An Optimal Algorithm for Layer Assignment of Bus Escape Routing on PCBs [*]

Qiang Ma[†] Evangeline F. Y. Young[‡] Martin D. F. Wong[†]
[†]Department of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign
[‡]Department of Computer Science and Engineering, The Chinese University of Hong Kong
qiangma1@illinois.edu, fyyoung@cse.cuhk.edu.hk, mdfwong@illinois.edu

ABSTRACT

Bus escape routing is a critical problem in modern PCB design. Due to the huge pin count and high density of the pin array, it usually requires multiple layers to route the buses without any conflict. How to assign the escape routing of buses to different layers becomes an important issue. In addition, some buses are required to be assigned on consecutive layers, which adds more difficulties to the layer assignment problem. In this paper, we propose a branch-and-bound based algorithm that optimally solves the layer assignment problem of bus escape routing. Our algorithm guarantees to produce a feasible layer assignment of the buses with a minimum number of layers. We applied our algorithm on industrial data and the experimental results validate our approach.

Categories and Subject Descriptors

B.7.2 [**Integrated Circuits**]: Design Aids

General Terms

Algorithm, Theory

Keywords

Escape routing, Optimal layer assignment, Branch-and-bound

1. INTRODUCTION

As the scale of modern electronic systems becomes larger and larger, the design of printed circuit boards (PCB) becomes more and more complex. Nowadays, a dense PCB hosts tens of thousands of pins. Manually routing such a large number of pins, which is what the industry has been doing, is a tedious and error-prone job. Therefore, design automation of PCB routing becomes a necessity.

The PCB routing problem is usually divided into two phases: (1) escape routing, which is to route nets from pin terminals to component (MCM, memory, etc.) boundaries (see the solid lines in Fig.1), and (2) area routing, which is to route nets between component boundaries (see the dashed line in Fig.1). In this paper, we focus on the escape routing part. In practice, nets are usually grouped as buses and the nets from the same bus are expected to be routed together [3, 4, 9]. Directly applying net-centric escape routing algorithms may mix up the nets of different buses (as illustrated in Fig.1), which is not desired. It is observed from industrial manual routing solutions that the pins are usually escaped straight to the component boundaries with minimal detours. Therefore, the escape routes of all the pins of a bus are typically within one of its projection rectangles, which is obtained by projecting the bounding box of the bus pin cluster to one of the component boundaries [3, 4, 9]. Fig.2(a) demonstrates the concept of projection rectangle of a bus.

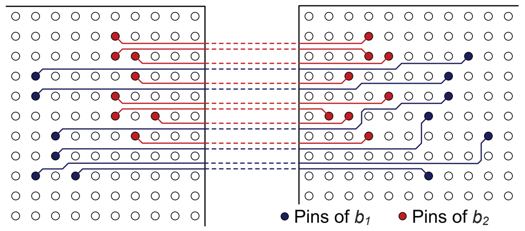
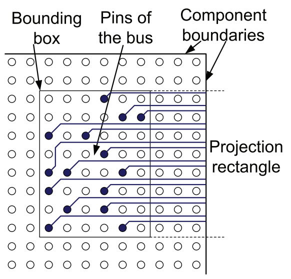


Figure 1: A sample net-centric escape routing solution for a problem with two buses. Nets of these two buses are mixed up.

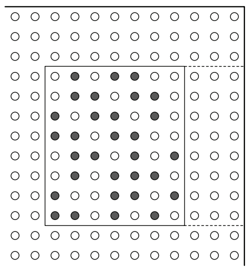
Due to the huge pin count and high density of the pin array, it usually requires multiple layers to escape the buses without any conflict. In fact, modern PCBs may contain more than 20 layers of routing [8]. The fabrication cost dramatically increases when more layers are needed, so we want to use as few layers as possible to accommodate all the buses. Therefore, how to assign the escape routing of buses to different layers becomes an important issue. Another practical issue we observed is that a large bus is usually decomposed into two or more smaller buses, and these smaller buses are required to be routed on consecutive layers, as illustrated in Fig.2. Fig.2(b) is a large bus, which contains too many pins to be routed within its projection rectangle on one layer, so it is decomposed into two smaller buses, as shown in Fig.2(c) and Fig.2(d). The two resultant buses are able to be routed on one layer each; however, they are required to be placed on consecutive layers. This adds more constraints to the bus layer assignment problem.

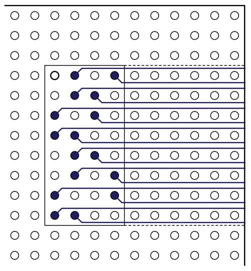
Several previous works addressed this layer assignment problem for bus escape routing. Kong *et al.* proposed in [4] an optimal algorithm to determine if all the buses can be assigned to a single layer. Yan *et al.* in [9] optimally solved the layer assignment for multiple layers. However, these two works are based on the assumption that all the buses are escaped along the same boundary of a component, which greatly restricted the solution quality. Let us take the two buses b_1 and b_2 in Fig.1 as an example. Two layers are needed if

[*]This work was partially supported by the National Science Foundation under grant CCF-1017516.



(a)


(b)


(c)

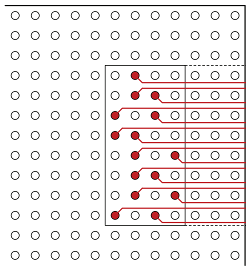

(d)

Figure 2: (a) illustrates the bus projection rectangle; (b), (c) and (d) show the bus decomposition issue: the large bus in (b) cannot be routed within its projection rectangle on one layer, so it is decomposed into two smaller buses ((c) and (d)), each of which is able to be routed on one layer. However, the two resultant buses are required to be routed on consecutive layers.

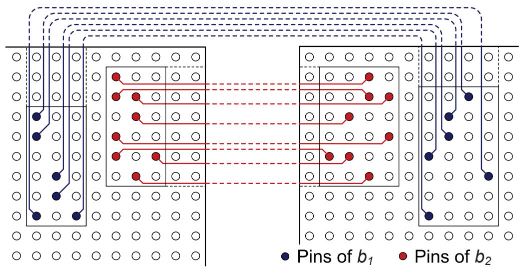


Figure 3: The two buses can be routed on one layer if bus b_1 is escaped to the top boundary.

both buses are routed to the right (left) boundary of the left (right) component, since otherwise they are mixed up. On the contrary, if bus b_1 is escaped to the top boundary, the two buses can be routed on one layer without mixing, as shown in Fig.3. Recently in [3], an optimal algorithm is presented for finding a maximum independent set of bus projection rectangles escaping from one component on a single layer. The general problem of optimal layer assignment of bus projection rectangles remains to be an open problem which is the subject of this paper.

Our contributions in this work are summarized as follows:

- We propose a branch-and-bound based algorithm, which optimally solves the problem of layer assignment for bus escape routing. The buses are allowed to be escaped to any of the component boundaries, and the constraints of consecutive layer assignment for buses generated after decomposition can also be properly taken care of during the searching.
- We observe through experiments that Integer Linear Programming (ILP) is very efficient for finding maximum independent set and minimum coloring of bus projection rectangles, which makes it practical to integrate ILP into the branch-and-bound searching for upper and lower bounds computation.
- An effective min-cut based algorithm is developed to preprocess the bus projection rectangles, so that the search tree is constructed in such a way that early pruning is more possible, which further improves the searching efficiency.

The rest of this paper is organized as follows: Section 2 defines the problem of layer assignment for bus escape routing; Section 3 presents our optimal algorithm for solving this problem; Section 4 reports the experimental results on some industrial data, and Section 5 concludes this paper.

2. PROBLEM FORMULATION

In our layer assignment problem of bus escape routing, we are given two components C_a and C_b, where C_a is located on the left and C_b is located on the right, facing each other. Other positions or orientations of the components can lead to similar discussions. We are also given a set B of n buses $\{b_1, b_2, \ldots, b_n\}$ connecting the two components, so we have n projection rectangles in each component representing the escape routing regions of the buses.

Some buses may have *internal conflict*, e.g., b_1 and b_2 in Fig.4(a). We say that two buses have *internal conflict* if and only if their projection rectangles overlaps in C_a or C_b. Some buses may have *external conflict*, e.g., b_1, b_2 and b_3 in Fig.4(b). We say that three or more buses have *external conflict* if and only if they cannot have a planar routing between the component boundaries (note that two buses can always be routed without external conflict). If a set of buses do not have either *internal conflict* or *external conflict*, we say that it is a *feasible set*. Fig.4(c) shows a *feasible set*.

Some buses come from the decomposition of a large bus, and they are required to be assigned to consecutive layers. We refer to this as a *consecutive assignment constraint*. In our problem input, we have a set of p consecutive assignment constraints, $\{G_1, G_2, \ldots, G_p\}$, where $G_i \subset B$, and the buses in G_i have to be assigned to consecutive layers, $\forall i \in \{1, 2, \ldots, p\}$.

Our objective is to find a feasible layer assignment of all the buses with a minimum number of layers, satisfying the consecutive assignment constraints. We now formally define the layer assignment problem of bus escape routing as follows:

DEFINITION 1. **Layer Assignment Problem of Bus Escape Routing** - *We are given a set B of n buses $\{b_1, b_2, \ldots, b_n\}$ connecting between two components C_a and C_b, as well as a set of p consecutive assignment constraints $\{G_1, G_2, \ldots, G_p\}$, where $G_i \subset B$, $\forall i \in \{1, 2, \ldots, p\}$. The objective is to assign all the buses to a minimum number of layers, such that the buses in each layer is a feasible set, and the consecutive assignment constraints are satisfied.*

3. OPTIMAL LAYER ASSIGNMENT

In this section, we present our optimal algorithm to solve the layer assignment problem. Our approach is based on branch-and-bound searching. We first give two algorithms which can efficiently compute an upper bound and a lower bound for the layer assignment problem, respectively; then we show how they are integrated into the searching process. Moreover, a min-cut based heuristic is developed to arrange the buses in a proper ordering in the search tree so that early pruning is encouraged, which helps improve the efficiency of the searching process.

3.1 Upper Bound Computation

We first ignore the consecutive assignment constraints, and we will show later how to take those constraints into account during the branch-and-bound searching. An upper bound can be computed by

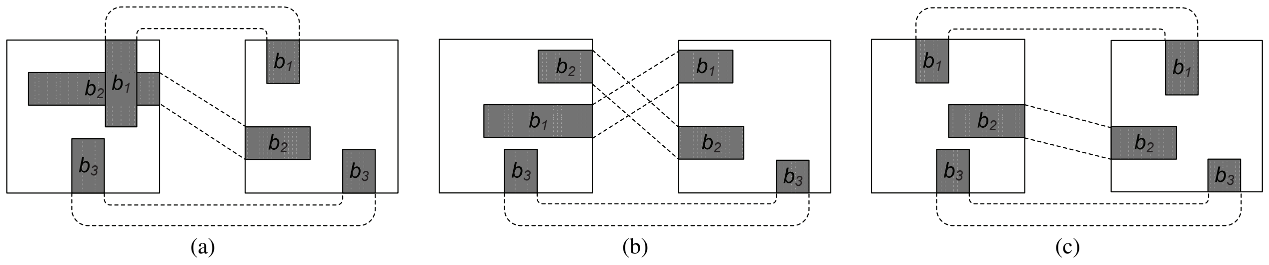


Figure 4: (a) Internal conflict, (b) External conflict, (c) A feasible set.

finding a feasible layer assignment, i.e., the buses assigned to each layer form a feasible set. To compute a feasible set, we first find a set of buses without internal conflict, then resolve the external conflict if there exists any. By iteratively finding a feasible set, a greedy method is developed to compute a feasible layer assignment.

3.1.1 *Resolving Internal Conflict*

Given a set of buses B, we first use ILP to compute a maximum set of buses S without internal conflict. Let $b_i \perp b_j$ denote that bus b_i and bus b_j have internal conflict, and let x_i be a 0-1 variable indicating whether bus $b_i \in B$ is selected into S or not. The following procedure ILP-MIS(B) returns the maximum set of buses S without internal conflict.

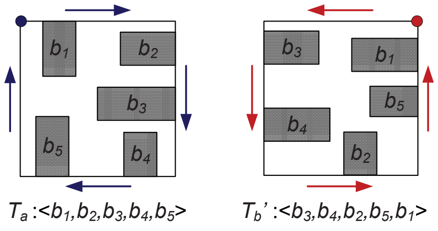
```
ALGORITHM ILP-MIS($B$):
  $S \leftarrow \emptyset$;
  Solve the following ILP:
      Maximize $\sum_{i:b_i \in B} x_i$
      Subject to
          $x_i + x_j \leq 1, \quad \text{if } b_i \perp b_j, \forall b_i, b_j \in B.$
          $x_i \in \{0,1\}, \forall b_i \in B.$
  for each bus $b_i \in B$ do
      if $x_i = 1$ then
          $S \leftarrow S \cup \{b_i\}$;
  return $S$;
```

We observe from experiments that this kind of ILP can be solved very efficiently by modern solvers. The paper [3] gave an $O(N^6)$ algorithm for computing the maximum independent set of bus projection rectangles within one component, where N is the total number of buses. We compare their algorithm with our ILP approach, and results show that our ILP approach can solve the problem much faster (see Section 4 for details).

3.1.2 *Resolving External Conflict*

The buses in S may still have external conflict, and we want to find from S a maximum subset S^* that has no external conflict. Fig.5 shows an example. We can obtain a sequence of the buses in S by traversing the boundaries of C_a clockwise (assume we start from its upper-left corner), and we call it a clockwise traversal of C_a (denoted by T_a). Similarly, we can obtain another sequence of the buses in S by traversing the boundaries of C_b counter-clockwise (assume we start from its upper-right corner), and we call it a counter-clockwise traversal of C_b (denoted by T'_b). T_a and T'_b in Fig.5 are respectively $< b_1, b_2, b_3, b_4, b_5 >$ and $< b_3, b_4, b_2, b_5, b_1 >$. The following theorem can be easily observed.

THEOREM 1. *A set of buses without internal conflict does not have external conflict if and only if T_a matches either T'_b or one of its rotations.*



b_1
b_2
b_3
b_5
b_4
b_3
b_1
b_5
b_4
b_2
T_a :<b_1,b_2,b_3,b_4,b_5>
T_b' :<b_3,b_4,b_2,b_5,b_1>

Figure 5: Buses without internal conflict have external conflict.

Note that we need to consider the rotations, since the traversal is actually a circular ordering and the starting position should not be fixed (if T_a is fixed, we need to check all the rotations of T'_b). For example, in Fig.5, the rotations of T'_b include $< b_4, b_2, b_5, b_1, b_3 >$, $< b_2, b_5, b_1, b_3, b_4 >$, and so on. We can assume T'_b itself is also one of its rotations. Based on Theorem 1, we can compute the Longest Common Subsequences (LCS) of T_a and each rotation of T'_b, and the one with the maximum length gives the maximum subset S^* without external conflict. In Fig.5, $T_a < b_1, b_2, b_3, b_4, b_5 >$ and one rotation of $T'_b < b_1, b_3, b_4, b_2, b_5 >$ give a LCS of maximum length, which is $< b_1, b_3, b_4, b_5 >$, so the maximum subset S^* without external conflict is $\{b_1, b_3, b_4, b_5\}$. Tang *et al.* showed in [7] that LCS can be computed in $O(N \log \log N)$ time where N is the length of the input sequences. We now have an algorithm to compute a feasible set, which is shown below.

```
ALGORITHM FINDFEASIBLESET($B$):
  $S \leftarrow$ ILP-MIS($B$);
  Get the traversals $T_a$ and $T'_b$ of the buses in $S$;
  Compute the LCSs of $T_a$ and each rotation of $T'_b$;
  Let $maxLCS$ be the one with maximum length;
  $S^* \leftarrow$ all the buses in $maxLCS$;
  return $S^*$;
```

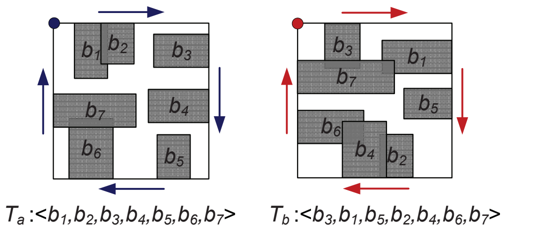
3.1.3 *Algorithm for Computing Upper Bound*

Iteratively finding a feasible set gives us an upper bound on the optimal number of layers:

```
ALGORITHM CALUPPERBOUND($B$):
  $upperbound \leftarrow 0$;
  while $B$ is not empty do
      $S^* \leftarrow$ FINDFEASIBLESET($B$);
      $B \leftarrow B \backslash S^*$;
      $upperbound \leftarrow upperbound + 1$;
  return $upperbound$;
```

3.2 Lower Bound Computation

Similarly, we first ignore the consecutive assignment constraints. Considering internal conflict and external conflict separately gives us two lower bounds on the optimal number of layers.



b_1
b_2
b_3
b_7
b_4
b_6
b_5
T_a :<b_1,b_2,b_3,b_4,b_5,b_6,b_7>
T_b :<b_3,b_1,b_5,b_2,b_4,b_6,b_7>

Figure 6: Clockwise traversal T_a and Clockwise traversal T_b.

If we only consider internal conflict, the layer assignment problem becomes a minimum coloring problem of all the bus projection rectangles such that the buses assigned the same color do not overlap in C_a and C_b. Theoretically, it is a hard problem[1]. Again, we use ILP to solve this problem. Let $\{x_{i1}, x_{i2}, \ldots x_{ik}\}$ be the set of 0-1 variables indicating whether bus b_i is assigned to layer 1, layer 2, ..., or layer k, respectively. The following procedure can be used to check if there is a way to assign the buses into k layers without internal conflict. A lower bound can be obtained by performing a binary search on k.

```
ALGORITHM ILP-COLOR($B$, $k$):
  Check if there exists a solution to the following constraints:
      $\sum_{l=1}^{k} x_{il} = 1, \forall b_i \in B.$
      $x_{il} + x_{jl} \leq 1, \forall l = 1, 2, \ldots, k, \quad \text{if } b_i \perp b_j, \forall b_i, b_j \in B.$
      $x_{i1}, x_{i2}, \ldots, x_{ik} \in \{0,1\}, \forall b_i \in B.$
  if Yes then return true;
  else return false;
```

If we only consider external conflict, another lower bound can be obtained. We first compute the clockwise traversal T_a of C_a and also the clockwise traversal T_b of C_b. Note that this time T_a and T_b contain all the buses of B. We then compute the LCSs of T_a and each rotation of T_b, and denote the one with maximum length by $maxLCS^R$, which means the Reversed LCS of maximum length. It is easy to figure out that any three buses in $maxLCS^R$ have external conflict, so they cannot be assigned to the same layer. As a result, $\lceil \frac{|maxLCS^R|}{2} \rceil$ is a lower bound of the number of layers required.

Fig.6 gives an example of seven buses. T_a is $< b_1, b_2, b_3, b_4, b_5, b_6, b_7 >$, T_b is $< b_3, b_1, b_5, b_2, b_4, b_6, b_7 >$, and $maxLCS^R$ is computed as $< b_1, b_2, b_4, b_6, b_7 >$. It is easy to check that any three of the five buses in $maxLCS^R$ cannot be assigned to the same layer. Therefore, $\lceil \frac{5}{2} \rceil = 3$ is a lower bound.

3.3 Branch-and-Bound Searching

In this section, we show how the layer assignment problem is solved optimally by using branch-and-bound searching. The consecutive assignment constraints can be properly taken care of during the searching. We describe the searching process by a search tree, in which an internal node represents a set of solutions while a leaf node represents either a single solution or an early termination of a searching branch.

3.3.1 Branching Rules

The root of the tree represents the set of all solutions (an optimal one exists for the original problem). We start from the root and

[1]The minimum coloring problem for bus projection rectangles is NP-complete even within one component, which can be proved by reduction from the Circular Arc Coloring problem [2].

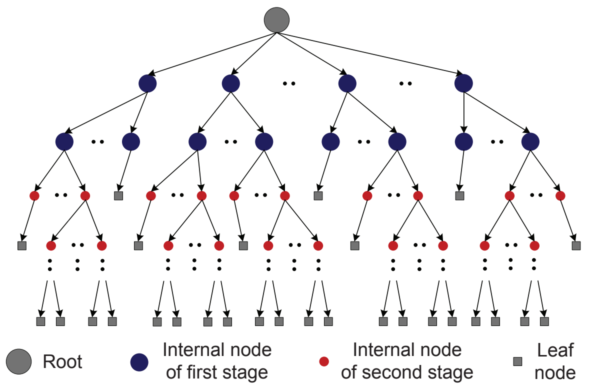


Figure 7: The Search Tree Structure.

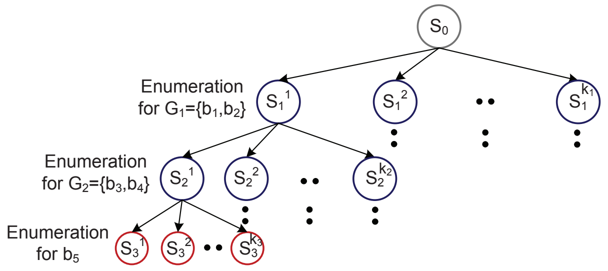


Figure 8: Sample of Branch-and-Bound Method.

visit different parts of the tree by assigning certain buses to specific layers. There are two stages of our search tree, as shown in Fig.7. The first stage deals with the groups of buses with consecutive assignment constraints, where the nodes on each level enumerate all possible layer assignments of one group; the second stage handles the other buses without constraints, where the nodes on each level enumerate all possible layer assignments of one bus. We assume that the ordering of the groups (buses) in which we branch out have already been determined, and we will show in Section 3.4 how this ordering can be found such that early pruning is more possible.

The consecutive assignment constraints are handled in the first stage. These constraints are guaranteed to be satisfied by a systematical enumeration. Let us consider the following example. We have a set B of 5 buses $\{b_1, b_2, b_3, b_4, b_5\}$ as well as two groups $G_1 = \{b_1, b_2\}$ and $G_2 = \{b_3, b_4\}$. Let us assume, w.l.o.g., that the ordering for branching is set to be G_1, G_2, b_5. The sample search tree is shown in Fig.8, with node S_0 as the root. In the first level, we enumerate all possible layer assignments for G_1, e.g., node S_1^1 represents the set of solutions with b_1 assigned to layer 1 and b_2 assigned to layer 2 (while node S_1^2 may represent the set of solutions with b_1 assigned to layer 2 and b_2 assigned to layer 1). At node S_1^1, we branch out to the next level, where the layer assignments for G_2 are enumerated, e.g., node S_2^1 represents the set of solutions with b_3 assigned to layer 1 and b_4 assigned to layer 2, in addition to the previous assignment of b_1 and b_2. Now, there are no more groups with consecutive assignment constraints, so the branching process can enter the second stage, and the layer assignment of bus b_5 is enumerated in the next level. In this way, we will branch out to many subproblems and search the whole solution space.

3.3.2 Pruning Rules

In the search tree, some nodes with their subtrees together will never contribute an optimal solution, and good pruning rules help to find such nodes to reduce runtime. If a node is infeasible, that is, internal conflict or external conflict exists for the pre-assigned

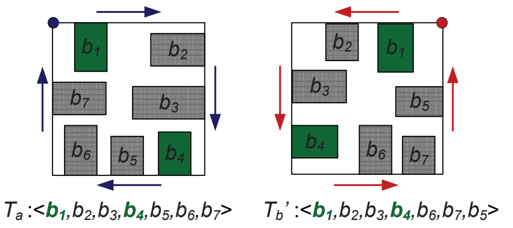

b_1
b_2
b_7
b_3
b_6
b_5
b_4
T_a :<b_1,b_2,b_3,b_4,b_5,b_6,b_7>
T_b' :<b_1,b_2,b_3,b_4,b_6,b_7,b_5>

Figure 9: Computing LCS when there are pre-assigned buses.

buses, it will be pruned. Besides, if a node's lower bound is no less than the global upper bound (denoted by O_{ub}), it will also be pruned. This can be checked by the procedure ILP-COLOR() in Section 3.2. Note that additional constraints for the pre-assigned buses need to be added into the ILP of ILP-COLOR() for this step. For example, at node S_1^1 of Fig.8, bus b_1 is assigned to layer 1 and bus b_2 is assigned to layer 2, so the constraints $x_{1,1} = 1$ and $x_{2,2} = 1$ have to be added. The global upper bound O_{ub} is maintained during the searching, and it is updated whenever a better one is found. Note that the upper bound is only computed for the nodes where the buses with consecutive assignment constraints are all pre-assigned to some layers, i.e., an upper bound will not be computed for a node in the middle of the first stage. This is because our upper bound computation (the procedure CALUPPERBOUND() in Section 3.1) does not enforce the consecutive assignment constraints. From the end of the first stage, upper bounds are computed for the nodes. Similarly, additional constraints for the pre-assigned buses need to be added to the ILP of the procedure ILP-MIS() in Section 3.1. In addition, the pre-assigned buses also need concern when LCS is computed between the two traversals. Basically, we compute the local LCSs between the pre-assigned buses and then concatenate them. Fig.9 shows an example where b_1 and b_4 are pre-assigned buses in this layer. The local LCS between b_1 and b_4 is $< b_2, b_3 >$, and the local LCS after b_4 is $< b_6, b_7 >$, so concatenating them gives the LCS: $< b_1, b_2, b_3, b_4, b_6, b_7 >$. With these bounds computation and pruning rules, we can search the solution space faster.

3.3.3 *Target-Oriented Searching Rules*

Our target-oriented branch-and-bound searching algorithm is divided into rounds. In each round, we set a target which is used as a threshold to decide whether a node should be visited in this round. For example, in the first round, the target will be computed as: $target = O_{lb} + C$, where O_{lb} is the global lower bound of the original problem (the root in Figure 7), and C is a constant (set to 2 in our experiments). Actually, *target* is a guess we make on the optimal number of layers. If the lower bound of a node we reached is greater than *target*, this node will be marked and its subtree will not be visited in this round (may be visited in the following rounds). After one round, the global lower bound O_{lb} is updated as $target + 1$, and $target$ will then be updated as $target + C$. Note that a subtree which is explored in a previous round will not be repeatedly visited in any subsequent rounds. In general, this target-oriented searching strategy will search nodes with smaller lower bounds in earlier rounds. This idea is derived from the branch-and-bound algorithm described in [5]. In our target-oriented branch-and-bound algorithm, there are three conditions under which we can stop searching: (1) A feasible solution is found at a node whose upper bound equals to the global lower bound; (2) Current round is finished and the global upper bound O_{ub} is less than or equal to *target+1*; (3) All possible nodes are visited and searched.

Putting all the above steps together, we have the following pseudocode for our Target-Oriented Branch-and-Bound (TOBB) algorithm.

```
ALGORITHM TOBB($B, \{G_1, G_2, \ldots, G_p\}$):
  Compute $O_{lb}$ using ILP-COLOR() and binary search;
  Compute $maxLCS^R$;
  $O_{lb} \leftarrow \lceil \frac{|maxLCS^R|}{2} \rceil$, if $\lceil \frac{|maxLCS^R|}{2} \rceil > O_{lb}$ ;
  $target \leftarrow O_{lb} + C$;  // init $target$
  $O_{ub} \leftarrow n$;  // init $O_{ub}$ as the number of buses
  $pool_{cur} \leftarrow \{$node $S_0\}$;  // a queue of internal nodes
  $pool_{next} \leftarrow \emptyset$;  // a queue of internal nodes
  $basket \leftarrow \emptyset$;  // a stack of internal nodes
  repeat
      while $pool_{cur}$ is not empty do
          $S \leftarrow$ get a node from $pool_{cur}$;
          Push $S$ into $basket$;
          while $basket$ is not empty do
              $S \leftarrow$ pop a node from $basket$;
              if conflict exists for pre-assigned buses in $S$ then
                  continue;  // this subtree is pruned
              if ! ILP-COLOR($B, O_{ub} - 1, S$) then
                  continue;  // this subtree is pruned
              if ! ILP-COLOR($B, target, S$) then
                  Put $S$ into $pool_{next}$;  // $S$ is not visited now
                  continue;
              if first stage finished then
                  $S_{ub} \leftarrow$ CALUPPERBOUND($B, S$);
                  if $S_{ub} = O_{lb}$ then  return $S_{ub}$;
                  if $S_{ub} < O_{ub}$ then  $O_{ub} \leftarrow S_{ub}$;
              Generate and push all children of $S$ into $basket$;
      if $O_{ub} \leq target + 1$ then  return $O_{ub}$;
      if $pool_{next}$ is empty then  return $O_{ub}$;
      $pool_{cur} \leftarrow pool_{next}$;  $pool_{next} \leftarrow \emptyset$;
      $O_{lb} \leftarrow target + 1$;  $target \leftarrow target + C$;
```

3.4 Min-Cut Based Bus Ordering Algorithm

The ordering of the buses in the search tree affects the searching efficiency. For example, two buses b_i and b_j overlap with each other, so they cannot be assigned to the same layer. Suppose bus b_i is assigned to a specific layer at some internal node S, and the subtree rooted at S is being searched. After going down the tree many levels by enumerating the layer assignments for some other buses, we arrive at the level for bus b_j. When we try to assign b_j to the same layer with b_i, we realize that conflict exists and this searching direction should be terminated. Actually, this searching direction could have been terminated much earlier if b_j were immediately after b_i in the bus ordering, so that the solution space could be explored much faster. Therefore, we develop an algorithm to determine the bus ordering for searching efficiency.

Intuitively, we want to construct the search tree in a way that the subtrees containing infeasible solutions are pruned as early as possible. It is observed that early pruning is favored when the buses with more conflict with each other stay closer in the bus ordering. Based on this observation, we develop our min-cut based bus ordering algorithm, which is described below with a concrete example.

- Build a correlation graph of the buses. This graph depicts the conflict information of all the buses. Each vertex represents a bus. For every two buses with internal conflict, we add an edge with weight w_{in} between the two corresponding vertices. For every three buses with external conflict (and without internal conflict), we add an edge with weight w_{ex} between each two of the three corresponding vertices. In our implementation, w_{in} is set to 10 and w_{ex} is set to 1. Fig.10(a) shows a problem instance with 7 buses and 2 groups, and Fig.10(b) is the correlation graph constructed.

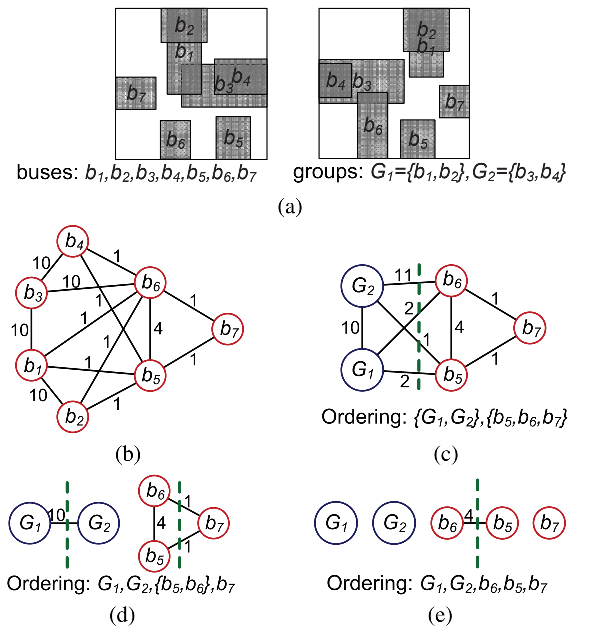

buses: b1,b2,b3,b4,b5,b6,b7
groups: G1={b1,b2},G2={b3,b4}
(a)
(b)
Ordering: {G1,G2},{b5,b6,b7}
(c)
Ordering: G1,G2,{b5,b6},b7
(d)
Ordering: G1,G2,b6,b5,b7
(e)

Figure 10: (a) A bus escape routing instance. (b) The correlation graph constructed. (c) The correlation graph after merging; the vertices representing groups and the ones representing other buses are separated. (d) Applying the min-cut algorithm. (e) Applying the min-cut algorithm again.

- Merge the vertices belonging to the same group. The edges connecting a vertex outside the group and the vertices inside the group are also merged, and their weights are summed up. Fig.10(c) shows the graph after merging. The groups need to be placed at the head of the ordering (in the upper level of the search tree), and the other buses are placed at the tail (in the lower level of the search tree), so a partial ordering is found and now the graph becomes two separate subgraphs. The partial ordering in Fig.10(c) is $\{G_1, G_2\}, \{b_5, b_6, b_7\}$.
- We then iteratively apply the min-cut algorithm on both subgraphs. Each subgraph further divides into two smaller subgraphs after cutting, and a finer partial ordering is obtained. Our strategy for deciding the relative ordering of the two smaller subgraphs is as follows: the one with more conflict with the buses in front is put closer to the head of the ordering, so that potentially more pruning can be done earlier in the search. In Fig.10(d), $\{G_1, G_2\}$ is cut into G_1 and G_2, and $\{b_5, b_6, b_7\}$ is cut into $\{b_5, b_6\}$ and b_7. The new partial ordering is $G_1, G_2, \{b_5, b_6\}, b_7$. Since $\{b_5, b_6\}$ has more conflict with $\{G_1, G_2\}$, it is put in front of b_7 in the ordering. The min-cut algorithm is iteratively applied until each subgraph contains only one vertex, and a complete ordering is decided. In Fig.10(e), $\{b_5, b_6\}$ is cut into b_5 and b_6, and b_6 is placed in front of b_5 since b_6 has more conflict with the buses in front. Thus, a complete ordering is obtained. The min-cut of an undirected edge-weighted graph $G(V,E)$ can be computed in time $O(|V||E| + |V|^2 \log |V|)$ [6].

4. EXPERIMENTAL RESULTS

We implemented our algorithms in C++, with Gurobi Optimizer [1] employed as our ILP solver. We first test using ILP to find Maximum Independent Set (MIS) and Minimum Coloring (MC) of bus projection rectangles. The results are shown in Table 1. The test

Table 1: Runtime of ILP on MIS and MC

Test Cases	# Bus	# Net	Runtime of MIS (sec) ILP	[3]	Runtime of MC (sec)
Ex1	8	272	0.003	0.005	0.010
Ex2	8	680	0.004	0.005	0.011
Ex3	11	712	0.004	0.007	0.016
Ex4	13	286	0.005	0.011	0.015
Ex5	14	644	0.005	0.015	0.016
Ex6	43	1539	0.012	0.272	0.423
Ex7	44	1428	0.012	0.320	0.439
Ex8	64	762	0.028	1.320	1.024

Table 2: Results of Our Branch-and-Bound Algorithm

Test Cases	# Bus	# Group	# Net	Results Opt. # Layer	Runtime (sec)
bb10	10	0	358	3	0.31
bb12	12	2	168	4	2.27
bb20	20	3	484	4	36.8
bb30	30	3	610	6	4.53
bb40a	40	0	432	10	932
bb40b	40	8	588	10	4299
bb50	50	6	946	7	546
bb60	60	8	1156	6	1842

cases are the same as those in [3], and each test case contains only one component. Results show that ILP is quite efficient in solving MIS and MC, making it practical to be embedded into branch-and-bound search. We can see that ILP solves MIS in much shorter time compared with the optimal algorithm proposed in [3]. We then test our branch-and-bound algorithm on a set of test cases derived from industrial data. Each test case contains a set of buses connecting two components, and some of them have consecutive assignment constraints. The results are displayed in Table 2. The column "# Bus" and the column "# Group" show the number of buses and the number of groups with consecutive assignment constraints, respectively. The column "# Net" indicates the total number of nets contained in the buses for each test case. The column "Results" reports the optimal number of layers and the runtime. The experiments are performed on a Linux workstation with two 3.0GHz Intel Xeon CPUs and 4GB memory.

5. CONCLUDING REMARKS

In this paper, we propose a novel branch-and-bound based optimal algorithm for the layer assignment problem of bus escape routing on PCBs. The practical consecutive assignment constraints are addressed and handled. This is the first optimal algorithm proposed in literature for this problem. Experimental results on industrial data validate our approach.

6. REFERENCES

[1] Gurobi optimizer. http://www.gurobi.com.

[2] M. R. Garey, D. S. Johnson, G. L. Miller and C. H. Papadimitriou. The complexity of coloring circular arcs and chords. *SIAM Journal of Algebraic Discrete Methods*, 1(2):216–227, 1980.

[3] H. Kong, Q. Ma, T. Yan and M. D. F. Wong. An optimal algorithm for finding disjoint rectangles and its application to PCB routing. *DAC '10: Proceedings of the 47th Annual Design Automation Conference*, pages 212–217, 2010.

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

[5] V. Stix. Target-oriented branch and bound method for global optimization. *Journal of Global Optimization*, 26:261–277, 2003.

[6] M. Stoer and F. Wagner. A simple min-cut algorithm. *Journal of the ACM (JACM)*, 44:585–591, 1997.

[7] X. Tang and M. D. F. Wong. FAST-SP: a fast algorithm for block placement based on sequence pair. *ASPDAC '01: Proceedings of the 2001 Asia and South Pacific Design Automation Conference*, pages 521–526, 2001.

[8] J. C. Whitaker. *The Electronics Handbook (2nd Edition)*. CRC Press, 2005.

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