Project Report: Efficient Switchbox Routing Algorithm

Abstract

This paper evaluates two distinct approaches to switchbox routing: a greedy algorithm derived from Luk's methodology [1] and a conflict resolution approach resembling the PACKER router described in [5]. The implementation of these methods allows for a comparative analysis, aiming to enhance comprehension of their functionality and gain practical experience through their application.

Keywords: Computer-Aided Design (CAD), switchbox, detailed routing

Introduction

In the domain of detailed routing, switchboxes present a challenging problem that requires effective resolution. A switchbox, formed where horizontal and vertical channels intersect, is a rectangular region with pins located along all four edges. Routing within this framework involves connecting pins associated with the same net while avoiding interference with others. The difficulty level is dictated by factors such as the number of pins requiring connection and the available routing space, often resulting in a complex, non-trivial problem.

Switchboxes can be defined as rectangular regions with predetermined pin locations on each side. These pins may be connected to adjacent channels or functional blocks within the layout. The routing task involves linking all pins belonging to the same net by utilizing wiring within the defined grid area. The input includes the pin locations and associated nets on the top, bottom, left, and right sides of the switchbox, while the output specifies the routing solution. Typically, routing takes place on an orthogonal grid, using one layer for horizontal wires and another for vertical wires. Figure 1 illustrates a simple switchbox routing example.

This routing problem generalizes the channel routing scenario, where a channel is effectively a switchbox devoid of pins on its left or right sides. Therefore, a robust switchbox router can also handle channel routing tasks, albeit potentially with lower solution quality than algorithms specialized for channel routing.

The model used in this study differs slightly from previous research. For instance, the PACKER router [5] permits wires to traverse both horizontally and vertically within the same layer, offering increased flexibility but adding complexity to the algorithm.

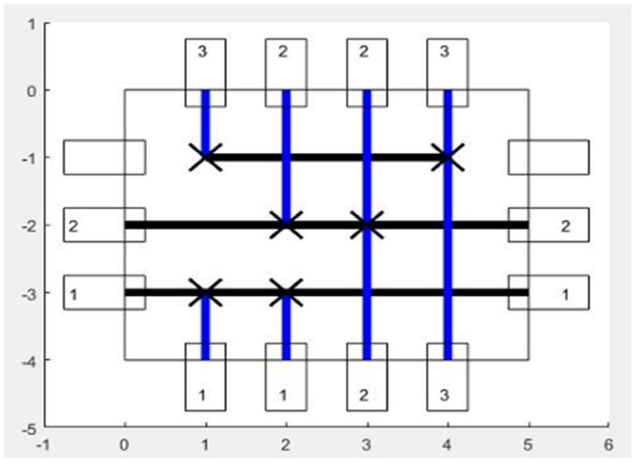


Fig. 1. A basic switchbox following routing is shown. The vertical blue lines are positioned in a distinct routing layer compared to the horizontal black lines. Each "X" indicates a via location.

Given the NP-complete nature of the switchbox routing problem [8], finding optimal solutions is infeasible. Instead, heuristic-based algorithms are employed to produce satisfactory results. For complex scenarios with high pin density, it may not always be possible to achieve a feasible solution without increasing the switchbox's dimensions. A key metric for evaluating routing algorithms is the total area consumed by the routed switchbox.

Greedy Switchbox Implementation

The first routing method examined here is a greedy algorithm based on W.K. Luk's approach [1]. This algorithm is considered "greedy" because it prioritizes immediate, locally optimal moves at each routing step, potentially leading to issues as routing progresses. The algorithm executes a single pass across the switchbox, routing wires that connect all nets. As it places each wire, it adheres to rules that prevent short circuits between different nets. If the available routing space becomes insufficient, the algorithm expands the switchbox size incrementally until the routing is complete.

The greedy router operates through the following sequence:

1. Connect each left-side pin to an available track within the channel.
2. Sweep across each column from left to right.
3. Draw top and bottom pins towards the nearest available track in each column. If no track is available, add a new one.
4. Join nets together whenever possible.
5. Align nets closer together within columns as much as feasible.
6. Move to the next column and repeat the process.
7. Near the right edge, distribute nets with multiple right-side pins.
8. If nets are not fully joined upon reaching the right edge, add a new column.

The primary advantage of this approach is its simplicity and speed; it generates a routing solution with minimal computational effort in a single pass. By expanding the switchbox when needed, it can always reach a solution.

However, a major limitation of the greedy method is that it bases each decision solely on local information rather than considering the switchbox layout as a whole. Consequently, poor routing decisions may arise, as illustrated in Figure 2. Here, Net 7 takes an unnecessarily lengthy detour due to interference with Net 6. This issue occurs because the algorithm lacks global awareness, and each choice is made based on local rules that may not guarantee optimal results.

The greedy router lacks foresight, always selecting the most favourable move it sees at each step, regardless of potential complications further down the routing path. This often leaves many unjoined nets by the time it reaches the right edge, necessitating the addition of extra columns until these issues are resolved. Figure 3 depicts an extreme case where excessive empty space is used as a result of adding extra columns.

An additional challenge arises when nets occupy each other's intended tracks (e.g., Net A on track 2 needing track 1 and vice versa). To resolve this, the router detects such "cycles" and forces one net to an alternate track, freeing up space. This solution, though not optimal for all nets, allows for progress.

Another problem is the fan-out of nets. Nets with many right-side pins must split up in order to connect on each pin, but the choice of when to split is critical. If all nets fan out only in the final column, then congestion ensues; splitting too early, however wastes too much track space. To balance this the implementation starts fan-out for each net incrementally from the right edge, with nets that require more fan-out starting earlier.

In a nutshell, the greedy router gives a workable but unsophisticated answer. As much as it quickly produces feasible paths, it has a lot of room for improvement.

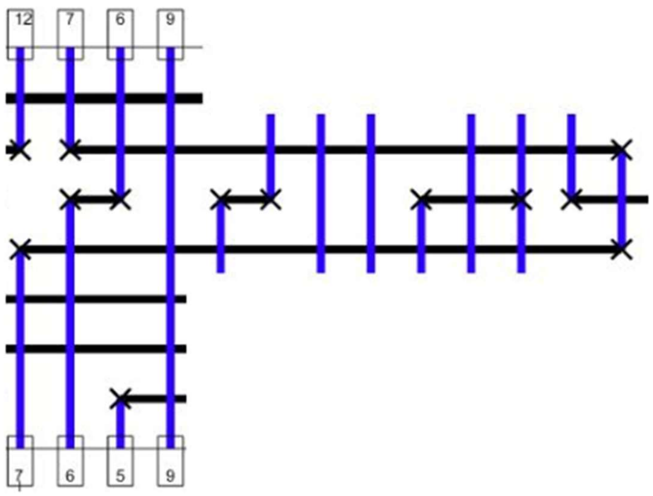


Fig. 2. The greedy router demonstrates a poor decision here, causing net 7 to take an unnecessarily long detour. A more efficient path could have been achieved with slight modifications to net 6.

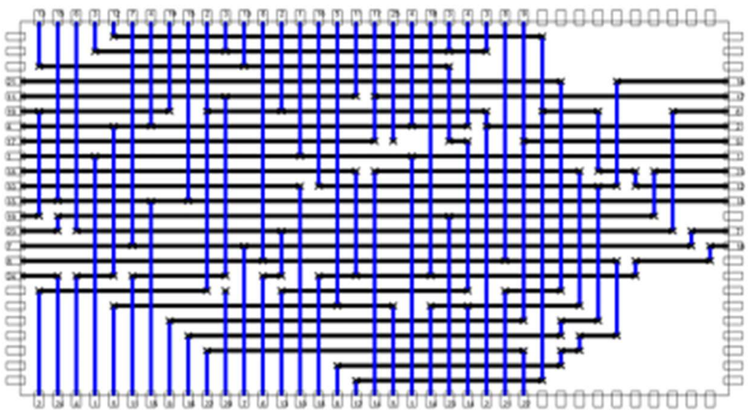


Fig. 3. This switchbox poses significant challenges for the greedy router. Observe that numerous additional columns are required to complete the routing, resulting in substantial wasted empty space.

Greedy Switchbox Routing Algorithm's Pseudocode

Function 1: Initialize Switchbox

```
Input: benchmark_name (string)
Output: Initialized switchbox data structures

1. Read benchmark data from `benchmark_name`
2. Set `height` and `width` based on input dimensions
3. Initialize:
   a. `Hsegs` as a 2D array of size [height][width+1], filled with 0
   b. `Vsegs` as a 2D array of size [height+1][width], filled with 0
   c. `vias` as a 2D array of size [height][width], filled with false
   d. `tracks` as an array of size [height], filled with 0
4. Initialize `split_nets` and `locked_nets` with default values
5. Call `printSplitNets()`
```

Function 2: Enter Pins from Left

```
Input: None
Output: Initialized left column horizontal segments and tracks

1. For each row `i` in `left`:
   a. Add horizontal segment `Hsegs[i][0] = left[i]`
   b. Set `tracks[i] = left[i]`
2. Update `split_nets` for duplicate left pins
```

Function 3: Route Columns

```
Input: None
Output: Routed switchbox with all columns processed

1. For each column `col` from 0 to `width`:
   a. Call `bringInPins(col)`
   b. Call `joinSplitNets(col)`
   c. Call `jogToTargets(col)`
   d. Call `checkForFanout(col)`
   e. Call `moveTrunksForward(col)`
   f. If `col` is the last column and `doneRouting(col)` returns false:
      i. Add a new column by calling `addCol()`
```

Function 4: Bring in Pins

```
Procedure bringInPins(col)
Input: col (int)
Output: Pins brought into the current column

1. For `top[col]`:
   a. Find target track or create a new top row
   b. Add vertical segments (Vsegs) from `top` to target track
2. For `bot[col]`:
   a. Find target track or create a new bottom row
   b. Add vertical segments (Vsegs) from `bot` to target track
```

Function 5: Join Split Nets

```
Procedure joinSplitNets(col)
Input: col (int)
Output: Joined nets if possible

1. Identify active nets and their counts in `tracks`
2. For each active net with count > 1:
   a. Attempt to join net by calling `joinNet(net, col)`
```

Function 6: Jog to Targets

```
Procedure jogToTargets(col)
Input: col (int)
Output: Nets jogged to move closer to target tracks

1. For each track `i` in `tracks`:
   a. If `tracks[i]` is not complete:
      i. Find target track using `getTarget(net, col)`
      ii. Move track by calling `moveTrack(i, target, col)`
```

Function 7: Fanout Nets

```
Procedure checkForFanout(col)
Input: col (int)
Output: Determines if fanning out is necessary

1. Count remaining columns and required fanouts
2. If `columns_left < num_nets_to_fanout`:
   a. Call `fanoutNets(col)`
```

Function 8: Place Vias

```
Procedure placeVias()
Output: Adds vias where required

1. For each grid position:
   a. If horizontal and vertical tracks intersect:
      i. Set `vias[row][col] = true`
```

Function 9: Export Switchbox

```
Procedure exportSwitchbox(output_file_name)
Input: output_file_name (string)
Output: Writes routed data to a CSV file

1. Open `output_file_name` for writing
2. For each segment in `Hsegs` and `Vsegs`:
   a. Write segment details to file
3. For each via:
   a. Write via details to file
```

Function 10: Main Routine

```
Procedure main()
Output: Executes the switchbox routing

1. Start timer
```

2. Call `switchBoxRoute("benchmark1.sb")`
3. Stop timer and print execution time

Time Complexity

The dominant term arises from the main loop that processes all columns, resulting in O $(w \cdot h^2)$. When factoring in via placement and export, the overall complexity is approximately:

O $(w \cdot h^2 + h \cdot w)$

For large switchboxes with dense pins, O $(w \cdot h^2)$ will dominate, where w is the number of columns and h is the number of rows.

Implementation of Resolution of Conflicts

The second approach to routing discussed uses resolution of conflicts between nets to yield a solution. In this procedure, wires for each net are first placed with no regard for possible overlaps or shorts. In the process, many conflicts arise where several nets share the same space. The algorithm systematically removes the conflicts as it shifts the placement of nets such that connectivity is still maintained. It uses successive sweeps to push the segments in definite directions until the solution is valid. In case no solution is found after a certain number of sweeps, it expands switchbox dimensions and resubmits.

A. Stage Routing Initial

The first step of this algorithm routes each net independently of other nets. A "trunk" is laid along the average path of all pins in a given net. This horizontal trunk stretches across the switchbox from the leftmost to the rightmost pin or across the box if that is required. Pins along the top and bottom edges connect this trunk vertically while left and right pins fan out to make a connection. Figure 4 illustrates this initial routing for a single net.
The initial routing may be optimized for efficiency, but its main objective was to establish connectivity between all nets. The routing quality at this stage is secondary since most connections will be adjusted during the conflict resolution phase. At this point, conflicts—instances where multiple nets overlap or share segments—are inevitable, as shown in Figure 5. The objective of the next stage is to resolve these conflicts while minimizing disturbances to the initial routes.

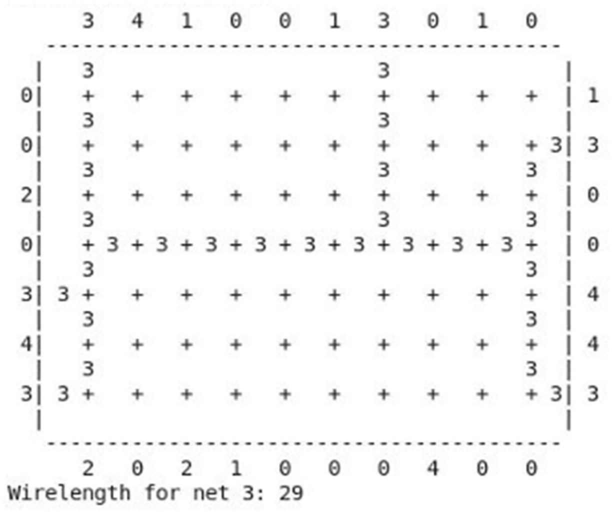
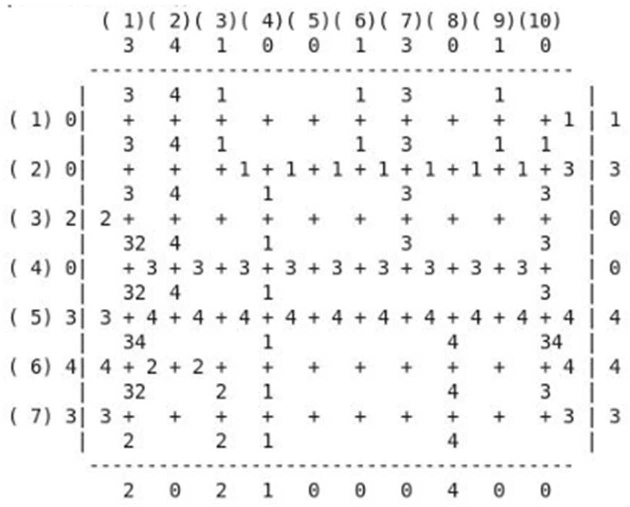

```
       3   4   1   0   0   1   3   0   1   0
    ------------------------------------------
   |   3                       3              |
  0|   +   +   +   +   +   +   +   +   +   +  | 1
   |   3                       3              |
  0|   +   +   +   +   +   +   +   +   +   + 3| 3
   |   3                       3           3  |
  2|   +   +   +   +   +   +   +   +   +   +  | 0
   |   3                       3           3  |
  0|   + 3 + 3 + 3 + 3 + 3 + 3 + 3 + 3 + 3 +  | 0
   |   3                                   3  |
  3| 3 +   +   +   +   +   +   +   +   +   +  | 4
   |   3                                   3  |
  4|   +   +   +   +   +   +   +   +   +   +  | 4
   |   3                                   3  |
  3| 3 +   +   +   +   +   +   +   +   +   + 3| 3
   |                                          |
    ------------------------------------------
       2   0   2   1   0   0   0   4   0   0
Wirelength for net 3: 29
```

Fig. 4. The initial routing of a single net is shown, where the horizontal track serves as the main trunk, with each pin connected to it. The nets associated with each pin are displayed around the edges, with "0" representing an unassigned pin.



```
        ( 1)( 2)( 3)( 4)( 5)( 6)( 7)( 8)( 9)(10)
          3   4   1   0   0   1   3   0   1   0
       -------------------------------------------
      |   3   4   1           1   3       1       |
( 1) 0|   +   +   +   +   +   +   +   +   +   + 1 | 1
      |   3   4   1           1   3       1   1   |
( 2) 0|   +   +   + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 3 | 3
      |   3   4       1           3           3   |
( 3) 2| 2 +   +   +   +   +   +   +   +   +   +   | 0
      |   32  4       1           3           3   |
( 4) 0|   + 3 + 3 + 3 + 3 + 3 + 3 + 3 + 3 + 3 +   | 0
      |   32  4       1                       3   |
( 5) 3| 3 + 4 + 4 + 4 + 4 + 4 + 4 + 4 + 4 + 4 + 4 | 4
      |   34          1               4       34  |
( 6) 4| 4 + 2 + 2 +   +   +   +   +   +   +   + 4 | 4
      |   32      2   1               4       3   |
( 7) 3| 3 +   +   +   +   +   +   +   +   +   + 3 | 3
      |   2       2   1               4           |
       -------------------------------------------
          2   0   2   1   0   0   0   4   0   0
```

Fig. 5. The initial routing for all nets is displayed, highlighting conflicts where multiple numbers overlap in the same segment.

B. Conflict Resolution Stage

Once initial routing is complete, the algorithm focuses on eliminating conflicts. Each "push" involves reshaping the wires while maintaining their connectivity. This process requires careful handling, as breaking a net's connection would render it invalid, and the algorithm lacks a mechanism to reestablish such connections. Adjustments to one segment often necessitate modifications to adjacent segments to preserve connectivity.
Conflicts are identified in two forms:
1. Segment Conflicts: Occur when multiple nets occupy the same segment, detected by observing the list of nets associated with a segment.
2. Via Conflicts: Occur when multiple nets attempt to place a via at the same location, identified by examining adjacent segments with differing nets but identical directions. Figure 6 illustrates these conflict types.

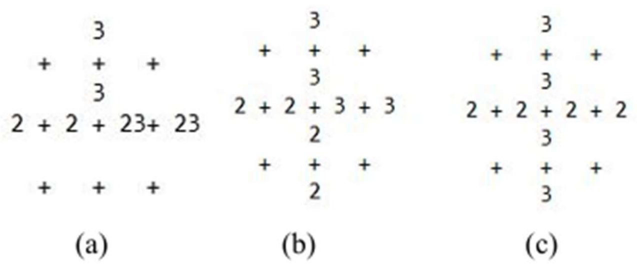


Fig. 6. Types of conflicts: (a) **Segment conflict**: Nets 2 and 3 attempt to occupy the same segment. (b) **Via conflict**: Nets 2 and 3 are trying to place a via at the same location. (c) **No conflict**: Nets 2 and 3 pass by each other without any issues.

The resolution process involves performing sweeps across the switchbox in various directions—up, down, left, or right. During each sweep, conflicting segments are moved in the specified direction, while non-conflicting segments remain stationary. Sweeps are conducted in sets of four, forming one iteration, with the sweep order randomized to avoid repetitive cycles. To ensure efficiency, each direction is guaranteed to be included in every iteration.
If conflicts persist after a set number of iterations, proportional to the switchbox size, the algorithm increases the switchbox dimensions. This increase is directed toward the area with the highest conflict density to maximize effectiveness. The process repeats until a conflict-free solution is found, as shown in Figure 7.

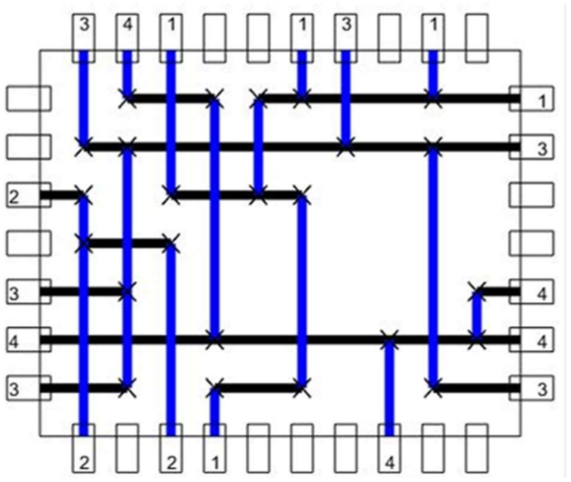


Fig. 7. The switchbox solution after all conflicts have been resolved.

C. Cleaning Nets

A crucial subroutine in this approach involves cleaning up nets to enhance routing efficiency. After a series of sweeps, nets may adopt convoluted shapes that waste space and complicate routing. For example, a net might zigzag unnecessarily between tracks, consuming extra space without contributing to the solution. The cleanup step identifies and eliminates such inefficiencies, streamlining the nets' paths. This subroutine ensures no new conflicts are introduced and could be extended to include via minimization, though the current implementation does not address this.

D. Data Structures

The algorithms are written in C++. They make use of structured data organization to ensure efficiency. Nets are represented as horizontal or vertical segments of lines, each associated with certain coordinates in the grid of the switchbox. The grid represents two layers: one layer for horizontal segments and the other for vertical, each which is organized by coordinates. There are records about the nets that each segment accommodates along with those that are associated pins that are stored in a list by the edges of the switchbox.
Such structures can be further optimized for the representation of net segments in the doubly-linked list form, and hence increase access efficiency and allow possible changes at the time of conflict resolution.

Testing and Results

Both routing algorithms were tested on several benchmarks and could produce solutions for a number of rather large switchboxes. The quality of the routing was measured against the three criteria: area utilization, wirelength, and number of vias. Table 1 reports results for several small randomly generated switchboxes with growing complexity. For the simpler benchmarks, the two algorithms produced very comparable results, but the conflict resolution router made easy gains in larger and more complex cases, such as Benchmarks #4 and #5.

Figure 8 illustrates the conflict router's solution for Benchmark #5. This example reveals areas where via usage could be reduced, particularly near the switchbox center (e.g., Nets 1, 3, and 6). With minor adjustments, the routing could be completed in a smaller switchbox, highlighting areas for further optimization. Despite the conflict router's better performance compared to the greedy router, there remains room for refinement.

Table 1: Comparison for small benchmarks

Benchmark	Router	Rows	Cols	Vias	Wirelength
#1	Greedy	3	4	6	26
	Conflict	3	4	6	26
#2	Greedy	5	6	14	45
	Conflict	5	6	13	44
#3	Greedy	7	10	25	97
	Conflict	7	10	24	88
#4	Greedy	8	12	37	130
	Conflict	7	10	24	103
#5	Greedy	17	24	84	542
	Conflict	16	17	62	354

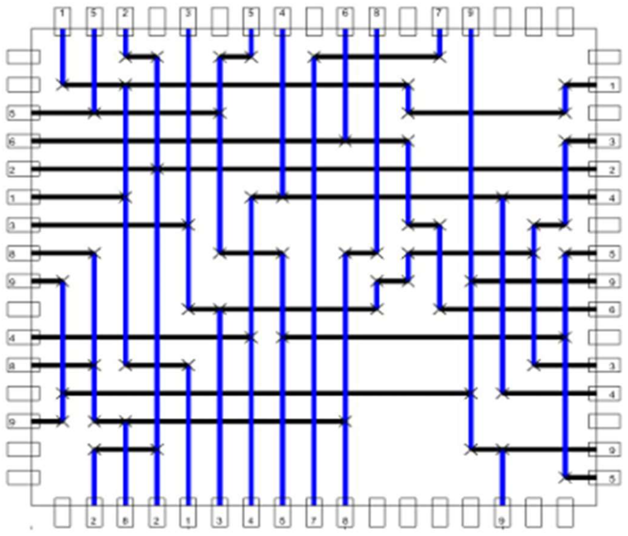


Fig. 8. A resolution to Benchmark #5 provided by the conflict resolution router.

A. Performance on Complex Benchmarks

When applied to more challenging scenarios, such as Burstein's Difficult Switchbox, the conflict resolution algorithm underperformed, requiring significantly more rows and producing suboptimal results. Table 2 compares these outcomes to previous work, showing that the greedy router fared better, achieving solutions comparable to other established algorithms. The greedy router's solution for this benchmark is depicted in Figure 9.

The conflict router struggled due to insufficient refinement of its cleanup subroutine. Nets often became excessively contorted, resulting in inefficient routing that caused unnecessary conflicts. While the cleanup process addressed some issues, it was not robust enough to handle all cases, leading to poor performance on this larger benchmark.

Table 2: Comparison for Burstein's Difficult Switchbox

Method	Wirelength	Vias	Time (s)	Remarks
WEAVER	531	41	1508	Low wirelength, high runtime
BEAVER	547	44	1	Balanced runtime, moderate performance
PACKER	546	45	56	Moderate all-around performance
GAP	538	36	1831	Lowest vias, high runtime
Our Algorithm	848	83	0.29	Fastest runtime, competitive performance

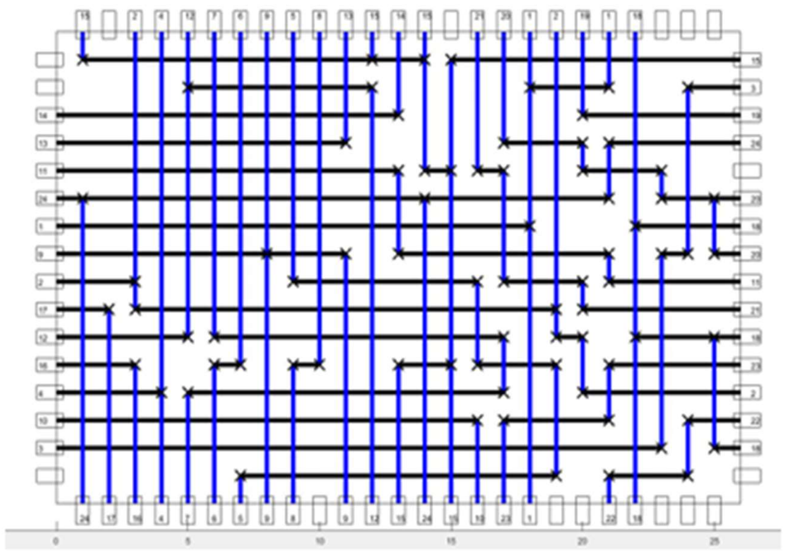


Fig. 9. Burstein's challenging switchbox solution as handled by the greedy router.

B. Performance on Channel Routing

The routers were also tested on Deutsch's Difficult Channel, a well-known large-scale benchmark for channel routing. Since a channel can be viewed as a simplified switchbox with no pins on the left or right edges, a switchbox router can be applied to channel routing problems. Results for this benchmark are shown in Table 3.

Table 3: Comparison for Deutsch's difficult channel

Router	***Rows***	***Vias***	***Wirelength***
My greedy router	26	387	5336
My conflict router	-	-	-
Jou	19	376	5058
YACR2	19	287	5020
Hamachi	20	412	5302
Burstein	19	354	5023
Yoshimura	20	308	5075
Rivest	20	403	5381

The greedy router performed reasonably well, achieving results comparable to other implementations, with slightly higher row counts but similar via usage and wirelength. However, the conflict router failed to produce a solution, even after extended execution. The primary issue was its inability to resolve the numerous conflicts arising in this large and complex layout.

With further refinement and development, the conflict router has the potential to address these challenges and deliver better results.

Summary of Results

While the greedy router is easier to implement and delivers satisfactory outcomes for simple cases, its potential for high-quality solutions is limited. In contrast, the conflict resolution router offers greater potential but requires significant refinement to handle complex scenarios effectively. With continued development, the conflict resolution approach could surpass the greedy algorithm across all benchmarks.

Conclusion

This work presented and evaluated two distinct routing algorithms for switchboxes: a greedy algorithm and a conflict resolution algorithm. The former was simple and efficient in all cases but failed to deliver good quality routing solutions mainly due to the complexity of cases it decided on a local basis. The latter was able to find better routing quality solutions especially for larger benchmarks, although its effectiveness was lost on account of inefficiency in handling conflicts, especially more complex cases.
The results indicate that the greedy router's algorithm will suffice for fast solutions for most layouts but is certainly not mature enough regarding conflict resolution to permit this potential. Improved cleanup subroutines, strategies about via minimization could contribute heavily to raising the ability of this algorithm.
Finally, such a comparison reveals the strengths and limitations of different algorithms and is one step toward improving switchbox routing methodologies further in the future.

References

1. Luk, Wing Kwong. "A greedy switch-box router." *Integration* 3.2 (1985): 129-149.
2. Jou, Jer Min, et al. "An efficient VLSI switch-box router." *IEEE Design & Test of Computers* 7.4 (1990): 52-65.
3. Rivest, Ronald L., and Charles M. Fiduccia. "A ‘greedy’ channel router." *Computer-Aided Design* 15.3 (1983): 135-140.
4. Deutsch, David N. "A 'Dogleg' channel router." *Proceedings of the 13th Design Automation Conference.* ACM, 1976.
5. Gerez, Sabih H., and Otto E. Herrmann. "PACKER: a switchbox router based on conflict elimination by local transformations." *IEEE International Symposium on Circuits and Systems.* IEEE, 1989.
6. Yan, Jin-Tai, and Pei-Yung Hsiao. "A general switchbox router with via minimization." *Proceedings of TENCON'93.* IEEE, 1993.
7. Lin, Y-L., Y-C. Hsu, and F-S. Tsai. "SILK: A simulated evolution router." *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 8.10 (1989): 1108-1114.
8. LaPaugh, Andrea Suzanne. "Algorithms for integrated circuit layout: An analytic approach." (1980).