Splitting Vertices in 2-Layer Graph Drawings

Reyan Ahmed, *Colgate University, Hamilton, NY, 13346, USA*
Patrizio Angelini, *John Cabot University, 00165, Rome, Italy*
Michael A. Bekos, *University of Ioannina, 451 10, Ioannina, Greece*
Giuseppe Di Battista, *Roma Tre University, 00154, Rome, Italy*
Michael Kaufmann, *Universität Tübingen, 72074, Tübingen, Germany*
Philipp Kindermann, *Universität Trier, 54296, Trier, Germany*
Stephen Kobourov, *University of Arizona, Tucson, AZ, 85721, USA*
Martin Nöllenburg, *TU Wien, 1040, Vienna, Austria*
Antonios Symvonis, *NTUA, 10682, Athens, Greece*
Anaïs Villedieu and Markus Wallinger, *TU Wien, 1040, Vienna, Austria*

Bipartite graphs model the relationships between two disjoint sets of entities in several applications and are naturally drawn as 2-layer graph drawings. In such drawings, the two sets of entities (vertices) are placed on two parallel lines (layers), and their relationships (edges) are represented by segments connecting vertices. Methods for constructing 2-layer drawings often try to minimize the number of edge crossings. We use vertex splitting to reduce the number of crossings, by replacing selected vertices on one layer by two (or more) copies and suitably distributing their incident edges among these copies. We study several optimization problems related to vertex splitting, either minimizing the number of crossings or removing all crossings with fewest splits. While we prove that some variants are NP-complete, we obtain polynomial-time algorithms for others. We run our algorithms on a benchmark set of bipartite graphs representing the relationships between human anatomical structures and cell types.

Multilayer networks are used in many applications to model complex relationships between different sets of entities in interdependent subsystems.[1] When analyzing and exploring the interaction between two such subsystems S_t and S_b, bipartite or 2-layer networks arise naturally. The nodes of the two subsystems are modeled as a bipartite vertex set $V = V_t \cup V_b$ with $V_t \cap V_b = \emptyset$, where V_t contains the vertices of the first subsystem S_t and V_b those of S_b. The interlayer connections between S_t and S_b are modeled as an edge set $E \subseteq V_t \times V_b$, forming a bipartite graph $G = (V_t \cup V_b, E)$. Visualizing this bipartite graph G in a clear and understandable way is then a key requirement for designing tools for visual network analysis.[2]

In a *2-layer graph drawing* of a bipartite graph, the vertices are drawn as points on two distinct parallel lines ℓ_t and ℓ_b, and edges are drawn as straight-line segments.[3] The vertices in V_t (*top vertices*) lie on ℓ_t (*top layer*) and those in V_b (*bottom vertices*) lie on ℓ_b (*bottom layer*). In addition to direct applications of 2-layer networks for modeling the relationships between two communities as mentioned above,[2] such drawings also occur in tanglegram layouts for comparing phylogenetic trees[4] or as components in layered drawings of directed graphs.[5]

The primary optimization goal for 2-layer graph drawings is to find permutations of one or both vertex sets V_t, V_b to minimize the number of edge crossings.

0272-1716 © 2023 IEEE
This article has supplementary downloadable material available at https://doi.org/10.1109/MCG.2023.3264244, provided by the authors.
Digital Object Identifier 10.1109/MCG.2023.3264244
Date of publication 6 April 2023; date of current version 18 May 2023.

While the existence of a crossing-free 2-layer drawing can be tested in linear time,[6] the crossing minimization problem is NP-complete even if the permutation of one layer is given.[3] Hence, both fixed-parameter algorithms[7] and approximation algorithms[8] have been published. Further, graph layouts on two layers have also been widely studied in the area of graph drawing beyond planarity.[9] However, from a practical point of view, minimizing the number of crossings in 2-layer drawings may still result in visually complex drawings.[10]

Hence, in this article, as an alternative approach to construct readable 2-layer drawings, we study vertex splitting.[11,14,12,13] The *vertex-split* operation (or *split*, for simplicity) for a vertex v deletes v from G, adds two new copies v_1 and v_2 (in the original vertex subset of G), and distributes the edges originally incident to v among the two new vertices v_1 and v_2. Placing v_1 and v_2 independently in the 2-layer drawing can in turn reduce the number of crossings.

Vertex splitting has been studied in the context of the *splitting number* of an arbitrary graph G, which is the smallest number of vertex-splits needed to transform G into a planar graph. The splitting number problem is NP-complete, even for cubic graphs,[15] but the splitting numbers of complete and complete bipartite graphs are known.[16,17] Vertex splitting has also been studied in the context of *split thickness*, which is the minimum maximum number of splits per vertex to obtain a graph with a certain property (e.g., a planar graph or an interval graph).[12]

We study variations of the algorithmic problem of constructing planar or crossing-minimal 2-layer drawings with vertex splitting. In visualizing graphs defined on anatomical structures and cell types in the human body,[18] the two vertex sets of G play different roles and vertex splitting is permitted only on one side of the layout. This motivates our interest in splitting only the bottom vertices. The top vertices may either be specified with a given context-dependent input ordering, e.g., alphabetically, following a hierarchy structure, or sorted according to an important measure, or we may be allowed to arbitrarily permute them to perform fewer vertex splits.

CONTRIBUTIONS

We prove that for a given integer k it is NP-complete to decide whether G admits a planar 2-layer drawing with an arbitrary permutation on the top layer and at most k vertex splits on the bottom layer (see Theorem 1). NP-completeness also holds if at most k vertices can be split, but each an arbitrary number of times (see Theorem 3).

If, however, the vertex order of V_t is given, then we present two linear-time algorithms to compute planar 2-layer drawings, one minimizing the total number of splits (see Theorem 2), and one minimizing the number of split vertices (see Theorem 4). In view of their linear-time complexity, our algorithms may be useful for practical applications; we perform an experimental evaluation of the algorithm for Theorem 2 using real-world datasets stemming from anatomical structures and cell types in the human body.[18]

We further study the setting in which the goal is to minimize the number of crossings (but not necessarily remove all of them) using a prescribed total number of splits. For this setting, we prove NP-completeness even if the vertex order of V_t is given (see Theorem 5). On the other hand, we provide an XP-time algorithm parameterized by the number of allowed splits (see Theorem 6), which, in other words, means that the algorithm has a polynomial running time for any fixed number of allowed splits.

PRELIMINARIES

We denote the order of the vertices in V_t and V_b in a 2-layer drawing by π_t and π_b, resp. If a vertex u precedes a vertex v, then we denote it by $u \prec v$. Although 2-layer drawings are defined geometrically, their crossings are fully described by π_t and π_b, as in the following folklore lemma.

Lemma 1. Let Γ be a 2-layer drawing of a bipartite graph $G = (V_t \cup V_b, E)$. Let (v_1, u_1) and (v_2, u_2) be two edges of E such that $v_1 \prec v_2$ in π_t. Then, edges (v_1, u_1) and (v_2, u_2) cross each other in Γ if and only if $u_2 \prec u_1$ in π_b.

In the following, we formally define the problems we study. For all of them, the input contains a bipartite graph $G = (V_t \cup V_b, E)$ and a split parameter k.

- *Crossing Removal with k Splits—CRS(k):* Decide if there is a planar 2-layer drawing of G after applying at most k vertex-splits to the vertices in V_b.
- *Crossing Removal with k Split Vertices—CRSV(k):* Decide if there is a planar 2-layer drawing of G after splitting at most k original vertices of V_b.
- *Crossing Minimization with k Splits—CMS(k, M):* Decide if there is a 2-layer drawing of G with at most M crossings after applying at most k vertex-splits to the vertices in V_b, where M is an additional integer specified as part of the input.

Note that in CRSV(k), once we decide to split an original vertex, then we can further split its copies without incurring any additional cost. The example in Figure 1(a)–(c) demonstrates the difference between CRS and CRSV.

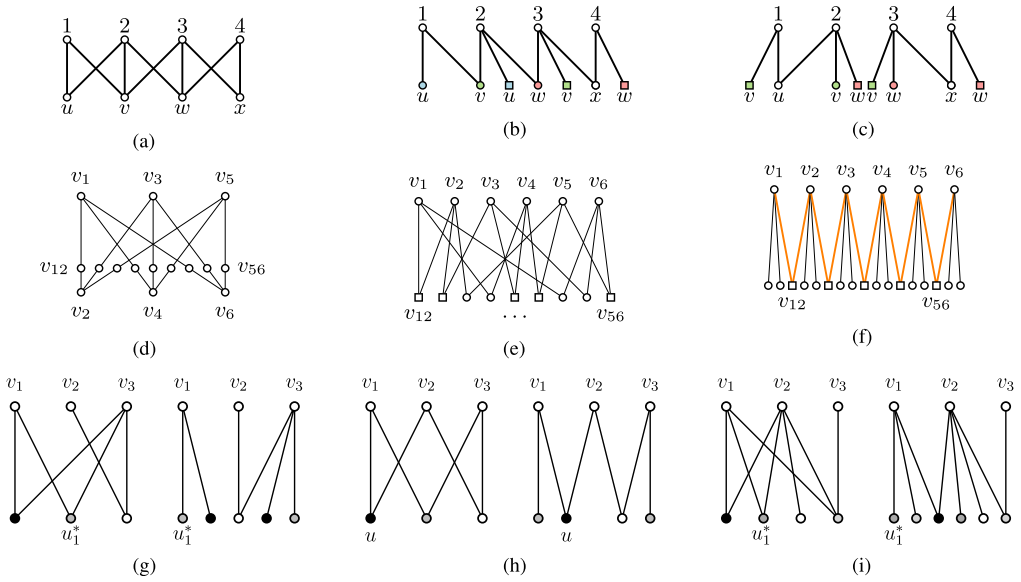

FIGURE 1. Differences between CRS and CRSV problems (a)–(c); illustrations for the reduction in Theorem 1 (d)–(f); illustrations for the optimization algorithm for CRS(k) with fixed order, where vertices in N^+ are colored in shades of gray (g)–(i). (a) Instance G. (b) Optimal CRS solution of G with three splits of three different vertices. (c) Optimal CRSV solution of G with two split vertices. (d) Subdivided graph G'. (e) Instance of CRS(k) for G'. (f) Splits are minimized if and only if G has a Hamiltonian path. (g) Case 1. (h) Case 2. (i) Case 3.

For all problems, we refer to the variant where the order π_t of the vertices in V_t is given as part of the input by adding the suffix *"with Fixed Order."*

The following lemma implies conditions under which a vertex split must occur.

Lemma 2: Let $G = (V_t \cup V_b, E)$ be a bipartite graph and let $u \in V_b$ be a bottom vertex adjacent to two top vertices $v_1, v_2 \in V_t$, with $v_1 \prec v_2$ in π_t. In any planar 2-layer drawing of G in which u is not split, we have that:

- C.1) A top vertex that appears between v_1 and v_2 in π_t can only be adjacent to u;
- C.2) In π_b, u is the last neighbor of v_1 and the first neighbor of v_2.

Proof: If there is a top vertex v' between v_1 and v_2 adjacent to a bottom vertex $u' \neq u$, then $(v,' u')$ crosses (v_1, u) or (v_2, u). If there is a neighbor u'' of v_1 after u in π_b, then the edges (v_1, u'') and (v_2, u) cross. A symmetric argument holds when there is a neighbor of v_2 before u in π_b. □

CROSSING REMOVAL WITH k SPLITS

In this section, we prove that the CRS(k) problem is NP-complete in general and linear-time solvable when the order π_t of the top vertices is part of the input.

Theorem 1: The CRS(k) problem is NP-complete.

Proof: The problem belongs to NP since, given a set of at most k splits for the vertices in V_b, we can check whether the resulting graph is planar 2-layer.[6] □

We use a reduction from the *Hamiltonian Path* problem to show the NP-hardness. Given an instance $G = (V, E)$ of the Hamiltonian Path problem, we denote by G' the bipartite graph obtained by subdividing every edge of G once [see Figure 1(d)]. We construct an instance of the CRS(k) problem [see Figure 1(e)] by setting the top vertex set V_t to consist of the original vertices of G, the bottom vertex set V_b to consist of the subdivision vertices of G', and the split parameter to $k = |E| - |V| + 1$. The reduction can be easily performed in linear time. We prove the equivalence.

Suppose that G has a Hamiltonian path $v_1, \ldots, v_n$. Set $\pi_t = v_1, \ldots, v_n$, and split all the vertices of V_b, except for the subdivision vertex of the edge (v_i, v_{i+1}), for each $i = 1, \ldots, n-1$ [see Figure 1(f)]. This results in $|V_b| - (n-1)$ splits, which is equal to k, since $|V_b| = |E|$ and $n = |V|$. We then construct π_b such that, for each $i = 1, \ldots, n-1$, all the neighbors of v_i appear before all the

neighbors of v_{i+1}, with their common neighbor being the last neighbor of v_i and the first of v_{i+1}. This guarantees that both conditions of Lemma 2 are satisfied for every vertex of V_b. Together with Lemma 1, this guarantees that the 2-layer drawing is planar.

Suppose now that G' admits a planar 2-layer drawing with at most $|E| - |V| + 1$ splits. Since $|E| = |V_b|$ and every vertex of V_b has degree exactly 2 (subdivision vertices), there exist at least $|V| - 1$ vertices in V_b that are not split. Consider any such vertex $u \in V_b$. By C.1 of Lemma 2, the two neighbors of u are consecutive in π_t. Also, these vertices are connected in G by the edge whose subdivision vertex is u. Since this holds for each of the at least $|V| - 1$ nonsplit vertices, we have that each of the $|V| - 1$ distinct pairs of consecutive vertices in V_t (recall that $V_t = V$) is connected by an edge in G. Thus, G has a Hamiltonian path.

Next, we focus on the optimization version of the CRS(k) with the Fixed Order problem. Our recursive algorithm considers a constrained version of the problem, where the first neighbor in π_b of the first vertex in π_t may be prescribed. At the outset of the recursion, there exists no prescribed first neighbor. The algorithm returns the split vertices in V_b and the corresponding order π_b.

In the base case, there is only one top vertex v (i.e., $|V_t| = 1$). Since all vertices in V_b have degree 1, no split takes place. We set π_b to be any order of the vertices in V_b where the first vertex is the prescribed first neighbor of v, if any.

In the recursive case when $|V_t| > 1$, we label the vertices in V_t as $v_1, \ldots, v_{|V_t|}$, according to π_t. If the first neighbor of v_1 is prescribed, we denote it by u_1^*. Also, we denote by N^1 the set of degree-1 neighbors of v_1, and by N^+ the other neighbors of v_1. Note that only the vertices in N^+ are candidates to be split for v_1. In particular, by C.1 of Lemma 2, a vertex in N^+ can avoid being split only if it is also incident to v_2. Further, since any vertex in N^+ that is not split must be the last neighbor of v_1 in π_b, by C.2 of Lemma 2, at most one of the common neighbors of v_1 and v_2 will not be split. Analogously, if u_1^* is prescribed, then it must be split, unless v_1 has degree 1.

In view of these properties, we distinguish three cases based on the common neighborhood of v_1 and v_2. In all cases, we will recursively compute a solution for the instance composed of the graph $G' = (V_t' \cup V_b,' E')$ obtained by removing v_1 and the vertices in N^1 from G, and of the order $\pi_t' = v_2, \ldots, v_{|V_t|}$. We denote by π_b' and s' the computed order and the number of splits for the vertices in V_b'. In the following, we specify for each case whether the first neighbor of v_2 in the new instance is prescribed or not, and how to incorporate the neighbors of v_1 into π_b'.

Case 1: v_1 and v_2 have no common neighbor; see Figure 1(g). In this case, we do not prescribe the first neighbor of v_2 in the instance composed of G' and π_t'. To compute a solution for the original instance, we split each vertex in N^+ so that one copy becomes incident only to v_1. We construct π_b by selecting the prescribed vertex u_1^*, if any, followed by the remaining neighbors of v_1 in any order and, finally, by appending π_b'. This results in $s = |N^+| + s'$ splits.

Case 2: v_1 and v_2 have exactly one common neighbor u: If $u = u_1^*$ and v_1 have a degree larger than 1, then u cannot be the last neighbor of v_1 and must be split. Thus, we perform the same procedure as in Case 1. Otherwise, in the instance composed of G' and π_t', we set u as the prescribed first neighbor of v_2; see Figure 1(h). To compute a solution for the original instance, we split each vertex in N^+, except u, so that one copy becomes incident only to v_1. We construct π_b by selecting the prescribed vertex u_1^*, if any, followed by the remaining neighbors of v_1 different from u in any order and, finally, by appending π_b'. This results in $s = |N^+| - 1 + s'$ splits.

Case 3: v_1 and v_2 have more than one common neighbor: If v_1 and v_2 have exactly two common neighbors u, u' and one of them is u_1^*, say $u = u_1^*$, then u cannot be the last neighbor of v_1, as v_1 has degree larger than 1. Thus, we proceed exactly as in Case 2, using u' as the only common neighbor of v_1 and v_2.

Otherwise, there are at least two neighbors of v_1 different from u_1^*; see Figure 1(i). We want to choose one of these vertices as the last neighbor of v_1, so that it is not split. However, the choice is not arbitrary as this may affect the possibility for v_2 to save the split for a neighbor it shares with v_3. In the instance composed of G' and π_t', we do not prescribe the first vertex of v_2. To compute a solution for the original instance, we simply choose as the last neighbor of v_1 any of its common neighbors with v_2 that has not been set as the last neighbor of v_2 in π_b'. Such a vertex, say u, always exists since v_1 and v_2 have at least two common neighbors different from u_1^*, and can be moved to become the first vertex in π_b'. Specifically, we split all the vertices in N^+, except for u, so that one copy becomes incident only to v_1. We construct π_b by selecting the prescribed vertex u_1^*, if any, followed by the remaining neighbors of v_1 different from u in any order. We then modify π_b' by moving u to be the first vertex. Note that this operation does not affect planarity, as it only involves reordering the set of consecutive degree-1 vertices incident to v_2. Finally,

we append the modified π_b'. This results in $s = |N^+| - 1 + s'$ splits.

Theorem 2: For a bipartite graph $G = (V_t \cup V_b, E)$ and an order π_t of V_t, the optimization version of CRS (k) with fixed order can be solved in $O(|E|)$ time.

Proof: By construction, for each $i = 1, \ldots, |V_t| - 1$, all neighbors of v_i precede all neighbors of v_{i+1} in π_b. Thus, by Lemma 1, the drawing is planar. The minimality of the number of splits follows from Lemma 2, as discussed before the case distinction. In particular, any minimum-splits solution can be shown to be equivalent to the one produced by our algorithm. The time complexity follows as each vertex only needs to check its neighbors a constant number of times. □

We conclude this section by mentioning that the CRS(k) problem had already been considered, under a different terminology, in the context of molecular QCA circuits design.[20] Here, the problem was claimed to be NP-complete, without providing a formal proof. In the same work, when the order π_t of the top vertices is part of the input, an alternative algorithm was proposed based on the construction of an auxiliary graph that has superlinear size. Exploiting linear-time sorting algorithms and observations that allow avoiding explicitly constructing all edges of this graph, the authors were able to obtain a linear-time implementation. We believe that our algorithm of Theorem 2 is simpler and more intuitive, and directly leads to a linear-time implementation.

CROSSING REMOVAL WITH k SPLIT VERTICES

In this section, we prove that the CRSV(k) problem is NP-complete in general and linear-time solvable when the order π_t of the top vertices is part of the input. Ahmed et al.[19] showed that CRSV(k) is FPT when parameterized by k. To prove the NP-completeness, we can use the reduction of Theorem 1. In fact, in the graphs produced by that reduction all vertices in V_b have degree 2. Hence, the number of vertices that are split coincides with the total number of splits.

Theorem 3: The CRSV(k) problem is NP-complete.

For the version with fixed order, we first use C.1 of Lemma 2 to identify vertices that need to be split at least once, and repeatedly split them until each has degree 1. For a vertex $u \in V_b$, we can decide if it needs to be split by checking whether its neighbors are consecutive in π_t and, if u has degree at least 3, all its neighbors different from the first and last have degree exactly 1.

We first perform all necessary splits. For each $i = 1, \ldots, |V_t| - 1$, consider the two consecutive top vertices v_i and v_{i+1}. If they have no common neighbor, no split is needed. If they have exactly one common neighbor u, then we set u as the last neighbor of v_i and the first of v_{i+1}, which allows us not to split u, according to C.2. Since u did not participate in any necessary split, if u is also adjacent to other vertices, then all its neighbors have degree 1, except possibly the first and last. Hence, C.2 can be guaranteed for all pairs of consecutive neighbors of u.

Otherwise, v_i and v_{i+1} have at least two common neighbors and thus have degree at least 2. Hence, all common neighbors of v_i and v_{i+1} must be split, except for at most one, namely the one that is set as the last neighbor of v_i and as the first of v_{i+1}. Since all these vertices are incident only to v_i and v_{i+1}, as otherwise they would have been split by C.1, we can arbitrarily choose any of them, without affecting the splits of other vertices.

At the end, we construct the order π_b so that, for each $i = 1, \ldots, |V_t| - 1$, all the neighbors of v_i precede all the neighbors of v_{i+1}, and the unique common neighbor of v_i and v_{i+1}, if any, is the last neighbor of v_i and the first of v_{i+1}. By Lemma 1, this guarantees planarity. Identifying and performing all unavoidable splits and computing π_b can be easily done in $O(|E|)$ time. Since we only performed unavoidable splits, as dictated by Lemma 2, we have the following.

Theorem 4: For a bipartite graph $G = (V_t \cup V_b, E)$ and an order π_t of V_t, the optimization version of CRSV (k) with fixed order minimizing the number of split vertices can be solved in $O(|E|)$ time.

CROSSING MINIMIZATION WITH k SPLITS

In this section, we consider minimizing crossings (not necessarily removing all), by applying at most k splits. We first prove NP-completeness of the decision problem CMS(k, M) with fixed order and then give a polynomial-time algorithm assuming the integer k is a constant.

Theorem 5: For a bipartite graph $G = (V_t \cup V_b, E)$, an order π_t of V_t, and integers k, M, problem CMS (k, M) with fixed order is NP-complete.

Proof: We reduce from the NP-complete DECISION CROSSING PROBLEM (DCP),[3] where given a bipartite 2-layer graph with one vertex order fixed, the goal is to find an order of the other set such that the number of crossings is at most a given integer M. Given an

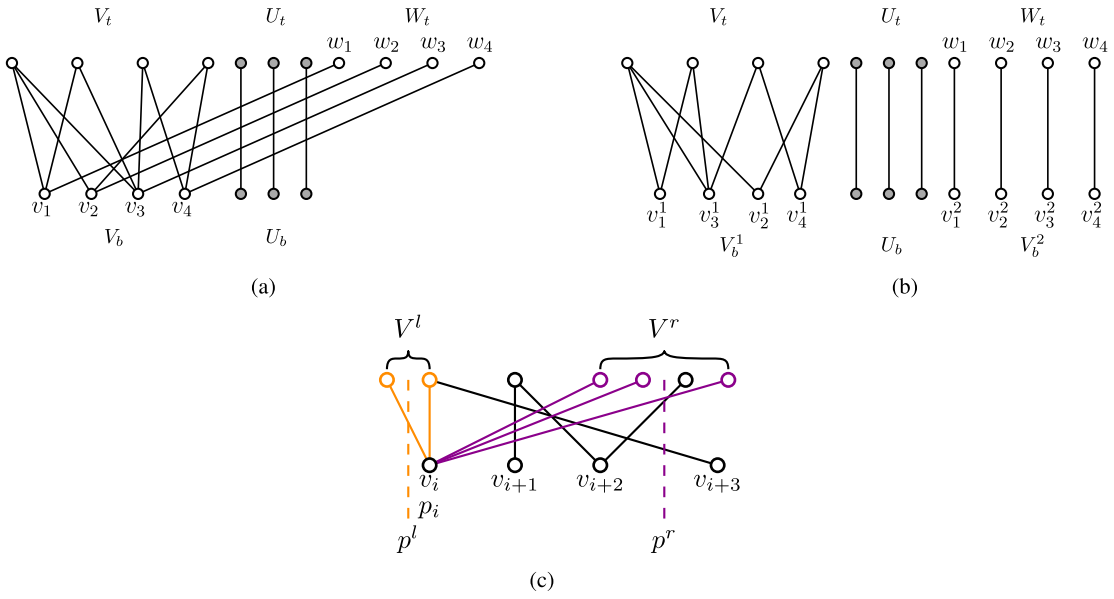

FIGURE 2. Illustrations for Theorem 5 (a)–(b); illustration for the crossing reduction heuristics (c). (a) Instance of CMS(k) constructed from a DCP instance, in light gray the vertices in $U_t \cup U_b$; before splitting. (b) After splitting. (c) Vertex v_i with span 6 and a split in V^l (span 1) and V^r (span 3). The barycenter of V^r is p^r. When moving right from p_i in *CR-count* we process v_{i+1} (reduces $|V^r|$ crossings) and v_{i+2} (reduces $|V^r|$ but adds 1 crossing), but not v_{i+3}.

instance of DCP, i.e., a 2-layer graph $G = (V_t \cup V_b, E)$, with ordering π_t of V_t and integer M, we construct an instance G' of CMS(k, M) where $k = |V_b|$. First let $G' = (V_t' \cup V_b', E')$ be a copy of G. We give an arbitrary ordering π_b to the vertices of V_b'. We then add, respectively, to each vertex set V_t and V_b a set U_t and U_b of $M+1$ vertices and connect each $u \in U_t$ to exactly one $v \in U_b$, forming a matching of size $M+1$. We add the vertices of U_t to π_t (resp. U_b to π_b) after all the vertices of V_t (V_b). We lastly add a set W_t of k vertices to V_t', placed at the end of π_t, such that each $w_i \in W_t$ ($i = 1, \ldots, k$) has exactly one neighbor $v_i \in V_b$ and vice versa; see Figure 2(a). □

Given an ordering π_b^* of V_b that results in a drawing of G with at most M crossings, we show that we can solve the CMS(k, M) instance G'. In G', we split each vertex of V_b to obtain the sets V_b^1 and V_b^2 in which we place exactly one copy of each original vertex. We place V_b^2 after the vertices of U_b in π_b in the same order that the vertices of W_t appear in π_t and draw a single edge between the copies and their neighbor in W_t. We place V_b^1 before the vertices of M_2 in π_b in the same order as in π_b^*. The graph induced by V_b^1 and V_t is the same graph as G, hence it has at most M crossings. Since V_t only has neighbors in V_b and all those neighbors are in V_b^1, it has no other outgoing edges, similarly, all edges incident to vertices in W_t are assigned to the copies in V_b^2. The remaining graph is crossing-free as the vertices in U_t and W_t form a crossing-free matching with the vertices in U_b and V_b^2.

Conversely, let G^* be a 2-layer drawing obtained from G' after k split operations that has at most M crossings. Since each vertex $v \in V_b$ has a neighbor $w \in W_t$, it induces $M+1$ crossings with edges induced by the vertices in $U_t \cup U_b$. Since the vertices in U_b have a single neighbor, they cannot be split, thus every vertex in V_b is split once, and their neighborhood are partitioned for each copy in the following way: one copy receives the neighbor in W_t and one copy receives the remaining neighbors, which are in V_t [see Figure 2(b)], thus avoiding at least $M+1$ crossings induced by $U_t \cup U_b$. Any other split would imply at least $M+1$ crossings. The graph induced by the copies that receive the neighbors in V_t has at most M crossings, thus, the ordering found for those copies is a solution to the DCP instance G.

Next, we present a simple XP-time algorithm for the crossing minimization version of CMS(,) parameterized by the number k of splits, i.e., the algorithm runs in polynomial time $O(n^{f(k)})$, where n is the input size, k is the parameter, and f is a computable

function. Let $G = (V_t \cup V_b, E)$ be a 2-layer graph with vertex orders π_t and π_b and let k be the desired number of splits. Our algorithm executes the following steps. First, it determines a set of splits by choosing k times a vertex from the n vertices in V_b—we enumerate all options. For any vertex $v \in V_b$ split i times in the first step, v is replaced by the set of copies $\{v_1, \ldots, v_{i+1}\}$. The neighborhood $N(v)$ of a vertex $v \in V_b$ is a subset of V_t ordered by π_t. We partition this ordered neighborhood into $i+1$ consecutive subsets, i.e., for each subset, all its elements are sequential in $N(v)$—again, we enumerate all possible partitions. Each set is assigned to be the neighborhood of one of the copies of v. The algorithm then chooses an ordering of all copies of all split vertices and attempts all their possible placements by merging them into the order π_b of the unsplit vertices of V_b. The crossing number of every resulting layout is computed and the graph with minimum crossing number yields the solution to our input. It remains to show that the running time of this algorithm is polynomial for constant k.

Theorem 6: For a 2-layer graph $G = (V_t \cup V_b, E)$ with vertex orders π_t, π_b and a constant $k \in \mathbb{N}$, we can minimize the number of crossings by applying at most k splits in time $O(n^{4\,k})$.

Proof: Let G^* be a crossing-minimal solution after applying k splits on V_b and let us assume that our algorithm would not find a solution with this number of crossings. As our algorithm considers all possibilities to apply k splits, it also attempts the splits applied in G^*. Similarly, the neighborhood partition of G^* and the copy placement are explicitly considered by the algorithm as it enumerates all possibilities. Hence, a solution at least as good as G^* is found, proving correctness. □

Let $n_t = |V_t|$ and $n_b = |V_b|$ with $n = n_t + n_b$. The algorithm initially chooses k times from n_b vertices leading to n_b^k possible sets of copies. Since a vertex has degree at most n_t, there are at most n_t^k possible neighborhoods for each copy. Additionally, there are $(2\,k)!$ orderings of at most $2\,k$ copies. Finally, there are $n_b^{2\,k}$ possible placement of the $2\,k$ ordered copies between the most n_b unsplit vertices in π_b. This leads to an overall runtime of $O((2\,k)! \cdot n^{4\,k}) = O(n^{4\,k})$ to iterate through all possible solutions and select the one with a minimum number of crossings.

CROSSING REDUCTION HEURISTICS

In this section we present two greedy heuristics to iteratively reduce crossings in a two-layer drawing by selecting and splitting vertices; see Algorithms 3–4 in the supplemental material for the pseudocode. The input to the algorithm is a bipartite graph $G = (V_t \cup V_b, E)$, order π_t of V_t and order π_b of V_b. Here we use the barycenter heuristic for the initial orders, but any initial order computed by a crossing-reduction algorithm can also be used. Additionally, an input parameter k is specified that represents a budget of available split operations. For both heuristics, we iteratively perform k splits by selecting the most promising vertex in V_b and a partition of its respective neighbors in V_t into a consecutive set of left neighbors V^l and a consecutive set of right neighbors V^r. After splitting, the original vertex receives the set V^l assigned as neighbors and the copy receives the set V^r. Next we describe how the vertex to be split is selected in each of the heuristics.

First, in the case of the *max-span* heuristic we select the vertex v with the maximum *span* (i.e., the maximum distance between its leftmost neighbor and rightmost neighbor in π_t). Then, we process v by iterating in order over its neighbors and assigning them to either the set of left neighbors V^l or right neighbors V^r depending on the index; see Figure 2(c). In each iteration, we compute the sum of the squared span of V^l and V^r. The minimum value indicates the best partition of v's neighbors. The complexity of the heuristic is linear in the number of edges $O(|E|)$.

Second, the *CR-count* heuristic selects promising split vertices by computing crossings that can be reduced and selects the vertex with potentially most reduced crossings. First, we assign each vertex v_i in V_b a position $p_i \in \mathbb{R}$, which is the barycenter of positions in π_t of its neighbors in V_t. Similarly to *max-span*, we iterate in order over the neighbors of all v_i creating partitions V^l and V^r. In each iteration, we compute the barycenter p^l of V^l and p^r of V^r; see Figure 2(c). Next, we start from position p_i and move in ascending order processing all other vertices in V_b until we reach p^r. To process a vertex $v_j \in V_b$, we look at all edges to its neighbors $N(v_j) \subseteq V_t$ and use a case distinction to count how many crossings can be reduced. In the first case, a neighbor in V_t is left of the leftmost vertex in V^r. Here, we would reduce $|V^r|$ crossings as the new position of v_r would not cross. In the second case, a neighbor is between the leftmost and rightmost vertex in V^r (i.e., we would reduce some crossings but add others). In the third case a neighbor is right of the rightmost vertex in V^r, i.e., no crossings can be reduced. Likewise, we process vertices to the left of v_i up to the barycenter p^l. Finally, after computing all potentially reducible crossings for each vertex and split combination, we

TABLE 1. Statistics about the organ graphs from the hubmap dataset.[18] The density of a graph $g = (v, e)$ with $v = v_t \cup v_b$ is defined as $2|e|/(|v|(|v| - 1))$.

Organ	$\|V\|$	$\|E\|$	Cell types	Genes/proteins	Density	Max degree
Blood	179	461	30	149	0.0289	57
Fallopian Tube	42	32	19	23	0.0371	3
Lung	231	231	69	162	0.008	8
Peripheral Nervous System	3	2	1	2	0.666	2
Thymus	552	658	41	511	0.00432	93
Heart	60	51	15	45	0.028	7
Lymph Nodes	299	491	44	255	0.0110	36
Prostate	43	36	12	31	0.039	3
Ureter	44	53	14	30	0.0560	9
Bone Marrow	343	662	45	298	0.011	25
Kidney	201	237	58	143	0.011	8
Skin	102	90	36	66	0.017	7
Urinary Bladder	46	55	15	31	0.053	9
Brain	381	346	127	254	0.004	5
Large Intestine	124	139	51	73	0.0182	8
Ovary	9	6	3	6	0.166	2
Small Intestine	18	13	5	13	0.084	4
Uterus	61	65	16	45	0.035	9
Eye	145	270	47	98	0.0258	68
Liver	73	57	26	47	0.0216	5
Pancreas	69	100	29	40	0.042	12
Spleen	290	414	65	225	0.009	23

select the combination reducing the crossings most, assign V_l to the original vertex and V_r to the new copy. Both vertices are positioned in their respective barycenters p^l and p^r in the order. The complexity of the algorithm is $O(|V_b|^2|V_t|)$.

EXPERIMENTAL RESULTS

We have experimentally evaluated four of the five algorithms described earlier with 22 real-world datasets: the exact algorithm of CRS(k) with fixed order, the exact algorithm of CRSV(k) with fixed order, and two heuristics for crossing reduction. The algorithm behind Theorem 6 is inefficient in practice. We analyze performance w.r.t. the number of crossings in the layouts, number of vertex splits, number of vertices that we split, the maximum number of splits, and running time.

Experimental Design

We have mentioned that 2-layer drawings have been applied in visualizing graphs defined on anatomical structures and cell types in the human body.[18] There exists a variety of cell types, genes, and proteins related to different organs of the human body. Hierarchical structures have been used to show the relationship between organs to anatomical structures, anatomical structures to cell types, and cell types to genes/proteins. Cell types and genes/proteins situate on a particular layer, unlike anatomical structures. Hence, we can consider a 2-layer graph G, where cell types represent one layer and genes/proteins represent another layer and analyze G before and after splitting. In this section, we consider the real-world 2-layer graphs generated from the dataset of different organs and show the experimental results obtained on those graphs.

Datasets

We use 22 real-world instances of 2-layer graphs;[18] see Table 1.

Interactive Tool Design

We developed an interactive tool, where the user can upload a dataset and visualize the corresponding 2-layer graph; see Figure 4. A dataset can be loaded by pasting JSON-formatted text in the input area on the left-hand side of the interface. We use blue and red colors to draw nodes that represent cell types and genes/proteins, respectively. There is a legend in the top left corner of the interface to describe the color code. Instead of using top and bottom layers, we use left and right layers, for easier node labeling (i.e., the left and right layers represent V_t and V_b). There are multiple radio buttons to the configuration of the drawing. The user can fix the order of either the blue vertices or the red vertices by selecting one set of radio buttons. The number of vertex splits depends on the initial layout. We consider two types of initial layouts: the vertices in each layer are positioned in alphabetical order, or in barycentric order.[5] The user can select an initial order from the input interface by using another set of radio buttons. There are "Draw" and "Split" buttons in the interface. Once the user selects an order for the left layer and clicks the draw button, then the initial ordered layout will be shown on the right side of the interface. Clicking the split button replaces the initial layout and shows the final layout on the right side of the interface.

The right output interface is interactive; the user can see further details using different interactions. When the graph is large the user can scroll up and down to see different parts of the layout. The user can highlight the adjacent edges by clicking on a particular vertex in case of dense layouts. We keep the label texts less than or equal to ten characters. If a label is longer then we show the first ten characters and truncate the rest. If the user puts the mouse over the label or the corresponding vertex, a pop-up message will show the full label. If the user moves out the mouse, the message will be removed too. Besides showing the full label, we also provide other useful information (e.g., the degree and ID of the vertex; see Figure 3).

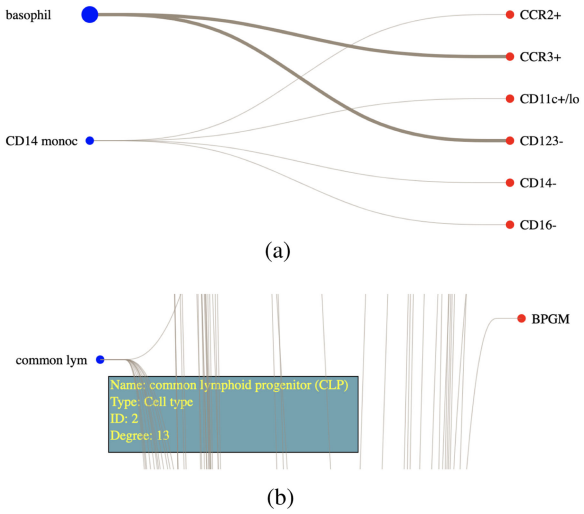

FIGURE 3. Interacting with the system. (a) The system highlights the adjacent edges when the user clicks on a vertex ("basophil" in this case). (b) A pop-up message showing the full label, and other related information.

Evaluation Results

We first evaluate the exact algorithms for CRS(k) and CRSV(k) with fixed order. We have run experiments on 22 organ graphs on four settings:

1) The blue vertices (cell type) are fixed and the initial layout is generated using alphabetical order.
2) The red vertices (gene/protein) are fixed and the initial layout is generated using alphabetical order.
3) The blue vertices are fixed and initial layout is generated using barycentric heuristic.
4) The red vertices are fixed and initial layout is generated using the barycentric heuristic.

For each setting, we provide the initial number of crossings, number of vertices in the top (or left) layers that have fixed order, the number of bottom (or right) vertices, the number of splits, the number of split vertices, and the maximum number of splits; see Tables 2–5 in the supplemental material. The number of crossings in the initial layouts generated from alphabetical order is 2.7 times larger in total than in layouts generated by the barycentric heuristic. The number of splits is 1.58 times larger in total when we fix the gene/protein vertices (Tables 3 and 5). Note that for all organ graphs, the number of gene/protein vertices is relatively larger compared to the cell type vertices. When the cell-type vertices are fixed, there is more flexibility for splitting. Hence, the number of splits in Tables 2 and 4 are smaller than in Tables 3 and 5. Similarly, the maximum number of splits is

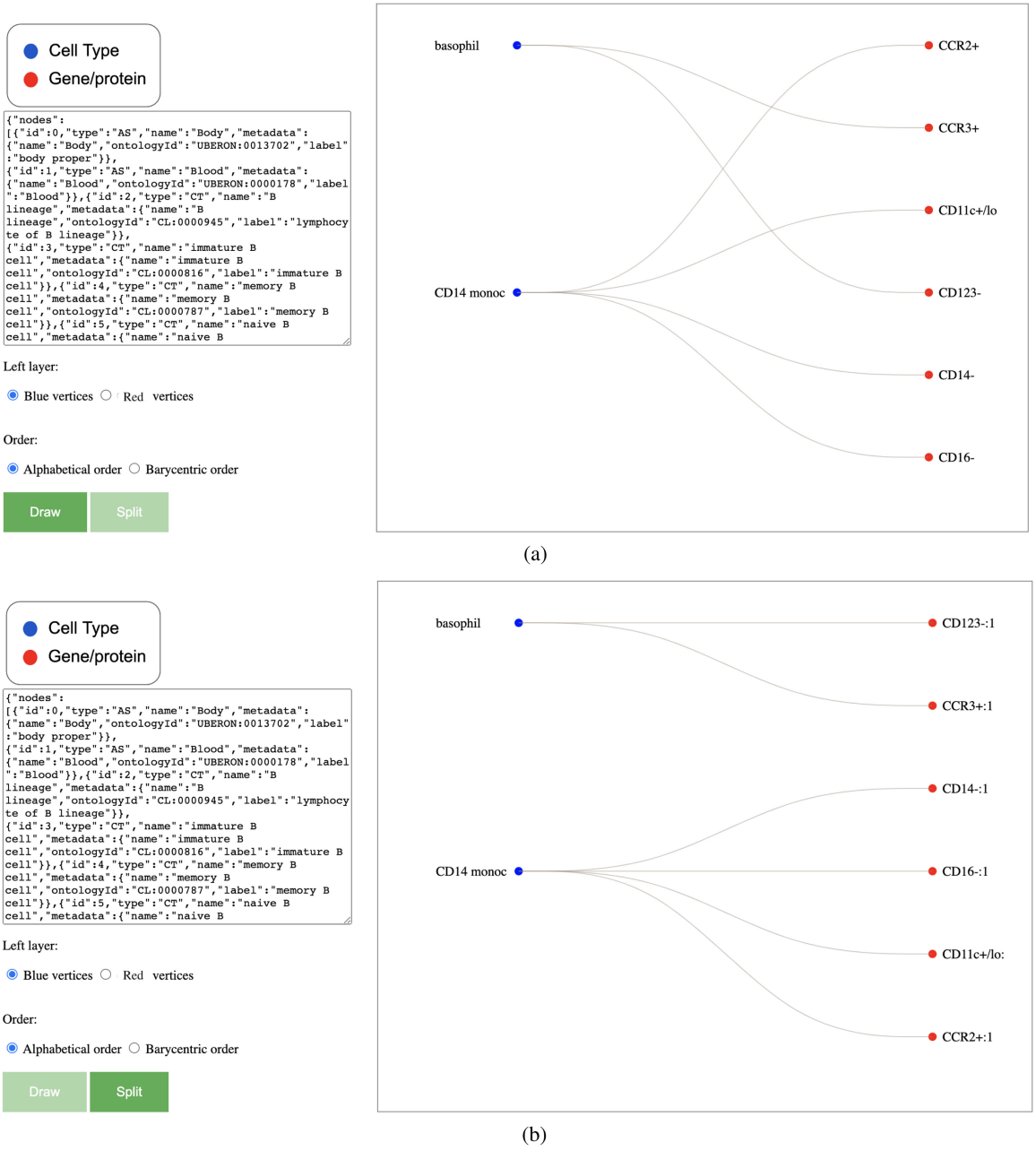

FIGURE 4. User interface. Datasets of the human body[18] can be uploaded, processed with the presented algorithms, and visualized. (a) The input layout on the right side appears after inserting the dataset into the text area and clicking the draw button. (b) The output layout appears on the right side after clicking the split button.

2.5 times larger in total when the gene/protein vertices are fixed.

A second set of experiments was conducted on the same 22 organ graphs to evaluate the crossing minimization heuristics. We set the maximum budget k of splits to 200 and computed the number of remaining crossings after each split. Additionally, we measured wall clock time after each iteration. Figure 5 shows the number of crossings in regards to k for one example graphs. Examples of other graphs, as well as runtime plots can be found in the supplemental material; see Figures 6–11. For both

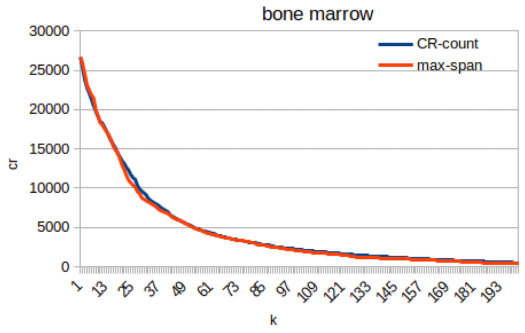

FIGURE 5. *max-span* and *CR-count* heuristic applied to the bone marrow dataset. Both algorithms have nearly identical performance regarding crossing reduction.

algorithms, we observed a similar performance regarding crossing reduction. In some cases, one algorithm slightly outperformed the other, but no clear trend is visible in the data. Intuitively, it seems that in the case of *max-span* the length of edges correlates with the number of crossings. Furthermore, the number of crossings declines steeply at the beginning and for some graphs nearly 30% of crossings are removed by the first ten splits.

The runtime experiments confirmed the asymptotic runtime analysis. *max-span* outperforms *CR-count* on every datasets in regards to total runtime and scalability.

All codes for user interface, algorithms, experimental data, and analysis are available on Github at https://github.com/abureyanahmed/split_graphs.

OPEN PROBLEMS

Minimizing the total number of splits, or the number of split vertices are natural problems. Other variants include minimizing the maximum number of splits per vertex and considering the case, where splits are allowed in both layers. Vertex splits can also be used to improve other quality measures of a 2-layer layout (besides crossings). When visualizing large bipartite graphs, a natural goal is to arrange the vertices so that a small window can capture all the neighbors of a given node, i.e., minimize the maximum distance between the first and last neighbors of a top vertex in the order of the bottom vertices.

ACKNOWLEDGMENTS

This work started at Dagstuhl Seminar 21152 "Multi-Level Graph Representation for Big Data Arising in Science Mapping." The authors would like to thank the organizers and participants for the discussions, particularly C. Raftopoulou.

REFERENCES

1. F. McGee et al., *Visual Analysis of Multilayer Networks*. San Rafael, CA, USA: Morgan & Claypool, 2021.
2. N. Pezzotti, J. D. Fekete, T. Höllt, B. P. F. Lelieveldt, E. Eisemann, and A. Vilanova, "Multiscale visualization and exploration of large bipartite graphs," *Comput. Graphics Forum*, vol. 37, no. 3, pp. 549–560, 2018.
3. P. Eades and N. C. Wormald, "Edge crossings in drawings of bipartite graphs," *Algorithmica*, vol. 11, no. 4, pp. 379–403, 1994.
4. C. Scornavacca, F. Zickmann, and D. H. Huson, "Tanglegrams for rooted phylogenetic trees and networks," *Bioinformatics*, vol. 27, no. 13, pp. i248–i256, 2011.
5. K. Sugiyama, S. Tagawa, and M. Toda, "Methods for visual understanding of hierarchical system structures," *IEEE Trans. Syst., Man, Cybern.*, vol. 11, no. 2, pp. 109–125, Feb. 1981.
6. P. Eades, B. McKay, and N. Wormald, "On an edge crossing problem," in *Proc. 9th Australian Comput. Sci. Conf.*, 1986, pp. 327–334.
7. Y. Kobayashi and H. Tamaki, "A fast and simple subexponential fixed parameter algorithm for one-sided crossing minimization," *Algorithmica*, vol. 72, no. 3, pp. 778–790, 2015.
8. C. Demetrescu and I. Finocchi, "Removing cycles for minimizing crossings," *J. Exp. Algorithmics*, vol. 6, no. 2, pp. 1–39, 2001.
9. W. Didimo, G. Liotta, and F. Montecchiani, "A survey on graph drawing beyond planarity," *ACM Comput. Surveys*, vol. 52, no. 1, pp. 4:1–4:37, 2019.
10. M. Jünger and P. Mutzel, "2-layer straightline crossing minimization: Performance of exact and heuristic algorithms," *J. Graph Algorithms Appl.*, vol. 1, no. 1, pp. 1–25, 1997.
11. P. Eades and C. F. X. de Mendonça N, "Vertex-splitting and tension-free layouts," in *Proc. Symp. Graph Drawing*, 1996, vol. 1027, pp. 202–211.
12. D. Eppstein et al., "On the planar split thickness of graphs," *Algorithmica*, vol. 80, pp. 977–994, 2018.
13. K. Knauer and T. Ueckerdt, "Three ways to cover a graph," *Discrete Math.*, vol. 339, no. 2, pp. 745–758, 2016.
14. A. Liebers, "Planarizing graphs–A survey and annotated bibliography," *J. Graph Algorithms Appl.*, vol. 5, no. 1, pp. 1–74, 2001.
15. L. Faria, C. M. H. de Figueiredo, and C. F. X. Mendonça, "Splitting number is NP-complete," *Discrete Appl. Math.*, vol. 108, no. 1, pp. 65–83, 2001.
16. N. Hartsfield, B. Jackson, and G. Ringel, "The splitting number of the complete graph," *Graphs Combinatorics*, vol. 1, no. 1, pp. 311–329, 1985.

17. B. Jackson and G. Ringel, "The splitting number of complete bipartite graphs," *Archiv der Mathematik*, vol. 42, no. 2, pp. 178–184, 1984.
18. "ASCT+B reporter," 2021. [Online]. Available: https://hubmapconsortium.github.io/ccf-asct-reporter/
19. R. Ahmed, S. G. Kobourov, and M. Kryven, "An FPT algorithm for bipartite vertex splitting," in *Proc. 30th Int. Symp. Graph Drawing Netw. Vis.*, 2022, vol. 13764, pp. 261–268, doi: 10.1007/978-3-031-22203-0_19.
20. A. Chaudhary, D. Z. Chen, X. S. Hu, M. T. Niemier, R. Ravichandran, and K. Whitton, "Fabricatable interconnect and molecular QCA circuits," *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, vol. 26, no. 11, pp. 1978–1991, Nov. 2007.

REYAN AHMED is a visiting assistant professor with Colgate University, Hamilton, NY, 13346, USA. His research interests include graph algorithms, network visualization, and data science. Ahmed received his Ph.D. degree in computer science from the University of Arizona. He is the corresponding author of this article. Contact him at rahmed1@colgate.edu.

PATRIZIO ANGELINI is an associate professor of computer science with John Cabot University, 00165, Rome, Italy. His research interests include graph drawing, network visualization, and graph algorithms. Angelini received his Ph.D. degree in computer science and engineering from Roma Tre University, Rome, Italy. Contact him at pangelini@johncabot.edu.

MICHAEL A. BEKOS is an assistant professor with the Department of Mathematics, University of Ioannina, 451 10, Ioannina, Greece; when the work on this project was conducted he was a postdoc at the University of Tübingen, 72074, Tübingen, Germany. His research interests stem from graph drawing, graph theory, and map labeling. Bekos received his Ph.D. degree from the National Technical University of Athens, Athens, Greece. Contact him at bekos@uoi.gr.

GIUSEPPE DI BATTISTA is a professor with the Department of Engineering, Roma Tre University, 00154, Rome, Italy. His research interests include graph drawing and networking. Contact him at giuseppe.dibattista@uniroma3.it.

MICHAEL KAUFMANN is a professor with the Wilhelm-Schickard-Institute for Informatics, University of Tübingen, 72074, Tübingen, Germany, since 1993. Kaufmann received his Graduate degree from the Universität des Saarlandes, Saarbrücken. Contact him at mk@informatik.uni-tuebingen.de

PHILIPP KINDERMANN is an assistant professor for algorithmics with the University of Trier, 54296, Trier, Germany. His research interests include graph drawing, network visualization, computational geometry, and graph algorithms. Kindermann received his Ph.D. degree in computer science from University of Würzburg, Würzburg, Germany. Contact him at kindermann@uni-trier.de.

STEPHEN KOBOUROV is a professor with the Department of Computer Science, University of Arizona, Tucson, AZ, 85721, USA. His research interests include information visualization, graph theory, and geometric algorithms. Kobourov received his Ph.D. degree from Johns Hopkins University. Contact him at kobourov@cs.arizona.edu.

MARTIN NÖLLENBURG is a professor in the Algorithms and Complexity Group, TU Wien, 1040, Vienna, Austria. His research interests include graph drawing, information visualization, computational geometry, and algorithm engineering. Nöllenburg received his Ph.D. degree in computer science from Karlsruhe Institute of Technology, Germany. Contact him at noellenburg@ac.tuwien.ac.at.

ANTONIOS SYMVONIS is a professor with the School of Applied Mathematical and Physical Sciences, National Technical University of Athens, 10682, Athens, Greece. His research interests include algorithms and complexity, graph drawing, and data visualization. Symvonis received his Ph.D. degree in computer science from the University of Texas at Dallas, USA. Contact him at symvonis@math.ntua.gr.

ANAÏS VILLEDIEU is currently working toward the Ph.D. degree with the Algorithms and Complexity Group, TU Wien, 1040, Vienna, Austria. Her research focuses on graph drawing and information visualization questions, with a focus on vertex splitting problems. Contact her at avilledieu@ac.tuwien.ac.at.

MARKUS WALLINGER is currently working toward the Ph.D. degree in the Algorithms and Complexity Group, TU Wien, 1040, Vienna, Austria. He received his M.S. degree in visual computing from TU Wien, Vienna, Austria. His research focuses on algorithms for information visualization, graph drawing, and linear orders. He is a graduate student member of IEEE. Contact him at mwallinger@ac.tuwien.ac.at.