Computational Boundaries for Escaping Rectangles

Akanksha Agrawal
Indian Institute of Technology Madras, India

Pradeesha Ashok
International Institute of Information Technology Bangalore, India

Matthias Bentert
Technische Universität Berlin, Germany

Satyabrata Jana
Indian Institute of Science Education and Research Berhampur, India

Abhishek Sahu
National Institute of Science Education and Research, An OCC of HBNI, Mumbai, India

Saket Saurabh
The Institute of Mathematical Sciences, Chennai, India
University of Bergen, Norway

Kushal Singanporia
The Institute of Mathematical Sciences, Chennai, India

Abstract

Ma and Wong [IEEE TCAD '12] introduced and studied the Rectangle Escape problem, motivated by bus escape routing in printed circuit board design. In this problem, we are given an axis-parallel rectangle $\mathcal{R}$, a set $\mathcal{S}$ of axis-parallel rectangles fully contained in $\mathcal{R}$, and an integer d. The goal is to determine whether each rectangle in $\mathcal{S}$ can be extended in one of the four axis-parallel directions (up, down, left, or right) to the boundary of $\mathcal{R}$ such that no point is covered by more than d extended rectangles. We revisit Rectangle Escape and resolve several open complexity questions.

Ahmadinejad et al. [TCS '17] studied Rectangle Escape and its variants where rectangles are only allowed to be extended in a subset of directions – most notably, in two directions, a variant they termed Bidirectional REP. They showed that the problem is NP-complete when extensions are limited to two adjacent directions and $d = 3$, but left open the complexity of the case when $d = 2$. Additionally, the case for two opposite directions remained unresolved for any $d \geq 2$. We resolve the first question by showing that Bidirectional REP is NP-complete even when extensions are restricted to two adjacent directions and $d = 2$. We also settle the complexity of Rectangle Escape with two opposite directions by proving that the problem is NP-complete when d is part of the input but solvable in $\mathcal{O}(n \log n)$ time for any constant d. Finally, we consider the special case where all extended rectangles must be disjoint, that is, $d = 1$. We show an unconditional lower bound of $\Omega(n \log n)$ with a matching upper bound of $\mathcal{O}(n \log n)$ for all variants. This improves upon a sequence of algorithms for the setting with all four directions allowed and $d = 1$, starting with an $\mathcal{O}(n^6)$-time algorithm, later improved to $\mathcal{O}(n^4)$, and then to $O(n^3)$.

2012 ACM Subject Classification Theory of computation → Computational geometry; Theory of computation → Problems, reductions and completeness

Keywords and phrases NP-hardness, Sweep-line algorithm, Fixed-parameter tractability, Tight lower and upper bounds

Digital Object Identifier 10.4230/LIPIcs.ESA.2026.49

Funding *Akanksha Agrawal*: Supported by Anusandhan National Research Foundation (ANRF), MATRICS (Grant No. ANRF/ARGM/2025/003218/MTR).
Matthias Bentert: Supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 819416) and the German Research Foundation (DFG) under research grant SPP 2378: ReNO-2 (project number 511099228).
Abhishek Sahu: Supported by Anusandhan National Research Foundation (ANRF), MATRICS (Grant No. ANRF/ARGM/2025/001585/MTR).

1 Introduction

Ma and Wong [7] introduced and studied the Rectangle Escape problem, motivated by bus escape routing in printed circuit board (PCB) design. In this problem, we are given an axis-parallel rectangle $\mathcal{R}$, a set $\mathcal{S}$ of axis-parallel rectangles fully contained in $\mathcal{R}$, and an integer d. The goal is to determine whether each rectangle in $\mathcal{S}$ can be extended in one of the four axis-parallel directions (up, down, left, or right) to the boundary of $\mathcal{R}$ such that no point is covered by more than d extended rectangles. The optimization variant of the problem – minimizing the *maximum density*, that is, the maximum number of extended rectangles covering any single point directly contributes to reducing manufacturing costs in PCB design. We refer the interested reader to the work of Ma and Wong [7] for a detailed discussion of these applications.

This article is motivated by the computational questions explored by Ma and Wong [7]. They showed that the problem is NP-complete and presented a factor-4 approximation algorithm for Rectangle Escape. This was followed by a series of results in both approximation algorithms and parameterized complexity. Indeed, the problem has been the subject of extensive algorithmic study [1, 8, 9, 13, 14, 15, 16]. The problem is formally defined below. See Figure 1 for an illustration.

Rectangle Escape

Input: An axis-parallel rectangle $\mathcal{R}$, a set $\mathcal{S}$ of n axis-parallel rectangles fully contained in $\mathcal{R}$, and an integer d.
Question: Can each rectangle in $\mathcal{S}$ be extended in one axis-parallel direction to the boundary of $\mathcal{R}$ such that no point is covered by more than d extended rectangles?

We revisit Rectangle Escape and resolve several open questions concerning its complexity.

Related Work and Our Results. The starting point of our work is the work by Ahmadinejad, Assadi, Emamjomeh-Zadeh, Yazdanbod, and Zarrabi-Zadeh [1], who studied Rectangle Escape and its variants where rectangles are allowed to be extended only in a subset of directions. For Rectangle Escape, allowing only a single escape direction results in a trivial case: all rectangles must be extended in that direction, and the maximum density can be computed in $\mathcal{O}(n \log n)$ time. Thus, interesting cases arise when two, three, or four directions are allowed. For two directions, it matters whether they are opposite (e.g., up and down) or adjacent (e.g., down and left). We refer to these two-direction variants as Rectangle Escape($\updownarrow$) and Rectangle Escape($\leftarrow\downarrow$), respectively, and to the three-direction variant as Rectangle Escape($\updownarrow\rightarrow$). A particularly relevant special case is when $d = 1$, meaning all extended rectangles must be disjoint. We refer to this case as Disjoint Rectangle Escape. We study different computational questions depending on the density parameter d (whether it is a fixed constant such as 2 or 3, or part of the input), and on the subset of directions in which rectangles are allowed to be extended. For Disjoint Rectangle Escape($\updownarrow$), Yan, Chung and Chen [14, 15] presented an $\mathcal{O}(n \log n)$-time algorithm. In particular, their algorithm computes the set of all rectangles that can be extended in a particular direction.

Ahmadinejad, Assadi, Emamjomeh-Zadeh, Yazdanbod, and Zarrabi-Zadeh [1] extended the work of Ma and Wong [7] and showed that approximating Rectangle Escape to any factor smaller than $\frac{3}{2}$ is also NP-hard. Moreover, they showed that Rectangle Escape is NP-complete for $d = 2$. Combined with the fact that the problem is solvable in polynomial time for $d = 1$ [2, 5], this result fully settles the computational complexity of the problem for all values of d. They also showed that Rectangle Escape($\leftarrow\downarrow$) is NP-complete when $d = 3$.

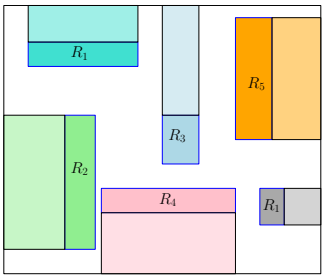

Figure 1 An instance of Rectangle Escape with $d = 1$. The input rectangles are shown in darker colors and a solution (an extension for each rectangle) is shown in lighter colors.

These findings naturally lead to the following questions, which were explicitly posed as open problems [1].

1. What is the computational complexity of Rectangle Escape($\leftrightarrows$) when $d = 2$?
2. What is the computational complexity of Rectangle Escape($\updownarrow$) for any $d \geq 2$? In particular, is the problem NP-hard when d is part of the input or even for fixed values of d?

We resolve both of these questions, but the answers turn out to be strikingly different. Our results are summarized in Table 1. Our first result is a hardness result that complements previous work [1].

▶ **Theorem 1.** Rectangle Escape($\leftrightarrows$) *is* NP*-hard even if* $d = 2$.

For the second question and to our surprise, unlike other variants of Rectangle Escape, the problem turns out to be solvable in polynomial $\mathcal{O}(n \log n)$ time for any fixed d.

▶ **Theorem 2** ($\star$)**.** Rectangle Escape($\updownarrow$) *can be solved in* $\mathcal{O}(4^d dn + n \log n)$ *time.*

However, if d is part of the input, then the problem becomes NP-hard again.

▶ **Theorem 3.** Rectangle Escape ($\updownarrow$) *is* NP*-hard.*

Note that Rectangle Escape with all four allowed directions can be solved naïvely in $\mathcal{O}(4^n n)$ time by exhaustively trying all possible direction choices for each rectangle. We significantly improve this algorithm by incorporating ideas from the proof of Theorem 2.

Finally, we study the case where $d = 1$. As noted earlier, the case $d = 1$ corresponds to the Disjoint Rectangle Escape problem and we show the following.

▶ **Theorem 4.** Disjoint Rectangle Escape *can be solved in* $\mathcal{O}(n \log n)$ *time.*

A simple reduction then establishes that Disjoint Rectangle Escape is polynomial-time solvable for all variants where the allowed extension directions form a subset of the four cardinal directions. We mention that Kong, Ma, Yan, and Wong [6] showed that Disjoint Rectangle Escape can be solved in $\mathcal{O}(n^6)$ time, which was later improved to $\mathcal{O}(n^4)$ time [1]. Moreover, Keil, Mitchell, Pradhan, and Vatshelle [4] showed that Maximum Weight Independent Set can be solved in $\mathcal{O}(n^3)$ time on outersting graphs. By replacing each rectangle with the outline of the four possible extended rectangles (each with weight 1) and then searching for an independent set of weight n, their algorithm finds a solution for Disjoint Rectangle Escape in $\mathcal{O}(n^3)$ time if it exists as for each rectangle, at most one of

the four extensions can be picked into the independent set as any two extended forms intersect in the original rectangle. To the best of our knowledge, this is the fastest algorithm even for the special case Disjoint Rectangle Escape($\leftarrow\downarrow$) and we improve this to $\mathcal{O}(n \log n)$. We also show a matching unconditional lower bound for Disjoint Rectangle Escape for any subset of allowed directions via a simple reduction from Segment Intersection.

To conclude this overview, we mention that Roy, Govindarajan, Misra, and Shetty [11] introduced the problem Runaway Rectangle Escape[1] and analyzed it from the perspective of parameterized complexity. In Runaway Rectangle Escape, we are given an additional integer k as part of the input, and the goal is to extend at least k rectangles such that the maximum density remains at most d, as opposed to extending all rectangles. Several algorithms for different variants of the problem have been implemented and evaluated in practice [6, 8, 9, 13, 14, 16].

Our Methods. Theorem 1 is based on a reduction from Monotone 2P1N-3-Sat (for which we observe NP-hardness via a known reduction for a slightly different problem). In the reduction, we represent each variable by two rectangles. Extending at least one of the two rectangles to the bottom will correspond to setting the variable to true and extending both to the left will correspond to setting the variable to false. We then add gadgets for each clause (which differ for positive and negative clauses) to ensure that all rectangles can be extended to the bottom or the left if and only if the input formula is satisfiable.

The proof of Theorem 2 combines a sweep-line approach with dynamic programming.

The proof of Theorem 3 is based on a reduction from Vertex Cover. The reduced instance contains very long rectangles in three rows and we ensure that all these rectangles in the upper two rows are extended to the top and all rectangles in the lower row are extended to the bottom. We then add a sequence of rectangles encoding each vertex in the input graph. The rectangles alternate between being located between the top two rows and between the bottom two rows. Moreover, we ensure that all rectangles corresponding to the same vertex are extended in the same direction (up or down). Extending all rectangles to the top corresponds to including the vertex in a solution. We then add a gadget for each edge in the input graph where only rectangles corresponding to the two endpoints are between the top two rows of long rectangles. If both are extended to the bottom, then k other rectangles will be extended to the top and together with the rectangles in the middle column, this will violate the density constraint in the middle column.

The proof of Theorem 4 is divided into two parts. First, we show that certain generalizations of Disjoint Rectangle Escape($\leftarrow\downarrow$) and Disjoint Rectangle Escape($\updownarrow\rightarrow$) can be solved in $\mathcal{O}(n \log n)$ time. Second, we use these generalizations to consider two possible cases. In the first case, we show that solutions for two instances of Disjoint Rectangle Escape($\updownarrow\rightarrow$) can be merged into a valid solution of the input instance in a straight-forward way. In the other (more difficult) case, we show that there are four rectangles with a very restricted structure in any possible solution. We then show that this restricted structure can be used to construct two solution candidates in $\mathcal{O}(n \log n)$ time such that if a solution exists, then at least one of the candidates is also a valid solution. We then simply verify for each candidate whether it is a solution and either return a valid solution or correctly determine that none exists.

[1] We note that the same problem has also appeared in the literature under the name Max-Route, and is sometimes referred to simply as Rectangle Escape. We adopt the name Runaway Rectangle Escape to distinguish it from the variant in which all rectangles must be extended.

Table 1 Overview of our results. A – means that the result follows from other entries in the table.

Directions	disjoint	$d = 2$	unbounded d
All	$\Theta(n \log n)$ (Thm. 4)	NP-hard [1]	$\mathcal{O}(3^n 4^d dn)$ (Cor. 10)
Three	–	–	$\mathcal{O}(2^n 4^d dn)$ (Cor. 10)
↰↳	–	NP-hard (Thm. 1)	–
↕	–	$\Theta(n \log n)$ (Thm. 2)	NP-hard (Thm. 3)
One	$\Theta(n \log n)$ (Prop. 6)	–	$\Theta(n \log n)$ ([15])

Notation. For a positive integer x, let $[x]$ be the set of integers $\{1, \ldots, x\}$. A *half-open* interval $[a, b)$ for two numbers $a \leq b$ is the set $\{x \in \mathbb{R} \mid a < x < b\} \cup \{a\}$[2]. A rectangle R is the Cartesian product of two half-open intervals $[x_1, x_2)$ and $[y_1, y_2)$. We sometimes also describe R by two of its corner points, e.g. (x_1, y_1) and (x_2, y_2). The x-coordinates of the left and right border of R are x_1 and x_2, respectively, and the y-coordinates of the bottom and top border of R are y_1 and y_2, respectively. Given a set $\mathcal{S}$ of rectangles, the maximum density of $\mathcal{S}$ is the maximum number of rectangles that contain a common point (x, y). Note that the rectangles R_1 with corners $(0, 0)$ and $(1, 1)$ and R_2 with corners $(1, 0)$ and $(2, 1)$ do not share a common point as e.g. the point $(1, 0.5)$ is only contained in R_2. As is custom in computational geometry, we use the real-RAM model of computation which e.g. allows to compare arbitrary real numbers in constant time.

2 Two Adjacent Directions

In this section, we show that Rectangle Escape(↰↳) is NP-hard even for $d = 2$. In order to do so, we first show that Monotone 2P1N-3-SAT – a variant of Satisfiability where all clauses are monotone, that is, they contain either only positive or only negated literals, each clause contains two or three literals, and each variable appears exactly twice positive and once negated in the input formula – is NP-hard.[3] This result closely follows the reduction showing that Monotone 3-SAT is NP-hard [3]. We were not able to find this observation anywhere else, so we include it in for the sake of completeness.

▶ **Proposition 5.** Monotone 2P1N-3-SAT *is* NP-hard.

Proof. We reduce from a variant of 3-SAT, where each clause contains exactly three literals and each variable appears exactly 2 times positive and 2 times negated. This variant is known to be NP-hard [3]. For the reduction, let ϕ be an input formula with clauses $\mathcal{C}$ over variables V where each clause contains three literals, all clauses are monotone, and each variable appears twice negated and twice positive. We start by introducing four variables $x_1, x_2, \bar{x}_1, \bar{x}_2$ for each variable $x \in V$.[4] To construct ϕ', we start with ϕ and replace the first positive occurrence of x by x_1, the second positive occurrence of x by x_2, the first negative occurrence of x by $\bar{x}_1$, and the second negative occurrence of x by $\bar{x}_2$. Note that so far, each variable appears exactly once positive and zero times negated. Hence, all clauses are trivially monotone. We next ensure that $x_1 = x_2 \neq \bar{x}_1 = \bar{x}_2$ in any satisfying assignment of the constructed formula ϕ'. Note

[2] Note that the definition is equivalent to $\{x \in \mathbb{R} \mid a \leq x < b\}$ whenever $a < b$ but ensures that $a \in [a, a)$.
[3] We mention in passing that if additionally each clause contains exactly three literals, then these formulas are always satisfiable by a simple matching argument [12].
[4] Note that $\bar{x}_1$ and $\bar{x}_2$ are variables and not negations of x_1 and x_2, respectively. The intuition is that we will later guarantee that x_1 and $\bar{x}_1$ cannot be set to the same truth value and setting x_1 to true corresponds to setting x to true and setting $\bar{x}_1$ to true corresponds to setting x to false.

that once this is ensured, it is easy to verify that ϕ is satisfiable if and only if ϕ' is satisfiable. To ensure the above property, we add the four clauses $(x_1 \vee \bar{x}_1), (\neg\bar{x}_1 \vee \neg x_2), (x_2 \vee \bar{x}_2)$, and $(\neg\bar{x}_2 \vee \neg x_1)$ to ϕ'. Since all clauses are monotone and each variable appears exactly once positive and once negated in one of the newly added clauses, it holds that ϕ' is monotone and each variable appears exactly once negated and exactly twice positive in ϕ'. It remains to show the claimed property.

To this end, note that setting x_1 and x_2 to true and $\bar{x}_1$ and $\bar{x}_2$ to false satisfies all newly added clauses. Moreover, setting x_1 and x_2 to false and $\bar{x}_1$ and $\bar{x}_2$ to true also satisfies all these clauses. It remains to show that no other assignment satisfies all of these clauses. We make a case distinction of whether x_1 is set to true or false and show in both cases that the above assignments are the only ones satisfying the four added clauses for x. So first assume that x_1 is set to true. To satisfy the clause $(\neg\bar{x}_2 \vee \neg x_1)$, we need to set $\bar{x}_2$ to false. Then, to satisfy the clause $(x_2 \vee \bar{x}_2)$, we need to set x_2 to true. Finally, to satisfy the clause $(\neg\bar{x}_1 \vee \neg x_2)$, we have to set $\bar{x}_1$ to false. If x_1 is set to false, then in order to satisfy the clause $(x_1 \vee \bar{x}_1)$, we need to set $\bar{x}_1$ to true. Then, to satisfy the clause $(\neg\bar{x}_1 \vee \neg x_2)$, we need to set x_2 to false. Finally, to satisfy the clause $(x_2 \vee \bar{x}_2)$, we have to set $\bar{x}_2$ to true. This concludes the proof. ◀

With Proposition 5 at hand, we are in a position to prove Theorem 1, that is, Rectangle Escape($\leftrightarrows$) with $d = 2$ is NP-hard. We present the reduction from Monotone 2P1N-3-SAT and give a high-level overview of the proof, but defer some details of the proof to the appendix.

▶ **Theorem 1.** Rectangle Escape($\leftrightarrows$) *is* NP*-hard even if* $d = 2$.

Proof. We reduce from Monotone 2P1N-3-SAT. To this end, let ϕ be an input formula with variables V and clauses $\mathcal{C} = \mathcal{C}_p \cup \mathcal{C}_n$, where $\mathcal{C}_p$ contains all positive clauses and $\mathcal{C}_n$ contains all negative clauses. Let s_p and s_n be the number of positive and negative clauses. We fix an arbitrary order $\prec$ on the set V of variables. Next, we sort all clauses in $\mathcal{C}_p$ lexicographically with respect to $\prec$. Let C_i^p be the i^{th} positive clause in this ordering for each $i \in [s_p]$. We do the same for $\mathcal{C}_n$ and denote the i^{th} negative clause by C_i^n for each $i \in [s_n]$.

To construct an equivalent instance of Rectangle Escape($\leftrightarrows$), we start with a boundary rectangle $\mathcal{R}$ of height $4|V| + 6s_n + 1$ and width $5|V| + 2s_p + 1$. We divide the rectangle into a grid with $4|V| + 6s_n + 1$ rows of height 1 and $5|V| + 2s_p + 1$ columns of width 1 and set $d = 2$. We refer to Figure 2 for an illustration and an example of the following construction.

We say the first row/column is the *border* row/column. The next $4|V|$ rows and $5|V|$ columns are the *variable* rows/columns. We say that rows $4i - 2$ to $4i + 1$ and columns $5i - 3$ to $5i + 1$ correspond to the i^{th} variable in V (with respect to $\prec$). The remaining $6s_n$ rows and $2s_p$ columns are the *clause* rows/columns where (counted from the bottom/left) the rows $6i - 5$ to $6i$ are the rows corresponding to C_i^n and the columns $2i - 1$ and $2i$ correspond to clause C_i^p. We first place *border rectangles* in the border column and the border row as follows. We start with a rectangle in the border column that covers all variable rows and a rectangle in the border row that covers all variable columns. After that, for each negative clause C_i^n, we add a rectangle in the border column that covers the highest two rows corresponding to C_i^n. For each positive clause C_i^p, we add a rectangle in the border row that covers the second column corresponding to C_i^p.

Next, we add two *variable rectangles* x_1 and x_2 for each variable $x \in V$. Each of these rectangles covers the intersection of two variable columns and one variable row as follows. Let i be the index such that x is the i^{th} variable in V in the order given by $\prec$. Then, x_1 is in the $4i - 1^{\text{st}}$ row and in the $5i - 2^{\text{nd}}$ and $5i - 1^{\text{st}}$ column. The rectangle x_2 is in the $4i + 1^{\text{st}}$ row and in the $5i^{\text{th}}$ and $5i + 1^{\text{th}}$ column.

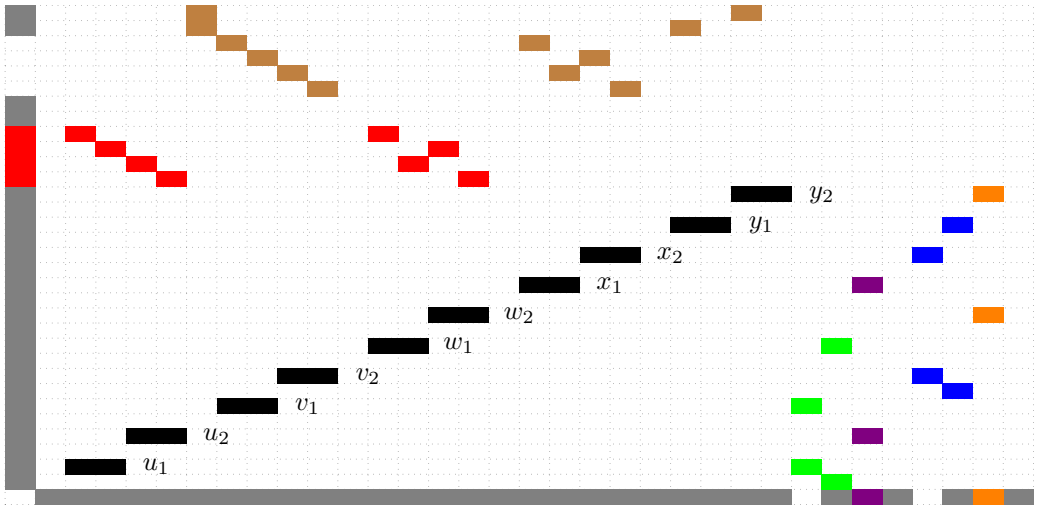

Figure 2 Example construction for the input formula

$\phi = (\neg u \vee \neg w) \wedge (\neg v \vee \neg x \vee \neg y) \wedge (u \vee v \vee w) \wedge (u \vee x) \wedge (v \vee x \vee y) \wedge (w \vee y)$.

The order $\prec$ is the alphabetic order, the border rectangles are gray, the variable rectangles are black, and the clause rectangles are color coded corresponding to the clause they represent.

Now, we add *clause rectangles* for each clause. The construction is different for positive and negative clauses and also depends on whether the clause contains two or three literals. We refer to Figure 2 for the different cases and provide a formal description in the appendix. The red rectangles correspond to a negative clause with two literals, the brown rectangles to a negative clause with three literals, the green and blue each correspond to a positive clause with three literals, and the purple rectangles correspond to a positive clause with two literals.

Since the reduction can clearly be computed in polynomial time in the length of ϕ, it only remains to show correctness, that is, ϕ is satisfiable if and only if the constructed instance of Rectangle Escape($\leftrightarrows$) has a solution with maximum density 2.

We defer the formal proof of correctness for space reasons to the appendix. We still give a high-level intuitive explanation here. We say that a solution to the constructed instance (if one exists) corresponds to a satisfying assignment by setting each variable x to true if and only if at least one of the two rectangles x_1 and x_2 are extended to the bottom. Note that since each variable appears only once negated, each clause rectangle (excluding the helper rectangles) for a negative clause is in a variable column containing exactly three rectangles (the clause rectangle itself, a variable rectangle, and a border rectangle). Hence, not both the clause rectangle and the respective variable rectangle can be extended to the bottom as this would create density three in the border row. Similarly, every non-helper clause rectangle for a positive clause is contained in a variable row with exactly three rectangles (itself, the respective variable rectangle, and a border rectangle) meaning that not both the variable rectangle and the clause rectangle can be extended to the left. We show that for each negative clause, there needs to be one variable such that all clause rectangles in columns corresponding to that variable have to be extended to the bottom. Note that this means that both variable rectangles for this variable have to be extended to the left, corresponding to setting this variable to false and satisfying the clause. Similarly, for each positive clause, at least one of the clause rectangles has to be extended to the left, meaning that at least one of the involved variable rectangles has to be extended to the bottom and this corresponds to setting the respective variable to true and satisfying the clause. ◀

3 Disjoint Rectangle Escape

In this section, we study Disjoint Rectangle Escape and show an $\mathcal{O}(n \log n)$ time algorithm for any set of allowed directions. This improves upon the best known algorithm for Disjoint Rectangle Escape by Keil, Mitchell, Yan, Wong [4] running in $\mathcal{O}(n^3)$ time (and previous algorithms running in $\mathcal{O}(n^6)$ time [5] and $\mathcal{O}(n^4)$ time [2], respectively). We also show that this bound is tight as Disjoint Rectangle Escape cannot be solved in $o(n \log n)$ time. This lower bound is unconditional, even holds for computing the maximum density of the input rectangles without extending any rectangle, and follows from a simple reduction from Segment Intersection. Therein, one is given a set of half-open intervals and the task is to decide whether at least two of them intersect. It is known that Segment Intersection has an unconditional $\Omega(n \log n)$ lower bound in the real-RAM model [10, Theorem 7.9].

▶ **Theorem 6.** Disjoint Rectangle Escape *cannot be solved in $o(n \log n)$ time. This lower bound even applies to checking whether the set of input rectangles are disjoint.*

Proof. We show that if Disjoint Rectangle Escape can be solved in $f(n) \in o(n \log n)$ time, then Segment Intersection can be solved in $\mathcal{O}(f(n) + n) \in o(n \log n)$ time, a contradiction to the fact that Segment Intersection cannot be solved in $o(n \log n)$ time [10]. To this end, let a set of half-open intervals $[a_i, b_i)$ be given as input. We can compute the smallest input number $a_{\min}$ and the largest input number $b_{\max}$ in linear time. Then, we construct an equivalent instance of Disjoint Rectangle Escape as follows. We start with a boundary rectangle $\mathcal{R}$ with corner points $(a_{\min}, 0)$ and $(b_{\max}, 1)$. For each interval $[a_i, b_i)$, we create the rectangle with corner points $(a_i, 0)$ and $(b_i, 1)$. Note that two intervals $[a_i, b_i)$ and $[a_j, b_j)$ intersect if and only if their two respective rectangles intersect. Moreover, the number of input rectangles is equal to the number of input intervals for Segment Intersection. Hence, if we could check whether two input rectangles intersect in $o(n \log n)$ time, then we could solve Segment Intersection in $o(n \log n)$ time. This concludes the proof. ◀

For the positive result, we first consider generalizations of the problem for two and three directions that we will subsequently use in the algorithm for all four directions. We start with an algorithm for Disjoint Rectangle Escape($\leftarrow\!\downarrow$) where we try to extend a maximum number of rectangles to the left. We will show that there is a unique solution to this problem which can be found in $\mathcal{O}(n \log n)$ time and which has an additional desirable property. Due to space constraints, we have to defer these proofs to a full version.

▶ **Lemma 7.** Disjoint Rectangle Escape($\leftarrow\!\downarrow$) *can be solved in $\mathcal{O}(n \log n)$ time and in the same time, we can compute a solution $S_{\leftarrow}$ with the following property. For each rectangle R, if there is any solution in which R is extended to the left, then R is also extended to the left by $S_{\leftarrow}$.*

We next turn to three directions. For Disjoint Rectangle Escape($\updownarrow\!\rightarrow$), we solve a generalization we call Partial Disjoint Rectangle Escape($\updownarrow\!\rightarrow$). Therein, the task is to find the minimal x-coordinate $x_{\min}$ and a direction (down, right, or up) for each rectangle R_j whose right boundary is larger than $x_{\min}$ such that no two rectangles overlap after the extension (and rectangles whose right boundary is at most $x_{\min}$ are not extended but still remain in the instance). Note that Partial Disjoint Rectangle Escape($\updownarrow\!\rightarrow$) is an optimization problem rather than a decision problem as if the instance $\mathcal{I}$ of Disjoint Rectangle Escape($\updownarrow\!\rightarrow$) for the same set of rectangles is a no-instance, then Partial Disjoint Rectangle Escape($\updownarrow\!\rightarrow$) still has to return a partial solution rather than just "no".

Moreover, observe the following two key facts. First, if there is a solution for $\mathcal{I}$, then a solution for Partial Disjoint Rectangle Escape($\updownarrow\!\!\rightarrow$) returns the x-coordinate of the left boundary of the bounding rectangle $\mathcal{R}$ and a solution for Disjoint Rectangle Escape($\updownarrow\!\!\rightarrow$). Second, if there is no solution for $\mathcal{I}$, then the value $x_{\min}$ is the right boundary of some rectangle R_i as otherwise, we can reduce $x_{\min}$ to the next smallest right boundary of a rectangle without changing the requirement for any rectangle. We will later use Partial Disjoint Rectangle Escape($\updownarrow\!\!\rightarrow$) to solve Disjoint Rectangle Escape in $\mathcal{O}(n \log n)$ time. To this end, we first show that Partial Disjoint Rectangle Escape($\updownarrow\!\!\rightarrow$) can be solved in that time.

▶ **Lemma 8.** Partial Disjoint Rectangle Escape($\updownarrow\!\!\rightarrow$) *is solvable in* $\mathcal{O}(n \log n)$ *time.*

With Lemmas 7 and 8 at hand, we can now present the algorithm for Disjoint Rectangle Escape. Before we present the algorithm formally, we first give a high-level description. The idea is to solve four instances of Partial Disjoint Rectangle Escape($\updownarrow\!\!\rightarrow$) for the same set of rectangles and with a different set of three allowed directions each. Let $x_{\min}$ and $x_{\max}$ be the x-values in the solutions of instances of Partial Disjoint Rectangle Escape($\updownarrow\!\!\rightarrow$) where rectangles can be extended to the top, bottom, and right and to the top, bottom, and left, respectively. Note that the definition of which rectangles have to be extended is only almost symmetric, that is, for the latter instance, we consider all rectangles whose left boundary has an x-value of at most $x_{\max}$ (rather than strictly smaller). We will solve two cases separately. If $x_{\min} \leq x_{\max}$, then we will show that a solution always exists and can be found by combining the two solutions for Partial Disjoint Rectangle Escape($\updownarrow\!\!\rightarrow$). The same holds if $y_{\min} \leq y_{\max}$, where these values are corresponding to the instances where the allowed directions are left, top, and right and left, bottom, and right, respectively. So assume in the following that $x_{\min} > x_{\max}$ and $y_{\min} > y_{\max}$. If there is a solution for Disjoint Rectangle Escape, then some rectangle L with a right boundary at least $x_{\min}$ has to be extended to the left. Hence, this extended rectangle will "cross" the entire distance from the vertical line at position $x_{\min}$ to the left boundary of $\mathcal{R}$. Similarly, some rectangle R in the solution extends all the way from $x_{\max}$ to the right boundary of $\mathcal{R}$, some rectangle T extends from $y_{\max}$ to the top boundary of $\mathcal{R}$, and some rectangle B extends all the way from $y_{\min}$ to the bottom boundary of $\mathcal{R}$ in the solution. We will show that even if we do not know these rectangles, their existence can be used to partition the instance into four subinstances with fairly restrictive properties. These properties allow us to find a solution in $\mathcal{O}(n \log n)$ time.

▶ **Theorem 4.** Disjoint Rectangle Escape *can be solved in* $\mathcal{O}(n \log n)$ *time.*

Proof. Throughout this proof, we will denote the rectangles in the input by R_i. For each i, let x_i^ℓ and x_i^r be the x-coordinates of the left and right boundary of R_i, respectively. Similarly, let y_i^b and y_i^t the the y-values of the bottom and top border of R_i, respectively. We first solve the four instances of Partial Disjoint Rectangle Escape($\updownarrow\!\!\rightarrow$) with the same set of rectangles and where each time a different set of three directions is permitted. This defines the following four values.

- $x_{\min}$: minimum x-coordinate such that all rectangles R_i with $x_i^r > x_{\min}$ can be extended to the top, right, or bottom.
- $x_{\max}$: maximum x-coordinate such that all rectangles R_i with $x_i^\ell \leq x_{\max}$ can be extended to the bottom, left, or top.
- $y_{\min}$: minimum y-coordinate such that all rectangles R_i with $y_i^b > y_{\min}$ can be extended to the left, top, or right.
- $y_{\max}$: maximum y-coordinate such that all rectangles R_i with $y_i^t \leq y_{\max}$ can be extended to the right, bottom, or left.

We next consider three cases based on whether $x_{\min} \leq x_{\max}$ and/or $y_{\min} \leq y_{\max}$.

Case 1 ($x_{\min} \leq x_{\max}$). By definition, there are two (partial) solutions S_1 and S_2 such that S_1 extends all rectangles R_i with $x_i^r > x_{\min}$ and S_2 extends all rectangles R_j with $x_j^\ell \leq x_{\max}$. Note that each rectangle R_j which is not extended by S_1 is extended by S_2 as $x_j^\ell \leq x_j^r \leq x_{\min} \leq x_{\max}$. We next show that extending all rectangles extended by S_1 as described by S_1 and all other rectangles as described by S_2 yields a valid solution for Disjoint Rectangle Escape. Let S be the computed solution and assume towards a contradiction that two rectangles R_1 and R_2 overlap in S. Since S_1 and S_2 are valid partial solutions, it must hold that R_1 and R_2 are not both extended as in S_1 and/or S_2. Thus, we may assume without loss of generality that R_1 is extended as described by S_1 and R_2 is extended as described by S_2. Moreover, R_1 does not intersect with the original form of R_2 and vice versa since S_1 and S_2 are valid solutions for Partial Disjoint Rectangle Escape($\updownarrow\!\!\rightarrow$). Thus, exactly one of them is extended to the top/bottom and the other is extended to the left/right. Since both cases are completely symmetric, we assume that R_1 is extended to the top or bottom and R_2 is extended to the left or right. Since R_2 is extended as described by S_2, it is extended to the left. It holds that $x_2^\ell \leq x_2^r \leq x_{\min}$ as otherwise R_2 would be extended as described by S_1. Thus, $x_2^\ell \leq x_{\min} < x_1^r$ holds, contradicting the assumption that extending R_2 to the left intersects with R_1 but not extending R_2 does not.

Case 2 ($y_{\min} \leq y_{\max}$). This case is completely symmetric to Case 1 and therefore omitted.

Case 3 ($x_{\min} > x_{\max}$ and $y_{\min} > y_{\max}$). Note that in any potential solution for Disjoint Rectangle Escape it holds that at least one rectangle R_ℓ with $x_\ell^r \geq x_{\min}$ is extended to the left. This is true as otherwise a solution extends all rectangles R_i with $x_i^r \geq x_{\min}$ to the top, right, or bottom and hence this yields a solution for Partial Disjoint Rectangle Escape($\updownarrow\!\!\rightarrow$) with a smaller value $x_{\min}$, a contradiction. Similarly, one rectangle R_r with $x_r^\ell < x_{\max}$ is extended to the right, one rectangle R_b with $y_b^t \geq y_{\min}$ is extended to the bottom, and one rectangle R_t with $y_t^b < y_{\max}$ is extended to the top. We next make a case distinction whether $x_t^r \leq x_{\min}$ or not.

- If $x_t^r > x_{\min}$, then note that in order for R_r to extend from $x_{\max} < x_{\min}$ to the right border of $\mathcal{R}$, it must hold that $y_r^t \leq y_{\max}$. Similarly, $x_b^r \leq x_{\max}, y_\ell^b \geq y_{\min}$, and $y_t^\ell \geq x_{\min}$. See Figure 3 for an illustration.
- If $x_t^r \leq x_{\min}$, then the case is analogous with the rectangles R_t and R_b swapped and R_ℓ and R_r swapped.

We guess[5] which case applies and in the following show how to find a solution for the case $x_t^r > x_{\min}$. We will call those solutions counter-clockwise oriented and otherwise they are clockwise oriented.

If any rectangle contains a point (x, y) with $x_{\max} < x < x_{\min}$ and $y_{\max} < y < y_{\min}$, then there cannot be a solution as this rectangle cannot be any rectangle in $\{R_\ell, R_r, R_t, R_b\}$ as shown above and it cannot be extended to the top (due to R_ℓ), the left (R_b), the bottom (R_r), or the right (R_t). This allows us to partition the rectangles into four sets B_1, B_2, B_3, and B_4 as follows. See Figure 4 for an example.

[5] Whenever we pretend to guess something, we iterate over all possible choices.

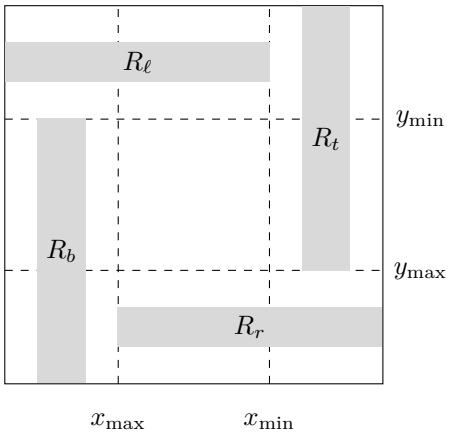

Figure 3 Illustration of the case where $x_t^r > x_{\max}$ with the rectangles R_ℓ, R_t, R_b, and R_r extended to their respective directions.

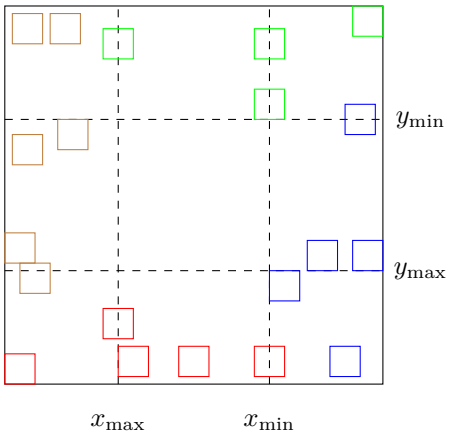

Figure 4 Example of the four sets B_1 (red), B_2 (blue), B_3 (green), and B_4 (brown).

- The set B_1 contains all rectangles R_i with $y_i^t \leq y_{\max}$ and $x_i^\ell < x_{\min}$. Note that due to R_ℓ, no rectangle in B_1 can be extended to the top in any counter-clockwise-oriented solution.
- The set B_2 contains all rectangles R_i with $x_i^\ell \geq x_{\min}$ and $y_i^b < y_{\min}$. Due to R_b, no rectangle in B_2 can be extended to the left in any counter-clockwise-oriented solution.
- The set B_3 contains all rectangles R_i with $y_i^b \geq y_{\min}$ and $x_i^r > x_{\max}$. Due to R_r, no rectangle in B_3 can be extended to the bottom in any counter-clockwise-oriented solution.
- The set B_4 contains all rectangles R_i with $x_i^r \leq x_{\max}$ and $y_i^t > y_{\max}$. Due to R_t, no rectangle in B_4 can be extended to the right in any counter-clockwise-oriented solution.

Note that B_1, B_2, B_3, B_4 indeed form a partition of all rectangles as no rectangle contains a point (x, y) with $x_{\max} < x < x_{\min}$ and $y_{\max} < y < y_{\min}$. This allows us to show in the appendix how to identify a rectangle $R_{r'}$ in B_1 that has to be extended to the right in any counter-clockwise-oriented solution. Note that this can be different from R_r if $x_{r'}^\ell > x_{\max}$. We then do the same to compute rectangles $R_{\ell'}, R_{t'}$, and $R_{b'}$. Afterwards, we show how these four rectangles can be used to construct a counter-clockwise-oriented solution for the entire instance in $\mathcal{O}(n \log n)$ time assuming that such a solution exists.

Let $C \subseteq B_1$ be the set of rectangles in B_1 that cannot be extended to the bottom without intersecting a different (unextended) rectangle. We next show that there is no counter-clockwise-oriented solution if $C = \emptyset$. Assume towards a contradiction that there is such a solution S. Then, we construct a different solution S' where all rectangles in B_2, B_3, and B_4 are extended as in S and all rectangles in B_1 are extended to the bottom. Note that S' is indeed a solution as S does not extend a rectangle in B_2 to the left due to R_ℓ and hence extending rectangles in B_1 to the bottom can only intersect with a different extended rectangle from B_1. Since all of those are extended to the bottom and by assumption do not overlap, S' is a solution. Moreover, S' contains the rectangle $R_t \in B_2$ with $x_t^\ell \geq x_{\min}$ and $y_t^b < y_{\max}$ extended to the top. This implies that S' is not clockwise oriented as shown next. Recall that each counter-clockwise-oriented solution must contain a rectangle R_r with $y_r^t \leq y_{\max}$ and $x_r^\ell < x_{\max}$ which is extended to the right. Since this rectangle is by definition contained in B_1, this contradicts that S' is a counter-clockwise-oriented solution. Thus, we may assume that $C \neq \emptyset$.

Let $R_{r'} \in C$ be the rectangle in C with maximum $x_{r'}^r$. It remains to show that $R_{r'}$ is extended to the right in any counter-clockwise-oriented solution. Assume towards a contradiction that there is a counter-clockwise-oriented solution S in which $R_{r'}$ is not extended to the right. Then, $R_{r'}$ is extended to the left in S as $R_{r'}$ is contained in B_1 and in C meaning that it cannot be extended to the top or bottom in any counter-clockwise-oriented solution. Let R_{r^*} be the rectangle in B_1 with minimum $x_{r^*}^\ell$ which is extended to the right in S. If $x_{r^*}^\ell \leq x_{r'}^r$, then R_{r^*} (extended to the right) and $R_{r'}$ (extended to the left) cover all possible x-values and therefore ensure that rectangle R_b cannot be extended to the bottom, a contradiction to S being a counter-clockwise-oriented solution. Hence, $x_{r^*}^\ell > x_{r'}^r$. Let $D \subseteq B_1$ be the set of all rectangles R_i with $x_i^\ell \leq x_{r'}^r$. Note that by the definition of $R_{r'}$, it holds that $C \subseteq D$ and that no rectangle in D is extended to the right in S. As in the case where $C = \emptyset$, we reach a contradiction by constructing a solution S' where all rectangles in B_2, B_3, B_4, and D are extended as in S and all rectangles in $B_1 \setminus D$ are extended to the bottom. Note that S' is indeed a solution as S does not extend a rectangle in B_2 to the left and since $C \subseteq D$, extending all rectangles in $B_1 \setminus D$ to the bottom can only create a point of density at least two together with a different extended rectangle from B_1. Since all of those are extended to the bottom or to the left, S' is a solution. Moreover, as all rectangles in B_2 are still extended as in S, the rectangle R_ℓ still fulfills $y_\ell^b \geq y_{\min}$ and is extended to the left. However, this means that no rectangle R_r exists as this rectangle needs to be contained in B_1 and extended to the right in any counter-clockwise-oriented solution. Hence, S' is not clockwise oriented, a contradiction. Thus, $R_{r'}$ is extended to the right in each counter-clockwise-oriented solution. The arguments to construct R_b', R_t', and R_ℓ' are analogous and therefore omitted.

We next show how the rectangles $R_{r'}, R_{\ell'}, R_{t'}$, and $R_{b'}$ can be used to find a counter-clockwise-oriented solution for the entire instance of Disjoint Rectangle Escape (if such a solution exists) and analyze the running time. Since the algorithm to find a clockwise oriented solution is symmetric, this concludes the algorithm.

Note that the rectangle $R_{r'}$ splits all rectangles in B_2 into two sets: the set B of those rectangles R_i with $y_i^t \leq y_{r'}^b$ and the set T of rectangles R_i with $y_i^b \geq y_{r'}^t$. If any rectangle in B_2 does not belong to B or T, then extending $R_{r'}$ to the right creates a point of density two and thus no counter-clockwise-oriented solution exists. Moreover in any counter-clockwise-oriented solution, no rectangle in B_2 can be extended to the left due to R_b, no rectangle in T can be extended to the bottom due to $R_{r'}$, and no rectangle in B can be extended to the top due to $R_{r'}$. Similarly, the rectangle $R_{b'}$ splits B_1 into a set L of rectangles which

cannot be extended to the right and a set R of rectangles which cannot be extended to the left. The rectangles in $R \subseteq B_1$ can only be extended to the bottom or the right. Now, we find a solution for the rectangles in $R \cup B$ using Lemma 7 in which each rectangle that can be extended to the bottom in any solution is extended to the bottom and all other rectangles are extended to the right. Extending a rectangle from $R \cup B$ to the right creates a potential conflict with a rectangle from B_4 extended to the bottom or from T extended to the top, while extending such a rectangle to the bottom only creates a potential conflict with rectangles from $R \cup B$ which are all handled by Lemma 7. Thus, if a counter-clockwise-oriented solution exists, then there also exists one in which all rectangles in $R \cup B$ are extended as computed. We do the same for the other three sets and check in the end whether these four solutions combine to a solution for the entire instance. As the four sets B_1, B_2, B_3, and B_4 partition all rectangles and each such set is partitioned into two sets, these eight subsets also form a partition of all rectangles. Since each such subset is considered in exactly one of the four calls to the algorithm behind Lemma 7, this yields a potential solution for the considered input instance of Disjoint Rectangle Escape. If it is a solution, we verify this and output it. If it is not a valid solution, then no counter-clockwise oriented solution exists and we try the symmetric algorithm for finding a clockwise oriented solution. If neither exists, then we correctly output that the input instance is a no-instance.

It remains to analyze the running time. Solving four instances of Partial Disjoint Rectangle Escape($\updownarrow\!\!\!\rightarrow$) takes $\mathcal{O}(n \log n)$ time. Determining whether $x_{\min} \leq x_{\max}$ or $y_{\min} \leq y_{\max}$ takes constant time. If either applies, we can solve the instance in $\mathcal{O}(n)$ time since we can check for each rectangle whether it belongs to S_1 or S_2 and look up the direction to extend it to in constant time. In the case where $x_{\min} > x_{\max}$ and $y_{\min} > y_{\max}$, we can check whether a clockwise-oriented solution or a counter-clockwise-oriented solution exists in $\mathcal{O}(n \log n)$ time each as shown next. Computing the sets B_1, B_2, B_3, and B_4 takes $\mathcal{O}(n)$ time. As we show in the appendix, we then compute rectangles R'_r, R'_b, R'_ℓ, and R'_t, partition each set B_i into two sets, and solve four instances of Disjoint Rectangle Escape($\leftarrow\!\!\!\downarrow$). Computing the rectangles R'_r, R'_b, R'_ℓ, and R'_t takes $\mathcal{O}(n \log n)$ time using the algorithm by Yan, Chung, and Chen [15] for finding all rectangles that can be extended to a given direction without intersecting other (unextended) rectangles and then ordering these rectangles. Partitioning the rectangles in each set B_i takes $\mathcal{O}(n)$ time and solving four instances of Disjoint Rectangle Escape($\leftarrow\!\!\!\downarrow$) takes $\mathcal{O}(n \log n)$ time using Lemma 7. Verifying whether a given extension for each rectangle yields a solution also takes $\mathcal{O}(n \log n)$ time. ◀

By placing a long rectangle blocking one of the borders of the bounding rectangle $\mathcal{R}$, we can reduce any variant of Disjoint Rectangle Escape with a limited set of allowed directions to Disjoint Rectangle Escape. Hence, all of these variants can also be solved in $\mathcal{O}(n \log n)$ time. This yields the following.

▶ **Corollary 9.** Disjoint Rectangle Escape($\leftarrow\!\!\!\downarrow$), Disjoint Rectangle Escape($\updownarrow$), *and* Disjoint Rectangle Escape($\updownarrow\!\!\!\rightarrow$) *can all be solved in* $\mathcal{O}(n \log n)$ *time.*

4 Two Opposite Directions

In this section, we prove Theorems 2 and 3.

▶ **Theorem 3.** Rectangle Escape ($\updownarrow$) *is* NP*-hard.*

Proof. We reduce from Vertex Cover, one of Karp's original 21 NP-hard problems. In Vertex Cover, we are given an undirected graph $G = (V, E)$ and an integer k as input. The question is whether there exists a vertex set $S \subseteq V$ of size at most k such that each edge

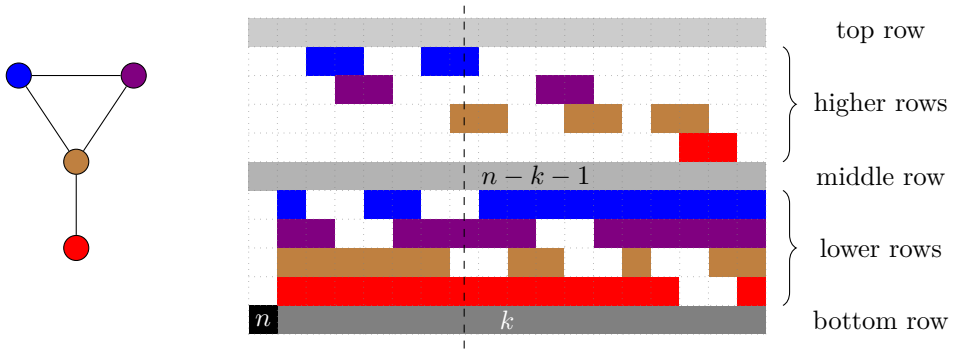

Figure 5 Example construction for the input graph on the left. The numbers in boxes show the input density (how many overlapping rectangles there are before extending any rectangle). The rectangles corresponding to vertices are color-coded and the vertical dashed line shows that extending all violet and red rectangles upwards and all blue and brown rectangles downwards is not a solution as it creates density $5 > n$ in the middle row (and this is indeed not a vertex cover).

in E has at least one endpoint in S. Let $(G = (V, E), k)$ be an input instance for Vertex Cover and let $n = |V|$ and $m = |E|$ be the number of vertices and edges in G, respectively. We assume without loss of generality that $V = \{v_1, v_2, \ldots, v_n\}$ and $E = \{e_1, e_2, \ldots, e_m\}$. We will next construct an equivalent instance of Rectangle Escape($\updownarrow$) with $d = n$. An illustration of the following reduction can be seen in Figure 5.

We start with a boundary rectangle $\mathcal{R}$ of height $2n + 3$ and width $4m + 2$. We divide the rectangle into a grid with $2n + 3$ rows of height 1 and $4m + 2$ columns of width 1.

The first row is the *bottom row*, row number $n+2$ is the *middle row*, and row number $2n+3$ is the *top row*. For each $v_i \in V$, the row $i + 1$ is the *lower row* for v_i and row $n + i + 2$ is the *higher* row for v_i. We collectively call all higher rows for any vertex the higher rows and all lower rows for any vertex the lower rows. The two leftmost columns are the *set-up columns* and we call the first column the left set-up column and the second column the right set-up column. Columns $4i - 1$ to $4i + 2$ correspond to e_i for each $i \in [m]$. All of the following rectangles have height one and if we say that we place a rectangle in a specific row, then we mean that the height of the rectangle completely fills this row. We first place one rectangle in the top row that spans all $3m + 2$ columns. Similarly, we place $n - k - 1$ (overlapping) rectangles in the middle row (also spanning all columns). Finally, we place k rectangles in the bottom row (also spanning all columns) plus $n - k$ additional rectangles that only span the left set-up column.

Next, we place rectangles for each vertex in G. For each vertex $v_i \in V$, we will place rectangles in the lower and higher row for v_i such that no such rectangle is in the left set-up column and for every other column, there is exactly one rectangle in either of the two rows for v_i. We start with one rectangle in the right set-up column in the lower row for each v_i. Let $e_j = \{v_{j_1}, v_{j_2}\}$ be an edge where $j_1 < j_2$. For each vertex $v \in V \setminus \{v_{j_1}, v_{j_2}\}$, there is a rectangle in the lower row for v in all four columns corresponding to e_j. For v_{j_1}, there is a rectangle in the first two columns corresponding to e_j in the higher row for v_{j_1} and a rectangle in third and fourth column corresponding to e_j in the lower row for v_{j_1}. For v_{j_2}, there is a rectangle in the first and fourth column corresponding to e_j in the lower row for v_{j_2} and a rectangle in the second and third column corresponding to e_j in the higher row for v_{j_2}. Whenever two rectangles in the same row touch each other, we merge them into a single rectangle. This concludes the construction. Before we prove the correctness, we first prove a couple of key facts about the reduced instances.

First, note that there are already n rectangles in the bottom row in the first set-up column. Hence, if any rectangle in the top or middle row is extended towards the bottom, then this creates density $n+1 > d$. Since all these rectangles occupy the entire width of their respective row, we already have density $n-k$ in the top row everywhere and therefore at most k other rectangles can be extended upwards in each column. Moreover, since there are k rectangles in the bottom row in each column (which can be assumed to be extended to the bottom without loss of generality as they already touch the lower boundary), at most $n-k$ other rectangles can be extended towards the bottom in each column.

Note that by construction, there are n rectangles in each column except for the first set-up column that were not considered so far and exactly one corresponds to each vertex in V. Moreover, we show next that between any two consecutive columns (excluding from the first to the second set-up column) there is exactly one vertex v such that either a rectangle in the higher row for v ends at that point and a rectangle in the lower row for v starts or the other way around, that is, a rectangle in the lower row for v ends and a rectangle in the higher row for v starts at that point. To this end, consider any two consecutive columns $i, i+1$. Let $e_j = \{v_{j_1}, v_{j_2}\}$ with $j_1 < j_2$ be the edge such that column $i+1$ corresponds to e_j. We make a case distinction on whether $i+1$ is the first, second, third, or fourth column corresponding to e_j. If $i+1$ is the first column corresponding to e_j, then a rectangle in the lower row for v_{j_1} ends in column i and a rectangle in the higher row for v_{j_1} starts in column $i+1$. If $i+1$ is the second column corresponding to e_j, then a rectangle in the lower row for v_{j_2} ends in column i and a rectangle in the higher row for v_{j_2} starts in column $i+1$. If $i+1$ is the third column corresponding to e_j, then a rectangle in the higher row for v_{j_1} ends in column i and a rectangle in the lower row for v_{j_1} starts in column $i+1$. If $i+1$ is the fourth column corresponding to e_j, then a rectangle in the higher row for v_{j_2} ends in column i and a rectangle in the lower row for v_{j_2} starts in column $i+1$. No other rectangles start or end between these two columns.

We next show that all rectangles corresponding to the same vertex are extended in the same direction in any solution. Assume towards a contradiction that there is a solution with maximum density $d = n$ such that for some vertex v, there are two rectangles corresponding to v where one is extended to the top and the other is extended to the bottom. Since there is exactly one rectangle corresponding to v in each column, there are two consecutive columns i and $i+1$ such that the rectangle corresponding to v in column i is extended in a different direction than the rectangle corresponding to v in column $i+1$. Since the two cases are analogous, we assume that the rectangle in column i is extended upwards and the rectangle in column $i+1$ is extended downwards. Note that since there are exactly $2n$ rectangles in column i, it holds that exactly n of them are extended to the top in the assumed solution and exactly n are extended to the bottom. Note that the n rectangles that are extended to the bottom do not end in column i and therefore also exist in column $i+1$. Together with the rectangle for v in column $i+1$ that is extended to the bottom, this creates density $n+1 > d$ in the bottom row, a contradiction.

We next show that the constructed instance is indeed equivalent to the input instance of Vertex Cover. Since the reduction can clearly be computed in polynomial time, this will conclude the proof. To show correctness, assume first that there is a vertex cover $S \subseteq V$ of size at most k in G. We extend all rectangles in the top and middle row and all rectangles in the lower or higher row for vertices in S to the top. We extend all other rectangles to the bottom. We next show that this creates maximum density $d = n$. Note that the top and bottom rows have density n everywhere except for the left setup column in the top row by construction. Each point in any lower row is only contained in rectangles corresponding to

vertices in any solution, that is, they have density at most n. Hence, it remains to analyze the middle row and the higher rows. Note that in each column, the density in these rows can only be greater than in the top row due to rectangles in higher rows being extended to the bottom. However, in each column there are only two such vertices and by construction, at least one of these vertices belongs to S. Thus, in the described area, the maximum density consists of at most one rectangle being extended to the bottom plus the $n-k-1$ rectangles in the middle row plus at most k rectangles being extended to the top (including at least one in a higher row). The maximum density is therefore at most $n-k-1+1+k=n=d$. This concludes the forward direction of the proof.

For the other direction, assume that there is a solution for the constructed instance of Rectangle Escape($\updownarrow$). As shown above, we can assume that all rectangles corresponding to a given vertex are extended in the same direction (upwards or downwards) and there are exactly k vertices for which rectangles are extended upwards. We next show that these vertices form a vertex cover in G. Assume towards a contradiction that this is not the case, that is, there is an edge $e=\{v_i,v_j\}$ such that all rectangles for v_i and for v_j are extended downwards. Consider the second column corresponding to e. Exactly k rectangles corresponding to vertices are extended upwards in this column and since we assume that v_i and v_j are not among those, all of these rectangles are below the middle row and therefore their extension intersects the middle row. Moreover, there are rectangles for v_i and for v_j in the considered column in the respective higher rows. Since we assume that these rectangles are extended downwards, they also intersect the middle row. As there are $n-k-1$ rectangles in the middle row initially, this leads to a density of $n-k-1+k+2=n+1>d$, a contradiction to the assumption that we started with a solution for Rectangle Escape($\updownarrow$) with density at most d. This concludes the proof. ◀

We next restate Theorem 2, but defer its proof to the appendix.

▶ **Theorem 2 (⋆).** Rectangle Escape($\updownarrow$) *can be solved in* $\mathcal{O}(4^d dn + n\log n)$ *time.*

A straight-forward generalization of Theorem 2 also allows us to solve Rectangle Escape($\updownarrow\!\!\rightarrow$) in $\mathcal{O}(2^n 4^d dn)$ time and Rectangle Escape in $\mathcal{O}(3^n 4^d dn)$ time. In a nutshell, we guess which rectangles are extended to the left and/or right and then use a small adaptation of Theorem 2 to compute an optimal way of extending the remaining rectangles to the top or bottom. Due to space constraints, we have to defer proof details to a full version.

▶ **Corollary 10.** Rectangle Escape *can be solved in* $\mathcal{O}(3^n 4^d dn)$ *time and* Rectangle Escape($\updownarrow\!\!\rightarrow$) *can be solved in* $\mathcal{O}(2^n 4^d dn)$ *time.*

5 Conclusion

We studied Rectangle Escape and Disjoint Rectangle Escape under various restrictions on the directions in which rectangles are allowed to escape. We resolved two open problems from the existing literature [1], showing that Rectangle Escape($\leftrightarrows$) is NP-hard for $d=2$, and that Rectangle Escape($\updownarrow$) is NP-hard when d is part of the input, but solvable in $\mathcal{O}(n\log n)$ time for any constant d.

For Disjoint Rectangle Escape, we proved an unconditional lower bound of $\Omega(n\log n)$ together with a matching upper bound for any (sub)set of directions. Moreover, we presented the first improvement over the naïve $\mathcal{O}(4^n n)$-time algorithm for Rectangle Escape when $d\in o(n)$ by designing an algorithm that runs in $\mathcal{O}(3^n 4^d dn)$ time. An exciting open question is whether one can design a truly $(4-\varepsilon)^n$-time algorithm for Rectangle Escape for some $\varepsilon>0$ and all d.

References

1 AmirMahdi Ahmadinejad, Sepehr Assadi, Ehsan Emamjomeh-Zadeh, Sadra Yazdanbod, and Hamid Zarrabi-Zadeh. On the rectangle escape problem. *Theoretical Computer Science*, 689:126–136, 2017. doi:10.1016/J.TCS.2017.05.040.

2 AmirMahdi Ahmadinejad and Hamid Zarrabi-Zadeh. Finding maximum disjoint set of boundary rectangles with application to PCB routing. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 36(3):412–420, 2017. doi:10.1109/TCAD.2016.2585761.

3 Andreas Darmann and Janosch Döcker. On simplified NP-complete variants of monotone 3-Sat. *Discrete Applied Mathematics*, 292:45–58, 2021. doi:10.1016/J.DAM.2020.12.010.

4 J. Mark Keil, Joseph S. B. Mitchell, Dinabandhu Pradhan, and Martin Vatshelle. An algorithm for the maximum weight independent set problem on outersting graphs. In *Proceedings of the 27th Canadian Conference on Computational Geometry (CCCG)*. Queen's University, 2015.

5 Hui Kong, Qiang Ma, Tan Yan, and Martin D. F. Wong. An optimal algorithm for finding disjoint rectangles and its application to PCB routing. In Sachin S. Sapatnekar, editor, *Proceedings of the 47th Design Automation Conference (DAC)*, pages 212–217. ACM, 2010. doi:10.1145/1837274.1837326.

6 Hui Kong, Tan Yan, and Martin D. F. Wong. Automatic bus planner for dense PCBs. In *Proceedings of the 46th Design Automation Conference (DAC)*, pages 326–331. ACM, 2009. doi:10.1145/1629911.1630000.

7 Qiang Ma and Martin D. F. Wong. NP-completeness and an approximation algorithm for rectangle escape problem with application to PCB routing. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 31(9):1356–1365, 2012. doi:10.1109/TCAD.2012.2193581.

8 Qiang Ma, Evangeline F. Y. Young, and Martin D. F. Wong. An optimal algorithm for layer assignment of bus escape routing on PCBs. In *Proceedings of the 48th Design Automation Conference (DAC)*, pages 176–181. ACM, 2011. doi:10.1145/2024724.2024764.

9 Muhammet Mustafa Ozdal, Martin D. F. Wong, and Philip S. Honsinger. An escape routing framework for dense boards with high-speed design constraints. In *Proceedings of the 2005 International Conference on Computer-Aided Design (ICCAD)*, pages 759–766. IEEE Computer Society, 2005. doi:10.1109/ICCAD.2005.1560166.

10 Franco P. Preparata and Michael Ian Shamos. *Computational Geometry: An Introduction*. Springer, 1985. doi:10.1007/978-1-4612-1098-6.

11 Aniket Basu Roy, Sathish Govindarajan, Neeldhara Misra, and Shreyas Shetty. On the d-runaway rectangle escape problem. In *Proceedings of the 26th Canadian Conference on Computational Geometry (CCCG)*. Carleton University, 2014.

12 Craig A. Tovey. A simplified NP-complete satisfiability problem. *Discrete Applied Mathematics*, 8(1):85–89, 1984. doi:10.1016/0166-218X(84)90081-7.

13 Pei-Ci Wu, Qiang Ma, and Martin D. F. Wong. An ILP-based automatic bus planner for dense PCBs. In *Proceedings of the 18th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 181–186. IEEE, 2013. doi:10.1109/ASPDAC.2013.6509593.

14 Jin-Tai Yan and Zhi-Wei Chen. Direction-constrained layer assignment for rectangle escape routing. In *Proceedings of the 25th IEEE IEEE International System-on-Chip Conference (SOCC)*, pages 254–259. IEEE, 2012. doi:10.1109/SOCC.2012.6398357.

15 Jin-Tai Yan, Jun-Min Chung, and Zhi-Wei Chen. Density-reduction-oriented layer assignment for rectangle escape routing. In *Proceedings of the 22nd ACM Great Lakes Symposium on VLSI (GLSVLSI)*, pages 275–278. ACM, 2012. doi:10.1145/2206781.2206848.

16 Tan Yan, Hui Kong, and Martin D. F. Wong. Optimal layer assignment for escape routing of buses. In *Proceedings of the 2009 International Conference on Computer-Aided Design (ICCAD)*, pages 245–248. ACM, 2009. doi:10.1145/1687399.1687444.

A Appendix

Missing proof details for Theorem 1

We provide a formal description of the clause rectangles in the different cases. We start with negative clauses. Let C_i^n be a negative clause. If C_i^n contains three literals $\neg x, \neg y$, and $\neg z$ with $x \prec y \prec z$, then we add 10 rectangles plus one special rectangle called the *helper rectangle* as follows. The helper rectangle cover the intersection of the first column corresponding to x and the 5^{th} and 6^{th} row corresponding to C_i^n. The other ten rectangles each cover the intersection of one row and one column as follows:

- the 1^{st} row corresponding to C_i^n and the 5^{th} column corresponding to x,
- the 1^{st} row corresponding to C_i^n and the 5^{th} column corresponding to y,
- the 2^{nd} row corresponding to C_i^n and the 4^{th} column corresponding to x,
- the 2^{nd} row corresponding to C_i^n and the 3^{rd} column corresponding to y,
- the 3^{rd} row corresponding to C_i^n and the 3^{rd} column corresponding to x,
- the 3^{rd} row corresponding to C_i^n and the 4^{th} column corresponding to y,
- the 4^{th} row corresponding to C_i^n and the 2^{nd} column corresponding to x,
- the 4^{th} row corresponding to C_i^n and the 2^{nd} column corresponding to y,
- the 5^{th} row corresponding to C_i^n and the 2^{nd} column corresponding to z, and
- the 6^{th} row corresponding to C_i^n and the 4^{th} column corresponding to z.

If C_i^n only contains two literals $\neg x$ and $\neg y$ with $x \prec y$, then we place one helper rectangle in the border column in the first four rows corresponding to C_i^n and eight other rectangles each covering the intersection of one row and one column as follows:

- the 1^{st} row corresponding to C_i^n and the 5^{th} column corresponding to x,
- the 1^{st} row corresponding to C_i^n and the 5^{th} column corresponding to y,
- the 2^{nd} row corresponding to C_i^n and the 4^{th} column corresponding to x,
- the 2^{nd} row corresponding to C_i^n and the 3^{rd} column corresponding to y,
- the 3^{rd} row corresponding to C_i^n and the 3^{rd} column corresponding to x,
- the 3^{rd} row corresponding to C_i^n and the 4^{th} column corresponding to y,
- the 4^{th} row corresponding to C_i^n and the 2^{nd} column corresponding to x, and
- the 4^{th} row corresponding to C_i^n and the 2^{nd} column corresponding to y.

We now turn towards positive clauses. Let C_i^p be a positive clause. If C_i^p contains three literals x, y, and z with $x \prec y \prec z$, then let $j_x \in \{1, 2\}$ be the index such that x appears for the ${j_x}^{\text{th}}$ time positive in C_i^p. We define j_y and j_z analogously for y and z, respectively. We place a helper rectangle in the intersection of the second column corresponding to C_i^p and the $2j_x - 1^{\text{st}}$ row corresponding to x. Moreover, we place three rectangles each covering the intersection of one row and one column as follows:

- the ${2j_x}^{\text{th}}$ row corresponding to x and the first column corresponding to C_i^p,
- the ${2j_y}^{\text{th}}$ row corresponding to y and the first column corresponding to C_i^p, and
- the ${2j_z}^{\text{th}}$ row corresponding to z and the second column corresponding to C_i^p.

If C_i^p contains two literals $x \prec y$, then we define j_x and j_y as before, place a helper rectangle covering the intersection of the border row and the first column corresponding to C_i^p and two rectangles each covering the intersection of one row and one column as follows:

- the ${2j_x}^{\text{th}}$ row corresponding to x and the first column corresponding to C_i^p and
- the ${2j_y}^{\text{th}}$ row corresponding to y and the first column corresponding to C_i^p.

This concludes the construction. Next, we prove that the construction is correct.

▷ Claim 11. The constructed instance of Rectangle Escape($\curvearrowleft\!\!\curvearrowright$) has a solution with maximum density two if and only if ϕ is satisfiable.

Proof. First, assume that ϕ is satisfiable and let β be a satisfying assignment. We start by extending all border rectangles in the border column to the left and all border rectangles in the border row to the bottom. Next, for each variable x, if $\beta(x)$ is true, then we extend x_1 and x_2 to the bottom and otherwise we extend x_1 and x_2 to the left. For each clause rectangle R (except helper rectangles) for a negative clause, let x_R be the variable such that R is contained in a variable column corresponding to x_R. If $\beta(x_R)$ is true, then we extend R to the left. Otherwise, we extend R to the bottom. For each negative clause C_i^n, if C_i^n contains three literals $\neg x, \neg y$, and $\neg z$ with $x \prec y \prec z$, then we extend the helper rectangle for C_i^n to the left if and only if $\beta(z)$ is false. If C_i^n contains only two literals, then we extend the helper rectangle to the left. For each clause rectangle R (again except helper rectangles) for a positive clause, let x_R be the variable such that R is contained in a variable row corresponding to x_R. If $\beta(x_R)$ is true, then we extend R to the left. Otherwise, we extend R to the bottom. For each positive clause C_i^p, if C_i^p contains three literals x, y, and z with $x \prec y \prec z$, then we extend the helper rectangle for C_i^p to the bottom if and only if $\beta(z)$ is true. If C_i^p contains only two literals, then we extend the helper rectangle to the bottom.

We will next show that the constructed solution indeed has maximum density two. Note that the following holds for each helper rectangle for a negative clause with three literals: In the variable column that contains this helper rectangle, there is only one other rectangle which is a border rectangle. This holds as each variable appears only once negated. Similarly by construction, each variable column containing a variable rectangle contains at most one clause rectangle and at most one of these two rectangles is extended to the bottom. The same argument applies to helper rectangles for positive clauses, variable rows, and extensions to the left as each variable appears at most twice and therefore at most once for the first time and once for the second time. Moreover, if a clause rectangle in a variable row (equivalently a clause rectangle for a positive clause) is extended to the left, then the respective variable rectangle is extended to the bottom and hence no clause rectangle in a variable column corresponding to that variable are extended to the bottom. Thus, in the area that is the intersection of variable rows and columns, the maximum density is at most two. Since no rectangles are placed in the intersection of clause rows and clause columns, we can analyze the clause rows and the clause columns individually.

We start with the clause rows. For a negative clause C_i^n containing two literals, note that by construction at least one of the two clause rectangles (excluding the helper rectangle) in each of the first four rows corresponding to C_i^n are extended to the bottom as β is by assumption a satisfying assignment. Moreover, the fifth and sixth row only contain a single border rectangle. Hence, the maximum density in these columns is at most two unless a clause rectangle for a clause C_j^n with $j > i$ is extended to the bottom. However, note that we sorted all clauses lexicographically. This means that all clauses appearing later only contain variables that appear after the first variable of C_i^n. Hence, in columns where rectangles from C_j^n can be extended to the bottom, the density of rectangles for C_i^n is at most one, yielding maximum density two in all clause rows. If C_i^n contains three literals, then the same argument as above shows that the density in every column containing a clause rectangle from a clause C_j^n with $j > i$ has maximum density two. The only remaining possibility to create density three in a clause row of C_i^n is if the helper rectangle is extended to the bottom and for one of the first four rows for C_i^n, both rectangles in that row are extended to the left. Note that this corresponds to β setting all three variables appearing in C_i^n to true, contradicting the fact that β is a satisfying assignment.

We next turn to the clause columns and mention that the arguments are basically the same as for clause rows. For a positive clause C_i^p containing two literals, at least one of the two non-helper clause rectangles in the first column corresponding to C_i^p are extended

to the left as β is by assumption a satisfying assignment. Moreover, the second column only contain a single border rectangle. The maximum density in these columns is therefore at most two unless a clause rectangle for a clause C_j^p with $j > i$ is extended to the left. However, these helper rectangles are above one of the two relevant rectangles due to the lexicographical order. Finally, the density in the border row and the border column is also at most two. If C_i^p contains three literals, then the same arguments as above apply showing that the density in every column containing a clause rectangle from a clause C_j^p with $j > i$ has maximum density two. The only remaining possibility to create density three in a clause row of C_i^p is if the helper rectangle is extended to the left and both rectangles in the first column for C_i^p are extended to the bottom. Note that this corresponds to β setting all three variables appearing in C_i^p to false, contradicting the fact that β is a satisfying assignment. This concludes the proof for the forward direction.

We next show the other direction. To this end, assume that there is a solution γ to the constructed instance of Rectangle Escape($\leftarrow\downarrow$). We construct a satisfying assignment β by setting $\beta(x)$ to true whenever at least one of the two rectangles x_1 and x_2 is extended to the bottom. Assume towards a contradiction that β is not a satisfying assignment. Then, there is at least one clause C which is not satisfied by β. We make a case distinction whether C is a positive or a negative clause and whether it contains two or three literals.

If C is a positive clause containing three literals x, y, and z with $x \prec y \prec z$, then by construction x_1, x_2, y_1, y_2, z_1 and z_2 are all extended to the left. Hence, all three clause rectangles for C except for the helper rectangle have to be extended to the bottom. If the helper rectangle for C is extended to the bottom, then it creates density three in the intersection of the second clause column corresponding to C and the border row. If the helper rectangle for C is extended to the left, then it creates density three in the intersection of the first (clause) column corresponding to C and the row containing the helper variable for C. Hence, in both cases we reach a contradiction to the fact that γ is a solution with maximum density two.

If C is a positive clause containing two literals x and y, then by construction x_1, x_2, y_1, and y_2 are all extended to the left. Then, both (non-helper) clause rectangles for C are extended to the bottom. Now, both of these rectangles intersect the helper rectangle in the border row creating density three, a contradiction to γ being a solution with maximum density two.

If C is a negative clause containing three literals $\neg x, \neg y$, and $\neg z$ with $x \prec y \prec z$, then by construction, the solution S extends at least one of the two rectangles z_1 and z_2 to the bottom. This means that at least one of the two (non-helper) rectangles in the fifth and sixth row corresponding to C is extended to the left. This implies that the helper rectangle for C is extended to the bottom. Moreover, at least one of x_1 and x_2 is extended to the bottom by construction and the same holds for y_1 and y_2. Let without loss of generality be x_1 and y_2 be extended to the bottom (the other three cases are analogous). Then, both rectangles in the third clause row corresponding to C have to be extended to the left. Note, however, that the density in the intersection of the first column corresponding to x and the third row corresponding to C has density three, a contradiction.

If C is a negative clause containing two literals $\neg x$ and $\neg y$ with $x \prec y$, then the solution γ extends at least one of the two rectangles x_1 and x_2 to the bottom and the same holds for y_1 and y_2. Let without loss of generality x_2 and y_1 be extended to the bottom (the other three cases are analogous). Then, both rectangles in the second row corresponding to C have to be extended to the left. Now, both of these rectangles intersect the helper rectangle in the border column creating density three, a contradiction to γ being a solution with maximum density $d = 2$.

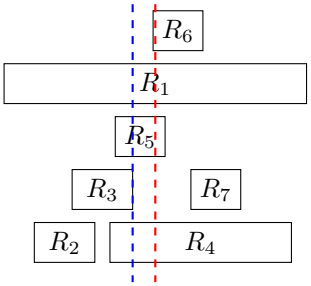

Figure 6 An example of Rectangle Escape($\updownarrow$) with $d = 2$. The dynamic program for the left (blue) sweep line has the following entries set to true: R_1 and R_5 escape to the top and R_4 to the bottom (which is possible if R_3 escapes to the bottom and R_2 in either direction) and R_1 escapes to the top and R_4 and R_5 escape to the bottom (which is possible when R_3 escapes to the top and R_2 to the bottom). For the right (red) sweep line, the dynamic program has only a single true entry: R_1 and R_6 escape to the top and R_4 and R_5 escape to the bottom.

Since we found a contradiction to the assumption that C is not satisfied by β for all cases of C, this shows that all clauses are in fact satisfied by β, showing that ϕ is satisfiable. This concludes the proof. ◁

Proof of Theorem 2

▶ **Theorem 2** ($\star$). Rectangle Escape($\updownarrow$) *can be solved in* $\mathcal{O}(4^d dn + n \log n)$ *time.*

Proof. We assume without loss of generality that the x-coordinates of the left and right border are distinct for each rectangle. We start by sorting the left and right border of all rectangles by their x-coordinate in $\mathcal{O}(n \log n)$ time. Next, we present a sweep-line dynamic-programming algorithm, that is, intuitively, we go through all points from left to right, keeping track of which rectangles intersect the current position of the vertical sweep line. For each combination of letting these rectangles escape to the top or bottom, we store whether there is a solution for all rectangles with their left endpoint to the left of the sweep line in which all "current" rectangles escape as described. See Figure 6 for an example.

We now proceed to state the dynamic program formally. To this end, let $p_1, p_2, \ldots, p_{2n}$ be the sorted x-positions where rectangles start or end. We say that $p_i = \ell_j$ if the left side of rectangle R_j has x-coordinate p_i and $p_i = r_j$ if the right side of R_j has x-coordinate p_i. For any $i \in [2n]$, let $A_i \subseteq \mathcal{S}$ be the set of rectangles that are active at position p_i, that is, $R_j \in A_i$ if and only if $\ell_j \leq p_i < r_j$. Note that due to the definition of rectangles as the Cartesian product of two half-open intervals, R_j is active at position ℓ_j but not at position r_j. Moreover, let $L_i = \bigcup_{j \leq i} A_j$ be the set of all rectangles which are active or were already active to the left of p_i. Our dynamic program $T \colon \mathbb{N} \times 2^{\mathcal{S}} \to \mathbb{B}$ will store for each position $i \in [2n]$ and each subset $B \subseteq A_i$, whether there is a solution for the instance induced by L_i where all rectangles in B escape to the top and all rectangles in $A_i \setminus B$ escape to the bottom. We compute $T[i, B]$ as follows (assuming that $T[i-1, B']$ has been correctly computed for all $B' \subseteq A_{i-1}$ where initially $T[0, \emptyset]$ is set to true).

If $p_i = \ell_j$ for some rectangle R_j, then we set $T[i, B]$ to true if $T[i-1, B \setminus \{R_j\}]$ is true and if letting all rectangles in B escape to the top and all rectangles in $A_i \setminus B$ to the bottom does not create density $d+1$ anywhere on the vertical line at x-position p_i. Otherwise, we set $T[i, B]$ to false. If $p_i = r_j$ for some rectangle R_j, then we set $T[i, B]$ to true if and only if $T[i-1, B]$ or $T[i-1, B \cup \{R_j\}]$ is set to true. After the entire table has been computed, we return true if and only if $T[2n, \emptyset]$ is set to true. This concludes the description of the algorithm.

We show that the algorithm correctly solves Rectangle Escape($\updownarrow$) via induction on i and afterwards, we analyze the running time. For the base case, note that setting $T[0, \emptyset]$ to true is correct as $A_0 = \emptyset$ and not extending any rectangles does not create any density. So for the induction step, assume that $T[i', B']$ is correctly computed for each $i' < i$ and each $B' \subseteq A_{i'}$. For the first direction, assume for some $B \subseteq A_i$ that all rectangles in L_i can escape in such a way that all rectangles in B escape to the top, all rectangles in $A_i \setminus B$ escape to the bottom, and the density everywhere is at most d. We show that $T[i, B]$ is set to true. By assumption, letting all rectangles in B escape to the top and all rectangles in $A_i \setminus B$ to the bottom does not create density $d+1$ anywhere so also not at position p_i. If $p_i = \ell_j$ for some rectangle R_j, then note that $T[i-1, B \setminus \{R_j\}]$ is set to true by the induction hypothesis and therefore by construction $T[i, B]$ is also set to true. If $p_i = r_j$ for some rectangle R_j, then we consider the two cases whether R_j escapes to the top or to the bottom in the assumed solution. If R_j escapes to the top, then note that by the induction hypothesis $T[i-1, B \cup \{R_j\}]$ is set to true and hence $T[i, B]$ is also set to true by construction. If R_j escapes to the bottom, then $T[i-1, B]$ is set to true by the induction hypothesis and $T[i, B]$ is then set to true by construction.

For the reverse direction, assume that $T[i, B]$ is set to true for some $B \subseteq A_i$. We show that all rectangles in L_i can escape in such a way that all rectangles in B escape to the top, all rectangles in $A_i \setminus B$ escape to the bottom, and the density everywhere is at most d. We make a case distinction whether $p_i = \ell_j$ or $p_i = r_j$ for some rectangle R_j. If $p_i = \ell_j$, then note that since $T[i, B]$ is set to true, it holds that $T[i-1, B \setminus \{R_j\}]$ has to also be set to true and letting all rectangles in B escape to the top and all rectangles in $A_i \setminus B$ does not create density $d+1$. By the induction hypothesis, this means that the density at every point with an x-coordinate less than p_i is at most d. Hence, the density is everywhere at most d. If $p_i = r_j$, then note that $T[i-1, B]$ or $T[i-1, B \cup \{R_j\}]$ must be set to true. By the induction hypothesis, this means that the density at every point with an x-coordinate less than p_i is at most d. Since the density can only drop at x-coordinate p_i (as a rectangle disappears and no new rectangle appears), the density is everywhere at most d. This concludes the induction step.

It remains to analyze the running time. Note that if $|A_i| > 2d$ for some $i \in [2n]$ (which can be tested in linear time after the sorting step), then we can immediately conclude that there is no solution as at least $d+1$ rectangles in A_i have to escape to either the top or the bottom and both clearly creates density $d+1$ somewhere. So we may assume that $|A_i| \leq 2d$ and hence, there are at most $2^{2d} = 4^d$ table entries for each position i. Hence, the number of table entries is overall in $\mathcal{O}(4^d n)$. We can precompute all sets A_i and sort their y-positions in overall $\mathcal{O}(nd \log d)$ time. Afterwards, we can compute each table entry in $\mathcal{O}(d)$ time by extending all at most d rectangles in B to the top, all at most d rectangles in $A_i \setminus B$ to the bottom and then go through the sorted list of y-coordinates and keep track of the maximum density in $\mathcal{O}(d)$ time. Thus, the overall running time of our algorithm is in $\mathcal{O}(4^d dn + n \log n)$. This concludes the proof. ◀