MINIMIZING CROSSINGS IN CONSTRAINED TWO-SIDED CIRCULAR GRAPH LAYOUTS

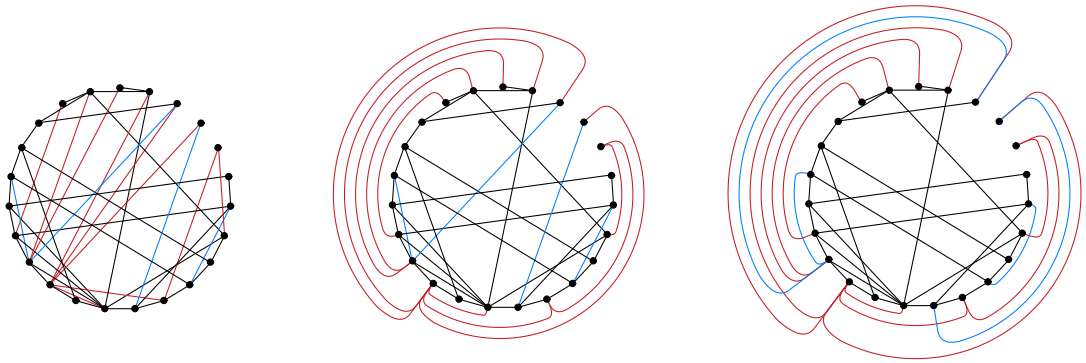
Fabian Klute, Martin Nöllenburg**

Abstract. Circular graph layout is a popular drawing style, in which vertices are placed on a circle and edges are drawn as straight chords. Crossing minimization in circular layouts is NP-hard. One way to allow for fewer crossings in practice are two-sided layouts, which draw some edges as curves in the exterior of the circle. In fact, one- and two-sided circular layouts are equivalent to one-page and two-page book drawings, i.e., graph layouts with all vertices placed on a line (the *spine*) and edges drawn in one or two distinct half-planes (the *pages*) bounded by the spine. In this paper we study the problem of minimizing the crossings for a fixed cyclic vertex order by computing an optimal k-plane set of exteriorly drawn edges for $k \geq 1$, extending the previously studied case $k = 0$. We show that this relates to finding bounded-degree maximum-weight induced subgraphs of circle graphs, which is a graph-theoretic problem of independent interest. We show NP-hardness for arbitrary k, present an efficient algorithm for $k = 1$, and generalize it to an explicit XP-time algorithm for any fixed k. For the practically interesting case $k = 1$ we implemented our algorithm and present experimental results that confirm its applicability.

1 Introduction

Circular graph layout is a popular drawing style to visualize graphs, e.g., in biology [25] or social network analysis [7], and circular layout algorithms [29] are included in standard graph layout software [20] such as yFiles, Graphviz, or OGDF. In a *circular graph layout* all vertices are placed on a circle, while the edges are drawn as straight-line chords of that circle, see Figure 1a. Minimizing the number of crossings between the edges is the main algorithmic problem for optimizing the readability of a circular graph layout. If the edges are drawn as chords, then all crossings are determined solely by the order of the vertices. By cutting the circle between any two vertices and straightening it, circular layouts immediately correspond to *one-page book drawings*, in which all vertices are drawn on a line (the *spine*) and all edges are drawn in one half-plane (the *page*) bounded by the spine. The concept of book drawings readily extends to k pages for any $k \in \mathbb{N}$, where every page is a separate half-plane bounded by the spine. More generally, given a graph $\mathcal{G}$ the book embedding problem [10] asks for the smallest number k of pages (also called the *page number* or *book thickness* of $\mathcal{G}$ [5]) such that $\mathcal{G}$ has a crossing-free book drawing on k pages. Chung et al. [10] showed that deciding whether a graph has book thickness k is NP-complete even for $k = 2$. For a given integer k the *book crossing number* [28] of a graph is the minimum number of edge crossings in a k-page book drawing of that graph. Finding a vertex order that minimizes the edge

* *Algorithms and Complexity Group, TU Wien, Austria*, {fklute,noellenburg}@ac.tuwien.ac.at



(a) One-sided layout with 125 crossings.

(b) Two-sided layout for $k = 0$ with 48 crossings.

(c) Two-sided layout for $k = 1$ with 30 crossings

Figure 1: Circular layouts of a graph $(\mathcal{G}, \pi)$ (23 vertices, 45 edges) computed by our algorithms.

crossings in a book drawing is NP-hard [26]. Heuristics and approximation algorithms have been studied in numerous papers. For example, Baur and Brandes [4] adapted the sifting approach from layered layouts and combined it with a simple greedy heuristic. Shahrokhi et al. [28] studied approximation algorithms for the book drawing problem and found a $O(\log^2 n)$ approximation algorithm. Klawitter et al. [22] evaluated a plethora of possible heuristics for minimizing the number of crossings in book drawings and also included a comprehensive literature review of heuristics for crossing minimization in book drawings.

Gansner and Koren [16] presented an approach to compute so-called improved circular layouts for a given input graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ in a three-step process. The first step computes a vertex order of $\mathcal{V}$ that aims to minimize the overall edge length of the drawing, the second step determines a crossing-free subset of edges that are drawn outside the circle to reduce edge crossings in the interior (comparable to the layout shown in Figure 1b), and the third step introduces edge bundling to save ink and reduce clutter in the interior. The layouts by Gansner and Koren draw edges inside and outside the circle and thus are called *two-sided circular layouts*. Again, it is easy to see that two-sided circular layouts are equivalent to two-page book drawings, where the interior of the circle with its edges corresponds to the first page and the exterior to the second page. Note, however, that the roles of the two pages are different. While in a book drawing both pages are equivalent, here one page has no restriction on the crossings, while the other is not allowed to have any.

Inspired by the approach of Gansner and Koren [16] we take a closer look at the second step of the above process, which, in other words, determines for a given cyclic vertex order an outerplane subgraph to be drawn outside the circle such that the remaining crossings of the chords are minimized. Gansner and Koren solve this problem in $O(|\mathcal{V}|^3)$ time.[1] In fact, the problem is equivalent to finding a maximum independent set in the corresponding circle graph $G^\circ = (V, E)$, which is the intersection graph of the chords (see Section 2 for further details). The maximum independent set problem in a circle graph can be solved in

[1] The paper claims $O(|\mathcal{V}|^2)$ time without a proof; the immediate running time of their algorithm is $O(|\mathcal{V}|^3)$.

$O(\ell)$ time [31], where $|\mathcal{E}| \leq \ell \leq |\mathcal{E}|^2$ is the sum of the lengths of the chords in the circle graph.[2]

Contribution. We generalize the above crossing minimization problem from finding an outerplane graph to finding an outer k-plane graph, i.e., we ask for an edge set to be drawn outside the circle such that none of the edges in this set has more than k crossings. This is motivated by the fact that, due to their detour in comparison to a straight-line chord, exterior edges are already harder to read and hence should not be further impaired by too many crossings. This is equivalent to asking for a page assignment of the edges in a two-page book drawing with a given fixed vertex order, such that in one of the two pages each edge has at most k crossings and the number of crossings is minimized, either in the whole drawing or only on one of the pages. For $k = 0$ this is exactly the same problem considered by Gansner and Koren [16] with a plane exterior edge set. An example for $k = 1$ is shown in Figure 1c. The general topic of k-planarity, i.e., studying drawings of non-planar graphs with at most k crossings per edge, and similar concepts extending the notion of planarity have attracted a lot of interest in graph drawing in recent years, see the surveys by Kobourov et al. [24] and Didimo et al. [13]. While the subclass of outer k-plane graphs is of independent interest [2, 8, 19] we use only the structure of the crossings to obtain our results.

We model the outer k-plane crossing minimization problem in two-sided circular layouts as a bounded-degree maximum-weight induced subgraph (BDMWIS) problem in the corresponding circle graph (see Section 2). The BDMWIS problem is a natural generalization of the weighted independent set problem (by setting the degree bound to $k = 0$), which was the basis for Gansner and Koren's approach [16]. It is itself a weighted special case of the bounded-degree vertex deletion problem [6, 12, 15], a well-studied algorithmic graph problem of independent interest. For arbitrary k we show NP-hardness of the BDMWIS problem in Section 3. Our algorithms in Section 4 are based on dynamic programming using interval representations of circle graphs. For the case $k = 1$, where at most one crossing per exterior edge is permitted, we solve the BDMWIS problem for circle graphs in $O(|\mathcal{E}|^4)$ time. We then generalize our algorithm and obtain a problem-specific XP-time algorithm for circle graphs and any fixed k, whose running time is $O(|\mathcal{E}|^{2k+2})$. We note that the pure existence of an XP-time algorithm can also be derived from applying a metatheorem of Fomin et al. [14] using counting monadic second order (CMSO) logic, but the resulting running times are far worse. Finally, in Section 5, we present the results of a first experimental study comparing the crossing numbers of two-sided circular layouts of various benchmark graphs for the cases $k = 0$ and $k = 1$.

2 Preliminaries

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a graph and π a cyclic order of $\mathcal{V}$. We arrange the vertices in order π on a circle C and draw edges as straight chords to obtain a (*one-sided*) *circular drawing* Γ, recall Figure 1a. Note that all crossings of Γ are fully determined by π: two edges cross if and only if their endpoints alternate in π. Our goal in this paper is to find a subset of edges

[2] See Section 4.2 for the precise definition of the total chord length ℓ.

to be drawn in the unbounded region outside C, with no more than k crossings per edge, in order to minimize either the total number of edge crossings or only the number of remaining edge crossings inside C.

More precisely, in a *two-sided circular drawing* Δ of $(\mathcal{G}, \pi)$ we still draw all vertices on a circle C in the order π, but we partition the edges into two disjoint sets $\mathcal{E}_1$ and $\mathcal{E}_2$ with $\mathcal{E}_1 \cup \mathcal{E}_2 = \mathcal{E}$. The edges in $\mathcal{E}_1$ are drawn as straight chords, while the edges in $\mathcal{E}_2$ are drawn as simple curves in the exterior of C, see Figure 1b and 1c. Asking for a set $\mathcal{E}_2$ that globally minimizes the crossings in Δ is equivalent to the NP-hard fixed linear crossing minimization problem in 2-page book drawings [27]. Hence we add the additional constraint that the exterior drawing induced by $\mathcal{E}_2$ is *outer k-plane*, i.e., each edge in $\mathcal{E}_2$ is crossed by at most k other edges in $\mathcal{E}_2$. The parameter k, which we assume to be fairly small, gives us control on the maximum number of crossings per exterior edge. Previous work [16] is limited to the case $k = 0$.

2.1 Problem transformation

Instead of working with a one-sided input layout Γ of $(\mathcal{G}, \pi)$ directly, we consider the corresponding *circle graph* $G^\circ = (V, E)$ of $(\mathcal{G}, \pi)$, i.e., the intersection graph of the chords in Γ. Accordingly, the vertex set V of G° has one vertex for each edge in $\mathcal{E}$ and two vertices $u, v \in V$ are connected by an edge (u, v) in E if and only if the chords corresponding to u and v cross in Γ, i.e., their endpoints alternate in π. The number of vertices $|V|$ of G° thus equals the number of edges $|\mathcal{E}|$ of $\mathcal{G}$ and the number of edges $|E|$ of G° equals the number of crossings in Γ. Moreover, the degree $\deg(v)$ of a vertex v in G° is the number of crossings of the corresponding edge in Γ.

Next we show that we can reduce our outer k-plane crossing minimization problem in two-sided circular layouts of $(\mathcal{G}, \pi)$ to an instance of the following bounded-degree maximum-weight induced subgraph problem for G°.

Problem 1 (Bounded-Degree k Maximum-Weight Induced Subgraph (k-BDMWIS))**.** *Let $G = (V, E)$ be a weighted graph with a vertex weight $w(v) \in \mathbb{R}^+$ for each $v \in V$ and an edge weight $w(u, v) \in \mathbb{R}^+$ for each $(u, v) \in E$, and let $k \in \mathbb{N}$. Find a set $V' \subset V$ such that the induced subgraph $G[V'] = (V', E')$ has maximum vertex degree k and maximizes the weight*

$$W = W(G[V']) = \sum_{v \in V'} w(v) - \sum_{(u,v) \in E'} w(u, v).$$

For general graphs it follows immediately from Yannakakis [32] that k-BDMWIS is NP-hard, but restricting the graph class to circle graphs makes the problem significantly easier, at least for constant k, as we show in this paper.

For our reduction it remains to assign suitable vertex and edge weights to G°. We define $w(v) = \deg(v)$ for all vertices $v \in V$ and $w(u, v) = 1$ or, alternatively, $w(u, v) = 2$ for all edges $(u, v) \in E$. The former will count only the crossings removed from the interior, ignoring the crossings in the outside, the latter counts the number of crossings saved in the interior minus the ones moved to the outside of the circle.

Lemma 1. *Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a graph with cyclic vertex order π and $k \in \mathbb{N}$. Then a maximum-weight degree-k induced subgraph in the corresponding weighted circle graph $G^\circ = (V, E)$ induces an outer k-plane graph that minimizes the number of crossings in the corresponding two-sided layout Δ of $(\mathcal{G}, \pi)$.*

Proof. Let $V^* \subset V$ be a vertex set that induces a maximum-weight subgraph of degree at most k in G°. Since vertices in G° correspond to edges in $\mathcal{G}$, we can choose $\mathcal{E}^* = V^*$ as the set of exterior edges in Δ. Each edge in G° corresponds to a crossing in the one-sided circular layout Γ. Hence each edge in the induced graph $G^\circ[V^*]$ corresponds to an exterior crossing in Δ. Since the maximum degree of $G^\circ[V^*]$ is k, no exterior edge in Δ has more than k crossings.

The degree of a vertex $v \in V^*$ in G° (and thus its weight $w(v)$) equals the number of crossings that are removed from Γ by drawing the corresponding edge in $\mathcal{E}^*$ in the exterior part of Δ. However, if two vertices in V^* are connected by an edge, their corresponding edges in $\mathcal{E}^*$ necessarily cross in the exterior part of Δ and we need to add a correction term, otherwise the crossing would be counted twice. So for edge weights $w(u, v) = 1$ the weight W maximized by V^* equals the number of crossings that are removed from the interior part of Δ. For $w(u, v) = 2$, though, the weight W equals the number of crossings that are removed from the interior, but not counting those that are simply shifted to the exterior of Δ. □

Lemma 1 tells us that instead of minimizing the crossings in two-sided circular layouts with an outer k-plane exterior graph, we can focus on solving the k-BDMWIS problem for circle graphs in the remainder of the paper. Any solution to k-BDMWIS with the above vertex and edge weights immediately yields a solution to our crossing minimization problem.

2.2 Interval representation of circle graphs

There are two alternative representations of circle graphs. The first one is the *chord representation* as a set of chords of a circle (i.e., a one-sided circular layout), whose intersection graph actually serves as the very definition of circle graphs. The second and less immediate representation is the *interval representation* as an *overlap graph*, which is more convenient for describing our algorithms. In an interval representation each vertex is represented as a closed interval $I \subset \mathbb{R}$. Two vertices are adjacent if and only if the two corresponding intervals partially overlap, i.e., they intersect but neither interval contains the other.

Gavril [18] showed that circle graphs and overlap graphs represent the same class of graphs. To obtain an interval representation from a chord representation Γ on a circle C the idea is to pick a point p on C, which is not the endpoint of a chord, rotate Γ such that p is the topmost point of Γ and project the chords from p onto the real line below C, see Figure 2. Every chord is then represented as a finite interval and two chords intersect if and only if their projected intervals partially overlap. We can further assume that all endpoints of the projected intervals are distinct by locally separating chords with a shared endpoint in Γ before the projection, such that the intersection graph of the chords does not change.

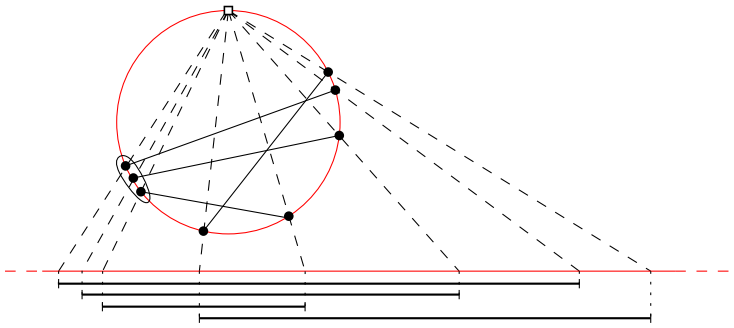


Figure 2: An example projection of the chord representation of a circle graph (here: $K_{1,3}$) to obtain an interval representation of the same graph as an overlap graph. Marked groups of endpoints indicate how chords incident to the same vertex are separated before the projection.

3 NP-hardness

For arbitrary, non-constant $k \in \mathbb{N}$ we show that k-BDMWIS is NP-hard, even on circle graphs. Our reduction is from the following Minimum Dominating Set problem, which is NP-hard on circle graphs [21].

Problem 2 (Minimum Dominating Set)**.** *Given a graph $G = (V, E)$, find a set $V' \subseteq V$ of minimum cardinality such that for each $u \in V \setminus V'$ there is a vertex $v \in V'$ with $(u, v) \in E$.*

Theorem 2. *k-BDMWIS for a graph $G = (V, E)$ and $k \in \mathbb{N}$ is NP-hard on circle graphs, even if $w(v) = 1$ for all $v \in V$ and $w(u, v) = 0$ for all $(u, v) \in E$.*

Proof. Given an instance of Minimum Dominating Set on a circle graph $G = (V, E)$ we construct an instance of k-BDMWIS in polynomial time. First let $G' = (V', E')$ be a copy of G. We set the degree bound k equal to the maximum degree of G and attach new leaves to each vertex $v \in V'$ until every (non-leaf) vertex in G' has degree $k + 1$. Note that G' remains a circle graph when adding leaves. We set the weights to $w(v) = 1$ for $v \in V'$ and $w(u, v) = 0$ for $(u, v) \in E'$. This implies that the weight W to be maximized is just the number of vertices in the induced subgraph.

Let $V_s \subseteq V'$ be a solution to the k-BDMWIS problem on G'. Again we can assume that V_s contains all leaves of G'. Otherwise let $u \in V' \setminus V_s$ be a leaf of G' with unique neighbor $v \in V'$. The only possible reason that $u \notin V_s$ is that $v \in V_s$ and the degree of v in $G'[V_s]$ is $\deg(v) = k$. If we replace in V_s a non-leaf neighbor w of v (which must exist) by the leaf u, the resulting set has the same cardinality and satisfies the degree constraint. Now let $V_d = V' \setminus V_s$. By our assumption V_d contains no leaves of G' and $V_d \subseteq V$. Since every vertex in $G'[V_s]$ has degree at most k we know that each $v \in V \setminus V_d$ must have a neighbor $u \in V_d$, otherwise it would have degree $k + 1$ in $G'[V_s]$. Thus V_d is a dominating set. Further, V_d is a minimum dominating set. If there was a smaller dominating set V'_d in G then $V' \setminus V'_d$ would be a larger solution than V_s for the k-BDMWIS problem on G', which is a contradiction. □

4 Algorithms for k-BDMWIS on circle graphs

Before describing our dynamic programming algorithms for $k = 1$ and the generalization to $k \geq 2$ in this section, we introduce the necessary basic definitions and notation using the interval perspective on k-BDMWIS for circle graphs.

4.1 Notation and Definitions

Let $G = (V, E)$ be a circle graph and $\mathcal{I} = \{I_1, \ldots, I_n\}$ an interval representation of G with n intervals that have $2n$ distinct endpoints as defined in Section 2.2. Let $\sigma(\mathcal{I}) = \{\sigma_1, \ldots, \sigma_{2n}\}$ be the set of all interval endpoints and assume that they are sorted in increasing order, i.e., $\sigma_i < \sigma_j$ for all $i < j$. We may in fact assume without loss of generality that $\sigma(\mathcal{I}) = \{1, \ldots, 2n\}$ by mapping each interval $[\sigma_l, \sigma_r] \in \mathcal{I}$ to the interval $[l, r]$ defined by its index ranks. Clearly the order of the endpoints of two intervals $[\sigma_l, \sigma_r]$ and $[\sigma_{l'}, \sigma_{r'}]$ is exactly the same as the order of the endpoints of the intervals $[l, r]$ and $[l', r']$, and thus the overlap or circle graph defined by the new interval set is exactly the same as the one defined by $\mathcal{I}$.

For two distinct intervals $I = [a, b]$ and $J = [c, d] \in \mathcal{I}$ we say that I and J *overlap* if $a < c < b < d$ or $c < a < d < b$. Two overlapping intervals correspond to an edge in G. For an interval $I \in \mathcal{I}$ and a subset $\mathcal{I}' \subseteq \mathcal{I}$ we define the *overlap set* $\mathcal{P}(I, \mathcal{I}') = \{J \mid J \in \mathcal{I}' \text{ and } I, J \text{ overlap}\}$. Further, for $I = [a, b]$, we define the *forward overlap set* $\overrightarrow{\mathcal{P}}(I, \mathcal{I}') = \{J \mid J = [c, d] \in \mathcal{P}(I, \mathcal{I}') \text{ and } c < b < d\}$ of intervals overlapping on the right side of I and the set $\mathcal{P}(\mathcal{I}') = \{\{I, J\} \mid I, J \in \mathcal{I}' \text{ and } J \in \mathcal{P}(I, \mathcal{I}')\}$ of all overlapping pairs of intervals in $\mathcal{I}'$. If $J \subset I$, i.e., $a < c < d < b$, we say that I *nests* J (or J is *nested* in I). Nested intervals do not correspond to edges in G. For a subset $\mathcal{I}' \subseteq \mathcal{I}$ we define the set of all intervals in $\mathcal{I}'$ that are nested in I as $\mathcal{N}(I, \mathcal{I}') = \{J \mid J \in \mathcal{I}' \text{ and } J \text{ is nested in } I\}$.

Let $\mathcal{I}' \subseteq \mathcal{I}$ be a set of n' intervals. We say $\mathcal{I}'$ is *connected* if its corresponding overlap or circle graph is connected. Further let $\sigma(\mathcal{I}') = \{i_1, \ldots, i_{2n'}\}$ be the sorted interval endpoints of $\mathcal{I}'$. The *span* of $\mathcal{I}'$ is defined as $\text{span}(\mathcal{I}') = i_{2n'} - i_1$ and the *fit* of the set $\mathcal{I}'$ is defined as $\text{fit}(\mathcal{I}') = \max\{i_{j+1} - i_j \mid 1 \leq j < 2n'\}$. Intuitively, the span of a set of intervals $\mathcal{I}'$ is the length of the union of all intervals in $\mathcal{I}'$, i.e., the minimum length an interval J' needs in order to fully contain $\mathcal{I}'$, while the fit of $\mathcal{I}'$ is the maximum length an interval J' may have such that it can still fit in between the two furthest consecutive interval endpoints in $\sigma(\mathcal{I}')$. For example consider Figure 3. Let $\mathcal{I}'$ be the red intervals, then their span is just $d - a$ and their fit is $b - c$.

For a weighted circle graph $G = (V, E)$ with interval representation $\mathcal{I}$ we can immediately assign each vertex weight $w(v)$ as an interval weight $w(I_v)$ to the interval $I_v \in \mathcal{I}$ representing v and each edge weight $w(u, v)$ to the overlapping pair of intervals $\{I_u, I_v\} \in \mathcal{P}(\mathcal{I})$ that represents the edge $(u, v) \in E$. We can now phrase the k-BDMWIS problem for a circle graph in terms of its interval representation, i.e., given an interval representation $\mathcal{I}$ of a circle graph G, find a subset $\mathcal{I}' \subseteq \mathcal{I}$ such that no $I \in \mathcal{I}'$ overlaps more than k other intervals in $\mathcal{I}'$ and such that the weight $W(\mathcal{I}') = \sum_{I \in \mathcal{I}'} w(I) - \sum_{\{I,J\} \in \mathcal{P}(\mathcal{I}')} w(I, J)$ is maximized. We call such an optimal subset $\mathcal{I}'$ of intervals a *max-weight k-overlap set*.

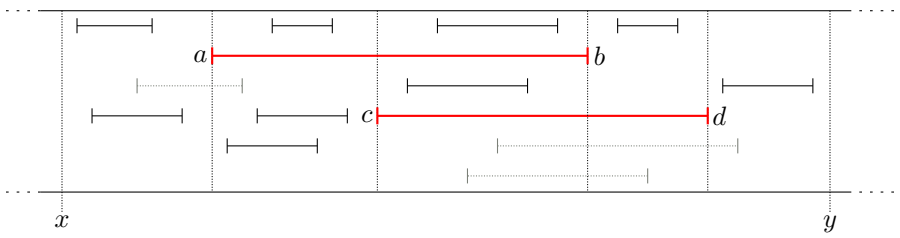


Figure 3: Split along the two thick red intervals. The dotted intervals are discarded and we recurse on the five sets with black intervals.

4.2 Properties of max-weight 1-overlap sets

The basic idea for our dynamic programming algorithm for $k = 1$ is to decompose any 1-*overlap set*, i.e., a set of intervals, in which no interval overlaps more than one other interval, into a sequence of independent single intervals and overlapping interval pairs. Consequently, we can find a max-weight 1-overlap set by optimizing over all possible ways to select a single interval or an overlapping interval pair and recursively solving the induced independent subinstances that are obtained by splitting the instance according to the selected interval(s).

Let $\mathcal{I}$ be a set of intervals. For $x, y \in \mathbb{R} \cup \{\pm\infty\}$ with $x \leq y$ we define the set $\mathcal{I}[x, y] = \{I \in \mathcal{I} \mid I \subseteq [x, y]\}$ as the subset of $\mathcal{I}$ contained in $[x, y]$. Note that $\mathcal{I}[-\infty, \infty] = \mathcal{I}$. For any $I = [a, b] \in \mathcal{I}[x, y]$ we can *split* $\mathcal{I}[x, y]$ *along* I into the three sets $\mathcal{I}[x, a]$, $\mathcal{I}[a, b]$, $\mathcal{I}[b, y]$. This split corresponds to selecting I as an interval without overlaps in a candidate 1-overlap set. All intervals which are not contained in one of the three sets will be discarded after the split. Similarly, we can split $\mathcal{I}[x, y]$ along a pair of overlapping intervals $I = [a, b], J = [c, d] \in \mathcal{I}$ to be included in a candidate solution. Without loss of generality let $a < c < b < d$. Then the split creates the five sets $\mathcal{I}[x, a]$, $\mathcal{I}[a, c]$, $\mathcal{I}[c, b]$, $\mathcal{I}[b, d]$, $\mathcal{I}[d, y]$, see Figure 3. Again, all intervals which are not contained in one of the five sets are discarded. Formally we obtain the following lemma.

Lemma 3. *For any $x \in \mathbb{R}$ at most two overlapping intervals $I = [a, b], J = [c, d] \in \mathcal{I}$ with $a \leq x \leq b$ and $c \leq x \leq d$ can be part of a 1-overlap set of $\mathcal{I}$.*

Our algorithm for the max-weight 1-overlap set problem extends some of the ideas of the algorithm presented by Valiente for the independent set problem in circle graphs [31]. In our analysis we use Valiente's notion of *total chord length*, where the chord length is the same as the *length* $\ell(I) = j - i$ of the corresponding interval $I = [i, j] \in \mathcal{I}$. The *total interval length* can then be defined as $\ell = \ell(\mathcal{I}) = \sum_{I \in \mathcal{I}} \ell(I)$. We use the following bound in our analysis.

Lemma 4. *Let $\mathcal{I}$ be a set of intervals and γ be the maximum degree of the corresponding overlap or circle graph, then $\sum_{I \in \mathcal{I}} \sum_{J \in \mathcal{P}(I, \mathcal{I})} (\ell(I) + \ell(J)) = O(\gamma \ell)$.*

Proof. We first observe that $J \in \mathcal{P}(I, \mathcal{I})$ if and only if $I \in \mathcal{P}(J, \mathcal{I})$. So in total each interval in $\mathcal{I}$ appears at most γ times as I and at most γ times as J in the double sum, i.e., no interval in $\mathcal{I}$ appears more than 2γ times and the bound follows. □

4.3 An algorithm for computing max-weight 1-overlap sets

Our algorithm to compute max-weight 1-overlap sets runs in two phases. In the first phase, we compute the weights of optimal solutions on subinstances of increasing size by recursively re-using solutions of smaller subinstances. In the second phase we optimize over all ways of combining optimal subsolutions to obtain a max-weight 1-overlap set.

The subinstances of interest are defined as follows. Let $\mathcal{I}' \subseteq \mathcal{I}$ be a connected set of intervals and let $l = l(\mathcal{I}')$ and $r = r(\mathcal{I}')$ be the leftmost and rightmost endpoints of all intervals in $\mathcal{I}'$. We define the value $1\text{MWOS}(\mathcal{I}')$ as the maximum weight of a 1-overlap set on $\mathcal{I}[l, r]$ that includes $\mathcal{I}'$ in the 1-overlap set (if one exists). Lemma 3 implies that it is sufficient to compute the 1MWOS values for single intervals $I \in \mathcal{I}$ and overlapping pairs $I, J \in \mathcal{I}$ since any connected set of three or more intervals cannot be a 1-overlap set any more.

We start with the computation of $1\text{MWOS}(I)$ for a single interval $I = [a, b] \in \mathcal{I}$. Using a recursive computation scheme of 1MWOS that uses increasing interval lengths, we may assume by induction that for any interval $J \in \mathcal{I}$ with $\ell(J) < \ell(I)$ and any overlapping pair of intervals $J, K \in \mathcal{I}$ with $\text{span}(J, K) < \ell(I)$, the sets $1\text{MWOS}(J)$ and $1\text{MWOS}(J, K)$ are already computed. If we select I for the 1-overlap set as a single interval without overlaps, we need to consider for $1\text{MWOS}(I)$ only those intervals nested in I. Informally, we have to consider three cases for every interval $J \in \mathcal{N}(I, \mathcal{I})$. Either J is discarded, added as a singleton, or added together with one interval from $\overrightarrow{\mathcal{P}}(J, \mathcal{I}[a, b])$. Each case splits the set of intervals $\mathcal{I}[a, b]$ into subinstances. Refer to Figure 4 for an illustration. The value of $1\text{MWOS}(I)$ is determined using an auxiliary recurrence $S_I[x]$ for $a \leq x \leq b$ and the weight $w(I)$ resulting from the choice of I:

$$1\text{MWOS}([a, b]) = S_I[a + 1] + w(I). \tag{1}$$

For a fixed interval $I = [a, b]$ the value $S_I[x]$ represents the weight of an optimal solution of $\mathcal{I}[x, b]$. To simplify the definition of recurrence $S_I[x]$ we define the set $D_S([c, d], \mathcal{I}[a, b])$ with $[c, d] \in \mathcal{I}[a, b]$, in which we collect all 1MWOS values for pairs composed of $[c, d]$ and an interval overlapping $[c, d]$ on the right in $\overrightarrow{\mathcal{P}}([c, d], \mathcal{I}[a, b])$ (see Figure 4c) as

$$D_S([c, d], \mathcal{I}[a, b]) = \left\{1\text{MWOS}([c, d], [e, f]) + S_I[f + 1] \mid [e, f] \in \overrightarrow{\mathcal{P}}([c, d], \mathcal{I}[a, b])\right\}. \tag{2}$$

The main idea of the definition of $S_I[x]$ is a maximization step over the already computed sub-solutions that may be composed to an optimal solution for $\mathcal{I}[x, b]$. To stop the recurrence

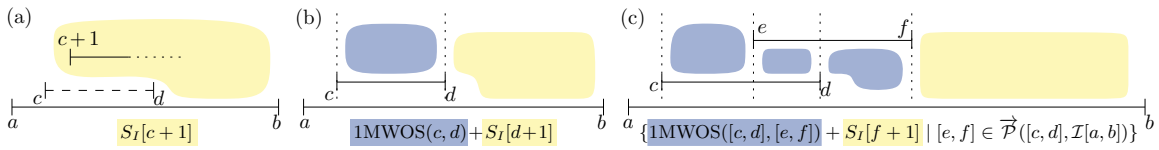


Figure 4: Illustration of Recurrence (3). The dashed intervals are discarded, while solid ones are considered in the solution.

we set $S_I[b] = 0$ and for every end-point d of an interval $[c, d] \in \mathcal{I}[a, b]$ we set $S_I[d] = S_I[d+1]$. It remains to define the recurrence for the start-point c of each interval $[c, d] \in \mathcal{I}[a, b]$:

$$S_I[c] = \max\left\{\{S_I[c+1], 1\text{MWOS}([c, d]) + S_I[d+1]\} \cup D_S([c, d], \mathcal{I}[a, b])\right\}. \tag{3}$$

Figure 4 depicts which of the possible configurations of selected intervals is represented by which values in the maximization step of Recurrence (3). The first option (Figure 4a) is to discard the interval $[c, d]$, the second option (Figure 4b) is to select $[c, d]$ as a single interval, and the third option (Figure 4c) is to select $[c, d]$ and one interval in its forward overlap set.

Lemma 5. *Let $\mathcal{I}$ be a set of intervals and $I \in \mathcal{I}$, then the value $1MWOS(I)$ can be computed in $O(\gamma\,\ell(I))$ time assuming all $1MWOS(J)$ and $1MWOS(J, K)$ values are computed for $J, K \in \mathcal{I}$, $\ell(J) < \ell(I)$ and $\text{span}(J, K) < \ell(I)$.*

Proof. Recurrence (1) is correct if $S[a+1]$ is exactly the weight of a max-weight 1-overlap set on the set $\mathcal{N}(I, \mathcal{I})$, the set of nested intervals of I. The proof is by induction over the number of intervals in $\mathcal{N}(I, \mathcal{I})$. In case $\mathcal{N}(I, \mathcal{I})$ is empty $b = a + 1$ and with $S[b] = 0$ Recurrence (1) is correct.

Now let $\mathcal{N}(I, \mathcal{I})$ consist of one or more intervals. As sketched above, by Lemma 3 there can only be three cases of how an interval $J \in \mathcal{N}(I, \mathcal{I})$ contributes. We can decide to discard J, to add it as a singleton interval, which allows us to split $\mathcal{N}(I, \mathcal{I})$ along J, or to add an overlapping pair $J, K \in \mathcal{N}(I, \mathcal{I})$ such that $K \in \overrightarrow{\mathcal{P}}(J, \mathcal{I}[a, b])$ and split $\mathcal{N}(I, \mathcal{I})$ along J, K.

For the start-point c of an interval $J = [c, d]$ the maximization in the definition of S in Recurrence (3) exactly considers these three possibilities (recall Figure 4). For all end-points aside from b we simply use the value of the next start-point or $S[b] = 0$ which ends the recurrence. Since all $1\text{MWOS}(J)$ and $1\text{MWOS}(J, K)$ are computed for $J, K \in \mathcal{I}$, $\ell(J) < \ell(I)$ and $\text{span}(J, K) < \ell(I)$ the auxiliary table S is computed in one iteration across $\sigma(\mathcal{I}[a, b])$.

The overall running time is dominated by traversing the D_S sets, which contain at most γ values. This has to be done for every start-point of an interval in $\mathcal{N}(I, \mathcal{I})$ which leads to an overall computation time of $O(\gamma\,\ell(I))$ for $1\text{MWOS}(I)$. □

Until now we only considered computing the 1MWOS value of a single interval, but we still need to compute 1MWOS for pairs of overlapping intervals. Let $I = [c, d], J = [e, f] \in \mathcal{I}$ be two intervals such that $J \in \overrightarrow{\mathcal{P}}(I, \mathcal{I})$. If we split $\mathcal{I}$ along these two intervals we find three independent regions (recall Figure 4c) and obtain

$$1\text{MWOS}(I, J) = L_{I,J}[c+1] + M_{I,J}[e+1] + R_{I,J}[d+1] + w(I) + w(J) - w(I, J). \tag{4}$$

The auxiliary recurrences $L_{I,J}, M_{I,J}, R_{I,J}$ are defined for the three independent regions induced by the endpoints $c < e < d < f$ in the very same way as S_I above with the exception that $L_{I,J}[e] = 0, M_{I,J}[d] = 0$ and $R_{I,J}[f] = 0$. Hence, following essentially the same proof as for Lemma 5 we obtain

Lemma 6. *Let $\mathcal{I}$ be a set of intervals and $I, J \in \mathcal{I}$ with $J \in \overrightarrow{\mathcal{P}}(I, \mathcal{I})$, then $1MWOS(I, J)$ can be computed in $O(\gamma \operatorname{span}(I, J))$ time assuming all $1MWOS(K)$ and $1MWOS(K, L)$ values are computed for $K, L \in \mathcal{I}$, $\ell(K) < \operatorname{fit}(I, J)$ and $\operatorname{span}(K, L) < \operatorname{fit}(I, J)$.*

We summarize the running times for computing the 1MWOS values for all intervals and overlapping interval pairs in $\mathcal{I}$.

Lemma 7. *Let $\mathcal{I}$ be a set of intervals. The $1MWOS$ values for all $I \in \mathcal{I}$ and all pairs $I, J \in \mathcal{I}$ with $J \in \overrightarrow{\mathcal{P}}(I, \mathcal{I})$ can be computed in $O(\gamma^2 \ell)$ time.*

Proof. For an interval $I \in \mathcal{I}$ the value $1\text{MWOS}(I)$ is computed in $O(\gamma\, \ell(I))$ time by Lemma 5. With $\ell = \sum_{I \in \mathcal{I}} \ell(I)$ the claim follows for all $I \in \mathcal{I}$.

By Lemma 6 the value $1\text{MWOS}(I, J)$ can be computed in $O(\gamma \operatorname{span}(I, J))$ time for each overlapping pair I, J with $J \in \overrightarrow{\mathcal{P}}(I, \mathcal{I})$. Since $\operatorname{span}(I, J) \leq \ell(I) + \ell(J)$ the time bound of $O(\gamma^2 \ell)$ follows by applying Lemma 4. □

Until now we have computed the 1MWOS values for every interval $I \in \mathcal{I}$ and every pair $I, J \in \mathcal{I}$ with $J \in \overrightarrow{\mathcal{P}}(I, \mathcal{I})$. It remains to compute the maximum 1-overlap set for $\mathcal{I}$ from these values. For this, we compute the maximum weight of a 1-overlap set for $\mathcal{I}$ by defining another recurrence $T[x]$ for $x \in \sigma(\mathcal{I})$ and re-using the 1MWOS values. The recurrence for T is defined similarly to the recurrence of S_I above. We set $T[2n] = 0$. Let $I = [a, b] \in \mathcal{I}$ be an interval and $b \neq 2n$, then

$$T[b] = T[b+1] \qquad T[a] = \max \left\{ \begin{array}{l} \{T[a+1]\} \cup \\ \{1\text{MWOS}([a, b]) + T[b+1]\} \cup \\ D_T(I, \mathcal{I}) \end{array} \right\}, \tag{5}$$

where D_T is defined analogously to D_S by replacing the recurrence S_I with T in (2). The maximum weight of a 1-overlap set for $\mathcal{I}$ is found in $T[1]$, since this entry contains the largest value we can obtain from any of the possible cases for the left-most interval in $I \in \mathcal{I}$, i.e, we ignore I in the solution, we consider I, but none of its overlapping intervals, or we consider I together with one interval $J \in \overrightarrow{\mathcal{P}}(I, \mathcal{I})$.

Theorem 8. *A max-weight 1-overlap set for a set of intervals $\mathcal{I}$ can be computed in $O(\gamma^2 \ell) \subseteq O(|\mathcal{I}|^4)$ time, where ℓ is the total interval length and γ is the maximum degree of the corresponding overlap graph.*

Proof. The time to compute all 1MWOS values is $O(\gamma^2 \ell)$ by Lemma 7. As argued above, the optimal solution is found by computing $T[1]$. The time to compute T in Recurrence (5) is again dominated by the maximization, which itself is dominated by the evaluation of the sets D_T. The size of these sets is exactly γ times the sum we bounded in Lemma 4. So the total time to compute T is $O(\gamma^2 \ell)$, which implies the total running time of $O(\gamma^2 \ell)$. From $\gamma \leq |\mathcal{I}|$ and $\ell \leq |\mathcal{I}|^2$ we obtain the coarser bound $O(|\mathcal{I}|^4)$.

It remains to show the correctness of Recurrence (5). Again we can treat it with the same induction used in the proof of Lemma 5. To see this we introduce an interval

$[0, 2n+1]$ with weight zero. Now the computation of the maximum weight of a 1-overlap set for $\mathcal{I}$ is the same as computing all 1MWOS values for the instance $\mathcal{I} \cup \{[0, 2n+1]\}$. Using the standard backtracking technique for dynamic programming, the same algorithm can be used to compute the max-weight 1-overlap set instead of only its weight. □

4.4 An XP-algorithm for max-weight k-overlap sets

In this section we generalize our algorithm to $k \geq 2$. While it is not possible to directly generalize Recurrences (3) and (5) we do use similar concepts. The difficulty for $k > 1$ is that the solution can have arbitrarily large connected parts, e.g., a 2-overlap set can include arbitrarily long paths and cycles. So we can no longer partition an instance along connected components into a constant number of independent sub-instances as we did above for the case $k = 1$.

We first generalize the definition of 1MWOS. Let $\mathcal{I}$ be a set of n intervals as before and $I = [a, b] \in \mathcal{I}$. We define the value $k\text{MWOS}(I)$ as the maximum weight of a k-overlap set on $\mathcal{I}[a, b]$ that includes I in the k-overlap set (if one exists). When computing such a value $k\text{MWOS}(I)$, we consider all subsets $\mathcal{J} \subseteq \mathcal{P}(I, \mathcal{I})$ of cardinality $|\mathcal{J}| \leq k$ of neighbors of I to be included in a k-overlap set, while all remaining neighbors in $\mathcal{P}(I, \mathcal{I}) \setminus \mathcal{J}$ are excluded. This consideration of all subsets of cardinality k causes an exponential dependency of the running time on k.

For keeping track of how many intervals are still allowed to overlap any interval we introduce the *capacity* of each interval boundary $i \in \sigma(\mathcal{I}) = \{1, 2, \ldots, 2n\}$. These capacities are stored in a vector $\lambda = (\lambda_1, \ldots, \lambda_{2n})$, where each λ_i is the capacity of the interval boundary $i \in \sigma(\mathcal{I})$. Each λ_i is basically a value in the set $\{0, 1, \ldots, k\}$ that indicates how many additional intervals may still overlap the boundary i of the corresponding interval, see Figure 5. We actually define $k\text{MWOS}_\lambda([a, b])$ as the maximum weight of a k-overlap set in $\mathcal{I}[a, b]$ subject to pre-defined capacities λ. The number of relevant vectors λ to consider for each interval can be bounded by $O(\gamma^k(k+1)^k)$, where γ is the maximum degree of the overlap graph corresponding to $\mathcal{I}$. Generally speaking, the idea of our dynamic programming algorithm is to recursively split an instance defined by some interval $[a, b] \in \mathcal{I}$ into a sequence of subinstances induced by smaller disjoint intervals in $\mathcal{I}[a, b]$. Unlike in 1MWOS there are dependencies between these subinstances, but they can be controlled by keeping track of only a bounded number of relevant capacity vectors.

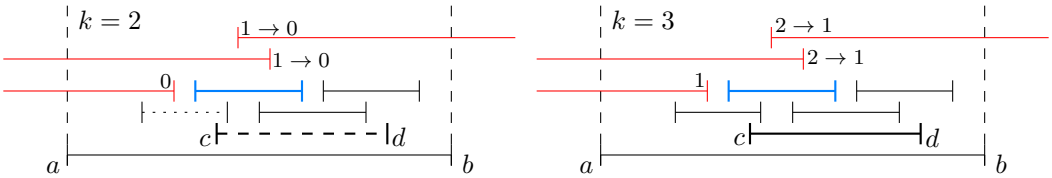


Figure 5: Examples for $k = 2$ and $k = 3$. The red intervals are in a solution set. The arrows indicate how the capacities change if the blue interval is included in a solution. For $k = 2$ we cannot use the interval $[c, d]$ since some capacities are zero, but for $k = 3$ this remains possible.

We start by describing the capacity vectors in more detail. Each capacity λ_i is in fact a value in the set $\{0, 1, \ldots, k, \infty, \perp\}$. If $\lambda_i = \perp$ we call it *undefined*, i.e., no decision about its interval has been made. If $\lambda_i = \infty$ we say it has *unlimited* capacity, which means that the corresponding interval is not selected for the candidate solution. Finally if $\lambda_i = \alpha \in \{0, 1, \ldots, k\}$ we say it has capacity α, meaning that we can still add up to α more intervals which overlap the interval corresponding to i.

For an interval $[a, b]$ we use the short-hand notation $\lambda_{a,b} = x$ to state that $\lambda_a = x$ and $\lambda_b = x$; likewise we use $\lambda_{a,b} \neq x$ to state that $\lambda_a \neq x$ and $\lambda_b \neq x$. Let $\mathcal{I}$ be a set of intervals and λ a capacity vector on $\mathcal{I}$. We call the set $\mathcal{I}(\lambda) = \{I = [a, b] \mid I \in \mathcal{I}$ *and* $\lambda_{a,b} \notin \{\infty, \perp\}\}$ the set of *selected* intervals for λ. A capacity vector λ on $\mathcal{I}$ is called *valid*, if its selected set $\mathcal{I}(\lambda)$ is a k-overlap set and for every selected $I = [a, b] \in \mathcal{I}(\lambda)$ we find $\lambda_a + \lambda_b \leq k - |\mathcal{P}(I, \mathcal{I}(\lambda))|$. Finally we call a capacity vector λ *I-valid*, if the interval $I \in \mathcal{I}$ is selected, for all $J = [c, d] \in \mathcal{P}(I, \mathcal{I})$ we find $\lambda_{c,d} \neq \perp$, and for all $J = [c, d] \in \mathcal{N}(I, \mathcal{I})$ we find $\lambda_{c,d} = \perp$. Two I-valid capacity vectors λ and λ' are said to be in the same equivalence class of capacity vectors, if they contain the same entries for all $i \in \sigma(\{I\} \cup \mathcal{P}(I, \mathcal{I}))$.

Definition 1. *Let $\mathcal{I}$ be a set of intervals, $I = [a, b] \in \mathcal{I}$ and λ a valid capacity vector for $\mathcal{I}$. Then λ' is a* legal successor *of λ for I (written $\lambda' \leftarrow_I \lambda$) if there is a set $\mathcal{J} \subseteq \mathcal{P}(I, \mathcal{I})$ with $|\mathcal{J}| \leq k$ and $\lambda_{i,j} \neq \infty$ for all $[i, j] \in \mathcal{J}$ such that for all $[x, y] \in \mathcal{I}$*

$$
\begin{aligned}
&\lambda'_x = \lambda_x, \quad \lambda'_y = \lambda_y && \text{if } [x,y] \notin \mathcal{P}(I,\mathcal{I}) \cup \{I\} && (A)\\
&\lambda'_{x,y} = 0 && \text{if } [x,y] = [a,b] = I && (B)\\
&\lambda'_{x,y} = \infty && \text{if } [x,y] \in \mathcal{P}(I,\mathcal{I}) \setminus \mathcal{J} && (C)\\
&\lambda'_x = \alpha, \quad \lambda'_y = k - t_{x,y} - \alpha \quad \text{for } \alpha \in [0, k - t_{x,y}] && \text{if } [x,y] \in \mathcal{J} \text{ and } \lambda_{x,y} = \perp && (D)\\
&\lambda'_x = \lambda_x - l_{x,y} \geq 0, \quad \lambda'_y = \lambda_y - r_{x,y} \geq 0 && \text{if } [x,y] \in \mathcal{J} \text{ and } \lambda_{x,y} \neq \perp, && (E)
\end{aligned}
$$

where

$$
\begin{aligned}
t_{x,y} &= |\{[i,j] \mid [i,j] \in \mathcal{P}([x,y],\mathcal{I}) \text{ and } ([i,j] \in \mathcal{J} \text{ or } \lambda_{i,j} \notin \{\infty,\perp\})\}|,\\
l_{x,y} &= |\{[i,j] \mid [i,j] \in \mathcal{J} \cup \{I\}, i < x < j, \lambda_{i,j} = \perp\}|, \text{ and}\\
r_{x,y} &= |\{[i,j] \mid [i,j] \in \mathcal{J} \cup \{I\}, i < y < j, \lambda_{i,j} = \perp\}|.
\end{aligned}
$$

Intuitively, a legal successor $\lambda' \leftarrow_I \lambda$ selects the interval I to be in the considered k-overlap set and determines for all intervals $J \in \mathcal{P}(I, \mathcal{I})$ whether they are selected for the k-overlap set as well ($J \in \mathcal{J}$) or not ($J \notin \mathcal{J}$), see Figure 6 for an illustration of the different cases (A)–(E) of Definition 1. Case (A) copies the capacities of all non-overlapping intervals from λ, (B) settles the capacity of I to 0, (C) sets the capacities of all non-selected intervals to ∞, (D) sets capacities for selected, but previously undefined intervals taking into account the number $t_{x,y}$ of existing overlaps, and (E) updates the capacities of previously selected intervals by subtracting the number of newly selected intervals overlapping their left ($l_{x,y}$) and right ($r_{x,y}$) interval boundaries. Capacities can only be decreased and if an interval $[i, j]$ has been discarded previously in λ, i.e., $\lambda_{i,j} = \infty$, then it cannot be contained in $\mathcal{J}$. Also if any capacity would become negative for a particular choice of $\mathcal{J}$, this is not a legal successor.

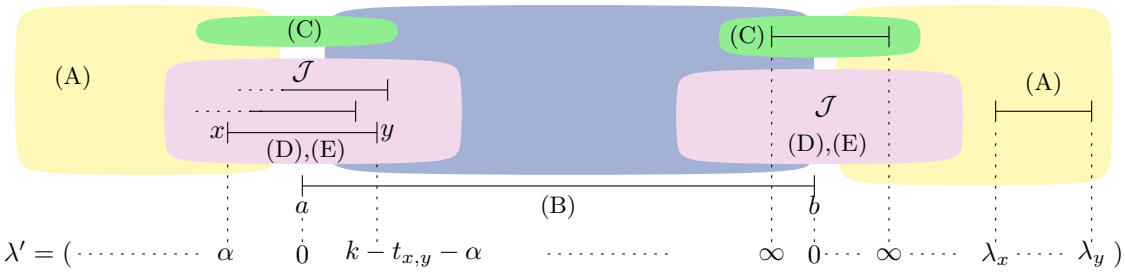


Figure 6: Illustration of a legal successor λ' for a capacity vector λ. The letters in parentheses denote the matching cases in Definition 1.

Definition 2. *Let $\mathcal{I}$ be a set of intervals, $I \in \mathcal{I}$ and λ a valid capacity vector, then*

$$L'(\lambda, I, \mathcal{I}) = \{\lambda' \mid \lambda' \leftarrow_I \lambda\}$$

is the set of all legal successors of λ for I.

First we show that for an interval set $\mathcal{I}$ there are not too many different I-valid capacity vectors λ and legal successors $\lambda' \leftarrow_I \lambda$.

Lemma 9. *Let $\mathcal{I}$ be a set of intervals, $I \in \mathcal{I}$, and γ the maximum degree of the corresponding overlap graph.*

1. *There are $O(\gamma^k(k+1)^k)$ equivalence classes of I-valid capacity vectors.*
2. *For each valid capacity vector λ the set $L'(\lambda, I, \mathcal{I})$ of legal successors has $O(\gamma^k(k+1)^k)$ elements.*

Proof. By definition we can ignore the capacities of positions that do not belong to intervals in $\mathcal{P}(I, \mathcal{I}) \cup \{I\}$. Moreover, all I-valid capacity vectors belong to the same equivalence class as long as they coincide on the capacities of positions belonging to $\mathcal{P}(I, \mathcal{I}) \cup \{I\}$. The set $\mathcal{P}(I, \mathcal{I})$ has at most γ elements, and, to be I-valid, at most k of those positions in λ may have a value in $\{0, 1, \ldots, k\}$. There are $O(\binom{\gamma}{k}) = O(\gamma^k)$ different combinations of up to k positions, and in each position up to $k+1$ different values, which yields the first bound.

For a valid capacity vector λ, a legal successor is determined by considering all $O(\gamma^k)$ choices for a set $\mathcal{J}$ of at most k overlapping intervals, and for each chosen interval there are up to $k+1$ ways of splitting the capacities between left and right index in case (D) of Definition 1. This gives again a bound of $O(\gamma^k(k+1)^k)$. □

As in Section 4.3 we first introduce the necessary recurrences, and then prove their correctness and running times in a series of subsequent lemmas. For our recursive definition we assume that, when computing $k\text{MWOS}_\lambda(I)$, all values $k\text{MWOS}_\lambda(J)$ with $J \in \mathcal{I}$ and $\ell(J) < \ell(I)$ are already computed. The following recurrence computes one $k\text{MWOS}_\lambda(I)$ value given an interval $I = [a, b] \in \mathcal{I}$ and an I-valid capacity vector λ, which, by definition, has $\lambda_{x,y} = \perp$ for all intervals $[x, y] \in \mathcal{I}[a, b]$.

$$k\text{MWOS}_\lambda([a, b]) = S_{I,\lambda}[a+1] + w([a, b]) \tag{6}$$

This recurrence is analogous to (1), where now $S_{I,\lambda}[x]$ represents the maximum weight of a k-overlap set in $\mathcal{I}[x,b]$ under the assumption that I is selected according to λ. The next step is to describe the computation of the recurrence for $S_{I,\lambda}$. In order to consider the correct weight for the corresponding solution we have to pay attention whenever a capacity is changed from $\perp$ to some $\alpha \in [0,k]$ so that the weight of its corresponding interval gets added. We introduce the following notation for the set of all intervals changed between capacity vectors λ' and λ. Let K be an interval in a set of intervals $\mathcal{I}'$, and λ' and λ two capacity vectors, then we define the set of newly selected intervals

$$\text{new}(\lambda', \lambda, K, \mathcal{I}') = \{[i,j] \mid [i,j] \in \mathcal{P}(K, \mathcal{I}') \wedge \lambda_{i,j} = \perp \wedge 0 \leq \lambda'_i + \lambda'_j \leq k\}$$

and for those intervals in $\mathcal{P}(K, \mathcal{I}')$ that are not in the set $\text{new}(\lambda', \lambda, K, \mathcal{I}')$ we define

$$\text{old}(\lambda', \lambda, K, \mathcal{I}') = \{[i,j] \mid [i,j] \in \mathcal{P}(K, \mathcal{I}') \setminus \text{new}(\lambda', \lambda, K, \mathcal{I}')) \wedge \lambda_{i,j} \notin \{\perp, \infty\}\}.$$

Secondly we introduce the weight function $w(\lambda', \lambda, K, \mathcal{I}')$, which for two capacity vectors λ' and λ, and one newly selected interval K computes the weight of all intervals newly considered in λ', minus the weight of all overlaps (edges) involving new intervals as needed for the objective function in Problem 1.

$$w(\lambda', \lambda, K, \mathcal{I}') = \sum_{L \in \text{new}(\lambda', \lambda, K, \mathcal{I}')} w(L) - \sum_{\substack{L \in \text{new}(\lambda', \lambda, K, \mathcal{I}') \\ \cup \text{old}(\lambda', \lambda, K, \mathcal{I}')}} w(K, L) - \tag{7}$$

$$\left(\sum_{L \in \text{new}(\lambda', \lambda, K, \mathcal{I}')} \left(\frac{1}{2} \sum_{\substack{M \in \text{new}(\lambda', \lambda, K, \mathcal{I}') \\ \cap \mathcal{P}(L, \mathcal{I}')}} w(L, M) + \sum_{\substack{M \in \text{old}(\lambda', \lambda, K, \mathcal{I}') \\ \cap \mathcal{P}(L, \mathcal{I}')}} w(L, M) \right) \right).$$

The first sum simply adds the weights of all newly selected intervals. To account for the edge weights correctly, there are three types of overlaps. The interval K overlaps new and old intervals (collected in the second term). A new interval L may overlap other new intervals in which case we consider only half its weight as it will be counted twice (third term), and it may overlap some old intervals (fourth term). This together forms the expression in (7).

Recall Recurrence (6) for $k\text{MWOS}_\lambda([a,b])$. As in the approach for computing the 1MWOS values the main work is done in recurrence $S_{I,\lambda}[x]$ where $x \in \sigma(\mathcal{I}[a,b])$. For the base case we set $S_{I,\lambda}[b] = 0$ and for every right endpoint d of an interval $J = [c,d] \in \mathcal{I}[a,b]$ we set $S_{I,\lambda}[d] = S_{I,\lambda}[d+1]$ as the subinstances for these two parameters are equivalent. For every left endpoint c we use the following maximization

$$S_{I,\lambda}[c] = \max\{\{S_{I,\lambda''}[c+1]\} \cup \{H_{I,\lambda'}([c,d]) \mid \lambda' \in L'(\lambda, [c,d], \mathcal{I}\}\}, \tag{8}$$

where λ'' is equal to λ, except for $\lambda''_{c,d} = \infty$, and thus $S_{I,\lambda''}[c+1]$ discards $[c,d]$. Further,

$$H_{I,\lambda'}([c,d]) = k\text{MWOS}_{\lambda'}([c,d]) + S_{I,\lambda'}[d+1] + w(\lambda', \lambda, [c,d], \mathcal{I}[a,b])$$

is a shorthand expression, which covers all cases with $[c,d]$ being selected. In the following we define $z = \gamma^k(k+1)^k$ as a shorthand.

Lemma 10. *Let $\mathcal{I}$ be a set of intervals, $I = [a, b] \in \mathcal{I}$, and λ an I-valid capacity vector on $\mathcal{I}$. Then the value $k\mathit{MWOS}_\lambda(I)$ can be computed in $O(z\,\ell(I))$ time once the $k\mathit{MWOS}_{\lambda'}(J)$ values are computed for all $J \in \mathcal{I}$ with $\ell(J) < \ell(I)$ and λ' a J-valid capacity vector.*

Proof. The proof is by induction over the size of $\mathcal{I}[a, b] = \mathcal{N}(I, \mathcal{I})$, the set of nested intervals of I, which is similar to the proof of Lemma 5 for 1MWOS. Suppose $\mathcal{N}(I, \mathcal{I}) = \emptyset$, then $k\mathrm{MWOS}_\lambda(I) = S_{I,\lambda}[a + 1] + w(I)$, and, regardless of the capacity vector λ, $S_{I,\lambda}[a + 1] = S_{I,\lambda}[b] = 0$ by definition.

Now consider $\mathcal{N}(I, \mathcal{I})$ with at least one interval. For every left endpoint $i \in \sigma(\mathcal{N}(I, \mathcal{I}))$ we have the choice of using the interval $J = [i, j] \in \mathcal{I}$ starting at i or discarding it. We consider the interval $J = [c, d] \in \mathcal{N}(I, \mathcal{I})$ with leftmost endpoint.

When the decision is not to use J the recurrence correctly continues with the next endpoint $c+1$ and sets the capacities of c and d to ∞. If J is selected, we have to decide, which intervals from $\mathcal{P}(J, \mathcal{I}[a, b])$ we additionally take into consideration for a solution and which are discarded. For this we exhaustively consider all possible combinations and capacities of intervals in $\mathcal{P}(J, \mathcal{I}[a, b])$. These are exactly the legal successors $\lambda' \in L'(\lambda, J, \mathcal{I})$ over which we maximize in Recurrence (8).

It remains to argue that the weight $k\mathrm{MWOS}_\lambda(I)$ is correct in the end. If J is not selected, its weight is not included and the recursion simply proceeds to the next interval. Whenever the recursion selects an interval $J = [c, d]$ it adds the value $k\mathrm{MWOS}_{\lambda'}([c, d])$, which we assume to be correct by the induction hypothesis and which contains the weight $w(J)$. The missing weights to be considered are those of other newly selected intervals in a legal successor $\lambda' \leftarrow_J \lambda$ from $\mathcal{P}(J, \mathcal{I}[a, b])$, as well as their overlaps. Recurrence (8) adds for each newly selected interval $L \in \mathrm{new}(\lambda', \lambda, J, \mathcal{I}[a, b])$ the weight $w(L)$ as part of the $w(\lambda', \lambda, [c, d], \mathcal{I}[a, b])$ term in $H_{I,\lambda'}([c, d])$, see (7). The weight of discarded intervals is ignored.

We further have to subtract the weight of all intersecting pairs $K, L \in \mathcal{I}[a, b]$, $K = [i, j], L = [x, y]$, that are both included in the solution set and where at least one of them is newly selected in the current step (if both have been added earlier, their edge weight has also been accounted for in an earlier step by the induction hypothesis). The first case is that one of the two, say K, is in fact the newly selected interval $K = J = [c, d]$. Then the weight $w(\lambda', \lambda, K, \mathcal{I}[a, b])$ in (7) correctly subtracts $w(K, L)$ for all $L \in \mathrm{new}(\lambda', \lambda, K, \mathcal{I}[a, b]) \cup \mathrm{old}(\lambda', \lambda, K, \mathcal{I}[a, b])$. Note that as the recursion proceeds from left to right, we have $\lambda_{c,d} = \perp$ and $\lambda'_{c,d} = 0$ in the legal successor $\lambda' \leftarrow_J \lambda$. Thus K itself is newly selected in λ' and all overlaps with intervals from $\mathrm{old}(\lambda', \lambda, K, \mathcal{I}[a, b])$ must be considered in this step. In the second case both K and L are newly selected in λ' and $K, L \neq J$. Thus the weight $1/2 \cdot w(K, L)$ is considered twice in (7), which yields the correct value. In the third and last case, assume K is new and L is old. In that case $w(K, L)$ is also considered correctly in (7).

The running time of $O(z\,\ell(I))$ follows from Lemma 9 (2). □

Lemma 11. *Let $\mathcal{I}$ be a set of intervals and $I \in \mathcal{I}$, then the values $k\mathit{MWOS}_\lambda(I)$ can be computed for all I-valid capacity vectors λ in $O(z^2\,\ell(I))$ time once the values $k\mathit{MWOS}_{\lambda'}(J)$ are computed for all $J \in \mathcal{I}$ with $\ell(J) < \ell(I)$ and for all J-valid capacity vectors λ'.*

Proof. We apply Lemma 10 for each I-valid capacity vector. By Lemma 9 (1) there are $O(z)$ such capacity vectors, which implies the time bound of $O(z^2\,\ell(I))$. □

Lemma 12. *Let $\mathcal{I}$ be a set of intervals, then the values $k\mathit{MWOS}_\lambda(I)$ can be computed for all $I \in \mathcal{I}$ and all I-valid capacity vectors λ in $O(z^2\ell)$ time.*

Proof. We apply Lemma 11 for every $I \in \mathcal{I}$ and obtain the running time of $O(z^2\ell)$ since $\sum_{I\in\mathcal{I}} \ell(I) = \ell$. □

Now that we know how to compute all values $k\mathrm{MWOS}_\lambda(I)$ for all $I \in \mathcal{I}$ and all I-valid capacity vectors λ, a max-weight k-overlap set can be obtained easily and yields our main result.

Theorem 13. *A max-weight k-overlap set for a set of intervals $\mathcal{I}$ can be computed in $O(\gamma^{2k}\ell) \subseteq O(|\mathcal{I}|^{2k+2})$ time, where ℓ is the total interval length and γ is the maximum degree of the corresponding overlap graph.*

Proof. For computing the maximum weight of a k-overlap set of $\mathcal{I}$ we can introduce a dummy interval $\hat{I} = [0, 2n{+}1]$ with weight $w(\hat{I}) = 0$ that nests the entire set $\mathcal{I}$, i.e., $\mathcal{N}(\hat{I}, \mathcal{I}\cup\{\hat{I}\}) = \mathcal{I}$. We define the $\hat{I}$-valid capacity vector $\hat{\lambda}$ that has $\hat{\lambda}_{0,2n+1} = 0$ and all other $\hat{\lambda}_i = \perp$. Using the computation scheme of Lemma 12 we obtain in $O(z^2\ell) = O((k+1)^{2k}\gamma^{2k}\ell) = O(\gamma^{2k}\ell)$ time all values $k\mathrm{MWOS}_\lambda(I)$, including $k\mathrm{MWOS}_{\hat{\lambda}}(\hat{I})$. Note that for this time bound we use that k is an arbitrary but fixed integer and thus $O((k+1)^k) = O(1)$. Clearly the solution corresponding to the value $k\mathrm{MWOS}_{\hat{\lambda}}(\hat{I})$ induces a max-weight k-overlap set for $\mathcal{N}(\hat{I}, \mathcal{I}\cup\{\hat{I}\}) = \mathcal{I}$. □

The running time in Theorem 13 implies that both the max-weight k-overlap set problem and the equivalent k-BDMWIS problem for circle (overlap) graphs are in XP.[3] This fact alone can alternatively be derived from a metatheorem of Fomin et al. [14] as follows.[4] The number of minimal separators of circle graphs can be polynomially bounded by $O(n^2)$ as shown by Kloks [23]. Further, since we are interested in a bounded-degree induced subgraph $G[V']$ of a circle graph $G = (V, E)$, where $V' \subseteq V$, we know from Gaspers et al. [17] that $G[V']$ has treewidth at most four times the maximum degree k. With these two pre-conditions the metatheorem of Fomin et al. [14] yields the existence of an XP-time algorithm for k-BDMWIS on circle graphs. However, the running time obtained from Fomin et al. [14] is $O(|\Pi_G| \cdot n^{t+4} \cdot f(t, \phi))$, where $|\Pi_G|$ is the number of potential cliques in G, t is the treewidth of $G[V']$, with $V' \subseteq V$ being the solution set, and f is a tower function depending only on t and on the CMSO (Counting Monadic Second Order Logic) formula ϕ (compare Thomas [30] proving this already for MSO formulas). Let k be the desired degree of a k-BDMWIS instance, then the treewidth of $G[V']$ is at most $4k$. Further by Kloks [23] we know $|\Pi_G| = O(n^2)$. Hence the running-time of the algorithm would be in $O(n^{4k+6} \cdot f(4k, \phi))$, whereas our problem-specific algorithm has running time $O(n^{2k+2})$.

[3] The class XP contains problems that can be solved in time $O(n^{f(k)})$, where n is the input size, k is a parameter, and f is a computable function.

[4] We thank an anonymous reviewer of an earlier version for pointing us to this fact.

5 Experiments

We implemented[5] the algorithm for 1-BDMWIS from Section 4.3 and the independent set algorithm by Valiente [31] for 0-BDMWIS in C++. The compiler was g++, version 7.2.0 with set -O3 flag. Further we used the OGDF library [9] in its snapshot version 2017-07-23. Experiments were run on a standard desktop computer with an eight core Intel i7-6700 CPU clocked at 3.4 GHz and 16 GB RAM, running Archlinux and kernel version 4.13.12.

Three experiments were conducted. The first one considers the amount of saved crossings and the running time for a large set of random graphs with 20 to 60 vertices and a *density*, i.e., edge-to-vertex ratio, between 1.0 and 5.0. The second experiment again looks into the crossings and running time, but uses the Rome graph library as an established set of real-world test instances. Finally, in the third experiment, we investigate the running time and its relation to the density more closely by considering random graphs with higher densities.

We generated the random graphs using OGDF and NetworkX. OGDF builds a random biconnected graph by starting with a triangle and then randomly splitting or adding edges. In NetworkX we start from a graph computed by the Erdős-Rényi model and add edges between any two components that are not biconnected.

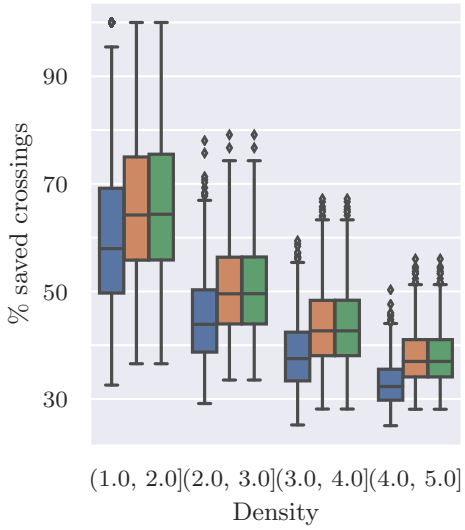
Random Graphs. For the first experiment we generated a set of 10,037 random non-planar biconnected graphs using OGDF. We varied the density between 1.0 and 5.0 and the number of vertices uniformly at random between 20 and 60. The average density is 2.83. For our experiment we first compute a circular layout for every graph using the circular layout algorithm of OGDF. This yields the required cyclic vertex order.

Figure 7a displays the percentage of crossings saved by the layouts with exterior edges versus the one-sided circular layout computed in OGDF. Compared to the approach without exterior crossings ($k = 0$) we find that allowing up to one crossing per edge in the exterior ($k = 1$) can save about 11% of the total crossings. Setting the edge weight in 1-BDMWIS to $w(e) = 1$ (i.e., counting exterior crossings in the optimization) or $w(e) = 2$ (i.e., not counting exterior crossings in the optimization), has only a very small effect below one percent. Likely this is due to the fact that it is rarely possible to trade a crossing pair of edges for a single edge, which would be needed to save more crossings.

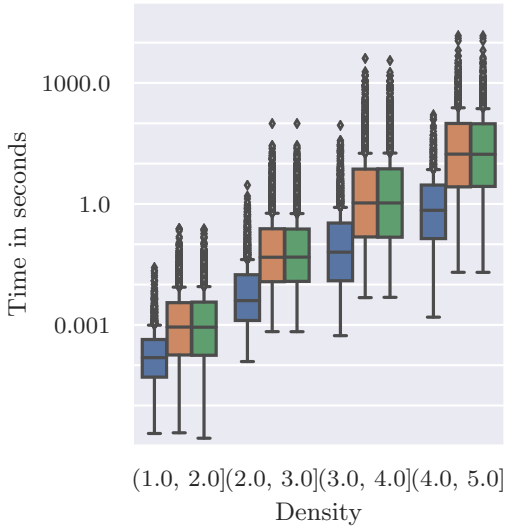
Figure 7b depicts the time needed to compute the layouts for the respective densities. The observations confirm the expected behaviour. The algorithm for $k = 0$ with $O(|\mathcal{E}|^2)$ time is significantly faster than the algorithm for $k = 1$ as the graphs get denser. Still for our sparse test instances our algorithm for $k = 1$ with $O(|\mathcal{E}|^4)$ time runs sufficiently fast to be used on graphs with up to 60 vertices and a density of 5.0 (82 seconds on average).

Rome Graphs. From the Rome graph library[6] we extracted 8,504 graphs with a density between 1.0 and 2.1 (on average 1.42) and 10 to 100 vertices. For these graphs we observed

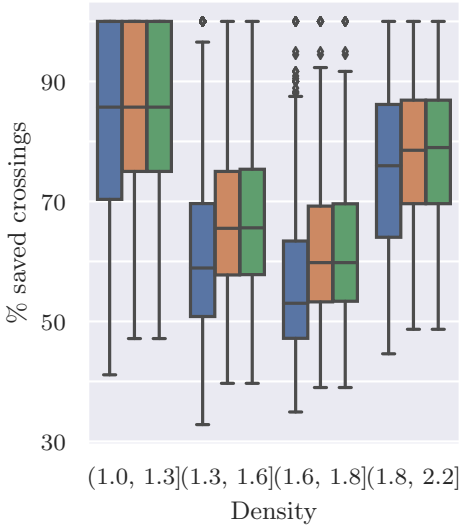
[5] Our implementation and data are available at https://www.ac.tuwien.ac.at/two-sided-layouts/.
[6] The Rome graphs can be downloaded from http://www.graphdrawing.org/data.html.



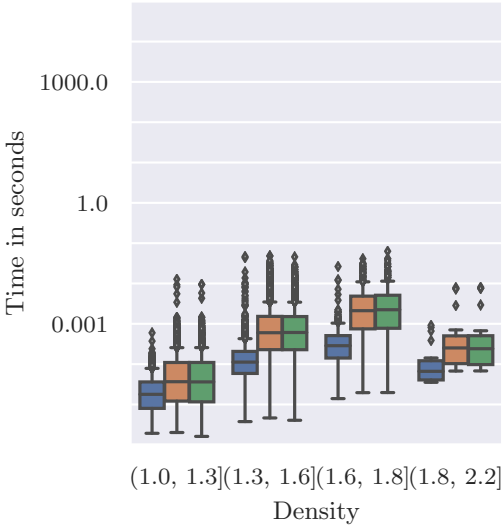
(a) Density vs percentage of saved crossings for the random graphs.



(b) Density vs computation time (log scale) for the random graphs.



(c) Density vs percentage of saved crossings for the Rome graphs.



(d) Density vs computation time (log scale) for the Rome graphs.

Figure 7: Boxplots for the benchmark graphs. The parameter $k \in \{0, 1\}$ is the number of allowed crossings per exterior edge and $w \in \{1, 2\}$ is the weight given to the edges of the circle graph, see Section 2.1.

odd results for both the saved crossings (Figure 7c) and the running time (Figure 7d) for densities above 1.8. Since there are only 20 graphs (less than 0.2% of the Rome graphs) with densities between 1.8 and 2.2, we decided to treat these graphs as outliers. When omitting this density interval the Rome graphs are in line with the above discussion, as we see similar behavior as for the random graphs. The additionally saved crossings when going from $k = 0$ to $k = 1$ are about 7.5% on average. Given that the Rome graphs are very sparse, the running times are also very fast.

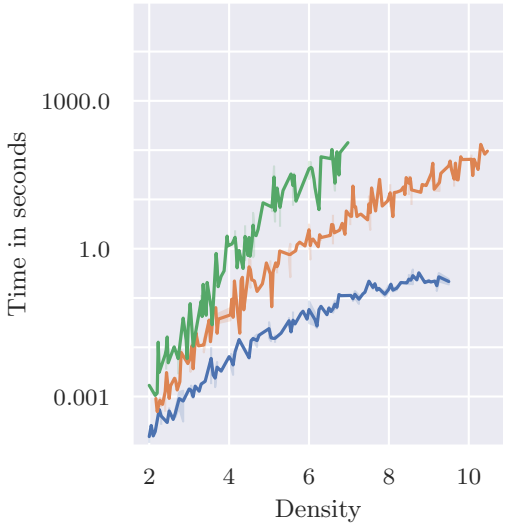
Overall our experiments on the two data sets show a clear improvement in crossing reduction when going from $k = 0$ to $k = 1$, which gives empirical support to our approach of using two-sided circular layouts with a k-plane exterior edge set for $k > 0$. Of course this comes with a non-negligible runtime increase. For the practically interesting sparse instances, though, 1-BDMWIS can be solved fast enough to be useful in practice for small to medium-size graphs.

Denser Random Graphs. To explore the dependencies of the running times on graph size and density we conducted another experiment. We generated graphs with 20, 30, and 40 vertices and densities between 2.0 and 10.0 for 20 and 30 vertices, and between 2.0 to 7.0 for 40 vertices using the NetworkX[7] library (version 2.3). For each interval $[d, d + 0.5)$ of densities and each number of vertices we generated 10 graphs. Figure 8 shows the result. As with the previous graph sets the running time increases with the density, see Figures 8a and 8b. If instead we plot the number of edges in the input graph versus the computation time, see Figures 8c and 8b, we observe that with increasing number of edges the time to compute the solution to the 0-BDMWIS and 1-BDMWIS is independent of the number of vertices in the input graph in the sense that all three curves overlap to a large extent. The fact that the curve for 20 vertices becomes flat in the end is also expected, since we have a nearly complete graph for these densities in every run.

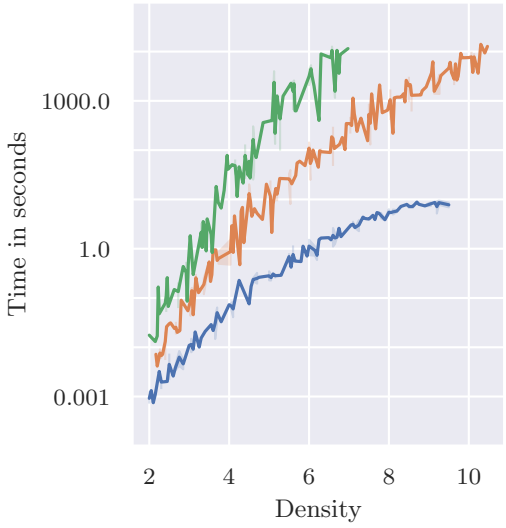
6 Open questions

The overall hardness of the k-BDMWIS problem on circle graphs, parameterized by just the desired degree k, remains open. While we could show NP-hardness, we do not know whether an FPT-algorithm exists or whether the problem is W[1]-hard. Looking at the general bounded-degree-deletion problem we immediately see that we get an FPT-algorithm if we parameterize by treewidth of the input graph plus k, due to Courcelle's Theorem [11]. Parameterizing only by the treewidth, though does not give an FPT-algorithm [6]. Since we are only interested in circle graphs it might be the case that an FPT-algorithm can be found even if parameterizing solely by the treewidth of the input graph. This is also interesting, since the tree-decomposition of a circle graph can be computed in polynomial time [23], although the treewidth might be unbounded. Furthermore it is known that if we bound the number of crossings in a circular layout [3] or the number crossings per edge [8] by $c \in \mathbb{N}$, the resulting circle graph has bounded treewidth in $O(c)$. This also gives an immediate FPT-algorithm using Courcelle's Theorem, but restricting the number of crossings in the

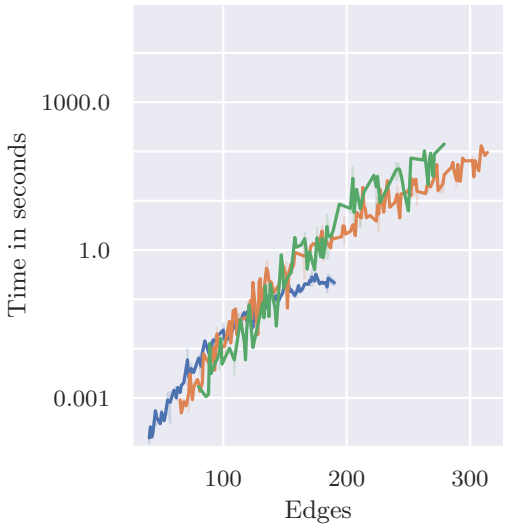
[7] https://networkx.github.io/



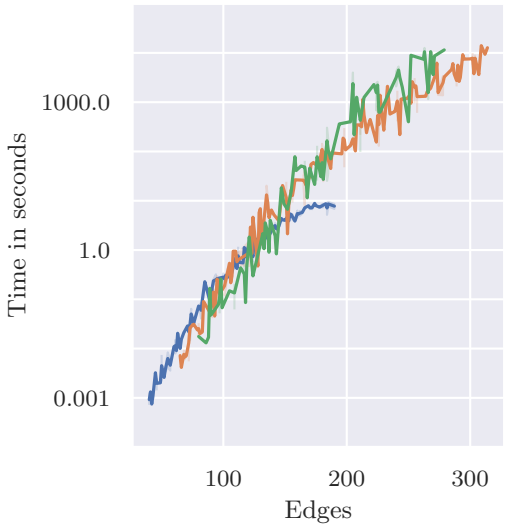
(a) Density of the input graph vs the computation time, $k = 0$.



(b) Density of the input graph vs the computation time, $k = 1$, $w = 1$.



(c) Number of edges in the input graph vs the computation time, $k = 0$.



(d) Number of edges in the input graph vs the computation time, $k = 1$, $w = 1$.

Figure 8: Computation times for the denser random graphs analyzed by density (a–b) and edge number (c–d). The colors indicate the vertex numbers. The parameter $k \in \{0, 1\}$ is the number of allowed crossings per exterior edge and $w \in \{1, 2\}$ is the weight given to the edges of the circle graph, see Section 2.1.

circular input layout itself seems very restrictive. It would be a lot more interesting to find an FPT-algorithm parameterized only by the number of crossings in the outside.

In this paper we always assumed a fixed ordering of the vertices. Bannister and Eppstein [3] studied the crossing minimization problem for two-page book drawings under the lens of parameterized complexity. While there is no hope for an FPT-algorithm for the two-page book drawing problem parameterized only by the number of crossings, they show it to be FPT for the number of crossings plus treewidth. Similar results should be attainable if we relax the fixed vertex order in our problem. Further it would be interesting to find algorithms not relying on Courcelle's Theorem.

In terms of the motivating graph layout problem, the number of crossings is known as a major factor that influences the readability. Yet, practical two-sided layout algorithms must also apply suitable vertex-ordering heuristics and they should further take into account the length and actual routing of exterior edges. Edge bundling approaches for both interior and exterior edges promise to further reduce visual clutter, especially for dense graphs, but then bundling and bundled crossing minimization [1] should be considered simultaneously. It would also be interesting to generalize the problem from circular layouts to other layout types, where many but not necessarily all vertices are on the boundary of the outer face.

References

[1] Md. Jawaherul Alam, Martin Fink, and Sergey Pupyrev. The bundled crossing number. In Y. Hu and M. Nöllenburg, editors, *Graph Drawing and Network Visualization (GD'16)*, volume 9801 of *LNCS*, pages 399–412. Springer, 2016. doi:10.1007/978-3-319-50106-2_31.

[2] Christopher Auer, Christian Bachmaier, Franz J. Brandenburg, Andreas Gleißner, Kathrin Hanauer, Daniel Neuwirth, and Josef Reislhuber. Outer 1-planar graphs. *Algorithmica*, 74(4):1293–1320, 2016. doi:10.1007/s00453-015-0002-1.

[3] Michael J. Bannister and David Eppstein. Crossing minimization for 1-page and 2-page drawings of graphs with bounded treewidth. *Journal of Graph Algorithms and Applications*, 22(4):577–606, 2018. doi:10.7155/jgaa.00479.

[4] Michael Baur and Ulrik Brandes. Crossing reduction in circular layouts. In J. Hromkovic, M. Nagl, and B. Westfechtel, editors, *Graph-Theoretic Concepts in Computer Science (WG'04)*, volume 3353 of *LNCS*, pages 332–343. Springer, 2004. doi:10.1007/978-3-540-30559-0_28.

[5] Frank Bernhart and Paul C. Kainen. The book thickness of a graph. *Journal of Combinatorial Theory, Series B*, 27(3):320–331, 1979. doi:10.1016/0095-8956(79)90021-2.

[6] Nadja Betzler, Robert Bredereck, Rolf Niedermeier, and Johannes Uhlmann. On bounded-degree vertex deletion parameterized by treewidth. *Discrete Applied Mathematics*, 160(1–2):53–60, 2012. doi:10.1016/j.dam.2011.08.013.

[7] Ulrik Brandes and Dorothea Wagner. visone – Analysis and visualization of social networks. In M. Jünger and P. Mutzel, editors, *Graph Drawing Software*, pages 321–340. Springer, 2004. doi:10.1007/978-3-642-18638-7_15.

[8] Steven Chaplick, Myroslav Kryven, Giuseppe Liotta, Andre Löffler, and Alexander Wolff. Beyond outerplanarity. In F. Frati and K.-L. Ma, editors, *Graph Drawing and Network Visualization (GD'17)*, volume 10692 of *LNCS*, pages 546–559. Springer, 2018. doi:10.1007/978-3-319-73915-1_42.

[9] Markus Chimani, Carsten Gutwenger, Michael Jünger, Gunnar W Klau, Karsten Klein, and Petra Mutzel. The open graph drawing framework (OGDF). In R. Tamassia, editor, *Handbook of Graph Drawing and Visualization*, chapter 17, pages 543–569. CRC Press, 2013.

[10] Fan R. K. Chung, Frank Thomas Leighton, and Arnold L. Rosenberg. Embedding graphs in books: A layout problem with applications to VLSI design. *SIAM Journal on Algebraic Discrete Methods*, 8(1):33–58, 1987. doi:10.1137/0608002.

[11] Bruno Courcelle. The monadic second-order logic of graphs. i. recognizable sets of finite graphs. *Information and Computation*, 85(1):12–75, 1990. doi:10.1016/0890-5401(90)90043-H.

[12] Anders Dessmark, Klaus Jansen, and Andrzej Lingas. The maximum k-dependent and f-dependent set problem. In K. W. Ng, P. Raghavan, N. V. Balasubramanian, and F. Y. L. Chin, editors, *Algorithms and Computation (ISAAC'93)*, volume 762 of *LNCS*, pages 88–97. Springer, 1993. doi:10.1007/3-540-57568-5_238.

[13] Walter Didimo, Giuseppe Liotta, and Fabrizio Montecchiani. A survey on graph drawing beyond planarity. *ACM Computing Surveys*, 52(1):4:1–4:37, 2019. doi:10.1145/3301281.

[14] Fedor V Fomin, Ioan Todinca, and Yngve Villanger. Large induced subgraphs via triangulations and CMSO. *SIAM Journal on Computing*, 44(1):54–87, 2015. doi:10.1137/140964801.

[15] Robert Ganian, Fabian Klute, and Sebastian Ordyniak. On Structural Parameterizations of the Bounded-Degree Vertex Deletion Problem. In *Theoretical Aspects of Computer Science (STACS'18)*, volume 96 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 33:1–33:14, 2018. doi:10.4230/LIPIcs.STACS.2018.33.

[16] Emden R Gansner and Yehuda Koren. Improved circular layouts. In M. Kaufmann and D. Wagner, editors, *Graph Drawing (GD'06)*, volume 4372 of *LNCS*, pages 386–398. Springer, 2007. doi:10.1007/978-3-540-70904-6_37.

[17] Serge Gaspers, Dieter Kratsch, Mathieu Liedloff, and Ioan Todinca. Exponential time algorithms for the minimum dominating set problem on some graph classes. *ACM Transactions on Algorithms*, 6(1):9:1–9:21, 2009. doi:10.1145/1644015.1644024.

[18] Fanika Gavril. Algorithms for a maximum clique and a maximum independent set of a circle graph. *Networks*, 3(3):261–273, 1973. doi:10.1002/net.3230030305.

[19] Seok-Hee Hong, Peter Eades, Naoki Katoh, Giuseppe Liotta, Pascal Schweitzer, and Yusuke Suzuki. A linear-time algorithm for testing outer-1-planarity. *Algorithmica*, 72(4):1033–1054, 2015. doi:10.1007/s00453-014-9890-8.

[20] Michael Jünger and Petra Mutzel, editors. *Graph Drawing Software*. Mathematics and Visualization. Springer, 2004. doi:10.1007/978-3-642-18638-7.

[21] J. Mark Keil. The complexity of domination problems in circle graphs. *Discrete Applied Mathematics*, 42(1):51–63, 1993. doi:10.1016/0166-218X(93)90178-Q.

[22] Jonathan Klawitter, Tamara Mchedlidze, and Martin Nöllenburg. Experimental evaluation of book drawing algorithms. In F. Frati and K.-L. Ma, editors, *Graph Drawing and Network Visualization (GD'17)*, volume 10692 of *LNCS*, pages 224–238. Springer, 2018. doi:10.1007/978-3-319-73915-1_19.

[23] Ton Kloks. Treewidth of circle graphs. *International Journal of Foundations of Computer Science*, 7(02):111–120, 1996. doi:10.1142/S0129054196000099.

[24] Stephen G. Kobourov, Giuseppe Liotta, and Fabrizio Montecchiani. An annotated bibliography on 1-planarity. *Computer Science Review*, 25:49–67, 2017. doi:10.1016/j.cosrev.2017.06.002.

[25] Martin I. Krzywinski, Jacqueline E. Schein, Inanc Birol, Joseph Connors, Randy Gascoyne, Doug Horsman, Steven J. Jones, and Marco A. Marra. Circos: An information aesthetic for comparative genomics. *Genome Research*, 19:1639–1645, 2009. doi:10.1101/gr.092759.109.

[26] Sumio Masuda, Toshinobu Kashiwabara, Kazuo Nakajima, and Toshio Fujisawa. On the NP-completeness of a computer network layout problem. In *Circuits and Systems (ISCAS'87)*, pages 292–295. IEEE, 1987.

[27] Sumio Masuda, Kazuo Nakajima, Toshinobu Kashiwabara, and Toshio Fujisawa. Crossing minimization in linear embeddings of graphs. *IEEE Transactions on Computers*, 39(1):124–127, 1990. doi:10.1109/12.46286.

[28] Farhad Shahrokhi, László A. Székely, Ondrej Sýkora, and Imrich Vrt'o. The book crossing number of a graph. *J. Graph Theory*, 21(4):413–424, 1996. doi:10.1002/(SICI)1097-0118(199604)21:4<413::AID-JGT7>3.0.CO;2-S.

[29] Janet M. Six and Ioannis G. Tollis. Circular drawing algorithms. In R. Tamassia, editor, *Handbook of Graph Drawing and Visualization*, chapter 9, pages 285–315. CRC Press, 2013.

[30] Wolfgang Thomas. Languages, automata, and logic. *Handbook of Formal Languages*, 3:389–455, 1996. doi:10.1007/978-3-642-59126-6_7.

[31] Gabriel Valiente. A new simple algorithm for the maximum-weight independent set problem on circle graphs. In I. Toshihide, K. Naoki, and O. Hirotaka, editors, *Algorithms and Computation (ISAAC'03)*, volume 2906 of *LNCS*, pages 129–137. Springer, 2003. doi:10.1007/978-3-540-24587-2_15.

[32] Mihalis Yannakakis. Node- and edge-deletion NP-complete problems. In R. J. Lipton, W. A. Burkhard, W. J. Savitch, E. P. Friedman, and A. V. Aho, editors, *Theory of Computing (STOC'78)*, pages 253–264. ACM, 1978. doi:10.1145/800133.804355.