Minimum capacity full array escape routing

Pouya Moetakef, Samir Khuller
Department of Computer Science, University of Maryland, College Park

Abstract

Layout design for integrated circuits (IC) is an important step in the fabrication process. As the device sizes are shrinking and more devices can be fit in the same area, the space left for wire routing diminishes, and the task of routing all devices to the wire bonding pads become more challenging. This work, presents two types of escape routing problems: 1) a minimum capacity escape routing of a full array, and 2) differential pairs escape routing.

The minimum capacity escape routing can provide a dense solution where the wire width and pitch can be optimized to achieve high yield in the fabrication process. Two solutions are presented for the minimum capacity escape routing, which is a polynomial algorithm designed by employing the shortest path algorithm, and a mixed integer programming (MIP) approach. The results of the two approaches are in agreement, while the polynomial algorithm provides a faster solution for larger arrays.

The differential pairs escape routing problem, involves finding routes that minimize the path length where members of each pair are traveling separately. After joining, the pair travels together to the boundary. The nature of this problem is NP-hard, thus there is no polynomial algorithm known to solve this problem. Therefore, in this work, a mixed integer programming (MIP) approach was used. This approach is designed to solve the problem in a single step to provide an optimal solution. This approach can be used to solve mixed differential pairs and single signals in the layout.

1. Introduction

Computer-aided design (CAD) tools were a huge step forward in speeding up the design process for integrated circuits (IC). As the device sizes become smaller, designers are able to fit more devices in smaller areas. These electronic devices still need to be wired to the outside world for incorporation in various electronic applications. As number of devices to be routed reaching large numbers (thousands and even millions), automated routing tools are becoming of immense importance to speed up the design process, and removing human-related errors in detecting the optimal route for wiring.

An example of such a scenario can be seen in the transition edge sensor (TES) arrays used for X-ray detection [1-2] (as shown schematically in Fig. 1). In such scenarios, the goal is to maximize the area used by a sensor to capture as much signal as possible. On the other hand, since all sensors are required to be wired, the routing is limited to the channel space available between every two sensors. For an optimal design, a designer requires information regarding the minimum capacity (maximum number of wires that can be fit in each channel space) to route all sensors in the array. With such information, a designer can decide the maximum wire width and pitch (therefore maximizing the fabrication yield) and maximum sensor size (minimum channel size).

There has been a considerable amount of work conducted in finding escape routing algorithms for routing selected number of pins with a predefined channel capacity by various research groups [3-5]. It has been shown that a maximum network flow algorithm [3], as well as linear programming [4], can be successfully used in solving escape routing problem families. However, to our knowledge, no work has been focused on finding the minimum capacity required to be able to route all the pins. It can be argued that available escape routing models can be used iteratively from a high capacity value while decrementing to find the minimum capacity. However, such an approach can take a long time to find the optimal solution. Hence in this work, we propose a polynomial time

algorithm designed using the shortest path algorithm to solve the problem in a very short time. Furthermore, a mixed integer programming approach was investigated for an optimal solution, and it was observed that the results of the two approaches are in agreement.

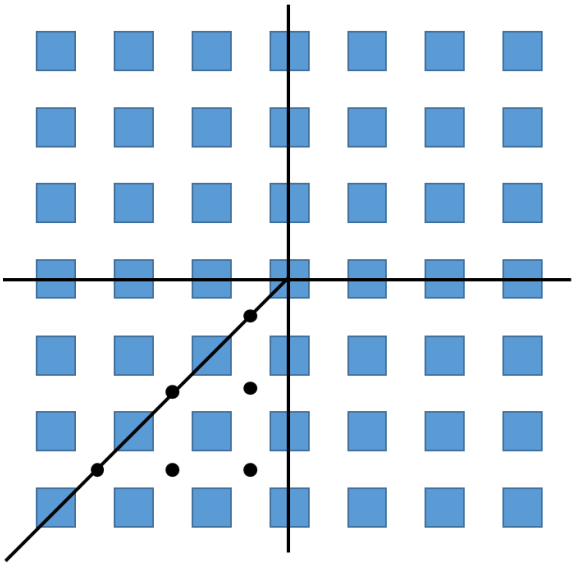

Fig. 1. Schematic of the device array geometry and available channel spacing for escape routing. The array can be split into 8 symmetrical regions. A solution for this 1/8 area can be mirrored for all other regions.

In electronic design, often, it is required that routes of certain devices travel together, to minimize the noise between them [6]. This problem is known as differential pair escape routing. This problem in nature is a multi-commodity flow problem which is NP-hard. Current solutions to this problem use 2-steps [6-8]. In the first step, non-crossing paths between each pair are determined and then polynomial algorithms such as network flow are used to route the paired routes to the boundary. Using a two-step algorithm, enhances the running time significantly, however, the solution is not always optimal [8]. Optimality becomes important when a dense array of pairs or a mix of device pairs and single signals are used in a layout. In such cases, it is important to use maximum available resources to be able to determine the minimum capacity required to route the layout or detect whether the layout is routable. Hence, in this work, we present a mixed integer programming approach to solving the problem in a single step to produce an optimal solution.

The structure of this report is as follows: Section (2) provides the details of a polynomial time algorithm design to solve minimum cost escape routing of a full array problem. Section (3) Solves the same problem as discussed in section 2 using a mixed integer programming approach. Section (4) provides a mixed integer programming approach for solving differential pairs escape routing problem. Section (5) concludes the work and provides future insights.

2. Minimum capacity escape routing for a full array of singular devices.

2.1 Problem formulation

As shown in Fig. 1, a grid layout for an array of $n \times n$ devices can be used to model the problem. The goal is to find the escape routed pattern as well as the minimum channel capacity to be able to route all devices to the grid boundary (exit channels), with design rules that wires cannot cross each other (avoid forming shorts), and the routing is limited to a single layer (i.e. no via allowed).

2.2 Definitions:

In this section, basic preliminaries such as nodes and edges used in the current formulation are defined and discussed.

2.2.1. Nodes:

In a flow model, a flow starts from a source and travels through channels to reach the sink. The junction between four channels is where a flow can change its direction, and hence flow nodes can be defined. Each junction can be represented by a node. Following are the different types of nodes used in this study (Fig. 1).

1) Junction nodes (n_i): Each junction can be represented by a node, which is referred to as a junction node.
2) Source nodes (n_{si}): Source nodes represent devices. These nodes can be considered the starting point of each flow (wire), and hence the flow can only exit the source nodes.
3) Sink node (n_s): Which represent the union of all boundary nodes, and the destination for a flow. Therefore, the sink node can only accept incoming flow.

Collection of all nodes are referred to as *V*.

2.2.2. Edges:

Edges represent the flow direction between two adjacent nodes. Edges between junction nodes are all orthogonal and bidirectional. The edges from source nodes to junction nodes are unidirectional, i.e. flow cannot enter a source node. Edges between boundary nodes and the sink node are also unidirectional, where the flow enters the sink node. Furthermore, nodes on the boundary are not connected to each other.

In this work, edges are represented by $e_{i,j}$ which corresponds to an edge from node *i* to node *j*. The collection of all edges is referred to as *E*.

There are few properties associated with each edge which are discussed in the following sections.

2.2.2.1. Edge cost:

The heuristic used in the algorithm is based on a variable cost for edges. A row of edges that are farther from the boundary is more expensive than those closer to the boundary. This heuristic ensures that a chosen route descends faster towards boundary rather than traveling within a row, to prevent trapping unvisited devices.

2.2.2.2. Edge capacity:

Edge capacity represents the maximum number of wires (amount of flow) that can be fit in an edge. Two capacity variables can be defined which are geometrically related to each other.

C_O: Orthogonal capacity, which corresponds to the capacity of the channels.

C_D: Diagonal capacity, which corresponds to the diagonal capacity at junctions.

Geometrically C_O and C_D are dependent, and hence both can be viewed as the same variable. In this study the ratio $C_D/C_O = d = \sqrt{2}$ was used, which means that:

$$C_D = round(\sqrt{2}\, C_O).$$

In this model all edges have a capacity C_O. Diagonal capacity is modeled by giving each node a capacity of C_D. Due to single commodity flow nature of the problem, at any junction only one of the two scenarios discussed below can be found:

1) Single incoming (outgoing) flow and multiple outgoing (incoming) flows: in this case the capacity within the node cannot exceed C_O which is dictated by the incoming flow, and hence the amount of flow is always $< C_O < C_D$.
2) Two incoming flows and two outgoing flows: In this case, each incoming flow can be maximum C_O and hence $2C_O$ maximum flow can enter and leave, which is more than C_D. Therefore, having a capacity limit of C_D on the node, it can be guaranteed that no more than C_D flow can enter and leave. It must be noted that a horizontal flow and a vertical flow can be replaced by non-crossing flows.

Therefore, as discussed having a flow limit of C_D on a node is sufficient to prevent illegal routes.

2.3 Minimum Capacity

One of the goals of this work is to find the minimum capacity (C_O) required for routing all devices (full array) to borders. As we can see, devices sitting on the border, face no restriction for the available routing space. Therefore, these devices get their routes for free. Out of $n \times n$ devices only $(n - 2) \times (n - 2)$ device array requires routing. Therefore, in total there are $(n-2)^2$ wires exiting the borders. Looking at the borders,

there are only (n – 1) channels available at each side, and hence, in total there would be 4 (n -1) exit channels. Since all the wires have to escape, the minimum capacity of channels will be:

$$C_O = ceil\left(\frac{(n-2)^2}{4(n-1)}\right)$$

Having solved the minimum capacity problem, the next objective is to find the escape routing pattern for the full array using minimum capacity.

2.4 Algorithm

In this section, algorithm design for solving minimum capacity escape routing of a full array is discussed, as shown below.

Prepare $G = (V, E)$ 1/8 segment routing network
visited_sources = empty
C_O = (n-2)^2/(4(n-1))
While size(visited_sources) < size(sources) do
    node = select next unvisited node using
        traversal method shown in Fig. 2
    path = Find shortest path from node to sink
    Add node to visited_sources
    Following nodes and edges on the path:
        Increment flow(edge) by 1
        Delete edge if flow(edge) exceeds
            C_O
        Delete all edges to node if sum of the
            flows exceed C_D
        Perform cleaning step on the node

As discussed in section (2.3) minimum capacity of channels can be calculated mathematically. Hence, in the first step of the algorithm, minimum capacity is calculated. The rest of the algorithm discusses the generation of the routing pattern.

Considering the symmetrical nature of this problem, the device array can be split into 8 regions as shown in Fig. 1. Solving any of the 1/8 regions will lead to a full solution after applying mirror operations. Hence, after finding a solution for the 1/8 segment, a mirror operation on the diagonal edge of the segment gives a solution to a quadrant. Performing a mirror operation on one of the edges of the quadrant gives half and another mirror operation on the half region gives the full solution. Therefore, in this study, the focus is given to solving the 1/8 region.

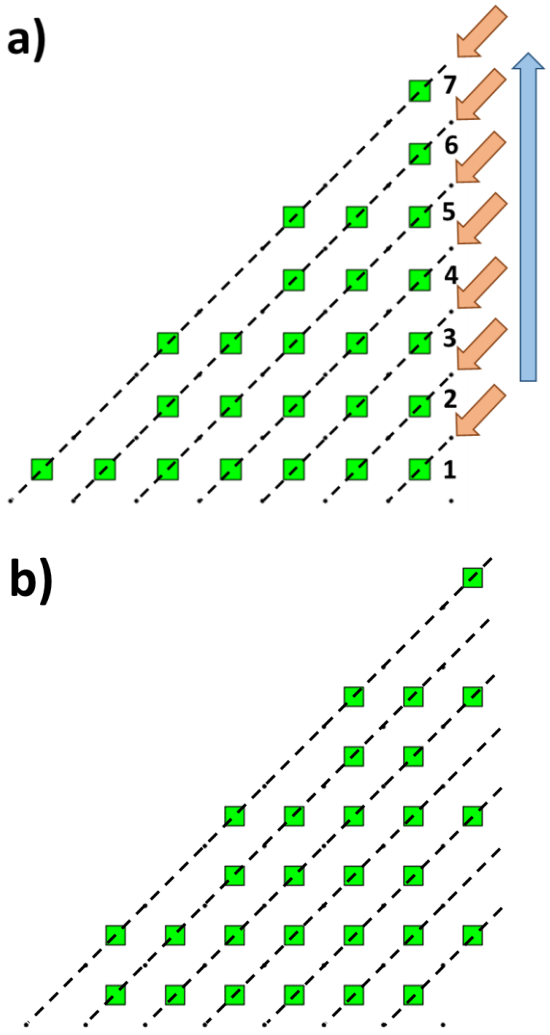

Fig. 2. Illustration of routing network construction for 1/8 segment for (a) 14 × 14, and (b) 15 × 15 grid size. Node traversals are shown by dashed lines. Blue arrow in (a) shows traversal direction between traversal layers, and orange arrows show traversal direction with a layer.

The next step is creating the routing network (graph) for the 1/8 segment, as shown in Fig. 2. The network is a directed graph $G = (V, E)$. Geometrically, depending on the number of devices (even or odd) the segments can share half of the devices or node capacity on their borders with other segments. These conditions are shown in Fig. 2. In Fig. 2a, an even number of n results in sharing half the capacity on the far right border

with a segment on its right. On the other hand, Fig. 2b shows that for an odd number of n, half of the devices on the far right border are being shared. For the diagonal border, devices are always shared between the two neighboring segments.

Initially, flow values for all nodes and edges are 0, except for the sources which is 1. As shown in Fig. 2a, algorithm traverses through each node via the traversal path shown (following blue arrow between layers, and orange arrows within each layer), and finds the shortest (least cost) path from the source to the sink, using Dijkstra's algorithm. Along the path, the flow value for all edges is incremented by 1. When an edge exceeds C_O, that edge is being deleted from the network graph. Furthermore, for each node on the path, if the sum of incoming flow or outgoing flow exceeds C_D, all incoming and outgoing edges involving that node are being removed. Furthermore, each node on the path will undergo a cleaning step. Since the flow problem is a single commodity, the crossing paths can be replaced by non-crossing path as discussed in [3]. Therefore, path shortening scenario shown in Fig. 3, can be applied on the resolved crossing routes. In the cleaning step, if there exists an incoming edge from source to node n_i, and an outgoing edge from n_i to n_j such that e_{ij} lies next to the same source (as shown in Fig. 3), the edge, e_{si}, can be replaced by the edge, e_{sj}.

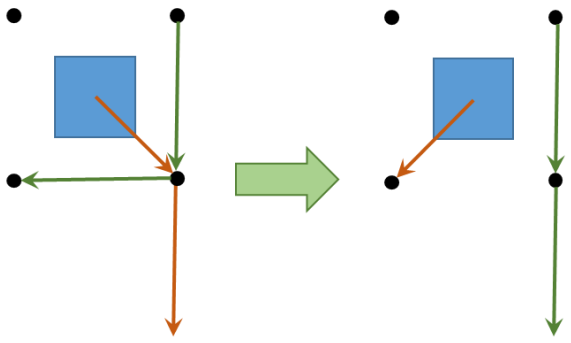

Fig. 3. Cleaning step to shorten a path after resolving crossings.

The process of traversing through source nodes, finding the shortest path to the boundary, update, and clean the edges, continues until all sources are routed to the sink. The flow results are then geometrically separated (as discussed in [3]) to produce the final routing pattern.

2.4.1. Proof:

The proof of optimality is composed of two parts: (1) proof that minimum capacity holds for any grid size, n, which is already proven through the derivation of the minimum capacity equation, and (2) proof that the C_D is never exceeded in the junctions. Following represents the proof for (2).

A cut can be viewed as a separation between two sides of a node column, as shown in Fig. 4 by the dashed line. All the devices on the right-hand side of the cut are required to be wired. As routing progresses, the exit nodes from right to left are being filled. Therefore, a cut also separates filled and empty exit nodes (exit nodes to the right of the cut are all filled, while the ones on the left side are all empty). In this condition, it can be observed that some devices can be routed to the available exit nodes, while the rest has to pass through the cut to be routed to empty ones. So, here it can be shown that the total capacity of junction nodes on the cut, each with a capacity of C_D, exceeds the number of remaining routes (devices).

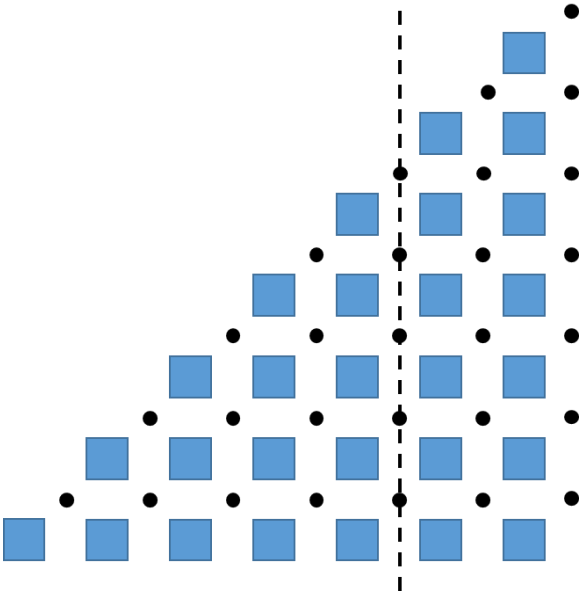

Fig. 4. Schematic of the cut (dashed line) in the grid geometry.

Assuming an $N \times N$ grid of devices, the 1/8 segment will have ${}^{1}/_{2}\left({}^{N}/_{2} \times {}^{N}/_{2}\right)$ devices. Considering a cut placed at node column i, the number of devices on the right side can be calculated as the following:

$$\left({}^{N}/_{2} - i\right)\left({}^{N}/_{2} - 1\right) - \frac{\left({}^{N}/_{2} - i\right)^{2}}{2}$$

Where the first term represents the number of devices in a rectangular shape, without considering the bottom row (since we get them for free), and the second term is the number of devices missing from the top triangular section.

Considering C_O is the capacity of each exit node, and the fact that a node column located at the right-hand border is shared between 2 segments, following equation provides the number of routes that can exit before reaching the cut:

$$\left({}^{N}/_{2} - i - 1 + {}^{1}/_{2}\right)C_O = \left({}^{N}/_{2} - i - {}^{1}/_{2}\right)C_O$$

Since the objective is to prove that the number of routes reaching cut is less than the diagonal capacity available at the cut, following inequality can be written.

$$\left({}^{N}/_{2} - i\right)\left({}^{N}/_{2} - 1\right) - \frac{\left({}^{N}/_{2} - i\right)^{2}}{2} - \left({}^{N}/_{2} - i - {}^{1}/_{2}\right)C_O \leq iC_D$$

Furthermore, from geometrical constrains the relationship between C_O and C_D is known:

$$C_D = \sqrt{2}C_O$$

And hence the inequality equation can be rewritten as:

$$\left[\left({}^{N}/_{2} - i\right)\left({}^{N}/_{2} - 1\right) - \frac{\left({}^{N}/_{2} - i\right)^{2}}{2} - \left({}^{N}/_{2} - i - {}^{1}/_{2}\right)C_O\right] - \sqrt{2}iC_O \leq 0$$

Which can be simplified as the following after substituting C_O with its equation:

$$2i - 1 - \frac{1}{N-1} - \left(N - 3 + \frac{1}{N-1}\right)\left(\frac{2\sqrt{2}i - 1}{N - 2i}\right) \leq 0$$

Given that $N - 3 > N - 2i$ for $i > 1$, therefore:

$$\frac{N - 3 + \varepsilon}{N - 2i} > 1$$

Which can be estimated to be

$$2i - \frac{1}{N-1} - 2\sqrt{2}i \leq 0$$

Which is always true (i.e. the left-hand is always negative).

As for $i = 1$ case, we have:

$$1 - \frac{1}{N-1} - \left(N - 3 + \frac{1}{N-1}\right)\left(\frac{1.8}{N-2}\right) \leq 0$$

The minimum value for N is 3, which gives the left-hand side a value of -0.4 < 0, which means inequality holds. For $N > 3$, the value approaches 1 – 1.8 = -0.8 < 0, meaning inequality holds for all $N > 3$.

As shown above, it can be observed that the inequality holds for all values of i and N, and hence it can be concluded that a number of wires passing a cut cannot exceed the diagonal capacity of that cut.

2.4.2. Running time:

The shortest path algorithm used is the Dijkstra's algorithm with a running time of $O(V \log V + E \log V)$. Considering we find shortest paths for $V = N^2$ devices, therefore the overall running time will be $O(V^2 \log V + EV \log V)$. However, it can be argued that the actual running time is much shorter since bottom row nodes reach the exit faster than top row nodes (having a shorter search domain).

2.5 Results and discussion

The algorithm was implemented in Python, and tests were performed on an Intel Core i7 CPU at 3.4 GHz, with 64 GB of RAM.

Figure 5, shows the routing results for the two case scenarios presented in Fig. 2 (e.g. even and odd grid size). As it can be seen in both cases, all design rules are respected, and no junction exceeds diagonal capacity.

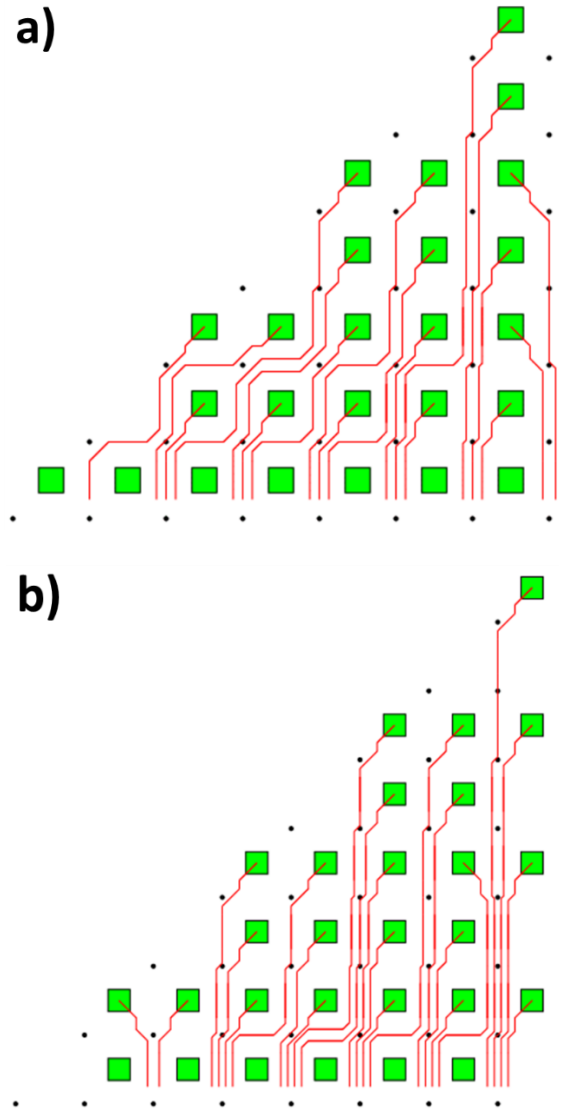

Fig. 5. Routing results for (a) 14×14, and (b) 15×15 grid size.

Figure 6, shows the solution for a larger network of $N = 35$, with more congested, routes. As expected, all design rules are respected, and all devices were routed to exit nodes using the calculated minimum capacity.

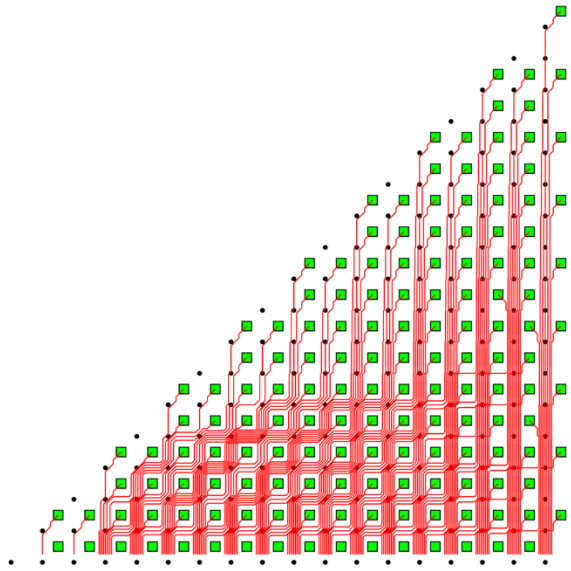

Fig. 6. Routing solution for 35×35 grid size.

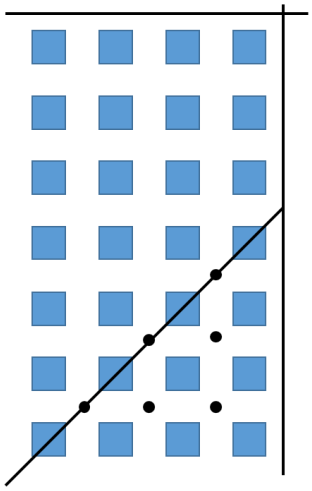

Fig. 7. Schematic of an $n \times m$ array quadrant split into 2 sub-sections.

2.6 Model extension

The minimum capacity escape routing model discussed was designed based on an $n \times n$ array of devices. This model can be extended to an $n \times m$ array. As shown in Fig. 7, an $n \times m$ array can be divided into 4 sections. Then each quadrant can be split into two subsections as shown in Fig. 7. Minimum capacity escape routing algorithm then can be applied to each subsection, and the union of the two can be mirrored to produce the full geometry. Following

equation, represents the revised minimum capacity for $n \times m$ array.

$$C_O = ceil\left(\frac{(n-2)(m-2)}{2(n-1)+2(m-1)}\right)$$

3. Minimum Capacity escape routing of a full array using mixed integer programming

3.1 Problem formulation

Similar to section (2), a grid layout of $m \times n$ device array can be used for this model. The goal is to find the escape routed pattern as well as the minimum channel capacity to be able to route all devices to the grid borders, with design rules that wires cannot cross each other (avoid forming shorts), and the routing is limited to a single layer (i.e. no via allowed).

3.2 Algorithm

This problem can be solved using a flow model [3, 4]. In the flow model, a flow starts from a source (in this case a device), travels through the channels, and ends up in a sink (in this case a channel on the border). The algorithm to solve this problem consists of constructing the routing network, $G = (V, E)$, as shown in Fig. 8. Then routing formulation is applied to a mixed integer programming. Gurobi optimizer [9] is used to solve the mixed integer programming. The flow results are then geometrically separated (as discussed in [3]) to produce the final routing pattern.

3.3 Definitions:

3.3.1. Nodes:

In a flow model, a flow starts from a source and travels through channels to reach the sink. The junction between four channels is where a flow can change its direction, and hence flow nodes can be defined. A tile can be defined as a collection of 4 neighboring junctions. Each junction can be represented by a node or set of nodes. Therefore, four types of nodes were introduced in this model (Fig. 8).

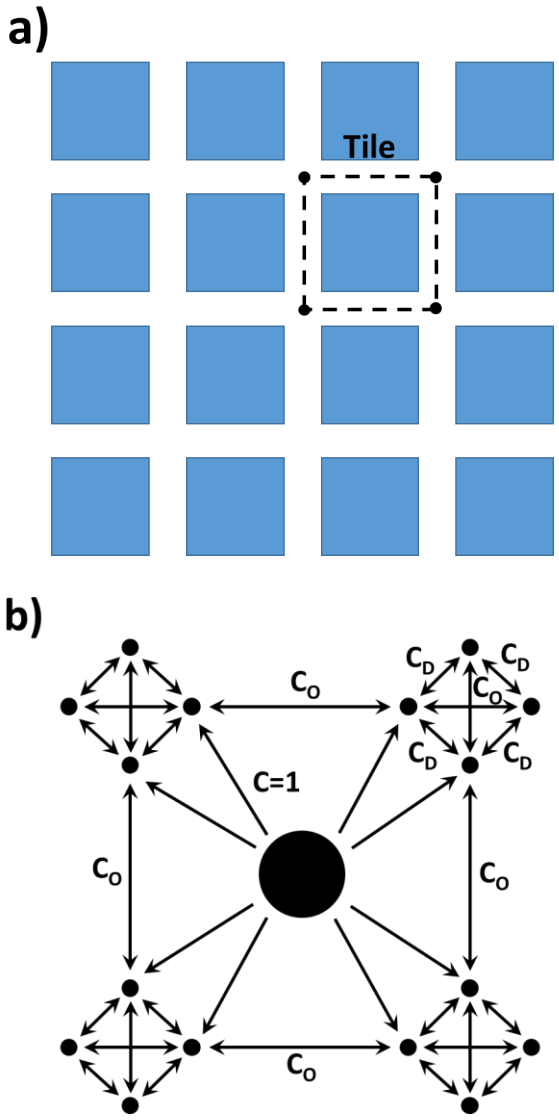

Fig. 8. a) Schematic of the device array geometry and available channel spacing for escape routing. A Tile consists of four neighboring tile nodes (as shown by dashed lines). b) A magnified schematic of a tile, where each tile node is replaced by 4 junction nodes to account for diagonal capacity (C_D), while tile edges are capped with orthogonal capacity (C_O). Each device is connected to neighboring junction nodes via directed edges with a capacity of unity.

1) Tile nodes (n_i): Each junction can be represented by a node, which is referred to as a tile node. The tile nodes denote the flow within channels.
2) Junction nodes ($n_{ik}\ \forall k \in [t, b, r, l]$, where [*t*, *b*, *r*, *l*] represent the top, bottom, right, and left junction nodes, respectively): To improve routability, it is of immense importance to take advantage of diagonal spacing at channel junctions. Hence, each tile node is divided into 4 nodes at the corners of the junction, which are referred to as junction nodes. These junction nodes correspond to the internal flow within a tile node.
3) Source nodes (n_{si}): Source nodes represent devices. These nodes can be considered the starting point of each flow (wire), and hence the flow can only exit the source nodes.
4) Sink node (n_s): Which represent the union of all boundary nodes, and the destination for a flow. Therefore, the sink node can only accept incoming flow.

A complete node layout schematic is presented in Fig. 8b.

3.3.2. Edges:

Edges represent the flow direction between two adjacent nodes. Here, an edge between two adjacent tile nodes is referred to as tile edge. Tile edges are all orthogonal and bidirectional. Edges between junction nodes are referred to as internal edges. As shown in Fig. 8b, six bidirectional internal edges between junction nodes can be observed, where four edges are diagonal and the other two are orthogonal. The edges from source nodes to junction nodes are unidirectional, i.e. flow cannot enter a source node. Edges between boundary nodes and the sink node are also unidirectional, where the flow only enters the sink node. Since these boundary nodes are not connected to each other, junction nodes were not considered for them.

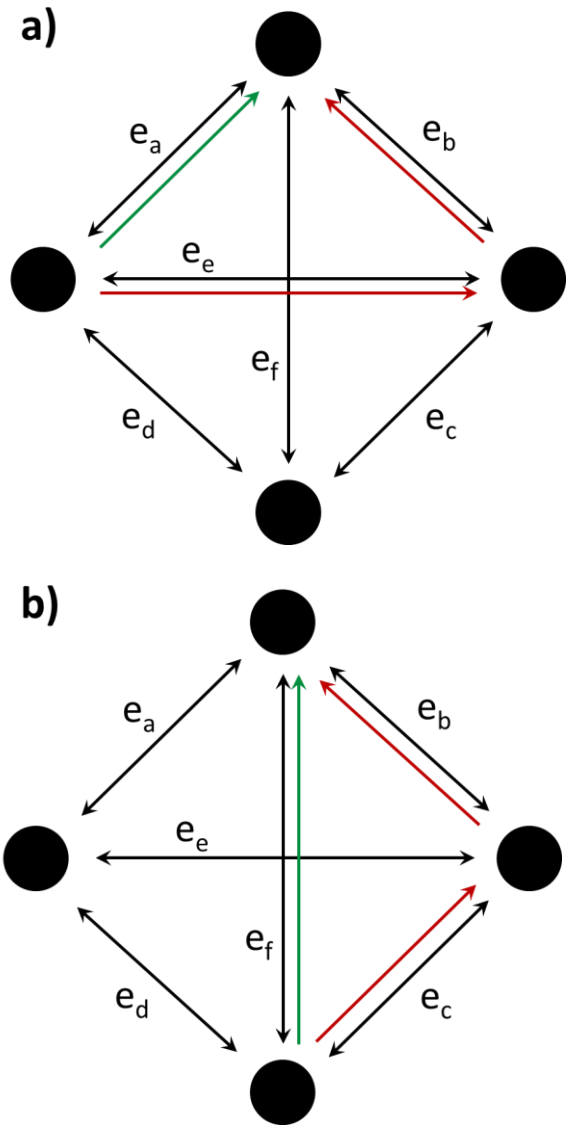

Fig. 9. Junction nodes and internal edges showing two possible configurations (red arrows) that can be substituted with a shorter length edge (green arrow), and hence show sub-optimality of the chosen path.

In this work, edges are represented by $e_{i,j}$ which corresponds to an edge from node *i* to node *j*. Based on this definition, $e_{i,j}$ signifies a tile edge between tile nodes, *i* and *j*, $e_{ik,il}\ \forall k, l \in [t, b, r, l]$ represents an internal edge in tile node *i*, between junction nodes *k* and *l*, $e_{si,jk}$ represents an edge from source *si* and junction node *jk*, and $e_{i,s}$ denotes an edge from tile node *i* to sink *s*. The collection of all edges is referred to as *E*.

There are few properties associated with each edge which are discussed in the following sections.

3.3.2.1. Edge length:

To minimize the length of wires in the solution, a length cost per unit length ($l(e_{i,j})$) was associated with each edge. Geometrically one can observe that the length cost ratio between diagonal internal edges to orthogonal internal edges is constant, $\sqrt{2}/_2$.

3.3.2.2. Edge capacity:

Edge capacity represents the maximum number of wires (amount of flow) that can be fit in an edge. Two capacity variables can be defined which are geometrically related to each other.

C_O: Orthogonal capacity, which corresponds to the capacity of the channels (tile edges).

C_D: Diagonal capacity, which corresponds to the diagonal capacity of the internal diagonal edges.

Geometrically C_O and C_D are dependent, and hence both can be viewed as the same variable. In this study the ratio $C_D/C_O = d = \sqrt{2}$ was used, which means that:

$$C_D = round(\sqrt{2}\, C_O).$$

In this model parallel diagonal edges are given the same C_D since each edge alone should be allowed to use the full C_D capacity while ensuring that the sum of the flow in parallel edges remains below C_D. Considering the schematic in Fig. 9, a mix of diagonal and orthogonal edges can be viewed as a legal choice. So, considering sum of the two diagonal edges with same orientation cannot exceed C_D and capacity of orthogonal orientation is capped by C_O, it can be observed that for every single orthogonal edge, C_D/C_O diagonal capacity will be used. Therefore, we can derive inequalities in equation (1) and (2) to account for the mix of diagonal and orthogonal internal edges.

$$e_b + e_d + \frac{C_D}{C_O}(e_e + e_f) \leq C_D \qquad (1)$$

$$e_a + e_c + \frac{C_D}{C_O}(e_e + e_f) \leq C_D \qquad (2)$$

Lemma 1: Considering constraint in inequalities (1) and (2), in an optimal solution, no configuration can be found to exceed C_D.

Constraints in equations (1) and (2) already address the case that flow in edges e_a, e_c, e_e, and e_f (Fig. 9) cannot exceed C_D. Now other cases can be considered.

Case I) Two diagonal edges with different orientation e.g. e_b and e_c, and an orthogonal edge e_e: In this case we know that the tile edge capacity from the node is capped by C_O. If all 3 nodes (e_b, e_c, and e_e) are either incoming or outgoing flow then their sum cannot exceed C_O (due to conservation of flow rule), and hence the flow cannot be higher than $C_O < C_D$.

If any two configurations of the 3 edges are used for incoming and the third is used for outgoing flow (as shown by red arrows in Fig. 9a and 9b), then we can find a shorter path between the two as shown by green arrows, which disagrees with optimality of the solution.

Case II) Two diagonal edges with different orientation e.g. e_b, and e_c, and an orthogonal edge e_f: In this case we can see that the orthogonal and diagonal edges carry the same flow, and since the cost of orthogonal edge is lower, the two diagonal edges can be substituted with an orthogonal edge, reducing the length cost, and hence disagrees with optimality of the solution.

As proven above considering inequalities in equations (1) and (2) no flow configuration greater than C_D can be within a tile node.

Directed edges with a capacity of unity shown in Fig. 8b represent flow from source nodes (devices). Each source node is connected to 8 neighboring junction nodes via 8 directed edges. The boundary nodes are connected to a single giant sink node out of the network via a directed edge each with a capacity C_O. The devices (sources) on the border connected with a single directed node (capacity of unity) to the giant sink node, as we can get a direct wire out of them for free.

3.4 Integer programming model:

Having explained all the preliminaries, an integer programming model can be formulated. Following are a few definitions used in this formulation.

$X(e_{i,j})\ \forall e_{i,j} \in E$ represents the flow variable for an edge $e_{i,j}$.

S_i represents the initial flow on node i. Therefore:

$$S_i = \begin{cases} 1\ \forall i \in [sources] \\ -\sum_j S_j\ \forall i \in [giant\ sink], \forall j\ \in [sources] \\ 0\ \forall i \in [otherwise] \end{cases}$$

3.4.1. Objective function:

The objective of this model is to minimize the length as well as orthogonal capacity (C_O). Since C_D is directly related to C_O, C_D is not used in the objective function. To define a priority on the two objectives, a linear combination of the two objectives with a priority factor (α) was used as the following:

$$Min\left\{(1-\alpha)\sum_{e_{i,j}\in E} X(e_{i,j})l(e_{i,j}) + (\alpha)C_O\right\}$$

Priority factor can be tuned to give higher priority to the capacity minimization or length minimization, which can result in different solutions. However, for the purpose of this study higher priority factor used to give higher priority to capacity minimization as this is the main objective of this study.

3.4.2. Constraints:

I) Conservation of flow: considering the initial flow S_i for node i, the difference between the sum of incoming flow and sum of outgoing flow should be equal to S_i.

$$\sum_{e_{i,j}\in E} X(e_{i,j}) - \sum_{e_{k,i}\in E} X(e_{k,i}) = S_i$$

II) The amount of flow for each edge is capped by orthogonal/diagonal capacity.

$$X(e_{i,j}) \le C_O\ \forall e_{i,j} \in [Orthogonal - E]$$

$$X(e_{i,j}) \le C_D\ \forall e_{i,j} \in [Diagonal - E]$$

III) Inequalities in equations (1) and (2) to ensure flow within internal edges do not exceed C_D.

$$X(\overrightarrow{e}_{ir,it}) + X(\overrightarrow{e}_{ib,il}) + d\left(X(\overrightarrow{e}_{il,ir}) + X(\overrightarrow{e}_{it,ib})\right) \le C_D\ \forall i \in [Tile\ nodes]$$

$$X(\overrightarrow{e}_{il,it}) + X(\overrightarrow{e}_{ib,ir}) + d\left(X(\overrightarrow{e}_{il,ir}) + X(\overrightarrow{e}_{it,ib})\right) \le C_D\ \forall i \in [Tile\ nodes]$$

IV) Constraint to ensure C_D will accept the correct value.

$$d * C_O - {}^1/_2 \le C_D \le d * C_O + {}^1/_2$$

V) Real and integer constrain for variables.

$$C_O, C_D \in [Positive\ Integer]$$

$$X(e_{i,j}) \ge 0\ \ \forall e_{i,j} \in E$$

3.5 Results and discussion

The algorithm was implemented in Python, and tests were performed on an Intel Core i7 CPU at 3.4 GHz, with 64 GB of RAM. Gurobi optimizer package [9] for Python was used to solve the mixed integer programming problem. During execution, the running time was measured through Gurobi package. For evaluation purposes, two sets of tests were designed.

1) In the first series, border nodes were only assigned to right and bottom side of the design, which can represent ¼ of a full design, with one difference, where in real ¼ design the left and top borders should be limited to ½ C_O capacity, while in current evaluation, left and top borders were kept at C_O capacity. This ¼ design can be used in future implementations to speed up the running time by >4 times since the other quadrants can be viewed as a mirror of the bottom-right quarter.

2) In the second series of the tests, a full design (with borders assigned to all 4 sides) was used.

To further evaluate the correctness of minimum capacity, another solver with a fixed capacity of C_O - 1 was implemented, and it was verified that Gurobi cannot produce any solution with given conditions.

Table 1. Minimum capacity and MIP running time for different sizes of test case series 1 (quadrant geometry).

Size	Number of devices	Min C_O	MIP run time (sec)
10×10	81	4	0.13
20×20	361	9	0.76
30×30	841	14	3.85
40×40	1521	20	19.52
60×60	3481	30	97.68
80×80	6241	40	596.21

Table 2. Minimum capacity and MIP running time for different sizes for test case series 2 (complete geometry).

Size	Number of devices	Min C_O	MIP run time (sec)
20×20	361	5	0.30
30×30	841	7	1.15
40×40	1521	10	3.26
60×60	3481	15	36.01
80×80	6241	20	364.09

Table 1 and 2, show the minimum capacity and running times for test cases within category 1 and 2, respectively. In all cases, the minimum C_O was verified to be the minimum capacity that can be achieved, through the fixed capacity solver. It must be noted that the grid size represents the size of tile nodes, and hence for $n \times m$ grid, there are $(n - 1) \times (m - 1)$ devices. Comparing running time results from table 1 to those of table 2 we can observe that solving quarters and then mirroring the results for other 3 quadrants can save time by about ×10 times.

Running times of the MIP approach are much slower than the polynomial time algorithm discussed in section 2. The algorithm in section 2 can compute the 80 × 80 grid size in 1.5 seconds, while MIP approach solves the same problem is 10 minutes. However, the advantage of the MIP approach is the optimality of the solution regarding the minimum length, while algorithm in section (2) cannot guarantee the minimum overall length of the solution. The results of the MIP approach, therefore, confirm the correctness of the minimum capacity determined via the algorithm discussed in section (2), while adding optimality to the minimum length to the solution.

Figure 10, demonstrates the routing results after performing a geometrical separation on the flow solution. As it can be seen all devices have been wired and all boundary nodes (except corner ones) are at their full capacity. Furthermore, no design rule was found to be violated.

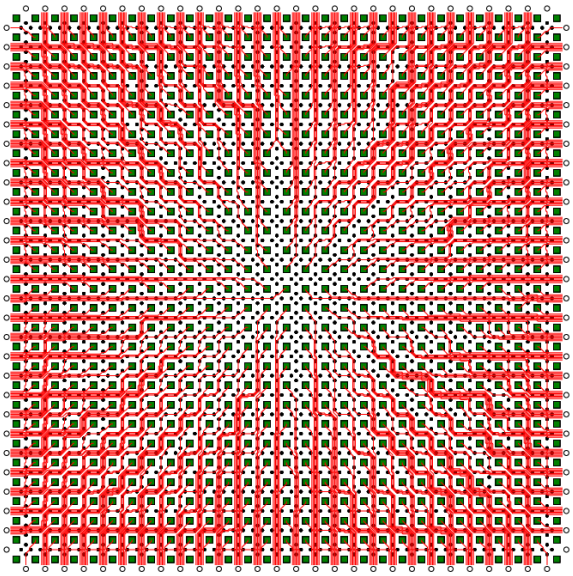

Fig. 10. A routing solution of minimum capacity for 30×30 grid size.

4 Escape routing of differential pairs using mixed integer programming

Escape routing of differential pairs is an NP-hard problem [7]. So far, the proposed methods involve 2 steps: 1) Finding the shortest path between each pair using various algorithms such as congestion routings [8] or linear programming [6, 7]. 2) Then solving the flow network from boundaries to any point on the shortest paths. Although all these methods produce solutions, their optimality is in question. For instance when solving for shortest paths between pairs, currently available models consider shared paths illegal [7], to simplify computation and achieve faster running times. Yet, considering such paths, can provide denser solutions, and utilize the available space to lower the minimum capacity of the channels. In this section, a mixed integer programming model for solving differential pairs escape routing problem in a single step is provided, which produces an optimal solution.

4.1 Problem formulation

Assuming a layout consisting of an array of $m \times n$ devices in a grid geometry, with designated devices as pairs, the goal is to find the escape routed pattern for all device pairs to the grid boundary (or designated exit nodes), with design rules that maximum capacity of each channel is given by C_O, wires cannot cross each other (avoid forming shorts), and the routing is limited to a single layer (i.e. no via allowed).

4.2 Algorithm

This problem can be solved using a multi-commodity flow model. Similar to single commodity flow model (used in section 3), a flow starts from a source (in this case a device), travels through the channels, and ends up in a sink (in this case a channel on the border). Each commodity represents the flow for a given pair. In multi-commodity flow problem, each path is distinct and hence unlike single commodity problem, crossings of two different commodities cannot be resolved at junctions. Hence it is necessary to mark solutions with crossings illegal. The algorithm to solve this problem consists of constructing the routing network, $G = (V, E)$, as shown in Fig. 8a. Then routing formulation is applied to a mixed integer programming. Gurobi optimizer [9] is used to solve the mixed integer programming. The flow results represent the final flow pattern.

It must be noted that the results presented in this section are preliminary and hence geometrical separation on the wires was not applied.

4.3 Definitions

4.3.1. Commodities:

Commodities (c) represent the type of flow for each pair. Therefore, the number of commodities is equal to the number of pairs in the problem.

4.3.2. Nodes:

For this model, a simple grid node geometry is used (as shown in Fig. 8a). Therefore, each junction is represented by a node (n_i), and each device is represented by a source node (n_{si}). A sink node (n_s) is a union of all the boundary exit nodes.

4.3.3. Edges:

Edges represent the flow direction between two adjacent nodes. In addition to direction property, each edge can support only a single commodity. Therefore, edges represented as $e(c, n_i, n_j)$, which represents the flow of type c commodity between nodes n_i and n_j. Edges between junction nodes are bidirectional, while edges between source nodes and junction nodes, and junction nodes and sink node are unidirectional.

All edges have a length of unity and a capacity of C_O (fixed). It must be noted to simplify the problem and enhance the running time, the diagonal capacity is ignored for this study.

4.4 Mixed integer programming model

This section presents a mixed integer programming model formulated to solve the escape routing of differential pair problem in a single step. Following are a few definitions used in this formulation.

$X(e_{c,i,j})\ \forall e_{c,i,j} \in E$ represents the flow variable for an edge $e_{c,i,j}$.

$Y(e_{c,i,j})\ \forall e_{c,i,j} \in E$ a binary variable which represents whether an edge $e_{c,i,j}$ is selected for flow (1 represents selected and 0 not selected).

S_i represents the initial flow on node i for commodity c. Therefore:

$$S_{c,i} = \begin{cases} 1\ \forall c, i \in [source\ pairs] \\ -\sum_j S_j\ \ \forall c, i \in [sink], \forall j\ \in [sources] \\ 0\ \forall i \in [otherwise] \end{cases}$$

4.4.1. Objective function:

The objective of this model is to minimize the length cost, $l(e_{c,i,j})$. Length cost only depends on the number of edges used. Therefore, if an edge carries a flow more than unity, the length cost will still be the same as if the flow value was unity. This ensures that the model would find the shortest path to join the flow from the same commodity to lower the cost.

$$Min\left\{\sum_{e_{c,i,j}\in E} X(e_{c,i,j}) + Y(e_{c,i,j})l(e_{c,i,j})\right\}$$

In order to solve a mixed differential pair and single signal escape routing problem, length cost for single signals commodity (all single signals are assumed to have the same commodity) edge from an exit node to sink node can be set to 0, which ensures that single signals do not have to travel together when they are far.

4.4.2. Constraints:

I) Conservation of flow: considering the initial flow $S_{c,i}$, the sum of incoming flow should be equal to the sum of outgoing flow for a given node n_i.

$$\sum_{c,j} X(e_{c,i,j}) - \sum_{c,k} X(e_{c,k,i}) = S_{c,i}$$

II) The amount of flow for each edge is capped by channel capacity.

$$\sum_{c} X(e_{c,i,j}) \le C_O$$

III) The maximum flow for each edge is zero if the edge is not selected.

$$X(e_{c,i,j}) \le C_O\ Y(e_{c,i,j})$$

IV) Real and binary constrain for variables.

$$X(e_{c,i,j}) \ge 0\ \ \forall e_{c,i,j} \in E$$

$$Y(e_{c,i,j}) \in \{0,1\}\ \ \forall e_{c,i,j} \in E$$

V) Crossing prevention: In order to prevent crossing, lazy constraints were used. Due to a large number of configurations that can define various types of crossings, it is impractical to implement all crossing constraints at runtime. Instead, lazy constraints are used, where at each iteration, the solution is scanned for possible crossing violations and only the constraint regarding the detected violation will be added to the model. Two types of crossings can be considered, which are termed as X-crossing, and L-crossing, and are illustrated in Fig. 11. In a nutshell, X-crossing would look like an X shape, where two flows cross at a given node, and L-crossing involves crossing while sharing a path.

X-Crossing: This type of crossing occurs at a given node when flows of 2 or more commodities arrive into a node and leave by crossing over each other. The algorithm to detect X-crossings is as the following:

```
For each node
  flows(c, e) = determine all flows for commodity c
  for every four edge from two commodities
    (e1, c1), (e2, c1), (e3, c2), (e4, c2)
    positions = assign numeric positions based on
      Fig. 12, and sort them using their positions
      (p1, c1), (p2, c2), (p3, c3), (p4, c4)
    If c2 and c3 are not the same commodity
      report X-crossing occurred
```

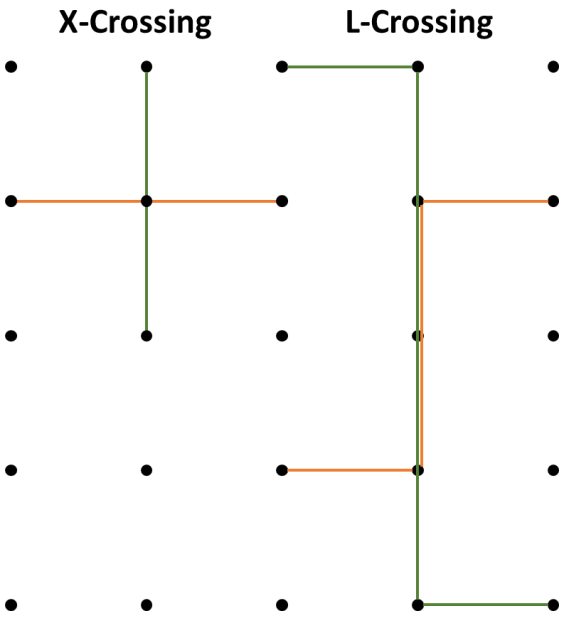

Fig. 11. Schematic of X-Crossing and L-Crossing scenarios.

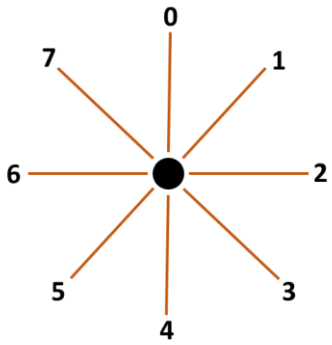

Fig. 12. Numeric labeling of edges for detecting X-crossings.

It must be noted that in the given algorithm, it is assumed that no two edges would occupy the same position, as occupying the same position requires checking for L-crossing, as discussed next.

L-crossing: This type of crossing occurs along a path. Thus, it cannot be detected only by analyzing a given node. Since in L-Crossing a part of the path is shared, only nodes exhibiting paths that are shared between two different commodities are being analyzed. The algorithm to detect L-crossing is as following:

For each node having shared outgoing edges between two commodities
  while following the path until outgoing edges are not shared
    record offset as a numerical position (Fig. 12) changes as the path changes direction
  collapse non-shared incoming edges and non-shared outgoing edges from the end of the path (while adding offset to their positions)
  use the X-crossing algorithm on collapsed path
  report L-crossing occurs, if X-crossing occurs on the collapsed path, or the paths are shared (i.e. not distinct)

In case any crossing is detected following lazy constraint will be added to the model:

$$\sum_{e_{c,i,j} \in E_C} Y(e_{c,i,j}) \leq 3$$

Where E_C is the collection of edges in X-crossing or non-sharing edges in L-crossing, for given commodities. Since 4 edges are involved in each crossing type, removing any edge would fix the problem, and hence the sum of 3 or fewer guarantees no such crossing would happen.

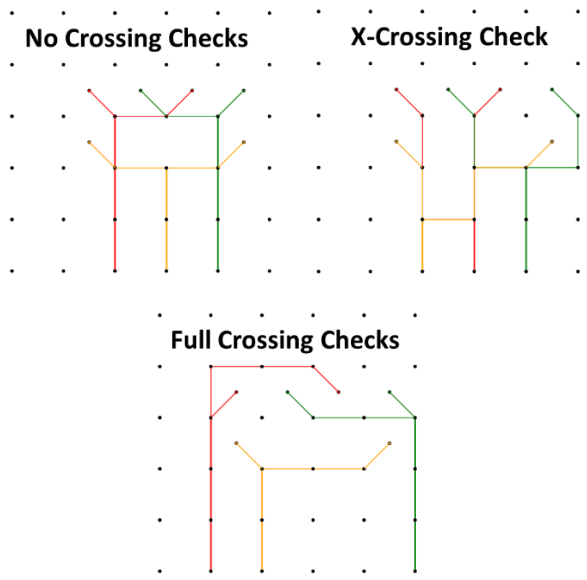

Fig. 13. Resulting differential pair escape routing patterns with and without crossing detection. Exit nodes located only at the bottom of the grid and channel capacities were capped by 2.

4.5 Results and discussion

Figure 13 shows the evolution of the pattern as crossing constraints is added. As it can be seen, without any crossing checks, all 3 pairs are crossing each other multiple time (all having X-type). Including X-crossing detection, removes all X-type crossing options, while the shortest paths now involve L-type crossings. Turning on both X- and L-type detections yields to a non-crossing pattern.

Figure 14, shows the escape routing pattern for a 30 × 30 grid and 14 pairs, with channel capacity of 2. Solving this problem takes 16 seconds, which is much slower than comparable problems reported in [6, 8]. Slower running time is mainly due to the implementation of lazy constraints and search for possible crossings which require a considerable amount of time at each iteration. However, the benefit of this approach is that the solution is produced in a single step (unlike other similar works [6-8]), where optimality of the solution is guaranteed. Furthermore, this approach utilizes the maximum capacity available for the routing and does not disregard shared legal routes between different pairs.

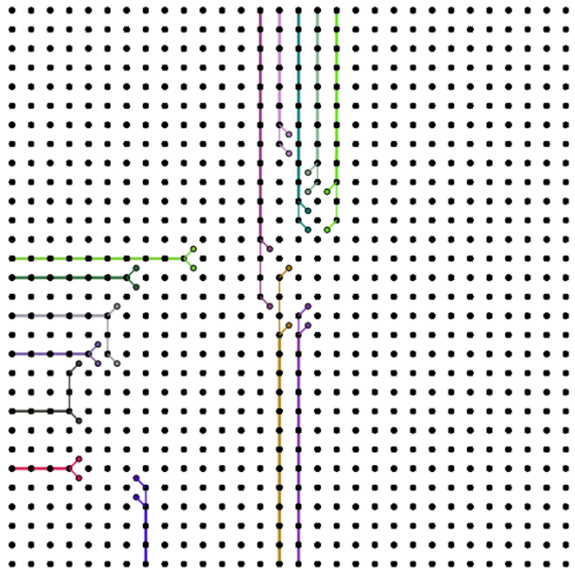

Fig. 14. Escape routing pattern for 14 differential pairs in a 30 × 30 grid, with a maximum capacity of 2.

5 Conclusion and future insights

In this work, two types of escape routing problems were solved. The first was the single signal minimum capacity escape routing of a full array of devices using a fast polynomial algorithm, and minimum length optimality guaranteed mixed integer programming approach. It was observed that the minimum capacity determined via both methods are in good agreement.

A future work can involve uniting the two methods together. This can be achieved by solving the problem via the polynomial time algorithm first, and then use the answer as an initial input for mixed integer programming model to further reduce the total length of the solution (if possible).

The second problem focused on the differential pairs escape routing and mixed single signal and differential pairs escape routing. It was shown that a mixed integer programming can be used to solve this type of problem in a single step to guarantee an optimal solution. Although it was found that the lazy constraints implementation hinders the running time, yet the model allows optimal usage of the resources available for routing.

Enhancements that can be considered for the differential pairs escape routing model include:

1) As mentioned earlier, to simplify the model and enhance the running time, a simple grid geometry of nodes was considered, which does not take into account the diagonal capacity. Therefore, future work can build upon this, and utilize the network graph presented in Fig. 8b, for multi-commodity flow. This would allow for consideration of diagonal capacity. However, due to the increase in the number of variables, the running time will be hindered even more.

2) Removing variable Y, and instead adding more constraints to simulate the effect of this variable. This change would lower the number of variables by half, which leads to the speed increase.

3) Faster algorithms for crossing detection, or even better removing them completely and

finding linear constraints that would disallow crossings.

4) Design of linear constraints that would allow transforming a multi-commodity flow problem into a single-commodity flow problem, reducing the number of variables greatly, and improving the running time significantly.

References

[1] S. J. Smith, J. S. Adams, S. R. Bandler, G. Betancourt-Martinez, J. A. Chervenak, M. E. Eckart, F. M. Finkbeiner, R. L. Kelley, C. A. Kilbourne, S. Lee, F. S. Porter, J. E. Sadleir, and E J. Wassell, "Uniformity of Kilo-Pixel Arrays of Transition-Edge Sensors for X-ray Astronomy," IEEE Transactions on Applied Superconductivity, 25 (2015) 2100505

[2] H Akamatsu, L. Gottardi, J. van der Kuur, C. P. de Vries, K. Ravensberg, J. S. Adams, S. R. Bandler, M. P. Bruijn, J. A. Chervenak, C. A Kilbourne, M. Kiviranta, A.J. van der Linden, B. D. Jackson, and S. J. Smith, "Development of frequency domain multiplexing for the X-ray Integral Field Unit (X-IFU) on the Athena," Proc. Of SPIE, 9905 (2016) 99055S-1

[3] T. Yan, M. D. F. Wong, "A correct network flow model for escape routing," Proceedings of the 46th Annual Design Automation Conference (2009), 332-335

[4] Y. K. Ho, H. C. Lee, and Y. W. Chang, "Escape Routing for Staggered-Pin-Array PCBs," IEEE/ACM International Conference on Computer-Aided Design (ICCAD) (2011), 306-309

[5] T. Yan and M. D. F. Wong, "Recent research development in PCB layout," Proceedings of IEEE/ACM International Conference on Computer-Aided Design, (2010) 398–403

[6] T.-H. Li, W.-C. Chen, X.-T. Cai, and T.-C. Chen, "Escape Routing of Differential Pairs Considering Length Matching," IEEE 17th Asia and South Pacific Design Automation Conference (ASP-DAC), (2012) 139 – 144

[7] K. Wang, H. Wang, S. Dong, "Escape Routing of Mixed-Pattern Signals Based on Staggered-Pin-Array PCBs," ISPD '13 Proceedings of the 2013 ACM International symposium on Physical Design, (2013) 93 – 100

[8] T. Yan, P. C. Wu, Q. Ma, M. D F Wong, "On the escape routing of differential Pairs," 2010 IEEE/ACM International Conference on Computer-Aided Design, ICCAD (2010), 614 - 620

[9] Gurobi Optimizer 7.5, available at http://www.gurobi.com/