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

Yasuaki Kobayashi[1,a)] Hisao Tamaki[1,b)]

Abstract: We give a subexponential fixed parameter algorithm for one-sided crossing minimization. It runs in $O(3^{\sqrt{2k}} + n)$ 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 tractable, Graph 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.
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 [8], even for sparse graphs [18]. On the positive side, Dujmović and Whitesides [7] showed that OSCM is fixed parameter tractable [5], 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 $O(1.4656^k + kn^2)$. Very recently, Fernau *et al.* [9] 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. Karpinski and Schudy [13] 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 [17] and $(1.2964 + 12/(\delta - 4))$-approximate algorithm when the minimum degree δ of a vertex in Y is at least 5 [16].

Our main result in this paper is the following.

Theorem 1.1. *OSCM can be solved in $O(3^{\sqrt{2k}} + n)$ 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 $O(3^{\sqrt{2k}})$ 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 [9], the only previously known subexponential algorithm for OSCM. Note that the running time of our algorithm is linear in n as long as $k \leq \frac{\log_3^2 n}{2} + O(1)$. 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) [11],

[1] Meiji University, Kawasaki, Japan 214-8571
[a)] yasu0207@cs.meiji.ac.jp
[b)] tamaki@cs.meiji.ac.jp

a well-known complexity assumption which states that, for each $k \geq 3$, 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).

Theorem 1.2. *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 [7] 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 [9] 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 [6], [7]. 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 [6], [7], 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 $[l_y, r_y) = \{x \in X \mid l_y \leq x < r_y\}$ 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 Section 3 does not need this assumption. A key observation in [7] (see Lemma 2.2 in the present paper), is that if $r_u \leq l_v$ 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 [6], [7] 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 Section 4 for details.

The proofs are omitted in this version and can be found in [14].

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

2. Preliminaries and outline of the algorithm

In this section, we give some preliminaries and outline our algorithm claimed in Theorem 1.1. 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 $|E(V)|$ 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. When U and V are disjoint finite sets, $\pi \in \Pi(U)$, and $\sigma \in \Pi(V)$, we denote by $\pi + \sigma$ the permutation on $U \cup V$ that is a concatenation of π and σ, the sequence consisting of π followed by σ.

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 $\mathrm{opt}(U) = \min\{c(\pi) \mid \pi \in \Pi(U)\}$. The goal of our algorithm is to decide if $\mathrm{opt}(Y) \leq k$.

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

Lemma 2.1. *If G has a two-layer drawing with at most k crossings then $|E(G)| \leq |V(G)| + k - 1$.*

We also need the following lemma due to Dujmović and Whitesides [7].

Lemma 2.2. (Lemma 1 in [7]) *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 is *orientable* if $c(u,v) > 0$ and $c(v,u) > 0$; *free* if $c(u,v) = 0$ and $c(v,u) = 0$. We use the above lemma in the following form.

Corollary 2.1. *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.*

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

Proposition 2.1. *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 Section 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 Section 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 $O(3^{\sqrt{2k}} + n)$ time. See Section 5.

The total running time of the algorithm is dominated by the dynamic programming part and is $O(3^{\sqrt{2k}} + n)$.

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 $1 \leq a_y < b_y \leq 2|Y|$.
J2 For each t, $1 \leq t \leq 2|Y|$, 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 3.1. *The system $\mathcal{J}$ of intervals defined above satisfies conditions J1, J2, J3.*

We restate Corollary 2.1 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 3.1. *Let U be an arbitrary subset of Y. There is an optimal permutation π on U that is consistent with $\mathcal{J}$.*

4. Computing the crossing numbers

Dujmović and Whitesides [7] 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 $d^{\le x}(y) = |\{z \in N(y) \mid z \le x\}|$. 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.

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

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 $d(v) \cdot (d(u) - d^{\le x}(u))$, the second term in (1), to $c[u,v]$.

Lemma 4.1. *Assuming that the given OSCM instance is feasible, the running time of the above algorithm is $O(n+k)$.*

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 $\mathrm{opt}(Y)$. 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 Section 3, where $J_y = [a_y, b_y]$.

A standard dynamic programming approach (see [4], [10], for example) gives us the following exponential upper bound on the complexity of computing $\mathrm{opt}(U)$, which we need for small subproblems.

Lemma 5.1. *Let $V \subseteq Y$ and assume that $c(u,v)$ is available in $O(1)$ time for each pair of distinct vertices $u, v \in V$. Then, $\mathrm{opt}(U)$ for all $U \subseteq V$ can be computed in $O(h2^h)$ total time where $h = |V|$.*

For each t, $1 \le t \le 2|Y|$, let $L_t = \{y \in Y \mid b_y \le t\}$, $M_t = \{y \in Y \mid a_y \le t < b_y\}$, 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, $1 \le t \le 2|Y|$, we compute the following and store the results in a table: (1) $c(L_t, \{y\})$, for each $y \in M_t$; (2) $\mathrm{opt}(L_t \cup S)$, 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 $2 \le t \le 2|Y|$ 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. Since $L_0 = \emptyset$, the base case $\mathrm{opt}(L_0 \cup \emptyset) = 0$ is trivial. To facilitate the induction steps, we define, for each $y \in Y$ and disjoint subsets Y_1, Y_2 of $Y \setminus \{y\}$,

$$\mathrm{opt}(Y_1, y, Y_2) = \min\{c(\pi) \mid \pi \in \Pi(Y_1 \cup \{y\} \cup Y_2), Y_1 <_\pi \{y\} <_\pi Y_2\},$$

where, by $U <_\pi V$, we mean $u <_\pi v$ for every $u \in U$ and every $v \in V$. In other words, $\mathrm{opt}(Y_1, y, Y_2)$ is the cost of the optimal permutation on $Y_1 \cup \{y\} \cup Y_2$ subject to the condition that it is of the form $\pi_1 + y + \pi_2$, where $\pi_1 \in \Pi(Y_1)$ and $\pi_2 \in \Pi(Y_2)$. Note that $\mathrm{opt}(Y_1, y, Y_2)$ can be computed by

$$\mathrm{opt}(Y_1, y, Y_2) = \mathrm{opt}(Y_1) + c(Y_1, Y_2) + c(Y_1, \{y\}) + c(\{y\}, Y_2) + \mathrm{opt}(Y_2).$$

Lemma 5.2. *Let $1 \le t \le 2|Y|$ and suppose $a_y = t$ for some $y \in Y$. For each $S \subseteq M_t$ with $y \in S$, we have*

$$\mathrm{opt}(L_t \cup S) = \min\{\mathrm{opt}(L_{t-1} \cup T, y, U) \mid T \cup U = S \setminus \{y\}, T \cap U = \emptyset\}.$$

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

Lemma 5.3. *Let $1 \le t \le 2|Y|$ and let $h = |M_t|$. Given a table that lists the values of $c(L_{t-1}, \{v\})$ for every $v \in M_{t-1}$ and $\mathrm{opt}(L_{t-1} \cup S)$ for every $S \subseteq M_{t-1}$, we can compute in $O(3^h)$ time the values of $c(L_t, \{v\})$ for all $v \in M_t$ and the values of $\mathrm{opt}(L_t \cup S)$ for all $S \subseteq M_t$.*

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 $O(k3^{\sqrt{2k}} + n)$ on the running time of the entire dynamic programming computation. For a tighter analysis, we need the following lemmas.

Lemma 5.4. *Assume that $\mathcal{J}$ has at most k intersecting pairs of*

intervals. Let $H = \lceil \sqrt{2k} \rceil + 1$ *and, for* $2 \leq h \leq H$*, let* c_h *denote the number of values of* t *with* $|M_t| = h$*. Then, we have* $c_h \leq 2^{H-h+2}$ *for* $2 \leq h \leq H$*.*

Lemma 5.5. *Assume that the given OSCM instance is feasible, the total running time of the dynamic programming algorithm based on Lemma 5.3 is* $O(3^{\sqrt{2k}} + n)$*.*

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

References

[1] Ailon, N., Charikar, M., and Newman, A.: Aggregating inconsistent information:Ranking and clustering. *Journal of the ACM, 55(5):Article No. 23, 2008*

[2] Alon, N., Lokshtanov, D. and Saurabh, S.: Fast FAST, In *Automata, Languages and Programming, 36th International Colloquium, ICALP 2009, Rhodes, Greece, July 5-12, 2009, Proceedings, Part I*, pp. 49–58, 2009.

[3] Bodlaender, H.: A Tourist Guide through Treewidth, *Acta Cybernetica*, Vol. 11, pp. 1–23, 1993.

[4] Bodlaender, H., Fomin, F., Kratsch, D. and Thilikos, D.: A Note on Exact Algorithms for Vertex Ordering Problems on Graphs, *Theory of Computing Systems*, Vol. 50(3), pp. 420–432, 2012.

[5] Downey, R. G. and Fellows, M. R.: *Parameterized Complexity*, Springer, 1998.

[6] Dujmović, V., Fernau, H. and Kaufmann, M.: Fixed parameter algorithms for one-sided crossing minimization revisited, *Journal of Discrete Algorithms*, Vol. 6(2), pp. 313–323, 2008.

[7] Dujmović, V. and Whitesides, S.: An Efficient Fixed Parameter Tractable Algorithm for 1-Sided Crossing Minimization, *Algorithmica*, Vol. 40(1), pp. 15–31, 2004.

[8] Eades, P. and Wormald, N. C.: Edge crossings in drawings of bipartite graphs, *Algorithmica*, Vol. 11(4), pp. 379–403, 1994.

[9] Fernau, H., Fomin, F. V., Lokshtanov, D., Mnich, M., Philip, G. and Saurabh, S.: Ranking and drawing in subexponential time, In *Proceedings of the 21st International Workshop On Combinatorial Algorithms*, IWOCA'10, pp. 337–348, 2010.

[10] Held, M. and Karp, R. M.: A dynamic programming approach to sequencing problems, *Journal of the Society for Industrial and Applied Mathematics*, Vol. 10, pp. 196-210, 1962.

[11] Impagliazzo, R. and Paturi, R.: On the complexity of k-SAT *Journal of Computer and System Sciences*, Vol. 62, pp. 367–375, 2001.

[12] Impagliazzo, R., Paturi, R. and Zane, F.: Which problems have strongly exponential complexity? *Journal of Computer and System Sciences*, Vol. 63, pp. 512–530, 2001.

[13] Karpinski, M. and 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'10, part I, pp. 3-14, 2010.

[14] Kobayashi, Y. and 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'12, pp. 683–694, 2012.

[15] Lokshtanov, D., Marx, D. and Saurabh, S.: Lower bounds based on the Exponential Time Hypothesis, The Complexity Column by Arvind V., *Bulletin of the EATCS*, pp. 41–72, 2011.

[16] Nagamochi, H.: On the one-sided crossing minimization in a bipartite graph with large degree, *Theoretical Computer Science*, Vol. 332, pp. 417–446, 2005.

[17] Nagamochi, H.: An improved bound on the one-sided minimum crossing number in two-layered drawings, *Discrete and Computational Geometry*, Vol. 33(4), pp. 569–591, 2005.

[18] Muñoz, X., Unger, W. and Vrt'o, I.: One sided crossing minimization is NP-hard for sparse graphs, In *Revised Papers from the 9th International Symposium on Graph Drawing*, GD'01, pp. 115–123, 2002.

[19] Sugiyama, K., Tagawa, S. and Toda, M.: Methods for visual understanding of hierarchical system structures., *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 11(2), pp. 109–125,1981.