Solving Optimization Problems on Orthogonal Ray Graphs

Steven Chaplick[1], Philipp Kindermann[2], Fabian Lipp[2], Alexander Wolff[2]

[1] Institut für Mathematik, TU Berlin, Germany
chaplick@math.tu-berlin.de
[2] Lehrstuhl für Informatik I, Universität Würzburg, Germany
http://www1.informatik.uni-wuerzburg.de/en/staff/

1 Introduction

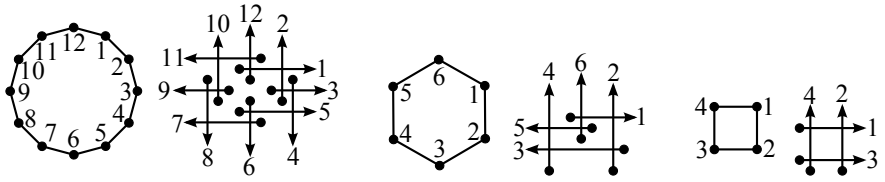
In this paper, we consider specialized subclasses of the intersection graphs of *rays* (or half-lines) in the plane. These were introduced by Kostocha and Nešetřil [5]. Formally, a graph $G = (V, E)$ is said to be a *ray graph* when each vertex $v \in V$ can be represented by a ray $r_v = (p_v, d_v)$ originating from the point p_v and continuing endlessly in direction d_v such that uv is an edge if and only if r_u and r_v intersect. The class of ray graphs is easily seen as a subclass of *segment graphs*, that is, the intersection graphs of line segments in the plane. More specifically, ray graphs are a subclass of *unit segment graphs*, that is, the intersection graphs of line segments in the plane where every line segment has the same length.

We study *orthogonal ray graphs* where the directions of the rays are restricted to be axis-parallel. Formally, a graph $G = (V, E)$ is an *orthogonal ray graph (ORG)* if and only if:

(i) each vertex v can be represented by a ray r_v in the plane originating from point $p_v = (x_v, y_v)$, its *anchor*, and continuing endlessly in the direction $d_v \in \{\uparrow, \downarrow, \leftarrow, \rightarrow\}$,
(ii) no point p_u lies on a ray r_v with $u \neq v$ (in other words, no two anchors lie on the same horizontal or vertical line),
(iii) uv is an edge if and only if the ray originating from p_v continuing in the direction d_v intersects with the ray originating from p_u and continuing in direction d_u.

For an ORG G, we refer to such a collection $\{r_v \mid v \in V(G)\}$ as an *ORG-model* of G. We will often identify a vertex v with its ray r_v and use v to refer to r_v as well. Correspondingly, we use V also for the set of rays, and subdivide it into the sets $V_\uparrow$, $V_\downarrow$, $V_\leftarrow$, and $V_\rightarrow$ of rays pointing $\uparrow$, $\downarrow$, $\leftarrow$, $\rightarrow$, respectively; we call the rays in these sets upwards, downwards, left, and right rays, respectively. We also use $V_{\text{ver}} = V_\uparrow \cup V_\downarrow$ and $V_{\text{hor}} = V_\leftarrow \cup V_\rightarrow$. Additionally we order the rays pointing in one direction by length: For $i, j \in V_\uparrow$ we say that i is longer than j if $y_i < y_j$ (analogously defined for $V_\downarrow$, $V_\leftarrow$, and $V_\rightarrow$).

ORGs were introduced by Shrestha et al. [7] in connection with defect tolerance schemes for nano-programmable logic arrays [6,9]. We further distinguish



(a) ORG: C_{12} (b) 3DORG: C_6 (c) 2DORG: C_4

Fig. 1: The longest cycles in the different ORG models.

two subclasses of orthogonal ray graphs. The first is *2D-orthogonal ray graphs (2DORGs)* where, w.l.o.g., we restrict the directions of the rays to be $\uparrow$ or $\rightarrow$. 2DORGs were introduced by Shretha et al. [8]). Additionally, we refer to an ORG-model with two directions as a *2DORG-model*. Similarly, *3D-orthogonal ray graphs (3DORGs)* are ORGs where, w.l.o.g., we restrict the directions of the rays to be $\uparrow$, $\rightarrow$, or $\leftarrow$, and we refer to an ORG-model with three directions as a *3DORG-model*.

Observation 1 *The following induced cycles are forbidden in ORGs:*

(i) C_i *($i > 4$) is a minimal forbidden induced subgraph for 2DORG. This has been shown by Shreshta et al. [8].*
(ii) C_j *($j > 6$) is a minimal forbidden induced subgraph for 3DORG.*
(iii) C_k *($k > 12$) is a minimal forbidden induced subgraph for ORG. This has been proved explicitly by Shrestha et al. [8] and implicitly by Kostochka and Nešetřil [5].*

Note that these values are tight, that is, induced cycles of smaller lengths are representable; see Fig. 1.

2 Edge-Asteroids

An *edge-asteroid*, introduced by Feder et al. [1], is a set of edges $\{e_0, e_1, \ldots, e_{2k}\}$ such that, for each $i = 0, 1, \ldots, 2k$, there is a path containing e_i and $e_{i+1(mod\ 2k+1)}$ that avoids the neighbors of the end-vertices of $e_{i+k+1(mod\ 2k+1)}$. Shreshta et al. [8] have shown the following property of 2DORGs.

Theorem 1 ([8]). *A bipartite graph is 2DORG if and only if its complement is a circular arc graph.*

Feder et al. [1] have proven the following theorem via the equivalence between 2DORGs and posets with height 2 and interval dimension 2.

Theorem 2 ([1]). *A bipartite graph G is the complement of a circular arc graph if and only if it contains no induced cycles of length at least 6 and no edge-asteroids.*

Fig. 2: The structure of a 3DORG-model with two independent edges.

Two edges uv, $u'v'$ are *independent* when there is no edge xy for $x \in \{u, v\}$, $y \in \{u', v'\}$. A set of edges is called *independent* when all pairs of edges in the set are independent. For an edge uv, its neighborhood $N(uv)$ is the union of the neighborhoods of its endpoints, that is, $N(uv) = N(u) \cup N(v)$. We now make some observations regarding independent sets of edges.

Observation 2 *In a 3DORG-model R of a connected graph G, every set E^* of independent edges is linearly ordered $e_1 < e_2 < \ldots < e_k$ by the left-to-right order of the vertical rays. Moreover, for every $i \in \{2, \ldots, k-1\}$, $G \setminus N(e_i)$ is not connected.*

Proof. This is easy to see since every path from e_1 to e_k must pass through the neighborhood of every e_i ($i \in \{2, \ldots, k-1\}$). □

Observation 3 *Let G be a 3DORG with an independent pair of edges $uv, u'v'$ such that $G \setminus N(uv)$ and $G \setminus N(u'v')$ are both connected. Then, every 3DORG representation of G must correspond to one of the two structures depicted in Fig. 2 where there can be no ray strictly contained in the shaded regions.*

These observations lead to the following corollary.

Corollary 1. *If a graph G contains an independent triple of edges $\{e_1, e_2, e_3\}$, where $G \setminus N(e_i)$ is connected for every $i \in \{1, 2, 3\}$, then G is not a 3DORG.*

With Corollary 1, we can show that nearly all of the minimal forbidden induced subgraphs of 2DORGs are also not 3DORG graphs. There are only a few graphs which need to be considered separately. In particular, C_6 is a 3DORG, C_8 (and longer cycles) are not 3DORG, and there are two infinite families from the forbidden characterization of 2DORG which do not have a "bad" independent triple of edges. The base cases for these infinite families are depicted in Fig. 3. We denote the first family by G_n^1 and the second family by G_n^2.

Definition 1. *Consider a 3DORG drawing of a graph with vertex set V. Then, V_{ver} denotes the vertices represented by vertical rays and V_{hor} denotes the vertices represented by horizontal rays.*

Definition 2. *Consider a 3DORG drawing of a graph. Then, the x-coordinates imply an ordering of the vertical rays. Let a, b be vertices represented by vertical rays. Then, $a \prec b$ states that the ray of a is left of b's ray.*

Observation 4 *Let $G = (V, E)$ a graph, $uv \in E$ and $G' = (V', E') = G \setminus N(uv)$. If G' is connected the following statement holds for every 3DORG drawing of G (w.l.o.g. $u \in V_{\text{ver}}$): for all $a \in V'_{\text{ver}}$, either $a \prec u$ or $u \prec a$ holds. In other words: all vertices in V' represented by vertical rays have to be on the same side of u.*

Proof. Assume there is a 3DORG drawing in which this does not hold, that is, there are two vertices $a, c \in V'_{\text{ver}}$ with $a \prec u \prec c$. Then every path from a to c must involve a neighbor of u or v; see Fig. 4). This is a contradiction as $G \setminus N(uv)$ is connected. □

Observation 5 *Let a, b, c ne vertical rays with $a \prec b \prec c$ and let z be a common neighbor of a and c that is not adjacent to b. Then, every neighbor of b is adjacent to a or c.*

Equivalent statement: *If a and c have a common neighbor that is not adjacent to b, and b has a neighbor which is adjacent neither to a nor to c, then there is no 3DORG drawing for this graph with $a \prec b \prec c$.*

Proof. Up to the direction of z, there is only one 3DORG drawing of a, b, c and z; see Fig. 5. In this drawing, b inside the area spanned by a, c and z. Thus, every horizontal ray that intersects b also has to intersect a or c. □

The infinite families $\mathcal{G}^1 = (G^1_n = (V^1_n, E^1_n))_{n\in\mathbb{N}}$ and $\mathcal{G}^2 = (G^2_n = (V^2_n, E^2_n))_{n\in\mathbb{N}}$ are given by the following sets for $n \geq 3$, where $[k] = \{1, 2, \ldots, k\}$ [8]. The graphs are depicted in Fig. 6:

$$V^1_n = \{a_i, b_i, c_i, d_i \mid i \in [n]\}$$

$$\begin{aligned} E^1_n = {} & \{(c_i, a_j) \mid i \in [n], j \in [i-1]\} \cup \\ & \{(c_i, b_j) \mid i \in [n], j \in [i]\} \cup \\ & \{(d_i, a_j) \mid i \in [n], j \in [i]\} \cup \\ & \{(d_i, b_j) \mid i \in [n], j \in [i-2]\} \cup \\ & \{(d_1, b_n)\} \end{aligned}$$

$$\begin{aligned} V^2_n = {} & \{a_i, d_i \mid i \in [n-1]\} \cup \\ & \{b_i, c_i \mid i \in [n]\} \cup \{x, y\} \end{aligned}$$

$$\begin{aligned} E^2_n = {} & \{(c_i, a_j) \mid i \in [n], j \in [i-2]\} \cup \\ & \{(c_i, b_j) \mid i \in [n], j \in [i]\} \cup \\ & \{(d_i, a_j) \mid i \in [n-1], j \in [i]\} \cup \\ & \{(d_i, b_j) \mid i \in [n-1], j \in [i-1]\} \cup \\ & \{(y, b_n), (x, c_1)\} \end{aligned}$$

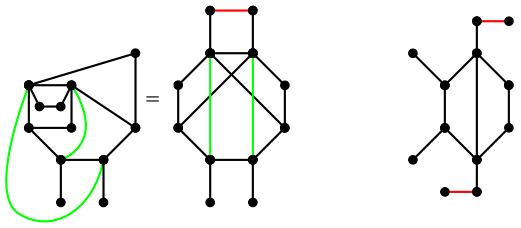


Fig. 3: Edge-asteroids without "bad" independent triples of edges. For the red edges uv, the graph $G \setminus N(uv)$ is connected, that is, we can use Observation 4.

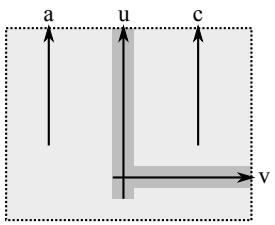


Fig. 4: Illustration of Observation 4.

Lemma 1. *The graphs in the families $\mathcal{G}^1$ and $\mathcal{G}^2$ cannot be represented by an 3DORG drawing.*

Proof. In the following, we give a detailed proof for the graphs in $\mathcal{G}^1$. We can use a similar argument for $\mathcal{G}^2$.

Assume there is a 3DORG drawing of G_n^1 for any $n \geq 3$. W.l.o.g., let $a_i, b_i \in V_{\text{hor}}$ and $c_i, d_i \in V_{\text{ver}}$ for $1 \leq i \leq n$. Applying Observation 4 for the edge $d_1 b_n$, we can infer that the rays for vertices $\{c_1, \dots c_{n-1}, d_2, \dots, d_n\}$ are on one side d_1, w.l.o.g., the right side. The ray for c_n also lies to the right of d_1—otherwisem the horizontal ray b_{n-1} connecting c_n and c_{n-1} would be adjacent to d_1). Hence, d_1 is the leftmost vertical ray.

We show $c_i \prec d_i$ for every $n \geq i \geq 1$ by induction: As d_1 and c_n have b_n as a common neighbor and d_n has neighbor a_n, Observation 5 claims that $d_1 \prec d_n \prec c_n$ cannot be the ordering of the rays. Because d_1 is the leftmost ray, c_n has to be left of d_n, that is, $c_n \prec d_n$. In the inductive step, by Observation 5 using b_{i-1} as the common neighbor of c_1 and c_{i-1}, and a_i as neighbor of d_i, it follows that $c_i \prec d_i \prec c_{i-1}$ is a forbidden ordering. Since, by induction, $c_i \prec d_i$, it follows that $c_{i-1} \prec d_i$. Further, the ordering $d_{i-1} \prec c_{i-1} \prec d_i$ is forbidden, as a_{i-1} is a common neighbor of d_{i-1} and d_i, and b_{i-1} is a neighbor of c_{i-1}. Thus, we conclude $c_{i-1} \prec d_{i-1}$. From this induction argument, we can deduce $c_1 \prec d_1$, which is a contradiction to d_1 being the leftmost vertical ray. □

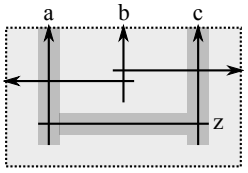


Fig. 5: Illustration of Observation 5. The ray for z is drawn without arrows because the direction does not matter.

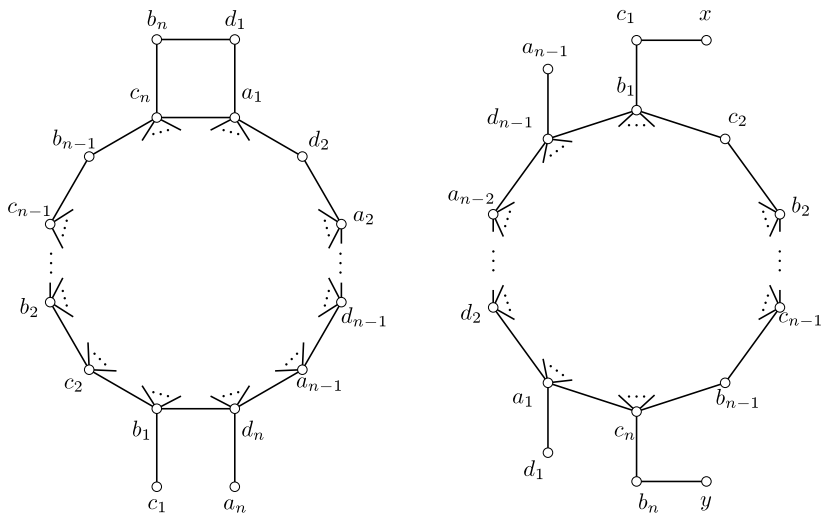


Fig. 6: Drawing of the graph families $\mathcal{G}^1$ (left picture) and $\mathcal{G}^2$ (right picture) [8].

3 Maximum Independent Set and Minimum Independent Dominating Set

In this section, we consider the problems Maximum-Weight Independent Set (MIS) and Minimum-Weight Independent Dominating Set (IDS). First, we show that MIS can be solved for 2DORG in $O(n \log n)$ time, using an algorithm of Felsner et al. [2] for Maximum-Weight Clique (MC) on trapezoid graphs. Then, we present an algorithm that solves Maximum-Weight Independent Set for 3DORGs in $O(n^2)$ time. Using this algorithm as a subroutine, we can solve MIS on 4DORGs in $O(n^4)$ time. With similar ingredients, we can solve MIS on the class of SegRay graphs (which is incomparable to the class of 4DORGs). Finally, we turn to IDS.

Theorem 3 ([2]). *MC on trapezoid graphs with given representation can be solved in $O(n \log n)$ time and $O(n)$ space.*

Corollary 2. Maximum-Weight Independent Set *on 2DORGs with given representation can be solved in $O(n \log n)$ time and $O(n)$ space.*

Proof. Let G be a graph with a 2DORG representation. The complement $\overline{G}$ is a trapezoid graph and its trapezoid representation can be computed in $O(n)$ time. Any maximum (weighted) independent set in G corresponds to a maximum (weighted) clique in $\overline{G}$. Felsner et al. [2] showed that MC can be solved in $O(n \log n)$ time and linear space using the trapezoid representation. □

Using the algorithm of Felsner et al. [2] as a subroutine, we can solve MIS on 3DORGs simply as follows. We "guess" the vertical ray r_i with the bottommost anchor p_i in a fixed optimal solution $V^\star$. This ray subdivides the plane into three

Algorithm 1: Compute values in table $T_{\uparrow}[i, j]$

```
1 Sort vertical rays by x-coordinate: v_1, ..., v_k from left to right
2 for i = 1 to k do
3     c ← 0
4     for j = i + 1 to k do
5         if v_j shorter than v_i then
6             c ← c + 1
7         else if v_j longer than v_i then
8             T↑[i, j] ← c
```

regions; see Fig. 8: the halfplane $H_{\mathrm{s}}[i]$ below p_i and the quarterplanes $Q_{\mathrm{nw}}[i]$ and $Q_{\mathrm{ne}}[i]$ to the left and right of r_i, respectively. Since our partition is based on the *bottommost* vertical ray r_i in $V^\star$, $H_{\mathrm{s}}[i]$ does not contain the anchor of any vertical ray in $V^\star$. Instead, all horizontal rays in $H_{\mathrm{s}}[i]$ are part of $V^\star$. Since we assumed that r_i is in $V^\star$, no ray with anchor in $Q_{\mathrm{nw}}[i]$ can point to the right and no ray in $Q_{\mathrm{ne}}[i]$ can point to the left. Hence, $V^\star$ consists of the solutions of the two independent 2DORG instances $(V_{\uparrow} \cup V_{\leftarrow}) \cap Q_{\mathrm{nw}}[i]$ and $(V_{\uparrow} \cup V_{\rightarrow}) \cap Q_{\mathrm{ne}}[i]$ and of all horizontal rays in $H_{\mathrm{s}}[i]$. Therefore, knowing the bottommost ray in $V^\star$, we can compute $V^\star$ in $O(n \log n)$ time. We can find the bottommost ray simply by testing each of the at most n candidates in V_{up}, which yields a running time of $O(n^2 \log n)$. We now show how to reduce this to $O(n^2)$ time.

Theorem 4. Maximum-Weight Independent Set *on 3DORGs with given representation can be solved in $O(n^2)$ time and $O(n^2)$ space.*

Proof. Let $G = (V, E)$ be a graph with a 3DORG representation. We use a dynamic program to find a maximum independent set S. In the following, we describe the version for maximum cardinality, but the algorithm can easily be adapted to maximize the weight of the independent set. We refer to a maximum independent set as a MIS.

For two vertical rays $i, j \in V_{\mathrm{ver}}$, with j being longer than i, we define the following values (see Fig. 7): $T_{\uparrow}[i, j]$ is the number of vertical rays between i and j that are shorter than i. $T_{\leftarrow}[i, j]$ is the number of horizontal rays k directed left with $x_k < x_j$ and $y_j < y_k < y_i$. Accordingly, $T_{\rightarrow}[i, j]$ is the number of horizontal rays k directed right with $x_k > x_j$ and $y_j < y_k < y_i$.

Algorithms 1 and 2 show how the tables $T_{\uparrow}[i, j]$ and $T_{\leftarrow}[i, j]$ are populated in $O(n^2)$ time. Clearly, $T_{\rightarrow}[i, j]$ can be filled up in the same manner.

For one vertical ray $j \in V_{\mathrm{ver}}$, we define the following values (see Fig. 8): $T_{\mathrm{nw}}[j]$ is the maximum cardinality of a set of left and upwards rays that start left of j and do not intersect each other. $T_{\mathrm{ne}}[j]$ is the maximum cardinality of a set of right and upwards rays that start right of j and do not intersect each other. $T_{\leftrightarrows}[j]$ is the number of horizontal rays that lie below j.

Obviously, $T_{\leftrightarrows}[j]$ can be computed in linear time for a given ray $j \in V_{\mathrm{ver}}$. The value $T_{\mathrm{nw}}[j]$ (and equally $T_{\mathrm{ne}}[j]$) requires a more difficult approach. We

Algorithm 2: Compute values in table $T_{\leftarrow}[i,j]$

```
Sort all rays by y-coordinate: v_1, ..., v_n from bottom to top
for j = 1 to n do
    if v_j is vertical then
        c ← 0
        for i = j + 1 to n do
            if v_i is horizontal directed left and starts left of v_j then
                c ← c + 1
            else if v_i is vertical and left of v_j then
                T_←[i, j] ← c
```

Algorithm 3: Compute MIS of a 3DORG instance

```
Compute tables T_↑, T_← and T_→, T_nw and T_ne, and T_⇆.
Using Equation (1), find the longest vertical ray j and hence an indep. set S
if |V_hor| > |S| then S ← V_hor
return S
```

show an algorithm that populates the table $T_{\mathrm{nw}}[j]$ for all $j \in V_{\mathrm{ver}}$ from left to right in total time $O(n^2)$. To get the value $T_{\mathrm{nw}}[j]$, we distinguish two cases: (A) there is no vertical ray left of j, (B) there is at least one vertical ray left of j. In case (A), we can select all horizontal rays left of j that are directed to the left. In case (B), we guess the longest vertical ray that is left of j, denoted by i. The maximum number of independent rays left of j including i as the second longest ray can now easily be computed from the values stored before (again, see Fig. 7). Note that no vertical rays in the bottom area can be selected (as they would be longer than i) and no horizontal rays in the top right area can be selected (as they would intersect i or j). The following formula describes the computation of $T_{\mathrm{nw}}[j]$.

$$T_{\mathrm{nw}}[j] = \max_{\substack{i \in V_{\mathrm{ver}} \\ i \text{ left of } j \text{ and shorter than } j}} (T_{\mathrm{nw}}[i] + T_{\uparrow}[i,j] + T_{\leftarrow}[i,j]) + 1$$

We now use the same technique to guess the longest vertical ray in the MIS, denoted by j. The selection of this longest vertical ray partitions the plane into three areas, see Fig. 8. Note that no vertical rays in the bottom area can be selected, and no horizontal rays in the top left or right area can intersect j. Hence, we can use the precomputed values for $T_{\mathrm{nw}}[j]$, $T_{\mathrm{ne}}[j]$, and $T_{\leftrightarrows}[j]$ to get the cardinality of a maximum independent set S:

$$|S| = \max_{j \in V_{\mathrm{ver}}} (T_{\mathrm{nw}}[j] + T_{\mathrm{ne}}[j] + T_{\leftrightarrows}[j]) + 1 \tag{1}$$

The complete algorithm for finding the MIS is summarized in Algorithm 3. As described for every single step of the algorithm, the total runtime is in $O(n^2)$.

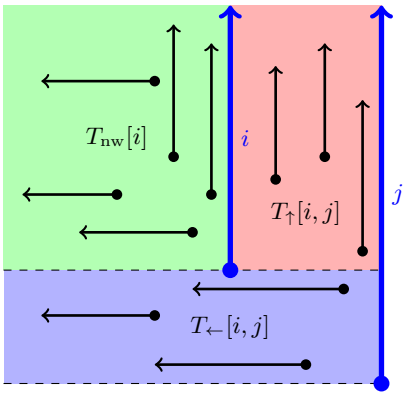


Fig. 7: The value of $T_{\text{nw}}[j]$ is composed of three parts. For computing $T_{\text{nw}}[j]$, we "guess" the second longest ray i to the left of ray j. This subdivides $Q_{\text{nw}}[j]$ into three regions that can be solved independently.

The tables for the dynamic program use $O(n^2)$ space. Finally, we check whether the set of all horizontal rays is larger than the set found before. □

Theorem 5. Maximum-Weight Independent Set *4DORGs with given representation can be solved in $O(n^4)$ time and $O(n^2)$ space.*

Proof. Let $G = (V, E)$ a graph with a 4DORG representation. In the following, we describe the version for maximum cardinality, but the algorithm can easily be adapted to maximize the weight of the independent set. The idea is to guess the longest rays $i \in V_{\leftarrow}$ and $j \in V_{\rightarrow}$ that are included in the MIS S. This selection splits up the instance into two 3DORG representations, see Fig. 9 resp. 10 (depending on the position of i and j). Algorithm 4 gives a formal description of our approach. The rays are partitioned into the sets $V_A, \ldots, V_G$ by the position of their anchor point.

In the first case (see Fig. 9) we filter the rays depending on their direction and association to one of the regions. For the upper 3DORG instance (consisting of the sets V_A, V_B, V_C, and V_D) we have the following restrictions: We cannot select any rays directed downwards as they would intersect i or j. We cannot select any rays directed right from V_A as they would be longer than j. Likewise, we cannot select any rays directed left from V_C as they would be longer than i. Finally, we cannot select vertical rays in V_D as they would intersect i or j. The restrictions for the lower instance are similiar. We split up the horizontal rays in V_D: rays directed to the right are asigned to the upper instance, those directed to the left are assigned to the lower instance. After that, we solve the upper and lower instance independently in $O(n^2)$ time using the algorithm from Theorem 4 and join them. This yields the MIS with respect to the chosen rays i and j.

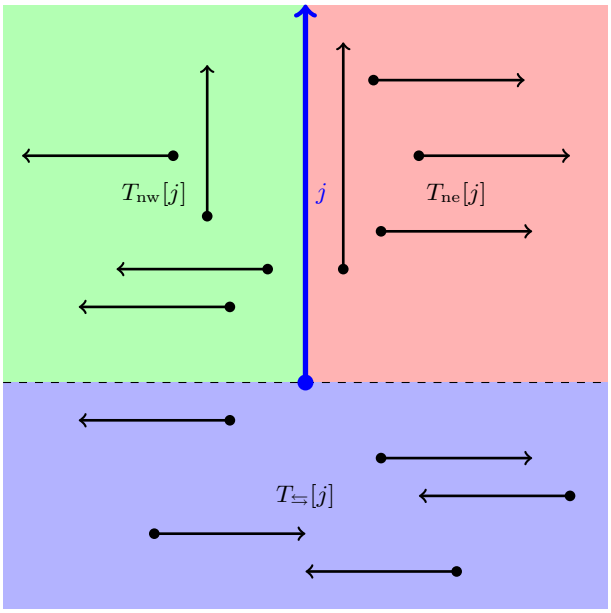


Fig. 8: The longest vertical ray j in a MIS decomposes the plane into three regions that can be solved independently.

In the second case (see Fig. 10) we filter the rays from V_A, V_C, V_E, and V_G in a similiar way and solve MIS for two independent 3DORG instances. After that, we add all vertical rays from V_D as they cannot intersect each other and any ray of the 3DORG instances.

In Algorithm 4 we assume that we always select at least one ray directed to the left and one ray directed to the right for our MIS. Therefore, we finally need to check if there is a better solution that avoids one of them or does not use horizontal rays at all.

The total runtime is in $O(n^4)$ as we need to solve the 3DORG instances for every pair of rays i, j. Filtering the rays to get the instances only needs linear time. We only need a constant amount of space to store the so far largest independent set S. So we only need $O(n^2)$ space in total like in Theorem 4. □

Theorem 6. Maximum-Weight Independent Set *on SegRay graphs with given representation can be solved in $O(n^3)$ time and $O(n^2)$ space.*

Proof. For SegRay graphs we use a similiar dynamic programming approach as in Theorem 4. We compute a two-dimensional table $T[i, k]$ where i and k are vertical rays and i is left of k. The entry $T[i, k]$ describes the maximum number of independent vertices that are completely contained in the rectangle $\{(x, y) \in \mathbb{R}^2 \mid x_i < x < x_k \text{ and } y > \max\{y_i, y_k\}\}$. We also allow $-\infty$ for i resp. ∞ for k to describe rectangles that are not bounded by a ray to the left resp. to the right.

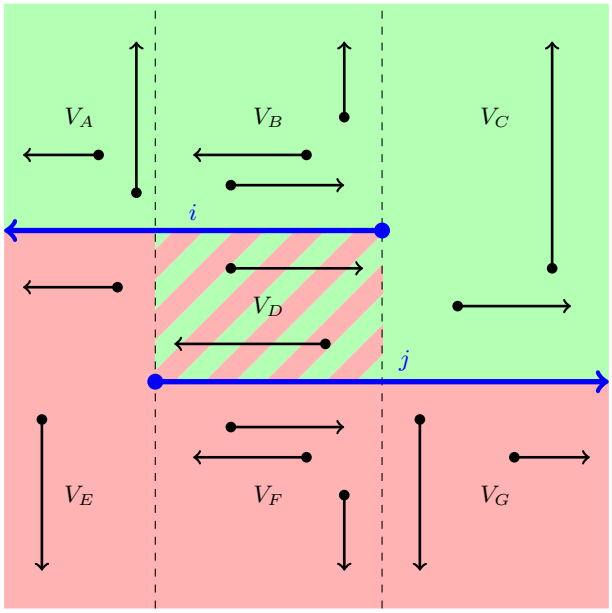


Fig. 9: $x_i > x_j$. Green: upper 3dorg-instance, red: lower 3dorg-instance. TODO. W.l.o.g. we assume that i is above j.

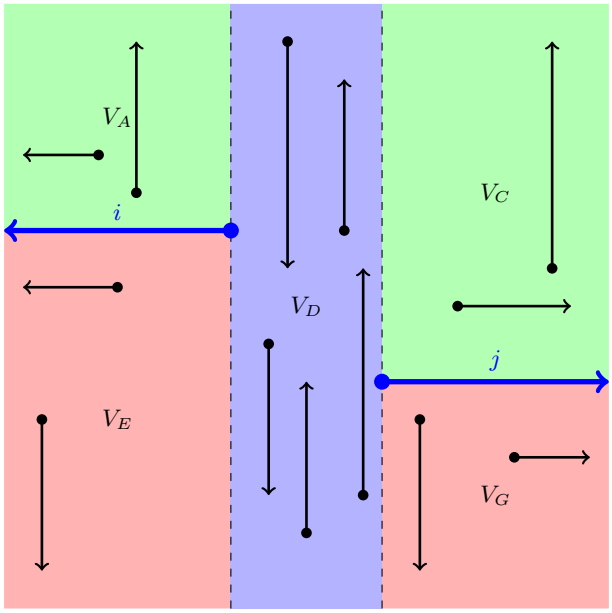


Fig. 10: $x_i < x_j$. TODO

Algorithm 4: Compute MIS of a 4DORG instance

```
1  S ← ∅
2  foreach i ∈ V← do
3      foreach j ∈ V→ do
4          if x_j < x_i then // see Fig. 9
5              V_upper ← (V_A ∩ (V← ∪ V↑)) ∪ (V_B \ V↓) ∪ (V_C ∩ (V→ ∪ V↑)) ∪ (V_D ∩ V→)
6              V_lower ← (V_E ∩ (V← ∪ V↓)) ∪ (V_F \ V↑) ∪ (V_G ∩ (V→ ∪ V↓)) ∪ (V_D ∩ V←)
7              S_upper ← MIS(V_upper)
8              S_lower ← MIS(V_lower)
9              S' = S_upper ∪ S_lower ∪ {i, j}
10         else // see Fig. 10
11             V_upper ← (V_A ∩ (V← ∪ V↑)) ∪ (V_C ∩ (V→ ∪ V↑))
12             V_lower ← (V_E ∩ (V← ∪ V↓)) ∪ (V_G ∩ (V→ ∪ V↓))
13             S_upper ← MIS(V_upper)
14             S_lower ← MIS(V_lower)
15             S' = S_upper ∪ S_lower ∪ {i, j} ∪ (V_D ∩ (V↑ ∪ V↓))
16         if |S'| > |S| then
17             S ← S'
18 return S
```

We start computing $T[i,k]$ for neighbouring vertical rays i and k by counting the number of horizontal segments between them. After that, we increase the interval between i and k step by step to fill the table. To compute the value $T[i,k]$ we try all candidates j for the longest ray in the considered rectangle. The selection of j decomposes the rectangle into three regions (see Fig. 11). To get the maximum number of independent vertices for a given j we need to add the previously computed values $T[i,j]$ and $T[j,k]$ with the number of horizontal segments below the anchor of j. We can compute this number for all feasible selections of j in linear time using a loop that runs through all vertices from the bottom to the top. We also have to consider the special case where we don't pick any vertical rays between i and k. We choose $T[i,k]$ as the maximum of these values.

Hence, we need linear time for each entry in the table and thereby $O(n^3)$ time to fill the whole table, which obviously needs $O(n^2)$ space. After the table is populated, the size of a MIS is stored in $T[-\infty,\infty]$. □

Next we present an efficient algorithm for the problem Minimum-Weight Independent Dominating Set on 3DORG graphs.

Theorem 7. Minimum-Weight Independent Dominating Set *on 3DORGs with given representation can be solved in $O(n^3)$ time and $O(n^2)$ space.*

Proof. For sake of simplicity, we show how to solve the unweighted version of the problem. However, our algorithm can be used to solve the weighted version

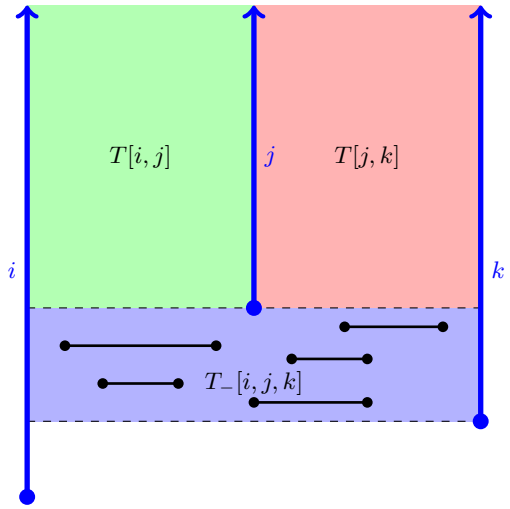


Fig. 11: The value of $T[i,k]$ can be computed by guessing the longest vertical ray j between i and k. This decomposes the area in three regions, which can be solved independently.

by simply changing the number of rays to the sum of their weights. We refer to a minimum independent dominating set as an IDS.

Let $G = (V, E)$ be a graph with a 3DORG representation. We use a similar dynamic programming approach as in Theorem 4. We guess the longest vertical ray $j \in V_{\text{ver}}$ of the IDS. The selection of this longest vertical ray partitions the plane into three areas, and the rays into four types; see Fig. 12. Since the horizontal rays in the bottom area lie completely below j, they cannot be dominated by a vertical ray and have to lie in the IDS (refer to the blue rays). Because of the independence constraint, the rays that intersect another ray that lies in the IDS cannot lie in the IDS themselves (refer to the gray dotted rays). The top left and top right area define two subproblems of IDS that can be solved independently (refer to the black solid rays). However, there can be vertical rays whose anchor lies below j and that are not dominated by a horizontal ray in the bottom area (refer to the black dashed rays); we call these rays *uncovered*. The uncovered rays have to be dominated by a solution to the subproblems in the top left and top right area.

Hence, a solution to the top left area looks for an IDS of left and vertical rays in this area that also dominates all uncovered rays. We do not have to consider all of them, though. To this end, observe that any left ray in the top left area that dominates the rightmost uncovered ray i that lies left of j dominates all uncovered rays. Thus, we only have to look for a solution to the top left area that also dominates i. The same argument can be applied to the top right region with the leftmost uncovered ray k that lies right of j.

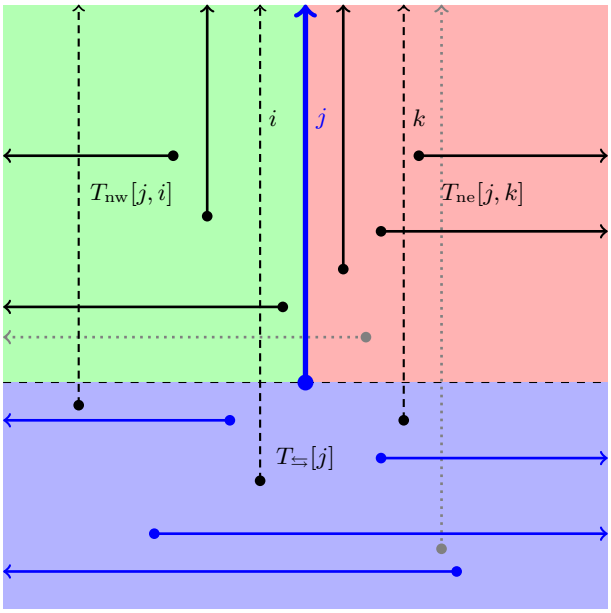


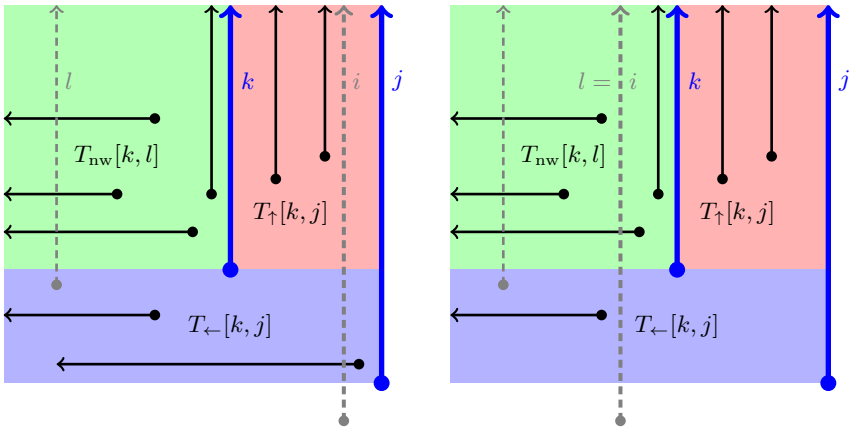
Fig. 12: The longest vertical ray j in a MIDS decomposes the plane into three regions with four types of rays: blue rays have to lie in the MIDS, gray dotted rays are dominated by blue rays, black rays are not dominated by blue rays, but black dashed rays cannot lie in the MIDS.

Let $j \in V_{\mathrm{ver}}$ be a vertical ray. We define $T_{\leftrightarrows}[j]$ as the number of horizontal rays that lie below j. For any $i \in V_{\mathrm{ver}}$ with $x_i < x_j$ and $y_i < y_j$, we define $T_{\mathrm{nw}}[j,i]$ as the minimum cardinality of a set of left and vertical rays that start left of j, do not intersect each other, intersect i, and intersect all other left and vertical rays that start left of j. Analogously, for any $k \in V_{\mathrm{ver}}$ with $x_j < x_k$ and $y_k < y_j$, we define $T_{\mathrm{nw}}[j,i]$ as the minimum cardinality of a set of right and vertical rays that start right of j, do not intersect each other, intersect k, and intersect all other right and vertical rays that start right of j.

Note that, at the top level, there is a unique rightmost (leftmost) uncovered ray l_j (r_j) that lies left (right) of j. Hence, the cardinality of an IDS is:

$$\max_{j \in V_{\mathrm{ver}}} \left(T_{\mathrm{nw}}[j,l_j] + T_{\mathrm{ne}}[j,r_j] + T_{\leftrightarrows}[j]\right) + 1 \tag{2}$$

We proceed by showing how to calculate a table entry $T_{\mathrm{nw}}[j,i]$. Again, we guess the longest vertical ray k with $x_k < x_j$ and $y_k > y_j$. This partitions $Q_{\mathrm{nw}}[j]$ into three regions; see Fig. 13. First, in the (red) region horizontally between k and j and above the anchor of k, all horizontal rays are dominated by k or j and cannot lie in the IDS. Thus, all vertical rays in this area have to lie in the IDS. We denote the number of these rays by $T_{\uparrow}[k,j]$. Second, in the (blue) region vertically between k and j and left of j, no vertical rays lie in the IDS, since they



(a) Case 1: $x_k < x_i < x_j$ (b) Case 2: $x_i < x_k$

Fig. 13: For computing $T_{\text{nw}}[j,i]$, we guess the longest ray k to the left of ray j. This subdivides $Q_{\text{nw}}[j]$ into three regions that can be solved independently.

are longer than k. Since there cannot be any vertical rays that intersect this region, all left rays have to lie in the IDS; the right rays are dominated by j. We denote the number of these rays by $T_{\leftarrow}[k,j]$. Finally, the vertical and left rays in the (green) region top left of the anchor of k form an independent subproblem together with the rightmost uncovered ray l that lies left of k. The solution of this subproblem corresponds to the table entry $T_{\text{nw}}[k,l]$.

However, the subproblem corresponding to $T_{\text{nw}}[j,i]$ might be infeasible. First, if there is an uncovered ray r with $x_k < x_r < x_j$, then the IDS contains no horizontal ray that intersects r, since the (red) region right of k allows only vertical rays; a contradiction. Thus, assume that all uncovered rays lie left of k. We distinguish between two cases.

Case 1: $x_k < x_i < x_j$; see Fig. 13a. In this case, there has to be a left ray in the (blue) area below k that intersects i; otherwise, i is uncovered, which contradicts the assumption.

Case 2: $x_i < x_k$; see Fig. 13b. In this case, i is either dominated by a left ray in the (blue) area below k, or by a left ray in the (green) area left of k. In the latter case, it can be that $l = i$, that is, i is the rightmost uncovered ray.

It remains to describe the case that the IDS contains no vertical ray k with $x_k < x_j$ and $y_k > y_j$. In this case, all left rays in the quadrant $Q_{\text{nw}}[j]$ lie in the IDS. However, this only yields an IDS if these left rays intersect i and all vertical rays in $Q_{\text{nw}}[j]$.

Because of the numerous cases and the necessity to check whether the choice of the next longest segment k yields an independent dominating set, we restrain from giving the detailed recursion formula. By using the same approach as in the proof of Theorem 4, we can populate the tables $T_{\leftarrow}$ and $T_{\uparrow}$ in $O(n^2)$ time.

Note that the rightmost uncovered ray l depends only on i, j and k; hence, we have only have to compare look at one entry of T_{nw} for each k. Assuming that we have precomputed a table $U[k, j]$ that contains the rightmost uncovered ray in the (blue) area vertically between k and j and left of j, we can find the rightmost uncovered ray for the next recursion step in constant time by comparing the x-coordinate of i (if it is uncovered) with the x-coordinate of $U[k, j]$.

To populate the table U, we once sort all vertical rays non-increasingly by their x-coordinate, and then all left rays non-increasingly by the x-coordinate of their anchor. Now, in order to compute a table entry $U[k, j]$, we walk through the two sorted lists in parallel. Every time we encounter a vertical ray r that lies in the (blue) area vertically between k and j and left of j, we look at the horizontal rays in this area by walking through the other list until we encounter the left-most left ray whose anchor lies right of r, and we maintain the highest y-coordinate of these left rays. Then, we can check whether r is intersected by a left ray by just comparing the y-coordinates. As soon as we find an uncovered ray, we save it in the table; otherwise, we save *nil*. Clearly, we have to traverse the list of vertical rays and the list of left rays once per table entry, so we require $O(n^3)$ time to populate the table.

Hence, we can populate the table $T_{\mathrm{nw}}[j, i]$ in $O(n^3)$ time. The table $T_{\mathrm{ne}}[j, k]$ can be populated analogously in $O(n^3)$ time. Since $T_{\leftrightarrows}$ can be computed in $O(n^2)$ time analogously to $T_{\leftarrow}$, we can find an IDS using Equation 2 in $O(n^3)$ time and $O(n^2)$ space.

□

4 Feedback Vertex Set

For a graph $G = (V, E)$, a subset V' of V is called a *feedback vertex set* if $G \setminus V'$ is acyclic, i.e., if $G \setminus V'$ is a forest. In the Minimum-Weight Feedback Vertex Set (FVS) problem one is given a weighted graph G and asked to determine a feedback vertex set with minimum weight. Note that finding a minimum feedback vertex set is equivalent to finding a maximum induced forest. FVS is among the first problems shown to be NP-complete by Karp [3].

Kloks et al. [4] recently showed how to solve unweighted FVS efficiently on chordal bipartite graphs, but they don't provide any explicit time bound. It is unclear whether their algorithm can be generalized to the weighted case. The chordal bipartite graphs are the bipartite graphs without induced k-cycles for $k \geq 6$. They are known to be a superclass of 2DORG, but do not contain 3DORG due to C_6 being a 3DORG.

In this section, we show that the FVS problem is efficiently solvable on ORGs. We begin by providing an $O(n^7)$ algorithm for a generalization of Minimum-Weight Feedback Vertex Set on 2DORG.

Theorem 8. *The* Minimum-Weight Feedback Vertex Set *problem can be solved in polynomial time on 4DORGs.*

As is often the case regarding the FVS problem, for our algorithms it is more convenient to discuss induced forests rather than feedback vertex sets. Our algorithms also easily generalize to the weighted case, but, for simplicity, we discuss the cardinality case.

We will solve the following generalization of the maximum induced forest problem on 2DORGs, then reduce the 3DORG case to it, and finally reduce the 4DORG to the 3DORG case.

Problem 1. Generalized 2DORG Maximum Induced Forest.
Input: A set of upward rays U, a set of rightward rays R, and a set W of downward and leftward rays.
Output: A subset F of $U \cup R$ such that the intersection graph of $F \cup W$ is acyclic and F is maximum.

The technique we use is similar to our approach for the Maximum-Weight Independent Set problem. Note that a maximum induced forest F of a ORG G always contains at least one vertical ray since a cycle requires at least two vertical rays. With this in mind we recursively describe the maximum induced forests F of an ORG G with respect to the longest vertical ray i in F together with some other related rays.

We summarize our dynamic programming approaches to produce a maximum induced forest F as follows. For a 2DORG G, F can be decomposed into disconnected forests by considering its longest upward ray u and u's two "highest" neighbours. For a 3DORG G, F is described by considering the three longest upward rays in F together with two neighbours of the middle ray. This set of up to 5 rays in F partition F into two generalized 2DORG subproblems on the left and right of the middle ray. The 4DORG case is similarly described by up to three pairs of "overlapping" upward and downward rays and up to 5 neighbours of the middle pair of rays. This set of up to 11 initial rays decompose F into maximum induced forests of two generalized 3DORG subproblems on the left and right of the middle pair.

4.1 2DORG

For a 2DORG G with upward rays U and rightward rays R, we describe the recursive structure of every maximum induced forest of G so that a maximum induced forest in G may be determined via dynamic programming. Consider a maximum induced forest F in G. We recursively decompose F into a collection of maximum subforests on disjoint induced subgraphs of G. From this discussion we will see that a maximum induced forest of G can be constructed by considering maximum induced forests of $O(n^5)$ induced subgraphs of G. This will further allow us to compute a maximum induced forest of G in $O(n^7)$ time via dynamic programming.

We begin by decomposing F with respect to its longest upward ray i. To do this, we consider three cases regarding the neighbours of i which belong to F. Namely, (1) when i has no neighbours in F, (2) one neighbour p in F, and

(3) at least two neighbours p, q in F. In the third case, we consider p and q to be the highest and second highest neighbours of i respectively. These cases are depicted in Fig. 14 and discussed as follows. While considering these cases we will see that two additional cases need to be considered (these are discussed as (4) and (5) below).

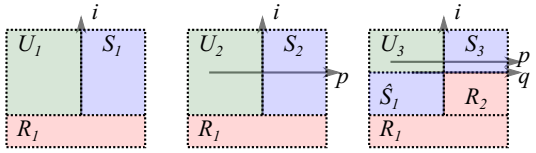


Fig. 14: The cases of the initial recurrence for FVS on 2DORG when including the ray i in F. From left-to-right when the ray i has: no neighbours in F, one neighbour in F, and at least two neighbours in F.

(1) $N(i) \cap F = \emptyset$. Consider the three rectangular regions as depicted in the first drawing of the figure: the green region (U_1: the upper-left quarter-plane with i as its right boundary), the red region (R_1: the lower half-plane whose upper boundary is the y-coordinate of i), and the blue region (S_1: the upper-right quarter-plane with i as its left boundary). Since i has no neighbours, F does not contain any rightward rays originating in U_1, i.e., F contains all upward rays from U_1. Also, since i is the longest upward ray in F, no upward ray from R_1 belongs to F, i.e., F contains all rightward rays in R_1. Furthermore, no ray in F originates outside of S_1 and intersects S_1. Thus, the remainder of F is a maximum induced forest using rays strictly contained within S_1. In particular, F decomposes into $\{i\}$, all upward rays in U_1, all rightward rays in R_1, and a maximum induced forest from S_1.

(2) $N(i) \cap F = \{p\}$. Here F similarly decomposes into three regions U_2, R_1, and S_2 (as depicted in the second drawing of the figure). Notice that p intersects the region U_2, but this does not prevent any upward ray in U_2 from belonging to F, i.e., all upward rays in U_2 belong to F regardless of the presence of p. Additionally, the region R_1 is as in (1) and we again have all rightward rays from R_1 in F. Now the remainder of F consists of rays contained in the blue region S_2. Notice that F cannot simply contain any maximum induced forest from S_2 since such a forest could create a cycle with p (i.e., we must include p as a part of S_2). Thus, F decomposes into $\{i, p\}$, the upward rays in U_2, the rightward rays from R_1, and a maximum induced forest from S_2 subject to not introducing a cycle with p. We call such a ray p which passes through a region a *crossing ray*.

(3) $|N(i) \cap F| \geq 2$. Let p and q be the highest rightward rays adjacent to i in F where $p_y > q_y$. Now F decomposes into five regions U_3, S_3, $\hat{S}_1$, R_1, and R_2 with respect to i, p, and q as depicted in the third drawing of the figure. The region

R_1 is as before and again F includes all rightward rays from it. However, the left and right sides of i are now split into the regions above q and below q. In particular, on the left of i and above q is U_3 in which F can only have upward rays since this is above the highest rightward ray of i. Also, on the right of i and below q is the region R_2 and F cannot contain any upward rays from R_2 otherwise such a ray would form a 4-cycle with i, p, and q. Thus, F contains all rightward rays from R_2. Now, similar to the region S_2 from (2), we observe that F contains a maximum induced forest F' from S_3 subject to F' including the crossing ray p. Finally, the region $\hat{S}_1$ may also contain both upward and rightward rays from F. However, rays in this region may be involved in cycles in G with any of i, p, or q. In particular, F contains a maximum induced forest F'' from $\hat{S}_1$ subject to including the "top" rays p and q and the right-bounding ray i. Thus, in this case, F consists of $\{i, p, q\}$ together with the rightward rays from R_1 and R_2, the upward rays from U_3, and the maximum induced forests F' from S_3 and F'' from $\hat{S}_1$.

The above discussion of the initial cases has provided us with further subcases which need to be explored (namely, the S and $\hat{S}$ regions). Notice that, S_1 and S_2 are special cases of S_3. In particular, they correspond to allowing the crossing ray p and bottom ray q of S_3 to be optional. With this in mind we first examine the structure of F inside S_3. This discussion will produce more general $\hat{S}$-like subproblems and they will be discussed afterwards.

(4) Decomposing F inside S-type regions. The S-type regions are parameterized by a required left-side upward ray l, and optionally by a bottom rightward ray b and a crossing rightward ray c (as depicted in the first drawing of Fig. 15). As in the initial case, F will always have at least one upward ray contained within S_3 since a single upward ray from S_3 together with l, b, c and all rightward rays from S_3 cannot form a cycle. Thus, we again enumerate the cases regarding the longest ray i in F from this region and when i has zero, one, or at least two neighbours in F. These cases are given by the other seven drawings of the figure and described in (4.1), (4.2), and (4.3) as follows.

(4.1) $N(i) \cap F = \emptyset$. This case follows similarly to (1) and is drawn second in the figure. The differences here are that: the region U_4 is now described by two upward rays forming its left and right boundary, and the boundary of region R_3 is described by l, b, and i (note that the crossing ray does not play a role here). Namely, F is $\{l, c, b, i\}$ together with the upward rays from U_4, the rightward rays from R_3, and a maximum induced forest from S_1.

(4.2) $N(i) \cap F = \{p\}$. This case follows similarly to (2) with the differences as in (4.1). The third and fourth drawings in the figure depict the two cases regarding whether p is equal to c. Notice that U_5 is to U_4 as U_2 is to U_1. In particular, U_5 simply has an additional rightward ray which intersects it. Again, this ray does not prevent F from including all upward rays from U_5, and U_5 is described by the two upward rays forming its left and right boundary. Additionally, R_4

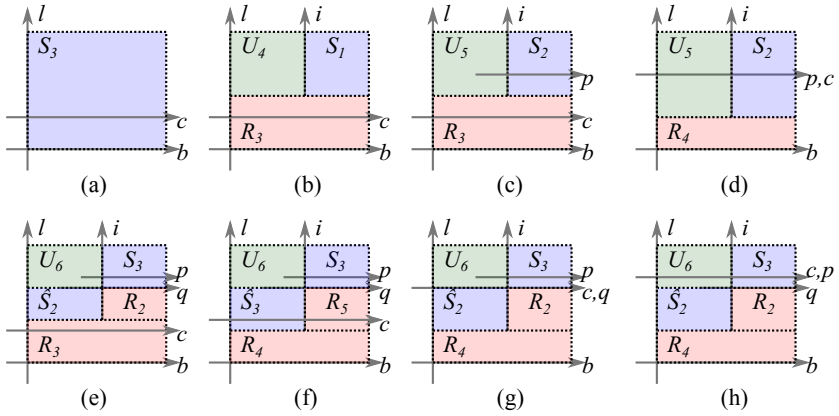


Fig. 15: The cases when recursing on an S_3 type subproblem regarding including the ray i in F. (a) The initial state, (b) $N(i) \cap F = \emptyset$, (c–d) $N(i) \cap F = \{p\}$, and (e–h) $|N(i) \cap F| \geq 2$.

is simply the special case of R_3 where the crossing ray c is absent. Thus, F is $\{l, c, b, p, i\}$ together with the upward rays from U_5, the rightward rays from R_3 (or R_4), and a maximum induced forest from S_2 subject to the inclusion of the crossing ray p.

(4.3) $|N(i) \cap F| \geq 2$. This case follows similarly to (3), where we again select p and q to be the highest neighbours of i. This results in four subcases regarding whether c is distinct from p and q (these are shown as the four drawings in the lower row of the figure).

The red region (labelled R_3 or R_4 depending on the whether c crosses through it) is as before and again F includes all rightward rays from it. The boundary of this region is described by l (on the left), i (on the top), and b on the bottom. The left-side and right-side of i are now split into the green region (U_6), the red region (R_2 or R_5), the upper-right blue region (S_3), and the lower-left blue region ($\hat{S}_2$ or $\hat{S}_3$).

Notice that the upper-right blue region is exactly as in (3) and as such we also label it S_3 and note that, as before, F includes a maximum induced forest from S_3 subject to the inclusion of the crossing ray p.

The green region (U_6) is described by two upward rays (l and i) and a bottom ray q. As usual, F includes all upward rays from U_6 since U_6 is above q and left of i. The red region (R_2 or R_5 depending on whether c crosses through it) is described by its left-side upward ray i and its top ray q. As in (3), this region (and the space below it) does not include upward rays due to i, p, and q being in F. Thus, F contains all rightward rays from it.

Finally, for the lower-left blue region ($\hat{S}_2$ or $\hat{S}_3$ depending on whether c crosses through it), we note that it is again as in (3) except that is now has a left boundary ray l and possible crossing ray c. Thus, F is composed of $\{l, i, b, c, p, q\}$

together with the upward rays from U_6, the rightward rays from the red regions (R_2, R_3, R_4, or R_5), a maxium induced forest from S_3 subject to the inclusion of the crossing ray p and maybe c, and a maximum induced forest from $\hat{S}_2$ maybe subject to the inclusion of c.

(5) Decomposing F inside $\hat{S}$-type regions. The $\hat{S}$-type regions are parameterized by a required right-side upward ray r, and required top rays t and t' together with an optional left-side upward ray l, and an optional crossing ray c (as depicted in the first drawing of Fig. 16). Notice that, $\hat{S}_1$ and $\hat{S}_2$ are special cases of $\hat{S}_3$. We again enumerate the cases regarding the longest ray i in F from this region and the neighbours of i in F. However, i cannot have more than one neighbour in F since all of its neighbours are also neighbours of r, i.e., any two neighbours of i together with r would form a 4-cycle. Thus, it is possible that F contains no upward ray from this region. This provides three cases in total to consider. Namely, (5.1) when F has no upward ray, and when the longest upward ray from this region in F has (5.2) zero or (5.3) one neighbour.

(5.1) F has no upward ray in this region. In this case we claim that F contains all rightward rays in the region (labelled R_6 in the second drawing of the figure). In particular, since R_6 contains no upward rays, the neighbourhood of each rightward ray p is a subset of the neighbourhoods of both t and t'. However, t' and t can only have one common neighbour in F (namely, r). Thus, the neighbourhood of each p in F is guaranteed to be precisely r, and as such from this region F includes $\{l, r, t, t', c\}$ and all rightward rays.

(5.2) The longest ray i in F from this region has no neighbours in F. This case follows similarly to (4.1) and is drawn third in the figure. Similar to U_4, the region U_7 is bounded by l, i, and additionally the ray t' describes its maximum y-coordinate. Again, F contains all upward rays from this region. In the region R_7 F will include all rightward rays for the same reason that F includes all rightward rays from R_6 as in (5.1). Finally, in the region $\hat{S}_2$ we see two top rays t and t', a right-side upward ray r, and the left-side upward ray i. Thus, F is composed of $\{l, i, r, t, t', c\}$, all upward rays from U_7, all rightward rays from R_7, and a maximum induced forest from $\hat{S}_2$ subject to $\{i, t, t', r\}$.

(5.3) The longest ray i in F from this region has one neighbour p in F. This case follows similarly to (4.2) and (5.1). The fourth, fifth, and sixth drawings in the figure depict the cases regarding whether p is equal to c or t' (further cases are omitted due to redundancy). As usual, F contains all upward rays from the left side of i (the green regions: U_7 or U_8), and all rightward rays from the region below i (the red regions: R_7 or R_8). Moreover, at most three rays are needed to describe these regions. Finally, in the blue region ($\hat{S}_2$ or $\hat{S}_3$) we see the same $\hat{S}$ type subproblem we have encountered before. Thus, F is composed of $\{l, i, p, r, t, t', c\}$, all upward rays from the green region, all rightward rays from the red region, and a maximum induced forest from the blue region subject to $\{i, t, t', r, p\}$.

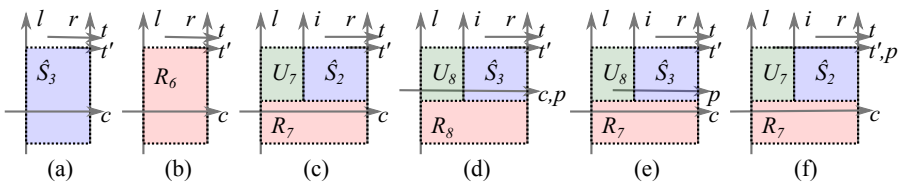


Fig. 16: The cases when considering the longest ray i in an $\hat{S}_5$-type subproblem with right-boundary r and optional left-boundary l, crossing ray c, and top rays t, t'. (a) The initial state, (b) F has no upward ray in this region. (c–f) The ray i has (c) no neighbors in F, (d) one neighbor p in F and $p = c$, (e) one neighbor p in F and $p \neq c$ and $p \neq t'$, and (f) one neighbor p in F and $p = t'$.

The Dynamic Programming Algorithm. Notice that throughout the above discussion we have shown that a maximum induced forest can be described by enumerating all regions of the following types: U_1, ..., U_8, R_1, ..., R_8, S_1, S_2, S_3, $\hat{S}_1$, $\hat{S}_2$, and $\hat{S}_3$. It is easy to see that these can all be built with a bottom up approach by considering the "small" regions first. We do not provide the explicit algorithm as it is a somewhat tedious and uninteresting exercise to extract it from the above discussion. However, it is easy to see that it will have a polynomial running time since each subproblem is described by at most 5 rays. We provide a more precise analysis in the next paragraph.

Complexity Considerations. Notice that, the U and R type subproblems are all parameterized by at most 4 rays, i.e., there are at most $O(n^4)$ of these subproblems. Additionally, to answer a U or R subproblem simply involves counting the upward or rightward rays (respectively) in the corresponding region, i.e., all such subproblems can be answered in $O(n^5)$ time. The S and $\hat{S}$ type subproblems are parameterized by at most 3 rays and at most 5 rays (respectively). Furthermore, to solve such a subproblems involves considering solutions to $O(n^3)$ and $O(n^2)$ "smaller" subproblems respectively, i.e., computing these tables takes $O(n^6)$ and $O(n^7)$ time (respectively). Thus, the total time to compute all entries in our tables is $O(n^7)$, i.e., a maximum induced forest (or FVS) of a 2DORG can be computed in $O(n^7)$ time.

Augmentation to handle the Genearlized 2DORG maximum induced forest problem. Notice that our treatment of the cases in the above discussion can be easily modified to handle any forest F' of downward and leftward rays which is required to be a subforest of F. In particular, we still follow the same process of recursively building the subproblems by considering the longest upward ray and its highest two neighbours. The presence of the elements of F' will limit the possible choices for i and its neighbours in a similar way that we have already seen with the $\hat{S}$ subproblems. Additionally, in the regions labelled U and R, we can no longer simply take all upward or rightward rays and need to take extra care to find a maximum set without forming cycles with elements of F'.

4.2 3DORG

Consider a maximum induced forest F in a 3DORG G with upward rays U, leftward rays L, and rightward rays R. The basic idea is that when F contains at least three upward rays, we can use the three longest upward rays to decompose F into two 2DORG subforests. Furthermore, when F contains less than three upward rays, we can easily construct it by brute force. Specifically, if F contains only one upward ray i then it also contains all of the horizontal rays, i.e., $F = \{i\} \cup R \cup L$. Furthermore, if F contains exactly two upward rays (i, j), then it contains at most one horizontal ray intersecting both i and j and all horizontal rays which do not intersect both i and j. Thus, these initial cases of maximum induced forests with at most two upward rays are trivially measured in $O(n)$ and $O(n^3)$ time respectively. It remains to consider the case when there are at least three upward rays in F.

Consider the three longest upward rays i, j, k in F where $i_x < j_x < k_x$. The subcases are distinguished by the neighbours of the middle ray j. Notice that, j can have at most one neighbour in common with i and at most one neighbour in common with k. We further distinguish the case when j is the shortest ray or otherwise. The subcases are depicted in Fig. 17 and enumerate as follows.

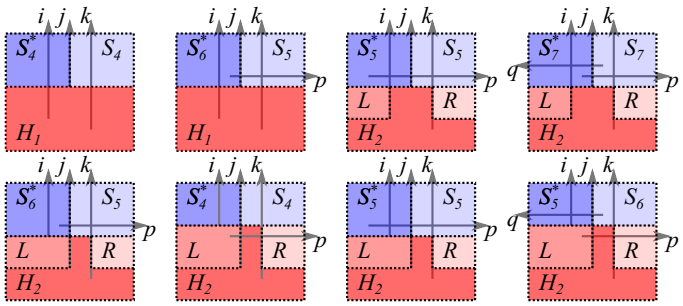


Fig. 17: The cases when the maximal induced forest contains at least three upward rays.

(1) j is shorter than i and k. These cases are given in the top row of the figure and further subcased depending on the neighbourhood of j as follows.

(1.1) j has no neighbours in F. F partitions into three regions: the dark blue region S_4^* on left of j, the blue region S_4 on right of j, and the red region H_1 below j. Since i, j, and k are the longest upward rays in F, there are no upward rays in H_1. However, F cannot simply contain all horizontal rays from H_1 since i and k may end up with two common neighbours. Thus, F restricted to H_1 is simply one horizontal ray h which intersects both i and k together with all horizontal rays which intersect at most one of i and k. It is easy to see that F

will further contain a maximum induced forest F' from S_4 consisting of upward and rightward rays subject to the inclusion of k. Similarly, F will contain a maximum induced forest F'' from S_4^* consisting of upward and leftward rays subject to the inclusion of i. In particular, F consists of $\{i, j, k, h\}$ together with F' and F''.

(1.2) j has a neighbour in common with k in F, but j has no common neighbour with i. F again partitions into three regions: the dark blue region S_6^* on left of j, the blue region S_5 on right of j, and the red region H_1 below j. The region H_1 is as in (1.1). The difference here is that S_6^* now has the rightward ray p which intersects it. In particular F contains a maximum induced forest from S_6^* using upward and leftward rays and subject to the inclusion of the rightward ray p and the upward ray i.

(1.3) j has a neighbour in common with i and k in F. This consists of two subcases. Namely, when there is a single horizontal ray p adjacent to all three, and when there are two separate rays p adjacent to j and k and q adjacent to i and j. These are depicted in the third and forth drawings of the figure. Essentially they have the same structure. Notice that below j there are three red regions L, H_2, and R. In particular, since i and k are already connected above j, they cannot have a common neighbour below j. Thus, F contains the leftward rays from L, rightward rays from R, and all horizontal rays from H_2.

(2) (wlog) i is shorter than j and k. These cases are depicted in the lower row of the figure. We omit the case when j has no common neighbour with either i or k as this is the same as (1.1). We distinguished according to: when j and k have a common neighbour but j and i do not have a common neighbour (first two cases); and when both i and j, and j and k have common neighbours (second two cases). Here we again obtain a trivial subproblem below the "shortest" of our longest rays (i.e., this is now i instead of j), and two generalized 2DORG subproblems. This follows similarly to (1.2) and (1.3).

Dynamic Programming and Complexity Considerations. Notice that we enumerate up to three longest rays and up to two of their neighbours. In each non-trivial case of the enumeration, we obtain one trivial subproblem (i.e., "below" the) and two generalized 2DORG subproblems. In particular, using the approach from the previous section, we can find a maximum induced forest in a 3DORG G in polynomial time.

4.3 4DORG

Consider a maximum induced forest F in a 4DORG G with upward rays U, downward rays D, leftward rays L, and rightward rays R. We describe how F partitions into a set of up to 11 rays and two maximum 3DORG subforests anchored from disjoint regions of the plane, i.e., by enumerating all possible sets of 11 rays (plus some additional considerations) and applying our 3DORG FVS

algorithm on the appropriate disjoint regions, we can solve FVS on 4DORG in polynomial time.

The idea here is based on *overlapping pairs* of upward/downward rays in F. Specifically, an upward ray i and a downward ray i' are an overlapping pair when the y-coordinate of i is less than the y-coordinate of j. We further call an overlapping pair (i, i') *close* when no vertical ray between i and i' forms an overlapping pair with either i or i'.

We consider up to three disjoint *close* overlapping pairs in F and show how F decomposes into 3DORG maximum forests in each of the cases. The first two cases (when F has no overlapping pairs or just "one" overlapping pair) are discussed as (1) and (2) and depicted in Fig. 18. We then discuss the case when F has "two" overlapping pairs (see (3)), and finally when F has at least three disjoint overlapping pairs (see (4)).

(1) F has no overlapping pairs. Consider the longest upward ray u and longest downward ray d. These two rays decompose F into two 3DORG subproblems and one trivial subproblem (as depicted in the first drawing of Fig. 18). Specifically, one 3DORG subproblem T_U is the half-plane above the y-coordinate of u, and the other subproblem T_D is the half-plane below the y-coordinate of d. Note that these two half-planes are disjoint and F will include all horizontal rays from the space H between them.

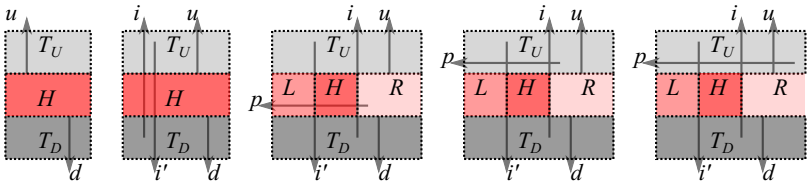


Fig. 18: The cases when F has no overlapping pairs and when $F \setminus \{i, i'\}$ has no overlapping pairs.

(2) F contains an overlapping pair (i, i') such that $F \setminus \{i, i'\}$ has no overlapping pairs. Consider the longest upward ray u and longest downward ray d in $F \setminus \{i, i'\}$. Once again this partitions the plane into three regions T_U, T_D, and H based on u and d (see Fig. 18). However, i and i' may intersect all three regions. Notice that a path from i to i' in F will be strictly contained in one of these regions. This provides three cases which cover all possibilities. Namely, when i and i' are disconnected in F restricted to: T_U and H, T_U and T_D, and H and T_D. In our dynamic programming algorithm this can easily be simulated by adding a "dummy" horizontal ray p intersecting both i and i', but disjoint from the two

regions in which they are meant to be disconnected[3]. The different cases for this dummy ray are also depicted in the figure.

(3) F contains two disjoint overlapping pairs (i, i'), (j, j') such that $F \setminus \{i, i', j, j'\}$ has no overlapping pairs. Consider the longest upward ray u and longest downward ray d in $F \setminus \{i, i'\}$ and again partition the plane into three regions T_U, T_D, and H based on u and d (see Fig. 19). As in (2), i,i',j, and j' can occur in all three regions. Thus, we again need to specialize our subproblems depending on whether there is path connecting a pair $a, b \in \{i, i', j, j'\}$ in either T_U or T_D. Notice that any such path may be simulated by using at most three rays. For example, in the second, third, and forth drawings of the figure we depict the different cases of recursing on T_U when there is a path from i to j' in T_D (the black rays in T_D are used to simulate the possible paths connecting i and j'). Ultimately, this provides a constant number of subproblems.

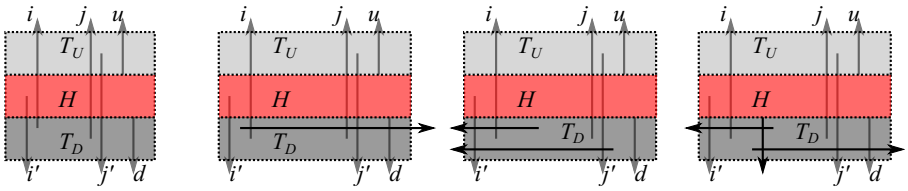


Fig. 19: Figure supporting (3).

(4) F contains three disjoint overlapping pairs (i, i'), (j, j'), (k, k') where (j, j') is the middle overlapping pair. As before, we take these pairs to be close overlapping pairs.

Notice that every neighbour of either j (or j') must be a neighbour of at least one of $\{i, i', k, k'\}$. In particular, since F contains $\{i, i', j, j', k, k'\}$, it can contain at most 5 elements from $N(j) \cup N(j')$. Two examples when no additional neighbours can be added to j or j' without creating a cycle are given in Fig. 20). The basic idea here is that with 5 neighbours the set of three overlapping pairs becomes connected and, as such, any further neighbour of a ray in the middle pair would create a cycle.

Thus, together with the three overlapping pairs F can contain up to 5 neighbours of the middle pair. In particular, since no other rays can intersect the middle overlapping pair, the remainder becomes two 3DORG subproblems T_L and T_R (as depicted in the figure). However, as in (2), these subproblems are not yet guaranteed to be independent. We must further indicate whether each pair of neighbours of j and j' are connected in T_R or T_L. This follows similarly to our

[3] We do not include the ray p when considering the region in which i and i' are not required to be disconnected.

approach in (3) by choosing up to three dummy rays for the pair of neighbours which are meant to be connected.

So, in the algorithm, we simply try all such combinations of up to 11 rays together with where the neighbours connect to each other and recursively compute the maximum.

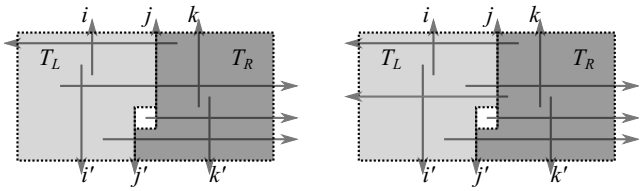


Fig. 20: Two cases of three overlapping pairs where the middle pair has 4 and 5 neighbours and any additional neighbour of j or j' would form a cycle.

Correctness and Complexity This completes the proof of Theorem 8.

References

1. Tomas Feder, Pavol Hell, and Jing Huang. List homomorphisms and circular arc graphs. *Combinatorica*, 19(4):487–505, 1999. 2
2. Stefan Felsner, Rudolf Müller, and Lorenz Wernisch. Trapezoid graphs and generalizations, geometry and algorithms. *Discrete Appl. Math.*, 74(1):13–32, 1997. 6
3. Richard M. Karp. Reducibility among combinatorial problems. In Raymond E. Miller, James W. Thatcher, and Jean D. Bohlinger, editors, *Complexity of Computer Computations*, IBM Res. Symp. Series, pages 85–103. Springer, 1972. 16
4. Ton Kloks, Ching-Hao Liu, and Sheung-Hung Poon. Feedback vertex set on chordal bipartite graphs. http://arxiv.org/abs/1104.3915, 2012. 16
5. A.V. Kostochka and J. Nešetřil. Coloring relatives of intervals on the plane, I: Chromatic number versus girth. *Europ. J. Combin.*, 19(1):103–110, 1998. 1, 2
6. Wenjing Rao, Alex Orailoglu, and Ramesh Karri. Logic mapping in crossbar-based nanoarchitectures. *IEEE Des. Test*, 26(1):68–77, 2009. 1
7. Anish Man Singh Shrestha, Satoshi Tayu, and Shuichi Ueno. Orthogonal ray graphs and nano-pla design. In *IEEE Int. Symp. Circuits Syst. (ISCAS'09)*, pages 2930–2933, 2009. 1
8. Anish Man Singh Shrestha, Satoshi Tayu, and Shuichi Ueno. On orthogonal ray graphs. *Discrete Appl. Math.*, 158(15):1650–1659, 2010. 2, 4, 6
9. M. B. Tahoori. A mapping algorithm for defect-tolerance of reconfigurable nano-architectures. In *Proc. IEEE/ACM Int. Conf. Computer-Aided Design (CAD'05)*, pages 668–672, 2005. 1