Computers & Graphics 116 (2023) 448–463

Contents lists available at ScienceDirect

Computers & Graphics

journal homepage: www.elsevier.com/locate/cag

Technical Section

Improving readability of static, straight-line graph drawings: A first look at edge crossing resolution through iterative vertex splitting☆

Henry Ehlers [a,*], Anaïs Villedieu [b], Renata G. Raidou [a], Hsiang-Yun Wu [c]

[a] TU Wien, Institute of Visual Computing and Human-Centered Technology, Favoritenstr. 9-11/E193-02, Stiege 2, 5th Floor, Vienna, A-1040, Austria
[b] TU Wien, Institute of Logic and Computation, Favoritenstr. 9-11/E192-01, Stiege 2, 4th Floor, Vienna, A-1040, Austria
[c] St. Pölten University of Applied Sciences, Department of Media and Digital Technologies, Campus-Platz 1, St. Pölten, A-3100, Austria

ARTICLE INFO

Article history:
Received 8 May 2023
Received in revised form 15 September 2023
Accepted 18 September 2023
Available online 22 September 2023

Keywords:
Network visualization
Vertex splitting
Edge crossings
Graph aesthetics

ABSTRACT

We present a novel vertex-splitting approach to iteratively resolve edge crossings in order to improve the readability of graph drawings. Dense graphs, even when small in size (10 to 15 nodes in size) quickly become difficult to read with increasing numbers of edges, and form so-called "hairballs". The readability of a graph drawing is measured using many different quantitative aesthetic metrics. One such metric of particular importance is the number of edge crossings. Classical approaches to improving readability, such as the minimization of the number of edge crossings, focus on providing overviews of the input graph by aggregating or sampling vertices and/or edges. However, this simplification of the graph drawing does not allow for detailed views into the data, as not all vertices or edges are rendered, and also requires sophisticated interaction approaches to perform well. To avoid this, our locally optimal vertex splitting approach aims to minimize the number of remaining edge crossings while also minimizing the number of vertices that need to be split. In each iteration, we identify the vertex contributing the largest number of edge crossings, remove it, locate the embedding locations of said vertex's two split copies, and determine each copy's unique adjacency. We conduct a user study with 52 participants to evaluate whether vertex splitting affects users' abilities to conduct a set of graph analytical tasks on graphs 12 nodes in size. Users were tasked with identifying a vertex's adjacency, determining the shared neighbors of two vertices, and checking the validity of a set of paths. We ultimately conclude that within the context of small, dense graphs, systematic vertex splitting is preferred by participants and even positively impacts user performance, though at the cost of the time taken per task.

1. Introduction

From social media to multi-omics interaction analyses, modern network visualization deals with increasingly large and dense (sub-)networks [1]. Visual analysis efforts of such networks focus on understanding not only the importance of individual entities, such as genes or persons but also their in-between relationships, such as biochemical interaction types or social relationship statuses. These networks are most commonly visualized as straight-line node-link diagrams, which illustrate entities as points of potentially different shapes or colors, and edges as straight lines connecting them (Fig. 1(a)). If not drawn by hand, such graphs are commonly laid out using force-directed layout algorithms, owing to their implementations' availability and computational tractability, such as (extensions of) Eades [2], Fruchterman–Reingold [3], or Kamada–Kawai [4]. These techniques optimize the graph's drawing in terms of physically modeled stress or cost functions. While some research effort has been made to incorporate one [5,6] or multiple [7,8] graph aesthetic metrics into force-directed algorithms, the majority do not. Subsequently, most readily available force-directed algorithm implementations do not scale well visually to dense and/or larger graphs, producing hard-to-read, or even completely unintelligible, drawings [9]; so-called "hairballs" [10,11]. While such drawings may indeed effectively communicate the complexity of the network in question, they fail to allow users to meaningfully understand the network's structure or any of its potential features of interest [12]. To handle "hairball" effects in larger graphs, existing work has focused largely on interactive graph summarization [13], i.e. presenting graphs at different levels of detail. By introducing such level-of-detail hierarchies, summarization unavoidably manipulates graph topology, thereby altering the perceived relationships within the graph and potentially confusing the mental map created by the user. For completeness' sake,

☆ This article was recommended for publication by T. Isenberg.
* Corresponding author.
E-mail address: henry.ehlers@tuwien.ac.at (H. Ehlers).

https://doi.org/10.1016/j.cag.2023.09.010

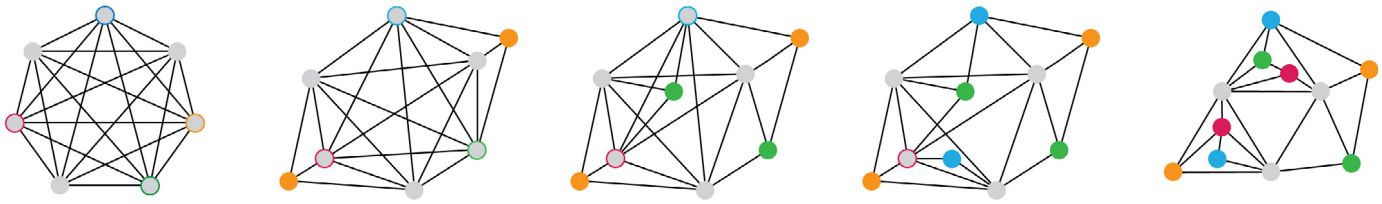


Fig. 1. A $k = 7$ complete graph, initially laid out using Kamada–Kawai (a), is iteratively split by hand, one vertex at a time, to resolve edge crossings (b–d) until all are resolved (e). To do so, four nodes are split. Before being split, these nodes are visualized as gray circles with a colored border around them. After splitting, they are shown as a correspondingly colored circle. It should be noted that, while the initial embedding (a) may be difficult to read, so is the final embedding (e). In (e) specifically, while no edge crossings remain, the number of split vertices complicates the reading of the graph's and individual node's topologies.

it should be mentioned that recent efforts have been made to create such level-of-detail hierarchies *without* altering the graph's topology [14].

1.1. The relevance of small, dense graphs

While graph data have become larger, the analysis and visualization of small, dense graphs remain important challenges, as domain experts are not only interested in a graph's topology on a global but on a local level. Hence, many interactive techniques and tools, in line with Shneiderman's "*Overview first, details on demand*" mantra [15], aim to provide a summarized view of a network's topology from which a subgraph of interest can then be investigated in more detail. For example, the Kyoto Encyclopedia of Genes and Genomes (*KEGG*), currently has over 40, 000, 000 genes recorded [16–18]. Research has so far not attempted to understand or navigate an organism's complete network of genes, metabolites, and chemicals. Instead, work focuses largely on visualizing and analyzing subgraphs, i.e. pathways, or collections of such subgraphs [19,20]. Outside of the context of *KEGG* pathways, similar *Focus+Context* visualizations of small subgraphs can be found in, for example, general systems biological network [21,22] or social network visualization [23], just to name a few. Thus, as visualization and visual analysis of small, dense graphs remain relevant to domain experts, so does the search for aesthetically optimized embeddings of such graphs.

1.2. Existing methods to edge crossings resolution

Within the context of smaller graphs ($10 \leq n \leq 15$), Purchase et al. [24–26] extensively studied the importance of edge crossings on user preference and performance. These findings were corroborated more recently by Kouburov et al. [27], who found that larger numbers of edge crossings indeed (statistically significantly) negatively impact the accuracy and efficiency of user performance in smaller graphs ($n = 40$). Unfortunately, finding some optimal embedding that produces the minimum (edge) crossing number k [28] is, even for highly restricted graph classes [29], NP-complete [27,30]. Thus, beyond "simply" determining whether a planar k-crossing graph exists or what a graph's minimal crossing number k may be, extensive work has focused on finding means of eliminating edge-crossings from non-planar graphs [31,32]. One such example involves computing the minimal sets of edges [33] or vertices [34,35] that must be deleted to resolve all edge crossings. While this elimination of edge crossing through the removal of some minimal set of edges or vertices may be an interesting solution from a graph theoretic perspective, it is not a useful paradigm for domain experts. In many biological and biochemical application domains, understanding the relationships between, as well as the attributes of, *all* entities of interest is crucial, making the outright removal of entities or their relationships not practical.

1.3. Approach and contribution

In this paper, we propose an approach to reduce graph complexity to improve the readability of graphs *without removing relationship or entity information* that is *agnostic of the graph drawing method chosen*. Our approach is centered on *splitting* "problematic" vertices, i.e., vertices whose edges are involved in large numbers of edge crossings, in a graph's drawing. As noted by Henry et al. [36], a set of terms have been used interchangeably to describe such approaches, such as *duplicating*, *cloning*, *aliasing*, or *mirroring*. Here, we keep to the term *vertex splitting*, common to the graph drawing community. Vertex splitting involves identifying a vertex v, removing it from the drawing, and replacing it with two non-adjacent copies of v_1 and v_2, while the original edges of vertex v are distributed across the two split copies [37]. As shown in Fig. 1(a) and Fig. 1(b), the orange-circled vertex in (a) is selected, split in two, and each copy now visualized as an orange-colored vertex (b). Importantly, the adjacency of the original vertex is now shared across both copies.

Given some initial graph embedding, this splitting could be achieved in a principled manner to optimize graph aesthetic criteria [38]. As discussed previously, while many different aesthetic criteria influence a graph's readability [26], overwhelmingly, the *number of edge crossings* have been shown to negatively affect user performance and preference. This is especially true for smaller graphs, ranging in size from 10 to 20 vertices, [24,25,39]. However, vertex splitting approaches featured in literature are not directly driven by graph aesthetic metrics, but an *a-priori* selection of "problematic" vertices [40,41], or the simple *complete splitting of selected vertices* based on groups/clusters featured within the data [36,42,43]. To the best of our knowledge, there exists no principled approach to vertex splitting which aims to improve graph aesthetic criteria.

The *contribution* of our work is the development, assessment, and discussion of a novel, iterative approach to vertex splitting for static, straight-line graph drawings. Specifically, our approach (i) minimizes the total number of edge crossings in the embedding, given a selection of embedding faces which (ii) maximizes the number of edge-crossing-free connections, while (iii) splitting the minimal number of vertices necessary. We achieve this by discretizing the graph drawing into edge-crossing-equivalent polygons, i.e. polygons within which all points induce the same number of edge-crossings when connected to any vertex in some graph drawing , thereby simplifying the search for a split vertex copy's placement within this embedding. To investigate the effect of our vertex splitting on the readability of small graphs —here 12 nodes in size— for which the importance of edge crossings has been thoroughly documented [24,25,39], we conduct a user study with 52 participants. The outcomes indicate that our approach has the potential to improve readability and aesthetic quality in static, straight-line graphs.

2. Related work

Many aesthetic criteria affect a user's ability to effectively navigate and analyze a graph drawing [26]. A criterion of particular importance is the number of edge crossings, which has been found to especially affect readability in small- and intermediate-sized graphs [25]. In addition to drawing criteria, the size, density, and topological structure of a graph will also influence the layout quality [44]. In this section, we summarize previous works related to our proposed approach.

Graph Drawing and Graph Aesthetics— Heuristic-based graph drawing, such as force or energy-based approaches, often produce difficult-to-read embeddings with poor graph aesthetic criteria, as they optimize these criteria only indirectly [27]. However, work has also been done on rendering more readable graphs by drawing them directly as a function of these aesthetic metrics. As these graph aesthetics often conflict with each other [45], most such approaches aim at satisfying either a single or a *balance* of multiple such criteria [46]. Indeed, the inclusion of one [5,6,46,47] or multiple [7,8,48] such aesthetic criteria for classical graph drawing approaches have been discussed for a while. Additionally, multi-objective neural-network-based approaches to graph embedding have also found popularity [45,49]. However, owing to their complexity and novelty, most approaches are simply not readily available.

Reducing Graph Complexity for Visualization—As the graph density increases, theoretical graph drawing techniques can only handle limited graph classes [50], which still cannot cover all types of real-world graphs. To solve the hairball problem for practical analysis purposes, several approaches have been developed. One common strategy called *summarization* aims to abstract a graph through vertex or subgraph aggregation, as summarized in the survey by Liu et al. [13]. This strategy includes *vertex aggregation*, where clustering that produces data hierarchy is often performed for interactive data navigation [51]. Another technique, *edge aggregation*, such as edge compression [52], edge concentration [53], and edge bundling [54,55] manipulate edge topology and geometry for better visual quality. The two ideas can be also adapted simultaneously. Dwyer et al. optimized power graphs to achieve lossless vertex and edge aggregation, which facilitates to follow paths in the graphs [56]. However, the problem has been proven to be NP-hard and no scalable technique is readily available. *Simplification* is another type of summarization, which deletes edges [33] or vertices [34,35]. In addition to deleting vertices, OntoVis [57] simplifies graph topology by also removing duplicated paths. More recently, *graph sampling* has been proposed to approximate large graphs for visualization purposes [58,59]. The aforementioned approaches either remove data or require interaction [44], which is sometimes not preferred for specific user tasks. Other techniques that use spanning trees and graphs as backbones [60,61] provided a partial solution—yet, the complexity of edge rendering persists. *Toroidal wrappings* allow for vertices, and thereby their edges, to be spread across a (2D) torus topology. This allows for links to wrap across the rectangular view boundary, thereby reducing the number of edge crossings had the network been embedded on a 2D surface [62]. Such embeddings not only improved the network's aesthetics metrics but also were shown to assist in user understanding [63]. However, this comes at the cost of both time taken and accuracy, especially for path tracing tasks [62,63].

Vertex Splitting in Applications—Vertex splitting is not a brand new idea: several overview maps of pathway databases (ReconMap [64] and KEGG map [16]) incorporate vertex splitting to reduce the visual complexity of pathway diagrams. However, these diagrams are often designed and curated *manually* by illustrators. Due to this common usage of vertex splitting, the idea has been investigated in several applied papers. For example, to produce a fully planar drawing, the ontology tool OntoRama [40] and the gene ontology tool TreePlus [41] allowed for vertices to be cloned. Cross-linked vertices present in multiple branches are simply fully duplicated to produce a more readable, tree-style layout. Beyond tree graphs, Wu et al. [43] incorporate different duplication schemes based on vertex properties and extend the idea on multi-level clustered graphs [65]. Lastly, Nielsen et al. [66] developed a learning-based approach to determine which vertices should be duplicated based on human-curated graph drawings featuring (iteratively) split vertices. However, these approaches do necessarily tackle the challenge of readability. Complete splitting [40,41] may produce too many copies of the same vertex, thereby hindering readability in and of itself; as discussed by Henry et al. [36]. On the other hand, splitting a vertex once per group assignment [43] does not necessarily ensure the readability of each group individually. Such split vertices may still be responsible for large numbers of edge crossings or be overly connected to other vertices within the group. Splitting these vertices again could further improve graph aesthetic criteria. Nonetheless, these approaches are strongly bound with specific applications and cannot be easily extended to more generic cases.

Vertex Splitting in Graph Drawing and Visualization—Beyond the examples learned from application domains, previous research tackled also more general solutions. The spring-based layout algorithm of Eades and Mendoça [67] aims to split vertices based on the "tension" of adjacent edges, to relax edge springs in the embedding. They demonstrate that even their heuristic approach allows for the resolution of edge crossings through vertex splitting. Henry et al. investigated NodeTrix [36], a hybrid visualization integrating node-link diagrams and matrix representations. This work researches the users' abilities on analytical tasks pertaining to relationships, and our project is primarily inspired by the positive findings of these two publications. Additionally, Lambert et al. [68] and Rohrschneider et al. [69] propose approaches to duplicating vertices within and across metabolic pathways. Additionally, vertex duplication has not been only used for graph visualization but is also applied to improve the legibility of Euler diagrams [42]. Lastly, Nöllenburg et al. [37] study the vertex splitting problem from a theoretical perspective; given some input drawing, they find that determining the smallest set of splits to produce a planar drawing, is NP-complete.

In summary, to the best of our knowledge, the effectiveness of vertex splitting has not been yet fully investigated in common visualization scenarios, although it has been used in some practical applications. In this paper, we aim to develop and propose a vertex-splitting approach to iteratively resolve edge crossings with the purpose of improving the readability of graph drawings. Our approach can be applied to network visual analytics by assisting users to navigate and read graphs, while also retaining local algorithmic guarantees.

3. Desiderata

Many different quantitative aesthetic metrics can negatively impact the readability of a graph drawing [26]. As we are interested in improving these metrics in a given embedding, any one, or multiple, metric(s) could form the basis of our vertex-splitting approach. However, as the edge crossing criterion [29] forms arguably the most important metric, especially for smaller graphs, we opt to focus on resolving edge crossings [37] specifically. Moreover, Eades and Mendoça's heuristic approach to tension reduction showed that splitting can indeed reduce the number of edge crossings [70]. However, as highlighted by Henry et al. [36]

splitting a vertex too often or splitting too many vertices, can itself have an undesirable impact on readability (Fig. 1(e)). Ultimately, we wish to resolve as many edge crossings as possible, while splitting as few vertices as necessary. To describe our goals more concretely, we define three desiderata which our approach aims to locally guarantee, namely:

(D1) Minimize the Number of Vertices Split: As discussed by Henry et al. [36,71], too many split vertices, or a vertex split too often, can negatively impact a user's ability to parse a graph's drawing. Thus, we aim to resolve as many edge crossings as possible, while locally minimizing the number of vertices to split. That is to say, we only ever split a vertex in two, i.e. the smallest split possible. Looking at Fig. 2 (D1), the orange embedding shows how edge crossings can be resolved by simply splitting a target vertex completely, i.e. splitting it once for every one of its edges. Such approaches result in a set of split copies of degree $d(v) = 1$ and add unnecessary visual clutter. Instead, we thus opt to only ever split a vertex in two, as seen in red, to ensure that splits minimally affect readability.

(D2) Maximize the number of edge-crossing-free connections: Embedding a vertex in a graph drawing which minimizes the overall number of edge crossings can be done in polynomial time. However, currently available methods are not practical for real-world applications owing to the amount of time needed [72]. Thus, we first employ a straightforward heuristic for the subsequent, and more general, desideratum **D3**: identifying an embedding location that minimizes the vertex's number of edge-crossing-free edges to its neighbors. In essence, we must first identify the *face* which is incident to a maximal number of neighbors of our vertex. For example, in Fig. 2 (D2) the red embedding is chosen over any of the orange ones, as the former allows for 3 edge-crossing-free connections, compared to any of the latter's 2.

(D3) Minimize the Drawing's Total Edge Crossings: While **D2** allows us to identify a face f within which to embed a vertex v, the vertices incident to f will most likely not encompass all of v's neighbors. Connecting to the remaining neighbors of v not incident to f will thus induce edge crossings, which will negatively impact readability [24,25,27,39]. Thus, within f, we must identify where to place this vertex, such that the number of induced edge crossings is minimized. In Fig. 2 (D3), the red embedding is selected, as it only indicates 1 edge crossings, compared to the 2 induced by the orange drawing.

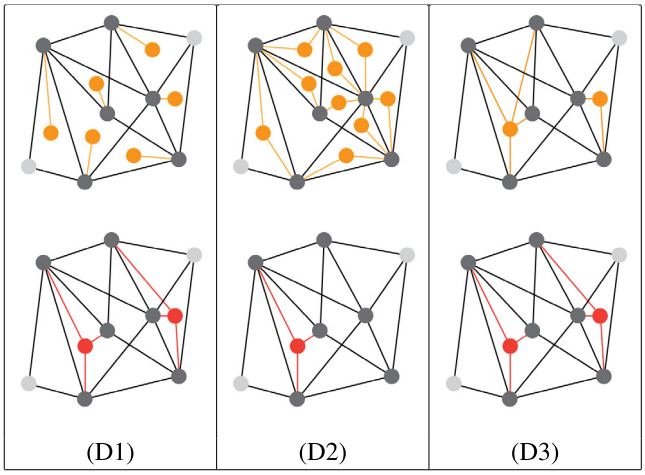


Fig. 2. A visual example of the approaches three desiderata. (D1) visualized the *minimization of split vertices* desideratum. Instead of resolving all edge crossings possible, by simply splitting a vertex completely (as shown in orange), we instead only ever split a vertex in two, as this is the smallest split possible. (D2) showcases the *maximization of edge crossing free connections* desideratum. Assume we aim to insert a single vertex into an existing embedding, adjacent to a set of given set of neighbors (shown in dark gray). Instead of embedding said vertex in any of the orange positions, which all only allow for 2 edge-crossing-free connections, we embed this vertex as shown in red, where 3 adjacent vertices can be reached edge-crossing-free. Lastly, (D3) illustrates the *minimization of edge crossings* desideratum. Given two split copies of a vertex embedded within two faces that fulfill D1, we now must find the adjacencies of each copy that minimize the total number of induced edge crossings. Here, the orange set of vertices and edges is discarded, in favor of the red one, as the former induces 2 edge crossings, compared to the latter's 1.

4. Vertex splitting algorithm

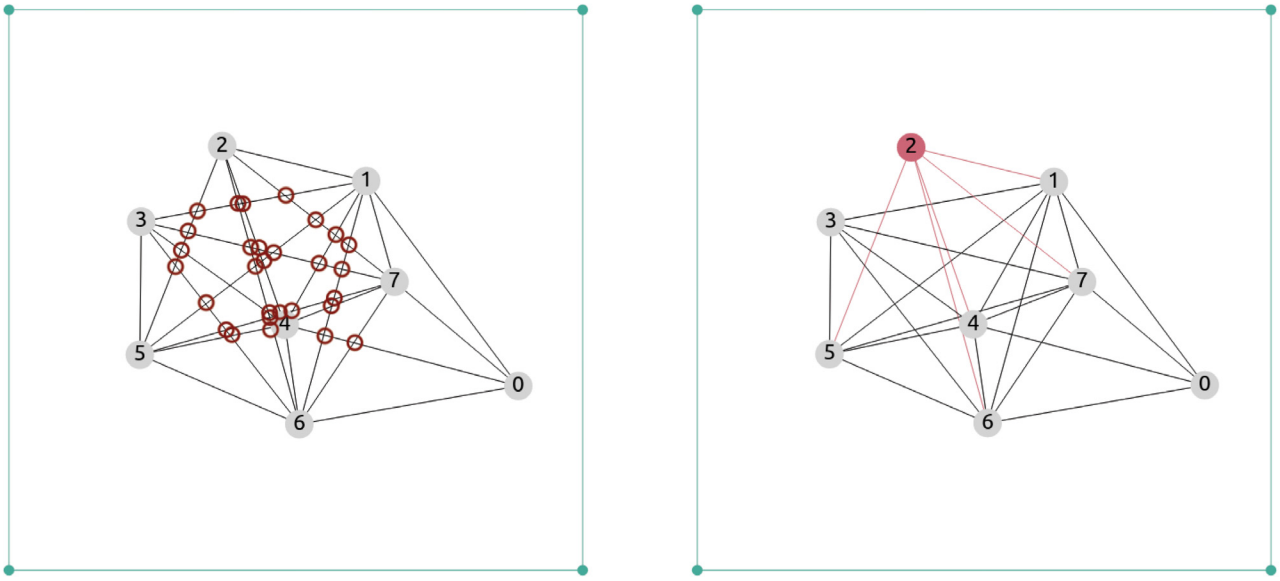
Here, we first provide a high-level description of our algorithm, visualized in Figs. 3–6 (a). Let $G = (V, E)$ be our input graph where $V = \{v_1, v_2, \ldots, v_n\}$ and $E = \{e_1, e_2, \ldots, e_m\}$. We additionally consider a two-dimensional, straight-line drawing D of G as input. Our algorithm functions in three steps:

Step 1: Target Vertex Selection First, the target vertex to be split must be identified. To do so, all edge crossings in the given embedding are counted, each mapped to the edges' incident vertices (Fig. 3(a)). The vertex whose incident edges are involved in the largest number of edge crossings is selected for splitting (Fig. 3(b)).

Step 2: Face Decomposition and Selection After removing the selected target vertex from the embedding, in line with desideratum **D1**, we select only a pair of faces to embed each of the two copies of the removed vertex. Moreover, we must also select a subset of the copies' neighborhoods, namely the neighbors who will not induce any crossings in the input drawing Fig. 4(b), as discussed in desideratum **D2**. Note that we only split a vertex into two at each iterative step. To do so, we start by computing a planarization of D to identify a set of faces $F = \{f_1, f_2, \ldots, f_k\}$ (Fig. 4(a)) that partition the canvas disjointly. When placing a vertex v in a non-convex face f, there is no guarantee that we will be able to draw crossing-free edges between v and the vertices incident to f. Thus, we compute a set of *sight cells* $S = \{s_1, s_2, \ldots, s_r\}$, which are convex polygons that partition non-convex faces (Fig. 4(b)). Note that an already convex face is its own sight cell.

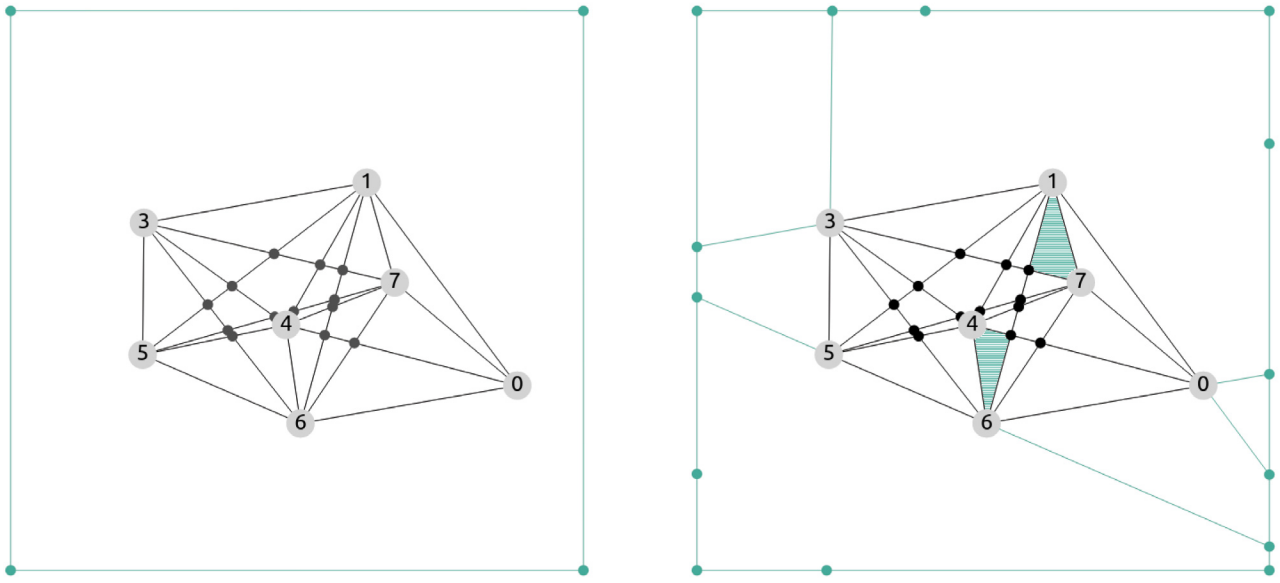
Step 3: Subface Decomposition and Selection Given two selected sight cells, we must now decide where within them to place each split copy of the selected vertex, such that the number of induced edge crossings is minimized. To achieve this, sight cells are further decomposed into *subfaces*; convex polygons that each form an equivalence class in regards to crossings (Fig. 5(a)). More precisely, for a region $b \in B$, any embedding of a vertex v within b will induce the same amount of crossings. Instead of infinitely many possible embedding locations with a selected sight cell, we can consider only a discrete set of subfaces within it. Now, in line with desideratum **D3**, we can select a region for each of the selected sight cells such that the smallest number of additional edge crossings are induced when connecting them to vertices non-incident to the selected (sub-)face (Fig. 5(b)).

Step 4: Embedding of the Split Vertex Copies With two subfaces selected, and the edge-crossing-minimizing adjacency of each subface determined, the final step is the simple embedding of the two split vertices and the drawing of the edges between their neighbors. Again, only two split copies are created to adhere to desideratum **D1** (Fig. 6(a)). Similar to Riche and Dwyer [42], in the interest of preserving the user's *Gestalt-theoretic* [73] mental map of the original graph embedding (Fig. 6 (b)) as much as possible, the graph is not redrawn using spring embedding.



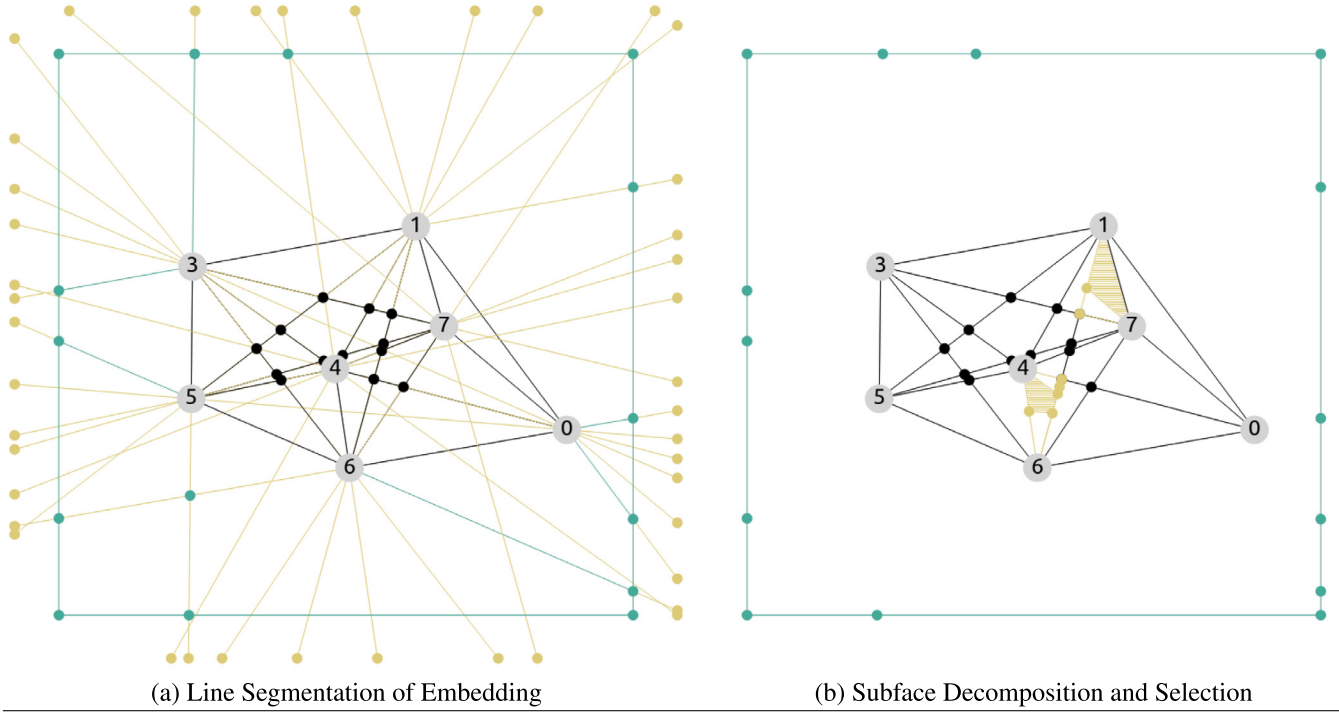
(a) Count Edge Crossings (b) Select Target Vertex

Fig. 3. *Step 1: Target Vertex Selection.* In order to ultimately detect the vertex to split, we first count all edge crossings present in the provided graph embedding (a), where edge crossings are circled in red. For each edge crossing, the two involved edges' incident sets of vertices have their crossing numbers incremented. In order to resolve as many edge crossings as possible, the vertex with the highest crossing number is selected as the target to be split (b), here vertex 2. The canvas border is shown in blue ( ).



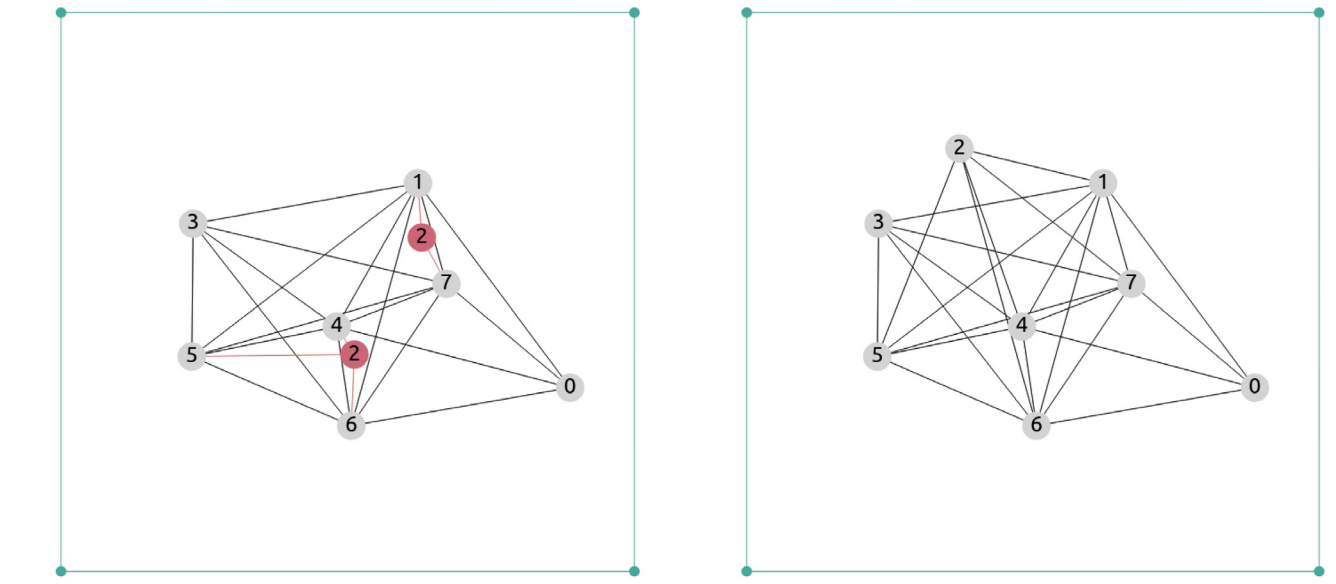
(a) Target Vertex Removal and Face Detection (b) Sight Cell Decomposition and Selection

Fig. 4. *Step 2: Face Decomposition and Selection.* The select target vertex, as well as its incident edges, are removed from the embedding, and the remaining graph is planarized (a) using Planarization vertices, illustrated as smaller, unlabeled, dark gray circles ( ). All faces in the embedding can now be identified. Given desideratum *D2*, we aim to maximize the number of edge-crossing-free connections when selecting the embedding faces. However, as not all inner faces (as well as the outer face) are convex, they must be decomposed into convex sight-cells polygons (b). This is achieved through line projection and the placement of tiling vertices and tiling edges within and until the canvas border, both shown in blue ( ). The selected sight cells are indicated as turquoise-color surfaces. As we are trying to achieve this with minimal numbers of split vertices (D1), only two faces, i.e. splits, are selected.



(a) Line Segmentation of Embedding (b) Subface Decomposition and Selection

Fig. 5. *Step 3: Subface Decomposition and Selection*. In line with desideratum D2, given a minimization of the number of edge-crossing-free connections, we aim to minimize the total number of edge crossings. To identify regions within the embedding that are equivalent in terms of their number of induced edge crossings, a complete line arrangement drawing of all pairs of real vertices is drawn bounded by user-defined embedding limits (a). As we are interested only in placing vertices within the two selected target faces (Fig. 4), we use these line segments to tile only them into sight cells in order to identify the pair of sight cells which induce the fewest number of edge crossings possible when connected to the original target vertex's neighbors (b). The newly placed tiling vertices and tiling edges are drawn in brown ( ), as are the highlighted selected subfaces. The canvas border is shown in blue ( ).



(a) Embed Split Vertices (b) Original Unsplit Graph

Fig. 6. *Step 4: Embedding of the Split Vertex Copies*. We now place the two split copies at the centroids of the selected sight cells and connect them to all neighbors vertices incident to their selected face (Fig. 3), as well as all other neighbors in accordance with their selected sight cell (Fig. 4(a)). Note the "suboptimal" placement cased by separating out desiderata **D2** and **D3**; placing it to the left of the edge connecting vertices 4 and 6 would have resolved an additional edge crossing. The original, unsplit graph is presented for the sake of comparison (b). The canvas border is shown in blue ( ).

4.1. Step 1: Target vertex selection

The first step of the algorithm involves the identification of the vertex to be split, based on the number of edge crossings in the embedding (Fig. 3(a)) The vertex involved in the largest number of edge crossings must be identified (Fig. 3(b)). By selecting the vertex with the largest number of edge crossings, we aim to thereby also resolve the largest number of edge crossings associated with other vertices, in line with desideratum **D3**. Specifically, for each $e \in E$, let $cr(e)$ denote the number of edges that cross e. For $v \in V$, let $cr_n(v)$ denote the sum of $cr(e_v)$ where $e_v \subseteq E$ are edges incident to v, i.e. $cr_n(v) = \sum cr(e_v)$. All vertices are then ranked by their number of edge crossings, i.e., $cr_n(v)$. The vertex with the largest crossing number is selected as the target vertex to be split, its adjacency is recorded and subsequently removed from the drawing D. Ties in vertices' edge crossings numbers are resolved based on their degree. To illustrate, consider the example graph $G(V, E)$ of size $|V| = 8$ shown in Fig. 3; (a) highlights all edge crossings which ultimately leads to the selection of vertex 2, as its crossing number, $cr(v_2) = 18$, is the largest.

4.2. Step 2: Face decomposition and selection

Desideratum **D2** states that we aim to not only minimize the number of edge crossings in the embedding but also maximize the number of edge crossing free connections. Thus, when selecting a potential embedding location for a copy, we first identify a face whose incident vertices are part of the split vertex's neighborhood. The aim of the second step of our algorithm is to select two convex regions that jointly maximize the number of edge-crossing-free-connections to the split vertex's neighborhood. First, we consider D_P, a planarization of D, where each line crossing induces a vertex in the graph. We obtain it by placing a vertex called a *planarization vertex* on each crossing and replacing the crossed edges by paths through that new vertex (Fig. 4(a)). Since D_P is a planar graph we can find its faces: $F = \{f_1, f_2, \ldots, f_k\}$. However, as there is no guarantee that the faces obtained after the planarization are all convex, we must further decompose the faces into convex polygons to ensure incident vertices can indeed be reached edge-crossing-free (Fig. 4(b)). Let $F_C \subseteq F$ denote the set of faces which are convex, and $F_{NC} \subseteq F$ the set of faces which are not convex, such that $F_C \cap F_{NC} = \emptyset$ and $F_C \cup F_{NC} = F$. Returning to the previous example, Fig. 4(a) showcases the planarization of the original graph with the selected target vertex removed, and (b) illustrates both the decomposition of the outer face into sight cells and the selection of two subfaces which jointly maximize the number of edge-crossing-free connections.

4.2.1. Face detection and selection

To compute F, we adapt the graph minimum cycle basis algorithm by Kavitha et al. [74]. For a Graph G with non-negative edge weights, they define a *cycle basis* as a maximal set of linearly independent cycles. Correspondingly, the *minimum cycle basis* is the cycle basis whose sum of edge weights is minimal. However, the minimum cycle basis only extracts cycles topologically and does not necessarily correspond to a graph's set of faces geometrically. Thus, two additional steps have been incorporated to identify drawing D_P's faces (Fig. 7).

First, we must ensure that not just all vertices, but *all edges are traversed* as well. To do so, we place a *cycle vertex* (shown as • in Fig. 7(a)) at the center of each edge in D_P, and replace this edge with a path through the newly placed vertex. Second, we must ensure that *all identified cycles are indeed faces*, i.e., that each cycle does not contain any vertices or edges within it. If an identified cycle does not contain any vertices within itself, it is deemed a valid face. Otherwise, its subgraph, defined by the cycle in question as well as all vertices and edges within it, need to be extracted as shown in Fig. 7(b). From this subgraph, edges are removed that can be mapped to two valid faces, or if it is part of the cycle in question and already mapped to one valid face. The minimum cycle basis is now again identified within this subgraph, and the face-check is recursively repeated.

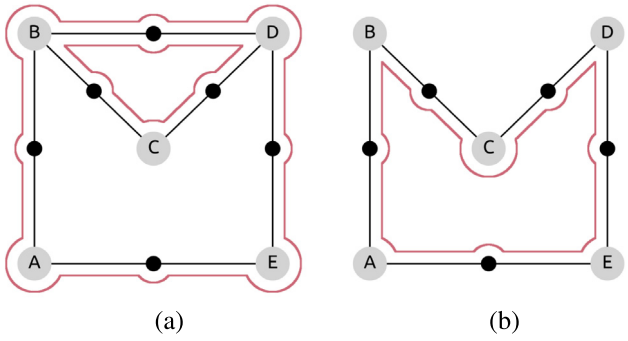


Fig. 7. Given in (a) is a graph with vertices $\{A, B, C, D, E\}$. *Cycle vertices*, shown as smaller, unlabeled circles, are placed at the center of all edges. The minimum cycle basis is identified; here (A, E, D, B, A) and (C, D, B). Cycle (C, D, B) is a valid face. Cycle (A, E, D, B, A) is not, as it contains vertex C. Thus, the cycles' subgraph is extracted. Edge E_{BD} mapped to one valid face (C, D, B) and is also part of the graph's outer face, so it is removed from the subgraph. The minimum cycle basis is again identified for the pruned subgraph (b). The only identified cycle, (A, E, D, C, B, A), is now also a valid face.

Lastly, to fulfill desiderata **D1** and **D2**, we need to select the *two* faces whose set of incident vertices, which are part of the target vertex's adjacency, is the largest. Ties are resolved using each edge's absolute Euclidean difference from its Kamda–Kawai's graph-theoretic optimal length. Fig. 4(b) showcases the selection of target embedding faces. The selected target vertex 2 's neighbors were vertices v_1, v_4, v_5, v_6, v_7. By selecting the blue highlighted faces, namely { 1 , 7 , •} and { 4 , 6 , •}, 4 or the 5 target neighbors can be reached edge-crossing free.

4.2.2. Decomposing non-convex faces

In accordance with desideratum **D2**, we aim to connect our copies to a face's incident vertices without inducing edge crossings. For convex faces, this can be straightforwardly achieved. For non-convex-shaped faces, however, this is not necessarily possible. Here, to ensure that a non-convex face's incidence can be realized as edge-crossing-free connections, the connections must be decomposed into convex polygons, which we call *sight cells*. We denote our sight cells as $S = \{s_1, s_2, \ldots, s_r\}$. For a non-convex face $f_k \in F_{NC}$, let its sight cells be denoted as $S_k \subseteq S$. These sight cells are created by placing additional *tiling vertices* V_t and *tiling edges* E_t to D_P (Fig. 8). Note that a convex face is its own sight cell.

First, given a non-convex face $f \in F_{NC}$, we identify at least one vertex v incident for f whose incident edges form an inner angle greater than 180°. For each edge pair e_{uv} and e_{vw}, two rays are projected from u and w respectively, through v (thus, inside f). We compute the intersection of each ray and the boundary of f. At the location of the intersection, a new tiling vertex $t \in V_T$ is placed, and the edge intersected is replaced by a path through t. An additional tiling edge connecting vertex v and t is added. For example, in Fig. 8, a hypothetical, non-convex face $\{A, B, C, D\}$ is decomposed along the rays projected from edges e_{AB} and e_{CB} as they form an angle greater than 180° around vertex B , resulting in sight cells (B, C, X, B), (A, B, Y, A), and (B, X, D, Y, B). These newly placed edges define the bounds of a vertex's "line of sight", i.e., a vertex can be connected edge-crossing-free from one side of this edge, but not the other. This is repeated for the

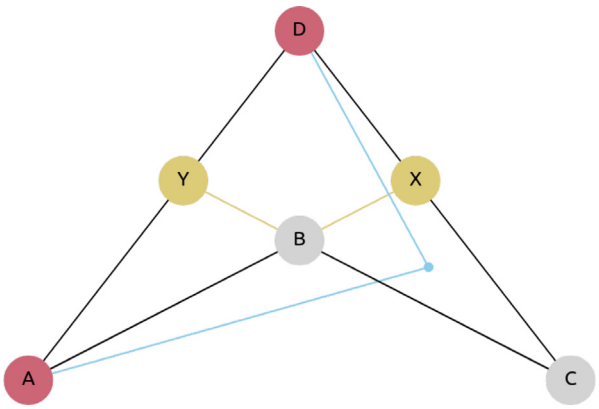


Fig. 8. Illustration of the decomposition of some non-convex inner face (A, B, C, D, A). Incident vertices A and D are adjacent to the target vertex to be split. Here, the angle formed by the edges incident to vertex B, (E_{AB}) and (E_{BC}) forms angle greater than 180°. Ray (AB) is projected until it intersects the closest edge, E_{CD}, where a new tiling vertex X is placed. The intersected edge, E_{CD}, is replaced by paths through that new vertex, i.e. E_{CX} and E_{XD}. A tiling edge is created between the newly placed tiling vertex X and vertex B. This process is repeated for ray (CB), creating *tiling vertex* Y. The identified sight cells are, (A, B, Y, A), (Y, B, X, D, Y), and (B, C, X, B). To illustrate the calculation of sight cell incidence, consider sight cells (B, C, X, B). Lines, in blue, are drawn between the sight cells' centroid ( ) and the two vertices adjacent to the split target. Vertex D can be reached without crossing any edges. Vertex A, however, cannot, as the project line crosses the edge E_{BC}. Thus, its incidence is only $\{A\}$. The incidence of the two remaining sight cells is $\{A, B\}$.

second edge incident to v, as well as all other vertices incident to f whose incident edges form angles greater than 180°. The subgraph defined by the vertices incident to f and its tiling vertices and tiling edges undergoes planarization, and all of its faces are identified as in Section 4.2.1. Finally, the incidence of a sight cell s is defined as the set of neighbors of the split vertex which can be reached from the centroid of s without crossing any non-tiling edges. Convex faces may be considered their own sight cells.

At this stage, the outer face e.g., (A, Y, D, X, C, B, A) in Fig. 8, can be considered a special case of a non-convex face, as unlike inner faces, the outer face is not bounded. Thus, a rectangular bound, the size of the drawing bounds, is added within which we identify the outer face's sight cells (Fig. 4(b)). The decomposition process is, however, largely the same with the exception that intersections with the bounding rectangles must also be considered. Thus we obtain the drawing D_t. For an example of non-convex face decomposition, consider the outer face's decomposition in Fig. 4(b) indicated in blue ( ).

4.2.3. Selecting sight cells

With all the incidences of the sight cells computed, we must select a pair of sight cells in which we will embed each copy. In line with desideratum **D2**, we aim to select two faces that jointly maximize the number of unique incidence vertices. Moreover, in line with desideratum **D1**, we only select two faces as we only split the selected target vertex in two. Ties are resolved by considering the absolute Euclidean difference between each edge and its Kamda–Kawai optimal graph theoretic length; *the smaller the difference, the better.*

We model each problem using an Integer Linear Programming (ILP) formulation, where we have only integers as variables in the linear system. We are given a set S of cells and the set N of neighbors of the split vertex and look for the pair of sight cells that maximize the incidence of crossing free edges to the copies of the split vertex. We denote by s_i the ith sight cell, v_j the jth neighbor of the split vertex, and $e_{i,j}$ the edge between a copy embedded in cell s_i connected to vertex v_j. We set $s_i = 1$ if a copy is embedded into cell s_i, and set the value to 0 otherwise. Similarly, if vertex v_j has its incident edge to the split vertex assigned to a copy embedded in cell s_i, then $e_{i,j} = 1$. We maximize the realized edges as:

$$\sum_{i \text{ s.t. } s_i \in S} \sum_{j \text{ s.t. } v_j \in N} e_{i,j} \tag{1}$$

under the following constraints: At most two cells can be selected: if $\sum s_i \leq 2$, a neighbor can only be assigned to one cell (meaning the copy to be embedded in that cell): if $\sum_i e_{i,j} \leq 1$ for each neighbor vertex. Additionally, a neighbor cannot be assigned to a cell that has no been selected: $s_i \geq e_{i,j}$. Lastly, if there is no direct visibility between the neighbor and the sight cell, no crossing free edge can be drawn to a copy embedded in that sight cell. We consider $a_{i,j}$ the binary variable that denotes with $a_{i,j} = 1$ the direct visibility of neighbor j to cell s_i. Thus, our last constraint is $e_{i,j} \leq a_{i,j}$.

Fig. 5(b) shows the selected sight cells as blue-shaded areas. These particular sight cells happen to correspond to the previously selected faces, as discussed in Section 4.2.1.

4.3. Step 3: Subface decomposition and selection

With two sight cells selected, we must now determine where within each face to embed the split copy of the target vertex. More specifically, in line with desideratum **D3**, we must find the locations which jointly minimize the number of remaining edge crossings, when connecting to the split vertex's two copies to neighbors not incident to either selected face.

4.3.1. Line segmentation

Given D_T, the drawing in which each face is a sight cell we create a drawing D_R by drawing lines between vertex pair (Fig. 5(a)). Note that, unlike the version presented by Sanatnama et al. [72], we are not considering an arrangement of lines, but rather of segments as we end the line segments on their intersection with the outer bounding rectangle. This allows us to skip area candidates located outside of the drawing canvas. Similar to the previous planarization steps, we replace crossing lines with non-crossing paths of segments, and we identify a set of cells in the resulting drawing. These cells are called *subfaces* $B = \{b_1, b2, \ldots, b_g\}$ (Fig. 5(b)). Note that there is a unique mapping of each subface to the face in F it belongs to.

Sanatnama et al. [72] proved that for such a decomposition of the space, all points within a particular subfaces are equivalent to one another in terms of the number of edge crossings induced when connecting to other vertices in the graph drawing. However, in line with desideratum **D2**, we do not consider all subfaces, but only those within the two selected sight cells (see Fig. 5(b)). Lastly, we compute the centroids of all the subfaces within the two selected faces. For each centroid c and each neighbor u, we store the number of crossings induced by the edge $e_{c,u}$ in an array. Using this array, we solve the problem by selecting one subface in each of the sight cells previously selected. Fig. 5(a) showcases the line decomposition of the previous example graph. The project lines are drawn in yellow ( ), terminating just beyond the user-defined canvas bounds shown in blue ( ). Once planarized, like the graph of Fig. 4(a), the line segmented graph's faces can be identified, highlighted as yellow-shaded areas in Fig. 5(b).

4.3.2. Subface selection

We use a similar process as for the sight cell selection step, but rather than having as input the visibility between the neighbors of the split vertex and the candidate sight cells, we have as input the number of crossings $x_{i,j}$ that would be induced by an edge between a vertex v_j and a copy of the split vertex embedded in subface b_i. We minimize the number of crossings using the following objective function:

$$\sum_{i \text{ s.t. } b_i \in B} \sum_{j \text{ s.t. } v_j \in N} x_{i,j} \tag{2}$$

For each sight cell we only want to select one sight cell, $\sum b_A(i) \leq 1$ where b_A corresponds to the subfaces of one of the previously selected sight cell, and b_B the others. We use the same constraint for the b_B. Similarly to previously, we define a constraint that limits a neighbor to be assigned to only one subcell, and additionally to a sight cell that has been chosen. This ILP returns the two subfaces which induce the least amount of crossings, as well as the assignments of the neighborhoods to the subfaces. Fig. 5(b) highlights the selected subfaces as yellow-shaded areas. Here, the subface incident to vertex 4 allows for the connection to target neighbor 5 by only inducing two additional edge-crossing. Admittedly, in this case, that would have held for all subfaces within the selection sight cell in question.

4.4. Step 4: Embedding of the split vertex copies

With two subfaces selected based on the induced number of edge crossings to non-sight cell-incident vertices, the process of embedding the two split copies is straightforward. While more complex embedding rules based on other graph aesthetics, could have been chosen, we simply put each copy on the coordinates of the centroids of the selected subfaces. Alternatives include basing the exact embedding location on the best edge-angle ratio or edge-angle ratio. Edges are subsequently drawn according to the assignment we just obtained (see Fig. 6(a)). Comparing this new embedding to the original graph drawing of Fig. 6(b), it has indeed successfully resolved 16 of the 18 edge crossings that vertex 2 's incident edges were involved in.

4.5. Implementation

The algorithm was implemented in *Python* around the *NetworkX* [75] package. Specifically, we used its implementation of the Kamada–Kawai [4] and Kavitha et al.'s [74] algorithms. The implementation is made publicly available on GitHub at https://github.com/henry-ehlers.

5. User study design

Lastly, we are interested in evaluating how this approach to vertex duplication affects a user's ability to perform a set of commonly encountered graph analysis tasks. These tasks and the data analyzed are representative of actual types of analyses a user could expect to face when visually analyzing or reading graph drawings. For the purpose of aesthetic simplicity, split vertex embeddings were limited to inner faces only.

5.1. Tasks

Within the context of the work of Lam et al. [76] on proposing a high-level taxonomy of information visualization tasks, we are interested in obtaining *subjective user feedback* on perceived effectiveness and preferences, i.e., *user experience*, and an *objective measure of how well and quickly* users performed select tasks, i.e., *user performance*. For the latter, we can further couch our objective evaluation goals within the mid-level typology of task types by Brehmer et al. [77]. Specifically, we are primarily interested in evaluating how effectively a user can *locate* or *extract* information in a split graph, as compared to a non-split graph. Depending on whether the target entity is clear, or its location known, the tasks performed fall within the *Search* category and its four subcategories of *Lookup*, *Browse*, *Locate*, or *Explore* [77]. Lastly, we define the three actual (or low-level) tasks for users to perform based on Lee et al. [78] taxonomy of graph analysis tasks. Here, we focus primarily on *topological* and *browsing* tasks, as we suspect these to be most affected by splitting operations [36], but also lend themselves more readily to objective measurement. Within the category of *topological tasks*, we define two task types of interest, **T1** and **T2**, and within the *browsing tasks* category, we define a final task **T3**. Thus, our study comprises the following three tasks:

T1 **Vertex adjacency**, i.e., identifying which vertices are adjacent to some given vertex;

T2 **Common connections**, i.e., identifying the set of vertices adjacent to two given vertices; and

T3 **Path tracing**, i.e. identifying which of a set of paths is indeed possible in the given graph drawing.

Each of these three tasks was presented to each participant twice; once for the normal Kamada–Kawai [4] embedded drawing, and once for that graph's split drawing, with randomized vertex labels.

5.2. Data

Normal Drawings—To produce graphs with small-world properties, we simulate three networks using the Watts–Strogatz graph model and embedded using Python's *NetworkX*'s [75] implementation of the Kamada–Kawai algorithm [4]. To ensure that the drawings had sufficient crossings for splitting while remaining readable for non-graph experts, we specifically simulate these graphs with a number of vertices $n = 10$, a mean degree $m = 6$, and a rewiring probability $p = 0.5$. The non-split graph cases, as selected for the study, are depicted in the upper row of Fig. 9, i.e. graphs A_{normal}, B_{normal}, and C_{normal}.

The small number of vertices $n = 10$ was selected for three key reasons. First, the (statistical) importance of edge crossings for graphs such small sizes has been thoroughly documented [24, 27], thus presenting a compelling argument for vertex splitting to be applied to graphs of this size. Second, as the participant list includes non-graph-experts and to ensure enough participants could be gathered, a smaller number of vertices was selected to ensure the study need not take longer than 20–30 min in total to complete. And lastly, our approach was able to consistently produce aesthetically sensible and pleasing results for graphs of such sizes. The combination of mean degree parameter $m = 6$ and rewiring probability $p = 0.5$ were selected to ensure that at least 2 split vertices were needed to resolve 50% of the edge crossings in the embedding.

Split Drawings—Comparable to the study performed by Kobourov et al. [27], each of these three graphs was iteratively split until 50% of the original embedding's edge crossings were resolved. As identifying the aesthetically optimal number of splits is beyond the scope of this paper, we opted to resolve 50% of edge crossings as, in informal previous testing, we found it provided enough challenge for users without being completely overwhelming. Specifically, to ensure participants would actually complete the study, we aimed for a completion time no

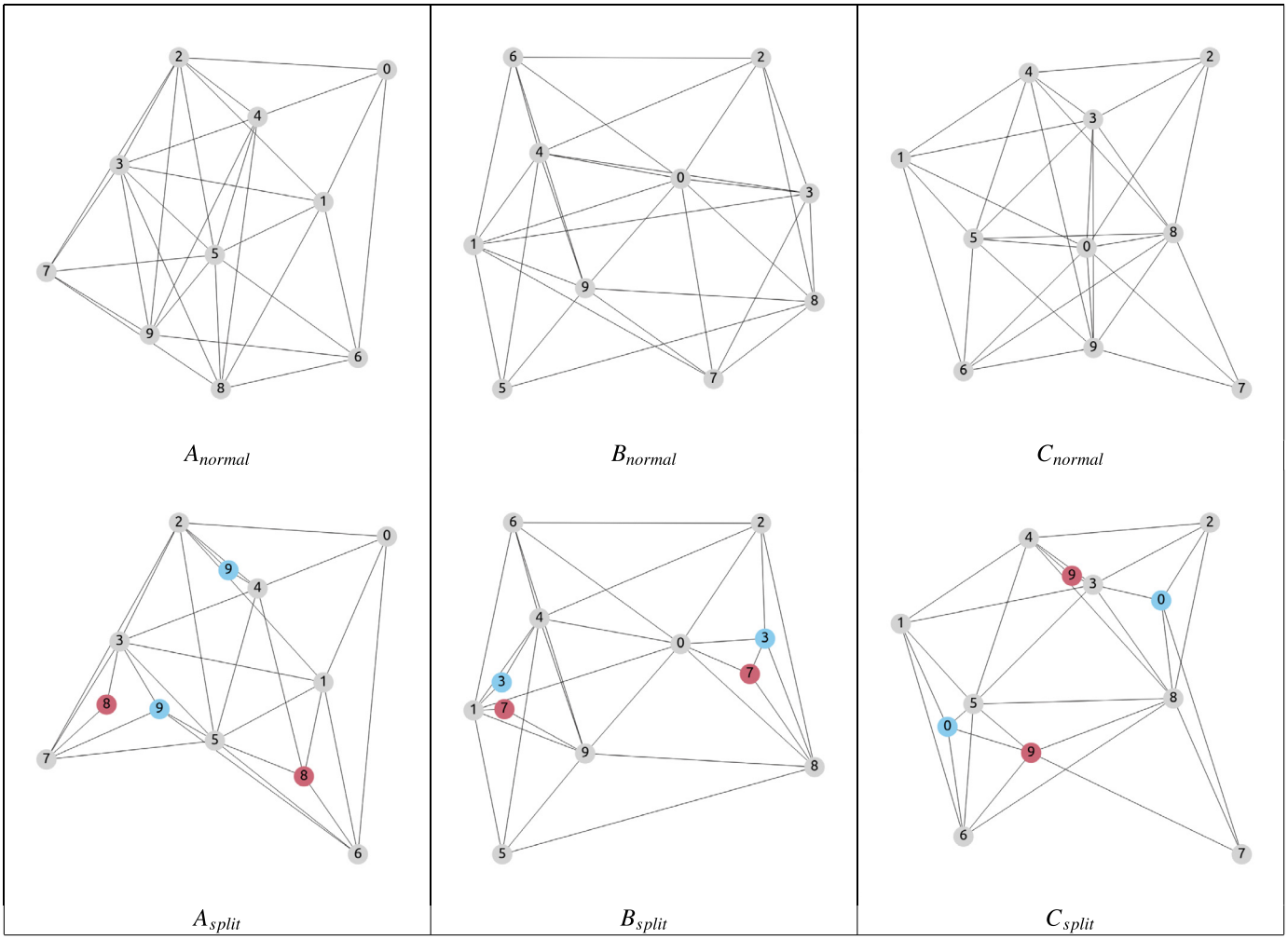


Fig. 9. Three Watts–Strogratz-simulated graphs, (*A*), (*B*), and (*C*), presented to participants during the conducted user study. All graphs were simulated using the same parameters of $n = 10$, $m = 6$, and $p = 0.5$. Each graph was laid out using Python's *NetworkX* library's Kamada–Kawai spring embedding (*normal*) and then split until 50% of the original drawing's edge crossings were resolved (*split*). Please note that, for illustration purposes, the graphs presented here neither have their labels randomized, nor their positions inverted. Split vertices share the same label and are color coded (e.g., 8 or 9) to communicate that they have been split. These graphs' vertices and labels have been enlarged threefold for the sake of readability in print.

longer than 30 min. In order to ensure split vertices were recognizable in non-interactive visualization, split vertices were color-coded using a color-blind-friendly palette [79]. Lastly, to avoid any learning effect, each graph has its labels randomized, and the original drawing had each vertex's positions inverted. The split graph cases, as selected for the study, are depicted in the lower row of Fig. 9, i.e A_{split}, B_{split}, and C_{split}. These are the split counterparts of the upper row of the figure.

5.3. Hypotheses

User Performance—Before beginning the study, participants were instructed on what graph drawings are, what split vertices are, and how the three tasks were to be performed. Users were made aware of the fact that the accuracy and time taken were being evaluated. Over the course of the anonymous evaluation, each participant was presented with each of the three simulated graphs. Such a within-subject experimental design was chosen to ensure we had each user's performance across all tasks and preference for both unsplit and split graphs. For each graph, once for the split and once for the non-split drawing, participants were tasked with answering one of the three aforementioned tasks (**T1**–**3**), as accurately and quickly as possible. Each task had a multiple-choice list from which to select the set of correct answers, which were known to us. Each participant was randomly assigned to one of the six unique assignments of tasks to graphs. The randomization ensured that no participant saw the same graph twice, be it split or unsplit. Additionally, the order in which graphs, tasks, and multiple-choice answers were presented was also randomized to ensure no potential learning effect. Thus, each participant was presented with six questions in total, i.e., vertex-adjacency for a normal and split graph, common-neighborhood for a normal and split graph, and path validity for a normal and split graph. For each question, we recorded the time taken by the participant, as well as the accuracy of the answer, measured as the percentage of choices correctly selected. For this performance evaluation, we posit the following hypotheses:

H1 The accuracy of determining a vertex's adjacency is positively affected by splitting operations, because each individual vertex drawing's degree is lower and, thus, its neighbors are easier to identify.

H2 The accuracy of determining the common neighbors of two given (split) vertex will be positively affected, as the connectivity of individual vertex is clearer.

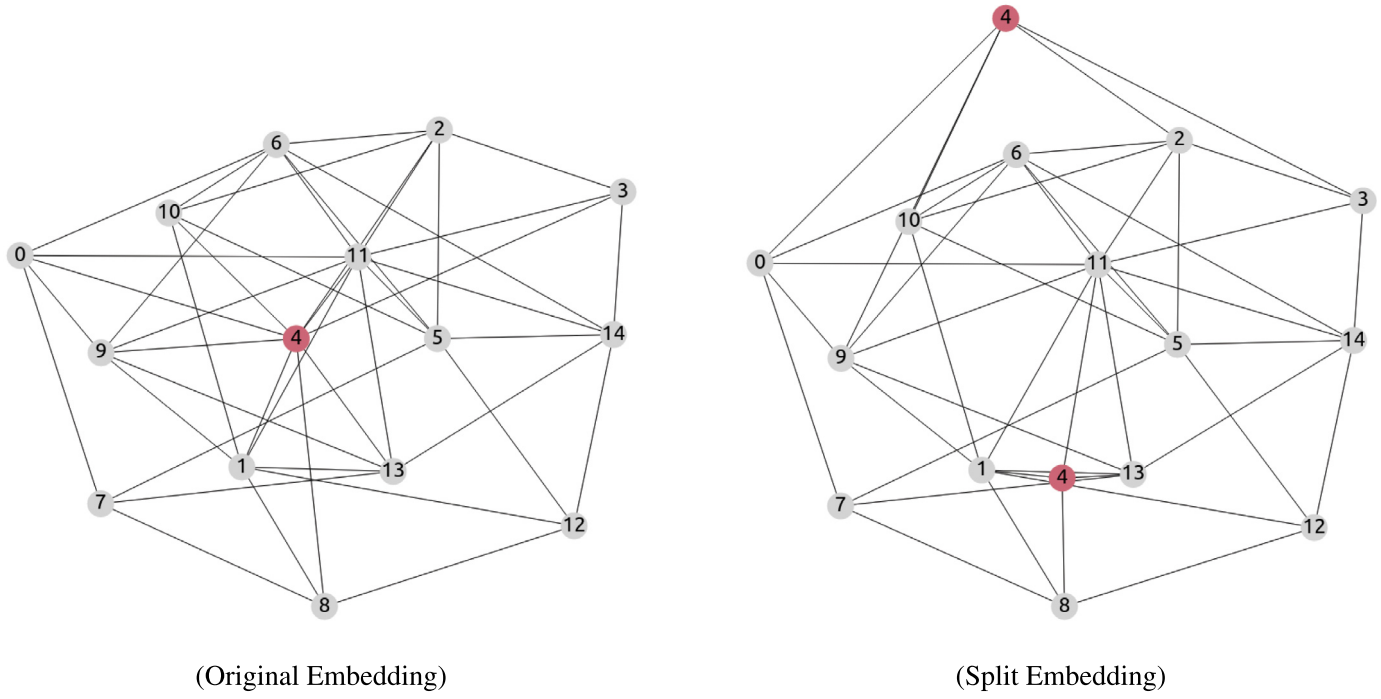


Fig. 10. Vertex Splitting applied to a Watts–Strogratz simulated graphs with numbers of vertices $n = 15$, mean degree of $m = 6$, and rewiring probability of $p = 0.5$: (Original Embedding) shows the initial Kamada–Kawai spring embedded drawing, and (Split Embedding) the drawing produced by splitting vertex 4.

H3 The accuracy of determining whether a path exists is negatively affected by splitting operations, as a path passing through a split vertex necessitates scanning all copies of the vertex to determine whether the next connection exists.

H4 The time necessary to answer all of these questions will be negatively affected by splitting operations, as more vertices are present in a drawing.

User Preference—Additionally, we are interested in evaluating user preferences. As outlined by Lam et al. [76] we aim to evaluate each participant's perceived effectiveness and preference, i.e. *user experience*. Specifically, at the end of the study, for each task, each participant was asked (i) whether they believed split embeddings allowed them to answer questions more accurately, and (ii) whether split embeddings allowed them to answer questions more quickly. Additionally, to evaluate *usefulness* and preference, we additionally probe whether split graphs are easy to learn, and easy to use. We also probe whether they are aesthetically preferable over non-split graphs [80]. All answers were given in the form of a 5-level Likert scale; (1) strongly disagree, (2) disagree, (3) neutral, (4) agree, and (5) strongly agree. For this preference evaluation, we posit the following hypotheses:

H5 Users will aesthetically prefer split over non-split drawings of graphs, as these feature fewer edge crossings.

H6 Users will find split graphs difficult to learn, as especially tasks such as path identification and common neighborhoods, involve some initially counter-intuitive thinking.

H7 Users will find split graphs more difficult to use, for reasons similar to those in **H6**.

6. User study results

52 participants took part in this anonymized, crowd-sourced, online user study. Users were encouraged to use large monitors when performing the study. In this setting, cheating was technically not preventable or controlled. The study was not preregistered. Participants were invited by snowballing, i.e. reaching out to university and academic contacts who were also encouraged to forward the study invitation to contacts of theirs. As all submissions were anonymous, we do not know exactly who participated in this study. Most participants fell within the 21–25, 26–30 and 31–35 year old age brackets with 9, 21, and 15 people respectively. Moreover, of the 52 participants, 18 self-reported being *experienced* with graphs, 18 as *very familiar*, and 6 as *expert*. Only 1 self-reported as being *completely unfamiliar*, and 9 as *not very familiar*. Most participants had completed either a Master's degree (28) or a Ph.D. (13), though 8 reported having completed a Bachelor's degree. Lastly, 16 participants identified as female, 25 as male, and 1 as non-binary. All participants had normal (potentially corrected) vision. As assumptions of normality could not be made or validated for the data at hand, answer accuracy and time taken per task were analyzed using a two-sided, paired Wilcoxon Signed Rank test (Figs. 11 and 12). Preferences were analyzed using a two-sided, one-way Wilcoxon Signed Rank test (Fig. 13) [81]. When probed using Wobbrock et al.'s [82] *Aligned Rank Transformed ANOVA*, no statistically significant or substantively notable association between the expertise of users and their performance or preference was found.

6.1. User performance

Task accuracy—*P*-values for a study such as this should be viewed with skepticism. However, it is interesting to note that the differences in answer accuracy were statistically significant for both the *Vertex Adjacency* and *Path Tracing* tasks, in favor of split graphs (Fig. 11). However, likely a product of the small graphs used in this study, the actual differences in answer accuracy observed are fairly small, i.e., $\Delta_{VertexAdjacency} = 0.091$, $\Delta_{CommonNeighbors} = 0.043$, and $\Delta_{PathTracing} = 0.101$ These findings are in agreement with hypotheses **H1**, i.e., that splitting would positively affect the user's ability to determine the adjacency of given vertices, and **H2**, i.e., that splitting would positively affect

the user's ability to determine the common neighbors of two given vertices. Interestingly, while we hypothesized that vertex splitting would negatively impact the accuracy of path tracing in **H3**, the opposite appears to be found. The results of this analysis are shown in Fig. 11.

Task completion time—In agreement with **H4**, the time needed to complete tasks was negatively impacted by vertex splitting, though, again, the mean differences were very small, $\Delta_{VertexAdjacency} = 1$ second, $\Delta_{CommonNeighbors} = 5$ seconds, and $\Delta_{PathTracing} = 10$ seconds. Differences were statistically significant for both the *Common Neighbors* and *Path Tracing* tasks, where more time was required to complete the tasks with the split graphs. The results of this analysis are shown in Fig. 12.

6.2. User preference

Interestingly, participants showed a fairly strong preference for split over non-split graphs (Fig. 13), with an average Likert scale agreement of 3.89 with the statement "*I found split graphs more aesthetically pleasing*", agreeing with our hypothesis **H5**. Participants also reported split graphs as being both easy to learn and understand, disagreeing with our posited hypotheses **H6** and **H7**. Lastly, on average, users perceived correctly that split graphs allow them to answer all questions more accurately, though no definitive conclusions regarding perceived completion time could be made. The results of this analysis are shown in Fig. 13.

6.3. Results summary

In summary, we find that vertex splitting does not meaningfully affect the users' ability to identify shared neighbors given two vertices. Vertex splitting even *improves the accuracy* in identifying the adjacency of a given vertex or checking the validity of a set of given paths. However, it also *negatively affected the time* taken to perform the three tasks—in particular the identification of valid paths. Furthermore, participants reported *notable aesthetic preference* for split over non-split drawings of the graphs shown. Participants also found split graphs *easy to learn and understand*. All in all, these results showcase that vertex splitting, and our approach to vertex splitting, could be useful in improving the readability and aesthetic quality of small graph drawings.

7. Discussion and future outlook

While we hope to have demonstrated that vertex splitting, and our approach in particular, can be useful in assisting users for small graph drawings, a number of issues, both conceptual and computational, remain to be addressed in future work.

7.1. Algorithm and implementation

Computational Scalability: To ensure such an approach could find application to larger and denser graph drawings, a more computationally scalable approach to the face identification problem is needed as our current approach is centered around Kavitha et al.'s [74] $O(m^2n)$ minimum cycle basis detection algorithm as implemented in *Networkx* for m edges and n vertices. This implementation forms the key bottleneck at three steps, namely the detection of sigh cells, the detection of subfaces, and, most severely, the detection and (recursive) checking of faces (Fig. 14). Here, moving the implementation out of *Networkx* would allow for the planar graphs to be stored as a rotation system, i.e. circular, doubly-connected edge lists, with pointers to faces.

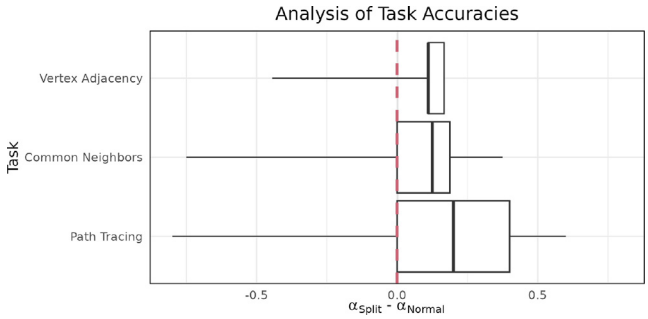


Fig. 11. Box plot of performed paired Wilcoxon signed rank test [83] for answer accuracy across the three task types (**T1**–**3**). *P*-values are calculated using a null hypothesized difference of $\mu_0 = 0$. At an *a-priori* specified significance level of $\alpha = 0.05$, accuracies are statistically significantly different. More specifically, for **T1** and **T3**, users were more accurate when using split graphs, with a median difference of 11.1% and 20.0%, and *p*-values of $p_{T1} = 0.000003$ and $p_{T2} = 0.0144$, respectively. Task **T2** was not statistically significant between groups with a median difference of 12.5% and $p_{T1} = 0.1104$.

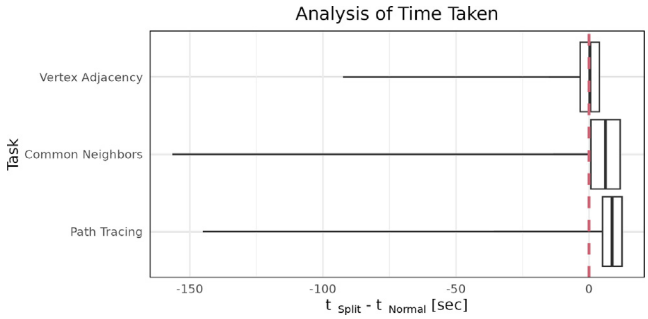


Fig. 12. Box plot of performed paired Wilcoxon signed rank test [83] for the time taken to answer the three task types (**T1**–**3**). *P*-values are calculated using a null hypothesized difference of $\mu_0 = 0$. At an *a-priori* specified significance level of $\alpha = 0.05$, the differences in time taken are statistically significantly different for **T2** and **T3**. More specifically, users were slower using split graphs, with a median difference of 6.17 s ($\sim$ 13.9% slower) and 8.61 s ($\sim$ 14% slower), and *p*-values of $p_{T2} = 0.036$ and $p_{T3} = 4.967E^{-05}$, respectively. Task **T1** was not statistically significant between groups with a median difference of 0.306 s and $p_{T1} = 0.866$. .

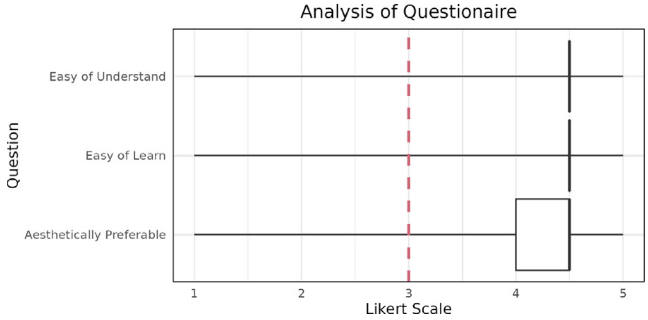


Fig. 13. Box plot of performed paired Wilcoxon signed rank tests [83] on Likert-scale user responses. Answers range from 1 (Strongly Disagree) to 5 (Strongly Agree). *P*-values calculated with a null hypothesized value of $\mu_0 = 3$. At an *a priori* specified significance level of $\alpha = 0.05$, **T1**, **T2**, and **T3** were all statistically significant with median differences of $\sim$ 4.5, and *p*-values of $4.436E - 08$, $2.656E - 07$, and $6.665E - 05$, respectively.

Additionally, moving the entire implementation to a more efficient, low-level programming language, or potentially parallelizing the face identification on GPU, could be useful in making this approach applicable to larger systems.

Beyond Edge Crossings: While this approach produces more aesthetically pleasing results for smaller graphs, it fails to do so consistently for increasing numbers of vertices. For example, consider in Fig. 10 the edges placed between the top-most split copy of vertex 4 and vertices 9 and 10, which are difficult to distinguish from one another. Here, a more sophisticated cost function incorporating not only the number of edge crossings, but other quantitative graph aesthetic criteria, would allow for a better selection of target vertices, sight cells, and subfaces. Additionally, such criteria could assist in choosing better embedding locations within selected subfaces. Here, some aesthetic criteria of interest could include, for example, edge crossing angles, edge length ratios, edge angle ratios, vertex density, or edge/vertex occlusion [26]. For the example given above, the edges $e_{(9,4)}$ and $e_{(10,4)}$ would produce a poor edge angle ratio, guiding the algorithm away from that particular embedding location. Additionally, this could allow for a more sophisticated approach to tie resolution. Currently, selected target vertex (Fig. 3), sight cell (Fig. 4), and subface (Fig. 5) ties are resolved quite naively, such as vertex degree to select an ultimate target vertex to split from a set of vertices with identical crossings numbers. In general, the application of vertex splitting to improve aesthetic criteria beyond edge crossings would be interesting. Of course, the inclusion of such additional criteria would also bring with it additional computational cost as well.

Beyond Local Optimality: Given a more computationally tractable implementation, one could consider relaxing desideratum **D2**, the minimization of edge-crossing-free connections, and focus instead only on desideratum **D3**, the minimization of the total edge-crossing number. The unfortunate side effect of first satisfying desideratum D2 and then D3 can be seen in Fig. 6(a), where the lower of the two split vertices is suboptimally placed. Had it been placed to the left of the edge connecting vertices 4 and 6, instead of the right, one additional edge crossing could have been resolved. However, as the embedding face was selected before the subface within it, these two faces were tied in the number of incident adjacent vertices during the face-selection step. Here, instead of selecting a face and then a subface within it, this would allow for the straightforward selection of 2 (or k) subfaces from the line-segment-tiled embedding (Fig. 5(a)). While the tiling of the entire embedding would be more computationally expensive, it would also allow for the identification of a vertex split configuration that minimizes the number of crossings for this graph embedding.

Beyond Single Splits: In line with desideratum **D3**, i.e. minimizing the number of split vertices, we only split a selected vertex in two. However, beyond identifying *where* to embed split copies, it would be interesting to also identify *how many* splits are necessary to minimize the selected graph aesthetic cost function. On the one hand, extending the ILP selection of sight cells and subfaces to allow for a selection of arbitrary numbers of splits would be conceptually straightforward. On the other hand, it would also necessitate tiling more sight cells into subfaces, thereby introducing additional computational overhead.

Combining *a priori*, Interactive, and Algorithmic Splitting: It would be interesting to combine this technique with existing approaches to vertex splitting, such as *a priori* specification of vertices to split completely. Such *a priori* split vertices would probably resolve many aesthetic issues from the beginning, making the subsequent algorithmic splitting operations easier and faster. Alternatively, an interactive, human-in-the-loop approach to graph drawing could allow users to select vertices to split and/or embedding locations, which could side-step some of the computational and conceptual complexity discussed. This would, of course, also require additional features, such as interactive highlighting of split copies [36].

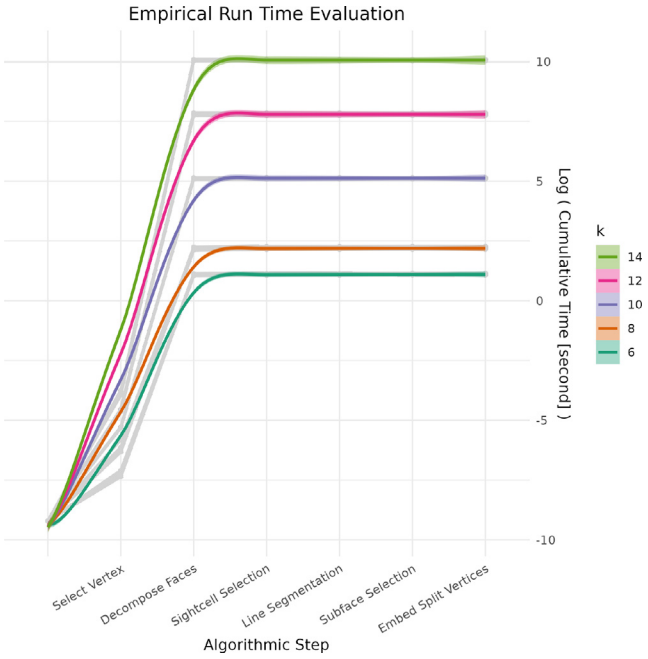


Fig. 14. Cumulative empirical (logarithmic) runtimes of vertex splitting method evaluated on k-complete graphs of increasing size, from $k = 6, 8, 10, 12$ and 14. Time was collected for the 6 steps of the algorithm, namely *Select Vertex* (Fig. 3(a) and (b)), *Decompose Faces* (Fig. 4(a)), *Subface Selection* (Fig. 4(b)), *Line Segmentation* (Fig. 5(a)), *Sight Cell Selection* (Fig. 5(b)), and *Embed Split Vertex* (Fig. 6(a)). For each scenario, 100 graphs were simulated and the time taken to perform these steps was recorded; shown as light gray lines and points. For each k graph, the smoothed average run time and 95% confidence interval of its 100 runs is calculated and overlaid in color. As expected, the face decomposition step, reliant on Kavitha et al.'s [74] $O(m^2n)$ algorithm, where m and n denote the number of edges and vertices respectively, forms the computational bottleneck of our implementation. .

Compromising the Mental Map: Motivated by the preservation of a user's mental map, graph drawings were not re-embedded after each vertex split. For graphs of this size ($10 \leq n \leq 15$) and density, this approach still functions. However, even for graphs of size $|V| = 20$, we encounter several limitations. First, as the number of vertices increases, so does the complexity of the line segmentation (Fig. 5(a)), resulting in smaller and smaller subfaces. Embedding a split vertex copy within such a subface can cause considerable occlusion of both incident and nearby edges, as well as nearby vertices. Moreover, for graphs of high edge density, a single split will often not resolve many edge crossings, and necessitate either splitting said vertex not twice, but k times, or splitting additional vertices. Here, allowing for a split graph drawing to be re-embedded using, for example, Kamada–Kawai, would allow for better graph aesthetic criteria in the final drawing, as well as additional edge crossings to be resolved "passively".

7.2. User performance and graph aesthetics

Extending the User Study: While the results of this user study are promising, and corroborate the findings of Henry et al. [36], they are, of course, limited by the size and complexity of the graphs chosen. Here, a larger user study featuring larger and denser graphs [27], more varied graph analysis tasks [78], and graphs simulated using various graphs models [84–88], would

provide the opportunity to more completely investigate the effect of vertex splitting on both performance and preference. Of course, this would also necessitate a much larger number of participants, and then require the use of a crowd-sourced or Amazon Mechanical Turk approach to data acquisition. However, such a study would be instrumental in uncovering both *how* and *when* to split, in order to most effectively assist users.

A Quantitative Evaluation: Beyond studying user performance and preference, it would be valuable to study the effectiveness of such an approach to vertex splitting quantitatively. More specifically, it would be interesting to calculate graph aesthetic metrics, beyond simply the number of edge crossings, for multiple graphs simulated using different random graph models, vertex sizes, and fixed split numbers, to observe how these metrics change across iterations (i.e. splits). Combining such empirical results with an additional user study could help us understand when vertex splitting is useful and most appropriate. This could be as simple as understanding for what graph sizes and edge densities users prefer split over non-split graphs, or to what degree they do so. This could subsequently lay the groundwork for developing a graph-aesthetic-based stopping criterion for vertex splitting. Beyond the number of edge crossings resolved, such a stopping criterion could be based on, for example, edge angle ratios, edge length ratios, or (remaining) edge crossing angles.

Application to Real-World Networks: Our results showcase the potential for vertex splitting as a means of improving readability. However, owing to the computational complexity of our approach, we are currently limited to smaller, but also *in-silico* results. Subsequently, beyond extending our method along the lines discussed here, studying their utility to actual biological pathways, social networks, or transportation networks would be interesting. Adapting our approach for such real-world networks would present additional domain-specific challenges, such as canonical, and potential non-straight line, representations, such as transportation maps or hand-drawn embeddings, and analytical tasks that may be negatively affected by vertex splitting.

8. Conclusion

Here, we have presented a novel, systematic approach to vertex splitting for static, straight-line graph drawings, agnostic of how their embedding was produced. We have shown its applicability to small-scale graphs. A conducted user study demonstrated that (this particular approach to) vertex splitting allowed the users to answer three graph analysis tasks more accurately, at the cost of requiring additional time to do so. Additionally, these split graphs were aesthetically preferred over non-split graphs, while still being considered easy to understand and learn. These results, as well as the pragmatic use of vertex splitting in the field, indicate to us that vertex splitting could be a useful approach to making dense graphs more readable. We note a sizeable gap between graph theory and visualization application when it comes to the study of vertex splitting; a gap hopefully closed with future research. Beyond making this approach more computationally scalable, its immediate application to small-scale domain networks, such as metabolic pathways, would be an interesting next step.

Acknowledgements

The authors acknowledge TU Wien Bibliothek for financial support through its Open Access Funding Programme.

CRediT authorship contribution statement

Henry Ehlers: Conceptualization, Methodology, Software, Validation, Data curation, Writing & drafting, Review & editing, Visualization. **Anaïs Villedieu:** Methodology, Formal analysis, Investigation, Data curation, Review & editing. **Renata G. Raidou:** Validation, Review & editing, Supervision, Project administration. **Hsiang-Yun Wu:** Investigation, Review & editing, Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request

References

[1] Koutrouli M, Karatzas E, Paez-Espino D, Pavlopoulos GA. A guide to conquer the biological network era using graph theory. Front Bioeng Biotechnol 2020;8:34. http://dx.doi.org/10.3389/fbioe.2020.00034.
[2] Eades P. A heuristic for graph drawing. Congr Numer 1984;42:149–60, URL https://cir.nii.ac.jp/crid/1573387448853684864.
[3] Fruchterman TMJ, Reingold EM. Graph drawing by force-directed placement. Softw - Pract Exp 1991;21(11):1129–64. http://dx.doi.org/10.1002/spe.4380211102.
[4] Kamada T, Kawai S. An algorithm for drawing general undirected graphs. Inform Process Lett 1989;31(1):7–15. http://dx.doi.org/10.1016/0020-0190(89)90102-6.
[5] Bekos MA, Förster H, Geckeler C, Holländer L, Kaufmann M, Spallek AM, et al. A heuristic approach towards drawings of graphs with high crossing resolution. Comput J 2021;64(1):7–26. http://dx.doi.org/10.1093/comjnl/bxz133.
[6] Shabbeer A, Ozcaglar C, Gonzalez M, Bennett KP. Optimal embedding of heterogeneous graph data with edge crossing constraints. 2010, URL https://cseweb.ucsd.edu/~lvdmaaten/workshops/nips2010/papers/bennett.pdf.
[7] Huang W, Eades P, Hong S-H, Lin C-C. Improving multiple aesthetics produces better graph drawings. J Vis Lang Comput 2013;24(4):262–72. http://dx.doi.org/10.1016/j.jvlc.2011.12.002.
[8] Ahmed R, De Luca F, Devkota S, Kobourov S, Li M. Graph drawing via gradient descent, $(GD)^2$. In: Auber D, Valtr P, editors. Graph drawing and network visualization. Lecture notes in computer science, Cham: Springer International Publishing; 2020, p. 3–17. http://dx.doi.org/10.1007/978-3-030-68766-3_1.
[9] Kobourov SG. Spring embedders and force directed graph drawing algorithms. 2012, http://dx.doi.org/10.48550/arXiv.1201.3011, arXiv.
[10] Pavlopoulos GA, Secrier M, Moschopoulos CN, Soldatos TG, Kossida S, Aerts J, et al. Using graph theory to analyze biological networks. BioData Min 2011;4(1):10. http://dx.doi.org/10.1186/1756-0381-4-10.
[11] Suh A, Hajij M, Wang B, Scheidegger C, Rosen P. Persistent homology guided force-directed graph layouts. IEEE Trans Vis Comput Graphics 2020;26(1):697–707. http://dx.doi.org/10.1109/TVCG.2019.2934802, http://dx.doi.org/10/ggmnnw, Conference Name: IEEE Transactions on Visualization and Computer Graphics.
[12] Yoghourdjian V, Dwyer T, Klein K, Marriott K, Wybrow M. Graph thumbnails: Identifying and comparing multiple graphs at a glance. IEEE Trans Vis Comput Graphics 2018;24(12):3081–95. http://dx.doi.org/10.1109/TVCG.2018.2790961, Conference Name: IEEE Transactions on Visualization and Computer Graphics.
[13] Liu Y, Safavi T, Dighe A, Koutra D. Graph summarization methods and applications: A survey. ACM Comput Surv 2018;51(3):62:1–34. http://dx.doi.org/10.1145/3186727.
[14] Gray K, Li M, Ahmed R, Rahman MK, Azad A, Kobourov S, et al. A scalable method for readable tree layouts. IEEE Trans Vis Comput Graphics 2023;1–15. http://dx.doi.org/10.1109/TVCG.2023.3274572, Conference Name: IEEE Transactions on Visualization and Computer Graphics.
[15] Shneiderman B. The eyes have it: a task by data type taxonomy for information visualizations. In: Proceedings 1996 IEEE symposium on visual languages. 1996, p. 336–43. http://dx.doi.org/10.1109/VL.1996.545307.
[16] Kanehisa M, Furumichi M, Sato Y, Kawashima M, Ishiguro-Watanabe M. KEGG for taxonomy-based analysis of pathways and genomes. Nucleic Acids Res 2022;gkac963. http://dx.doi.org/10.1093/nar/gkac963.

[17] Kanehisa M. Toward understanding the origin and evolution of cellular organisms. Prot Sci 2019;28(11):1947–51. http://dx.doi.org/10.1002/pro.3715.

[18] Kanehisa M, Goto S. KEGG: kyoto encyclopedia of genes and genomes. Nucleic Acids Res 2000;28(1):27–30. http://dx.doi.org/10.1093/nar/28.1.27.

[19] Mustafin ZS, Lashin SA, Matushkin YG, Gunbin KV, Afonnikov DA. Orthoscape: a cytoscape application for grouping and visualization KEGG based gene networks by taxonomy and homology principles. BMC Bioinformatics 2017;18(1):1–9. http://dx.doi.org/10.1186/s12859-016-1427-5.

[20] Arakawa K, Kono N, Yamada Y, Mori H, Tomita M. KEGG-based pathway visualization tool for complex omics data. In Silico Biol 2005;5(4):419–23, URL https://content.iospress.com/articles/in-silico-biology/isb00199.

[21] Wang J, Zhong J, Chen G, Li M, Wu F-X, Pan Y. ClusterViz: A cytoscape APP for cluster analysis of biological network. IEEE/ACM Trans Comput Biol Bioinform 2015;12(4):815–22. http://dx.doi.org/10.1109/TCBB.2014.2361348, http://dx.doi.org/10/gnhcss, Publisher: Institute of Electrical and Electronics Engineers (IEEE).

[22] Jia M, Swaminathan S, Wurtele ES, Dickerson J. MetNetGE: Visualizing biological networks in hierarchical views and 3D tiered layouts. In: 2009 IEEE international conference on bioinformatics and biomedicine workshop. 2009, p. 287–94. http://dx.doi.org/10.1109/BIBMW.2009.5332109.

[23] Jia Y, Garland M, Hart JC. Social network clustering and visualization using hierarchical edge bundles. Comput Graph Forum 2011;30(8):2314–27. http://dx.doi.org/10.1111/j.1467-8659.2011.02037.x.

[24] Purchase HC, Carrington D, Allder J-A. Empirical evaluation of aesthetics-based graph layout. Empir Softw Eng 2002;7(3):233–55. http://dx.doi.org/10.1023/A:1016344215610.

[25] Purchase H. Effective information visualisation: a study of graph drawing aesthetics and algorithms. Interact Comput 2000;13(2):147–62. http://dx.doi.org/10.1016/S0953-5438(00)00032-1.

[26] Purchase HC. Metrics for graph drawing aesthetics. J Vis Lang Comput 2002;13(5):501–16. http://dx.doi.org/10.1006/jvlc.2002.0232.

[27] Kobourov SG, Pupyrev S, Saket B. Are crossings important for drawing large graphs?. In: Bayro-Corrochano E, Hancock E, editors. Progress in pattern recognition, image analysis, computer vision, and applications. Lecture notes in computer science, vol. 8827, Cham: Springer International Publishing; 2014, p. 234–45. http://dx.doi.org/10.1007/978-3-662-45803-7_20.

[28] Chuzhoy J, Makarychev Y, Sidiropoulos A. On graph crossing number and edge planarization. In: Proceedings of the 2011 annual ACM-SIAM symposium on discrete algorithms. Proceedings, Society for Industrial and Applied Mathematics; 2011, p. 1050–69. http://dx.doi.org/10.1137/1.9781611973082.80.

[29] Garey MR, Johnson DS. Crossing number is NP-complete. SIAM J Algebr Discrete Methods 1983;4(3):312–6. http://dx.doi.org/10.1137/0604033, Publisher: Society for Industrial and Applied Mathematics.

[30] Hliněný P. Crossing number is hard for cubic graphs. J Combin Theory Ser B 2006;96(4):455–71. http://dx.doi.org/10.1016/j.jctb.2005.09.009.

[31] Ozawa T, Takahashi H. A graph-planarization algorithm and its application to random graphs. In: Saito N, Nishizeki T, editors. Graph theory and algorithms. Berlin, Heidelberg: Springer; 1981, p. 95–107. http://dx.doi.org/10.1007/3-540-10704-5_9.

[32] Thulasiraman K, Jayakumar R, Swamy M. On maximal planarization of nonplanar graphs. IEEE Trans Circuits Syst 1986;33(8):843–4. http://dx.doi.org/10.1109/TCS.1986.1085997, Conference Name: IEEE Transactions on Circuits and Systems.

[33] Goldschmidt O, Takvorian A. An efficient graph planarization two-phase heuristic. Networks 1994;24(2):69–73. http://dx.doi.org/10.1002/net.3230240203, _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/net.3230240203.

[34] Jansen BMP, Lokshtanov D, Saurabh S. A near-optimal planarization algorithm. In: Proceedings of the 2014 annual ACM-SIAM symposium on discrete algorithms. Proceedings, Society for Industrial and Applied Mathematics; 2013, p. 1802–11. http://dx.doi.org/10.1137/1.9781611973402.130.

[35] Jansen BMP, de Kroon JJH, Włodarczyk M. Vertex deletion parameterized by elimination distance and even less. In: Proceedings of the 53rd annual ACM SIGACT symposium on theory of computing. STOC 2021, New York, NY, USA: Association for Computing Machinery; 2021, p. 1757–69. http://dx.doi.org/10.1145/3406325.3451068.

[36] Henry N, Bezerianos A, Fekete J-D. Improving the readability of clustered social networks using node duplication. IEEE Trans Vis Comput Graphics 2008;14(6):1317–24. http://dx.doi.org/10.1109/TVCG.2008.141, Conference Name: IEEE Transactions on Visualization and Computer Graphics.

[37] Nöllenburg M, Sorge M, Terziadis S, Villedieu A, Wu H-Y, Wulms J. Planarizing graphs and their drawings by vertex splitting. 2022, http://dx.doi.org/10.48550/arXiv.2202.12293, arXiv.

[38] Eades PD, McKay BD, Wormald NC. On an edge crossing problem. In: Proceedings of the ninth Australian computer science conference. 1986, p. 327–34, URL https://research.monash.edu/en/publications/on-an-edge-crossing-problem.

[39] Purchase H. Which aesthetic has the greatest effect on human understanding? In: DiBattista G, editor. Graph drawing. Lecture notes in computer science, Berlin, Heidelberg: Springer; 1997, p. 248–61. http://dx.doi.org/10.1007/3-540-63938-1_67.

[40] Eklund P, Roberts N, Green S. OntoRama: Browsing RDF ontologies using a hyperbolic-style browser. In: First international symposium on cyber worlds, 2002. Proceedings. 2002, p. 405–11. http://dx.doi.org/10.1109/CW.2002.1180907.

[41] Lee B, Parr C, Plaisant C, Bederson B, Veksler V, Gray W, et al. TreePlus: Interactive exploration of networks with enhanced tree layouts. IEEE Trans Vis Comput Graphics 2006;12(6):1414–26. http://dx.doi.org/10.1109/TVCG.2006.106, Conference Name: IEEE Transactions on Visualization and Computer Graphics.

[42] Riche NH, Dwyer T. Untangling Euler diagrams. IEEE Trans Vis Comput Graphics 2010;16(6):1090–9. http://dx.doi.org/10.1109/TVCG.2010.210.

[43] Wu H-Y, Nöllenburg M, Sousa FL, Viola I. Metabopolis: scalable network layout for biological pathway diagrams in urban map style. BMC Bioinformatics 2019;20(1):187. http://dx.doi.org/10.1186/s12859-019-2779-4.

[44] Yoghourdjian V, Archambault D, Diehl S, Dwyer T, Klein K, Purchase HC, et al. Exploring the limits of complexity: A survey of empirical studies on graph visualisation. Vis Inform 2018;2(4):264–82. http://dx.doi.org/10.1016/j.visinf.2018.12.006.

[45] Wang X, Yen K, Hu Y, Shen H-W. Deepgd: A deep learning framework for graph drawing using GNN. IEEE Comput Graph Appl 2021;41(5):32–44. http://dx.doi.org/10.1109/MCG.2021.3093908.

[46] Didimo W, Liotta G, Romeo SA. Topology-driven force-directed algorithms. In: Brandes U, Cornelsen S, editors. Graph Drawing. Lecture notes in computer science, Berlin, Heidelberg: Springer; 2011, p. 165–76. http://dx.doi.org/10.1007/978-3-642-18469-7_15.

[47] Davidson R, Harel D. Drawing graphs nicely using simulated annealing. ACM Trans Graph 1996;15(4):301–31. http://dx.doi.org/10.1145/234535.234538.

[48] Kalamaras I, Drosou A, Tzovaras D. Multi-objective optimization for multimodal visualization. IEEE Trans Multimed 2014;16(5):1460–72. http://dx.doi.org/10.1109/TMM.2014.2316473.

[49] Tiezzi M, Ciravegna G, Gori M. Graph neural networks for graph drawing. IEEE Trans Neural Netw Learn Syst 2022;1–14. http://dx.doi.org/10.1109/TNNLS.2022.3184967, Conference Name: IEEE Transactions on Neural Networks and Learning Systems.

[50] Didimo W, Liotta G, Montecchiani F. A survey on graph drawing beyond planarity. ACM Comput Surv 2019;52(1):4:1–37. http://dx.doi.org/10.1145/3301281.

[51] Elmqvist N, Fekete J-D. Hierarchical aggregation for information visualization: Overview, techniques, and design guidelines. IEEE Trans Vis Comput Graphics 2010;16(3):439–54. http://dx.doi.org/10.1109/TVCG.2009.84.

[52] Dwyer T, Henry Riche N, Marriott K, Mears C. Edge compression techniques for visualization of dense directed graphs. IEEE Trans Vis Comput Graphics 2013;19(12):2596–605. http://dx.doi.org/10.1109/TVCG.2013.151.

[53] Onoue Y, Kukimoto N, Sakamoto N, Koyamada K. Minimizing the number of edges via edge concentration in dense layered graphs. IEEE Trans Vis Comput Graphics 2016;22(6):1652–61. http://dx.doi.org/10.1109/TVCG.2016.2534519.

[54] Ersoy O, Hurter C, Paulovich F, Cantareiro G, Telea A. Skeleton-based edge bundling for graph visualization. IEEE Trans Vis Comput Graphics 2011;17(12):2364–73. http://dx.doi.org/10.1109/TVCG.2011.233.

[55] Bach B, Riche NH, Hurter C, Marriott K, Dwyer T. Towards unambiguous edge bundling: Investigating confluent drawings for network visualization. IEEE Trans Vis Comput Graphics 2017;23(1):541–50. http://dx.doi.org/10.1109/TVCG.2016.2598958.

[56] Dwyer T, Mears C, Morgan K, Niven T, Marriott K, Wallace M. Improved optimal and approximate power graph compression for clearer visualisation of dense graphs. In: 2014 IEEE Pacific visualization symposium. 2014, p. 105–12. http://dx.doi.org/10.1109/PacificVis.2014.46.

[57] Zeqian Shen, Kwan-Liu Ma, Eliassi-Rad T. Visual analysis of large heterogeneous social networks by semantic and structural abstraction. IEEE Trans Vis Comput Graphics 2006;12(6):1427–39. http://dx.doi.org/10.1109/TVCG.2006.107.

[58] Nguyen QH, Hong S-H, Eades P, Meidiana A. Proxy graph: Visual quality metrics of big graph sampling. IEEE Trans Vis Comput Graphics 2017;23(6):1600–11. http://dx.doi.org/10.1109/TVCG.2017.2674999.

[59] Hu J, Chu TT, Hong S-H, Chen J, Meidiana A, Torkel M, et al. BC tree-based spectral sampling for big complex network visualization. Appl Netw Sci 2021;6(1):60. http://dx.doi.org/10.1007/s41109-021-00405-3.

[60] van Ham F, Wattenberg M. Centrality based visualization of small world graphs. Comput Graph Forum 2008;27(3):975–82. http://dx.doi.org/10.1111/j.1467-8659.2008.01232.x.

[61] Lin C-C, Huang W, Liu W-Y, Wu S-F. A novel centrality-based method for visual analytics of small-world networks. J Vis 2019;22(5):973–90. http://dx.doi.org/10.1007/s12650-019-00582-5.
[62] Chen K-T, Dwyer T, Marriott K, Bach B. DoughNets: Visualising networks using torus wrapping. In: Proceedings of the 2020 CHI conference on human factors in computing systems. New York, NY, USA: Association for Computing Machinery; 2020, p. 1–11. http://dx.doi.org/10.1145/3313831.3376180.
[63] Chen K-T, Dwyer T, Bach B, Marriott K. It's a wrap: Toroidal wrapping of network visualisations supports cluster understanding tasks. In: Proceedings of the 2021 CHI conference on human factors in computing systems. New York, NY, USA: Association for Computing Machinery; 2021, p. 1–12. http://dx.doi.org/10.1145/3411764.3445439.
[64] Noronha A, Daníelsdóttir AD, Gawron P, Jóhannsson F, Jónsdóttir S, Jarlsson S, et al. ReconMap: an interactive visualization of human metabolism. Bioinformatics 2017;33(4):605–7. http://dx.doi.org/10.1093/bioinformatics/btw667.
[65] Wu H-Y, Nöllenburg M, Viola I. Multi-level area balancing of clustered graphs. IEEE Trans Vis Comput Graphics 2022;28(7):2682–96. http://dx.doi.org/10.1109/TVCG.2020.3038154, Conference Name: IEEE Transactions on Visualization and Computer Graphics.
[66] Nielsen SS, Ostaszewski M, McGee F, Hoksza D, Zorzan S. Machine learning to support the presentation of complex pathway graphs. IEEE/ACM Trans Comput Biol Bioinform 2021;18(3):1130–41. http://dx.doi.org/10.1109/TCBB.2019.2938501.
[67] Eades P, de Mendonça N CFX. Vertex splitting and tension-free layout. In: Brandenburg FJ, editor. Graph Drawing. Lecture notes in computer science, Berlin, Heidelberg: Springer; 1996, p. 202–11. http://dx.doi.org/10.1007/BFb0021804.
[68] Lambert A, Dubois J, Bourqui R. Pathway preserving representation of metabolic networks. Comput Graph Forum 2011;30(3):1021–30. http://dx.doi.org/10.1111/j.1467-8659.2011.01951.x, _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1467-8659.2011.01951.x.
[69] Rohrschneider M, Heine C, Reichenbach A, Kerren A, Scheuermann G. A novel grid-based visualization approach for metabolic networks with advanced focus&context view. In: Eppstein D, Gansner ER, editors. Graph drawing. Lecture notes in computer science, Berlin, Heidelberg: Springer; 2010, p. 268–79. http://dx.doi.org/10.1007/978-3-642-11805-0_26.
[70] Mendonça C, Schaffer K, Xavier E, Stolfi J, Faria L, Figueiredo C. The splitting number and skewness of Cn × Cm. Ars Combin 2002;63. URL https://dblp.uni-trier.de/rec/journals/arscom/NetoSXSFF02.html.
[71] Henry N, Fekete J-D, McGuffin MJ. NodeTrix: a hybrid visualization of social networks. IEEE Trans Vis Comput Graphics 2007;13(6):1302–9. http://dx.doi.org/10.1109/TVCG.2007.70582, Conference Name: IEEE Transactions on Visualization and Computer Graphics.
[72] Sanatnama H, Amini A, Haghighi AB, Brahimi F. Positioning a new vertex that minimize the number of new crossings. J Appl Sci 2011;11(12):2260–4. http://dx.doi.org/10.3923/jas.2011.2260.2264.
[73] Koffka K. Principles of Gestalt Psychology. Oxford, England: Harcourt, Brace; 1935, p. 720.
[74] Kavitha T, Mehlhorn K, Michail D, Paluch KE. An (O^2n) algorithm for minimum cycle basis of graphs. Algorithmica 2008;52(3):333–49. http://dx.doi.org/10.1007/s00453-007-9064-z.
[75] Hagberg AA, Schult DA, Swart PJ. Exploring network structure, dynamics, and function using networkx. In: Varoquaux G, Vaught T, Millman J, editors. Proceedings of the 7th Python in science conference. 2008, p. 11–5, URL https://conference.scipy.org/proceedings/scipy2008/paper_2/.
[76] Lam H, Bertini E, Isenberg P, Plaisant C, Carpendale S. Seven guiding scenarios for information visualization evaluation. Research Report 2011-992-04 hal- 00723057, French Institute for Research in Computer Science and Automation (INRIA); 2011, p. 20, URL https://inria.hal.science/hal-00723057.
[77] Brehmer M, Munzner T. A multi-level typology of abstract visualization tasks. IEEE Trans Vis Comput Graphics 2013;19(12):2376–85. http://dx.doi.org/10.1109/TVCG.2013.124, Conference Name: IEEE Transactions on Visualization and Computer Graphics.
[78] Lee B, Plaisant C, Parr CS, Fekete J-D, Henry N. Task taxonomy for graph visualization. In: Proceedings of the 2006 AVI workshop on beyond time and errors novel evaluation methods for information visualization. Venice, Italy: ACM Press; 2006, p. 1. http://dx.doi.org/10.1145/1168149.1168168.
[79] Harrower M, Brewer CA. ColorBrewer.org: An online tool for selecting colour schemes for maps. Cartogr J 2003;40(1):27–37. http://dx.doi.org/10.1179/000870403235002042.
[80] Gao M, Kortum P, Oswald F. Psychometric evaluation of the USe (usefulness, satisfaction, and ease of use) questionnaire for reliability and validity. In: Proceedings of the human factors and ergonomics society annual meeting, vol. 62, no. 1. 2018;1414–8. http://dx.doi.org/10.1177/1541931218621322, Publisher: SAGE Publications Inc.
[81] Wilcoxon F. Individual comparisons by ranking methods. In: Kotz S, Johnson NL, editors. Breakthroughs in statistics: Methodology and distribution. Springer series in statistics, New York, NY: Springer; 1992, p. 196–202. http://dx.doi.org/10.1007/978-1-4612-4380-9_16.
[82] Wobbrock JO, Findlater L, Gergle D, Higgins JJ. The aligned rank transform for nonparametric factorial analyses using only anova procedures. In: Proceedings of the SIGCHI conference on human factors in computing systems. New York, NY, USA: Association for Computing Machinery; 2011, p. 143–6. http://dx.doi.org/10.1145/1978942.1978963.
[83] Bauer DF. Constructing confidence sets using rank statistics. J Amer Statist Assoc 1972;67(339):687–90. http://dx.doi.org/10.1080/01621459.1972.10481279, Publisher: Taylor & Francis _eprint: https://www.tandfonline.com/doi/pdf/10.1080/01621459.1972.10481279.
[84] Watts DJ, Strogatz SH. Collective dynamics of 'small-world' networks. Nature 1998;393(6684):440–2. http://dx.doi.org/10.1038/30918.
[85] Barabási A-L, Bonabeau E. Scale-free networks. Sci Am 2003;288(5):60–9, URL https://www.jstor.org/stable/26060284.
[86] Albert R. Scale-free networks in cell biology. J Cell Sci 2005;118(21):4947–57. http://dx.doi.org/10.1242/jcs.02714.
[87] Barabási A-L, Albert R. Emergence of scaling in random networks. Science 1999;286(5439):509–12. http://dx.doi.org/10.1126/science.286.5439.509, Publisher: American Association for the Advancement of Science.
[88] Bollobás B. Random Graphs. Cambridge studies in advanced mathematics, 2nd ed. Cambridge: Cambridge University Press; 2001, http://dx.doi.org/10.1017/CBO9780511814068.