQP fewer crossings can be observed. Here, the ratio is on average 1.24 of the optimal solution. Surprisingly, barycenter performs better (1.53) than the remaining combinations. The other combinations perform equally (~ 2.0). The plot for all instances that could not be solved optimally with the full-weighted IQP can be found in the supplemental material. However, the general patterns remain the same.

Lastly, the overall performance regarding the total number of crossings and runtime does not deviate from what can be observed from the individual pipeline steps.

Recommendations: To summarize, while the one-sided and two-sided iterative IP models perform generally better than the heuristic models in terms of crossing minimization, it is questionable if the increase in runtime is justifiable in many use cases. Furthermore, the iterative IP models

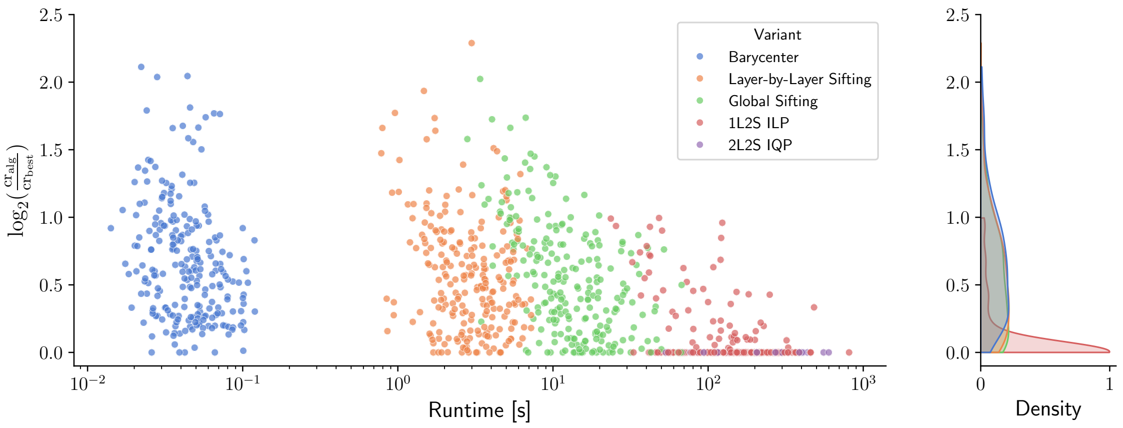


Figure 11: $\log_2$ performance ratio of inter-axis crossings in regards to the runtime of five different heuristics. Only instances are shown where the full IQP model ran into a timeout before an optimal solution was found. The ratio considers the number of crossings of a variant divided by the crossings in the best solution found for each instance. A density plot of the $\log_2$ ratio is shown on the right.

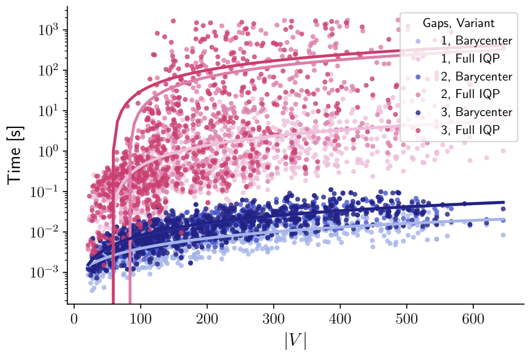
performed worse than the full models regarding the number of crossings, but, the overall runtime of computing a smaller model for each axis several times quickly adds up and is similar to computing the full model with its provably optimal solutions. However, one redeeming quality of the iterative IP models is that they use less memory. Several of the largest instances ran out of memory on our system for the full-weighted IQP model.

Moreover, a similar pattern can be observed for using an IP model to optimize the circular arrangement problem for finding an optimal axis order. The simulated annealing heuristic is highly likely to find an optimal or nearly optimal solution. However, under our current use case assumptions it is unlikely to actually use a hive plot layout with more than 14 axes, thus, computing an optimal solution can still be done in a sensible time frame.

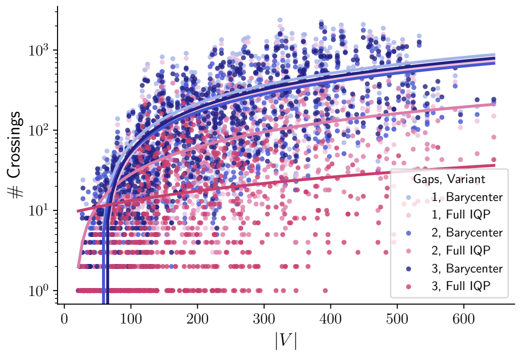
To conclude, if optimality is not required, the heuristic variants provide a good alternative to computing an optimal solution. As the performance regarding crossings was similar for all three heuristics, but the barycenter heuristic computed a solution fastest, we recommend the barycenter heuristic as a good trade-off between speed and quality. However, if optimality is required or longer computation times can be tolerated, then the full or full-weighted IQP should be preferred. The full-weighted model can reliably solve instances up to 100 vertices, but if intra-axis edge crossings are less of a concern, the full model is able to solve instances with up to 500 vertices and 400 edges.

7 Discussion and Conclusion

We have introduced a combinatorial framework for optimizing and drawing hive plots. Our framework introduces a pipeline that can compute fast heuristic solutions and exact solutions for each stage. Our edge routing guarantees that vertices are never occluded by edges, which is generally not the case in frequently used algorithms such as force-based layouts. The focus of our approach lies on showing inter-axis connections, i.e., proper and long edges, which reduces the visual complex-



(a)



(b)

Figure 12: Runtime (a) and number of inter-axis crossings (b) of two pipeline variants for $g = \{1, 2, 3\}$ gaps. Additionally, we show a first-order regression model for each combination.

ity, but at the same time emphasizes the between-cluster connectivity in networks. Nonetheless, interactivity can be used to show intra-axis edges on demand and thus give a more detailed view on the dense parts of the network. Since the aspect ratio of hive plots is fixed, the layout can easily be integrated into a multi-view visual analytics system.

Obviously, there are also some limitations to our approach. The original hive plots [16] are deterministic renderings of networks based on vertex attributes, whereas the current implementation of our approach partially uses non-deterministic algorithms, e.g., for the clustering step, which may lead to different hive plots for the same data or data with small changes. In the clustering step, other approaches that were not considered in our prototype implementation could potentially improve the layout. Even for an optimal axis order the presence of many long edges decreases the readability quickly. The angular resolution for more than 8-12 axes becomes too small to precisely show connectivity details, especially for vertices closer to the origin. See Fig. 14 for an illustration of the limitations. Therefore, our framework has limited visual scalability and is recommended mostly for small to medium graphs with less than 500 vertices and no more than 8–12 clusters. It is possible to hide some visual complexity by collapsing individual axes. However, cloning axes to show intra-cluster connectivity requires understanding two sets of edges that belong to the same vertex, which may decrease the overall readability of our hive plot framework. Lastly, our experimental dataset focused on instances of graphs that exhibit a relatively clear community structure. Hence, we did not investigate how well our insights generalize to other types of graphs. Moreover, we did not investigate how specific graph characteristics beyond the number of edges impact solution quality or runtime behavior and leave this as an open question.

Future Work: In terms of future work, several follow-up questions arise. We considered routing edges according to the shortest cyclic distance between two axes. Potentially, some edge crossings can be resolved by also considering routing edges along the longer distance. We adapted several heuristics and IP models to fit into our framework. For both, heuristics and IP models, questions remain regarding solution quality and computational performance. Furthermore, the gap handling procedure seems to degrade the performance of the iterative heuristic with an increasing number of gaps. Our conjecture is that the overall procedure is stuck in a local minimum, and it could be of interest to thoroughly investigate this behavior. Similarly, we only looked at aggregates of

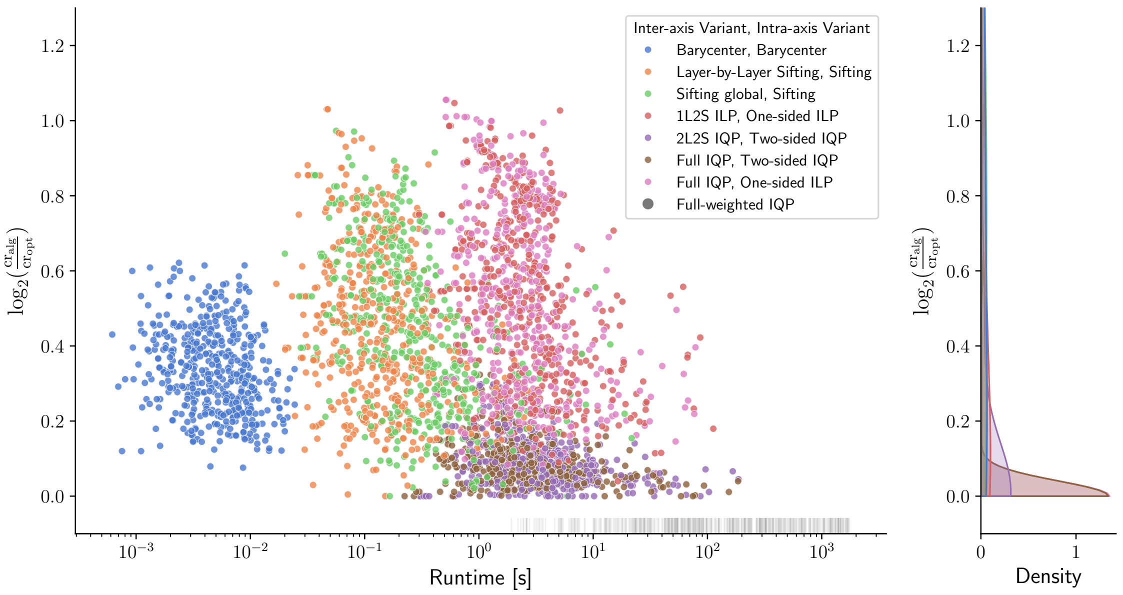


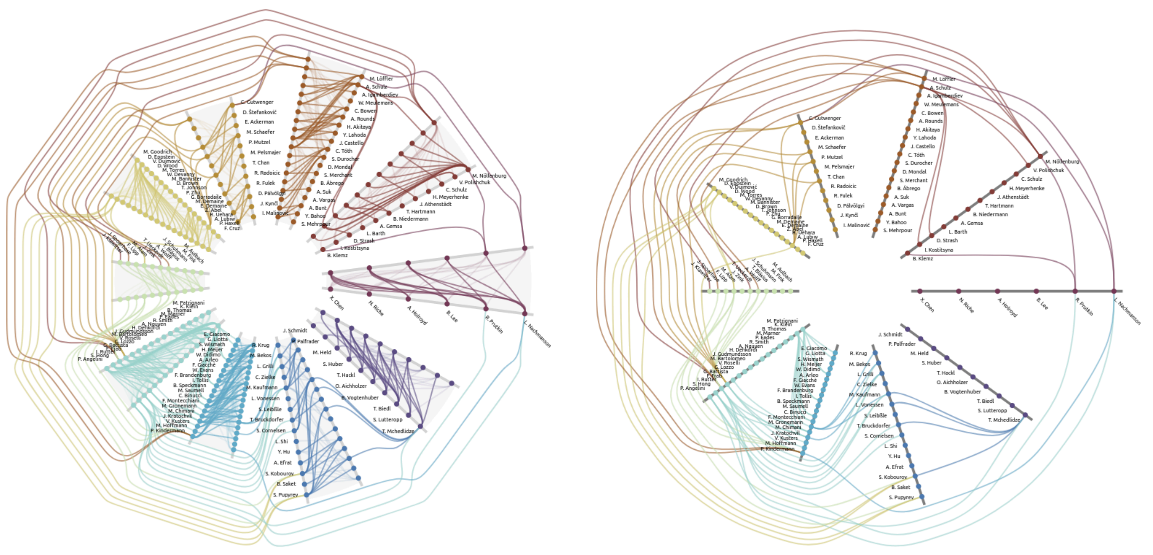
Figure 13: $\log_2$ optimality ratio of intra-axis crossings in regards to the runtime. Only instances are shown where the full-weighted IQP model computed an optimal solution before a timeout occurred. Thus, the best solution of a variant is compared against the optimal solution for each instance. The runtime distribution of the full-weighted IQP model is shown as a rug plot on the bottom. A density plot of the $\log_2$ optimality ratio is shown on the right.

axes and the cyclic length when optimizing order, which does not consider the actual Euclidean length of edges. Potentially, we can improve scalability in the number of axes if we arrange them on an ellipse to increase space between axes. From a human-computer interaction perspective, it would be interesting to see how our hive plot framework compares to other layouts for visualizing small to medium-sized graphs in a formal human-subject study. Similarly, at the moment it is unclear how well users perform typical network visualization tasks on our combinatorial hive plot layouts. Hence, a user study could provide interesting insights. Furthermore, it is also unclear which specific optimization targets benefit users the most and a user study exploring the full design space of hive plots in this regard would be an important investigation. Finally, adding more interactivity and integrating our framework as an alternative view for data exploration into a visual analytics platform could provide additional insights.

References

[1] Christian Bachmaier, Franz-Josef Brandenburg, Wolfgang Brunner, and Ferdinand Hübner. "Global k-Level Crossing Reduction". In: *J. Graph Algorithms Appl.* 15.5 (2011), pp. 631–659. DOI: 10.7155/JGAA.00242.

[2] Christian Bachmaier, Franz-Josef Brandenburg, Wolfgang Brunner, and Gergö Lovász. "Cyclic Leveling of Directed Graphs". In: *Graph Drawing, 16th International Symposium, GD 2008, Heraklion, Crete, Greece, September 21-24, 2008. Revised Papers*. Ed. by Ioannis G. Tollis



(a) collapsed axes (b) expanded axes

Figure 14: A different co-author graph from the citation dataset. The graph has 140 vertices and 397 edges with 47 being proper and 37 being long. Here, simulated annealing was used and the total computation time of the layout was approximately 51ms.

and Maurizio Patrignani. Vol. 5417. Lecture Notes in Computer Science. Springer, 2008, pp. 348–359. DOI: 10.1007/978-3-642-00219-9_34.

[3] Christian Bachmaier, Wolfgang Brunner, and Christof König. “Cyclic Level Planarity Testing and Embedding”. In: *Graph Drawing, 15th International Symposium, GD 2007, Sydney, Australia, September 24-26, 2007. Revised Papers.* Ed. by Seok-Hee Hong, Takao Nishizeki, and Wu Quan. Vol. 4875. Lecture Notes in Computer Science. Springer, 2007, pp. 50–61. DOI: 10.1007/978-3-540-77537-9_8.

[4] Vincent D. Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. “Fast Unfolding of Communities in Large Networks”. In: *Journal of Statistical Mechanics: Theory and Experiment* 2008.10 (2008), P10008. DOI: 10.1088/1742-5468/2008/10/P10008.

[5] Aaron Clauset, M. E. J. Newman, and Cristopher Moore. “Finding community structure in very large networks”. In: *Phys. Rev. E.* 70 (6 2004), p. 066111. DOI: 10.1103/PhysRevE.70.066111.

[6] Fabio Crameri, Grace E. Shephard, and Philip J. Heron. “The misuse of colour in science communication”. In: *Nature communications* 11.1 (2020), p. 5444. DOI: 10.1038/s41467-020-19160-7.

[7] William E. Devanny, Philipp Kindermann, Maarten Löffler, and Ignaz Rutter. “Graph Drawing Contest Report”. In: *Graph Drawing and Network Visualization - 25th International Symposium, GD 2017, Boston, MA, USA, September 25-27, 2017, Revised Selected Papers.* Vol. 10692. Lecture Notes in Computer Science. Springer, 2017, pp. 575–582. DOI: 10.1007/978-3-319-73915-1_44.

[8] Peter Eades and Sue Whitesides. "Drawing Graphs in Two Layers". In: *Theor. Comput. Sci.* 131.2 (1994), pp. 361–374. DOI: 10.1016/0304-3975(94)90179-1.

[9] Michael Forster. "A Fast and Simple Heuristic for Constrained Two-Level Crossing Reduction". In: *Graph Drawing, 12th International Symposium, GD 2004, New York, NY, USA, September 29 - October 2, 2004, Revised Selected Papers.* Vol. 3383. Lecture Notes in Computer Science. Springer, 2004, pp. 206–216. DOI: 10.1007/978-3-540-31843-9_22.

[10] Santo Fortunato. "Community Detection in Graphs". In: *Physics Reports* 486.3 (2010), pp. 75–174. DOI: 10.1016/j.physrep.2009.11.002.

[11] Murali K. Ganapathy and Sachin Lodha. "On Minimum Circular Arrangement". In: *STACS 2004, 21st Annual Symposium on Theoretical Aspects of Computer Science, Montpellier, France, March 25-27, 2004, Proceedings.* Ed. by Volker Diekert and Michel Habib. Vol. 2996. Lecture Notes in Computer Science. Springer, 2004, pp. 394–405. DOI: 10.1007/978-3-540-24749-4_35.

[12] Emden R. Gansner and Yehuda Koren. "Improved Circular Layouts". In: *Graph Drawing, 14th International Symposium, GD 2006, Karlsruhe, Germany, September 18-20, 2006. Revised Papers.* Ed. by Michael Kaufmann and Dorothea Wagner. Vol. 4372. Lecture Notes in Computer Science. Springer, 2006, pp. 386–398. DOI: 10.1007/978-3-540-70904-6_37.

[13] Michael Guarino, Pablo Rivas, and Casimer DeCusatis. "Towards Adversarially Robust DDoS-Attack Classification". In: *2020 11th IEEE Annual Ubiquitous Computing, Electronics Mobile Communication Conference (UEMCON).* 2020, pp. 0285–0291. DOI: 10.1109/UEMCON51285.2020.9298167.

[14] Michael Jünger and Petra Mutzel. "Exact and Heuristic Algorithms for 2-Layer Straightline Crossing Minimization". In: *Graph Drawing, Symposium on Graph Drawing, GD '95, Passau, Germany, September 20-22, 1995, Proceedings.* Ed. by Franz-Josef Brandenburg. Vol. 1027. Lecture Notes in Computer Science. Springer, 1995, pp. 337–348. DOI: 10.1007/BFB0021817.

[15] Michael Jünger and Petra Mutzel. "2-Layer Straightline Crossing Minimization: Performance of Exact and Heuristic Algorithms". In: *J. Graph Algorithms Appl.* 1.1 (1997), pp. 1–25. DOI: 10.7155/JGAA.00001.

[16] Martin Krzywinski, Inanç Birol, Steven J. M. Jones, and Marco A. Marra. "Hive plots - rational approach to visualizing networks". In: *Briefings Bioinform.* 13.5 (2012), pp. 627–644. DOI: 10.1093/BIB/BBR069.

[17] Martin Krzywinski, Ka Ming Nip, Inanc Birol, and Marco Marra. "Differential Hive Plots: Seeing Networks Change". In: *Leonardo* 50.5 (2017), pp. 504–504. DOI: 10.1162/LEON_a_01278.

[18] Vincenzo Liberatore. "Circular arrangements and cyclic broadcast scheduling". In: *J. Algorithms* 51.2 (2004), pp. 185–215. DOI: 10.1016/J.JALGOR.2003.10.003.

[19] Sumio Masuda, Toshinobu Kashiwabara, Kazuo Nakajima, and Toshio Fujisawa. "On the NP-completeness of a computer network layout problem". In: *Proc. IEEE International Symposium on Circuits and Systems.* IEEE Computer Society Press Los Alamitos. 1987, pp. 292–295.

[20] Christian Matuszewski, Robby Schönfeld, and Paul Molitor. "Using Sifting for k-Layer Straightline Crossing Minimization". In: *Graph Drawing, 7th International Symposium, GD'99, Stirín Castle, Czech Republic, September 1999, Proceedings*. Ed. by Jan Kratochvíl. Vol. 1731. Lecture Notes in Computer Science. Springer, 1999, pp. 217–224. DOI: 10.1007/3-540-46648-7_22.

[21] Martin Nöllenburg and Markus Wallinger. "Computing Hive Plots: A Combinatorial Framework". In: *Graph Drawing and Network Visualization (GD'23)*. Ed. by Michael Bekos and Markus Chimani. Vol. 14466. LNCS. Springer, 2023, pp. 153–169. DOI: 10.1007/978-3-031-49275-4_11.

[22] Charles Perin, Romain Vuillemot, and Jean-Daniel Fekete. "SoccerStories: A Kick-off for Visual Soccer Analysis". In: *IEEE Trans. Vis. Comput. Graph.* 19.12 (2013), pp. 2506–2515. DOI: 10.1109/TVCG.2013.192.

[23] Dietmar Pils, Anna Bachmayr-Heyda, Katharina Auer, Martin Svoboda, Veronika Auner, Gudrun Hager, Eva Obermayr, Angelika Reiner, Alexander Reinthaller, Paul Speiser, Ioana Braicu, Jalid Sehouli, Sandrina Lambrechts, Ignace Vergote, Sven Mahner, Astrid Berger, Dan Cacsire Castillo-Tong, and Robert Zeillinger. "Cyclin E1 (CCNE1) as Independent Positive Prognostic Factor in Advanced Stage Serous Ovarian Cancer Patients – A Study of the OVCAD Consortium". In: *European Journal of Cancer* 50.1 (2014), pp. 99–110. DOI: 10.1016/j.ejca.2013.09.011.

[24] Pablo Rivas, Casimer DeCusatis, Matthew Oakley, Alex Antaki, Nicholas Blaskey, Steven LaFalce, and Stephen Stone. "Machine Learning for DDoS Attack Classification Using Hive Plots". In: *2019 IEEE 10th Annual Ubiquitous Computing, Electronics Mobile Communication Conference (UEMCON)*. 2019, pp. 0401–0407. DOI: 10.1109/UEMCON47517.2019.8993021.

[25] Christopher C. Skiscim and Bruce L. Golden. "Optimization by simulated annealing: A preliminary computational study for the TSP". In: *Proceedings of the 15th conference on Winter simulation, WSC 1983, Arlington, VA, USA, December 12-14, 1983*. ACM, 1983, pp. 523–535. DOI: 10.5555/800044.801546.

[26] Kozo Sugiyama, Shojiro Tagawa, and Mitsuhiko Toda. "Methods for Visual Understanding of Hierarchical System Structures". In: *IEEE Transactions on Systems, Man, and Cybernetics* 11.2 (1981), pp. 109–125. DOI: 10.1109/TSMC.1981.4308636.

[27] James M. Varanelli and James P. Cohoon. "A two-stage simulated annealing methodology". In: *5th Great Lakes Symposium on VLSI (GLS-VLSI '95), March 16-18, 1995, The State University of New York at Buffalo, USA*. IEEE Computer Society, 1995, pp. 50–53. DOI: 10.1109/GLSV.1995.516023.

[28] Edward J Wegman. "Hyperdimensional data analysis using parallel coordinates". In: *Journal of the American Statistical Association* 85.411 (1990), pp. 664–675.

[29] Zhao Yang, Juan I. Perotti, and Claudio J. Tessone. "Hierarchical benchmark graphs for testing community detection algorithms". In: *Phys. Rev. E.* 96 (5 Nov. 2017), p. 052311. DOI: 10.1103/PhysRevE.96.052311.

[30] yWorks GmbH. *yEd*. Version 3.20. June 8, 2022.

[31] Lanbo Zheng and Christoph Buchheim. “A New Exact Algorithm for the Two-Sided Crossing Minimization Problem”. In: *Combinatorial Optimization and Applications, First International Conference, COCOA 2007, Xi'an, China, August 14-16, 2007, Proceedings*. Vol. 4616. Lecture Notes in Computer Science. Springer, 2007, pp. 301–310. DOI: 10.1007/978-3-540-73556-4_32.

[32] Johanna Zoppi, Jean-François Guillaume, Michel Neunlist, and Samuel Chaffron. “MiBiOmics: An Interactive Web Application for Multi-Omics Data Exploration and Integration”. In: *BMC Bioinformatics* 22.1 (2021), p. 6. DOI: 10.1186/s12859-020-03921-8.