On the 2-Layer Window Width Minimization Problem*

Michael A. Bekos[1], Henry Förster[2], Michael Kaufmann[2], Stephen Kobourov[3], Myroslav Kryven[3], Axel Kuckuk[2], and Lena Schlipf[2]

[1] Department of Mathematics, University of Ioannina, Ioannina, Greece
bekos@uoi.gr

[2] Department of Computer Science, University of Tübingen, Tübingen, Germany
henry.foerster@uni-tuebingen.de, michael.kaufmann@uni-tuebingen.de, axel.kuckuk@uni-tuebingen.de, lena.schlipf@uni-tuebingen.de

[3] Department of Computer Science, University of Arizona, Tucson, Arizona, USA
kobourov@cs.arizona.edu, kryven@cs.arizona.edu

Abstract. When interacting with a visualization of a bipartite graph, one of the most common tasks requires identifying the neighbors of a given vertex. In interactive visualizations, selecting a vertex of interest usually highlights the edges to its neighbors while hiding/shading the rest of the graph. If the graph is large, the highlighted subgraph may not fit in the display window. This motivates a natural optimization task: find an arrangement of the vertices along two layers that reduces the size of the window needed to see a selected vertex and all its neighbors. We consider two variants of the problem; for one we present an efficient algorithm, while for the other we show NP-hardness and give a 2-approximation.

Keywords: Graph drawing · Bipartite graphs · 2-layer drawings · Window width

1 Introduction

Two-layer networks model relationships between two disjoint sets of entities in various applications. Such networks are naturally modeled by bipartite graphs and are usually visualized with 2-layer drawings, where vertices are drawn as points on two distinct parallel lines ℓ_t and ℓ_b, and edges are straight-line segments [5]. Such drawings occur as components in layered drawings of directed graphs [15] and also as final drawings, e.g., in tanglegrams for phylogenetic trees [1,2,6,14] or in network layouts highlighting relationships between two communities [4,10,13].

A common task in the exploration of such networks is to identify the neighbors of a vertex of interest. A typical approach is to click on this vertex and highlight the edges to its neighbors, while hiding/shading the rest of the graph.

* H. Förster, M. Kaufmann and A. Kuckuk are supported by DFG grant Ka512/18-2. L. Schlipf is supported by the Ministry of Science, Research and the Arts Baden-Württemberg (Germany).

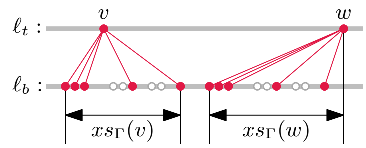


Fig. 1: The x-spans of vertices v and w.

Of course, the highlighted edges should fit in the display window. This motivates a natural optimization task: *find permutations of the vertices that minimize the size of the window needed to see any vertex and all its neighbors.* Related is the problem of minimizing the number of crossings instead, which is an NP-complete problem [5,7,11] and does not always result in easy-to-read drawings.

In applications, the vertex orders cannot always be treated as permutations; the vertices may have specific coordinates in one of the two layers ℓ_t or ℓ_b. For instance, the *ASCT+B Reporter* [8], a tool for displaying anatomical structures, cell types, and biomarkers, exemplifies this issue; by selecting a cell type its related biomarkers are highlighted. Minimizing the actual window width makes the tool easier to use. Note that in this use-case, the window widths of cell types are very important while the corresponding widths for biomarkers are negligible.

Our contribution. Motivated by the discussion above, we study the following problem. The input consists of a bipartite graph $G = (A \cup B, E)$. The output is a 2-layer drawing Γ of G, that is, one in which the vertices in A and B are placed at distinct integer coordinates on two parallel lines ℓ_t and ℓ_b, respectively (w.l.o.g., $\ell_t : y = 1$ and $\ell_b : y = 0$; *top* and *bottom layers*). The objective is to minimize the *window width* of Γ, i.e., the maximum taken over all vertices v in A of the maximum x-distance between all neighbors of v along ℓ_b including the projection of v to ℓ_b. Motivated by common assumptions in layered graph drawing [3,9] we consider two variants, where the x-coordinates of the vertices of either A or B on ℓ_t or ℓ_b, respectively, are fixed. The former is NP-complete (Theorem 3); the latter is efficiently solvable (Theorem 1).

Preliminaries. For a vertex v in a drawing Γ denote by $x_\Gamma(v)$ and $y_\Gamma(v)$ the x- and y-coordinate of v in drawing Γ; when the reference drawing is clear, we simplify the notation to $x(v)$ and $y(v)$. Given a bipartite graph $G = (A \cup B, E)$ with $n_A = |A|$ and $n_B = |B|$, the *x-span $xs_\Gamma(v)$ of a vertex* $v \in A$ in a 2-layer drawing Γ of G is the maximum x-distance of all neighbors of v in B including v itself. To be more formal, $xs_\Gamma(v) = \max_{u,w \in N[v]}\{|x_\Gamma(u) - x_\Gamma(w)|\}$ where $N[v] = \{v\} \cup \{w|(v,w) \in E\}$ is the *closed neighborhood* of v. We define the *window width $ww(\Gamma)$ of the drawing Γ* as the maximum of the x-spans over all vertices in A, that is, $ww(\Gamma) = \max_{v \in A}\{xs_\Gamma(v)\}$, see Fig. 1. In the *2-layer window width minimization problem*, we seek to determine the *window width $ww(G)$ of a graph* G, which is the minimum window width of all of its 2-layer drawings.

2 Window Width Minimization with Bottom Layer Fixed

We present an efficient algorithm to find a 2-layer drawing of minimum window width when the x-coordinates of the vertices of B along ℓ_b are fixed.

Theorem 1. *Given a bipartite graph $G = (A \cup B, E)$ and a function $\xi_B \colon B \to \mathbb{Z}$, there is an $O(n_A \log n_A + |E|)$-time algorithm to determine a 2-layer drawing Γ of G with minimum window width $k^\star$ and $x_\Gamma(b) = \xi_B(b)$ for each $b \in B$.*

Proof. For each vertex $v \in A$ it suffices to focus on its leftmost neighbor $\ell(v)$ in ξ_B and rightmost neighbor $r(v)$ in ξ_B (ignoring intermediate ones). Note that $\ell(v) = r(v)$ is possible. This preprocessing, which can be done in $O(|E|)$ time, allows us to continue with a graph of $O(n_A)$ vertices and edges, called the *critical part* of G. We now determine the x-coordinate of each vertex v in A.

Let k_0 be the maximum x-distance between $\ell(v)$ and $r(v)$ over all vertices v in A and note that k_0 is a lower bound for $k^\star$. We describe an $O(n_A \log n_A)$-time algorithm to compute $k^\star$ and a corresponding solution. In this process, we start by attempting to find a drawing with window width $k = k_0$. If at some point, we conclude that the current value of k is too small, we increase k by 1 and proceed. When the algorithm terminates it will hold that $k = k^\star$.

Let $I(v) = [x(r(v)) - k, x(\ell(v)) + k]$ be the *interval* of $v \in A$; the x-distance of v to $\ell(v)$ and $r(v)$ is at most k if and only if its x-coordinate is in $I(v)$.

We sweep the intervals of the vertices from left-to-right by a vertical sweep line L, which is a data-structure maintaining a set of *active intervals* (i.e., those intersected by L whose vertices in A have not been placed yet) assumed to be *sorted by their right endpoints*. In this process, we distinguish three different types of events: *start*, *placement* and *end*. If during the sweep multiple events occur at the same x-coordinate i we first perform all start events at i, followed by a possible placement event at i before finally performing the end events at i.

Start event. It occurs at the left endpoint i of each interval $I(v)$. Here, the interval $I(v)$ is inserted into L. We add a placement event at i, if there is none.

Placement event at i. We remove the first active interval $I(v)$ from L, set $x(v) := i$ and mark $I(v)$ as *inactive*. If L is not empty, we add a placement event at $i + 1$. Note that placement events always place a vertex, hence there is only a linear number of placement events in total.

End event. It occurs at the right endpoint i of each interval $I(v)$. We check if $I(v)$ is marked as inactive. If this is the case, we proceed. If not, we failed to place v at a position within $I(v)$. We increase k by 1 (i.e., all start events and already placed vertices are moved by -1 and all end events by $+1$ on the x-axis) and replace the already existing placement event with a new placement event at i.

Correctness. We begin with two useful observations. First, once our algorithm failed to place a vertex v within $I(v)$, the partial solution obtained by increasing k by one and shifting all placed vertices one unit to the left is identical to the one that would be obtained by restarting the algorithm with window width $k + 1$. Second, by increasing k the ordering of the start events of the intervals remains the same and the same holds true for the end events. Consequently, the following property holds. Assume that we increased k by 1 at x-coordinate i after failing to place a vertex v with $I(v) = [\ell, r]$. Note that $r = i$ before increasing k and $r = i + 1$ after increasing k. Now let P_i denote the set of vertices that has been placed by our algorithm so far and let S_i denote the set of vertices whose start event occurs at i after increasing k to $k+1$. Then, after handling the end event, for each $p \in P_i$ with interval $I(p) = [\ell_p, r_p]$ it holds for each $s \in S_i$ with $I(s) = [\ell_s, r_s]$ that $r_p \leq r_s$ since $r_p \leq r = i+1$ and $i = \ell_s < r_s$.[1]

To complete the correctness proof, we show that we increase k only if it is necessary. Recall that we increase k if a vertex v cannot be placed within $I(v) = [\ell, r]$. Hence, all x-coordinates of $I(v)$ have been assigned to previously placed vertices. Let $\ell' < \ell$ be the largest x-coordinate our algorithm assigned no vertex from A and let $A_v \subset A$ be the vertices placed in $I'(v) = [\ell' + 1, r]$. We prove that in each solution with window width k, all vertices in A_v have to be placed in $I'(v)$. Assume for a contradiction that there is a vertex $a \in A_v$ that can be placed outside of $I'(v)$ such that its x-span is at most k. To this end, recall that a has x-span at most k if and only if it is placed within $I(a)$. First, a cannot be placed at an x-coordinate greater than x_r, since a has been placed before v by the algorithm, i.e., the right end of $I(a)$ is at an x-coordinate of at most x_r. Second, a cannot be placed at an x-coordinate smaller than $x'_\ell + 1$ as our algorithm would have placed a at coordinate x'_ℓ (or even beforehand) if its interval would have started at an x-coordinate smaller or equal to x'_ℓ; contradiction.

Time complexity. We store the start and end events in two left-to-right sorted lists, while we maintain at most one placement event (with associated x-coordinate). The active intervals are stored in a binary min heap (the keys are the right endpoints). By keeping offset values for start and end events, as well as for the last placed vertex, the performed shifts can be done in $O(n_A)$ time with one additional right-to-left pass. Since L maintains at most $O(n_A)$ intervals the running time is $O(n_A \log n_A)$, after computing the critical part of G in $O(|V| + |E|)$ time. □

Remark 1. The core of the algorithm, given sorted start and end events, can be completed in $O(n_A \log k^\star)$ time since the number of intervals in L is actually bounded by $2k^\star$.

Proof. Consider some x-coordinate i at which there are $2k^\star + 2$ intervals maintained in L. Since there can only be one vertex placed on each integer coordinate, there must be one placed on x-coordinate $i + 2k^\star + 1$, let this be vertex v with interval $I(v) = [\ell, r]$. Note that since this interval is active at i it must hold that $\ell \leq i$. With the definition of $I(v)$ it follows $x(r(v)) \leq \ell + k^\star \leq i + k$. The interval is maximal if $r(v) = \ell(v)$, thus $x(\ell(v)) + k^\star \leq r(v)) + k^\star \leq i + 2k^\star$ which contradicts the placement of v at $i + 2k^\star + 1$. □

Next, we show that a variant of our algorithm can be used to optimize the *maximum edge-length*.

[1] We point out that the latter relation $\ell_s < r_s$ does not hold if $k = 0$, but since we increased k by 1, it holds $k \geq 1$.

Theorem 2. *Given a bipartite graph $G = (A \cup B, E)$ and a function $\xi_B : B \to \mathbb{Z}$, there is an $O(n_A \log n_A + |E|)$-time algorithm to determine a 2-layer drawing Γ of G that minimizes the maximum x-distance $k^\star$ between any vertex in A and any adjacent vertex in B and $x_\Gamma(b) = \xi_B(b)$ for each $b \in B$.*

Proof. As in the proof of Theorem 1, we first identify the critical part of G which has $O(n_A)$ vertices and edges. In the following, we determine the x-coordinate of each vertex v in A such that the maximum x-distance between adjacent vertices, denoted by k, is minimized in the critical part, which implies that it is minimized in G as well. As in the proof of Theorem 1, for a sufficiently large value of k, we define for each vertex $v \in A$ an *interval* $I(v)$ such that v is placed on any x-coordinate in $I(v)$ if and only if its x-distance to any neighbor of v is at most k. More precisely, $I(v) = [x(r(v)) - k, x(\ell(v)) + k]$. We start the algorithm of Theorem 1 with $k = k_0$, where $k_0 := \lceil \frac{k_{\max}}{2} \rceil$ and $k_{\max}$ denotes the maximum x-distance between $\ell(v)$ and $r(v)$ over all vertices v in A (that is, k_0 is the trivial lower bound for $k^\star$). During the algorithm, we might conclude that the current value of k is not sufficient, thus k is increased by 1 before proceeding.

Since the rest of the algorithm of Theorem 1 consists of finding placements of all vertices within their intervals and increasing the intervals if necessary, this part of the algorithm can be completely adopted. Both the correctness and the time complexity of the algorithm follow analogously to Theorem 1. □

3 Window Width Minimization with Top Layer Fixed

In contrast to the positive result from Theorem 1, we prove here that the problem is NP-complete when the order of the vertices A on the top layer ℓ_t is fixed.

Theorem 3. *Given a bipartite graph $G = (A \cup B, E)$, a function $\xi_A : A \to \mathbb{Z}$ and an integer k, it is* NP*-complete to test whether a 2-layer drawing Γ of G exists, such that $ww(\Gamma) = k$ and $x_\Gamma(a) = \xi_A(a)$ for each $a \in A$.*

Proof. Membership in NP is obvious. To prove NP-hardness, we adapt a reduction by Papadimitriou from the Exact-3-Sat problem to the Bandwidth problem [12]. Let φ be an instance of Exact-3-Sat, that is, a Boolean formula with n variables and m clauses (each with 3 different literals). We assume w.l.o.g. that $n \geq 5$ and reduce the problem of determining whether φ is satisfiable to an instance of our problem consisting of a bipartite graph $G = (A \cup B, E)$, a function $\xi_A : A \to \mathbb{Z}$ and the integer $k = 6n + 3$. We first sketch the general idea of the reduction by Papadimitriou and discuss the relation to our construction; for an example illustration see Fig. 2.

Introduction to the reduction. A central concept in the reduction for the Bandwidth problem[2] is a subgraph $\mathcal{H}$ that contains a *literal-vertex* for each possible literal (i.e., for each variable x_i, it contains vertices ℓ_{x_i} and $\ell_{\neg x_i}$) and two additional vertices denoted by M and M'. By fixing the value of the bandwidth, it

[2] Given $k \in \mathbb{N}$ and a graph $G = (V, E)$ the Bandwidth problem asks for an ordering $\prec$ of V so that for each $(u, v) \in E$ there are at most k vertices between u and v in $\prec$.

can be ensured that in any layout of $\mathcal{H}$ exactly n of the literal-vertices appear in a sequence P to the left of M and M' whereas the remaining n literal-vertices appear to the right of M and M' in a sequence Q. The vertices placed in P correspond to the satisfied literals, while the vertices placed in Q correspond to unsatisfied literals. In our reduction, we achieve the same behavior using *block-gadgets* and *$\mathcal{H}$-gadgets* where our *B_2-blocks* correspond to vertices M and M' in Papadimitriou's reduction.

In the reduction for the bandwidth problem, there are $n+m$ consecutive copies of $\mathcal{H}$ that are "synchronized" via the bandwidth restriction. Namely, additional edges ensure that each literal consistently occurs either in every sequence P or in every sequence Q. We achieve the same behavior using the *propagation gadgets*. Papadimitriou associates each of the first n copies of $\mathcal{H}$ with a *variable-gadget* that checks that only one of the literal-vertices corresponding to x and $\neg x$ occurs within Q, namely, as the leftmost vertex in Q. Finally, each of the last m copies of $\mathcal{H}$ is associated with a *clause-gadget* that ensures that at most two literals of a given clause can occur within sequence Q, namely, as the leftmost two vertices. In our construction, we use similar gadgets exploiting this idea.

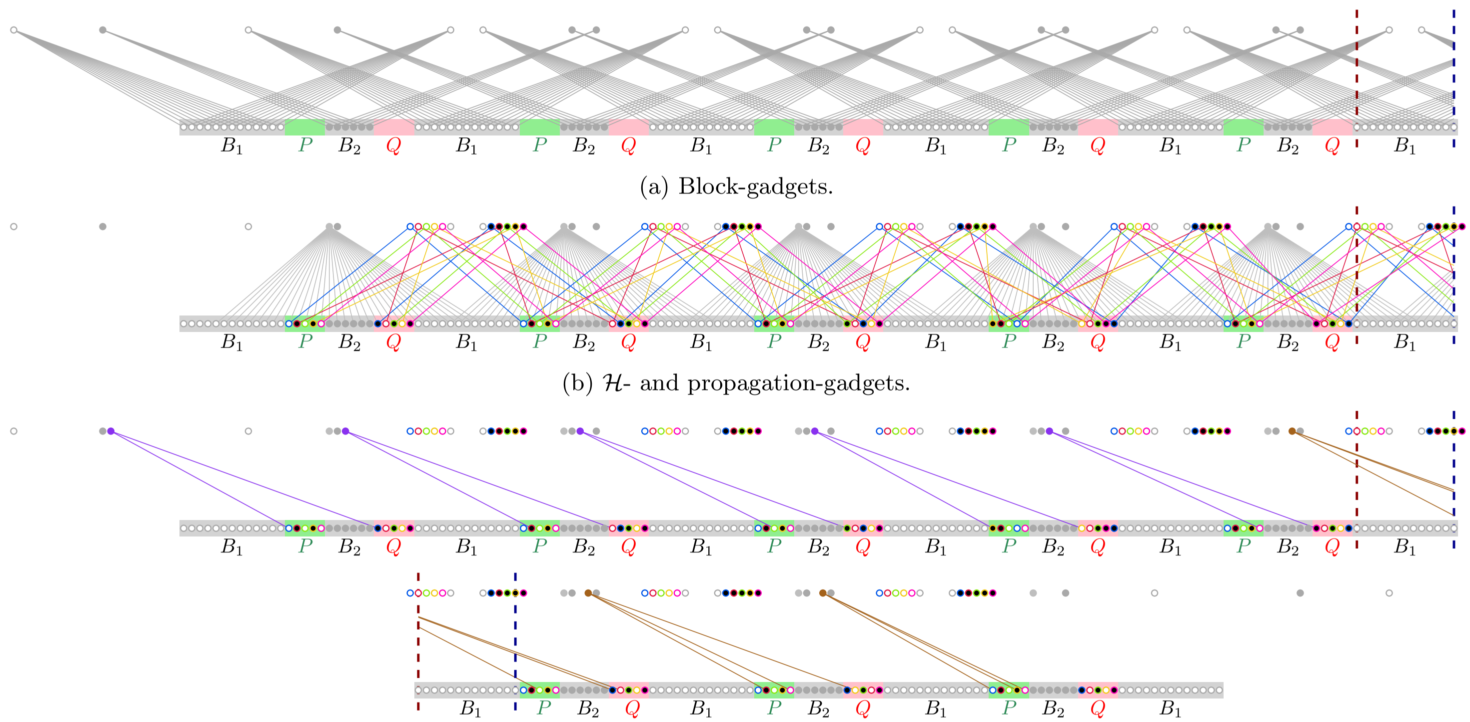
Finally, it is worth remarking that in contrast to the bandwidth problem, in the window width minimization problem vertices in B are restricted to certain positions along ℓ_b by inputs ξ_A and k. With these additional restrictions fixing vertices to certain intervals (e.g., one copy of each literal in each $\mathcal{H}$-gadget) is simplified, however, it also becomes less apparent that the model still allows for enough flexibility to show NP-hardness (as for instance required in the propagation between consecutive $\mathcal{H}$-gadgets).

Next, we provide a description of the gadgets of our construction. The functionality of each gadget is ensured by introducing one or two vertices at appropriate coordinates along ℓ_t. We start by introducing the basic structure of our construction consisting of block- and $\mathcal{H}$-gadgets.

Block-gadget. The purpose of the block-gadget is to fix a certain number β of vertices of B to be consecutive at fixed x-coordinates $i, \dots, i+\beta-1$ so that no other vertex can be placed there; see Fig. 3a. Hence, these β *block vertices* occupy a block of x-coordinates where no other vertex of B may be placed. To achieve this property, we introduce two vertices $a_\ell, a_r \in A$ with $\xi_A(a_\ell) = i - (k - \beta + 1)$ and $\xi_A(a_r) = i + k$ which both are connected to all β block vertices. It is easy to verify that each block vertex has x-distance at most k to both a_ℓ and a_r if and only if it is located inside the interval $[i, i+\beta-1]$ in B (the order of the β vertices inside the interval is free).

We use two types of blocks, namely, one with $\beta_1 = 2n+3$ vertices of B (empty dark gray circles in Fig. 2a) and one with $\beta_2 = n+1$ vertices of B (filled dark gray circles in Fig. 2a). We call the B-vertices of such blocks *B_1- and B_2-blocks*, respectively. Further, we assume that the vertices of a B_1-block are partitioned into three parts B_1^ℓ, B_1^m and B_1^r. Part B_1^m has exactly n vertices, while B_1^ℓ and B_1^r have $\lfloor (n+3)/2 \rfloor$ and $\lceil (n+3)/2) \rceil$ vertices, respectively; see Fig. 3b.

B_1- and B_2-blocks alternate from left-to-right so that in total there are $n+m+1$ B_1-blocks and $n+m$ B_2-blocks. Between a B_1-block and a B_2-block there is a



(a) Block-gadgets.

(b) $\mathcal{H}$- and propagation-gadgets.

(c) Variable- and clause-gadgets.

Fig. 2: Example of our NP-hardness reduction for $\varphi = (\neg x_1 \vee x_2 \vee x_3) \wedge (\neg x_2 \vee x_4 \vee x_5) \wedge (x_1 \vee \neg x_4 \vee x_5)$ with satisfying assignment $x_1 = x_3 = x_5 = \top$ and $x_2 = x_4 = \bot$. Literal-vertices of x_1, x_2, x_3, x_4 and x_5 are colored blue, red, green, yellow and pink, respectively, and filled white or black, if associated with the positive or negative literal, respectively. Subfigures (a) and (b) show only part of the $\mathcal{H}$-gadgets, in subfigure (c) all $\mathcal{H}$-gadgets are shown (split into two lines at the B_1-block separated with dashed lines). Subfigures (b) and (c) also show the vertices of the gadgets introduced in the previous subfigures.

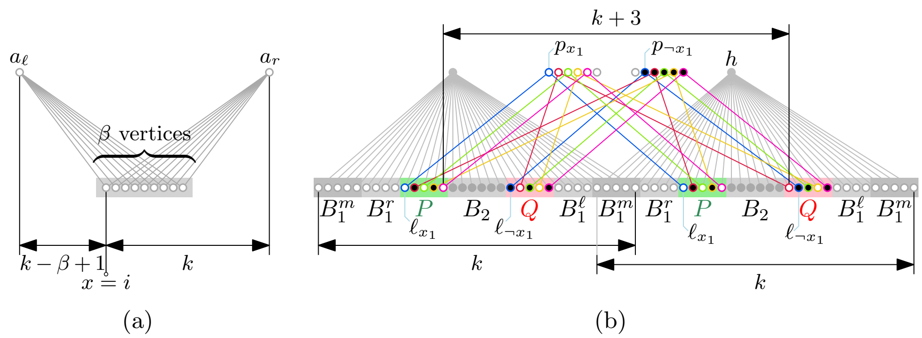


Fig. 3: (a) Block-gadget. (b) $\mathcal{H}$-gadget and propagation-gadget.

Table 1: First and last x-coordinate of the i-th B_1-, P-, B_2-, and Q-block as enumerated from left-to-right starting at 1.

Block Type	**First x-coordinate**	**Last x-coordinate**
B_1-block	$p \cdot (i-1) + 1$	$p \cdot (i-1) + 2n + 3$
P-block	$p \cdot (i-1) + 2n + 4$	$p \cdot (i-1) + 3n + 3$
B_2-block	$p \cdot (i-1) + 3n + 4$	$p \cdot (i-1) + 4n + 4$
Q-block	$p \cdot (i-1) + 4n + 5$	$p \cdot (i-1) + 5n + 4$

P-block while between a B_2-block and a B_1-block there is a *Q-block*. Both P- and Q-blocks are intervals supporting n x-coordinates each and correspond to sequences P and Q in Papadimitriou's reduction. Note that the total number of vertices in a B_1-, P-, B_2- and Q-block is $p = 5n + 4$. We let the first B_1-block start at x-coordinate 1 and obtain intervals for the block types shown in Table 1.

More precisely, we ensure the correct positions of B_1- and B_2-blocks as follows. For the i-th B_1-block, vertex a_ℓ is placed at $\xi_A(a_\ell) = p \cdot (i-2) + n + 4$ while vertex a_r is placed at $\xi_A(a_r) = p \cdot i + n$. In other words, a_ℓ is placed above the $(n+4)$-th vertex (left-to-right) of the previous B_1-block and a_r is placed above the n-th vertex of the next B_1-block. Further, for the i-th B_2-block, vertex a_ℓ is placed at $\xi_A(a_\ell) = p \cdot (i-2) + 3n + 5$ whereas vertex a_r is placed at $\xi_A(a_r) = p \cdot i + 4n + 3$. Intuitively, vertex a_r is placed above the n-th vertex of the next B_2-block and vertex a_ℓ is placed above the second vertex of the previous B_2-block.

$\mathcal{H}$-gadget. The purpose of the $\mathcal{H}$-gadget is to introduce *literal-vertices* for all literals of φ, that is, literals ℓ_{x_i} and $\ell_{\neg x_i}$ for each variable x_i ($2n$ in total; see red, blue, green, yellow and pink vertices in Figs. 2b and 3b). Each $\mathcal{H}$-gadget is associated with a B_2-block b and ensures that each of its $2n$ literal-vertices is placed either in the P-block preceding b (containing all satisfied literals) or in the Q-block succeeding b (containing all unsatisfied literals); Fig. 3b depicts two consecutive copies of the $\mathcal{H}$-gadget; note that there is a shared part of n vertices, denoted by B_1^m.

More precisely, there exists one $\mathcal{H}$-gadget H for each B_2-block b. H contains a vertex h in A that is incident to all vertices of b, to the B_1^m- and B_1^r-vertices of the B_1-block preceding b and to the B_1^ℓ- and B_1^m-vertices of the B_1-block succeeding b, i.e., $\mathcal{H}$-gadgets corresponding to consecutive B_2-blocks share $n = |B_1^m|$ vertices. If H corresponds to the i-th B_2-block, vertex h is placed at $\xi_A(h) = p\cdot(i-1)+3n+4$, that is, above the first B-vertex of its associated B_2-block. Further, h is connected to a literal-vertex for each literal of φ. Since the leftmost vertex of the B_1^m-block preceding b and the rightmost vertex of the B_1^m-block succeeding b are at distance k, all literal-vertices connected to h must be placed between these two blocks. The only available positions in this range are covered by the P-block preceding b and the Q-block succeeding b. Note that in the following, no further edges incident to vertices in a B_1-block are introduced, i.e., the vertex-order inside a B_1-block is only restricted by h-vertices. Finally, observe that the h-vertices have x-span k if the vertices in B_1^ℓ precede (left-to-right) those in B_1^m, which precede those in B_1^r.

In the following, we assume literal-vertices in P-blocks and Q-blocks to correspond to satisfied and unsatisfied literals, respectively. Next, we ensure consistency.

Propagation-gadget. The propagation-gadgets (see red, blue, green, yellow and pink vertices and edges in Figs. 2b and 3b) ensure consistency, that is, literals in P-blocks are satisfied, while literals in Q-blocks are unsatisfied in φ. Namely, the propagation gadget for x_i ensures that the literal-vertex $\ell_\lambda \in \{\ell_{x_i}, \ell_{\neg x_i}\}$ occurring in the P-block of an $\mathcal{H}$-gadget H_1 will also occur in the P-block of the next $\mathcal{H}$-gadget H_2 in their left-to-right order. Since all vertices from P-blocks are propagated, literal-vertices in the Q-blocks are also propagated from H_1 to H_2. Note that literal-vertices do not necessarily have the same order in H_1 and H_2.

More formally, for each B_1-block b and for each variable x_i there is a copy of the propagation-gadget containing two *propagation-vertices* p_{x_i} and $p_{\neg x_i}$. Let H_1 and H_2 be the two (consecutive) $\mathcal{H}$-gadgets incident to the B_1^m-vertices of b. Then, vertex p_{x_i} is connected to the literal-vertices ℓ_{x_i} of H_1 and H_2 while $p_{\neg x_i}$ is connected to the literal-vertices $\ell_{\neg x_i}$ of H_1 and H_2. If b is the j-th B_1-block, we set $\xi_A(p_{x_i}) = p \cdot (j-1) + (i-1)$ and $\xi_A(p_{\neg x_i}) = p \cdot (j-1) + (n+4) + i$, i.e., all propagation-vertices with positive literals are to the left of the a_r-vertex above b while all propagation-vertices with negative literals are to the right of the a_ℓ-vertex above b. Note that p_{x_1} is above the last vertex in the Q-block preceding b while $p_{\neg x_n}$ is above the first vertex in the P-block succeeding b; the remaining literal-vertices are placed on unique x-coordinates above b. Since the distance between the leftmost literal-vertex in the P-block of H_1 and the rightmost literal-vertex in the P-block of H_2 is $k - n + 1$, we can reorder all literal-vertices freely in the P-blocks of H_1 and H_2; the same holds for the corresponding Q-blocks. On the other hand, the rightmost literal-vertex ℓ_λ of the P-block of H_1 cannot occur in the Q-block of H_2 as otherwise their connecting vertex p_λ has x-span at least $k+3$; see Fig. 3b. As already mentioned above, since all literals from the P-block are propagated from H_1 to H_2, all literals from the Q-block are propagated as well.

Now each literal is either consistently satisfied (in P-blocks) or unsatisfied (in Q-blocks). It remains to encode the logic of φ with variable- and clause-gadgets.

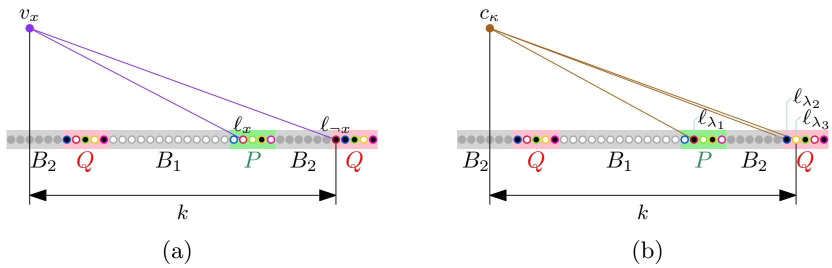


Fig. 4: (a) Variable-gadget. (b) Clause-gadget.

Variable-gadget. The variable-gadget for variable x_i ensures that only one of the literal-vertices ℓ_{x_i} and $\ell_{\neg x_i}$ can be placed within Q-blocks. Since these gadgets guarantee that at most one literal for each variable is false, it is only possible to place all $2n$ literals if exactly one literal per variable is true while the other is false. Hence, each variable is either true or false consistently in all $\mathcal{H}$-gadgets.

More precisely, the first n (in left-to-right-order) $\mathcal{H}$-gadgets are augmented with a variable-gadget. Namely, each variable gadget is associated with a unique variable x and ensures that one of the literals x and $\neg x$ must be true. The variable gadget associated with $\mathcal{H}$-gadget H consists of both literal-vertices ℓ_x and $\ell_{\neg x}$ of H and an additional *variable-vertex* v_x connected to ℓ_x and $\ell_{\neg x}$; see Fig. 4a and purple vertices and edges in Fig. 2c. We set the x-coordinate of v_x so that it is at distance k to the left of the leftmost vertex in the Q-block of H, i.e., if H is the i-th H-gadget we have $\xi_A(v_x) = p \cdot (i-2) + 3n + 6$. As a result, v_x is placed above the third vertex of the B_2-block preceding the B_2-block of H. Clearly, the x-span of v_x is at most k if at most one of ℓ_x and $\ell_{\neg x}$ is in the Q-block of H. As mentioned above, since for each variable the variable gadget guarantees this property, it is only possible to place all $2n$ literals if exactly one literal per variable is true while the other is false. Thus, each variable is either consistently true or consistently false.

Clause-gadget. There are m clause-gadgets associated with the last m copies of $\mathcal{H}$-gadgets. The clause-gadget for a clause $\kappa = (\lambda_1 \vee \lambda_2 \vee \lambda_3)$ ensures that at most two of the literal-vertices ℓ_{λ_1}, ℓ_{λ_2} and ℓ_{λ_3} can be placed within Q-blocks. At least one literal must be placed inside the P-blocks and thus, κ contains at least one satisfied literal.

To this end, the clause gadget for clause $\kappa = (\lambda_1 \vee \lambda_2 \vee \lambda_3)$ consists of a vertex $c_\kappa \in A$ connected to the three literal-vertices $\ell_{\lambda_1}, \ell_{\lambda_2}, \ell_{\lambda_3}$; see Fig. 4b and brown vertices and edges in Fig. 2c. We assign the x-coordinate of c_κ so that it is at distance $k-1$ to the left of the leftmost vertex in the Q-block of H, i.e., if H is the i-th H-gadget, then $\xi_A(c_\kappa) = p(i-2) + 3n + 7$. Hence, c_κ is placed above the fourth vertex of the B_2-block preceding H. Since the distance between the second vertex in the Q-block of H and c_κ is k, the x-span of c_κ is at most k if at most two of $\ell_{\lambda_1}, \ell_{\lambda_2}, \ell_{\lambda_3}$ are in the Q-block, i.e., one of $\lambda_1, \lambda_2, \lambda_3$ is true.

Table 2: Placements of vertices in A above the j-th vertex within B_1- and B_2-blocks. If a vertex x of A is above the j-th vertex of the i-th B_1-block, $\xi_A(x) = p \cdot (i-1) + j$, while $\xi_A(x) = p \cdot (i-1) + 3n + 3 + j$ if x is above the j-th vertex of the i-th B_2-block.

B_1-**block**	j	B_2-**block**
$p_{x_{j+1}}$ of prop.-gadget	1	h of associated $\mathcal{H}$-gadget
	2	a_ℓ of next B_2-block
	3	v_x associated with the next $\mathcal{H}$-gadget
	4	c_κ associated with the next $\mathcal{H}$-gadget
	$[5, n-1]$	———
a_r of previous B_1-block	n	a_r of previous B_2-block
———	$[n+1, n+3]$	———
a_ℓ of next B_1-block	$n+4$	
$p_{\neg x_{j-(n-4)}}$ of prop.-gadget	$[n+5, 2n+3]$	

Polynomial time of the reduction and equivalence. The construction can clearly be done in $O(n \cdot (n+m))$ time. Next, we prove that no two vertices of A share the same x-coordinate, i.e., ξ_A is injective.

Recall that we have placed vertices in A only on coordinates that are covered by B_1- and B_2-blocks (or are located to the left of the first B_1-block or to the right of the last B_1-block); see Fig. 2. Table 2 summarizes the positioning described in the construction. The two exceptions to this are vertices p_{x_1} and $p_{\neg x_n}$ of propagation gadgets which are located above the last vertex of P-blocks and above the first vertex of Q-blocks, respectively. For $n \geq 5$ (as assumed at the beginning) indeed no x-coordinate is assigned twice by ξ_A.

It remains to prove that φ is satisfiable if and only if there is a drawing Γ with $ww(\Gamma) \leq k$ and $x_\Gamma(a) = \xi_A(a)$ for $a \in A$. First, assume that φ is satisfiable. We can construct a drawing with window width at most k by placing all satisfied literals in the P-block and each unsatisfied literal in the Q-block of each $\mathcal{H}$-gadget. The literal-vertices are sorted so that the unsatisfied literals in a variable- or clause-gadget are the leftmost ones in the corresponding Q-block. Second, assume that there is a drawing Γ with $ww(\Gamma) \leq k$ and $x_\Gamma(a) = \xi_A(a)$ for $a \in A$. As discussed above, each variable is either true or false while each clause contains a satisfied literal. Thus, a satisfying truth assignment for φ can be read from any P-block of Γ. □

Next, we prove that our algorithm from Theorem 2 can be used for a 2-approximation algorithm for the window width minimization problem with fixed top layer.

Theorem 4. *Given a bipartite graph $G = (A \cup B, E)$ and a function $\xi_A \colon A \to \mathbb{Z}$, there is an $O(n_B \log n_B + |E|)$-time 2-approximation algorithm for computing the minimum value $k^\star$ such that there is a 2-layer drawing Γ of G with $ww(\Gamma) = k^\star$ and $x_\Gamma(a) = \xi_A(a)$ for each $a \in A$ that also produces a corresponding solution.*

Proof. The idea is to use the optimization algorithm from the proof of Theorem 2, where the vertex sets A and B are interchanged, to compute a placement of the vertices of B in time $O(n_B \log n_B + |E|)$, so that the length k' of the longest edge is minimized. Let k denote the window width of the obtained 2-layer drawing Γ.

Let $k^\star$ be the minimum window width of a 2-layer drawing $\Gamma^\star$ of G with $x_{\Gamma^\star}(a) = \xi_A(a)$. We show that $k \leq 2k^\star$. First, recall that the longest edge in Γ has length k'. Thus $k \leq 2k'$ as in the worst case, a vertex $v \in A$ has distance k' to both its leftmost and its rightmost neighbor. Second, consider the 2-layer drawing $\Gamma^\star$. Since the longest edge in $\Gamma^\star$ has length at most $k^\star$ and k' is chosen optimally, we obtain $k' \leq k^\star$. Combining both arguments, we obtain $k \leq 2k' \leq 2k^\star$.

Finally, drawing Γ is a corresponding solution as stated in the theorem. □

4 Open Problems

We conclude with some open problems. First, the case where all vertices can be freely positioned along ℓ_t and ℓ_b may be investigated in future work. Second, the setting of Theorem 3 with the additional constraint that the vertices in A are degree-restricted, is of interest. Third, other optimization criteria could be useful in practice. For instance, one may try to minimize the average x-span while potentially also weighting spans of important vertices differently.

References

1. Buchin, K., Buchin, M., Byrka, J., Nöllenburg, M., Okamoto, Y., Silveira, R., Wolff, A.: Drawing (complete) binary tanglegrams. Algorithmica **62**(1–2), 309–332 (2012). https://doi.org/10.1007/s00453-010-9456-3
2. Czabarka, É., Székely, L.A., Wagner, S.G.: A tanglegram Kuratowski theorem. J. Graph Theory **90**(2), 111–122 (2019). https://doi.org/10.1002/jgt.22370
3. Di Battista, G., Eades, P., Tamassia, R., Tollis, I.G.: Graph Drawing: Algorithms for the Visualization of Graphs. Prentice-Hall (1999)
4. Dumas, M., McGuffin, M.J., Robert, J.M., Willig, M.C.: Optimizing a radial layout of bipartite graphs for a tool visualizing security alerts. In: van Kreveld, M., Speckmann, B. (eds.) Graph Drawing (GD'11). LNCS, vol. 7034, pp. 203–214. Springer-Verlag (2012). https://doi.org/10.1007/978-3-642-25878-7_20
5. Eades, P., Wormald, N.C.: Edge crossings in drawings of bipartite graphs. Algorithmica **11**(4), 379–403 (1994). https://doi.org/10.1007/BF01187020
6. Fernau, H., Kaufmann, M., Poths, M.: Comparing trees via crossing minimization. J. Comput. Syst. Sci. **76**(7), 593–608 (2010). https://doi.org/10.1016/j.jcss.2009.10.014
7. Garey, M.R., Johnson, D.S.: Crossing number is NP-complete. SIAM Journal on Algebraic Discrete Methods **4**(3), 312–316 (1983)
8. HuBMAP Consortium: CCF ASCT+B Reporter, `https://hubmapconsortium.github.io/ccf-asct-reporter/`
9. Kaufmann, M., Wagner, D. (eds.): Drawing Graphs, Methods and Models, LNCS, vol. 2025. Springer (2001). https://doi.org/10.1007/3-540-44969-8

10. Meulemans, W., Schulz, A.: A tale of two communities: Assessing homophily in node-link diagrams. In: Di Giacomo, E., Lubiw, A. (eds.) Graph Drawing (GD'15). LNCS, vol. 9411, pp. 489–501. Springer International Publishing (2015). https://doi.org/10.1007/978-3-319-27261-0 40
11. Muñoz, X., Unger, W., Vrto, I.: One sided crossing minimization is NP-hard for sparse graphs. In: Mutzel, P., Jünger, M., Leipert, S. (eds.) Graph Drawing. LNCS, vol. 2265, pp. 115–123. Springer (2001). https://doi.org/10.1007/3-540-45848-4_10
12. Papadimitriou, C.H.: The NP-completeness of the bandwidth minimization problem. Computing **16**(3), 263–270 (1976). https://doi.org/10.1007/BF02280884
13. Pezzotti, N., Fekete, J.D., Höllt, T., Lelieveldt, B.P.F., Eisemann, E., Vilanova, A.: Multiscale visualization and exploration of large bipartite graphs. Computer Graphics Forum **37**(3), 549–560 (2018). https://doi.org/10.1111/cgf.13441
14. Scornavacca, C., Zickmann, F., Huson, D.H.: Tanglegrams for rooted phylogenetic trees and networks. Bioinformatics **27**(13), i248–i256 (2011). https://doi.org/10.1093/bioinformatics/btr210
15. Sugiyama, K., Tagawa, S., Toda, M.: Methods for visual understanding of hierarchical system structures. IEEE Trans. Syst. Man Cybern. **11**(2), 109–125 (1981). https://doi.org/10.1109/TSMC.1981.4308636