An Efficient Fixed Parameter Tractable Algorithm for 1-Sided Crossing Minimization*

Vida Dujmović and Sue Whitesides

School of Computer Science, McGill University, 3480 University St.,
Montreal, Québec, Canada H3A 2A7,
{vida, sue}@cs.mcgill.ca

Abstract. We give an $\mathcal{O}(\phi^k \cdot n^2)$ fixed parameter tractable algorithm for the 1-Sided Crossing Minimization problem. The constant ϕ in the running time is the golden ratio $\phi = \frac{1+\sqrt{5}}{2} \approx 1.618$. The constant k is the parameter of the problem: the number of allowed edge crossings.

1 Introduction

A common method for drawing directed acyclic graphs is to produce *layered drawings* or *hierarchical drawings* as introduced by Tomii *et al.* [22], Carpano [1], and Sugiyama *et al.* [21]. In these drawings, the vertices are arranged on two or more "layers", i.e. on parallel horizontal lines, and edges are drawn straight between vertices on adjacent layers. Edges between vertices on the same layer are not permitted, and no point between layers may lie on more than two edges. Layouts of this kind have applications, for example, in visualization [5], in DNA mapping [25], and in row-based VLSI layout [17].

The readability of layered drawings is believed to depend crucially on the number of edge crossings. Once vertices have been assigned to layers, this number is determined by the orderings of the vertices within the layers. Unfortunately, the problem of choosing vertex orderings that minimize the number of edge crossings in layered drawings is in fact an NP-complete problem [14] even if there are only two layers [12]. The problem of choosing vertex orderings that minimize the number of edges whose removal leaves the graph planar is also NP-complete, even for two layers [11].

Most techniques for producing layered drawings first assign vertices to layers (sometimes this is determined by the context), and then do a *layer-by-layer sweep*. A permutation π_1 for the vertices in the top layer L_1 is chosen and fixed. Then for each succeeding layer L_i, a permutation π_i is sought that keeps to a minimum the number of edge crossings among the edges between L_{i-1} and L_i.

A key step in this method is to minimize crossings between two adjacent layers when the ordering on one layer is fixed. This problem is called 1-Sided Crossing Minimization. Unfortunately, the 1-Sided Crossing Minimization problem is also NP-complete [12]. The problem is NP-complete even for graphs with only degree-1 vertices in the fixed layer and vertices of degree at most 4 in the other layer [18], i.e. for a forest of 4-stars.

* Research supported by FCAR and NSERC.

The 1-Sided Crossing Minimization problem is the focus of this paper. Many heuristics have been proposed (e.g. [2, 7, 10, 12, 21, 23, 24]). Jünger and Mutzel [16] gave an exact integer linear programming algorithm for the 1-Sided Crossing Minimization problem. They also surveyed heuristics and made performance comparisons with optimal solutions generated by their methods. They reported that the *iterated barycentre* method of Sugiyama *et al.* [21] performs best in practice. However, from a theoretical point of view the *median heuristic* of Eades and Wormald [12] is a linear 3-approximation algorithm, whereas the barycentre heuristic is a $\Theta(\sqrt{n})$-approximation algorithm. The most recent heuristic based on the computation of feedback arc sets and experimental results has been reported in [4].

When only a small number, k, of edge crossings is acceptable, then an algorithm for 1-Sided Crossing Minimization whose running time is exponential in k but polynomial in the size of the graph may be useful. The theory of parameterized complexity [6] addresses complexity issues of this nature, in which a problem is specified in terms of one or more parameters. Such a problem with input size n and parameter size k is *fixed parameter tractable*, or in the class *FPT*, if there is an algorithm to solve the problem in $f(k) \cdot n^{\alpha}$ time, where α is a constant independent of k and n, and f is an arbitrary function dependent only on parameter k. A problem in FPT is thus solvable in polynomial time for any fixed value of k.

Instances of 1-Sided Crossing Minimization problem for dense graphs are of little interest either from practical or theoretical point of view. From the practical point of view, an instance with a high number of crossings in its optimal drawing, is hardly worthwhile optimizing since the resulting drawing will be unreadable anyway. From the theoretical point of view, not only is the problem still NP-complete for very sparse graphs [18], but in addition Eades and Wormald [12] proved that the ratio of crossings in an arbitrary ordering to crossings in an optimal ordering approaches 1 if graphs become dense.

The more general h-Layer Crossing Minimization problem as well as the related h-Layer Planarization problem have been studied from the fixed parameter tractability point of view in [9]. Here h represents the number of layers and planarization means to remove some number k of edges so that the remaining graph can be drawn without crossings (see [11]). According to [9], bounded pathwidth techniques prove both these general problems (which include 1-Sided Crossing Minimization) are in the class FPT. Unfortunately, the pathwidth-based approach is only of theoretical interest, since the running time of the algorithms is $O(2^{32(h+2k)^3} n)$. In [8], other FPT techniques are used to derive an $O(k \cdot 6^k + |G|)$ time algorithm for 2-Layer Planarization of a graph G, and an $O(3^k \cdot |G|)$ time algorithm for 1-Layer Planarization.

The general crossing minimization problem, where vertices are not restricted to lie on parallel lines nor are edges restricted to be straight, has been studied extensively by mathematicians and computer scientists. Although the general problem is not particularly relevant to its layered counterpart, it is worth noting that Grohe [15] has shown recently that the general crossing minimization problem is in FPT. Since, the approach relies on deep structure theorems from the Robertson-Seymour graph minors project, the FPT result of Grohe does not yield a practical algorithm.

In this paper, we give an algorithm for 1-Sided Crossing Minimization that runs in $\mathcal{O}(\phi^k|L_2|^2 + |L_1||L_2|)$ time, where L_2 is the set of vertices on the free layer where the permutation π_2 of vertices is to be chosen, L_1 is the set of vertices on the fixed layer where the permutation π_1 of vertices is fixed, and the constant ϕ is the golden ratio. The algorithm is based on the FTP technique called *Bounded Search Tree*. This technique relies on exploring some search space, and then proving that its size depends only upon the parameter k. The search space thus becomes *constant size* and the algorithm is then polynomial time for each fixed k.

The remainder of this paper is organized as follows. After definitions and preliminary results in Section 2, we study properties of optimal drawings in Section 3. Our algorithm for the 1-Sided Crossing Minimization problem is then given in Section 4. Section 5 concludes.

2 Problem Statement, Notation, and Some Facts

A graph $G = (V, E)$ with vertex set V and edge set $E \subseteq V \times V$ is called *bipartite* if there is a partition of V into two disjoint non-empty sets L_1 and L_2 such that $V = L_1 \cup L_2$ and $E \subseteq L_1 \times L_2$. The number of vertices and edges of G are respectively denoted by $n = |V|$ and $m = |E|$. Let d_v denote the degree of a vertex v. We assume $d_v \geq 1$.

In a *2-layer drawing* of a graph $G = (L_1, L_2; E)$, the vertices in L_1 and L_2 are positioned on two distinct parallel lines (layers), and the edges are drawn straight. Since edges are not allowed within a layer, G must be bipartite. Let L_1 denote the top, *fixed* layer, whose vertex ordering π_1 is fixed. Let L_2 denote the bottom, *free* layer, whose vertices are free to be permuted. Figure 1 illustrates this terminology.

We study the following problem:
Problem:1-Sided Crossing Minimization
Instance: a bipartite graph $G = (L_1, L_2; E)$, an integer k, and a fixed ordering π_1 for the vertex set L_1 on the top layer.
Question: Is there a 2-layer drawing of G that respects π_1 and that has at most k crossings?

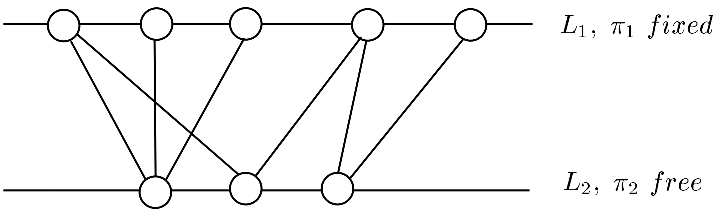

Fig. 1. A 2-layer drawing. The ordering π_1 of L_1 is *fixed*. Vertices of L_2 are *free*.

From now on, we assume that input graphs are bipartite, with minimum degree at least 1, and that an ordering π_1 has been specified for the top layer. We do not consider multiple edges, although these are easy to handle. For some details on that refer to Section 4.3.

Let $\langle G, \pi_1, k\rangle$ denote an instance of the 1-Sided Crossing Minimization problem, and let (G, π_1, π_2) denote a combinatorial representation of a 2-layer drawing of G, with π_1 and π_2 giving the permutations for the vertices on layers L_1 and L_2, respectively. Let the number of crossings in the drawing (G, π_1, π_2) be denoted by $cr(G, \pi_1, \pi_2)$, and let the minimum possible number of crossings subject to the vertices of L_1 being ordered by π_1 be denoted by $\mathrm{opt}(G, \pi_1)$. Note that $\mathrm{opt}(G, \pi_1) = \min_{\pi_2}\{cr(G, \pi_1, \pi_2)\ \}$, where π_2 ranges over all permutations for L_2. Let $v < w$ denote an ordered pair of vertices on the same layer, and let v, w denote an unordered pair of vertices. Sometimes it is convenient to denote unordered pairs of vertices by (v, w), which is also used to denote an edge. The meaning will be clear from the context. Throughout the paper, the term "pair" refers to a pair of *distinct* objects. The following two simple observations reinforce the important fact that the 1-Sided Crossing Minimization problem is combinatorial in nature.

Fact 1 *[5] Two edges (v, v') and (w, w'), where $v, w \in L_2$ and $v', w' \in L_1$, cross in a 2-layer drawing if and only if $v < w$ and $w' < v'$, or $w < v$ and $v' < w'$.*

Fact 2 *[5] For vertices v and w in the free layer L_2, the number of crossings of the edges incident to v with the edges incident to w is completely determined by the relative ordering of v and w.*

Having pointed out that the nature of the problem fundamentally concerns orderings, the next definition and fact relate the orderings of pairs to the total number of edge crossings any drawing, the goal of our optimization. In fact, from pairs alone, we get a lower and upper bound for $\mathrm{opt}(G, \pi_1)$.

Definition 1. *Consider a problem instance $\langle G, \pi_1, k\rangle$, and let v and w be vertices in L_2. The* crossing number *c_{vw} is the number of crossings that edges incident with v make with edges incident with w in drawings having $v < w$; the* crossing number *c_{wv} is for $w < v$.*

Fact 3 *[5] The total number of crossings in a 2-layer drawing (G, π_1, π_2) is:*

$$cr(G, \pi_1, \pi_2) = \sum_{\forall v < w \in \pi_2} c_{vw}, \tag{1}$$

where the summation is over all ordered pairs $v < w$ of elements of π_2; furthermore,

$$\sum_{v,w \in L_2} \min(c_{vw}, c_{wv}) \leq \mathrm{opt}(G, \pi_1) \leq \sum_{v,w \in L_2} \max(c_{vw}, c_{wv}), \tag{2}$$

where the summations are over all unordered pairs v, w of vertices of L_2.

Let $\mathrm{lb}(G, \pi_1)$ denote the lower bound $\sum_{v,w\in L_2} \min(c_{vw}, c_{wv})$. As pointed out in the introduction, Eades and Wormald [12] showed that $\mathrm{opt}(G, \pi_1) \leq 3\,\mathrm{lb}(G, \pi_1)$. Recently, Nagamochi [19] improved that to $\mathrm{opt}(G, \pi_1) \leq 1.47\,\mathrm{lb}(G, \pi_1)$.

3 Properties of Optimal Drawings

In this section, we establish a property of optimal drawings (G, π_1, π_{opt}) (see Lemma 1) that will be fundamental for our algorithm in the next section.

For each vertex v in L_2, let l_v denote the leftmost neighbor of v in L_1, and let r_v denote the rightmost neighbor of v in L_1. Note that if $v \in L_2$ has degree 1, then $l_v = r_v$.

Now consider two vertices v and w in L_2. We say that v and w are a *suited pair* if $r_v \leq l_w$, or if $r_w \leq l_v$; otherwise we call the pair *unsuited*. For example, in Fig. 2 v, u is a suited pair and so is pair w, u; pair v, w is not suited. If v and w each have degree 1 and have the same neighbor in L_1 (i.e. $l_v{=}r_v{=}l_w{=}r_w$), we say that v and w are a *trivial* suited pair. The following fact, which is an immediate consequence of Fact 1 and the definition of unsuited pair, indicates the importance of the notion of suitable pairs.

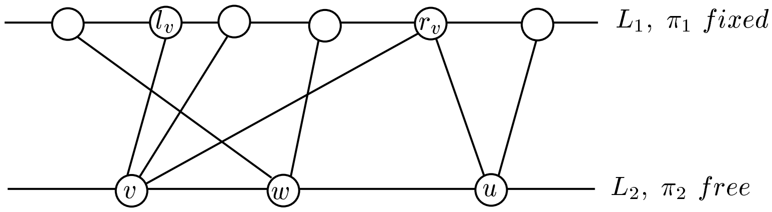

Fig. 2. Pair v, w is unsuited, while v, u and w, u are suited pairs.

Fact 4 *A pair of vertices $v, w \in L_2$ is unsuited if and only if $c_{vw} \geq 1$ and $c_{wv} \geq 1$.*

On the other hand, the edges of a suited pair v, w do not cross if v and w appear in their *natural ordering* in π_2, i.e., if $v < w$ when $r_v \leq l_w$, and $w < v$ when $r_w \leq l_v$. If v, w is a trivial suited pair, then $c_{vw} = 0$ and $c_{wv} = 0$, and we say that both $v < w$ and $w < v$ are natural orderings for the pair v, w.

The notion of natural ordering leads to a useful fact for our algorithm.

Fact 5 *Suppose v, w is a suited pair for π_1 with natural ordering $v < w$. Then for any π_2, the drawing (G, π_1, π_2) satisfies: (i) $c_{vw}{=}0$; (ii) if $r_v \neq l_w$, then $c_{wv}{=} d_v \cdot d_w$; (iii) if $r_v = l_w$, then $c_{wv}{=} (d_v \cdot d_w) - 1$; and finally, (iv) unless v and w are a trivial suited pair, $c_{wv} > 0$.*

Note that natural ordering is only defined for pairs of suited vertices. For general pairs, we say that $v < w$ is the *preferred ordering* for a pair v, w if $c_{vw} < c_{wv}$. Thus the natural ordering is the preferred ordering for non-trivial suited pairs. The following Lemma is the basis for our algorithm.

Lemma 1. *For fixed π_1, let $\Gamma_{opt} = (G, \pi_1, \pi_{opt})$ be a drawing with the minimum possible number of crossings. Then all suited pairs appear in π_{opt} in their natural ordering.*

To prove Lemma 1 the following lemma will be useful.

Lemma 2. *For $1 \leq i \leq |L_2|$, let v_i denote the vertex in the i^{th} position in π_2 of some drawing (G, π_1, π_2) with π_1 fixed. Moving any vertex $v_i \in L_2$ from its starting position i across the t consecutive vertices $v_{i+1}, v_{i+2}, \ldots, v_{i+t}$ to the right creates a new drawing (G, π_1, π_2') with:*

$$cr(G, \pi_1, \pi_2') = cr(G, \pi_1, \pi_2) \ + \ \sum_{j=1}^{t} (c_{v_{i+j} v_i} \ - \ c_{v_i v_{i+j}}). \tag{3}$$

Similarly, if v_i is moved to the left over t consecutive vertices, then the above summation is from $j = -1$ to $j = -t$ and the sign in front of the summation is "$-$".

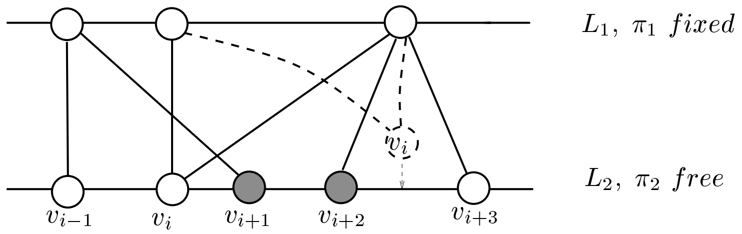

Fig. 3. Illustration for the proof of Lemma 2.

Proof: Assume, without loss of generality, that vertex v_i moves to the right across t consecutive vertices in (G, π_1, π_2). This creates a new drawing (G, π_1, π_2'). For instance, in Figure 3, v_i moves from its starting position over $t = 2$ (shaded) vertices to its final position. Dotted lines depict v_i and its incident edges as they travel across 2 shaded vertices to the new position of v_i, between v_{i+2} and v_{i+3}. The only pairs of vertices in (G, π_1, π_2) whose relative ordering changes in (G, π_1, π_2') are the pairs v_i, v_j for $i + 1 \leq j \leq i + t$. Hence by Fact 2, these are the only pairs whose crossing number might change. In particular, the crossing number for a pair v_i, v_j for j in the range $[i + 1, i + t]$ changes from $c_{v_i v_j}$ to $c_{v_j v_i}$. Substituting these changes into equation (1) of Fact 3 gives equation (3) above. □

Now we give the proof of Lemma 1.

Proof of Lemma 1: The proof is by contradiction. Assume that in $\mathit{\Gamma}_{opt} = (G, \pi_1, \pi_{opt})$ there is a suited pair v, w whose ordering in π_{opt} is not its natural ordering. Note that v and w are not degree-1 vertices with a common neighbor, as both orderings would be natural in that case. Assume that $v < w$ is the (unique) natural ordering of v, w.

By Fact 5 the crossing number $c_{vw} = 0$ and the crossing number $c_{wv} > 0$.

For a contradiction, we now prove that either v or w can be moved in $\mathit{\Gamma}_{opt}$ such that the resulting drawing $\mathit{\Gamma}_{new} = (G, \pi_1, \pi_{new})$ satisfies $cr(\mathit{\Gamma}_{new}) < cr(\mathit{\Gamma}_{opt})$.

Let i and j denote the positions of w and v, respectively, in π_{opt}. Here $i < j$ since v and w appear in the order $w < v$ in π_{opt}.

If $|j-i| = 1$, we can interchange v and w without affecting any other pair of vertices in Γ_{opt}. Equation (3) in Lemma 2 gives the number of crossings in the resulting drawing Γ_{new}:

$$cr(\Gamma_{new}) = cr(\Gamma_{opt}) - c_{wv} + c_{vw} = cr(\Gamma_{opt}) - c_{wv} + 0.$$

Since $c_{wv} > 0$, we have $cr(\Gamma_{new}) < cr(\Gamma_{opt})$, which contradicts the optimality of Γ_{opt}.

If $|j-i| > 1$, let $u_{i+1}, u_{i+2}, \ldots, u_{j-1}$ denote the vertices between w and v in π_{opt}, listed in order of appearance in π_{opt}. Regard these vertices as a frozen block U inside which no changes are made. Figure 4 illustrates this terminology.

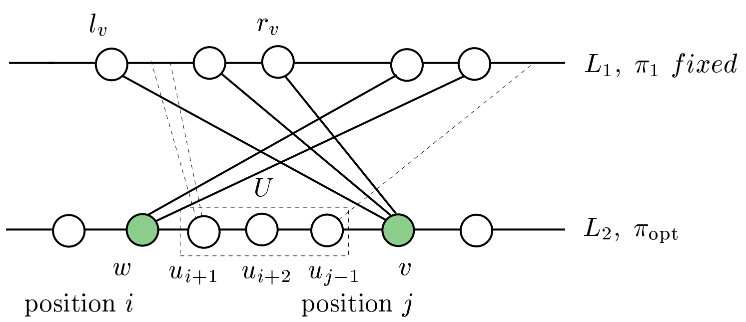

Fig. 4. Γ_{opt} for case $|j-i| > 1$ of Lemma 1

According to Lemma 2, moving v or w from one side of block U to the other may only affect the crossing number contributions of pairs of the form u, w and u, v for $u \in U$. Let c_{Up} denote the number of crossings that the edges incident to vertices in U have with the edges incident to a vertex p to the right of U: $c_{Up} = \Sigma_{u \in U}\ c_{up}$. Similarly, let c_{pU} denote this number of crossings when p lies to the left of U.

Since Γ_{opt} is optimal, we claim we have the strict inequality

$$c_{Uv} < c_{vU}. \tag{4}$$

Otherwise, we could move v to the left side of U and then interchange v with w to obtain a drawing with the following total number of crossings: $cr(\Gamma_{new}) = \Gamma_{opt} - c_{Uv} + c_{vU} - c_{wv} + 0$. Since $c_{wv} > 0$, if $c_{Uv} \geq c_{vU}$, then $cr(\Gamma_{new}) < cr(\Gamma_{opt})$, a contradiction.

observation: To conclude the case $|j-i| > 1$, it suffices to show that $c_{Uv} < c_{vU}$ implies $c_{wU} \geq c_{Uw}$, for this means we can move w to the right side of U without increasing the total number of crossings in the resulting drawing and then interchange w and v to produce a drawing with fewer crossings than Γ_{opt}. This gives a contradiction, and so proves that the assumption that π_{opt} contains a suited pair not ordered by its natural ordering cannot hold.

To establish the desired inequality $c_{wU} \geq c_{Uw}$, we first derive some intermediate inequalities for c_{Uv}, c_{vU}, c_{wU}, and c_{Uw} in terms of sizes of the following sets: E_R = the set of edges in Γ_{opt} with one endpoint in U and the other endpoint strictly greater than r_v in the ordering π_1; E_L = the set of edges with one endpoint in U and the other endpoint strictly less than r_v in the ordering π_1; N_v = the neighbors of v; and N_w = the neighbors of w.

By the definition of E_R, all the vertices in N_v occur in π_1 strictly before the L_1 endpoint of each edge in E_R. By the definition of E_L and by the fact that v, w is a suited pair with the natural ordering $v < w$, the vertices in N_w occur in π_1 strictly after the L_1 endpoints of the edges in E_L.

Fact 2 implies the following inequalities for crossing numbers:

$c_{Uv} \geq d_v \cdot |E_R|$: The edges incident to v and the edges in E_R all pairwise intersect, creating $d_v \cdot |E_R|$ crossings. Since E_R is a subset of the edges incident to U, $c_{Uv} \geq d_v \cdot |E_R|$.

$c_{vU} \leq d_v \cdot |E_L|$: This holds because no edge incident to v crosses any edge incident to U that is not in E_L.

$c_{wU} \geq d_w \cdot |E_L|$: The edges incident to w and the edges in E_L all pairwise intersect, so $c_{wU} \geq d_w \cdot |E_L|$.

$c_{Uw} \leq d_w \cdot |E_R|$: This holds because no edge incident to w intersects any edge incident to U that is not in E_R.

Recall inequality (4), that $c_{Uv} < c_{vU}$. Since $c_{Uv} \geq d_v \cdot |E_R|$ and $c_{vU} \leq d_v \cdot |E_L|$, we have $d_v \cdot |E_R| \leq c_{Uv} < c_{vU} \leq d_v \cdot |E_L|$, which implies that $|E_R| < |E_L|$. This, and the fact that $c_{wU} \geq d_w \cdot |E_L|$, and the fact that $c_{Uw} \leq d_w \cdot |E_R|$ together imply that $c_{wU} > c_{Uw}$. By the *observation* above, this completes the proof. □

4 An Efficient FPT Algorithm

4.1 The bounded search tree approach for the algorithm

One of the basic methods for developing FPT algorithms is the method of *bounded search trees* (see Chapter 3.1 in Downey and Fellows [6]). In this method one builds a search tree, which is exhaustively traversed for a solution. The critical observation for many parameterized problems is that, while the computation done at each node of the tree may depend on the problem size, the size of the tree itself depends only on the parameters.

In this section we present an FPT algorithm for the 1-Sided Crossing Minimization problem based on the bounded search tree approach. The key observations for building a search tree for this problem lie in Lemma 1 and Fact 4. Here is an overview of our algorithm.

Lemma 1 allows us, at the start, to fix the relative ordering of each non-trivial suited pair of vertices in L_2 according to its unique natural ordering. The remaining unordered pairs of vertices in L_2 are either trivial suited pairs, or unsuited pairs which will each, by Fact 4, create a crossing no matter which relative ordering is chosen. We build a search tree based on the unsuited pairs. (It turns out that the trivial pairs neighbours can be delt with later in the algorithm.) The input to every node of the search tree is a budget B giving the remaining number of allowed edge crossings, and a relation D containing all pairs of L_2 ordered thus far. We will formally define relation D shortly. At each node of the search tree some unordered pair (v, w) is chosen (i.e. a pair not in D) such that $c_{vw} \neq c_{wv}$.

Then the node branches to two recursive subproblems. In one branch, the ordering of (v, w) is fixed to $v < w$ and the budget B is reduced by c_{vw}. In the other branch the ordering of (v, w) is fixed to $w < v$ and B is reduced by c_{wv}. Since we only work with unsuited pairs in building the tree, we know that $c_{vw} \geq 1$ and $c_{wv} \geq 1$. Therefore, since the initial budget $B = k$, the height of the search tree is at most k.

As a matter of fact the situation is better than that, for two reasons. Firstly, since $c_{vw} \neq c_{wv}$, then either c_{vw} or c_{wv} is at least 2, so one of the two branches of the search tree node reduces B by at least 2. Secondly, since $<$ is a transitive relation, fixing an ordering of the pair (v, w) at a node of the search may in fact impose an ordering of another as yet unordered pair (p, q) in the relation D at that node. Hence B can be reduced not only by c_{vw} but also by either c_{pq} or c_{qp}, depending on which relative ordering is imposed on (p, q).

4.2 The algorithm

The following definitions will be useful for the description of the algorithm.

Let D be a directed acyclic graph (DAG) that represents a binary relation " $<$ " on the set of vertices L_2. In particular, the vertices of L_2 are represented by nodes of a DAG D and an ordered pair of vertices $v < w$ is represented by a directed edge from v to w (denoted henceforth by vw) in D. The DAG D is stored as an $|L_2| \times |L_2|$ matrix. We use D to denote both the set of pairs in the current binary relation " $<$ " and the associated DAG that represents these pairs as directed edges. The algorithm labels nodes in the search with DAGs. The DAG associated with the root will be transitively closed, acyclic, and directed. As the algorithm progresses, it computes a DAG label for each child node it generates in the search by choosing a directed edge to add to the DAG of the parent and then taking the transitive closure of this.
The following algorithm solves the 1-Sided Crossing Minimization problem.

Algorithm: **1-Sided Crossing Minimization**
Input: $\langle G, \pi_1, k \rangle$
Output: π_{opt} if $\langle G, \pi_1, k \rangle$ is a YES instance, else NO

Step 0. **Computing crossing numbers:** Compute the crossing numbers c_{vw} and c_{wv} for all pairs of vertices in L_2, stopping the computation of a particular crossing number as soon as it is known to exceed k. (The algorithm for computing the crossing numbers, c_{vw} and c_{wv}, efficiently can be found in Appendix A.)

Step 1. **Checking for extreme values**: Compare k with the upper and lower bound as per Fact 3.

if $k < \Sigma_{(v,w)} \min(c_{vw}, c_{wv})$ then output NO and HALT;
if $k \geq \Sigma_{(v,w)} \max(c_{vw}, c_{wv})$ then output an arbitrary π_2 and HALT.

Step 2. **Initialization**: Precompute the following information required by the search tree.

$C = \{(v, w) | c_{vw} = c_{wv}\}$;

$D_0 =$ a DAG (V, E), where $V = L_2$, and the directed edges $vw \in E$ correspond to the naturally ordered pairs (v, w) that satisfy $c_{vw} = 0$ and $c_{wv} \neq 0$. (It is easy to check that D_0 is transitively closed.);

$B_0 =$ initial budget $= k - \Sigma_{vw \in D_0} c_{vw} - \Sigma_{(v,w) \in C} c_{vw}$. Note that $\Sigma_{vw \in D_0} c_{vw} = 0$. Also note that we reduce the budget k by the eventual cost of the pairs in C even though these pairs do not appear in D_0.

Step 3. **Building and exploring the search tree**: This step effectively builds and explores the search tree simultaneously. A node of the search tree has at most two children. Each node has a label (D, B). The label D of a node represents a "possible" partial solution, i.e. a partial ordering of vertices of L_2. The label B represents the remaining budget for crossings. For instance the label of the root is: $D= D_0$ and $B = B_0$.

We now build the search tree as follows. Label the **root** of the tree with (D, B) where $D = D_0$ and $B = B_0$. In general, for a non-leaf node labeled (D, B), choose a pair (v, w) such that D contains no edge joining v and w and such that $c_{vw} \neq c_{wv}$. A pair (v, w) is thus an unordered pair not in C. In any ordering π_2, the pair (v, w) is ordered as either vw or wv, so we create at most two children (D_1, B_1) and (D_2, B_2) of the non-leaf node (D, B) corresponding to these two possibilities. No child is created if its budget would be negative. Thus a node labeled (D, B) is a **leaf** if and only if, either it does not have an unordered pair $(v, w) \notin C$; or there remains any pair (v, w) for which both B_1 and B_2 are negative.
For a non-leaf node (D, B), we label one of its two children by (D_1, B_1) where:
$D_1 =$ transitive closure of $D \cup vw$, and $B_1 = B - c_{vw} - \sum_{pq} c_{pq}$.
Here $D \cup vw$ represents the addition of directed edge vw to D. The summation in B_1 is over the directed edges that are added to $D \cup vw$ by the transitive closure and that have $c_{pq} \neq c_{qp}$. That is, the sum is over pq s.t. $pq \in D_1$ and $pq \notin D \cup vw$ and $c_{pq} \neq c_{qp}$.
Similarly, we label the other child of node (D, B) with (D_2, B_2), where:
$D_2 =$ transitive closure of $D \cup wv$, and $B_2 = B - c_{wv} - \sum_{pq} c_{pq}$.
These concepts are illustrated in Figure 5.
If a leaf is created whose label D has the property that
$\forall (v, w)$ if $vw \notin D$ and $wv \notin D$ then $(v, w) \in C$,
then **output π_2 = topological sort of D**. Also, update the minimum number of crossings found so far to $k - B$, where B is the budget of the leaf, and update the best ordering so far to π_2. We call such a node a *solution leaf*.

If, after exploring the entire tree, no solution leaf is found, **output NO** and **HALT**; otherwise, output the best ordering found, which is π_{opt}, and **HALT**.

Remarks. In Step 0, we stop computing c_{vw} as soon as it becomes $k + 1$, even though c_{vw} may be bigger than that. This is because a child with $v < w$ would have a negative budget. Hence it suffices to know that $c_{vw} \geq k + 1$.

Step 3 of the algorithm effectively creates and explores the search tree simultaneously. This can be done by depth-first search, or by breath-first search. The depth-first way requires less space and is thus the preferred choice.

Also notice that, when creating a child by choosing, say, to order v and w as vw, we reduce the budget for the child by an amount computed not only for the

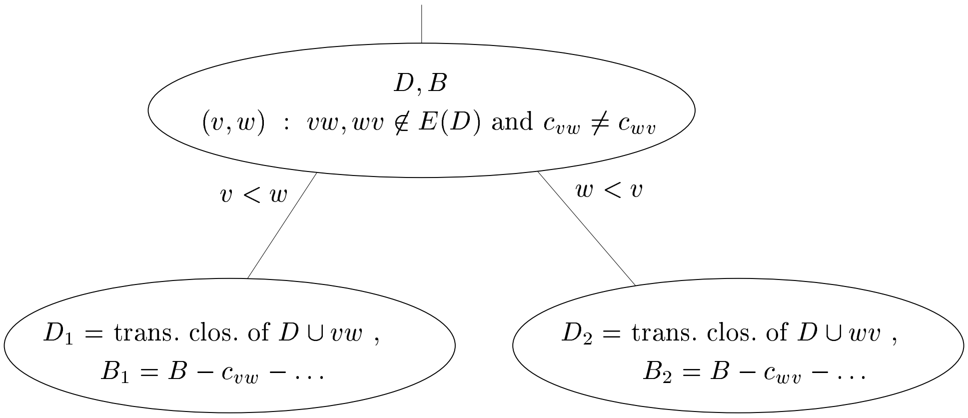

Fig. 5. Illustration for step 3 of the algorithm.

ordered pair vw, but also for the pairs that are newly ordered by the transitive closure of $D \cup vw$. However, we only do this for newly ordered pairs whose two crossing numbers are not the same. Those whose crossing numbers are the same have already been accounted for in B_0.

Theorem 6. *Given a bipartite graph $G = (L_1, L_2; E)$, a fixed permutation π_1 of L_1, and an integer k, algorithm* 1-Sided Crossing Minimization(G, π_1, k) *determines in $O(\phi^k \cdot |L_2|^2 + |L_1||L_2|)$ time if $\mathrm{opt}(G, \pi_1) \leq k$ and if yes produces a 2-layer drawing (G, π_1, π_2) with the optimal number of crossings. The constant ϕ in the running time is the golden ratio $\phi = \frac{1+\sqrt{5}}{2} \approx 1.618$.*

Proof: Step 3 of the algorithm creates and explores the search tree simultaneously. For every node (D, B) of the search tree we maintain the following two invariants: (i) D is a transitively closed, directed, acyclic graph; (ii) The budget B at node (D, B) is $B = k - \sum_{vw \in C} c_{vw} - \sum_{vw \in D \& vw \notin C} c_{vw}$. This is true for the root node (D_0, B_0). Suppose this is true for a node labeled (D, B). At this node, the algorithm chooses an unordered pair (v, w) with $c_{vw} \neq c_{wv}$. This pair is used to create up to two child nodes. We claim that both $D \cup vw$ and $D \cup wv$ are acyclic. Suppose, on the contrary, that $D \cup vw$ contains a directed cycle. Then D must contain a directed path from w to v. Since D is transitively closed, it contains edge wv, contradicting the fact that (v, w) is unordered in D. Similarly for $D \cup wv$. Since the transitive closure of a directed acyclic graph is again acyclic, the graph labels D_1, D_2 for any child nodes created at a node labeled D are again transitive and acyclic. Thus all the graph labels in the search tree are directed, acyclic, and transitively closed. The fact that labels B_1 and B_2 agree with formula (ii), follows directly from the formulas used to compute these two labels from the parent label B in Step 3 of the algorithm.

As the tree is built, either a solution leaf is found, or the tree is completely explored without finding such a leaf. A solution leaf (D, B) has, by definition, a non-negative budget B. By the invariant (ii), the cost of all the crossings arising from the ordered pairs in D has been taken into account. By the definition of a solution leaf, all pairs (v, w) not ordered by directed edges in D are in C and satisfy $c_{vw} = c_{wv}$; hence the total cost directly attributable to them has already been deducted from the initial budget k. Hence any topological sort of D produces a total ordering consistent with D and having total cost $k - B$, where $0 \leq B \leq k$. By the invariant (i), the label D of every node of the search tree is an acyclic graph and it necessarily has a topological sort. Based on this argument, a solution leaf (D, B) encodes an ordering π_2 such that $cr(G, \pi_1, \pi_2) \leq k$.

It is not difficult to verify that the solution leaves of the search tree implicitly store all the orderings π_2 for which $cr(G, \pi_1, \pi_2) \leq k$ and in which all the suited pairs are ordered by their natural ordering. Lemma 1 implies that in order to decide if $\langle G, \pi_1, k \rangle$ is a YES or NO instance it is enough to consider only such orderings π_2. Therefore, if there is an ordering π_2 such that $cr(G, \pi_1, \pi_2) \leq k$ the algorithm finds one. In fact, since the algorithm updates the best solution found so far, when it terminates it outputs an optimal ordering π_{opt}.

We now discuss the running time of the algorithm.

Only for an unordered unsuited pair v, w that has $c_{vw} \neq c_{wv}$ are child nodes created, of which there are at most two. Therefore, in one child node the budget is reduced by at least 1 and in the other by at least 2. A node with $B = 0$ must be a leaf node, because any child of such a node would have a negative budget. Therefore, no further branching is allowed. At a node for which budget $B = 1$, at most one child can have a budget B_1 that is non-negative, and in this case, $B_1 = 0$ and the child must be a leaf. Thus a recurrence relation that generates an upper bound for the number of nodes in this search tree is:

$s_B = s_{B-1} + s_{B-2} + 1$ for $B \geq 2$; $s_0 = 1$, $s_1 = 2$.

It can be verified by induction that for $B \geq 0$, $s_B = F_{B+2} + F_{B+1} - 1$ where F_B is the B-th Fibonacci number. From the bound on Fibonacci numbers and given that $B_0 \leq k$, it follows that $s_{B_0} < \frac{\phi^{k+2}}{\sqrt{5}} + \frac{\phi^{k}+1}{\sqrt{5}} - 1 < 1.2 \cdot \phi^{k+1}$. Thus the search tree has $\mathcal{O}(\phi^k)$ nodes.

The time taken at each node of the search tree is dominated by updating a transitive closure of its label D after insertion of one ordered pair $v < w$ (or $w < v$). Updating the transitive closure after one insertion can be done in $\mathcal{O}(|L_2|^2)$ time (See problem 25-1, page 641 in [3]). These updates are needed to generate the labels for the children, of which there are at most two. Thus the time taken in the third step of the 1-Sided Crossing Minimization algorithm is $\mathcal{O}(\phi^k \cdot |L_2|^2)$.

It can be shown that the time taken in steps 0 - 2 of the algorithm is $\mathcal{O}(k \cdot |L_2|^2 + |L_1||L_2|)$. (For details see Appendix A.) Thus the total running time of the algorithm is $\mathcal{O}(\phi^k \cdot |L_2|^2 + |L_1||L_2|)$. □

Corollary 1. *Given a bipartite graph $G = (L_1, L_2; E)$, and a fixed permutation π_1 of L_1, algorithm* 1-Sided Crossing Minimization$(G, \pi_1, \lceil 1.47\,\mathrm{lb}(G, \pi_1) \rceil)$ *produces a 2-layer drawing (G, π_1, π_2) with the optimal number of crossings in $O(\phi^{\mathrm{opt}(G,\pi_1)} \cdot |L_2|^2 + |L_1||L_2|)$ time.*

Proof: By the results of [19] and Fact 3, $\mathrm{lb}(G, \pi_1) \leq \mathrm{opt}(G, \pi_1) \leq 1.47\,\mathrm{lb}(G, \pi_1)$. Therefore, by Theorem 6, the algorithm 1-Sided Crossing Minimization$(G, \pi_1, \lceil 1.47\,\mathrm{lb}(G, \pi_1) \rceil)$ finds the optimal solution.

Since parameter k is not given as a part of the input, and is instead set to the value $k = \lceil 1.47\,\mathrm{lb}(G, \pi_1)\rceil$ we need to consider the time it takes to compute $\mathrm{lb}(G, \pi_1)$. To determine this lower bound, we need to compute the smaller of the two crossing numbers, c_{vw} and c_{wv}, for each pair of vertices v, w. This can be easily achieved by slightly modifying the algorithm in the appendix. In particular, the while loop needs to compute both crossing numbers simultaneously until one of them, say c_{vw}, is completed. If at that moment $c_{vw} \le c_{wv}$, then the while loop terminates, otherwise it continues computing c_{wv} until either c_{wv} becomes greater than c_{vw} or, until the computation of c_{wv} terminates. For the same reasons as those presented in the proof of Lemma 3, the running time of this modified algorithm is $\mathcal{O}(\mathrm{lb}(G, \pi_1)|L_2|^2 + |L_1||L_2|) \in \mathcal{O}(\mathrm{opt}(G, \pi_1)|L_2|^2 + |L_1||L_2|)$. This together with Theorem 6 implies the running time claimed in this corollary. $\square$

4.3 A special case and two generalizations

In this section, we first describe a method for improving the running time of our algorithm in practice; then we show how to extend the algorithm to allow drawing edges within single layers.

1-layer cut vertex The fact that the 1-Sided Crossing Minimization problem is NP-complete even for a forest of 4-stars [18] means that a divide-and-conquer approach based on cut-vertices is not useful. However, we now show that there is a way to decompose the input graph into parts, to which our algorithm can be applied separately.

A vertex q in L_1 is called a *1-layer cut vertex* if we can partition the vertices of L_2 into two nonempty sets L_2' and L_2'' such that: $\forall v \in L_2'$, $r_v \le q$, and $\forall v \in L_2'', l_v \ge q$. The set of all the neighbors of vertices in L_2' is denoted by $N(L_2')$. The set $N(L_2'')$ is defined similarly. The graphs induced by vertex sets $V' = L_2' \cup N(L_2')$ and $V'' = L_2'' \cup N(L_2'')$ are denoted by $G[V']$ and $G[V'']$, respectively. We say $G[V']$ and $G[V'']$ are *1-layer components* of $\langle G, \pi_1, k\rangle$. Note that q may or may not belong to each component. These concepts are illustrated in Fig. 6.

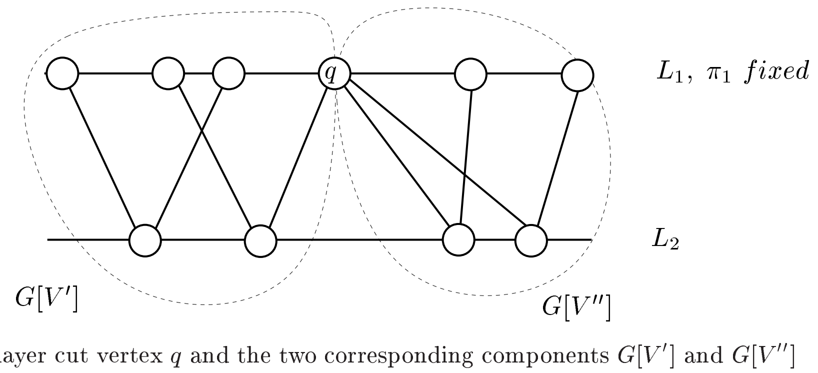

Fig. 6. A 1-layer cut vertex q and the two corresponding components $G[V']$ and $G[V'']$

Corollary 2. *Let q be a 1-layer cut vertex for $\langle G, \pi_1, k\rangle$ that splits G into two 1-layer components $G[V']$ and $G[V'']$. In an optimal drawing (G, π_1, π_{opt}), no edge of $G[V']$ crosses any edge of $G[V'']$.*

Proof: This follows directly from Lemma 1. Namely, for all v, w such that $v \in V'$ and $w \in V''$, it is true that v, w is a suited pair with the natural ordering $v < w$. So all the vertices of V' lie before the first vertex of V'' in any π_{opt}. Thus the vertices of V' and V'' are pairwise suited, so none of their edges cross in an optimal drawing (G, π_1, π_{opt}). □

This corollary shows that a problem instance $\langle G, \pi_1, k\rangle$ can sometimes be divided into two independent instances $\langle G[V'], \pi_1[V'], k\rangle$ and $\langle G[V''], \pi_1[V''], k''\rangle$. We can then solve the first instance, and if it is a NO instance then the original instance is also a NO instance. If it is a YES instance, we move on to solving the second instance with the input parameter $k'' = k - cr(G[V'], \pi_1[V'], \pi_{opt}[V'])$. In some cases this may cut the exponent in the running time by half. As a matter of fact, if the number of 1-layer cut vertices in $\langle G, \pi_1, k\rangle$ is c, then the exponential part of the running time may drop to $\phi^{\frac{k}{c+1}}$. This behavior should be expected in YES instances where the number of crossings is evenly distributed over all the 1-layer components.

Multiple edges We now briefly discuss how to deal with instances of the 1-Sided Crossing Minimization problem that have multiple edges. For every pair of vertices connected by s edges, replace the edges by one edge with *weight s*. Thus, we obtain a variant of the 1-Sided Crossing Minimization problem where each edge has a positive weight. If two edges weighted s_1 and s_2 cross in a drawing, their contribution to the total number of crossings is $s_1 \cdot s_2$. Having this in mind the Definition 1 of crossing numbers c_{vw} and c_{wv} for a pair of vertices v, w remains the same. All the other definitions, facts and lemmas, except for Fact 5 and Lemma 1, follow through without any changes. We now modify the definition of d_v to mean the sum of the weights of all the edges incident to a vertex v. Then case (iii) of Fact 5 becomes $c_{wv} = d_v \cdot d_w - s_{vr_v} \cdot s_{wl_w}$ where s_{vr_v} and s_{wl_w} denote the weights of the two edges vr_v and wl_w. In the proof of Lemma 1, we also modify the definition of $|E_L|$ and $|E_R|$ to mean the sum of the weights of all the edges in the sets E_L and E_R, respectively. The correctness of the lemma, and hence the whole algorithm, follows through without any other changes.

Improper 2-layer drawings Our 1-Sided Crossing Minimization algorithm can easily be extended to manage the version of 2-layer drawings called *improper 2-layer drawings*. Here edges are allowed between vertices lying next to each other in the same layer (see [13]).

Let $G[L_1]$ and $G[L_2]$ be the graphs induced by the vertex sets L_1 and L_2, respectively, and suppose that $G[L_1]$ and $G[L_2]$ are forests of paths. Let p be a path in $G[L_2]$ with vertices $p_1, p_2, \ldots p_j$. Furthermore, let c_p be the minimum number of crossings amongst the edges incident to a path p in one of the two possible ways to draw that path p in π_2 (one way is to have $p_1 p_2 \ldots p_j$ consecutively in π_2, and the other it to have $p_j p_{j-1} \ldots p_1$ consecutively in π_2).

The algorithm is now modified by adding the following preprocessing step. For each path p in $G[L_2]$, all the edges (p_i, p_{i+1}) of p in $G[L_2]$ are contracted into one vertex. Consequently, the parameter k is reduced by c_p for each path p. See Fig. 7. Contracting all the paths in $G[L_2]$ gives an instance of 1-Sided Crossing Minimization problem that may have multiple edges, which our algorithm can deal with as described earlier in this previous section. This completes the description of the modifications to the original algorithm.

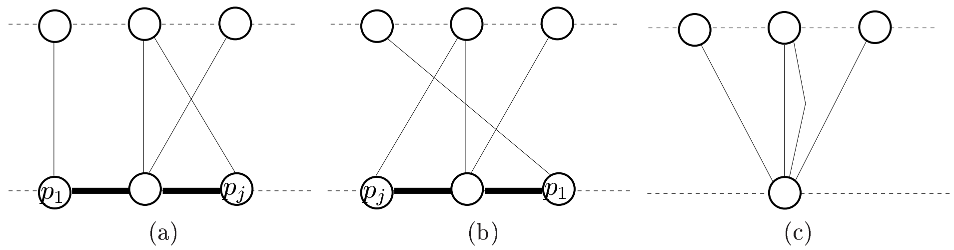

Fig. 7. Edges of p are depicted in bold. (a),(b) The two ways to draw p. The first creates 1 crossing, the second creates 3 crossings; (c) After p is contracted, k is reduced by $c_p = \min\{1, 3\} = 1$.

5 Conclusion

We have studied the 1-Sided Crossing Minimization problem and presenting a very easy-to-implement FPT algorithm for its solution. Moreover, the algorithm finds a drawing with the smallest possible number of crossings in the case that this number does not exceed k. In case an optimal solution is desired no matter how many crossings it has, as Corollary 1 points out, our algorithm finds it by setting k to $1.47\,\mathrm{lb}(G, \pi_1)$, in time $\mathcal{O}(\phi^{\mathrm{lb}(G,\pi_1)} \cdot n^2) \in \mathcal{O}(\phi^{\mathrm{opt}(G,\pi_1)} \cdot n^2)$.

The exponential part of the running time of the algorithm is 1.618^k. In many instances the base of this exponent will be even smaller. The reason is that a pair of vertices v, w will often have both crossing numbers c_{vw} and c_{wv} bigger than 1, or at least one of them bigger than 2; thus each time a node of the search tree branches on such a pair of vertices, the resulting search tree will be even smaller.

An interesting investigation for future research would be to compare experimentally the performance of the other known method for optimal 1-Sided Crossing Minimization, namely, integer linear programming [16], with our FPT algorithm with k set to $1.47\,\mathrm{lb}(G, \pi_1)$. In the case of the related 2-layer planarization problem, recent experimental comparisons [20] suggest that the FPT method is competitive with the ILP method. Hence an experimental study of 1-Sided Crossing Minimization would be worthwhile.

Numerous graph drawing problems involve optimizations that are hard. It is therefore interesting to investigate whether fixed parameter tractability provides a useful approach for dealing with these problems.

References

1. M. J. Carpano. Automatic display of hierarchized graphs for computer aided decision analysis. *IEEE Trans. Syst. Man Cybern.*, SMC-10(11):705– 715, 1980.
2. C. Catarci. The assignment heuristic for crossing reduction. *IEEE Trans. on Systems, Man, and Cybernetics*, 25(3), 1995.
3. T. Corman, R. Rivest C. Leiserson, and C. Stein. *Introduction to Algorithms - Second Edition.* MIT Press, 2001.
4. C. Demetrescu and I. Finocchi. Break the right cycles and get the best drawing. In *Proc. of the 2nd Workshop on Algorithm Engineering and Experiments (ALENEX'00)*, pages 171–172, 2000.
5. G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs.* Prentice-Hall, 1999.
6. R. G. Downey and M. R. Fellows. *Parametrized complexity.* Springer, 1999.
7. S. Dresbach. A new heuristic layout algorithm for directed acyclic graphs. In U. Derigs, A. Bachem, and A. B. A. Drexl, editors, *Operations Research Proceedings (1994)*, pages 121–126, 1995.
8. V. Dujmović, M. Fellows, M. Hallett, M. Kitching, G. Liotta, C. McCartin, N. Nishimura, P. Ragde, F. Rosemand, M. Suderman, S. Whitesides, and D. R. Wood. A fixed-parameter approach to two-layer planarization. In P. Mutzel, editor, *Proc. Graph Drawing: 9th International Symposium (GD'01)*, volume 2265 of *Lecture Notes in Comput. Sci.*, pages 1–15, 2001.
9. V. Dujmović, M. Fellows, M. Hallett, M. Kitching, G. Liotta, C. McCartin, N. Nishimura, P. Ragde, F. Rosemand, M. Suderman, S. Whitesides, and D. R. Wood. On the parameterized complexity of layered graph drawing. In *Proc. 9th European Symposium on Algorithms (ESA 2001)*, volume 2161 of *Lecture Notes in Comput. Sci.*, pages 488–499, 2001.
10. P. Eades and D. Kelly. Heuristics for drawing 2-layered networks. *Ars Combin.*, 21(A):89–98, 1986.
11. P. Eades and S. Whitesides. Drawing graphs in two layers. *Theoret. Comput. Sci.*, 131(2):361–374, 1994.
12. P. Eades and N. C. Wormald. Edge crossings in drawings of bipartite graphs. *Algorithmica*, 11(4):379–403, 1994.
13. S. Felsner, G. Liotta, and S. Wismath. Straight-line drawings on restricted integer grids in two and three dimensions. In P. Mutzel, editor, *Proc. Graph Drawing: 9th International Symposium (GD'01)*, volume 2265 of *Lecture Notes in Comput. Sci.*, pages 328–342, 2001.
14. M. R. Garey and D. S. Johnson. Crossing number is NP-complete. *SIAM J. Algebraic Discrete Methods*, 4(3):312–316, 1983.
15. M. Grohe. Computing crossing numbers in quadratic time. In *Proc. 32nd Annual ACM Symposium on Theory of Computing (STOC'01)*, pages 231–236, 2001.
16. M. Jünger and P. Mutzel. 2-layer straightline crossing minimization: performance of exact and heuristic algorithms. *J. Graph Algorithms Appl.*, 1(1):1–25, 1997.
17. T. Lengauer. *Combinatorial Algorithms for Integrated Circuit Layout.* John Wiley, 1990.
18. X Muñoz, U Unger, and I Vrťo. One sided crossing minimization is NP-hard for sparse graphs. In P. Mutzel, editor, *Proc. Graph Drawing: 9th International Symposium (GD'01)*, volume 2265 of *Lecture Notes in Comput. Sci.*, pages 115–123, 2001.
19. H. Nagamochi. An improved approximation to the one-sided bilayer drawing. In G. Liotta, editor, *Proc. Graph Drawing: 11th International Symposium (GD'03)*, volume 2912 of *Lecture Notes in Comput. Sci.*, pages 406–418, 2003.
20. M. Suderman and S. Whitesides. Experiments with the fixed-parameter approach for two-layer planarization. In G. Liotta, editor, *Proc. Graph Drawing: 11th International Symposium (GD'03)*, volume 2912 of *Lecture Notes in Comput. Sci.*, pages 345–356, 2003.

21. K. Sugiyama, S. Tagawa, and M. Toda. Methods for visual understanding of hierarchical system structures. *IEEE Trans. Systems Man Cybernet.*, 11(2):109–125, 1981.
22. N. Tomii, Y. Kambayashi, and S. Yajima. On planarization algorithms of 2-level graphs. *Papers of tech. group on elect. comp., IECEJ*, EC77-38:1–12, 1977.
23. V. Valls, R. Marti, and P. Lino. A branch and bound algorithm for minimizing the number of crossing arcs in bipartite graphs. *J. Operational Res.*, 90:303–319, 1996.
24. J. N. Warfield. Crossing theory and hierarchy mapping. *IEEE Trans. Systems Man Cybernet.*, SMC-7(7):505–523, 1977.
25. M. S. Waterman and J. R. Griggs. Interval graphs and maps of DNA. *Bull. Math. Biol.*, 48(2):189–195, 1986.

A Computing crossing numbers c_{vw} and c_{wv}

For completeness, to justify the correctness of the running time claimed in Theorem 6, we give here an $\mathcal{O}(k|L_2|^2+|L_1||L_2|)$ time algorithm for Step 0 of the 1-Sided Crossing Minimization algorithm. The algorithm computes the crossing numbers c_{vw} and c_{wv} for all pairs of vertices. Although the algorithm is simple, some care needs to be taken not to exceed the claimed running time. For instance, we must stop the computation of a particular crossing number as soon as it is known to exceed k. In place of crossing numbers that do exceed k we record some number strictly bigger than k.

Let the graph G in the input instance $\langle G, \pi_1, k\rangle$ be given as an $|L_2| \times |L_1|$ adjacency matrix $A = [a_{i,j}]$. The columns are labeled by the vertices of L_1 in the order of their appearance in π_1. The rows are labeled by the vertices of L_2. An element $a_{i,j} = 1$ if vertex i of L_2 is adjacent to vertex j of L_1; otherwise $a_{i,j} = 0$. We augment every element $a_{i,j}$ of the adjacency matrix A with the following information:

$p_{i,j}$: the index of the first neighbor of vertex i that is to the right of j. More precisely, if $\exists j' > j$ s.t. $a_{i,j'} = 1$ and $a_{i,j''} = 0, \forall j < j'' < j'$ then $p_{i,j} = j'$; otherwise $p_{i,j} = |L_1| + 1$.

$r_{i,j}$: the number of neighbors of i that are to the right of j. More precisely,
$$r_{i,j} = \sum_{j'=j+1}^{|L_1|} a_{i,j'}.$$

In addition, for every row i, we store the following information:

l_i : the left-most neighbor of vertex i. More precisely $l_i = j$ where $a_{i,j} = 1$ and $a_{i,j'} = 0, \forall j' < j$

r_i : the right-most neighbor of vertex i. More precisely, $r_i = j$ where $a_{i,j} = 1$ and $a_{i,j'} = 0, \forall j' > j$.

The following algorithm computes an $|L_2| \times |L_2|$ matrix $C = [c_{v,w}]$. At the end of the algorithm, the matrix entry $c_{v,w}$ equals the crossing number c_{vw} provided this is equal or less than k; otherwise, the entry $c_{v,w}$ equals some number greater than k.

Algorithm: **Pair Crossing Numbers**
Input: $|L_2| \times |L_1|$ adjacency matrix A of G
Output: $|L_2| \times |L_2|$ matrix C

---

1. Augment adjacency matrix A as described above.
2. **for** $v = 1$ to $|L_2|$
3.   **for** $w = 1$ to $|L_2|$
4.   **if** $v \neq w$
5. $c_{v,w} = 0$ /* initialize $c_{v,w}$ */
6. $w' = l_w$ /* start examining the neighbours w' of w starting with the left-most neighbour, l_w */
7.   **while** $w' \leq r_w$ and $w' < r_v$ and $c_{v,w} \leq k$ **do**
8. $c_{v,w} = c_{v,w} + r_{v,w'}$ /* increment $c_{v,w}$ by the number of crossing points on edge (w, w') created by edges incident to v */
9. $w' = p_{w,w'}$ /* advance to the next neighbour of w */

Lemma 3. *The* Pair Crossing Numbers *algorithm computes the exact values of all the crossing numbers that do not exceed k in time $\mathcal{O}(k|L_2|^2 + |L_1||L_2|)$.*

Proof: Lines $5 - 9$ of the algorithm compute the crossing number c_{vw} for a pair of vertices v, w ordered $v < w$. The correctness of the computation follows from the next observation.

By Fact 1, for an ordered pair $v < w$, the number of edges incident to v that cross an edge (w, w') incident to w is precisely the number of neighbours of v strictly to the right of w'. By definition that number is $r_{v,w'}$. Thus $c_{vw} = \sum r_{v,w'}$ where the sum is over all w' adjacent to w. This formula is still correct if the sum is taken only over $\{w' \mid w' < r_v\}$, since otherwise $r_{v,w'} = 0$ by Fact 1.

Now consider the complexity of this algorithm. It is simple to verify that the original matrix A can be augmented by traversing each of its rows once from right to left. Per one iteration of the inner for-loop, the while-loop is executed at most $k + 1$ times, since each execution of the while loop increases c_{vw} by at least one. Given that the inner for-loop is executed $|L_2|^2$ times, the total running time of the algorithm is $\mathcal{O}(k|L_2|^2 + |L_1||L_2|)$. $\square$