The Maximum Disjoint Set of Boundary Rectangles

AmirMahdi AhmadiNejad* Hamid Zarrabi-Zadeh*

Abstract

We consider the problem of finding a maximum disjoint set of boundary rectangles, where all rectangles are attached to the boundary of a bounding box. We present an algorithm for solving the problem in $O(n^4)$ time, improving upon the best previous $O(n^6)$-time solution available for the problem.

1 Introduction

In this paper, we study the problem of finding a maximum disjoint set of boundary rectangles, which is defined as follows.

Problem 1 *Given an axis-parallel rectangular region $\mathcal{R}$, and a set $S = \{1, 2, \ldots, n\}$ of n boundary rectangles inside $\mathcal{R}$, where all rectangles are attached to the boundary of $\mathcal{R}$, find a subset $T \subseteq S$ of disjoint rectangles such that $\sum_{i \in T} w_i$ is maximized, where w_i is the weight of rectangle i.*

Figure 1 illustrates an instance of the problem, in which all rectangles have the same weight. The problem is mainly motivated by the following *bus escape routing* application in VLSI design [2, 8]. Suppose n chips are placed on a circuit board, where all chips and the circuit board are axis-parallel rectangles. We want to route all chips to the boundary of the board using a set of disjoint horizontal/vertical buses of minimum total length/area. One can reduce this problem to Problem 1 by replacing each chip by four boundary rectangles obtained by extending the chip in four directions, and assigning to each boundary rectangle an appropriate weight, e.g., its area or length.

The problem of finding a maximum disjoint set of axis-parallel rectangles in the plane is known to be NP-hard [6]. The current best approximation algorithm for the problem is due to Chan and Har-Peled [4], which has an approximation factor of $O(\log \log n / \log n)$. Adamaszek and Wiese [1] have introduced a quasi-polynomial time $(1+\varepsilon)$-approximation algorithm for the problem. When all rectangles are fat (i.e., have bounded aspect ratio), PTASs are available [3, 5]. Kong *et al.* [7] have recently considered a special case of the problem

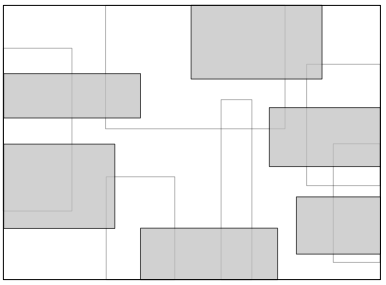


Figure 1: A set of boundary rectangles. A maximum disjoint set is shown in gray.

where all rectangles are attached to the boundary of a bounding box (i.e., Problem 1). They showed that this problem can be solved optimally in $O(n^6)$ time.

Our Contribution. In this paper, we present an improved $O(n^4)$-time algorithm for the problem of finding a maximum-weight disjoint set of boundary rectangles. Our algorithm improves the best previous $O(n^6)$-time algorithm for the problem presented by Kong *et al.* [7]. As a byproduct, we show that a 2-approximation to the problem can be computed in $O(n^2)$ time.

The idea is to use a dynamic programming technique, solving the problem using solutions to the smaller subproblems. In particular, similar to [7], we first solve the restricted k-sided subproblems, in which all boundary rectangles are attached to only k sides of the bounding box. The 1-sided case is equivalent to the problem of finding a maximum-weight set of disjoint intervals, and can be solved in $O(n \log n)$ time via dynamic programming.

For the 2-sided case, we present a new $O(n^2)$-time dynamic programming algorithm, improving the $O(n^3)$-time solution provided in [7]. Our algorithm is not only faster, but is also simpler and self-contained. In particular, our algorithm avoids using topological sorting and DAG shortest paths used as subroutines in [7]. For the 3-sided case, Kong *et al.* claimed an $O(n^3)$-time algorithm which is not fully correct, as it misses to consider all possible configurations of the optimal solution (see Section 4 for more details). A simple adaption of their solution to fix this issue increases the running time of the 3-sided case to $O(n^4)$, and hence, increases the total complexity of their algorithm to $O(n^7)$. Our solution to the 3-sided case uses a novel decomposition, enabling

*Department of Computer Engineering, Sharif University of Technology. am_ahmadinejad@ce.sharif.edu, zarrabi@sharif.edu

us to solve all possible 3-sided subproblems in $O(n^4)$ total time. Finally, for the general 4-sided problem, we devise a new decomposition, and manage to solve the whole problem in $O(n^4)$ overall time.

2 Preliminaries

Let $\mathcal{R}$ be a rectangular region in the plane, and $S = \{1, 2, \ldots, n\}$ be a set of n boundary rectangles inside $\mathcal{R}$, where each *boundary rectangle* is attached to one of the four sides of $\mathcal{R}$. Each rectangle $i \in S$ has a weight $w_i \geqslant 0$. Throughout this paper, we assume that all rectangles are axis-parallel. Two rectangles are *disjoint*, if their interior do not intersect. We denote by S_ℓ, S_r, S_t, and S_b the subsets of rectangles in S attached to the left, right, top, and bottom sides of $\mathcal{R}$, respectively, with ties being broken arbitrarily.

We denote by i_t and i_b the (y coordinates of the) top and bottom sides of rectangle i, respectively. Similarly, we use i_ℓ and i_r to denote the (x coordinates of the) left and right side of rectangle i, respectively. For ease of presentation, we assume that no two rectangles have vertical or horizontal sides at the same x or y coordinates. This restriction can be easily relaxed by imposing a total ordering on the rectangles sides to properly order sides with the same x or y coordinates.

Given two rectangles i and j in $S_\ell \cup S_r$, attached to the opposite sides of $\mathcal{R}$, we define the following types of *borders*:

- The border $\delta(i_t)$ is obtained by extending i_t. Similarly, $\delta(i_b)$ is obtained by extending i_b.
- If $j_b \leqslant i_t \leqslant j_t$, the border $\delta(i_t, j_b)$ is obtained by extending i_t until it hits rectangle j, then moving downward to reach j_b, and then finishing the border by adding j_b to it (see Figure 2).
- If $j_b \leqslant i_b \leqslant j_t$, the border $\delta(i_b, j_b)$ is obtained just like $\delta(i_t, j_b)$, except first extending i_b instead of i_t. If $i_b \leqslant j_b \leqslant i_t$, we set $\delta(i_b, j_b) = \delta(j_b, i_b)$.

There are $O(n^2)$ different borders. Given a border δ, let $R(\delta)$ denote the subregion of $\mathcal{R}$ bounded from above by δ. Let $B(\delta)$ be the set of all borders specified by the rectangles in $R(\delta)$. We define next(δ) to be a border $\delta' \in B(\delta)$ with the largest $R(\delta')$, i.e., with no other border of $B(\delta)$ lying between δ and δ'.

Lemma 1 *For each border δ,* next(δ) *can be computed in $O(1)$ time, after $O(n^2)$ preprocessing time.*

Proof. For each rectangle i, we keep a sorted list L_i of all borders containing i_b as their lower segment. If a border consists of only one segment, its lower and upper segments are the same. Furthermore, we define

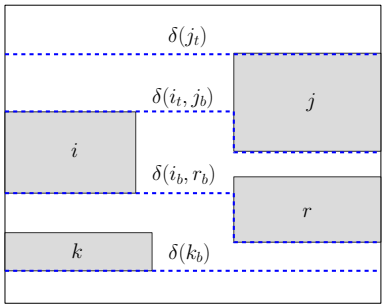


Figure 2: Different types of borders.

$n(i_b)$ (resp., $n(i_t)$) to be the rectangle with the highest top which is below i_b (resp., below i_t).

Suppose that the lower segment of δ is j_b. We look for the next element of δ in L_j. If such border exists, it is next(δ). Otherwise, next(δ) = $\delta(n(j_b))$, since there is no border after δ in L_j, and the highest border below δ should be completely below bottom(δ) = j_b. If $\delta = \delta(i_t)$, then next(δ) is equal to $\delta(n(i_t))$.

Now we show how to compute L_i and $n(\cdot)$ function for all rectangles in $O(n^2)$ time. For each rectangle i, $n(i_b)$ (resp., $n(i_t)$) can be computed by scanning all the rectangles and finding the one with the highest top which is below i_b (resp., below i_t). This takes $O(n)$ time for one rectangle, and $O(n^2)$ time for all rectangles. To compute L_i's, we first sort the set of all horizontal segments of borders (i.e., the top and bottom sides of the rectangles) in the decreasing order of their y coordinates in $O(n \log n)$ time. After that, for each rectangle i, L_i can be computed by a simple scan over the sorted list and generating all valid borders containing the bottom side of i in $O(n)$ time. Therefore, we can compute L_i for all rectangles in $O(n^2)$ overall time. □

3 The 2-Sided Problem

We start by solving the 2-sided problem, in which all input rectangles are attached to only two sides of the region $\mathcal{R}$. If these two sides are parallel, an *opposite* case arise, otherwise, it is called a *corner* case.

3.1 The Opposite Case

In the opposite case, either $S = S_\ell \cup S_r$ or $S = S_t \cup S_b$. We assume, w.l.o.g., that $S = S_\ell \cup S_r$. Given a border δ, we denote by Opposite(δ) the weight of an optimal solution lying completely inside $R(\delta)$. The answer to the opposite case is equal to Opposite($\delta(k_t)$), where k is the topmost rectangle in S.

Theorem 2 *For all borders δ,* Opposite(δ) *can be computed in $O(n^2)$ total time.*

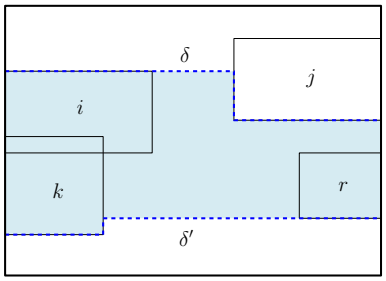


Figure 3: The region $R(\delta, \delta')$.

Proof. Let OPT be an optimal solution for the rectangles inside $R(\delta)$. We recursively compute Opposite(δ) as follows:

- $\delta = \delta(i_t, j_b)$: In this case, either $i \in$ OPT or not. If $i \in$ OPT, then Opposite(δ) $= w_i +$ Opposite($\delta(i_b, j_b)$). Otherwise, Opposite(δ) $=$ Opposite(next(δ)).
- $\delta = \delta(i_b, j_b)$: In this case, Opposite(δ) is simply equal to Opposite(next(δ)), since there is no valid rectangle between δ and next(δ).
- $\delta = \delta(i_t)$: Again, either $i \in$ OPT or not. If $i \in$ OPT, let k be the topmost rectangle below i_t in the opposite side of i, which does not intersect i. If $k_t \in [i_b, i_t]$ then Opposite(δ) $= w_i +$ Opposite($\delta(k_t, i_b)$), otherwise, Opposite(δ) $= w_i +$ Opposite($\delta(i_b)$). If $i \notin$ OPT, then Opposite(δ) $=$ Opposite(next(δ)).
- $\delta = \delta(i_b)$: In this case, Opposite(δ) is simply equal to Opposite(next(δ)).

By Lemma 1, the next borders are accessible in $O(1)$ time after $O(n^2)$ preprocessing time. Moreover, the rectangle k in the third case can be obtained similar to $n(i_t)$ function in Lemma 1 in $O(1)$ time after $O(n^2)$ preprocessing time. In all cases described above, Opposite(δ) can be computed using a constant number of previously-computed borders in a dynamic programming fashion. Since the number of borders is $O(n^2)$, Opposite(δ) for all borders δ can be computed in $O(n^2)$ total time. □

We define another related subproblem here. Given two borders δ and δ', we denote by $R(\delta, \delta')$ the region of $\mathcal{R}$ lying between δ and δ' (see Figure 3), and denote by Opposite(δ, δ') the weight of an optimal solution for the rectangles lying inside $R(\delta, \delta')$.

Corollary 1 *For all pairs of borders δ and δ', Opposite(δ, δ') can be computed in $O(n^4)$ total time.*

Proof. Fix a border δ'. We remove all rectangles below δ' in $O(n)$ time. We then use Theorem 2 to obtain the solutions to Opposite(δ, δ'), for all borders δ (above δ'), in $O(n^2)$ time. Since the number of borders δ' is $O(n^2)$, the total time needed is $O(n^4)$. □

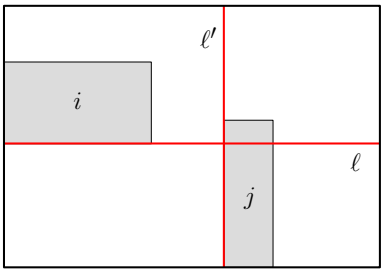


Figure 4: Illustration for the splitter line.

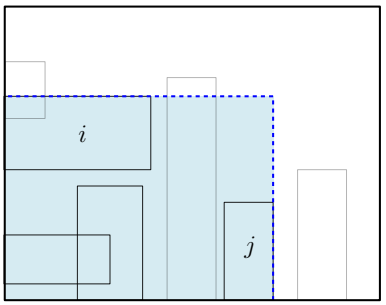


Figure 5: The region corresponding to Corner(i, j).

3.2 The Corner Case

In the corner case, all input rectangles are attached to only two neighboring sides of $\mathcal{R}$. We assume, w.l.o.g., that the two neighbor sides are the left and the bottom ones, i.e., $S = S_\ell \cup S_b$.

Consider an optimal solution to the corner case. We call a line a *splitter*, if it does not cut any rectangle in the optimal solution. The following lemma reveals a nice structural property of the optimal solution.

Lemma 3 *Let* OPT *be an optimal solution to the corner case, i be the topmost rectangle in* OPT $\cap S_\ell$*, and j be the rightmost rectangle in* OPT $\cap S_b$*. If ℓ and ℓ' are the lines obtained by extending i_b and j_ℓ, respectively, then either ℓ or ℓ' is a splitter (see Figure 4).*

Proof. Suppose, to the contrary, that both lines cut some rectangles in OPT. Note that ℓ can only cut rectangles from S_b, and ℓ' can only cut rectangles from S_ℓ. Suppose that ℓ cuts a rectangle $k \in S_b \cap$ OPT. Then k is either j or a rectangle completely to the left of j. Now any rectangle in $S_\ell \cap$ OPT intersecting ℓ' (which is either i or a rectangle below it) must also intersect rectangle k, which contradicts the disjointness of rectangles in OPT. □

Given two rectangles $i \in S_\ell$ and $j \in S_b$, we denote by $R(i_t, j_r)$ the region below i_t and to the left of j_r (see Figure 5). The regions $R(i_b, j_r)$, $R(i_t, j_\ell)$, and $R(i_b, j_\ell)$ are defined analogously. We use Corner(i_t, j_r) to denote the weight of an optimal solution to the corner case composed of the rectangles within $R(i_t, j_r)$.

Corner(i_t, j_ℓ), Corner(i_b, j_r), and Corner(i_b, j_ℓ) are defined analogously. We set Corner(i, j) = Corner(i_t, j_r).

Theorem 4 *For all pairs of rectangles $i \in S_\ell$ and $j \in S_b$,* Corner(i, j) *can be computed in $O(n^2)$ total time.*

Proof. Let $H = \{i_t : i \in S_\ell\} \cup \{i_b : i \in S_\ell\}$, and $V = \{j_\ell : j \in S_b\} \cup \{j_r : j \in S_b\}$. For each $x \in H$, we denote by $n_b(x)$ the topmost item in H below x. Moreover, for each $v \in V$, we denote by $n_\ell(v)$ the rightmost item in V to the left of v.

By Lemma 3, there is always a splitter that separates one rectangle of the optimal solution from the others. We consider two cases:

- Case 1: The splitter is defined by a rectangle in S_b. This rectangle is either j or some rectangle to the left of j_r. Therefore, Corner(i_t, j_r) = $\max\{w_j + \text{Corner}(i_t, j_\ell), \text{Corner}(i_t, n_\ell(j_r))\}$. Moreover, Corner$(i_t, j_\ell)$ = Corner$(i_t, n_\ell(j_\ell))$, since there is no valid rectangle between j_ℓ and $n_\ell(j_\ell)$. Corner(i_b, j_r) and Corner(i_b, j_ℓ) are computed similarly in this case.

- Case 2: The splitter is defined by a rectangle in S_ℓ. This rectangle is either i or some rectangle below i_t. Therefore, Corner(i_t, j_r) = $\max\{w_i + \text{Corner}(i_b, j_r), \text{Corner}(n_b(i_t), j_r)\}$. Moreover, Corner$(i_b, j_r)$ = Corner$(n_b(i_b), j_r)$. Corner(i_t, j_ℓ) and Corner(i_b, j_ℓ) are computed similarly in this case.

By the above formulae, we need to check a constant number of subproblems in order to compute Corner(i_x, j_z) for $x \in \{t, b\}$ and $z \in \{\ell, r\}$. Since the values of $n_b(\cdot)$ and $n_\ell(\cdot)$ are accessible in $O(1)$ time, after $O(n \log n)$ preprocessing (sorting) time, and the total number of subproblems is $O(n^2)$, we can compute Corner(i, j), for all pairs (i, j), in $O(n^2)$ total time. □

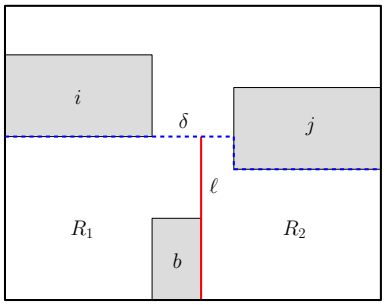
4 The 3-Sided Problem

In the 3-sided problem, one of the subsets S_ℓ, S_r, S_b and S_t is empty. We assume, w.l.o.g., that S_t is empty. Therefore, $S = S_\ell \cup S_b \cup S_r$. For a border δ, we define 3-sided(δ) to be the weight of an optimal solution for the rectangles inside $R(\delta)$.

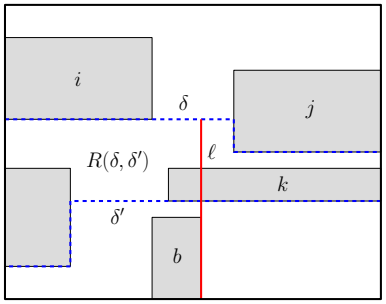
Theorem 5 *For all borders δ,* 3-sided(δ) *can be computed in $O(n^4)$ total time.*

Proof. Assume $\delta = \delta(i_t, j_b)$. (The other borders are handled analogously.) Let OPT be an optimal solution realizing 3-sided(δ). We have two cases:

- Case 1: $S_b \cap \text{OPT} = \emptyset$. Since there is no rectangle from S_b in OPT, the problem reduces to a 2-sided opposite case, i.e., 3-sided(δ) = Opposite(δ).



(a) Subcase 2.1



(b) Subcase 2.2

Figure 6: Case 2 of the 3-sided problem.

- Case 2: $S_b \cap \text{OPT} \neq \emptyset$. Here, there is at least one rectangle from S_b in OPT. Let b be the tallest such rectangle. We extend b_r until it touches δ to obtain the line segment ℓ (see Figure 6). The following two subcases arise:

  – Subcase 2.1: No rectangle in OPT intersects ℓ. Here, the region $R(\delta)$ is divided by ℓ into two disjoint subregions R_1 and R_2 (see Figure 6a), each of which contains rectangles from only two sides. Therefore, 3-sided(δ) = Corner(R_1) + Corner(R_2).

  – Subcase 2.2: At least one rectangle in OPT, say k, is intersecting ℓ (see Figure 6b). We assume, w.l.o.g., that $k \in S_r$. We construct a border δ' by extending k_b until it reaches the other side of the rectangular region $\mathcal{R}$, or it reaches another rectangle in $S_\ell \cap \text{OPT}$. Note that no rectangle in $S_b \cap \text{OPT}$ intersects δ', because b is the tallest rectangle in $S_b \cap \text{OPT}$. Now the region $R(\delta)$ is divided into two subregions $R(\delta, \delta')$ and $R(\delta')$. $R(\delta, \delta')$ is an opposite case and $R(\delta')$ is a 3-sided subproblem with δ' below δ. Therefore, 3-sided(δ) = Opposite(δ, δ') + 3-sided(δ').

The final solution, 3-sided(δ), is obtained by taking the maximum of all the above cases. In Case 1, we need the solution to Opposite(δ), which is available in $O(1)$ time, after $O(n^2)$ preprocessing time by Theorem 2. In Case 2.1, we remove all rectangles not in $R(\delta)$, and use Theorem 4 to compute all corner subproblems in

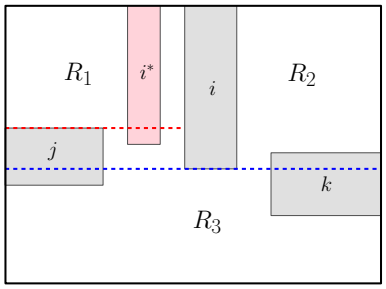


Figure 7: Illustrating a bug in Kong *et al.*'s algorithm.

$R(\delta) \cap (S_\ell \cup S_b)$ and $R(\delta) \cap (S_r \cup S_b)$ in $O(n^2)$ time. After that, Corner(R_1)+Corner(R_2) can be computed, for all rectangles $b \in S_b \cap R(\delta)$, in $O(n)$ time. Case 2.2 requires Opposite(δ, δ'), which is available in $O(1)$ time, after $O(n^4)$ preprocessing time by Corollary 1. Since δ' is below δ, by a dynamic programming approach, we compute 3-sided(δ') before 3-sided(δ), and hence, it is available upon computing 3-sided(δ) in $O(1)$ time. Overall, since there are $O(n^2)$ borders, and each requires $O(n^2)$ time, computing 3-sided(δ) for all borders δ takes $O(n^4)$ time, plus an $O(n^4)$ preprocessing time. □

Kong *et al.* [7] claimed an $O(n^3)$-time algorithm for the 3-sided case. Here, we briefly mention a bug in their solution. Let i be the tallest rectangle in $S_t \cap \text{OPT}$, and j be a rectangle in $S_\ell \cap \text{OPT}$ intersecting the line obtained by extending i_b (see Figure 7). Kong *et al.* solved this case by partitioning the region $\mathcal{R}$ into three subregions R_1, R_2, and R_3, where R_1 is defined as the region above j_t and to the left of i_ℓ. The final solution is then obtained by taking Corner(R_1) + Corner(R_2) + Opposite(R_3) + w_i + w_j. However, the solution computed this way may not be optimal. In particular, there can be a rectangle $i^* \in S_t \cap \text{OPT}$ with $i^*_b < j_t$, which is not included in the computed solution. The same problem occurs when the extension line of i_b intersects a rectangle $k \in S_r \cap \text{OPT}$.

5 The 4-Sided Problem

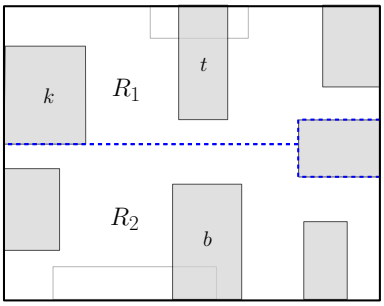
In this section, we consider the general 4-sided problem, in which the input rectangles can be attached to any side of the region $\mathcal{R}$. Let OPT be an optimal solution, and let t, b, ℓ, and r be the tallest boundary rectangles in OPT attached to the left, top, right and bottom sides of $\mathcal{R}$, respectively. If at least one of these rectangles is not present in OPT, then the problem reduces to a 3-sided case, whose solution can be computed using Theorem 5. Otherwise, there are two possible structures for the optimal solution as follows.

- **Disjoint Structure** — This structure occurs when either $b_t \leqslant t_b$ or $\ell_r \leqslant r_\ell$. We assume, w.l.o.g., that the first condition holds here. There are two possible cases:

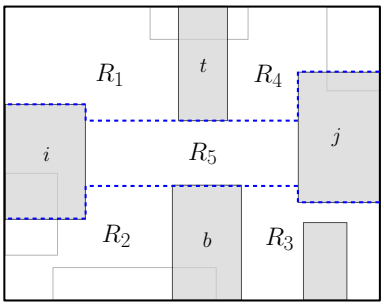
  – Case 1: There is a rectangle $k \in (S_\ell \cup S_r) \cap \text{OPT}$ whose top (or bottom) side lies in the range $[b_t, t_b]$. We draw two borders by extending k's top (or bottom) as shown in Figure 8a. Note that these two borders may be the same. Since t and b are the tallest rectangles in $S_t \cap \text{OPT}$ and $S_b \cap \text{OPT}$, respectively, no rectangle in OPT can cross these two borders. Therefore, the region $\mathcal{R}$ is divided by these two lines into two subregions R_1 and R_2. Now rectangles in R_1 and R_2 form two 3-sided subproblems whose optimal solution can be obtained using Theorem 5. A similar case applies when no rectangle of OPT intersects the range $[b_t, t_b]$.

  – Case 2: The boundary of $\mathcal{R}$ between b_t and t_b is covered by exactly two rectangles i and j from OPT (see Figure 8b). The four rectangles t, b, i, and j divide the region $\mathcal{R}$ into five subregions R_1,R_2,R_3,R_4 and R_5, as shown in Figure 8b. R_5 has no rectangle other than i, j. The other four regions form corner problems, each of which specified by two rectangles. Thus there are $O(n^2)$ such corner subproblems to consider.

- **Wheel Structure** — The wheel structure occurs when $b_t > t_b$ and $\ell_r > r_\ell$ (see Figure 9). Assume, w.l.o.g., that b is to the left of t. The decomposition in this case is the same as [7]. Let t' be



(a) Case 1



(b) Case 2

Figure 8: Two 4-sided disjoint structures.

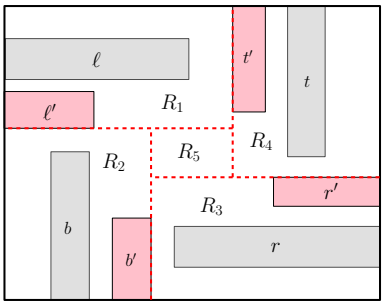


Figure 9: The 4-sided wheel structure.

the leftmost rectangle in S_t not intersecting ℓ. We similarly define rectangles b', r', and ℓ'. Now we obtain five regions R_1 to R_5 by extending t'_ℓ, b'_r, r'_t and ℓ'_b, as shown in Figure 9. R_5 contains no rectangle, and the other four regions are all corner cases. All of these corner problems are specified by two rectangles, and therefore, the number of such subproblems is $O(n^2)$.

Theorem 6 *Problem 1 can be solved in $O(n^4)$ time.*

Proof. The optimal solution is obtained by taking the maximum of all the possible cases described above. If one of the four rectangles t, b, r, and ℓ is not present in OPT, the problem reduces to a 3-sided subproblem. We set each of S_ℓ, S_t, S_r and S_b to $\emptyset$, and solve four 3-sided problems to consider this case. This takes $O(n^4)$ time by Theorem 5.

For Case 1 of the disjoint structure, we consider each possible border as a separating border and find the optimum solution for its top and bottom region. This can be done in $O(n^2)$ time after $O(n^4)$ preprocessing time by Theorem 5. For Case 2, we first preprocess, for each of possible borders, the optimal solution to $O(n^2)$ corner subproblems. This takes $O(n^4)$ time by Theorem 4. Then, considering every four rectangles as t, b, r, and ℓ, we can compute $\text{Corner}(R_1) + \text{Corner}(R_2) + \text{Corner}(R_3) + \text{Corner}(R_4)$ in $O(1)$ time. Therefore, we need $O(n^4)$ time to compute all possible configuration for this case.

The wheel structure can be handled similarly, by considering every four rectangles as t', b', r', and ℓ', and computing $\text{Corner}(R_1) + \text{Corner}(R_2) + \text{Corner}(R_3) + \text{Corner}(R_4)$ in $O(1)$ time. The total time needed is therefore $O(n^4)$. □

6 Approximation Algorithms

In practical application, where the number of input rectangles is high, one may desire to obtain a faster solution at the expense of relaxing the optimality condition.

Theorem 7 *A 2-approximation to Problem 1 can be computed in $O(n^2)$ time.*

Proof. Given an instance of Problem 1, we construct two opposite subproblem P_1 and P_2, consisting of the rectangles in $S_\ell \cup S_r$ and $S_t \cup S_b$, respectively. We then compute optimal solutions O_1 and O_2 to P_1 and P_2, respectively, and return the maximum of the two. To justify the approximation factor, let OPT be an optimal solution to the main problem. Then it is easy to see that $w(\text{OPT}) \leqslant w(O_1) + w(O_2)$, since the restriction of OPT to $S_\ell \cup S_r$ (resp., $S_t \cup S_b$) is a feasible solution to P_1 (resp., P_2). Therefore, $\max\{w(O_1), w(O_2)\} \geqslant \frac{1}{2} w(\text{OPT})$. The whole procedure takes $O(n^2)$ time by Theorem 2. □

Remark. Using a similar strategy, we can obtain a 4/3-approximation in $O(n^3)$ time. Details will be provided in the full version.

7 Conclusions

In this paper, we presented an $O(n^4)$-time algorithm for the maximum disjoint set of boundary rectangles, improving upon the previous algorithm of Kong *et al.* [7] by a factor of $O(n^2)$. It remains open whether the running time of the algorithm can be further improved. Finding better approximation algorithms, with improved approximation factors and/or running time is another interesting problem.

References

[1] A. Adamaszek and A. Wiese. Approximation schemes for maximum weight independent set of rectangles. In *Annu. IEEE Sympos. Found. Comput. Sci.*, FOCS '13, pages 400–409, 2013.

[2] S. Assadi, E. Emamjomeh-Zadeh, S. Yazdanbod, and H. Zarrabi-Zadeh. On the rectangle escape problem. In *Proc. 25th Canad. Conf. Computat. Geom.*, CCCG '13, pages 235–240, 2013.

[3] T. M. Chan. Polynomial-time approximation schemes for packing and piercing fat objects. *J. Algorithms*, 46(2):178–189, 2003.

[4] T. M. Chan and S. Har-Peled. Approximation algorithms for maximum independent set of pseudo-disks. *Discrete Comput. Geom.*, 48(2):373–392, 2012.

[5] T. Erlebach, K. Jansen, and E. Seidel. Polynomial-time approximation schemes for geometric intersection graphs. *SIAM J. Comput.*, 34(6):1302–1323, 2005.

[6] R. J. Fowler, M. S. Paterson, and S. L. Tanimoto. Optimal packing and covering in the plane are NP-complete. *Inform. Process. Lett.*, 12(3):133–137, 1981.

[7] H. Kong, Q. Ma, T. Yan, and M. D. F. Wong. An optimal algorithm for finding disjoint rectangles and its application to PCB routing. In *Proc. 47th ACM/EDAC/IEEE Design Automation Conf.*, DAC '10, pages 212–217, 2010.

[8] Q. Ma, H. Kong, M. D. F. Wong, and E. F. Y. Young. A provably good approximation algorithm for rectangle escape problem with application to PCB routing. In *Proc. 16th Asia South Pacific Design Automation Conf.*, ASPDAC '11, pages 843–848, 2011.