Switchbox Router Implementation

Mark Sears
Department of Electrical Engineering
University of Texas at Dallas
Dallas, TX

***Abstract*— Two methods for routing switchboxes are examined in detail: a greedy switchbox router with an algorithm following Luk [1] and a conflict resolution switchbox router with an algorithm similar to the PACKER router in [5]. These two routers are implemented and compared with the intent of gaining insight into their function and experience from the process.**

***Keywords*— CAD, switchbox, detail routing**

I. Introduction

In the detailed routing process, switchboxes pose a difficult problem that must be solved. When a horizontal channel intersects a vertical channel, a rectangular switchbox is formed with pins on all four sides. Each net must be routed through the switchbox to maintain connections without interfering with other nets. Depending on how many pins must be connected and how much space is available, this can create a complex problem with no obvious solutions.

A switchbox is a rectangle with a defined set of pins on each side. These pins may be from adjacent channels, or pins from a functional block in the layout. The problem task is to connect all pins of the same net together by routing wires between them within the switchbox. The input to the switchbox problem is a list of pins with associated nets for the top, bottom, left, and right sides of the switchbox. The output is the wire routing that connects all of the pins. Routing is performed on an orthogonal grid with one layer for horizontal wires and one layer for vertical wires. An example of a simple switchbox is shown in Fig. 1. A switchbox is a more general case than the channel routing problem, because a channel is also a switchbox that has no pins on the left or right sides. Therefore, any switchbox router is also capable of channel routing, but might not reach as high quality of a solution compared to a dedicated channel routing algorithm.

It is worth noting that this model is slightly different than those used in some previous work. For the PACKER router in [5], wires in both layers are able to run either vertically or horizontally. This constraint is more flexible, and may allow for some higher quality solutions, but also increase algorithm complexity.

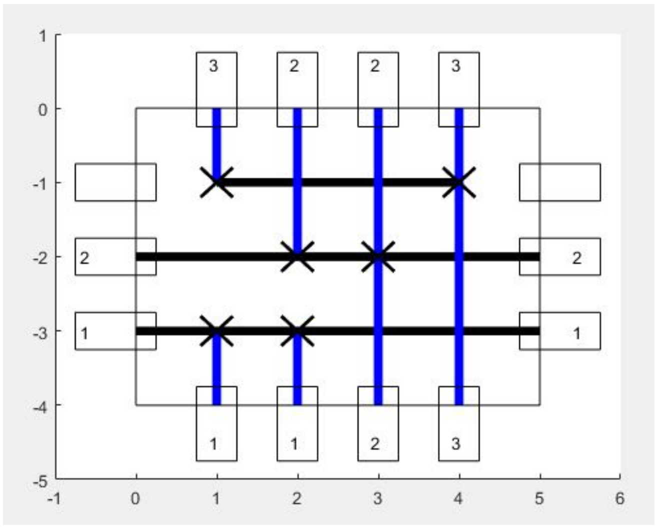


Fig. 1. A simple switchbox after being routed. Vertical blue lines are in a separate routing layer from horizontal black lines. Each "X" marks a via location.

Since there is no known method to find an optimal routing for switchboxes and the problem is NP-complete [8], routing algorithms must rely on heuristics to create a satisfactory solution. For any arbitrary switchbox with a fixed size there is no guarantee that a solution exists at all, especially if there is a high pin density. In order to find solutions, routers must be allowed to increase the switchbox size. A major criterion for comparing routers is the total amount of area used to route the switchbox.

II. Greedy Switchbox Implementation

The first router examined is a greedy algorithm described by W.K. Luk [1]. It is greedy because is always tries to take the "best" move at a particular spot, even though it might lead to problems in further routing. The router does a single sweep across the switchbox, filling in wire as it goes that connects all nets. As it places wire, it must follow rules to avoid shorting two nets together. If it runs out of room, it increases the size of the switchbox accordingly until routing can be completed.

The greedy router uses the following steps to place wires:

1) For each pin on the left side, bring it into the channel.
2) Sweep from the left side to the right side, for each column:
3) Bring in top and bottom pins to the closest available track. If no tracks are available, add a new track.
4) Join nets together if possible.
5) Jog nets towards each other as much as possible.
6) Move to next column and repeat.
7) When close to the right edge, fanout nets that have more than one right pin.
8) If the right edge is reached and nets haven't been joined, add a new column.

This greedy approach has advantages in that it is easy to implement and generates a solution to the problem very quickly in a single pass. It can guarantee a solution since it continues increasing the switchbox size until all necessary wiring can be placed.

A major drawback of this approach is that each decision is made based on local information at a point, and does not consider the switchbox as a whole. It is easy to find a bad routing decision made by this greedy router. Fig 2. shows a net routing that could easily be improved. Net 7 is brought into tracks very close to each other, but cannot be joined due to net 6. Net 7 takes a very long detour in order to join up with itself, but this detour could have been easily avoided if only net 6 had been placed one track lower. This issue arises because the greedy router does not see the whole picture when it makes these decisions. Each decision is made locally based on a set of rules, and no matter how carefully crafted those rules are there can always be nets that are poorly routed.

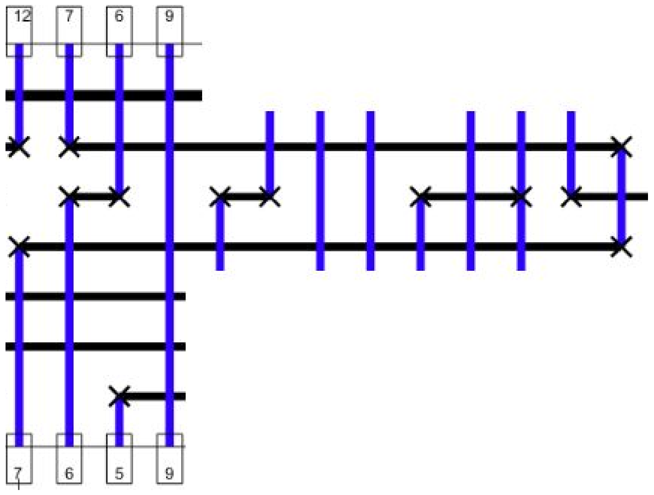


Fig. 2. A bad decision made by the greedy router. Net 7 goes on a long detour, when it could take a much more direct route with only minor adjustments to net 6.

The greedy router does not have any foresight, it always takes the "best" move that it sees at the point it is at, without any regard for what problems it might be causing further down the channel. Often times the router will reach the end of the channel and still have many nets which haven't been joined or placed in the correct track to meet their right pins. In this case, it has no choice but to add more columns until it is able to resolve all such issues. Fig. 3 shows an egregious example of this, needing many extra columns to be added.

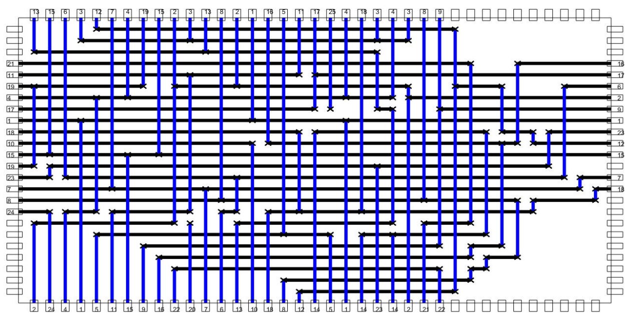


Fig. 3. A particularly troublesome switchbox for the greedy router. Notice that many columns need to be added in order to complete the route, and how much empty space is wasted as a consequence.

One tricky scenario occurs when nets happen to be sitting in each other's preferred tracks. That is, net A is on track 2 but needs to be on track 1, and net B is on track 1 but needs to be on track 2. Since each net cannot move to the correct position, they both continue on the wrong track. This can occur with more than 2 nets, with each net occupying the track needed by another. This is solved by detecting a "cycle" and forcing one of the nets out of the way to an unused track, even though it may not benefit the net immediately.

Another interesting issue is that of fanning out. When nets have more than one pin on the right side, they must split at some point to meet all right pins. But how soon should they split? If all nets wait until the last column to start fanning out, they will conflict with each other and many extra columns will need to be added. If they fan out too soon, they take up valuable track space unnecessarily. The rule of thumb used by this implementation is to start fanning nets out one column earlier for each net that needs fanning out. For example, if 5 nets need to be fanned out to meet right pins, then these nets will start trying to fanout 5 columns before the edge is met.

Overall, the greedy router does its job. It is a quick and dirty method that generates valid switchboxes, and has a relatively simple implementation. The solutions are valid, but it is easy to see that there is plenty of room for improvement.

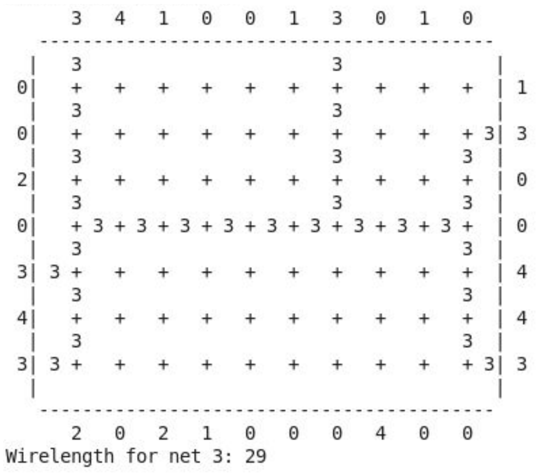
III. Conflict Resolution Implementation

The second type of router examined relies on resolving conflicts between nets to find a solution. The routing starts by placing wires for all nets without regard for short circuits. Each net is wired independently, which usually results in many conflicts arising where more than one net is trying to occupy the same segment of space. The algorithm attempts to resolve all conflicts between nets. It accomplishes this by pushing nets around while maintaining net connectivity after each move. Sweeps are performed to push segments in a given direction, and this is repeated until a solution is found. If no solution is found after a set number of iterations, then the switchbox size is increased and the sweeps begin again.

A. *Initial Routing Stage*

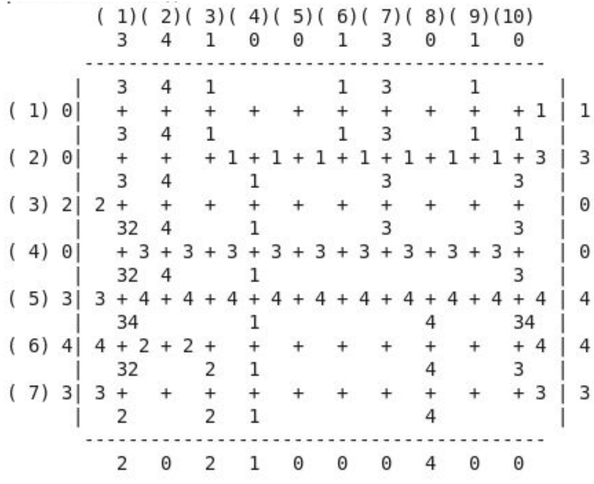
As the first step, each net is routed without regard for other nets. A trunk is placed in the track that is the average of all pins on a given net. The trunk extends from the left-most pin to the right-most pin, or simply across the entire switchbox if pins are on both left and right edges. Next, all the pins on the net are connected to the trunk. Pins on the top and bottom edges are simply run vertically to the trunk. Left and right edge pins are fanned out to make the connection. An example of initial routing is shown in Fig. 4. This initial routing could likely be improved upon with some effort, but since the conflict resolution will displace most of these segments, it would likely not have much impact on the success of the full algorithm. The main goal of this initial phase is to obtain net connectivity, with routing quality being a secondary objective.

This process is repeated for all nets in the switchbox. Note that no consideration is made for other nets during this initial routing, and so conflicts are inevitable, as seen in Fig. 5. At this point, each net is close to a minimum routing. The goal of the next stage is to fix all the conflicts between nets while perturbing each net as little as possible from this initial state.



```
     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. Initial routing of a single net. The horizontal track is used as the trunk, and each pin is connected to the trunk. Nets that each pin belong to are shown around the perimeter, with "0" indicating an empty pin.



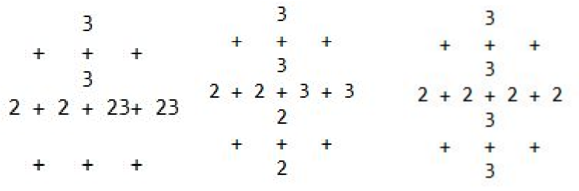
```
        ( 1)( 2)( 3)( 4)( 5)( 6)( 7)( 8)( 9)(10)
          3   4   1   0   0   1   3   0   1   0
        -----------------------------------------
      |   3   4   1               1   3       1       |
( 1) 0|   +   +   +   +   +   +   +   +   +   + 1 | 1
      |   3   4   1               1   3       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. Initial routing for all nets. Conflicts can be seen where multiple numbers appear in the same segment.

B. *Conflict Resolution Stage*

Once all nets have been initially routed, the algorithm must push them around to remove all conflicting segments. Each "push" involves transforming the shape of the wires while still maintaining connectivity. Great care must be taken to ensure that nets are not broken, because in this implementation there is no mechanism to connect them again. When a segment is pushed, it often must modify adjacent segments as well to maintain connectivity. All possibilities need to be considered. Since each net maintains connectivity, it means that when all conflicts are resolved the resulting routing will be a solution to the switchbox.

Conflicts are detected in one of two ways: a "segment conflict" occurs when more than one net is occupying the same segment, which can be detected simply by checking the size of a segment's list of nets. A "via conflict" can occur when two nets want to place a via in the same place (a via is placed whenever a net switches from horizontal to vertical segments). This can be detected by looking for adjacent segments with the same direction but a different net. Fig. 6 illustrates these conflict types. If the routing does not contain any segment conflicts or via conflicts, then the routing is a valid solution to the switchbox.



```
      3
  +   +   +
      3
2 + 2 + 23+ 23

  +   +   +
```

(a)

```
      3
  +   +   +
      3
2 + 2 + 3 + 3
      2
  +   +   +
      2
```

(b)

```
      3
  +   +   +
      3
2 + 2 + 2 + 2
      3
  +   +   +
      3
```

(c)

Fig. 6. Types of conflict:
(a) segment conflict: nets 2 and 3 try to occupy the same segment.
(b) via conflict: nets 2 and 3 are trying to put a via at the same point.
(c) nets 2 and 3 pass by each other without conflict.

The pushes are accomplished by a series of sweeps across the switchbox. A sweep can occur in any direction: up, down, left, or right. During each sweep, segments are pushed in the direction of the sweep to remove conflicts. Segments which are currently not causing conflicts are left alone, but segments causing a conflict are moved.

Sweeps are performed in sets of four, which forms an iteration. For each iteration, the order of sweep directions is shuffled to avoid being caught in a fruitless cycle where nets are pushed back and forth in the same positions. One aspect that was explored was performing each sweep in a random direction while still guaranteeing that each direction was performed at least once.

Some switchboxes may be difficult or impossible to solve in the space allotted. In this implementation, a number of iterations were performed proportional to the switchbox size:

$$\mathit{Max\ Iterations} = 10 * \mathit{width} * \mathit{height} + 500$$

If the max iterations is reached, the switchbox is increased in size and the iteration count is reset. In practice this gives the algorithm enough attempts to find solutions, but at some point it must give up and try in a bigger box. The extra +500 attempts was added to ensure that very small switchboxes did not prematurely increase in size due to a very low number of max iterations.

The switchbox can be increased in size on any of the four edges. The algorithm attempts to choose the side that has the most conflicts nearby, so as to hopefully make it easier to resolve the highest number of conflicts. This process of increasing the switchbox repeats indefinitely until a solution with no conflicts is found. Fig. 7 shows one such example

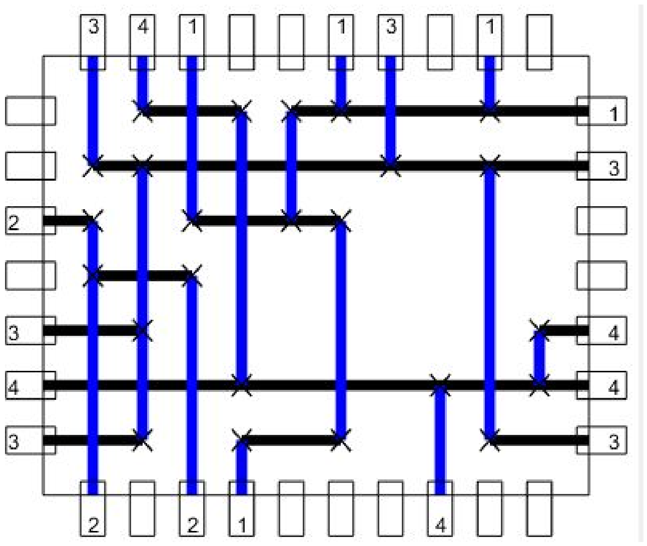


Fig. 7. Switchbox solution after resolving all conflicts

C. Cleaning Nets

An important subroutine for this router is to clean up nets by removing structures in the wiring that are space inefficient. After an iteration of sweeping, a particular net remains connected but may be contorted into strange shapes that make routing more difficult. For instance, a net wiring which moves up a track, then immediately back down a track makes no progress and takes up extra space. This kind of unhelpful shape can be avoided using a clean up step.

During this step, each net is examined in turn for shapes which can be removed to shorten the overall net. Usually this makes nets have a more direct path to their targets. Care must be taken to not unintentionally undo the conflict resolution steps, so the cleanup step only performs moves that do not cause any new conflicts.

This cleanup could be expanded to include via minimization. Currently my implementation does not attempt to minimize the number of vias after conflicts are resolved, but this would surely improve the final result.

D. Data Structures

The algorithms were implemented in C++. Careful organization of data is paramount to efficient execution of any algorithm. The wires to route each net consisted of a set of line segments, either horizontal or vertical, located at a point within the switchbox. The switchbox was represented as a grid of segments that could be accessed by a coordinate system. For example, to represent a 10x10 switchbox there would be a 10x10 set of horizontal segments and a 10x10 set of vertical segments. Each segment also has a net(s) associated with it. Each side of the switchbox has a vector containing the pins of that edge.

After working with these algorithms, insight can be gained into what data structures might be more suitable for the specific problem. One possibility that may be worth exploring for the conflict resolution method is to organize the segments of a net as a doubly-linked list. This would allow quick access to all adjacent segments and would make modification to net wiring more efficient.

IV. Testing and Results

Both of the routers implemented were tested on many benchmarks, and successfully generated solutions to many switchboxes. The quality of the routing can be determined by the area, wirelength and number of vias used. Table 1 shows results for some small, randomly generated switchboxes. These are easy examples of increasing complexity to show basic functionality of the routers. For very small benchmarks, the two give nearly identical solutions. On larger benchmarks such as #4 and #5, the conflict resolution router shows significant improvement over the greedy implementation.

Fig. 8 shows the resulting solution produced by the conflict router for benchmark #5. In this routing, it can be seen where vias could be minimized towards the center of the switchbox (especially nets 1, 3, and 6). Through manual inspection, it can also be seen that with some simple manipulation the routing could be completed in a smaller box. If routing was slightly improved, it would not need as many columns. Even though the conflict router appears to do a much better job on this benchmark, there is still obvious room for improvement.

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

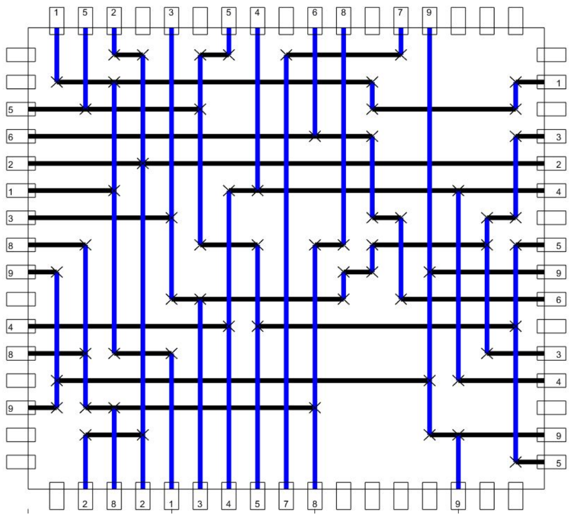


Fig. 8. A solution to Benchmark #5 from the conflict resolution router.

Unfortunately, the performance of the conflict router drops significantly on larger more complex switchboxes. The benchmark for Burstein's Difficult Switchbox did a very poor routing job, requiring twice as many rows before the conflict router found a solution at all. Table 2 shows results for this benchmark compared to other previous work. The greedy router was much better at routing Burstein's with comparable results to other routers. This solution is shown in Fig. 9.

The reason the conflict router performs so poorly on this more complex switchbox is that it has not been refined sufficiently. I believe that the cleanup subroutine for nets is important, but has not been tested enough to work well on larger benchmarks. During routing, nets become contorted into long snaking shapes which are inefficient and take up lots of space. The cleanup corrects some of these, but it is not refined enough to correct all instances. The result is that nets become a winding mess that cause many more conflicts than necessary, which yields a very poor routing.

Table 2: Comparison for Burstein's Difficult Switchbox

Router	***Rows***	***Vias***	***Wirelength***
My greedy router	16	77	640
My conflict router	31	251	1115
Jou	15	52	547
Weaver	15	41	531
Luk	15	58	577
Sadowska	16	59	561
Hamachi	15	67	564
Hsieh	15	63	567

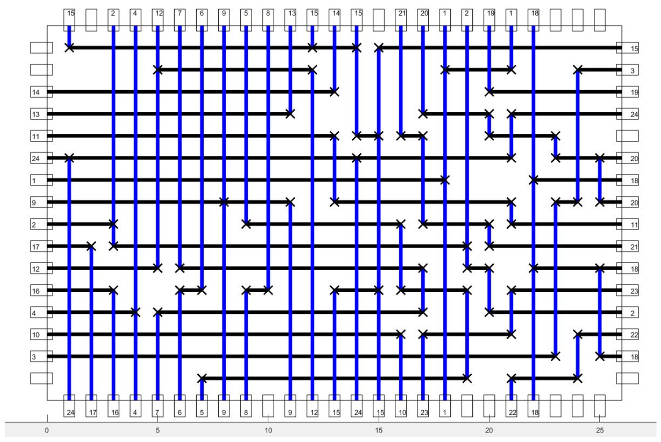


Fig. 9. Burstein's difficult switchbox solution from the greedy router.

Deutch's difficult channel is a very large example of a channel, not a switchbox. However, since the switchbox problem is merely a generalization of the channel problem, a switchbox router is able to route any channel (a channel can be viewed as a switchbox with no pins on the left or right). Results comparing my result for Deutch's to other published works is in Table 3. The greedy router does fairly well, with a similar number of vias and wirelength to other implementations. The number of rows is somewhat larger, but I believe quite good for a first attempt at router implementation.

The conflict router has thus far been unable to complete a route for the difficult channel. Given the problems it has with larger examples, the router cannot resolve all the conflicts even after hours of execution time. With more time and effort the conflict router could be refined to achieve a solution.

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

While the greedy router was a much easier algorithm to implement and yields acceptable results, it has a much lower potential than other routers. Other types of routers, once refined, can achieve better routing quality. My conflict router was a much more challenging algorithm to implement, and I believe that with more time and effort it could achieve better results than the greedy algorithm on all switchboxes.

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 Region 10 International Conference on Computers, Communications and Automation*. Vol. 1. 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).