Algorithmica (2015) 72:778–790
DOI 10.1007/s00453-014-9872-x

A Fast and Simple Subexponential Fixed Parameter Algorithm for One-Sided Crossing Minimization

Yasuaki Kobayashi · Hisao Tamaki

Received: 28 January 2013 / Accepted: 18 January 2014 / Published online: 30 January 2014

Abstract We give a subexponential fixed parameter algorithm for one-sided crossing minimization. It runs in $O(k2^{\sqrt{2k}} + n)$ time, where n is the number of vertices of the given graph and parameter k is the number of crossings. The exponent of $O(\sqrt{k})$ in this bound is asymptotically optimal assuming the Exponential Time Hypothesis and the previously best known algorithm runs in $2^{O(\sqrt{k}\log k)} + n^{O(1)}$ time. We achieve this significant improvement by the use of a certain interval graph naturally associated with the problem instance and a simple dynamic program on this interval graph. The linear dependency on n is also achieved through the use of this interval graph.

Keywords Fixed parameter algorithm · Graph drawing · Subexponential time

1 Introduction

A *two-layer drawing* of a bipartite graph G with bipartition (X, Y) of $V(G)$ places vertices in X on one line and those in Y on another line parallel to the first and draws edges as straight line segments between these two lines. We call these parallel lines *layers* of the drawing. A *crossing* in a two-layer drawing is a pair of edges that intersect each other at a point not representing a vertex. Note that the set of crossings in a two-layer drawing of G is completely determined by the order of the vertices in X on one layer and the order of the vertices in Y on the other layer. We consider the following problem.

OSCM (One-Sided Crossing Minimization)
Instance: $(G, X, Y, <, k)$, where G is a bipartite graph on $X \cup Y$ with $E(G) \subseteq X \times Y$, $<$ is a total order on X, and k is a positive integer.

Y. Kobayashi (✉) · H. Tamaki
Meiji University, Kawasaki, Kanagawa, Japan
e-mail: yasu0207@cs.meiji.ac.jp

Question: Is there a total order $<'$ on Y such that the two-layer drawing of G in which the vertices in X are ordered by $<$ in one layer and those in Y are ordered by $<'$ in the other layer has k or fewer crossings?

OSCM is a key subproblem in a popular approach to multi-layer graph drawing, called the "Sugiyama approach" [19], which repeatedly solves OSCM for two adjacent layers as it sweeps the layers from top to bottom and vice versa, in hope of reducing the total number of crossings in the entire drawing.

OSCM is known to be NP-complete [7], even for sparse graphs [16]. On the positive side, Dujmović and Whitesides [5] showed that OSCM is fixed parameter tractable [4], that is, it can be solved in $f(k)n^{O(1)}$ time for some function f. More specifically, the running time of their algorithm is $O(\psi^k \cdot n^2)$, where $n = |V(G)|$ and $\psi \sim 1.6182$ is the golden ratio. This result was later improved by Dujmović, Fernau, and Kaufmann [6] who gave an algorithm with running time $O(1.4656^k + kn^2)$. Fernau et al. [8] reduced this problem to weighted FAST (feedback arc sets in tournaments) and, using the algorithm of Alon, Lokshtanov, and Saurabh [2] for weighted FAST, gave a subexponential time algorithm that runs in $2^{O(\sqrt{k}\log k)} + n^{O(1)}$ time. This reduction also gave a PTAS using the algorithm of Kenyon-Mathieu and Schudy [13]. Karpinski and Schudy [12] considered a different version of weighted FAST proposed in [1], which imposes certain restrictions called probability constraints on the instances, and gave a faster algorithm that runs in $2^{O(\sqrt{OPT})} + n^{O(1)}$ time where OPT is the cost of an optimal solution. However, reducing OSCM to this version of FAST seems difficult: a straightforward reduction produces an instance that does not satisfy the required probability constraints. Nagamochi gave a polynomial time 1.4664-approximate algorithm [18] and $(1.2964 + 12/(\delta - 4))$-approximate algorithm when the minimum degree δ of a vertex in Y is at least 5 [17].

Our main result in this paper is the following.

Theorem 1 *OSCM can be solved in $O(k2^{\sqrt{2k}} + n)$ time, assuming that G is given in the adjacency list representation and X is given in a list sorted in the total order $<$.*

Our algorithm is faster than any of the previously known parameterized algorithms. Both the dependency $O(k2^{\sqrt{2k}})$ on k and the dependency $O(n)$ on n are strictly better than the algorithms cited above. In particular, the exponent $\sqrt{2k}$ does not contain the $\log k$ factor or any hidden constant as in the exponent $O(\sqrt{k}\log k)$ of [8], the only previously known subexponential algorithm for OSCM. Note that the running time of our algorithm is linear in n as long as $k \le \frac{\log^2 n}{2} - 2\log\log n$, where the base of the logarithm is 2. The improvement is not only of theoretical but also of practical importance: the range of k for which the problem can be practically solvable is significantly extended.

Moreover, the exponent of $O(\sqrt{k})$ in our bound is asymptotically optimal under the Exponential Time Hypothesis (ETH) [9], a well-known complexity assumption which states that, for each $k \ge 3$, there is a positive constant c_k such that k-SAT cannot be solved in $O(2^{c_k n})$ time where n is the number of variables. ETH has been used to derive lower bounds on parameterized and exact computation (see [15] for a survey). The proof of the following theorem is in Sect. 6.

Theorem 2 *There is no $2^{o(\sqrt{k})}n^{O(1)}$ time algorithm for OSCM unless ETH fails.*

Another advantage of our algorithm over the previous algorithms is simplicity. The algorithm in [5] involves several reduction rules for kernelization and the improvement in [6] is obtained by introduction of additional reduction rules which entail more involved analysis. The algorithm in [8] relies on the algorithm in [2] for the more general problem of FAST. Our result suggests that OSCM is significantly easier than FAST in that it does not require any advanced algorithmic techniques or sophisticated combinatorial structures used in the algorithm of [2] for FAST, in deriving a subexponential algorithm.

Our algorithm is along the lines of earlier work [5, 6]. We emphasize that our improvement does not involve any complications but rather comes with simplifications. Our algorithm does not require any kernelization. It is a straightforward dynamic programming algorithm on an interval graph associated with each OSCM instance. This interval graph is implicit in the earlier work [5, 6], but is neither made explicit nor fully exploited in the previous work. Once we recognize the key role this interval graph plays in the problem, the design and analysis of an efficient algorithm becomes rather straightforward. Below we sketch how this works.

Fix an OSCM instance $(G, X, Y, <, k)$. For each vertex $y \in Y$, let l_y (r_y, resp.) denote the smallest (largest, resp.) $x \in X$ adjacent to y, with respect to the given total order $<$. We denote the half-open interval $[l_y, r_y) = \{x \in X \mid l_y \le x < r_y\}$ in the ordered set $(X, <)$ by I_y and denote the system of intervals $\{I_y \mid y \in Y\}$ by $\mathcal{I}$. For simplicity, we assume here that the degree of each vertex y in Y is at least 2 so that the interval I_y is non-empty. Our formal treatment in Sect. 3 does not need this assumption. A key observation in [5] (see Lemma 2 in the present paper), is that if $r_u \le l_v$ for distinct vertices $u, v \in Y$ then u precedes v in any optimal ordering of Y. Therefore, to determine the optimal ordering on Y, we only need to determine the pairwise order for each pair $\{u, v\}$ such that $l_v < r_u$ and $l_u < r_v$, that is, such that the intervals I_u and I_v intersect each other. Thus, the problem can be viewed as that of orienting edges of the interval graph defined by the interval system $\mathcal{I}$. The fact exploited in earlier work [5, 6] to obtain fixed parameter algorithms for OSCM is that, in our terminology, this interval graph has at most k edges in feasible instances of OSCM, as each pair of u and v such that I_u and I_v intersect each other contributes at least one crossing to the drawing no matter which ordering of this pair in Y is chosen. Our interval graph view tells us more: the clique size of this interval graph for a feasible instance is at most $\sqrt{2k} + 1$, as otherwise it has more than k edges, and hence it has a path-decomposition of width at most $\sqrt{2k}$ (see [3], for example, for interval graphs and their path-decompositions). Our algorithm is a natural dynamic programming algorithm based on this path-decomposition and runs in time exponential in the width of the decomposition.

We remark that the interval system $\mathcal{I}$ also plays an important role in reducing the dependency of the running time on n to $O(n)$. See Sect. 4 for details.

A preliminary version [14] of the present paper showed that the algorithm runs in time $O(3^{\sqrt{2k}} + n)$. In this paper, we improve the running time to $O(k2^{\sqrt{2k}} + n)$.

The rest of this paper is organized as follows. In Sect. 2, we give preliminaries of the problem and outline our entire algorithms. In Sect. 3, we describe the construction of the interval systems used in our algorithm. In Sect. 4, we describe a preprocessing phase of our algorithm. In Sect. 5, we describe our dynamic programming algorithm. Finally, in Sect. 6, we give a proof of Theorem 2.

2 Preliminaries and Outline of the Algorithm

In this section, we give some preliminaries and outline our algorithm claimed in Theorem 1. Throughout the remainder of this paper, $(G, X, Y, <, k)$ will always be the given instance of OSCM. We assume that G does not have any parallel edges or isolated vertices. We denote the number of vertices $|V(G)|$ by n and the number of edges $|E(G)|$ by m. For each $v \in X \cup Y$, we denote the set of neighbors of v in G by $N(v)$ and its degree $|N(v)|$ by $d(v)$. We assume that $N(v)$ is given as a list, together with its length $d(v)$. We also assume that X is given as a list in which the vertices are ordered by $<$.

For each pair of distinct vertices $u, v \in Y$, we denote by $c(u, v)$ the number of pairs (x, x') with $x \in N(u)$, $x' \in N(v)$, and $x' < x$. Note that $c(u, v)$ is the number of crossings between the edges incident with u and those incident with v when the position of u precedes that of v in the layer for Y. We extend this notation for sets: for each disjoint subsets U and V of Y, we define $c(U, V) = \sum_{u \in U, v \in V} c(u, v)$.

We represent total orderings by permutations in our algorithm. Let U be a finite set. A *permutation* on U, in this paper, is a sequence of length $|U|$ in which each member of U appears exactly once. We denote the set of all permutations on U by $\Pi(U)$. Let $\pi \in \Pi(U)$. We define the total order $<_\pi$ on U naturally induced by π: for $u, v \in U$, $u <_\pi v$ if and only if u appears before v in π.

For each subset U of Y and a permutation π on U, we denote by $c(\pi)$ the number of crossings among the edges incident with U when the vertices in U is ordered by π, that is,

$$c(\pi) = \sum_{u, v \in U, u <_\pi v} c(u, v).$$

For each subset U of Y, we define $\text{opt}(U) = \min\{c(\pi) \mid \pi \in \Pi(U)\}$. The goal of our algorithm is to decide if $\text{opt}(Y) \leq k$.

We need the following simple observation to bound the number of edges in feasible instances of OSCM.

Lemma 1 *If G has a two-layer drawing with at most k crossings then $|E(G)| \leq |V(G)| + k - 1$.*

Proof The proof is by induction on k. The base case $k = 0$ is trivial since then G must be a forest. Note that a two-layer drawing of a cycle has at least one crossing. Suppose $k > 0$ and fix a two-layer drawing of G with at most k crossings. Let e be an edge that crosses some edges in the drawing. We apply the induction hypothesis to $G \setminus \{e\}$ to have $|E(G)| - 1 \leq |V(G)| + (k - 1) - 1$ and hence $|E(G)| \leq |V(G)| + k - 1$. □

We also need the following lemma due to Dujmović and Whitesides [5].

Lemma 2 (Lemma 1 in [5]) *Suppose u and v are distinct vertices in Y such that $c(u,v)=0$. Then we have $u <_\pi v$ in every optimal permutation on Y, unless we also have $c(v,u)=0$.*

Motivated by this lemma, let us call an unordered pair $\{u,v\}$ of distinct vertices in Y *forced to* (u,v) if $c(u,v)=0$ and $c(v,u)>0$. We say that it is *forced* if it is forced either to (u,v) or to (v,u). We say such an unordered pair $\{u,v\}$ is *orientable* if $c(u,v)>0$ and $c(v,u)>0$; *free* if $c(u,v)=0$ and $c(v,u)=0$. Let us note that pair $\{u,v\}$ is free if u and v are false twins, i.e., $N(u)=N(v)$, and moreover this common neighbor set is a singleton. We use the above lemma in the following form.

Corollary 1 *Let π be an optimal permutation on Y and let u,v be distinct vertices in Y. If $\{u,v\}$ is forced to (u,v) then we have $u <_\pi v$. If $\{u,v\}$ is free, then the permutation π' obtained from π by swapping the positions of u and v is also optimal.*

Proof The first part is a restatement of Lemma 2. Suppose pair $\{u,v\}$ is free. This means that $N(u)=N(v)=\{x\}$ for some $x\in X$. Clearly, u and v are indistinguishable in our problem. Therefore the second part holds. □

Since each orientable pair contributes at least one crossing in any ordering of Y, the following is obvious.

Proposition 1 *Assuming that the given OSCM instance is feasible, the number of orientable pairs is at most k.*

The following is an outline of our algorithm.

1. If $m \geq n+k$ then stop with "No".
2. Construct the interval system $\mathcal{I}$ described in the introduction and another interval system $\mathcal{J}$, which inherits the property of $\mathcal{I}$ that each intersecting pair of intervals contributes at least one crossing in the drawing and is designed to allow degree-1 vertices and to facilitate dynamic programming. The construction of these interval systems can be done in $O(m)$ time. See Sect. 3.
3. If $\mathcal{J}$ contains more than k intersecting pairs, stop with "No".
4. Precompute $c(u,v)$ and $c(v,u)$ for all orientable pairs of vertices $u,v\in Y$. This can be done in $O(n+k)$ total time. If infeasibility is detected during this precomputation, stop immediately with "No". See Sect. 4 for details of this step.
5. Compute $\mathrm{opt}(Y)$ by a dynamic programming algorithm based on the interval system $\mathcal{J}$. In this computation, the values of $c(u,v)$ are needed only for orientable pairs. If infeasibility is detected during this computation, stop immediately with "No". If the computation is successful and $\mathrm{opt}(Y)\leq k$ then answer "Yes"; otherwise answer "No". This step can be performed in $O(k2^{\sqrt{2k}}+n)$ time. See Sect. 5.

The total running time of the algorithm is dominated by the dynamic programming part and is $O(k2^{\sqrt{2k}}+n)$.

It is straightforward to augment the dynamic programming tables so that, when the last step is complete, an optimal permutation on Y can be constructed. We note that this optimal solution is correct even if $\mathrm{opt}(Y) > k$, as long as the dynamic programming computation is completed.

3 Interval Systems

We refer to the interval system $\mathcal{I} = \{I_y \mid y \in Y\}$ defined in the introduction as the *naive interval system*. Recall $I_y = [l_y, r_y)$, where l_y is the smallest neighbor of y and r_y is the largest neighbor of y, with respect to the total order $<$ on X. The construction of $\mathcal{I}$ can be done in $O(m)$ time: we scan X in the given total order $<$ and, as we scan $x \in X$, we do necessary book-keeping to record l_y and r_y for each $y \in N(x)$.

We need another system $\mathcal{J} = \{J_y \mid y \in Y\}$ of intervals which is slightly more complicated than the naive system. This complication comes from the need to deal with vertices in Y of degree 1 and to facilitate dynamic programming. The system $\mathcal{J}$ will satisfy the following conditions. Let $J_y = [a_y, b_y]$ for each $y \in Y$.

J1 For each y, a_y and b_y are integers satisfying $1 \le a_y < b_y \le 2|Y|$.
J2 For each t, $1 \le t \le 2|Y|$, there is a unique vertex $y \in Y$ such that $a_y = t$ or $b_y = t$.
J3 If $b_u < a_v$ for $u, v \in Y$, then $c(u, v) = 0$.

Conditions J1 and J2 are for the sake of the ease of dynamic programming described in the next section, while condition J3 is the essential property that $\mathcal{J}$ shares with the naive interval system.

Let $P = \{(y, l_y, 0) \mid y \in Y\} \cup \{(y, r_y, 1) \mid y \in Y\}$. For each $y \in Y$, $(y, l_y, 0)$ and $(y, r_y, 1)$ are intended to represent the left and the right ends of the interval J_y, respectively. Our strategy is to define a total order on P and let a_y (b_y, resp.) be the rank of $(y, l_y, 0)$ ($(y, r_y, 1)$, resp.) in this total order. For each $p \in P$, we denote by $y(p)$, $x(p)$, and $i(p)$ the first, second, and the third element of p.

The total order $<$ on P is defined as follows. This definition is based on the given total order $<$ on X.

The order is primarily based on the second component: if $x(p) < x(q)$ then $p < q$. For each $x \in X$, let $P_x = \{p \in P \mid x(p) = x\}$. To describe the order $<$ within each P_x, we first partition P_x into three subsets: $P_x^1 = \{p \in P_x \mid d(y(p)) > 1, i(p) = 1\}$, $P_x^2 = \{p \in P_x \mid d(y(p)) = 1\}$, and $P_x^3 = \{p \in P_x \mid d(y(p)) > 1, i(p) = 0\}$. We let $p < q$ if $p \in P_x^i$ and $q \in P_x^j$ with $i < j$. The order of elements within P_x^1 and within P_x^3 is arbitrary. Elements of P_x^2 come in pairs: $(x, y, 0)$ and $(x, y, 1)$, where $y \in N(x)$ with $d(y) = 1$. The order in P_x^2 is chosen so that $(x, y, 0) < (x, y, 1)$ for each pair and these pairs are not interleaved: we do not have y, y' with $(x, y, 0) < (x, y', 0) < (x, y, 1)$ or $(x, y, 0) < (x, y', 1) < (x, y, 1)$.

Now we list the elements of P as $p_1, \ldots, p_{2|Y|}$ in the total order just defined. For each $y \in Y$, we let $a_y = t$ where t is such that $p_t = (y, l_y, 0)$ and $b_y = t$ where t is such that $p_t = (y, r_y, 1)$. This completes the description of the interval system $\mathcal{J}$.

The construction of $\mathcal{J}$ can also be done in $O(m)$ time. The set P is constructed as a list by scanning X. This list is already sorted in the primary key x. The partitioning of P_x into P_x^1, P_x^2, P_x^3 and the pairing in P_x^2 are done in $O(d(x))$ time for each x and hence in $O(m)$ time for all $x \in X$.

Proposition 2 *The system $\mathcal{J}$ of intervals defined above satisfies conditions* J1, J2, J3.

Proof From the construction of $\mathcal{J}$ based on the total order on P, it is obvious that J1 and J2 are satisfied. That J3 is satisfied is also obvious, as $b_u < a_v$ for $u, v \in Y$ implies that $r_u \leq l_v$ in X. □

We restate Corollary 1 using our interval system $\mathcal{J}$. We say that a permutation π on $U \subseteq Y$ is *consistent with* $\mathcal{J}$ if $b_u < a_v$ implies $u <_\pi v$ for every pair $u, v \in U$.

Lemma 3 *Let U be an arbitrary subset of Y. There is an optimal permutation π on U that is consistent with $\mathcal{J}$.*

Proof Let π be an optimal permutation on U. For each $x \in X$, let U_x denote the set of vertices in U that are adjacent to x but no other vertices in X. For each x, each pair of distinct vertices in U_x is free and therefore, applying Corollary 1 to the instance where Y is replaced by U, we may assume that π restricted on U_x is consistent with $\mathcal{J}$. Now, let u, v be arbitrary vertices in U and suppose $b_u < a_v$. By property J3 of $\mathcal{J}$, we have $c(u, v) = 0$. If $c(v, u) > 0$ then $\{u, v\}$ is forced to (u, v) and we must have $u <_\pi v$. Otherwise, $\{u, v\}$ is free and $u, v \in U_x$ for some $x \in X$. By the assumption on π above, we have $u <_\pi v$ in this case as well. □

4 Computing the Crossing Numbers

Dujmović and Whitesides [5] give an algorithm for computing the crossing numbers $c(u, v)$ for all pairs $\{u, v\}$ in $O(kn^2)$ time. We spend $O(n + k)$ time for precomputing $c(u, v)$ for all orientable pairs, ignoring forced and free pairs.

We use the naive interval system $\mathcal{I} = \{I_y \mid y \in Y\}$, where $I_y = [l_y, r_y)$, in this computation.

For each $y \in Y$ and $x \in X$, let $d^{<x}(y) = |\{z \in N(y) \mid z < x\}|$ and $d^{\leq x}(y) = |\{z \in N(y) \mid z \leq x\}|$. Then, we have $c(u, v) = \sum_{x \in N(u)} d^{<x}(v)$.

It turns out helpful to decompose the above sum as follows.

$$c(u, v) = \left[\sum_{x \in N(u), l_v < x \leq r_v} d^{<x}(v) \right] + d(v) \cdot \big(d(u) - d^{\leq r_v}(u) \big). \tag{1}$$

For each $x \in X$, let $Y_x = \{y \in Y \mid l_y < x < r_y\}$ be the set of vertices in Y whose corresponding intervals strictly contain x.

In the following, we call an ordered pair (u, v) *orientable* if the corresponding unordered pair is orientable. We evaluate these sums simultaneously for all orientable pairs (u, v), using a counter $c[u, v]$ for each pair. We represent these counters by a $|Y| \times |Y|$ two-dimensional array. Since we cannot afford to initialize all of its elements, we initialize $c[u, v]$ to 0 only for orientable pairs (u, v). Our algorithm proceeds as follows.

1. Scan X in the total order $<$, maintaining Y_x as we scan x. When we scan $x \in X$, we initialize $c[u, v]$ to 0 for each $u \in N(x)$ and each $v \in Y_x$.

2. Scan X again in the total order $<$, maintaining Y_x and $d^{<x}(y)$ for each $y \in Y$, as we scan x. Suppose we are scanning $x \in X$. For each $u \in N(x)$ and each $v \in Y_x$, we add $d^{<x}(v)$, the summand in (1), to $c[u, v]$. Moreover, for each $u \in Y_x$ and $v \in N(x)$ such that $r_v = x$, we add $d(v) \cdot (d(u) - d^{\leq x}(u))$, the second term in (1), to $c[u, v]$.

Lemma 4 *Assuming that the given OSCM instance is feasible, the running time of the above algorithm is $O(n + k)$.*

Proof Assume that the given OSCM instance is feasible. In both scans, we maintain Y_x as a doubly linked list with each entry for vertex y having a back-pointer from y. This allows the addition to and removal from Y_x of a single vertex to be done in $O(1)$ time. Therefore, Y_x can be maintained in $O(m) = O(n + k)$ total time for each scan. In the first scan, the number of times the counters are initialized is at most $2k$, since each pair $(u, (x, v))$ with $u, v \in Y$ and $(x, v) \in E(G)$ that leads to an initialization of $c[u, v]$ contributes at least one crossing in any ordering of Y and with each crossing at most two such pairs are associated. Therefore the running time of the first scan is $O(n + k)$. The second scan also consumes $O(n + k)$ time for maintaining Y_x and updating those counters. It remains to show that the maintenance of $d^{<x}(v)$ for each $v \in Y$ can be done in $O(n + k)$ total time. Using a simple counter for each $v \in Y$, which is incremented when each $x \in X$ adjacent to v is scanned, the maintenance of $d^{<x}(v)$ for all $v \in Y$ can be done in $O(m) = O(n + k)$ time. □

To control the running time for infeasible instances, we count the number of times the initialization of a counter occurs in the first scan. As soon as the number exceeds $2k$, we stop the computation and report infeasibility.

5 Dynamic Programming

In this section, we describe our dynamic programming algorithm for computing $\text{opt}(Y)$. Owing to the previous section, we assume in this section that $c(u, v)$ and $c(v, u)$ are available for all orientable pairs $\{u, v\}$.

We use the interval system $\mathcal{J} = \{J_y \mid y \in Y\}$ we have defined in Sect. 3, where $J_y = [a_y, b_y]$.

For each t, $1 \leq t \leq 2|Y|$, let $L_t = \{y \in Y \mid b_y \leq t\}$, $M_t = \{y \in Y \mid a_y \leq t < b_y\}$, and $R_t = \{y \in Y \mid t < a_y\}$. Note that

1. if $t = a_y$ for some $y \in Y$ then $L_t = L_{t-1}$, $M_t = M_{t-1} \cup \{y\}$, and $R_t = R_{t-1} \setminus \{y\}$;
2. if $t = b_y$ for some $y \in Y$ then $L_t = L_{t-1} \cup \{y\}$, $M_t = M_{t-1} \setminus \{y\}$, and $R_t = R_{t-1}$.

In other words, when interval J_y opens at t, y is moved from the “right set” to the “middle set”; when it closes at t, y is moved from the “middle set” to the “left set”.

For each integer t, $1 \leq t \leq 2|Y|$, we compute the following and store the results in a table: (1) $c(L_t, \{y\})$, for each $y \in M_t$; (2) $\text{opt}(L_t \cup S)$, for each $S \subseteq M_t$.

The recurrences for (1) are straightforward. The base case is $c(L_1, \{y\}) = 0$, where $L_1 = \emptyset$ and y is the unique element of M_1. Let $2 \leq t \leq 2|Y|$ and suppose first that

$t = a_y$ for some $y \in Y$. Note that $L_t = L_{t-1}$ and $M_t \setminus M_{t-1} = \{y\}$. Therefore, for $v \in M_t \setminus \{y\}$, we have $c(L_t, \{v\}) = c(L_{t-1}, \{v\})$. Since $b_u < a_y$ for each $u \in L_t = L_{t-1}$, we have $c(L_t, \{y\}) = 0$. Suppose next that $t = b_y$ for some $y \in Y$. Note that $L_t \setminus L_{t-1} = \{y\}$ and $M_{t-1} \setminus M_t = \{y\}$ in this case. For each $v \in M_t$, we have $c(L_t, \{v\}) = c(L_{t-1} \cup \{y\}, \{v\}) = c(L_{t-1}, \{v\}) + c(y, v)$. Note that pair (y, v) is orientable, as $y, v \in M_{t-1}$, and hence $c(y, v)$ is available. Thus, in either case, the table entries of type (1) for t can be computed from the entries for $t - 1$ in $O(h)$ time, where $h = |M_t|$.

We now turn to the recurrences for type (2) entries. The following lemma helps us to compute type (2) entries.

Lemma 5 *There is an optimal permutation π on $L_t \cup S$ whose last vertex belongs to S.*

Proof By Corollary 1, we may assume that π consistent with $\mathcal{J}$. Since π is consistent with $\mathcal{J}$ and $b_u < a_y$ for all $u \in L_t$, we have $u <_\pi y$ for all $u \in L_t$. Then, there must be some $v \in S$ such that either $y <_\pi v$ or $y = v$, that is, the last vertex of π is contained in S. □

Lemma 6 *Let $1 \le t \le 2|Y|$ and let $h = |M_t|$. Given a table that lists the values of $c(L_t, \{v\})$ for every $v \in M_t$ and $\mathrm{opt}(L_{t-1} \cup S)$ for every $S \subseteq M_{t-1}$, we can compute in $O(h2^h)$ time the values of $\mathrm{opt}(L_t \cup S)$ for all $S \subseteq M_t$.*

Proof First, we compute $c(S, \{x\})$ for $x \in M_t$ and for $S \subseteq M_t \setminus \{x\}$, and store the values in a table. Obviously, this computation is done in $O(h2^h)$ time.

Suppose $t = b_y$ for some $y \in Y$. Then $L_t = L_{t-1} \cup \{y\}$. Since $(S \cup \{y\}) \subseteq M_{t-1}$, for each $S \subseteq M_t$, $\mathrm{opt}(L_t \cup S) = \mathrm{opt}(L_{t-1} \cup (S \cup \{y\}))$ is available in the table. Suppose $t = a_y$ for some $y \in Y$. Then, we have $L_t = L_{t-1}$ and $M_t = M_{t-1} \cup \{y\}$. Let $S \subseteq M_t$. If $y \notin S$, then $\mathrm{opt}(L_t \cup S) = \mathrm{opt}(L_{t-1} \cup S)$ is available in the table, since $S \subseteq M_{t-1}$. Suppose $y \in S$. By Lemma 5, there is an optimal permutation π on $L_t \cup S$ whose last vertex belongs to S. Then we have

$$\begin{aligned}\mathrm{opt}(L_t \cup S) &= \min_{x \in S}\bigl\{\mathrm{opt}\bigl(L_t \cup S \setminus \{x\}\bigr) + c\bigl(L_t \cup S \setminus \{x\}, \{x\}\bigr)\bigr\} \\ &= \min_{x \in S}\bigl\{\mathrm{opt}\bigl(L_t \cup S \setminus \{x\}\bigr) + c\bigl(L_t, \{x\}\bigr) + c\bigl(S \setminus \{x\}, \{x\}\bigr)\bigr\}.\end{aligned}$$

The second and third terms are in the tables. Hence, we can compute the values $\mathrm{opt}(L_t \cup S)$ for all $S \subseteq M_t$ by a standard dynamic programming approach in $O(h2^h)$ time. □

The dynamic programming gives us the optimal solution $\mathrm{opt}(Y)$ since $L_{2|Y|} = Y$.

Each pair of vertices in M_t contributes at least one crossing in any ordering of Y. Therefore, for the given instance to be feasible, we have $h(h-1)/2 \le k$ and hence $h \le \sqrt{2k} + 1$, where $h = |M_t|$. Using this bound and an observation that $|M_t| \ge 2$ for at most k values of t, it is straightforward to derive a bound of $O(k^{3/2}2^{\sqrt{2k}} + n)$ on the running time of the entire dynamic programming computation. For a tighter analysis, we need the following lemma.

Lemma 7 *Assume that $\mathcal{J}$ has at most k intersecting pairs of intervals. Let $H = \lceil\sqrt{2k}\rceil + 1$ and, for $2 \le h \le H$, let c_h denote the number of values of t with $|M_t| = h$. Then, we have $c_h \le 2^{H-h+2}$ for $2 \le h \le H$.*

Proof Fix $h, 2 \le h \le H$. Let $t_1 < t_2 < \cdots < t_{c_h}$ be the members of $\{t \mid |M_t| = h\}$. The first set M_{t_1} contains $h(h-1)/2$ pairs of vertices each corresponding to an intersecting pair of intervals. We claim that M_{t_i} for each $2 \le i \le c_h$ contributes at least $h-1$ new intersecting pairs of intervals. This is obvious if $M_{t_i} \ne M_{t_{i-1}}$. If $M_{t_i} = M_{t_{i-1}}$, then $M_{t_{i-1}+1}$ must contain a vertex not in $M_{t_{i-1}}$ that contributes h new intersecting pairs. Therefore we have $h(h-1)/2 + (c_h - 1)(h-1) \le k$. Solving this inequality for c_h, we have $c_h \le k/(h-1) - h/2 + 1$.

If $H - h + 2 \ge \log_2 k$, the claimed bound $c_h \le 2^{H-h+2}$ is obvious since then $2^{H-h+2} \ge k$. So suppose $j = H - h + 2 < \log_2 k$. Then, we have $(H - j + 1)(H + 2j) = H^2 + H + j(H - 2j + 2) > H^2 > 2k$ since $H \ge \sqrt{2k} + 1 \ge 2(\log_2 k - 1)$. Therefore, we have $k/(h-1) = k/(H - j + 1) < (H + 2j)/2 = H/2 + j$ and hence

$$c_h < H/2 + j - (H - j + 2)/2 + 1 = \frac{3j}{2} \le 2^j.$$

This is the claimed bound. □

Lemma 8 *Assume that the given OSCM instance is feasible, the total running time of the dynamic programming algorithm based on Lemma 6 is $O(k2^{\sqrt{2k}} + n)$.*

Proof Let H and c_h, $2 \le h \le H$, be as in Lemma 7. Then, from Lemma 6, it follows that the total running time of the dynamic programming algorithm is $O(\sum_{2 \le h \le H} c_h h 2^h + n)$. By Lemma 7, the first term is bounded by

$$\begin{aligned}\sum_{2 \le h \le H} c_h h 2^h &\le \sum_{2 \le h \le H} 2^{H-h+2} h 2^h \\ &= \sum_{2 \le h \le H} h 2^{H+2} \\ &\le 2^{H+1} H(H+1) \\ &= O\big(k 2^{\sqrt{2k}}\big)\end{aligned}$$

and therefore we have the claimed bound. □

To control the running time for infeasible instances, we compute c_h for each $2 \le h \le H$ and, if c_h exceeds the proved bound, we immediately stop the computation as we have detected infeasibility.

6 Proof of Theorem 2

Impagliazzo, Paturi, and Zane [10] have shown that, under ETH, there is no $2^{o(m)}$ time algorithm for 3-SAT where m is the number of clauses. We confirm in the lemmas below that the standard chain of reductions from 3-SAT to OSCM showing the

NP-completeness of sparse OSCM is such that the number r of edges in the resulting OSCM instance is linear in the number m of clauses of the input 3-SAT instance. Theorem 2 follows, since a $2^{o(\sqrt{k})}n^{O(1)}$ time algorithm for OSCM would imply a $2^{o(m)}$ time algorithm for 3-SAT as $k \leq r^2 = O(m^2)$ in our reduction and hence would violate ETH.

Lemma 9 [11] *There is a polynomial time algorithm that, given a* 3*-CNF formula* ϕ *with* n *variables and* m *clauses, computes a graph* G_ϕ *such that* ϕ *is satisfiable if and only if* G_ϕ *has a vertex cover of size at most* $n + 2m$*. Moreover,* G_ϕ *has at most* $2n + 3m$ *vertices and at most* $n + 6m$ *edges.*

Proof (sketch) Let $X = \{x_1, x_2, \dots, x_n\}$ be the set of variables of ϕ and $C = \{c_1, c_2, \dots, c_m\}$ be the set of clauses of ϕ. We construct the graph G_ϕ as follows. For each variable $x_i \in X$, G_ϕ contains a pair of vertices $u_i, \bar{u_i}$ and an edge $\{u_i, \bar{u_i}\}$. For each clause $c_i \in C$, G_ϕ contains a cycle $C_i = \{v_i^1, v_i^2, v_i^3\}$ of length 3. If the kth literal of c_j is x_i for some i, then we add an edge $\{v_j^k, u_i\}$ to G_ϕ. Otherwise, kth literal of c_j is $\bar{x_i}$ for some i, then we add an edge $\{v_j^k, \bar{u_i}\}$ to G_ϕ. G_ϕ has $2n + 3m$ vertices and $n + 6m$ edges as claimed. The correctness proof of this reduction can be found in [11]. □

Lemma 10 [11] *There is a polynomial time algorithm that, given a graph* G *with* n *vertices and* m *edges, and an integer* k*, computes a directed graph* D *such that* G *has a vertex cover of size at most* k *if and only if* D *has a feedback arc set of size at most* k*. Moreover,* D *contains at most* $2n$ *vertices and at most* $n + 2m$ *arcs.*

Proof (sketch) Let $V = \{u_1, u_2, \dots, u_n\}$ be the set of vertices of G. We construct the directed graph D as follows. For each vertex $u_i \in V$, D contains a pair of vertices v_i, w_i together with an arc (v_i, w_i). For each edge $\{u_i, u_j\}$ of G, D contains a pair of arcs (w_j, v_i) and (w_i, v_j). G_ϕ has $2n$ vertices and $n + 2m$ arcs as claimed. The correctness proof of this reduction can be found in [11]. □

Finally, we describe a reduction from the feedback arc set problem to OSCM. Eades and Wormald [7] first gave such a polynomial time reduction. But the reduced OSCM instance has $\Theta(nm)$ edges, where n is the number vertices and m is the number of arcs, of a directed feedback arc set instance. For our purposes, we need a reduction to sparse OSCM due to [16].

Lemma 11 (Sect. 4 in [16]) *There is a polynomial time algorithm that, given an integer* t *and a directed graph with* n *vertices and* m *edges, computes an OSCM instance* $I = (G, X, Y, <, k)$ *with* $5(n + m)$ *vertices and* $4(n + m)$ *edges such that* D *has a feedback arc set of size at most* t *if and only if* I *is feasible.*

Proof (sketch) We denote the vertex set of D by V and the arc set of D by A. Let X_i, $1 \leq i \leq 5$, be disjoint sets each of cardinality $|V| + |A|$ and let ϕ_i be a bijection from $V \cup A$ to X_i for $1 \leq i \leq 5$. Let $G = (X \cup Y, E)$ be a bipartite graph with bipartition

(X, Y) where

$$X = X_1 \cup X_2 \cup X_3 \cup X_4,$$
$$Y = X_5,$$
$$E = \big\{\big\{\phi_i(w), \phi_5(w)\big\} \mid 1 \le i \le 4, w \in V \cup A\big\}.$$

Clearly, $|X \cup Y| = 5(n+m)$ and $|E| = 4(n+m)$. Then we construct the OSCM instance $I = (G, X, Y, <, k)$ where the total order $<$ and the crossing number k are appropriately defined. We omit the definition of $<$ and k together with the correctness proof of the reduction, which can be found in [16]. □

To summarize these reductions, we obtain the following lemma.

Lemma 12 *There is a polynomial time algorithm that, given a* 3*-CNF formula* ϕ *with* m *clauses, computes an OSCM instance* $I = (G, X, Y, <, k)$ *such that* ϕ *is satisfiable if and only if* I *is feasible. Moreover,* G *contains* $O(m)$ *edges.*

7 Future Work

Fernau et al. [8] gave a PTAS for OSCM using the PTAS of Kenyon-Mathieu and Schudy [13] for weighted FAST. It would be interesting to know whether our approach can be used to develop a PTAS. Some exponential space dynamic programming algorithms can be transformed into polynomial space algorithms with additional time consumption. Our dynamic programming described in Sect. 5 uses exponential space. It is possible that this algorithm can also be turned into a polynomial space algorithm, but this does not appear straightforward and probably is another research topic. We expect that our algorithm performs well on practical instances since these instances are typically sparse and have small crossing numbers. We already have a partial implementation with positive preliminary results. For more extensive evaluation, we need to fully implement the algorithm and evaluate it on benchmark instances available in the graph drawing community.

Acknowledgements We are grateful to anonymous referees for valuable comments about the optimality of our result under ETH and the suggestions for improving the presentation of the paper. We also thank Marek Cygan for helpful ideas for the improvement of our running time.

References

1. Ailon, N., Charikar, M., Newman, A.: Aggregating inconsistent information: ranking and clustering. J. ACM **55**(5), 1–23 (2008)
2. Alon, N., Lokshtanov, D., Saurabh, S.: Fast FAST. In: Proceedings of the 36th International Colloquium on Automata, Languages and Programming, ICALP 2009, Part I. Lecture Notes in Computer Science, vol. 5555, pp. 49–58. Springer, Berlin (2009)

3. Bodlaender, H.: A tourist guide through treewidth. Acta Cybern. **11**, 1–23 (1993)
4. Downey, R.G., Fellows, M.R.: Parameterized Complexity. Springer, Berlin (1998)
5. Dujmović, V., Whitesides, S.: An efficient fixed parameter tractable algorithm for 1-sided crossing minimization. Algorithmica **40**(1), 15–31 (2004)
6. Dujmović, V., Fernau, H., Kaufmann, M.: Fixed parameter algorithms for one-sided crossing minimization revisited. J. Discrete Algorithms **6**(2), 313–323 (2008)
7. Eades, P., Wormald, N.C.: Edge crossings in drawings of bipartite graphs. Algorithmica **11**(4), 379–403 (1994)
8. Fernau, H., Fomin, F.V., Lokshtanov, D., Mnich, M., Philip, G., Saurabh, S.: Ranking and drawing in subexponential time. In: Proceedings of the 21st International Workshop on Combinatorial Algorithms, IWOCA 2010. Lecture Notes in Computer Science, vol. 6460, pp. 337–348. Springer, Berlin (2010)
9. Impagliazzo, R., Paturi, R.: On the complexity of k-SAT. J. Comput. Syst. Sci. **62**, 367–375 (2001)
10. Impagliazzo, R., Paturi, R., Zane, F.: Which problems have strongly exponential complexity? J. Comput. Syst. Sci. **63**, 512–530 (2001)
11. Karp, R.M.: Reducibility among combinatorial problems. In: Complexity of Computer Computations, pp. 85–103. Plenum, New York (1972)
12. Karpinski, M., Schudy, W.: Faster algorithms for feedback arc set tournament, Kemeny rank aggregation and betweenness tournament. In: Proceedings of the 21st International Symposium on Algorithms and Computation, ISAAC 2010, Part I. Lecture Notes in Computer Science, vol. 6506, pp. 3–14. Springer, Berlin (2010)
13. Kenyon-Mathieu, C., Schudy, W.: How to rank with few errors. In: Proceedings of the 39th Annual ACM Symposium on Theory of Computing, STOC 2007, pp. 95–103. ACM, New York (2007)
14. Kobayashi, Y., Tamaki, H.: A fast and simple subexponential fixed parameter algorithm for one-sided crossing minimization. In: Proceedings of the 20th Annual European Symposium on Algorithms, ESA 2012. Lecture Notes in Computer Science, vol. 7501, pp. 683–694. Springer, Berlin (2012)
15. Lokshtanov, D., Marx, D., Saurabh, S.: Lower bounds based on the exponential time hypothesis. Bull. Eur. Assoc. Theor. Comput. Sci. **105**, 41–72 (2011)
16. Muñoz, X., Unger, W., Vrt'o, I.: One sided crossing minimization is NP-hard for sparse graphs. In: Proceedings of the 9th International Symposium on Graph Drawing, GD 2002. Lecture Notes in Computer Science, vol. 2265, pp. 115–123. Springer, Berlin (2002)
17. Nagamochi, H.: On the one-sided crossing minimization in a bipartite graph with large degree. Theor. Comput. Sci. **332**, 417–446 (2005)
18. Nagamochi, H.: An improved bound on the one-sided minimum crossing number in two-layered drawings. Discrete Comput. Geom. **33**(4), 569–591 (2005)
19. Sugiyama, K., Tagawa, S., Toda, M.: Methods for visual understanding of hierarchical system structures. IEEE Trans. Syst. Man Cybern. **11**(2), 109–125 (1981)