Finding Maximum Disjoint Set of Boundary Rectangles with Application to PCB Routing

Amirmahdi Ahmadinejad, Hamid Zarrabi-Zadeh

***Abstract*—Motivated by the bus escape routing problem in printed circuit boards (PCBs), we study the following optimization problem: given a set of rectangles attached to the boundary of a rectangular region, find a subset of non-overlapping rectangles with maximum total weight. We present an efficient algorithm that solves this problem optimally in $O(n^4)$ time, where n is the number of rectangles in the input instance. This improves over the best previous $O(n^6)$-time algorithm available for the problem. We also present two efficient approximation algorithms for the problem that find near-optimal solutions with guaranteed approximation factors. The first algorithm finds a 2-approximate solution in $O(n^2)$ time, and the second one computes a $4/3$-approximation in $O(n^3)$ time. The experimental results demonstrate the efficiency of both our exact and approximation algorithms.**

***Index Terms*—Escape routing, maximum independent set, approximation algorithm, printed circuit board (PCB) routing.**

I. Introduction

The problem of finding a *maximum disjoint set of boundary rectangles* (MDBR) is defined as follows: given an axis-parallel rectangular region $\mathcal{R}$, and a set of n boundary rectangles inside $\mathcal{R}$, where each *boundary rectangle* is attached to one of the four sides of $\mathcal{R}$, find a disjoint (non-overlapping) subset of boundary rectangles with maximum total weight. Figure 1 illustrates an instance of the problem, in which all boundary rectangles are assumed to have equal weight.

The MDBR problem is motivated by the *escape routing* problem in printed circuit boards (PCBs), whose aim is to route nets from their pins to the component boundaries. Most solutions available for this problem in the literature [4], [6], [9], [14], [18], [21], [22] are net-centric, in the sense that they focus on routing the nets, without considering a top-level bus structure. However, in practice, nets are usually grouped into buses, and the nets from the same bus are preferred to be routed together [11]–[13], [15], [16], [19], [20]. The routing of a bus in this setting is obtained by projecting the bounding box of the bus pin cluster onto one of the four boundaries of the component. Figure 2 illustrates an example. As noted in [11], it is easy to see that the bus escape routing problem can be reduced to the MDBR problem by considering for each bus four boundary rectangles obtained from projecting the bounding box of the bus pin cluster onto the four boundaries of the component.

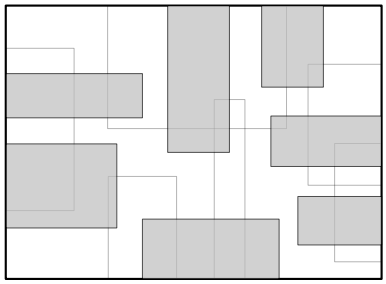


Fig. 1: An instance of the MDBR problem. A maximum disjoint subset is shown in gray.

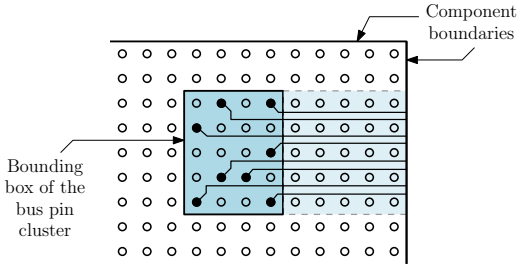


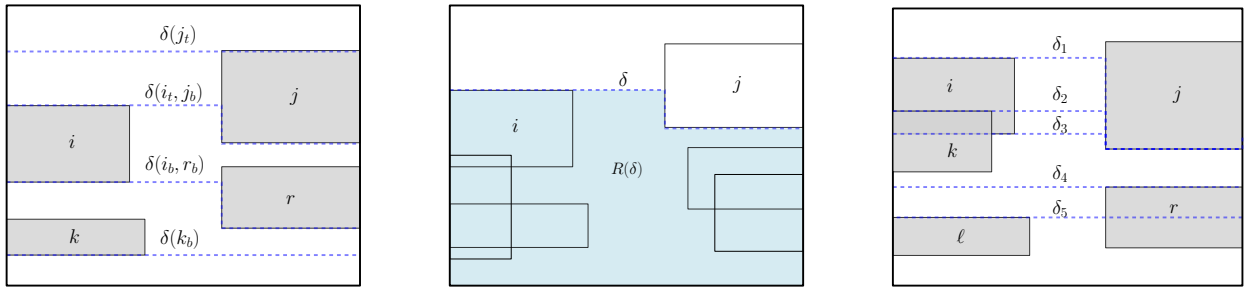
Fig. 2: A boundary rectangle obtained from projecting the bounding box of the bus pin cluster onto the component boundary.

The MDBR problem is a special case of the *maximum independent set* (MIS) problem, which is well-known to be NP-hard, and is also hard to approximate to within a factor of $n^{1-\varepsilon}$ for any constant $\varepsilon > 0$ [8]. The MIS problem for a set of rectangles in the plane is still NP-hard [7], even if all rectangles are axis-aligned squares of the same size [10]. The current best algorithm for the MIS of axis-parallel rectangles has an approximation factor of $O(\log n / \log\log n)$ for the weighted case [3], and an approximation factor of $O(\log\log n)$ for the unweighted case [1]. When all rectangles are fat (i.e., have bounded aspect ratio), the problem can be approximated to within a factor of $1+\varepsilon$, for any $\varepsilon > 0$ [2], [5]. For the special case where all rectangles are attached to the boundary of a bounding region (i.e., the MDBR problem), the problem is recently shown to be solvable in polynomial time [11].

In this paper, we revisit the MDBR problem, and present an efficient algorithm that solves the problem optimally in $O(n^4)$ time. Our algorithm improves over the best previous $O(n^6)$-time algorithm for the problem presented by Kong *et al.* [11].

The authors are with the Department of Computer Engineering, Sharif University of Technology, Tehran, 14588 89694 Iran (e-mail: am_ahmadinejad@ce.sharif.edu, zarrabi@sharif.edu).

This work is accomplished as the first author's master's thesis at Sharif University of Technology. He is now pursuing his PhD at the Management Science and Engineering Department, Stanford University, Stanford, California 94305 USA.



(a) (b) (c)

Fig. 3: Illustration of borders. (a) Four different types of borders. (b) The region $R(\delta)$ for a border $\delta = \delta(i_t, j_b)$. (c) A sequence of borders $\delta_1 = \delta(i_t, j_b)$, $\delta_2 = \delta(k_t, j_b)$, $\delta_3 = \delta(i_b, j_b)$, $\delta_4 = \delta(r_t)$, and $\delta_5 = \delta(\ell_t)$, where $\delta_{i+1} = \text{next}(\delta_i)$, for $1 \leqslant i \leqslant 4$.

Our algorithm also improves an $O(n^5)$-time algorithm by Piliposyan [17] that solves a special case of the MDBR problem in which all input boundary rectangles attached to each of the four sides of the bounding region are assumed to be disjoint.

The main idea of our algorithm is to exploit optimal substructures of the problem, and use dynamic programming to solve the problem optimally using optimal solutions to the smaller subproblems. In particular, similar to [11], we first solve some restricted k-sided subproblems, in which all boundary rectangles are attached to only k sides of the bounding region. 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 [11]. Our 2-sided 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 [11]. For the 3-sided case, we use a novel decomposition, enabling us to solve all possible 3-sided subproblems in $O(n^4)$ total time, improving over the $O(n^6)$ time required by Kong *et al.*'s algorithm. 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.

When optimality can be compromised a little, we show that near-optimal solutions can be obtained much faster. In particular, we present two efficient approximation algorithms that achieve approximation factors of 2 and $4/3$ in $O(n^2)$ and $O(n^3)$ time, respectively. The experimental results show that our algorithms can find optimal and near-optimal solutions for large test cases in few seconds.

II. Preliminaries

We first introduce notations used throughout this paper. 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-aligned. 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 sides of rectangles to properly order those 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. (See Figure 3a.)
- 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 3a.)
- If $j_b \leqslant i_b \leqslant j_t$, the border $\delta(i_b, j_b)$ is obtained just like $\delta(i_t, j_b)$, with the only exception that we first extend 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, we denote by $R(\delta)$ the subregion of $\mathcal{R}$ bounded from above by δ. An example is shown in Figure 3b.

For each border δ, we denote by $\text{top}(\delta)$ and $\text{bottom}(\delta)$ the y coordinates of the top and the bottom horizontal segments of δ, respectively. If δ consists of only one segment, then $\text{top}(\delta) = \text{bottom}(\delta)$ by definition. For each border δ, we define $\text{next}(\delta)$ as follows. If there is a border δ' with $\text{bottom}(\delta') = \text{bottom}(\delta)$ and $\text{top}(\delta') < \text{top}(\delta)$, we define $\text{next}(\delta)$ to be such border with maximum $\text{top}(\delta')$. Otherwise, we define $\text{next}(\delta)$ to be a border $\delta' = \delta(i_t)$ with maximum i_t, which lies completely below $\text{bottom}(\delta)$. (See Figure 3c.)

Lemma 1: For each border δ, $\text{next}(\delta)$ 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. Furthermore, we define $n(i_b)$ (resp., $n(i_t)$) to be the top side of the highest rectangle which is below i_b (resp., below i_t). For example, in Figure 3c, $L_j = \langle i_t, k_t, i_b \rangle$ in order from left to right, $n(j_b) = r_t$, and $n(r_t) = \ell_t$.

To find $\text{next}(\delta)$ for a border δ we do the following. 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 $\text{next}(\delta)$. Otherwise, $\text{next}(\delta) = \delta(n(j_b))$, since there is no border after δ in L_j, and the highest border below δ should be completely below $\text{bottom}(\delta) = j_b$. If $\delta = \delta(i_t)$, then $\text{next}(\delta)$ is equal to $\delta(n(i_t))$.

Now we show how to compute L_i and $n(i_b)$ (resp., $n(i_t)$) 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. □

III. The 2-Sided Problem

To solve the MDBR problem in its general form (i.e., the 4-sided case), we decompose it into a set of 3-sided and 2-sided subproblems, as described in Section V. The 3-sided case itself reduces to solving a set of 2-sided subproblems. In this section, 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.

A. 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 $\text{Opposite}(\delta)$ the weight of an optimal solution lying completely inside $R(\delta)$. The answer to the opposite case is equal to $\text{Opposite}(\delta(k_t))$, where k is the topmost rectangle in S.

Theorem 1: For all borders δ, $\text{Opposite}(\delta)$ can be computed in $O(n^2)$ total time.

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

- $\delta = \delta(i_t, j_b)$: In this case, either $i \in \text{OPT}$ or not. If $i \in \text{OPT}$, then $\text{Opposite}(\delta) = w_i + \text{Opposite}(\delta(i_b, j_b))$. Otherwise, $\text{Opposite}(\delta) = \text{Opposite}(\text{next}(\delta))$.
- $\delta = \delta(i_b, j_b)$: In this case, $\text{Opposite}(\delta)$ is simply equal to $\text{Opposite}(\text{next}(\delta))$, since there is no valid rectangle between δ and $\text{next}(\delta)$.
- $\delta = \delta(i_t)$: Again, either $i \in \text{OPT}$ or not. If $i \in \text{OPT}$, 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 $\text{Opposite}(\delta) = w_i + \text{Opposite}(\delta(k_t, i_b))$, otherwise, $\text{Opposite}(\delta) = w_i + \text{Opposite}(\delta(i_b))$. If $i \notin \text{OPT}$, then $\text{Opposite}(\delta) = \text{Opposite}(\text{next}(\delta))$.
- $\delta = \delta(i_b)$: In this case, $\text{Opposite}(\delta)$ is simply equal to $\text{Opposite}(\text{next}(\delta))$.

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, $\text{Opposite}(\delta)$ can be computed using a constant number of previously-computed borders in a dynamic programming fashion. Since the number of borders is $O(n^2)$, $\text{Opposite}(\delta)$ for all borders δ can be computed in $O(n^2)$ total time. □

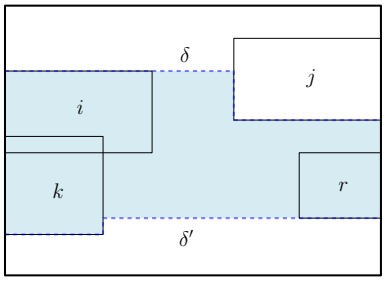


Fig. 4: The region $R(\delta, \delta')$.

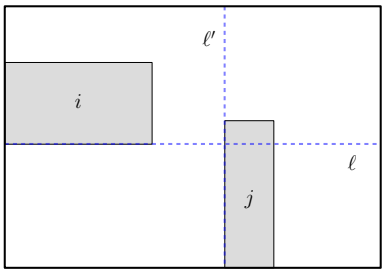


Fig. 5: Illustrating the splitter line.

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 4), and denote by $\text{Opposite}(\delta, \delta')$ the weight of an optimal solution for the rectangles lying inside $R(\delta, \delta')$.

Corollary 1: For all pairs of borders δ and δ', $\text{Opposite}(\delta, \delta')$ 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 1 to obtain the solutions to $\text{Opposite}(\delta, \delta')$, 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 $O(n^4)$. □

B. 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 2: Let OPT be an optimal solution to the corner case, i be the topmost rectangle in $\text{OPT} \cap S_\ell$, and j be the rightmost rectangle in $\text{OPT} \cap S_b$. 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 5).

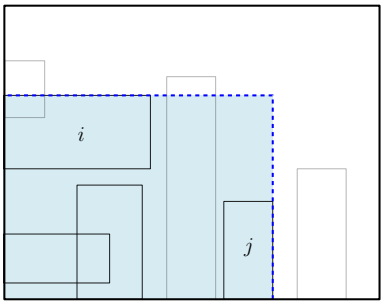


Fig. 6: The region corresponding to $\text{Corner}(i, j)$.

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 $k \in S_b \cap \text{OPT}$. Then k is either j or a rectangle completely to the left of j. Now any rectangle in $S_\ell \cap \text{OPT}$ 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 and j in S, we denote by $R(i_t, j_r)$ the region below i_t and to the left of j_r (see Figure 6). The regions $R(i_b, j_r)$, $R(i_t, j_\ell)$, and $R(i_b, j_\ell)$ are defined analogously. We use $\text{Corner}(i_t, j_r)$ to denote the weight of an optimal solution to the corner case composed of the rectangles within $R(i_t, j_r)$. $\text{Corner}(i_t, j_\ell)$, $\text{Corner}(i_b, j_r)$, and $\text{Corner}(i_b, j_\ell)$ are defined analogously. We set $\text{Corner}(i, j) = \text{Corner}(i_t, j_r)$.

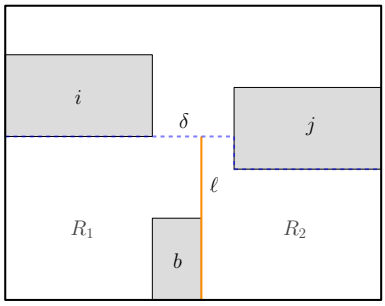
Theorem 2: For all pairs of rectangles $i, j \in S$, $\text{Corner}(i, j)$ can be computed in $O(n^2)$ total time.

Proof: Let $H = \{i_t : i \in S\} \cup \{i_b : i \in S\}$, and $V = \{j_\ell : j \in S\} \cup \{j_r : j \in S\}$. 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.

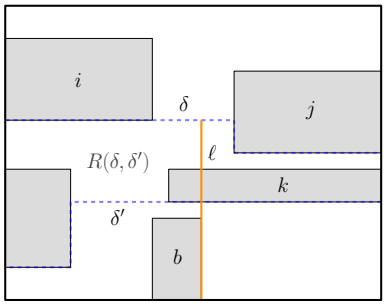
By Lemma 2, there is always a splitter that separates one rectangle of the optimal solution from the others. We consider two cases:

- 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, $\text{Corner}(i_t, j_r) = \max\{w_j + \text{Corner}(i_t, j_\ell), \text{Corner}(i_t, n_\ell(j_r))\}$. Moreover, $\text{Corner}(i_t, j_\ell) = \text{Corner}(i_t, n_\ell(j_\ell))$, since there is no valid rectangle between j_ℓ and $n_\ell(j_\ell)$. $\text{Corner}(i_b, j_r)$ and $\text{Corner}(i_b, j_\ell)$ 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, $\text{Corner}(i_t, j_r) = \max\{w_i + \text{Corner}(i_b, j_r), \text{Corner}(n_b(i_t), j_r)\}$. Moreover, $\text{Corner}(i_b, j_r) = \text{Corner}(n_b(i_b), j_r)$. $\text{Corner}(i_t, j_\ell)$ and $\text{Corner}(i_b, j_\ell)$ are computed similarly in this case.

By the above formulae, we need to check a constant number of subproblems in order to compute $\text{Corner}(i_x, j_z)$ 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 $\text{Corner}(i, j)$, for all pairs (i, j), in $O(n^2)$ total time. □



(a) Subcase 2.1



(b) Subcase 2.2

Fig. 7: Two subcases of the 3-sided problem.

IV. THE 3-SIDED PROBLEM

In the 3-sided problem, one of the sets $\{S_\ell, S_r, S_b, S_t\}$ 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 3: 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$(\delta) = \text{Opposite}(\delta)$.
- 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 7). 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 7a), each of which contains rectangles from only two sides. Therefore, 3-sided$(\delta) = \text{Corner}(R_1) + \text{Corner}(R_2)$.

– SUBCASE 2.2: At least one rectangle in OPT, say k, is intersecting ℓ (see Figure 7b). 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

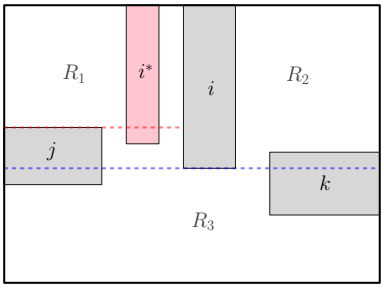


Fig. 8: An alternative partitioning for the 3-sided problem.

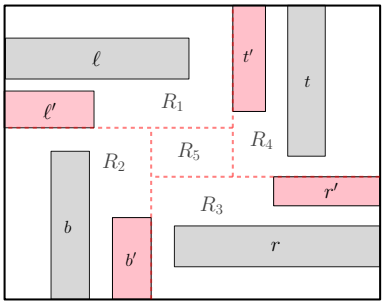


Fig. 9: The first case of the 4-sided problem.

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 1. In Case 2.1, we remove all rectangles not in $R(\delta)$, and use Theorem 2 to compute all corner subproblems in $R(\delta) \cap (S_\ell \cup S_b)$ 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. □

An alternative $O(n^3)$-time algorithm was proposed in [11] for the 3-sided case. However, that solution has a missing case, which is briefly mentioned below. 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 8). Partition 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 in [11] is 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 $k \in S_r \cap \text{OPT}$.

V. The 4-Sided Problem

In this section, we consider the general 4-sided problem, where the input rectangles can be attached to any side of the region $\mathcal{R}$. This leads to the main result of this paper.

Theorem 4: The MDBR problem can be solved exactly in $O(n^4)$ time.

Proof: 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 3. Otherwise, there are two possible cases:

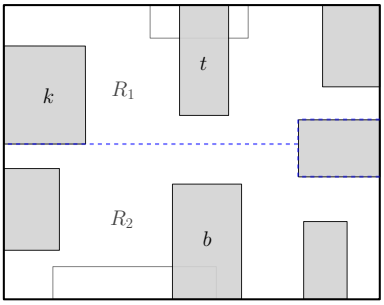
- Case 1: $b_t > t_b$ and $\ell_r > r_\ell$. Assume, w.l.o.g., that b is to the left of t (see Figure 9). Let t' be the leftmost rectangle in S_t on the right of ℓ. We define rectangles b', r', and ℓ' analogously. Now we obtain five regions R_1 to R_5 by extending t'_ℓ, b'_r, r'_t and ℓ'_b, as shown 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 $O(n^2)$.
- Case 2: $b_t \leqslant t_b$ or $\ell_r \leqslant r_\ell$. Assume, w.l.o.g., that $b_t \leqslant t_b$ (see Figure 10). Two subcases arise:

– Subcase 2.1: 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 10a. 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 3. A similar case applies when no rectangle of OPT intersects the range $[b_t, t_b]$.

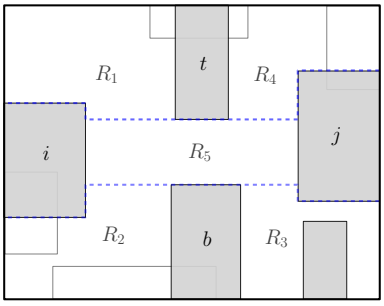
– Subcase 2.2: The boundary of $\mathcal{R}$ between b_t and t_b is covered by exactly two rectangles i and j from OPT (see Figure 10b). The four rectangles t, b, i, and j divide the region $\mathcal{R}$ into five subregions R_1,R_2,R_3,R_4 and R_5, as shown in Figure 10b. R_5 has no rectangle other 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.

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 3.

For Case 1, we first find the optimal solutions to corner subproblems in $O(n^2)$ time using Theorem 2. We then consider every four rectangles as t', b', r', and ℓ', and compute Corner(R_1) + Corner(R_2) + Corner(R_3) + Corner(R_4) in



(a) Subcase 2.1



(b) Subcase 2.2

Fig. 10: The second case of the 4-sided problem.

$O(1)$ time. This requires $O(n^4)$ total time. For Subcase 2.1, 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 3. For Subcase 2.2, 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 2. Then, considering every four rectangles as t, b, r, and ℓ, we can compute $\mathrm{Corner}(R_1)+\mathrm{Corner}(R_2)+\mathrm{Corner}(R_3)+\mathrm{Corner}(R_4)$ in $O(1)$ time. Therefore, we need $O(n^4)$ time to compute all possible configuration for this case. The total time needed for the whole algorithm is therefore $O(n^4)$. □

VI. Approximation Algorithms

In practical applications, where the number of input rectangles is huge, it is usually preferable to obtain a faster solution at the expense of relaxing the optimality condition. In this section, we provide two approximation algorithms for the MDBR problem, with guaranteed approximation factors of 2 and $4/3$, respectively, which are much faster than the exact algorithm.

The pseudocode of our approximation algorithms are provided in Algorithms 1 and 2, respectively. In these two algorithms, we denote by $\mathrm{Opposite}(\cdot)$, $3\text{-sided}(\cdot)$, and $1\text{-sided}(\cdot)$ the optimal solutions to the corresponding opposite, 3-sided, and 1-sided subproblems, respectively. Moreover, for two sets of rectangles S and T, we denote by $S \triangle T$ the set S from which all rectangles intersecting any rectangle in T are removed.

Theorem 5: Algorithm 1 computes a 2-approximation to the MDBR problem in $O(n^2)$ time.

Algorithm 1 2-Approximation(S)

1: let $S_1 = S_\ell \cup S_r$ and $S_2 = S_t \cup S_b$
2: **for** $i \in \{1,2\}$ **do**
3: $\quad T_i = \mathrm{Opposite}(S \setminus S_i)$
4: $\quad O_i = T_i \cup \mathrm{Opposite}(S_i \triangle T_i)$
5: **return** the maximum-weight set among $\{O_1, O_2\}$

Algorithm 2 $4/3$-Approximation(S)

1: **for** $i \in \{\ell, r, t, b\}$ **do**
2: $\quad T_i = 3\text{-sided}(S \setminus S_i)$
3: $\quad O_i = T_i \cup 1\text{-sided}(S_i \triangle T_i)$
4: **return** the maximum-weight set among $\{O_\ell, O_r, O_t, O_b\}$

Proof: Let OPT be an optimal solution to the MDBR problem. Consider two opposite instances $Q_i = S \setminus S_i$, $i = 1, 2$. We have $w(\mathrm{OPT}) = w(\mathrm{OPT} \cap Q_1) + w(\mathrm{OPT} \cap Q_2) \leqslant w(T_1) + w(T_2) \leqslant w(O_1) + w(O_2)$, where the first inequality holds because the restriction of OPT to Q_1 (resp., Q_2) is a feasible solution for the opposite case Q_1 (resp., Q_2). Therefore, $\max\{w(O_1), w(O_2)\} \geqslant \frac{1}{2} w(\mathrm{OPT})$, which proves an approximation factor of 2 for the algorithm. The total runtime of the algorithm is $O(n^2)$ by Theorem 1. □

Algorithm 2 achieves a better approximation factor by solving four 3-sided subproblems. Recall that by Theorem 3, the 3-sided problem can be solved for all borders in $O(n^4)$ time. However, this running time is not desirable here, as it matches the runtime of the exact algorithm. However, the observation here is that we do not need to solve the 3-sided problem for "all" borders, as opposed to what we needed for the general 4-sided problem. Therefore, we first present a new algorithm that solves the 3-sided problem in $O(n^3)$ time.

As an ingredient, we need to define a new type of the corner case. Suppose that all input rectangles are in $S_\ell \cup S_b$. For each rectangle $k \in S_\ell$, let Corner_k denote a corner case subproblem that is restricted to contain rectangle k in its solution. We define $\mathrm{Corner}_k(i_t, j_r)$, $\mathrm{Corner}_k(i_t, j_\ell)$, $\mathrm{Corner}_k(i_b, j_r)$, and $\mathrm{Corner}_k(i_b, j_\ell)$, similar to the ordinary corner case in Section III-B.

Lemma 3: For all rectangles $k \in S_\ell$ and $i, j \in S_\ell \cup S_b$, $\mathrm{Corner}_k(i_x, j_y)$ for all $x \in \{b, t\}$ and $y \in \{\ell, r\}$ can be computed in $O(n^3)$ time.

Proof: Since rectangle k must be included in the optimal solution, none of the rectangles intersecting k can be simultaneously present in the optimal solution. Therefore, to solve Corner_k subproblem for a fixed k, we simply remove rectangle k and all rectangles intersecting it to obtain an ordinary corner case subproblem, which can be then solved in $O(n^2)$ time using Theorem 2. Therefore, Corner_k, for all rectangles k, can be solved in $O(n^3)$ total time. □

Theorem 6: The 3-sided problem can be solved in $O(n^3)$ time.

Proof: Consider an instance of the 3-sided problem in which,

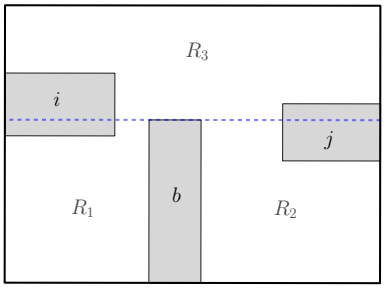


Fig. 11: Region decomposition of the 3-sided problem.

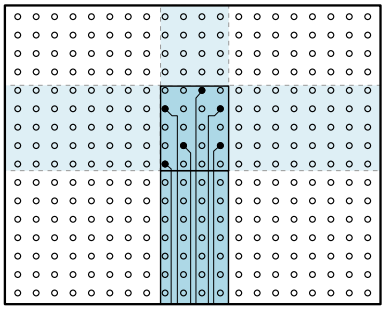


Fig. 12: The four possible directions for routing a bus. The bus is routed through the shaded boundary rectangle.

w.l.o.g., $S_t = \emptyset$. Let OPT be an optimal solution to the problem. If there is no rectangle from S_b in OPT, then the problem reduces to a 2-sided case, which can be solved in $O(n^2)$ time by Theorem 1. Otherwise, let b be the tallest rectangle in $S_b \cap \mathrm{OPT}$. Suppose that the extension of b_t intersects two rectangles $i \in S_\ell \cap \mathrm{OPT}$ and $j \in S_r \cap \mathrm{OPT}$ (see Figure 11). The region $\mathcal{R}$ is then divided by the extension of b_t into three subregions R_1, R_2, and R_3 as shown in Figure 11. R_1 is a corner case subregion, which is specified by $\mathrm{Corner}_i(b_t, b_\ell)$. If no rectangle from $S_\ell \cap \mathrm{OPT}$ intersects the extension of b_t, then R_1 is simply a corner case specified by $\mathrm{Corner}(b_t, b_\ell)$. Similarly, R_2 specifies a corner case subproblem. By Lemma 3, after $O(n^3)$ preprocessing time, we can compute optimal solutions to R_1 and R_2 in $O(1)$ time. Moreover, no rectangle in $S_b \cap \mathrm{OPT}$ intersects the region R_3, and therefore, it forms an opposite case subproblem. Hence, by Theorem 1, after $O(n^2)$ preprocessing time, we can compute an optimal solution to R_3 in $O(1)$ time. Since there are $O(n^3)$ possible triples of rectangles (b, i, j), it takes $O(n^3)$ time to enumerate all of them and compute the optimal solution. □

Theorem 7: Algorithm 2 computes a $4/3$-approximation to the MDBR problem in $O(n^3)$ time.

Proof: Let OPT be an optimal solution to the MDBR problem. Consider four 3-sided instances $Q_i = S \setminus S_i$, for $i \in \{\ell, r, t, b\}$. Since each rectangle belongs to three of the substes Q_i, we have $3 \times w(\mathrm{OPT}) = \sum_i w(\mathrm{OPT} \cap Q_i) \leqslant \sum_i w(T_i) \leqslant \sum_i w(O_i) \leqslant 4 \times \max_i \{w(O_i)\}$, where the sum and max is taken over the range $i \in \{\ell, r, t, b\}$. Therefore, the output of the algorithm is within factor $4/3$ of the optimal solution. Since the one-sided problem can be solved in $O(n \log n)$ time, and the 3-sided problem requires $O(n^3)$ time by Theorem 6, the total runtime of the algorithm is $O(n^3)$. □

VII. Experimental Results

To compare the performance of our algorithms against that of Kong *et al.* [11], we consider the same input setting as theirs. In particular, we consider instances of the PCB bus escape routing problem, where we are given n buses on a component board, and the goal is to maximize the total number (or the total weight) of the buses that can be routed to the boundary of the component without making any conflict. As noted in [11], this PCB bus scape routing problem can by reduced to the MDBR problem as follows. For each bus, we represent the bounding box of its pin cluster by a rectangle inside the component board. We then extend each rectangle toward four possible directions to obtain four boundary rectangles corresponding to the four possible routings of the bus (see Figure 12). Since all four boundary rectangles generated for a bus contain the rectangle corresponding to the pin cluster of that bus, the maximum disjoint set of boundary rectangles in the constructed instance is equal to the maximum number of buses that can be routed without crossing.

We have implemented the exact and approximation algorithms presented in this paper for the MDBR problem, and compared their performances against each other, and against the algorithm of Kong *et al.* [11]. The four algorithms implemented include our $O(n^4)$-time exact algorithm, which we denote by MDBR, the $O(n^6)$-time algorithm of Kong *et al.*, which we denote by MDS, and the two approximation algorithms presented in Section VI, which we denote by 2-APX and $4/3$-APX, respectively. We implemented the algorithms in C++[1], and applied them to ten industrial and machine-generated bus escape routing instances. We performed the experiments on a Linux machine with Intel Xeon 3.2GHz CPU and 8GB RAM. The experimental results are shown in Table I.

The first three columns of Table I show the test cases used in our experiments. The number of buses in our test cases ranges from 8 to 120, and the number of rectangles ranges from 32 to 480. Some of our test cases are illustrated in Figure 13.

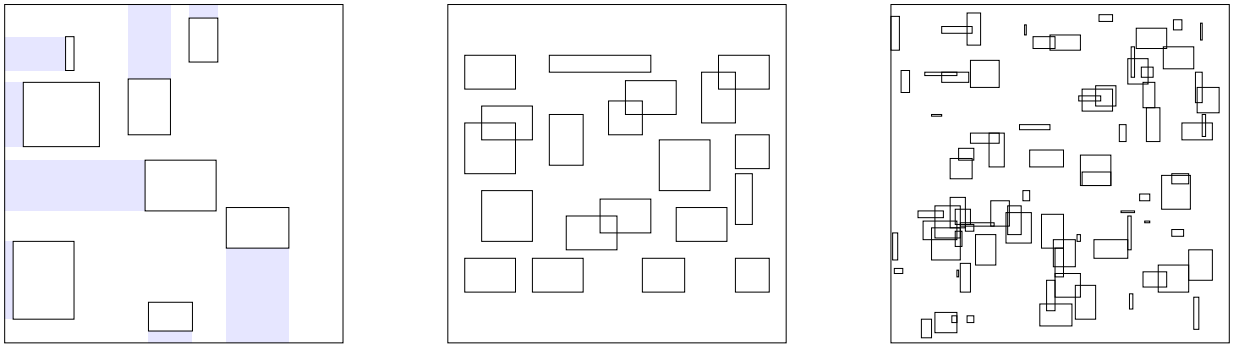
As shown in Table I, our MDBR algorithm is much faster than the MDS algorithm of Kong *et al.* (see the first two columns of section Runtime). For example, on a test case with 50 buses, our algorithm finds an optimal solution in only 6 seconds, while the MDS algorithm takes more than 5 minutes to finish. On a larger test case with 100 buses, our algorithm is more than 100 times faster than Kong *et al.*'s algorithm. The runtime of the algorithms are plotted in Figure 14 in logarithmic scale for a better comparison.

The solutions produced by our approximation algorithms are pretty close to the optimal ones (see the last four columns of section Solutions). In particular, the ratio of the optimal solution to the approximate one is very close to 1 in most of the cases. On average, the approximation ratios of the 2-APX and $4/3$-APX algorithms are 1.14 and 1.05, respectively. It means that in practice, the approximation ratios of these algorithms are much lower than the guaranteed theoretical upper bounds.

[1]The source codes of the four implemented algorithms are publicly available at: http://sharif.edu/~zarrabi/projects/mdbr/

TABLE I: Experimental Results

Test Case	# Buses	# Rect.	Solutions						Runtime (sec)			
			Optimal		Approximation							
			MDS	MDBR	$\frac{4}{3}$-APX	Ratio	2-APX	Ratio	MDS	MDBR	$\frac{4}{3}$-APX	2-APX
1	8	32	8	8	8	1	8	1	0.07	0.03	0.03	0.01
2	8	32	7	7	7	1	7	1	0.06	0.02	0.03	0.01
3	12	48	12	12	12	1	12	1	0.23	0.04	0.03	0.01
4	15	60	9	9	9	1	9	1	0.55	0.09	0.03	0.01
5	20	80	14	14	13	1.08	11	1.27	2.49	0.24	0.04	0.01
6	30	120	25	25	23	1.09	21	1.19	24.5	0.83	0.06	0.01
7	50	200	34	34	32	1.06	30	1.13	320	6.07	0.17	0.02
8	80	320	46	46	40	1.15	34	1.35	3302	41.5	0.50	0.03
9	100	400	48	48	45	1.07	40	1.20	11932	106	0.94	0.04
10	120	480	53	53	50	1.06	44	1.20	27049	247	1.69	0.06
Average Ratio			1.0		1.05		1.14		—			



(a) Test Case #1 with an optimal solution (b) Test Case #5 (Ref: [15]) (c) Test Case #8

Fig. 13: Examples of test cases used in our experiments.

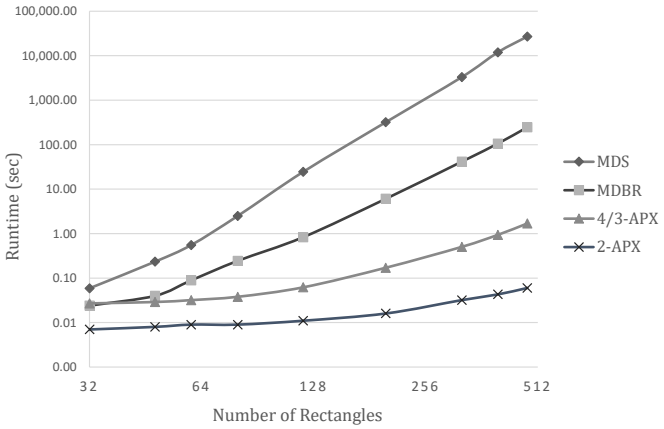


Fig. 14: The runtime of the algorithms presented in Table I. The vertical and horizontal axes of the plot are scaled logarithmically to the bases 10 and 2, respectively.

On the other hand, the approximation algorithms are much faster than the optimal algorithms. For example, on a test case with 120 buses, the 2-APX algorithm is 4,000 times faster than the MDBR algorithm, and 450,000 times faster than the MDS algorithm. Therefore, in real-world applications, if optimality can be compromised a little, we can get very good solutions efficiently, even for a large number of buses and rectangles.

VIII. Conclusions

In this paper, we presented an $O(n^4)$-time algorithm for computing a maximum disjoint set of boundary rectangles, improving over the best previous $O(n^6)$-time algorithm by Kong *et al.* [11]. We also presented two efficient approximation algorithms with approximation factors of 2 and $4/3$, respectively. It remains open whether the running time of the exact algorithm can be further improved. Finding better approximation algorithms, with improved approximation factors and/or running times is another interesting problem.

References

[1] P. Chalermsook and J. Chuzhoy. Maximum independent set of rectangles. In *Proceedings of the 20th ACM-SIAM Symposium on Discrete Algorithms*, pages 892–901, 2009.

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

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

[4] W.-T. Chan and F. Y. Chin. Efficient algorithms for finding the maximum number of disjoint paths in grids. *Journal of Algorithms*, 34(2):337–369, 2000.

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

[6] J.-W. Fang, I.-J. Lin, Y.-W. Chang, and J.-H. Wang. A network-flow-based RDL routing algorithm for flip-chip design. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 26(8):1417–1429, 2007.

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

[8] J. Håstad. Clique is hard to approximate within $n^{1-\varepsilon}$. *Acta Mathematica*, pages 627–636, 1996.

[9] J. Hershberger and S. Suri. Efficient breakout routing in printed circuit boards. In *Algorithms and Data Structures*, pages 462–471. 1997.

[10] H. Imai and T. Asano. Finding the connected components and a maximum clique of an intersection graph of rectangles in the plane. *Journal of Algorithms*, 4(4):310–323, 1983.

[11] 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 *Proceedings of the 47th ACM/EDAC/IEEE Design Automation Conference*, DAC '10, pages 212–217, 2010.

[12] H. Kong, T. Yan, and M. D. Wong. Automatic bus planner for dense pcbs. In *Proceedings of the 46th ACM/EDAC/IEEE Design Automation Conference*, pages 326–331, 2009.

[13] H. Kong, T. Yan, M. D. F. Wong, and M. M. Ozdal. Optimal bus sequencing for escape routing in dense PCBs. In *Proceedings of the 2007 IEEE/ACM International Conference on Computer-Aided Design*, ICCAD '07, pages 390–395, 2007.

[14] L. Luo and M. D. Wong. Ordered escape routing based on boolean satisfiability. In *Proceedings of the the 2008 Asia South Pacific Design Automation Conference*, ASP-DAC '08, pages 244–249, 2008.

[15] Q. Ma and M. 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.

[16] Q. Ma, E. Young, and M. D. F. Wong. An optimal algorithm for layer assignment of bus escape routing on PCBs. In *Proceedings of the 48th ACM/EDAC/IEEE Design Automation Conference*, DAC '11, pages 176–181, 2011.

[17] E. Piliposyan. A note on maximum weight independent set in outer-rectangle graphs. *Mathematical Problems of Computer Science*, 36:51–56, 2012.

[18] R. Wang, R. Shi, and C.-K. Cheng. Layer minimization of escape routing in area array packaging. In *Proceedings of the 2006 IEEE/ACM International Conference on Computer-Aided Design*, ICCAD '06, pages 815–819, 2006.

[19] P.-C. Wu, Q. Ma, and M. D. Wong. An ILP-based automatic bus planner for dense PCBs. In *Proceedings of the 18th Asia South Pacific Design Automation Conference*, ASPDAC '13, pages 181–186, 2013.

[20] T. Yan, H. Kong, and M. D. F. Wong. Optimal layer assignment for escape routing of buses. In *Proceedings of the 2009 IEEE/ACM International Conference on Computer-Aided Design*, ICCAD '09, pages 245–248, 2009.

[21] T. Yan and M. D. Wong. A correct network flow model for escape routing. In *Proceedings of the 46th ACM/EDAC/IEEE Design Automation Conference*, DAC '09, pages 332–335, 2009.

[22] M.-F. Yu and W. W.-M. Dai. Single-layer fanout routing and routability analysis for ball grid arrays. In *Proceedings of the 1995 IEEE/ACM International Conference on Computer-Aided Design*, ICCAD '95, pages 581–586, 1995.

Amirmahdi Ahmadinejad received the B.S. and the M.S. degrees in computer engineering from Sharif University of Technology, Tehran, Iran in 2012 and 2014, respectively. He is now a Ph.D. Student at Stanford University, Stanford, CA. He was a gold medal winner in the Iranian National Olympiad in Informatics in 2007, and ranked 2nd in the ACM International Collegiate Programming Contest, Asia Region, 2012. He was the coach of the Iranian team participating the International Olympiad in Informatics (IOI) in Taipei, Taiwan, 2014. His research interests include approximation algorithms, algorithmic game theory, computational geometry, and combinatorial optimization.

Hamid Zarrabi-Zadeh received the Ph.D. degree in computer science from the University of Waterloo, Waterloo, ON, Canada, in 2008. He was a Postdoctoral Fellow at Carleton University, Ottawa, ON, Canada, from 2009 to 2011. Since 2011, he has been with the Sharif University of Technology, where he is currently an Assistant Professor with the Department Computer Engineering. He has been a member of the International Scientific Committee (ISC) for the International Olympiad in Informatics (IOI) from 2014 to 2015. His current research interests include the design and analysis of algorithms, computational geometry, approximation and randomized algorithms, and algorithmic graph theory.