EECE6086 - HW 2

Max Thrun — Samir Silbak

March 24, 2014

1 Objective

The objective of this lab is to implement placement and routing algorithms capable of placing and routing the given net lists into the smallest square area possible.

2 Implementation Details

2.1 Placement

2.1.1 Placement Grid Dimensions

One of the main goals of this project is to have a final layout that is as close to a perfect square as possible. We call this the `squareness` of our layout which is the ratio of width to height (`squareness = width / height`). The closer this value is to 1 the 'more square' the layout is. The main issue with achieving this is that we do not know up front how tall our layout will end up since the channels between the cells are variable and the width and height of the grid directly effect them.

In order to figure out what width and height we should make our grid we need to come up with some heuristic that can estimate the squareness of a layout given only the placement grid dimensions, number of cell, and number of nets.

$$squareness = f\left(num\_cells, num\_nets, grid_w, grid_h\right)$$

Figure 1: Squareness Function

Once we have an equation that can estimate the squareness we can plug in different grid dimensions and figure out one that gets us the best squareness. In order to figure out the formula that estimates our squareness we manually specified various grid dimensions for each benchmark and recorded the resulting squareness. This data can be found in the Appendix section at the end of the report. We then used the program Eureqa to evolve an equation that fit our data. After running Eureqa for 15 minutes we had an equation that fit our data with an $R^2 = 0.9965$.

$$squareness = 0.341 + \frac{2.41}{grid_h} + grid_h\, 0.62^{grid_w} + \frac{0.303\, num\_cells}{grid_h^{\,2}} - 0.000162\, num\_nets - 0.303\, num\_cells\, 0.62^{grid_w}$$

Figure 2: Squareness Equation

To calculate the placement grid size we start the grid as a perfect square by taking the square root of the number of cells. We then decrease the height and increase the width and check the squareness of this size. We continue doing this and keeping track of which size results in the best squareness until the height is 0.

The table below shows the resulting squareness of the original method (just making a perfectly square grid) versus using the equation. We can see that by using the equation we achieved a much more square layout for all but benchmark 9. This could be a result of not having enough data points to fit a better curve with.

	Without Equation			With Equation			
Benchmark	**Grid Width**	**Grid Height**	**Layout Squareness**	**Grid Width**	**Grid Height**	**Layout Squareness**	**Percent Improvement**
1	7	7	0.7245	9	5	1.0789	19.66 %
2	10	9	0.7365	12	8	0.9845	24.80 %
3	14	13	0.6767	18	10	1.0914	23.18 %
4	19	19	0.7029	28	13	1.2579	3.92 %
5	27	27	0.6125	45	16	1.2060	18.16 %
6	32	32	0.6060	56	18	1.1735	22.05 %
7	23	22	0.6452	39	13	0.9823	33.71 %
8	30	30	0.6306	50	18	1.3123	5.70 %
9	29	28	0.6343	45	18	1.4323	-6.67 %
10	25	24	0.6715	36	17	1.2772	5.13 %

Table 1: Improvement from using squareness estimate equation

2.1.2 Force Directed Placement

We use a force directed placement algorithm in order to reduce the overall complexity. Our algorithm follows the one in the book exactly with the exception of what happens when the target cell is locked. In the original algorithm you search for a vacant spot to place the seed cell if the target cell is locked, we expand that to also include unlocked spots. If we choose an unlocked spot then that cell we are replacing becomes the new seed cell. We found through experimentation that this achieves less complex placements. The gray cells are cells with no connections.

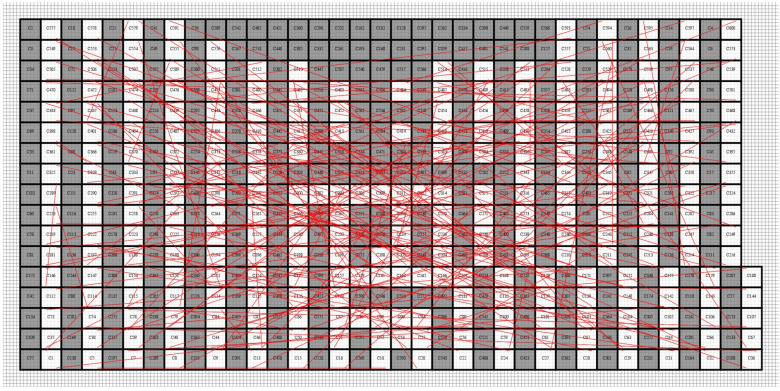

Figure 3: Original

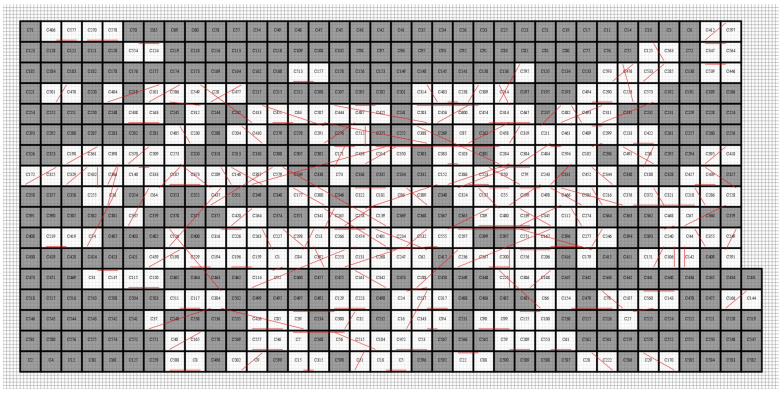

Figure 4: Force Directed

In order to figure out the best abort limit we recorded the layout area for each benchmark for various limits. We found the abort limit that achieved the minimum area and used that minimum area to normalize the areas of the others as a percent increase. We found that an abort limit of 1 gives us the lowest average percent increase in area across all the benchmarks.

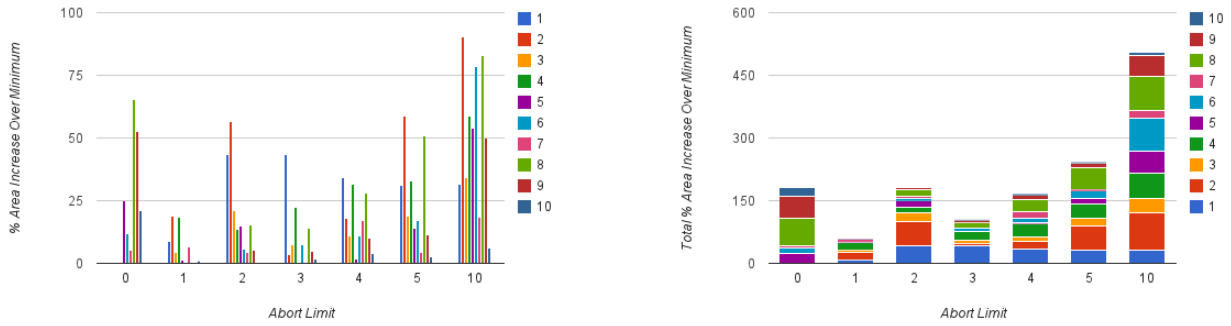

Figure 5: Percent increase in layout area versus abort limit

2.1.3 Flipping Cells

The force directed algorithm does not take into account the positions of the terminals on each cell, it assumes all connections go to the center of the cell. This results in a lot of connections that unnecessarily span vertically across a row. To fix these we simply go through each cell and try all possible flip orientations and see which one reduces the wirelength the most. This approach is not ideal as it does not take into consideration chains of cells that all may need to be flipped. An approach similar to KL where we walk down the connection tree trying flips and only actually flip as many as what improves the total wirelength would be interesting. Unfortunately we did not have time to explore this.

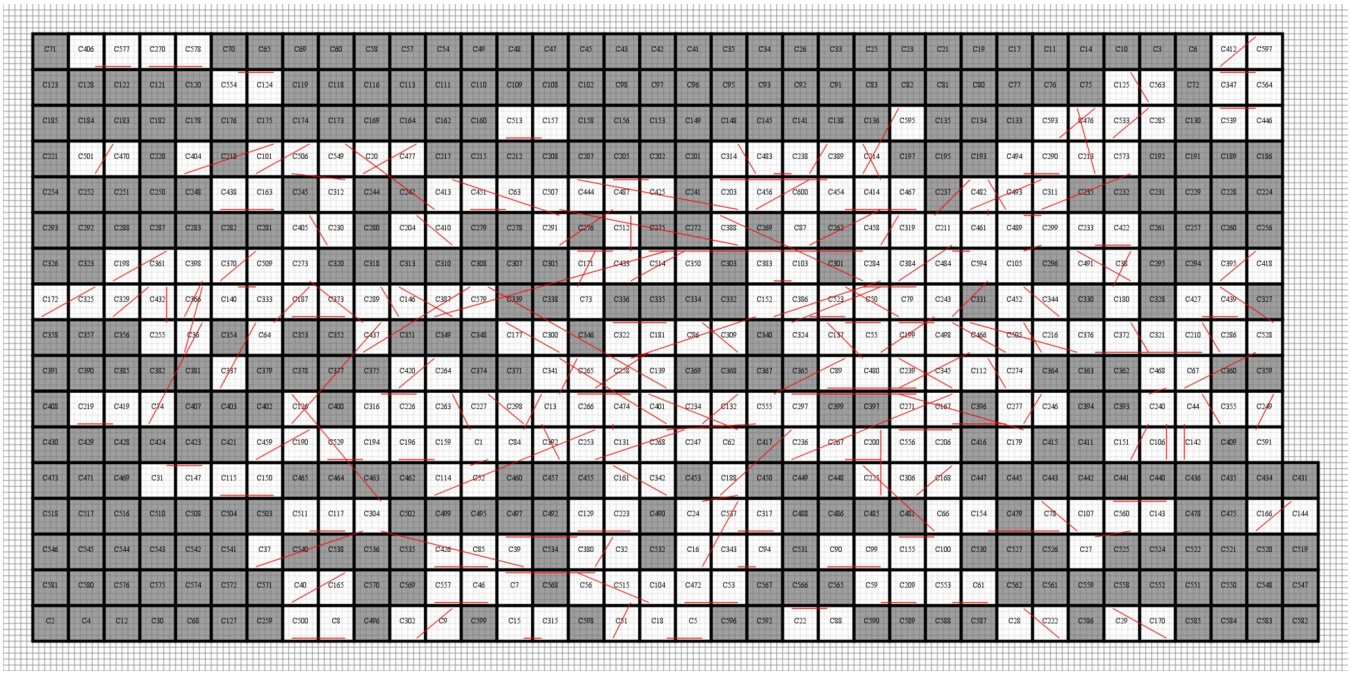

Figure 6: Original

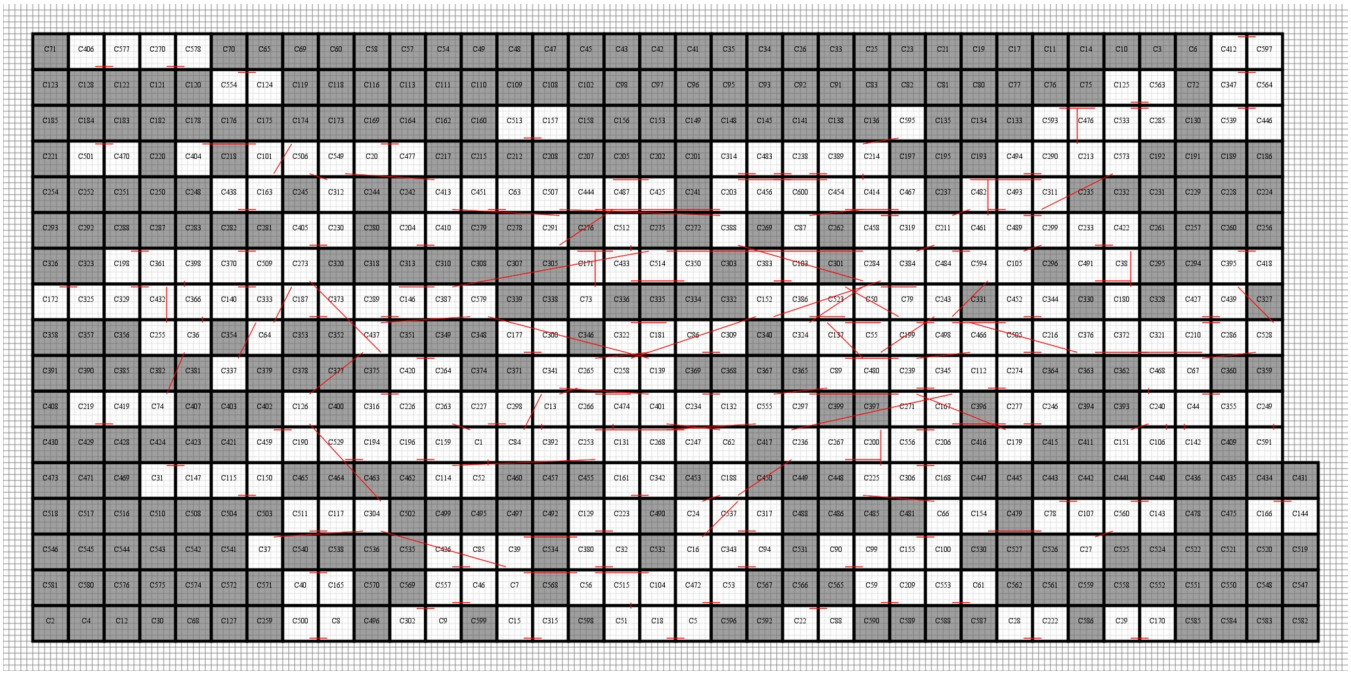

Figure 7: Flipped

2.1.4 Adding Feed-Through Cells

In order to channel route we cannot have connections that span rows. To fix this we add feed-through cells (shown in light gray) in order to transfer the nets between rows. The process of adding feed-through cells starts at the bottom row. It goes through each terminal of each cell and detects if the terminal is in the same channel as its destination terminal. If it's not, it figures out if it needs to add a feed-through to the current row (say the terminal is at the bottom of the cell and it needs to connect to somewhere above it) or if it needs to add the feed-through to the next row up (if the terminal is at the top of the cell but it connects to the top of the cell above it). If a feed-through cell is added the whole row is rescanned since we might need to add yet another feed-through cell for the feed-through we just placed.

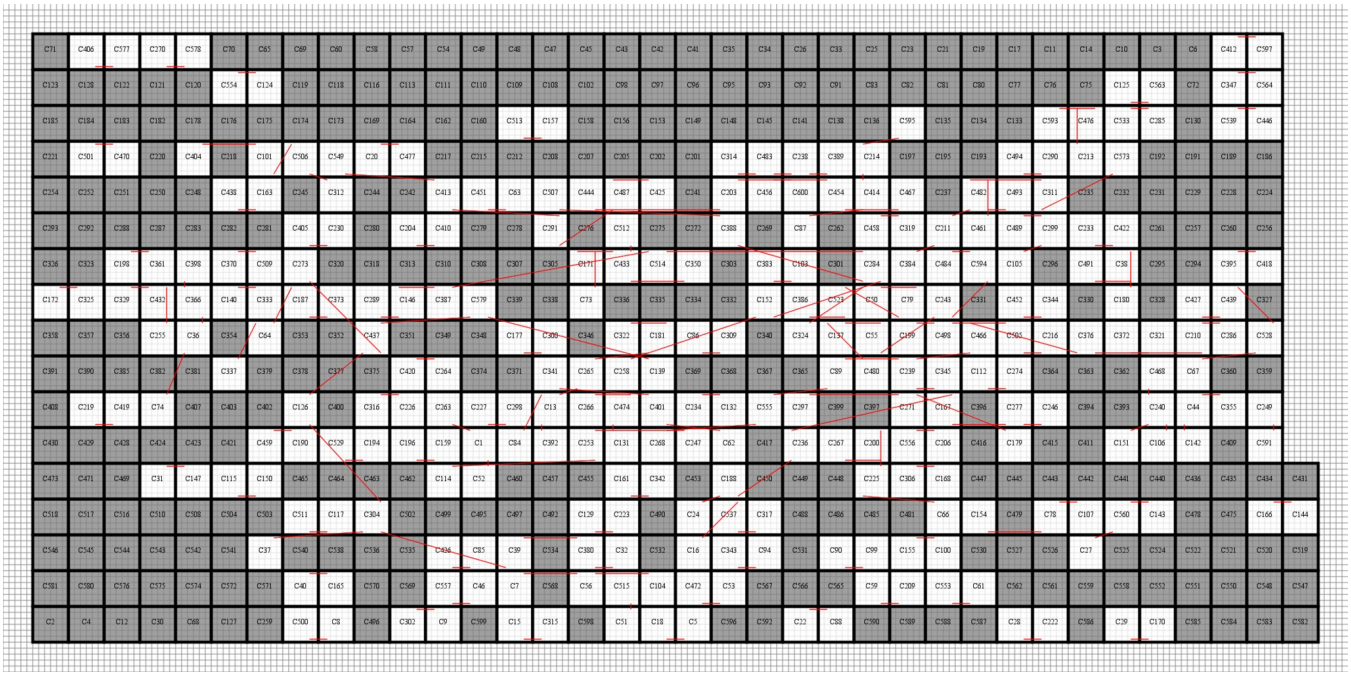

Figure 8: Original

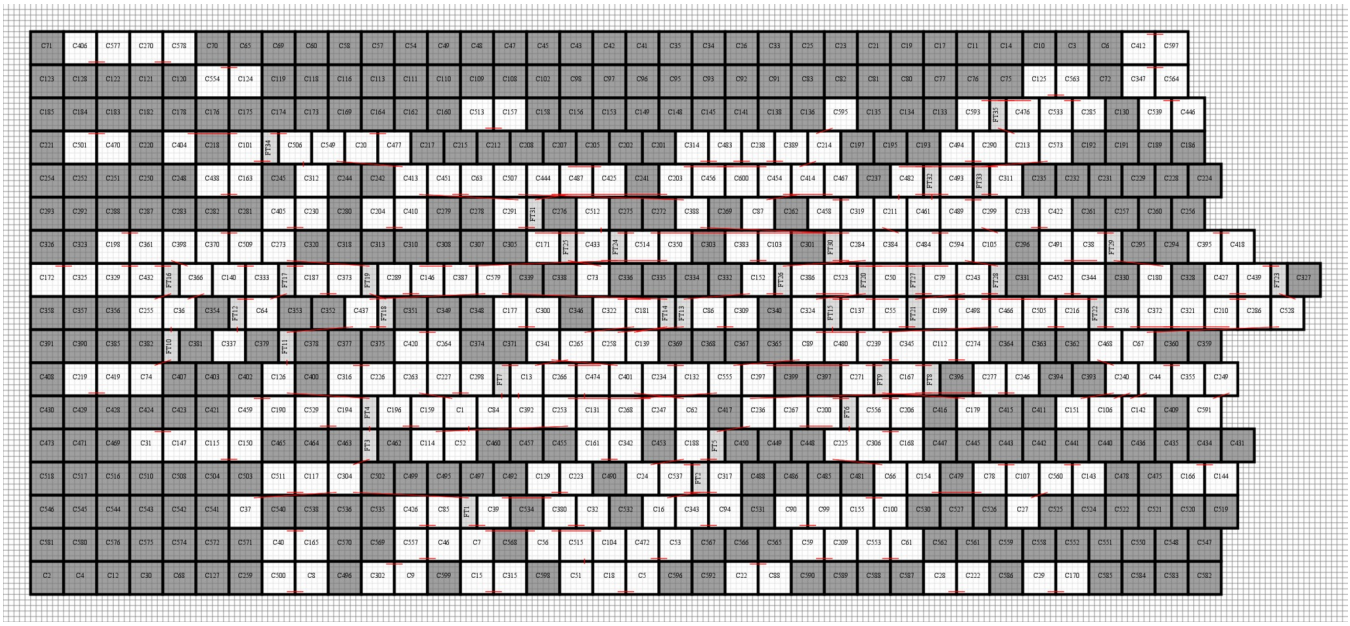

Figure 9: With feed-throughs

2.1.5 Evening Row Lengths

After adding the feed-through cells the row lengths become uneven. In order to fix this we add all unconnected cells to a queue and then go through them adding them to which ever row is currently the shortest. Additionally, we alternate placing them on either side and we also skip every other cell. The reason behind this is that since we just messed up a lot of the force directed work by removing all the unconnected cells we will have to re-force direct them within their rows. Spacing them out will allow them to more easily move around and have a higher chance of finding a vacant spot close to their target.

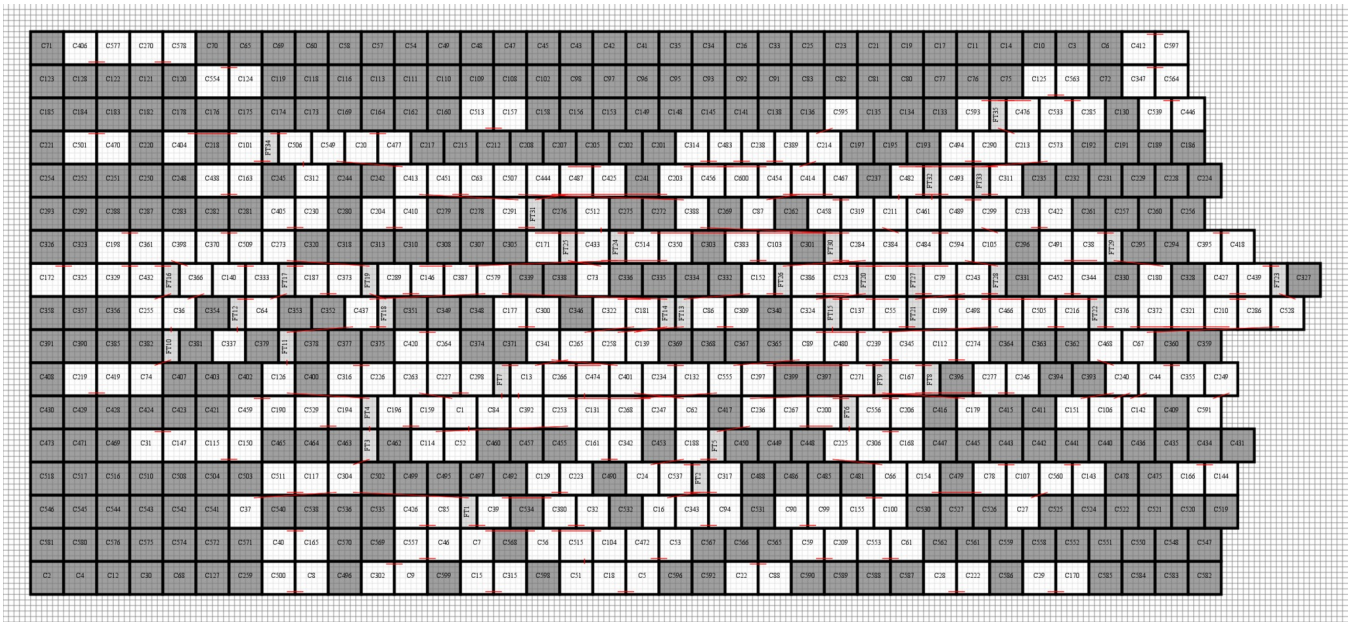

Figure 10: Original

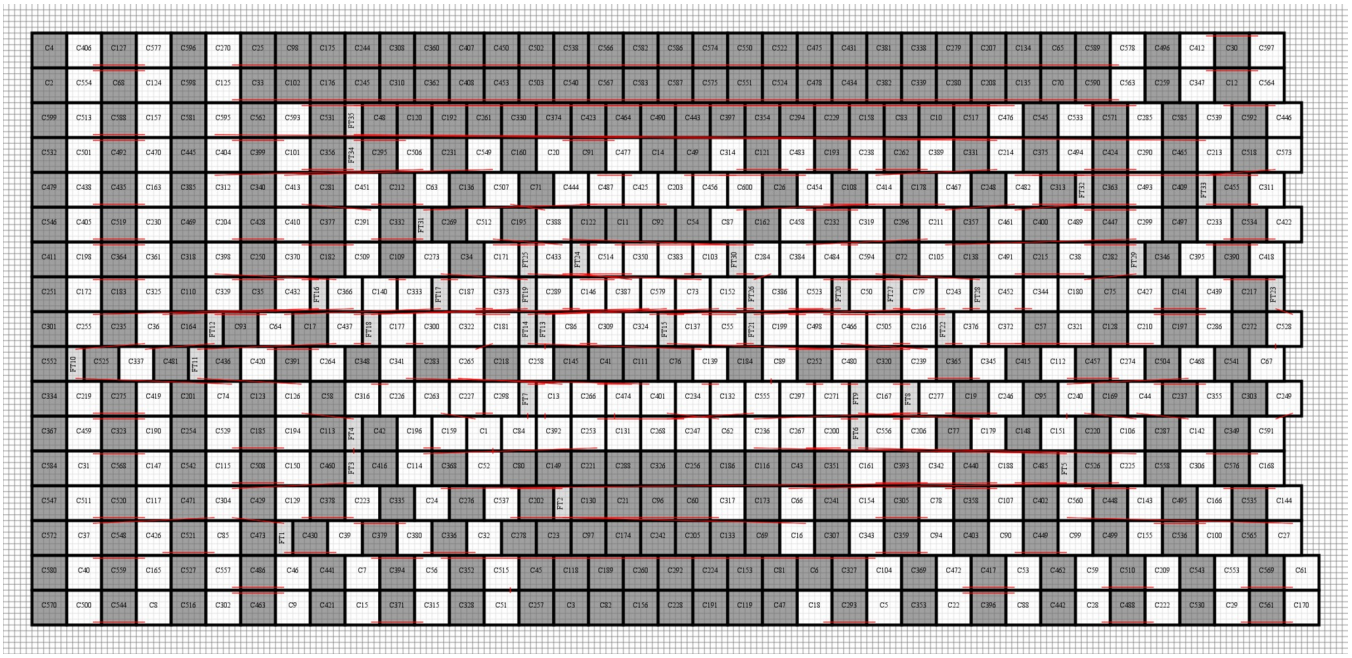

Figure 11: Even length rows

2.1.6 Pulling Cells Closer

After evening out the row lengths we are left with a lot of space between connected cells. In order to fix this we run a simplified force directed algorithm which goes through each cell, calculates its zero force location on the x axis, and then finds the closest unconnected cell to that location and swaps its position with it.

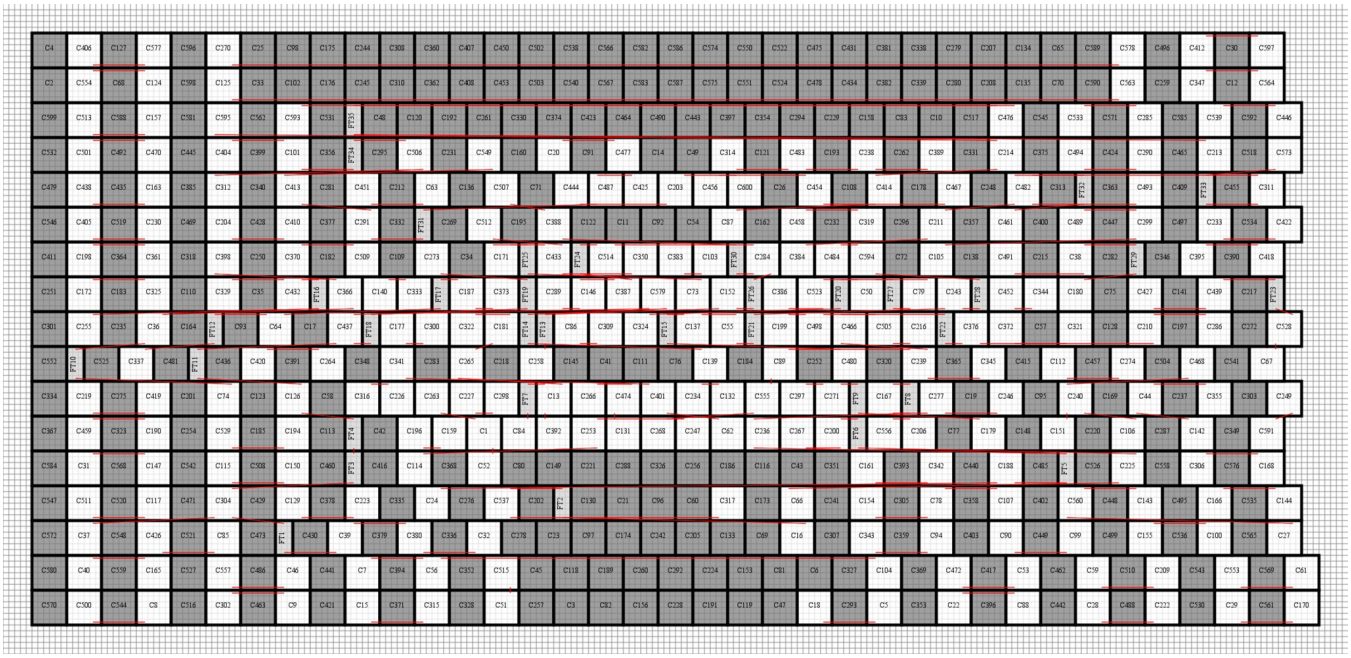

Figure 12: Original

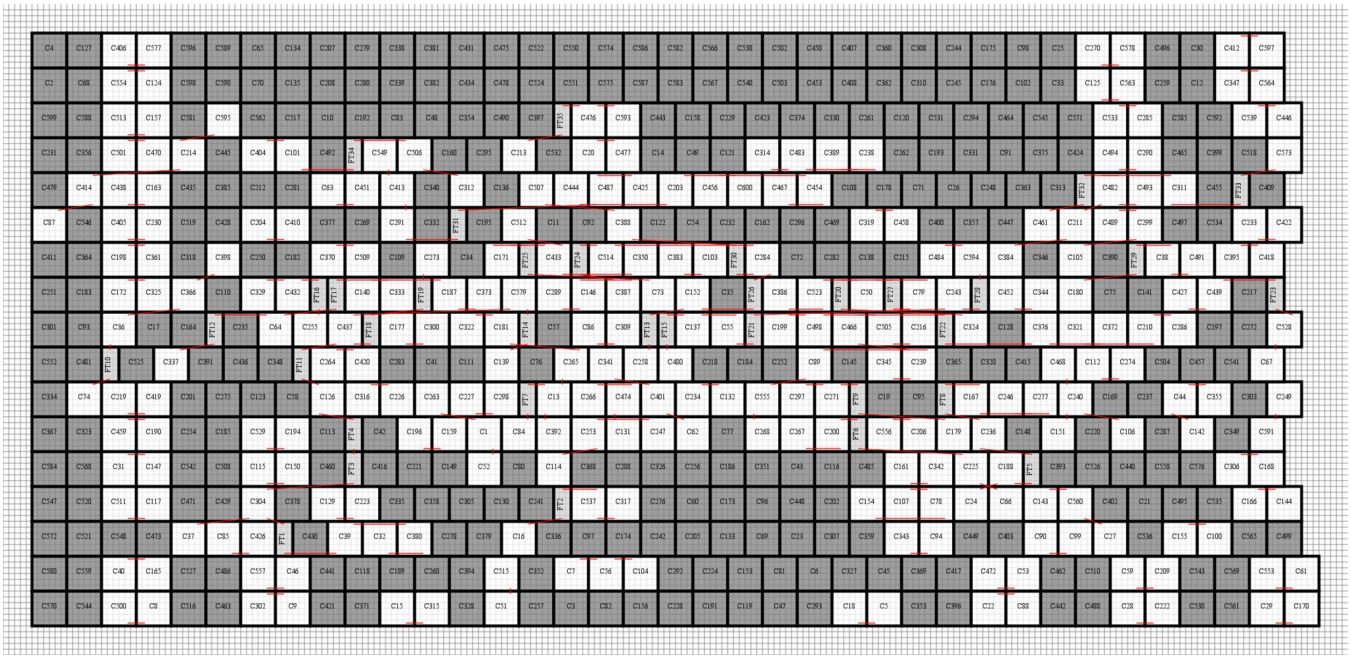

Figure 13: Cells pulled closer

2.1.7 Moving Feed-Through Cells

Sometimes, because it can not be know at the time of placement, the feed-through cells end up on the far side of the cell as shown in the 'before' picture. We make an effect to detect these and move them closer. We always want the feed-through cell to horizontally between the two terminals it is connected to. (This is done before routing begins, the example images below show the circuit post routing as it better illustrates the situation)

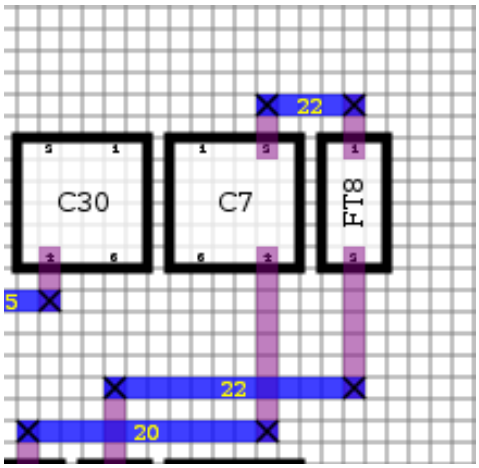

Figure 14: Before

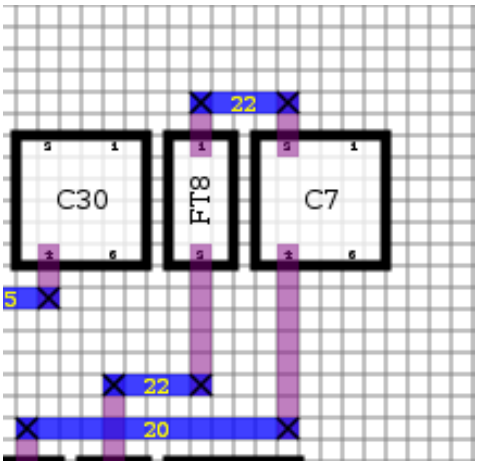

Figure 15: After

2.2 Routing

2.2.1 Assigning nets to track

Routing begins by first assigning each net to its own track. Net collisions are not considered at this point.

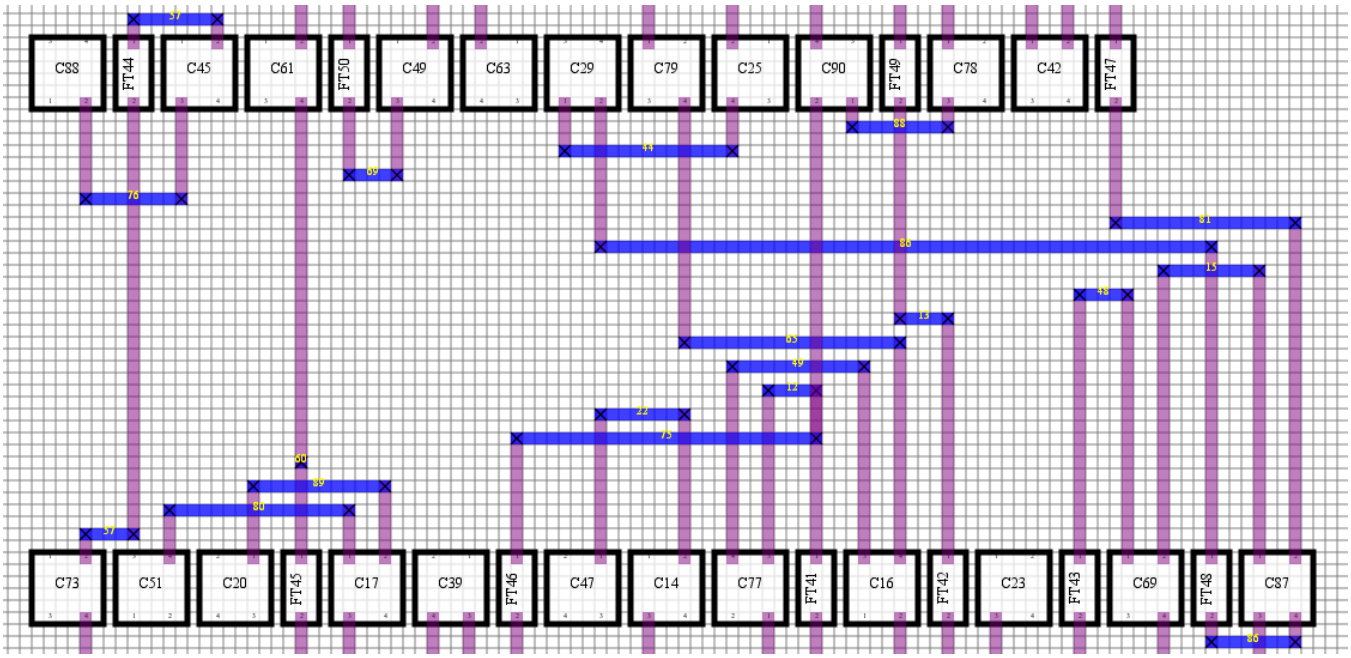

Figure 16: Each net has its own track

2.2.2 Expand the tracks

Each net is checked to see if it conflicts with another net. Some conflict configurations can be solved by adding another track to the channel (expanding the channel). The image below shows the same circuit at the previous image but with the conflict now solved and circled in red.

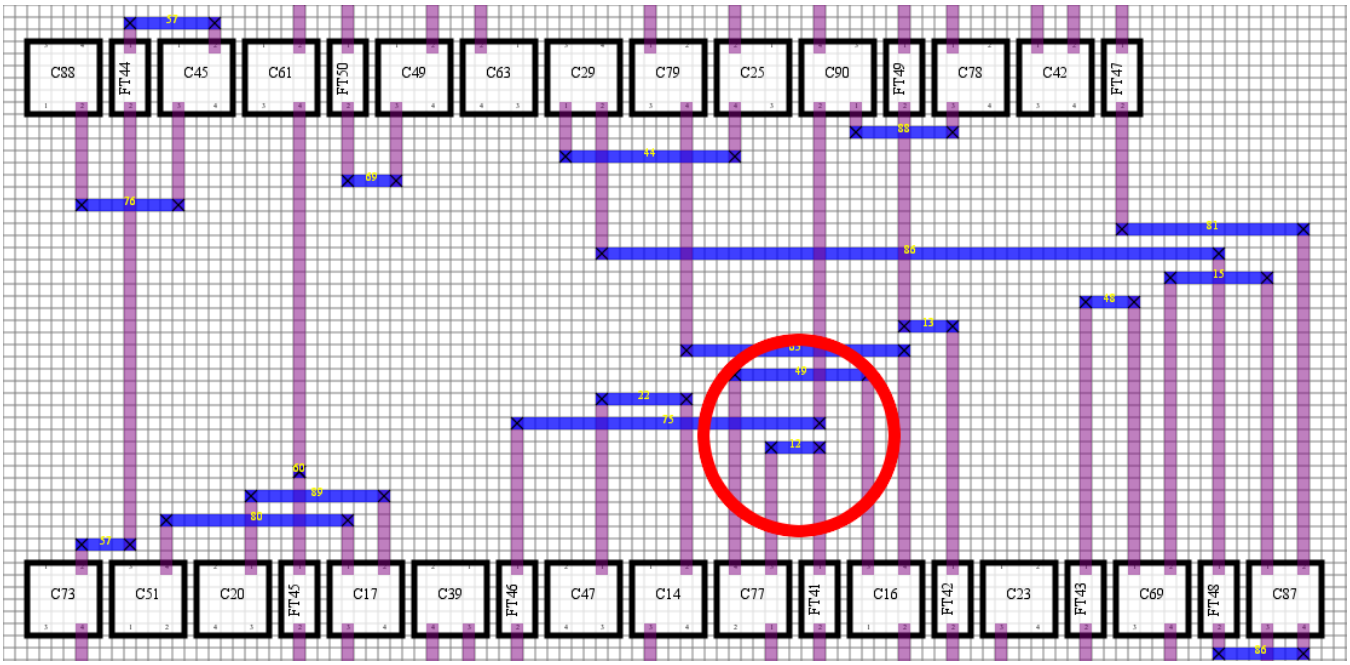

Figure 17: Conflict is solved by expanding the channel

2.2.3 Fixing overlaps

Some conflicts are true overlaps that cannot be solved no matter what net ordering you try. The only way to solve these conflicts is to move the cells in the rows over. The image on the left shows an overlap between nets 11 and 29. The only way to solve this is to move cell C36 over to the right, which is shown in the image on the right.

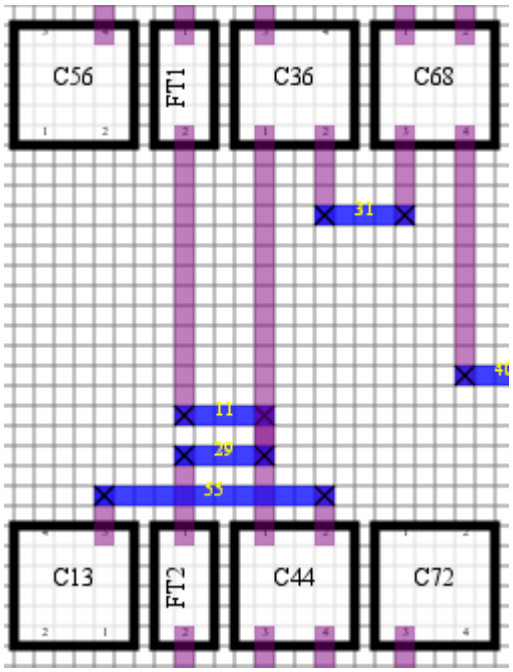

Figure 18: Overlapping nets

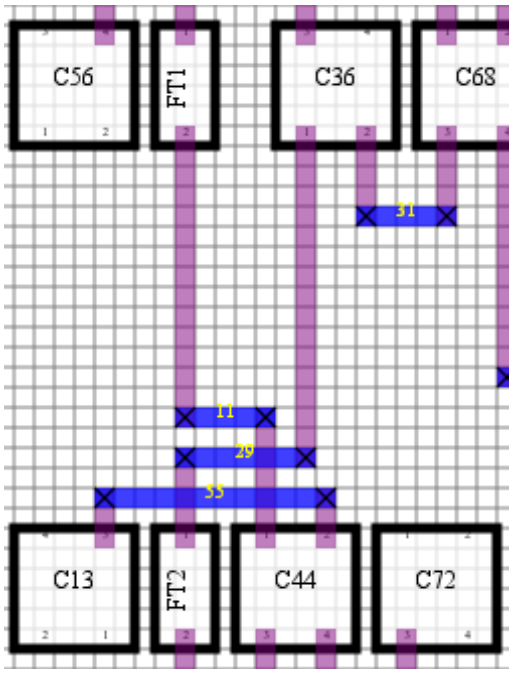

Figure 19: Conflict is solved by moving cells over

2.2.4 Finding Vertical Nets

For terminals that line up directly with their destination we can use a purely vertical net that doesn't take up a track. The image on the left shows a small horizontal segment that is currently occupying a track. We simply mark this net as purely vertical and remove it from any future track related operations.

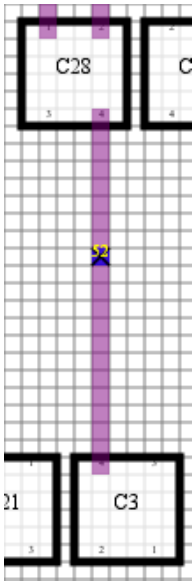

Figure 20: Vertical net taking up a track

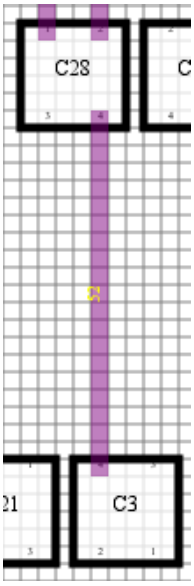

Figure 21: Unnecessary horizontal segment removed

2.2.5 Pulling Nets Up

We next want to try to shrink the channel so the first step is to pull all the nets either up or down. Nets are pulled up as far as possible without causing a conflict. We do not consider any specific ordering about which nets we pull up first although it can effect the final configuration. The image below shows the result of pulling all the nets up and then removing tracks which not longer have any nets on it.

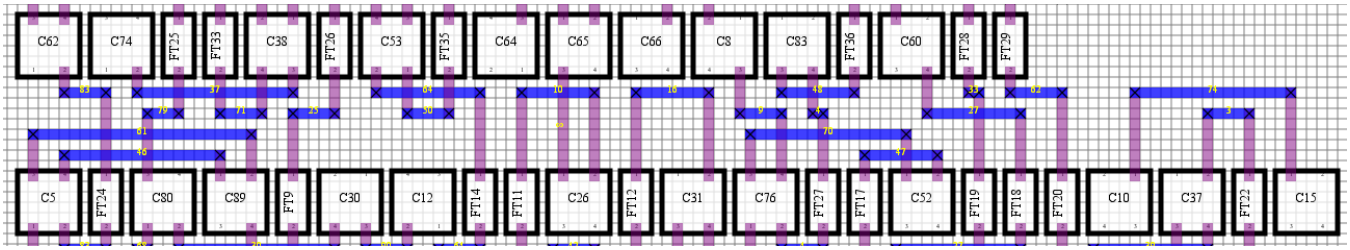

Figure 22: All nets pulled up and empty tracks removed

2.2.6 Pulling Nets In

For nets whos terminals are both on the same side of the channel we want to bring that net back to that side (pull it in to the side the terminals are on). The image below shows the result of doing that.

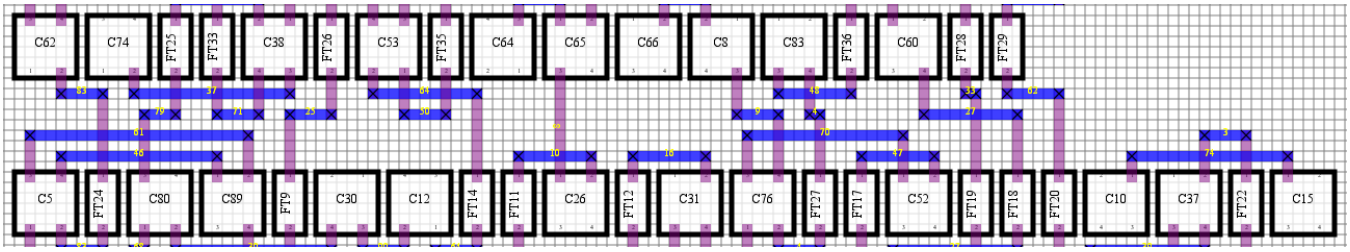

Figure 23: Nets pulled in closest to the cells the connect to

In our actual implementation we pull the cells up, then in, then down, then in again. This routine seems to be able to achieve fairly high channel density.

3 Usage

Building is accomplished via a `Makefile` which generates an executable names `main`. Additional build opions are shown below.

```
$ make              # regular build
$ make clang        # builds using clang++ instead of g++
$ make debug        # enable debug printfs
$ make clean        # removes all binaries, object files, and benchmark results
$ make benchmarks   # runs all the benchmarks in the 'benchmark' directory
$ make pngs         # converts all the benchmark .svg files to .png
$ make jpgs         # converts all the .png files to reduced resolution .jpg files
```

The program accepts a single argument which is a file containing the netlist formatted as specified in the assignment. All log and debug output is printed to `stdout` and the output .mag and .svg files are named the same as the input file.

```
$ ./main benchmark/1 > benchmark/1.log
$ ls benchmark/1.*
benchmarks/1.log  benchmarks/1.mag benchmarks/1.svg
```

4 Results

The results of all the benchmark netlists are tabulated below. The wirelength is the length post routing.

Benchmark	Box X	Box Y	Box Area	# FTs	Wirelength	# Vias	Execution Time	Memory Usage
1	82	76	6232	17	1093	112	0.009873s	296 kB
2	127	129	16383	50	2547	270	0.021806s	340 kB
3	191	175	33425	112	6114	562	0.044696s	424 kB
4	317	252	79884	213	14108	1126	0.111609s	588 kB
5	486	403	195858	464	41469	2336	0.384445s	916 kB
6	602	513	308826	685	70536	3322	0.670760s	1204 kB
7	1110	1130	1254300	1738	443767	5448	1.581445s	1512 kB
8	437	333	145521	297	22907	1994	0.360278s	912 kB
9	328	229	75112	69	6097	842	0.206668s	664 kB
10	258	202	52116	35	2881	404	0.157362s	544 kB

Table 2: Result Summary

4.1 Benchmark 1 - Placement Steps

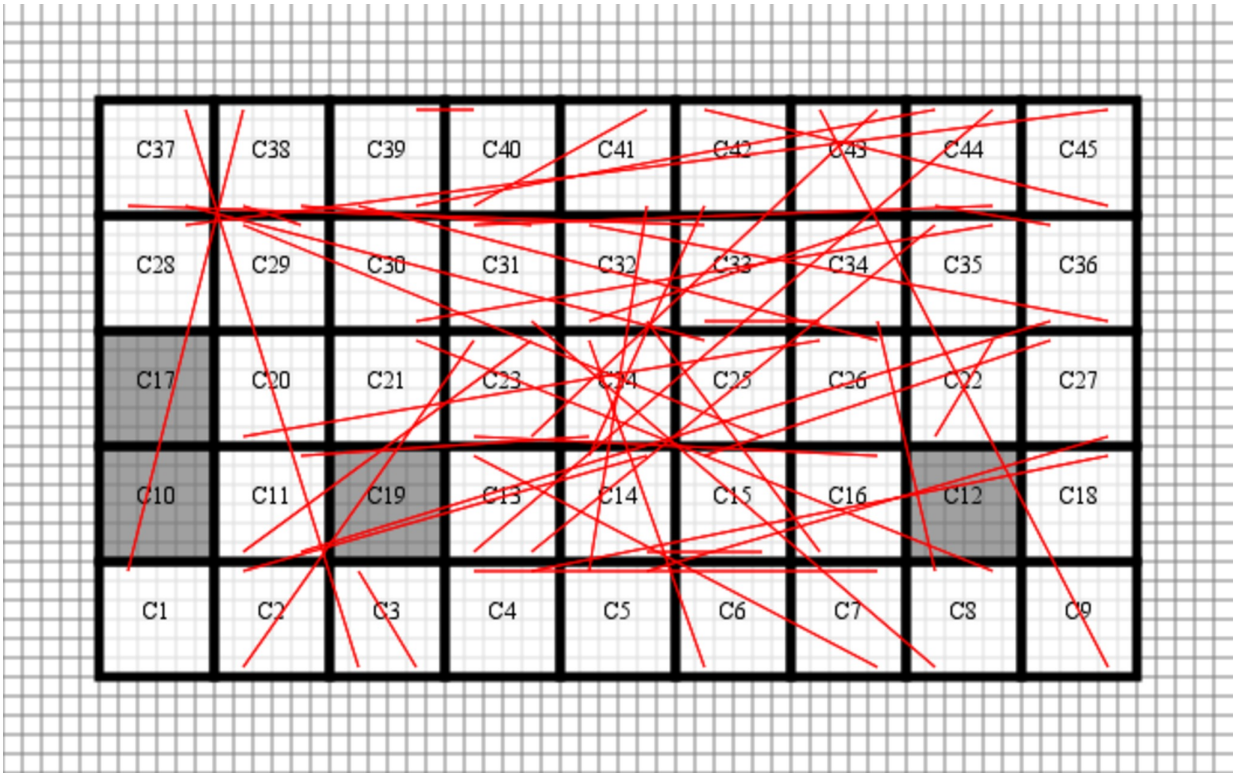

Figure 24: Benchmark 1 - Step 1/7 - Initial placement

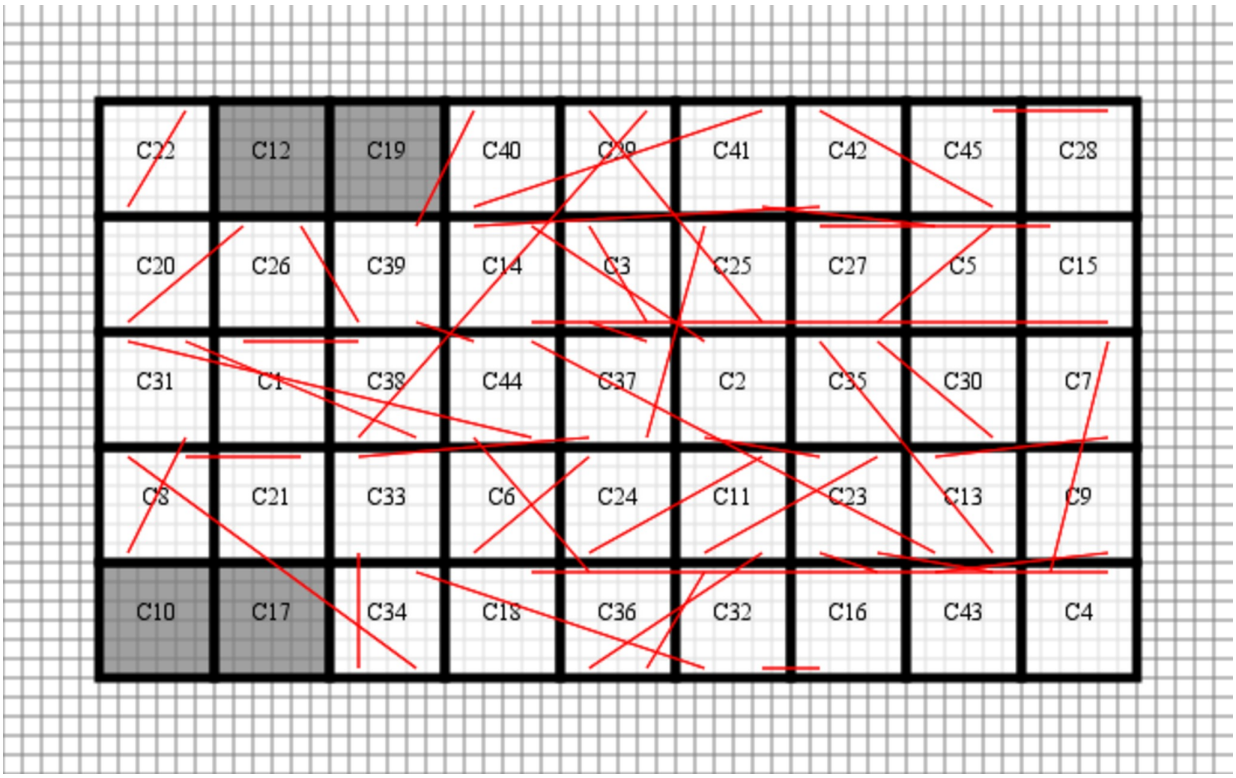

Figure 25: Benchmark 1 - Step 2/7 - Force directed placement

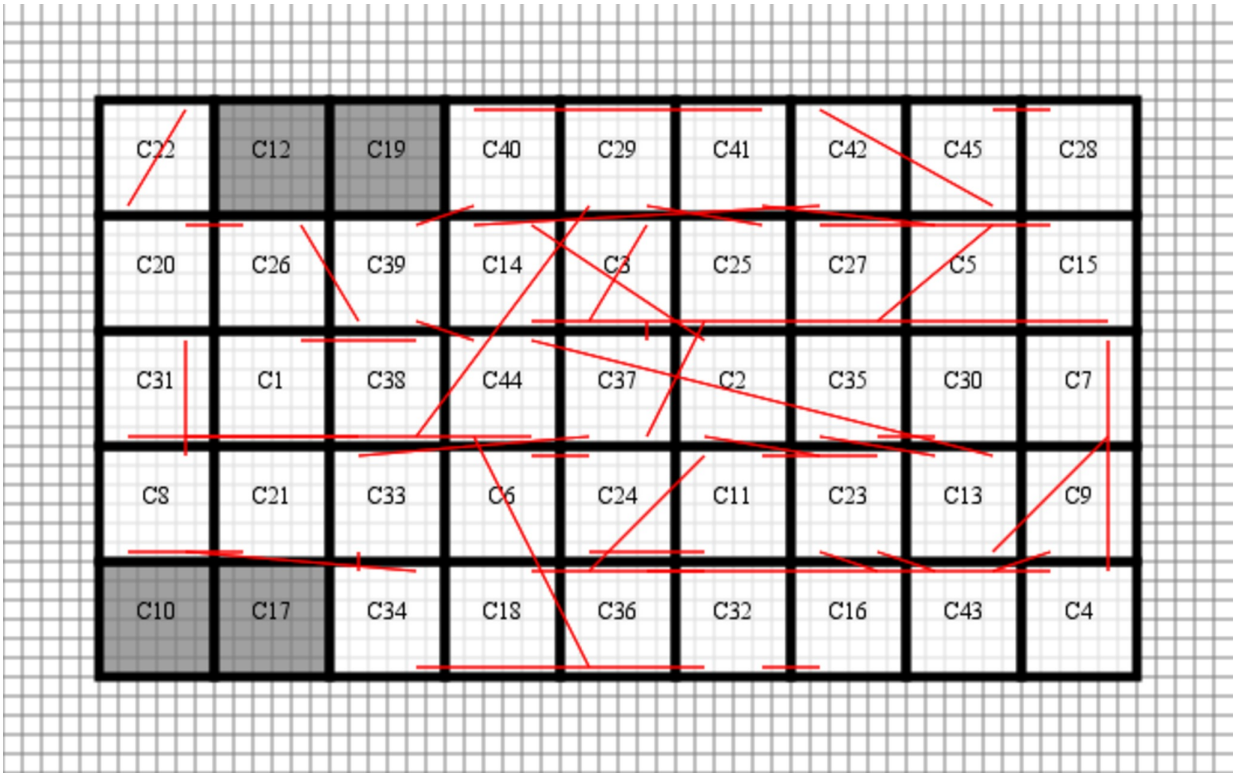

Figure 26: Benchmark 1 - Step 3/7 - Flip cells

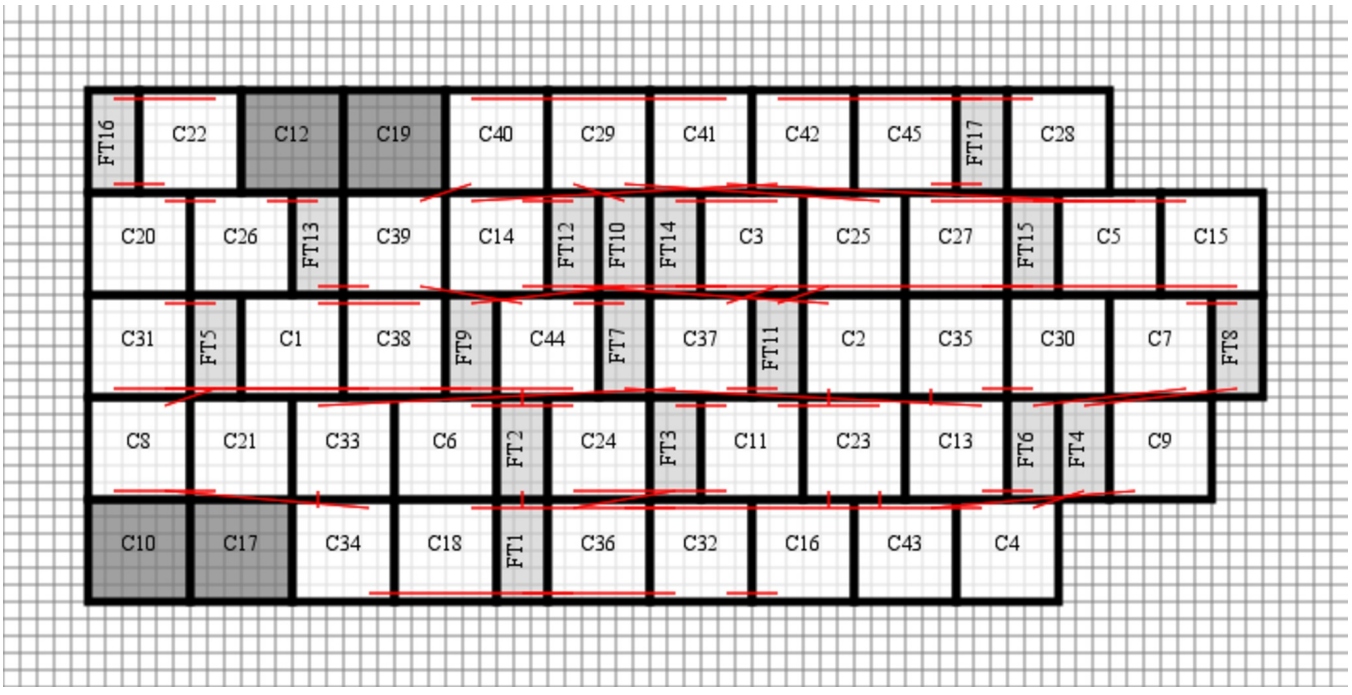

Figure 27: Benchmark 1 - Step 4/7 - Add feed-throughs

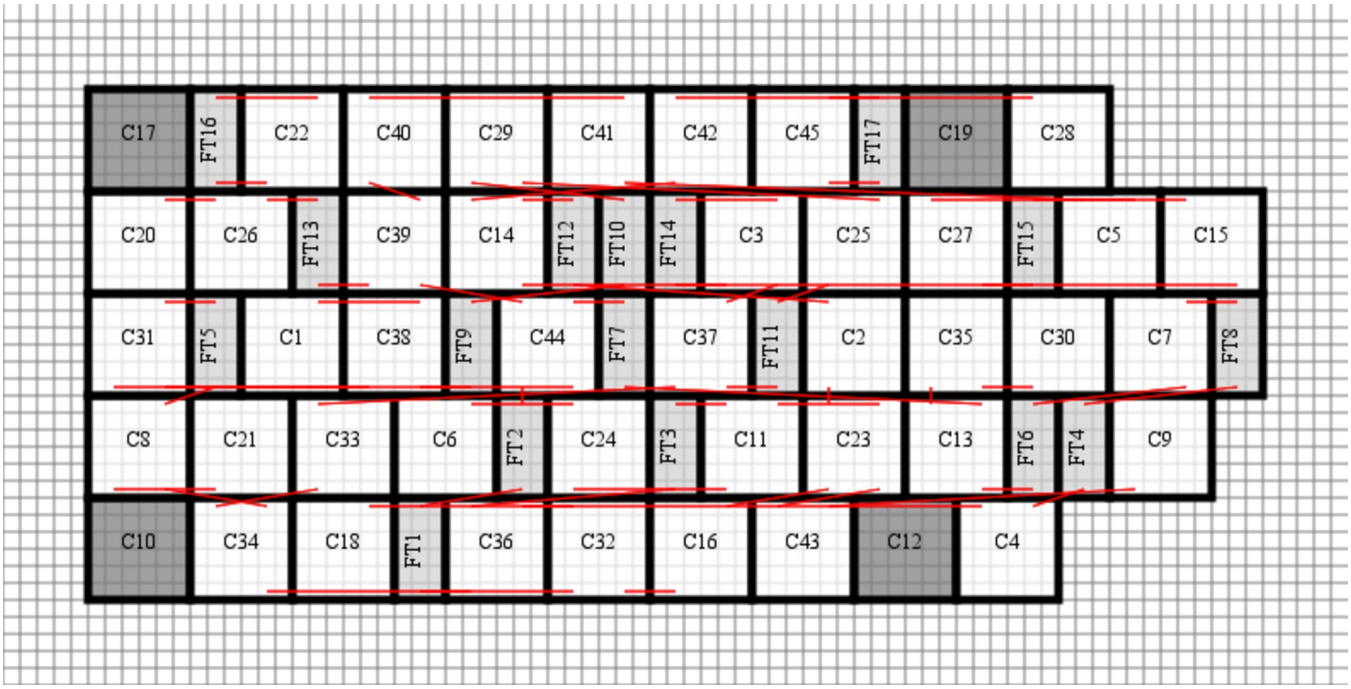

Figure 28: Benchmark 1 - Step 5/7 - Even out the row length

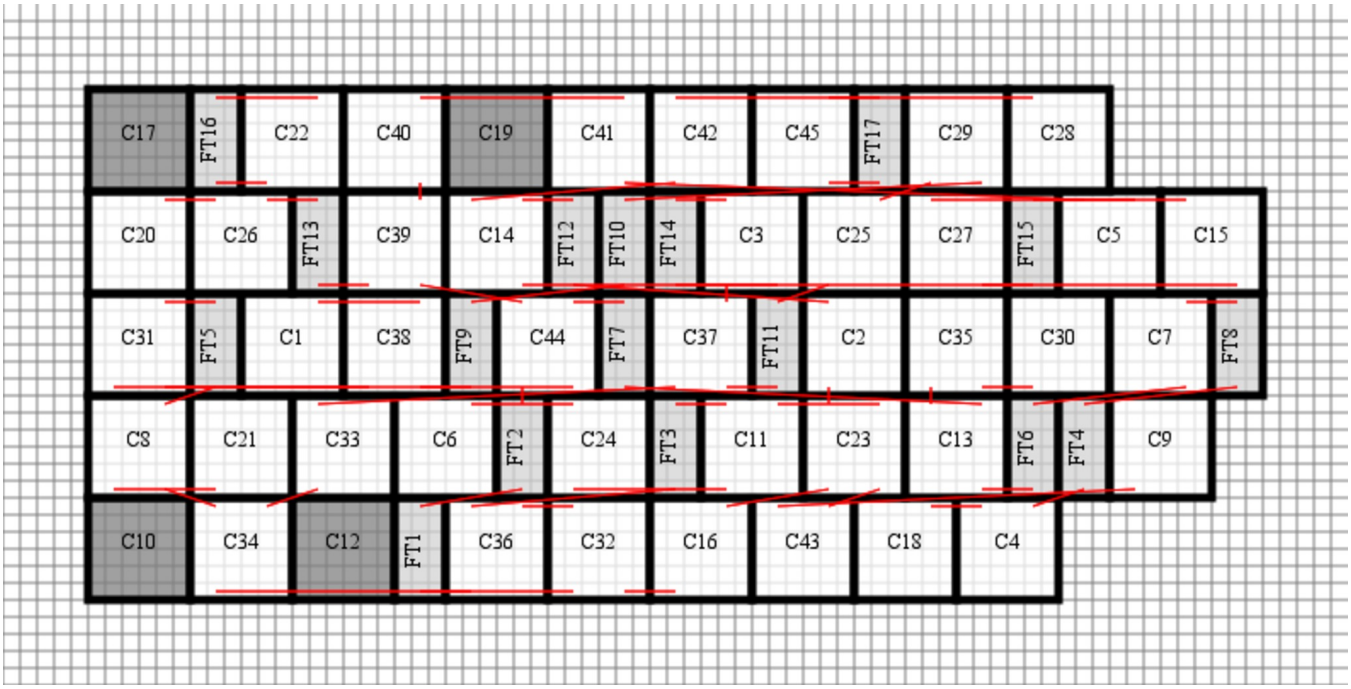

Figure 29: Benchmark 1 - Step 6/7 - Pull cells in the rows closer together

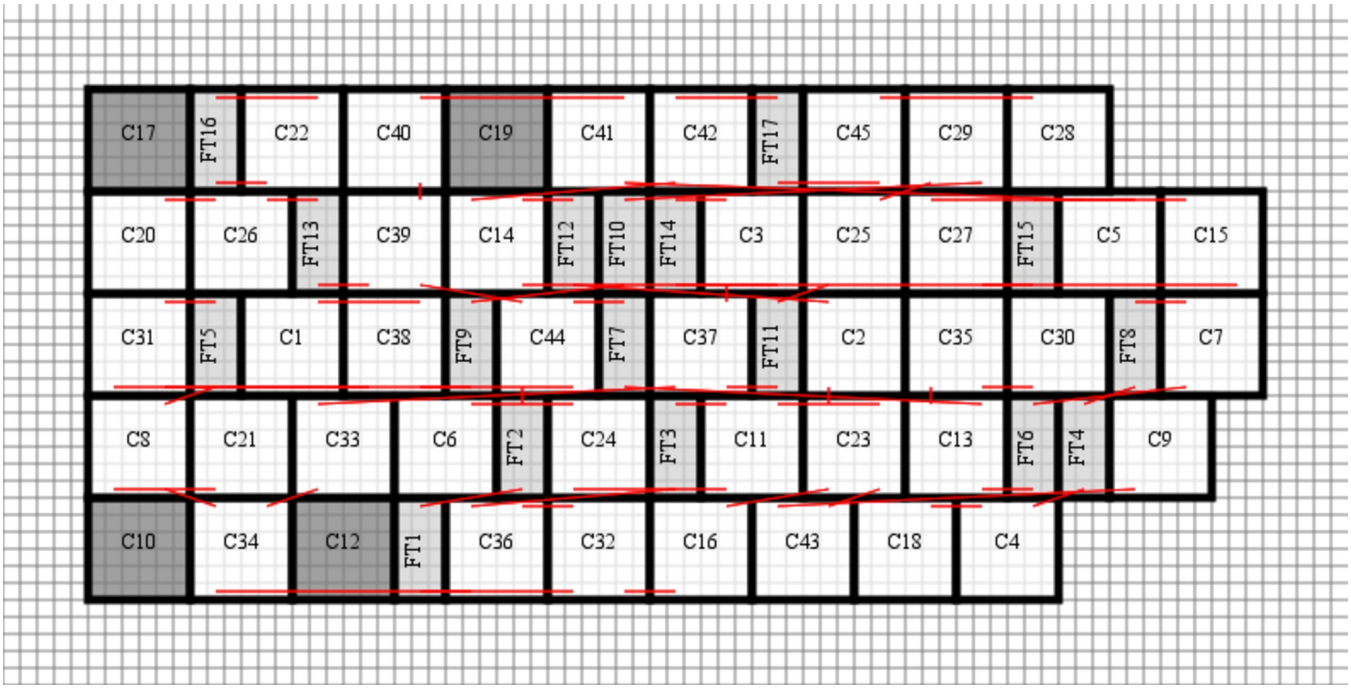

Figure 30: Benchmark 1 - Step 7/7 - Move feed-throughs to optimal location

4.2 Benchmark 1 - Final Routing

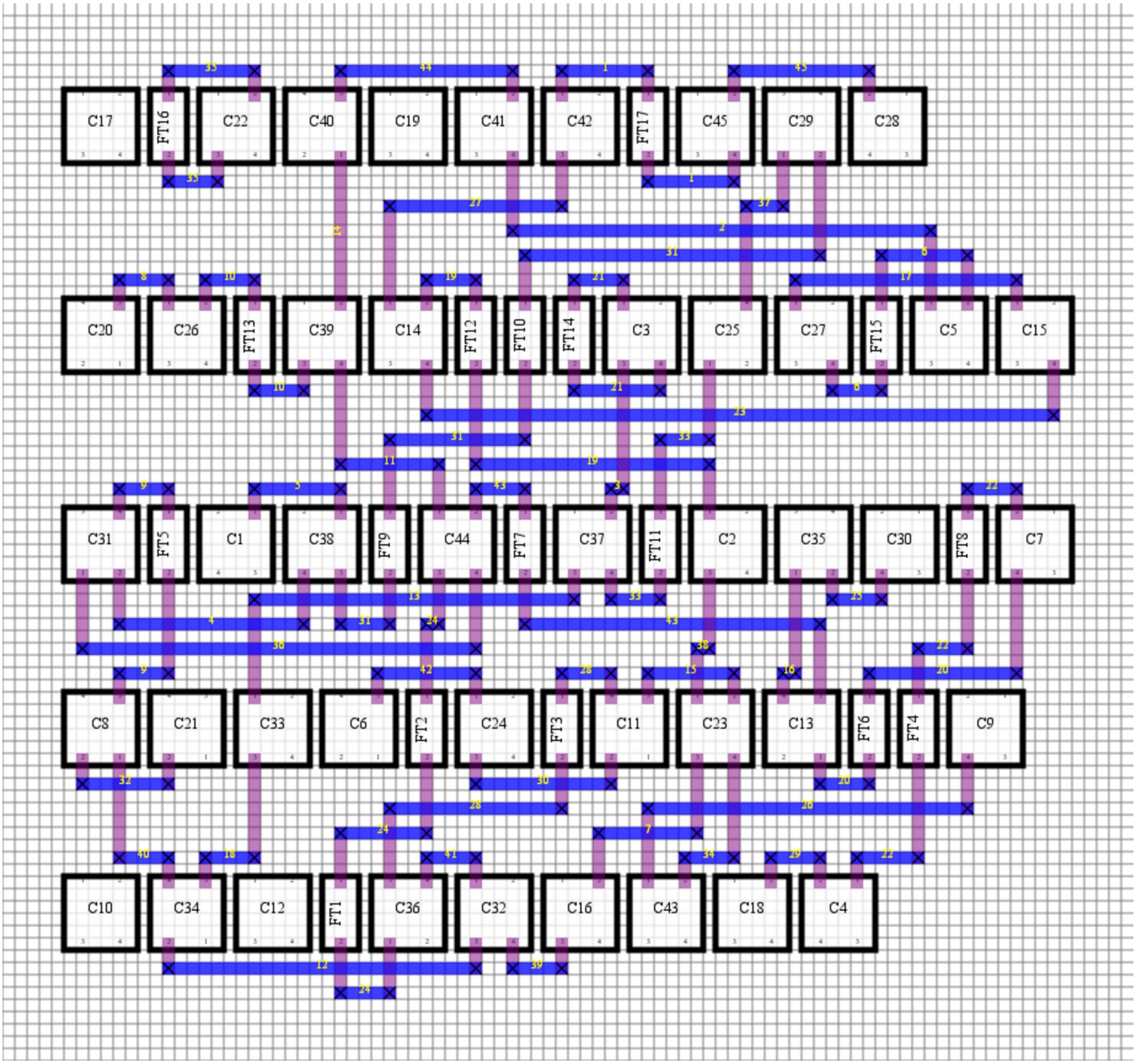

Figure 31: Benchmark 1 - Final routing

4.3 Benchmark 1 - Magic Screenshot

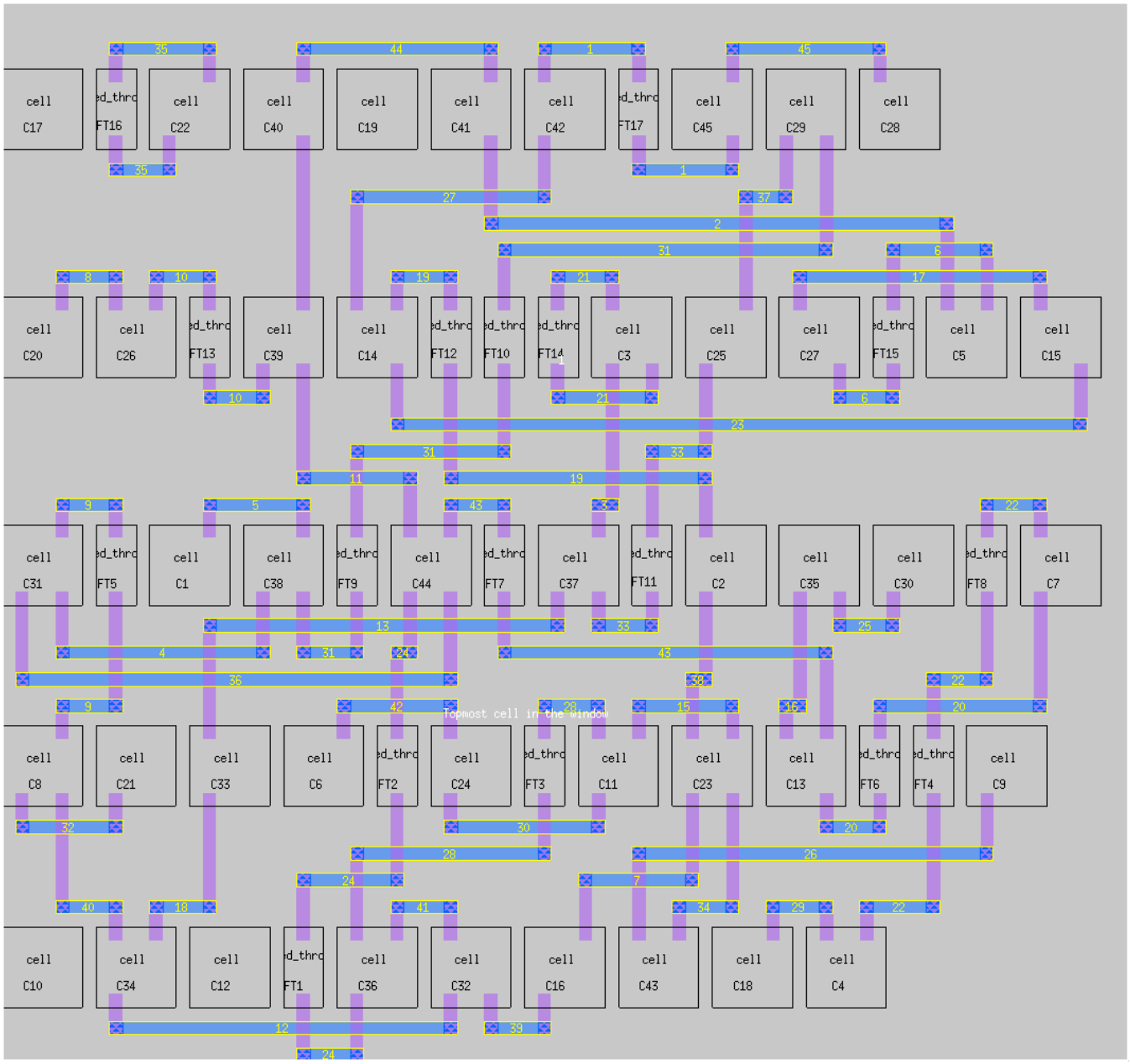

Figure 32: Benchmark 1 - Magic Screenshot

4.4 Benchmark 1 - Log file

```
Number of cell: 45
Number of nets: 45

Placement
---------
Initial grid size: 9 5
Number of feed through cells: 17

Routing
-------
Number of vias: 122
Total number of tracks: 21
Total wirelength: 1093

Size
----
Total width: 82
Total height: 76
Total area: 6232
Squareness (width/height): 1.078947

Time
----
Place time: 0.005293s
Route time: 0.000916s
Total time: 0.009209s

Memory used: 292.000000 kB
```

Listing 1: Benchmark 1 - Log

4.5 Benchmark 2 - Placement Steps

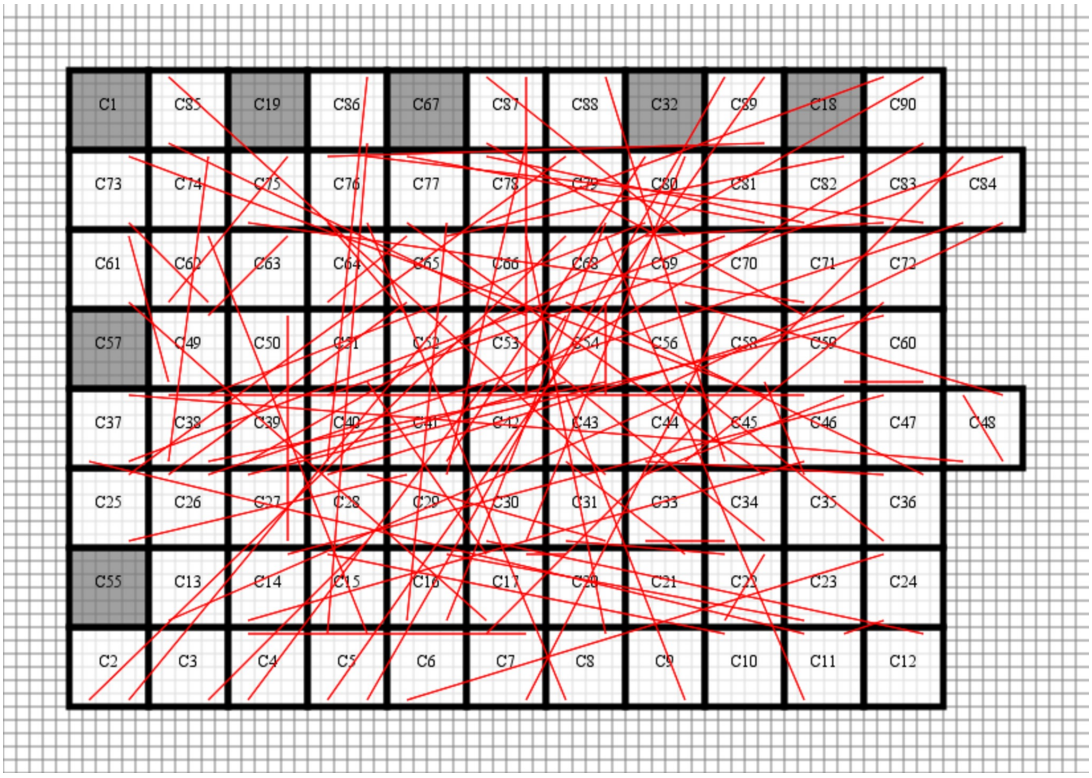

Figure 33: Benchmark 2 - Step 1/7 - Initial placement

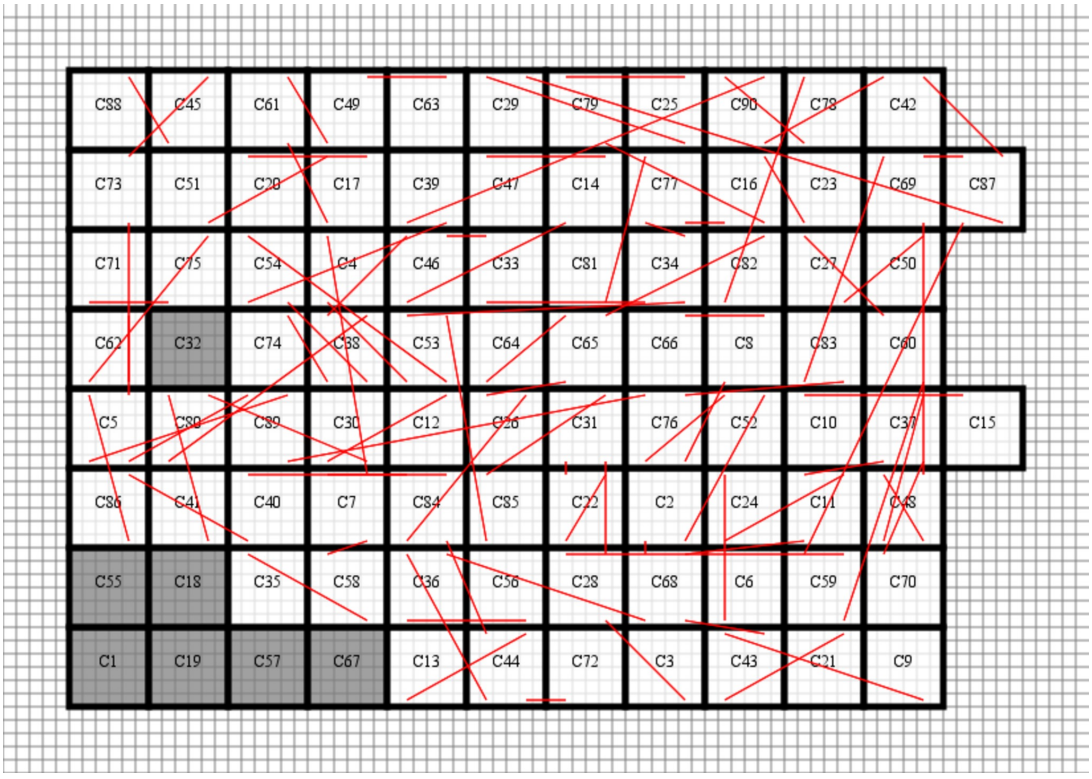

Figure 34: Benchmark 2 - Step 2/7 - Force directed placement

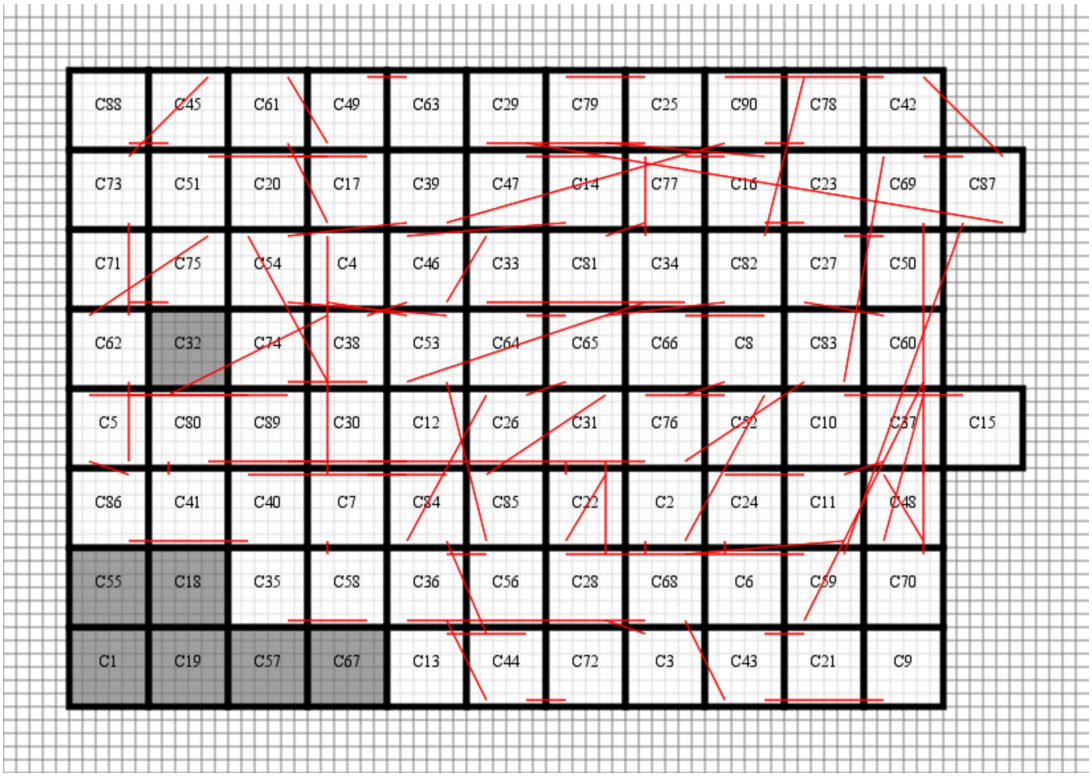

Figure 35: Benchmark 2 - Step 3/7 - Flip cells

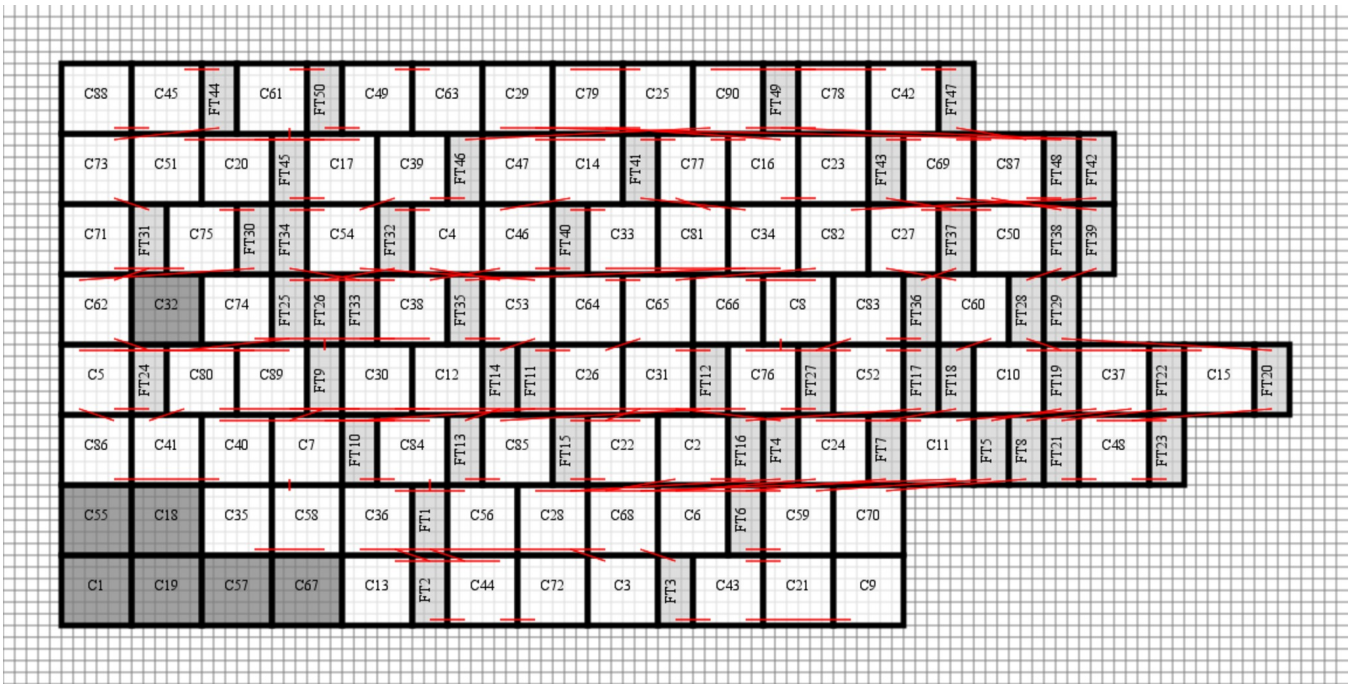

Figure 36: Benchmark 2 - Step 4/7 - Add feed-throughs

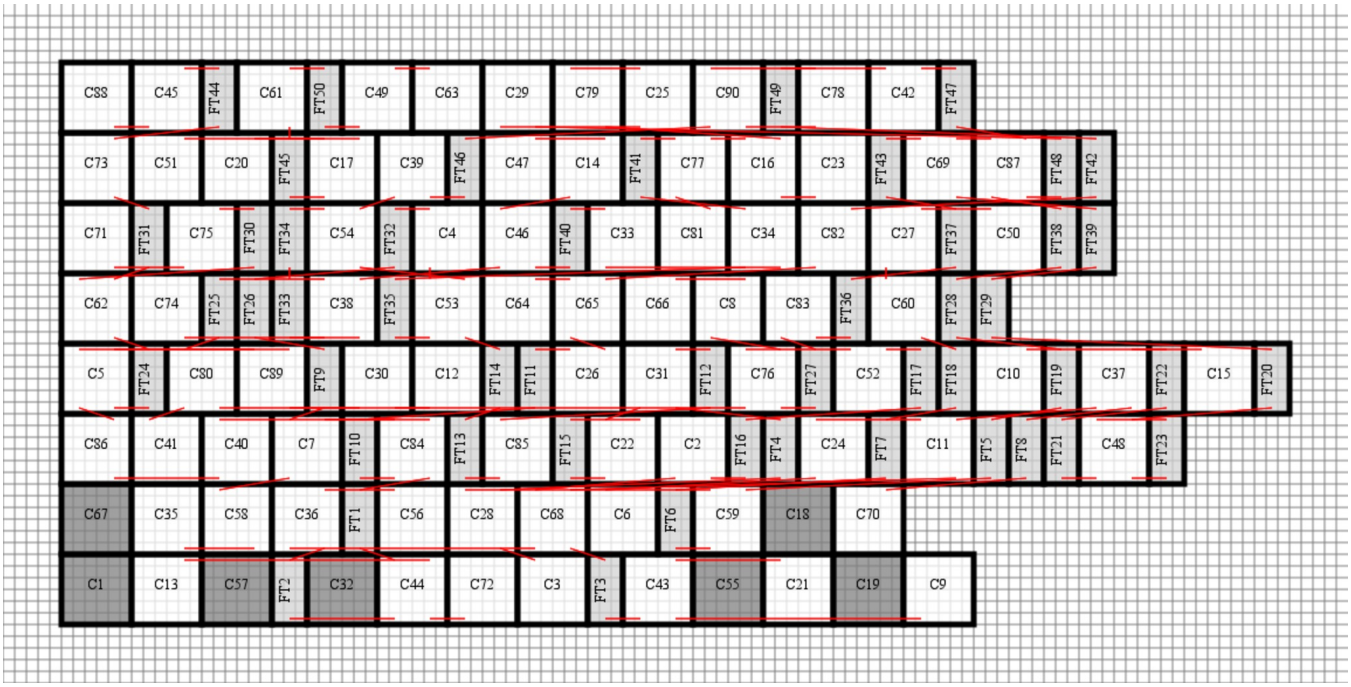

Figure 37: Benchmark 2 - Step 5/7 - Even out the row length

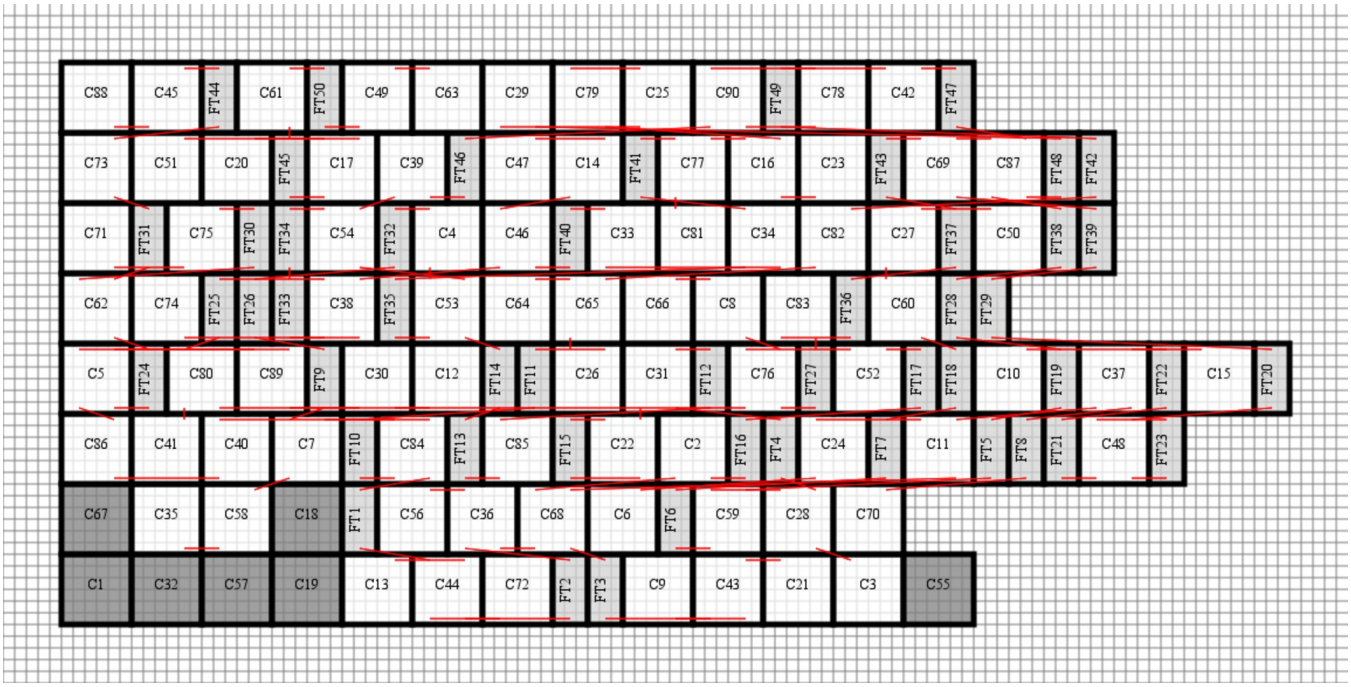

Figure 38: Benchmark 2 - Step 6/7 - Pull cells in the rows closer together

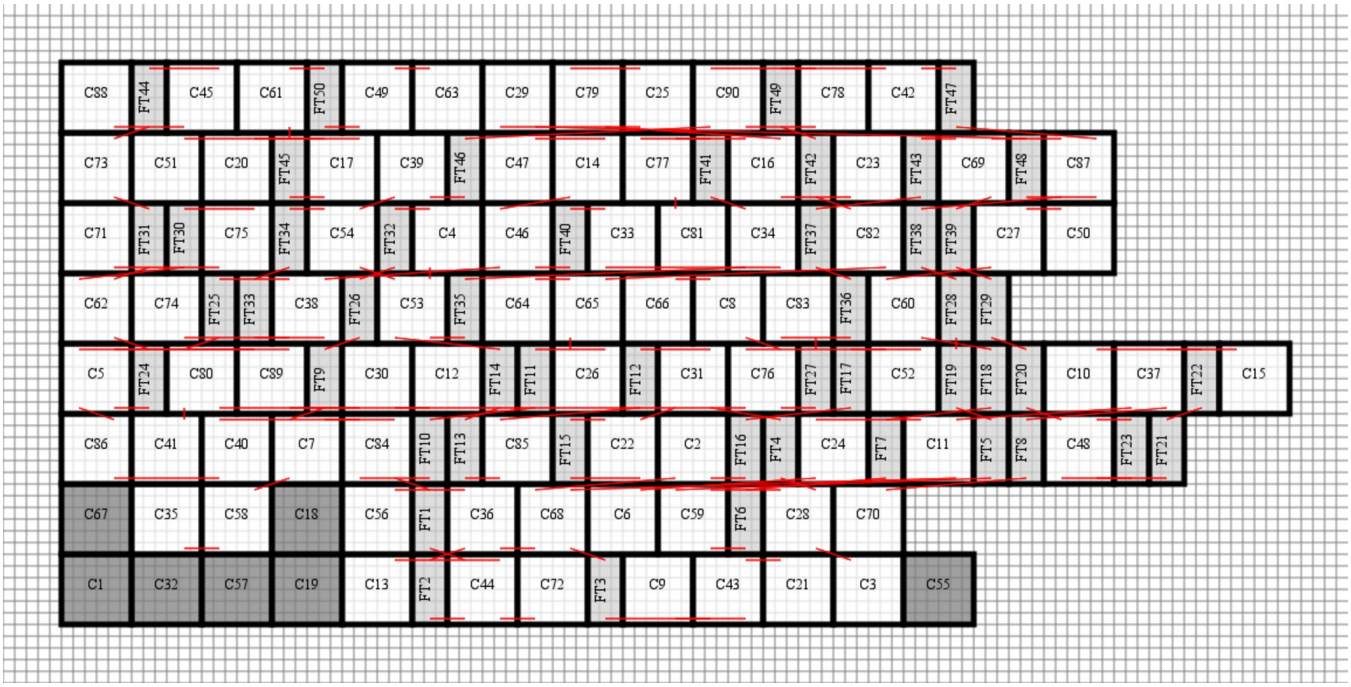

Figure 39: Benchmark 2 - Step 7/7 - Move feed-throughs to optimal location

4.6 Benchmark 2 - Final Routing

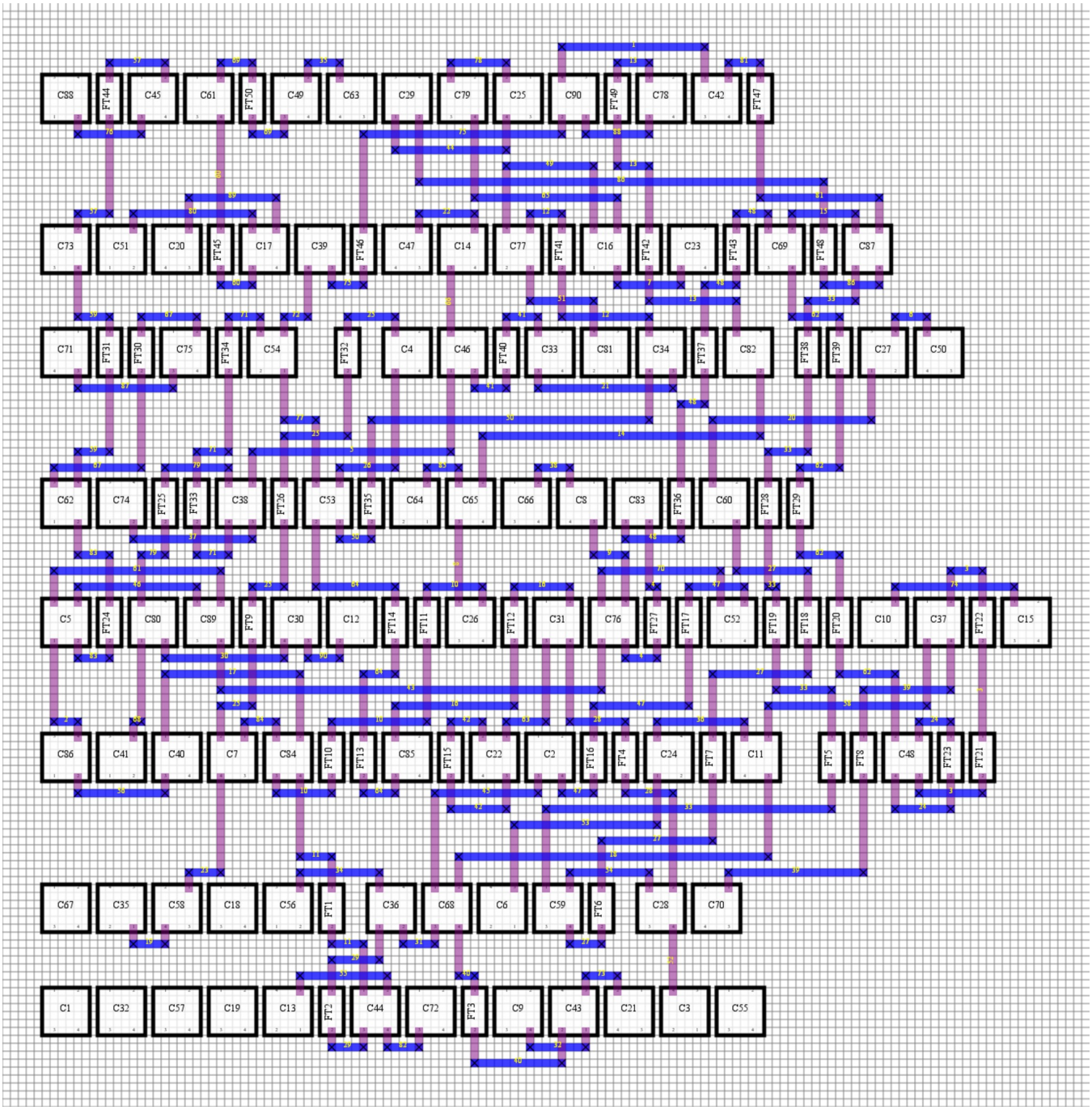

Figure 40: Benchmark 2 - Final routing

4.7 Benchmark 2 - Magic Screenshot

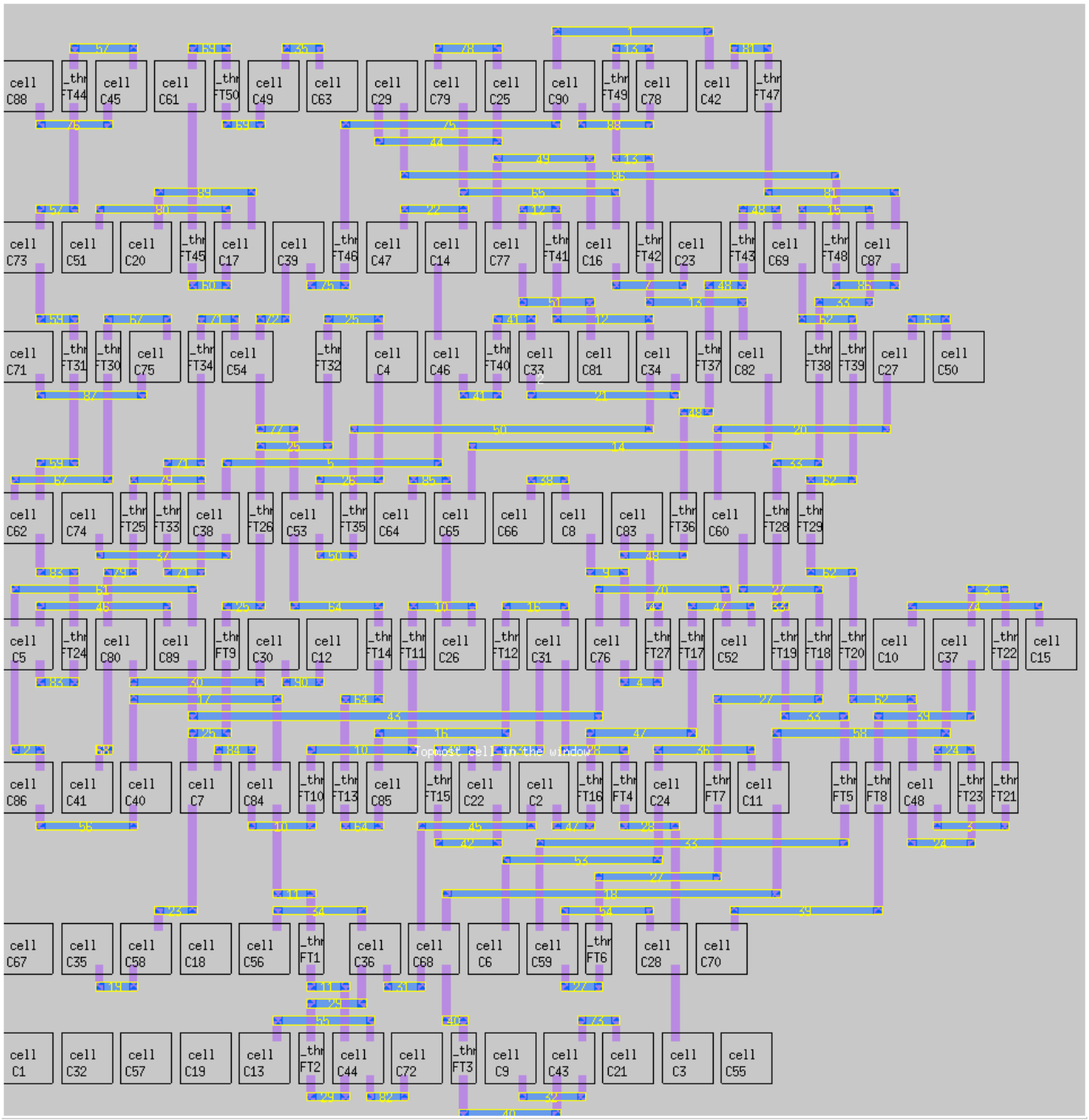

Figure 41: Benchmark 2 - Magic Screenshot

4.8 Benchmark 2 - Log file

```
Number of cell: 90
Number of nets: 90

Placement
---------
Initial grid size: 12 8
Number of feed through cells: 50

Routing
-------
Number of vias: 270
Total number of tracks: 37
Total wirelength: 2547

Size
----
Total width: 127
Total height: 129
Total area: 16383
Squareness (width/height): 0.984496

Time
----
Place time: 0.010814s
Route time: 0.002603s
Total time: 0.019800s

Memory used: 340.000000 kB
```

Listing 2: Benchmark 2 - Log

4.9 Benchmark 3 - Placement Steps

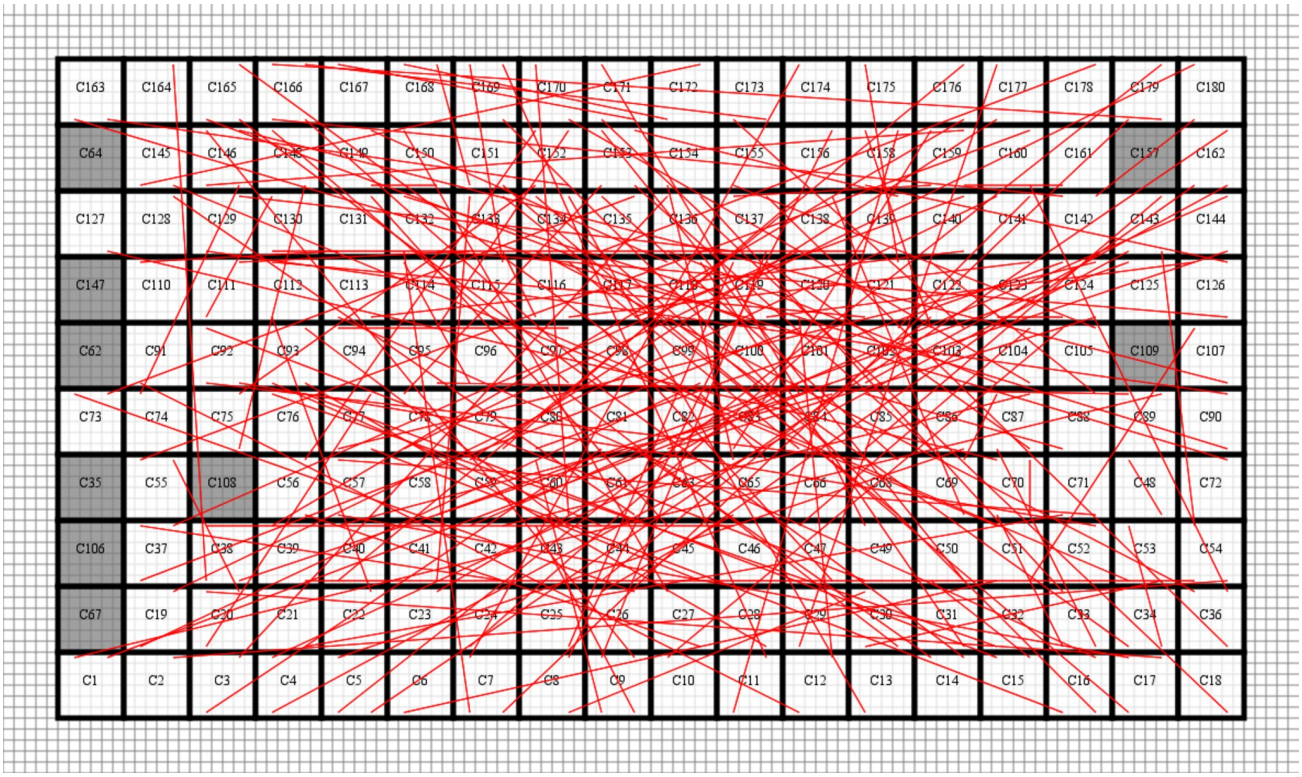

Figure 42: Benchmark 3 - Step 1/7 - Initial placement

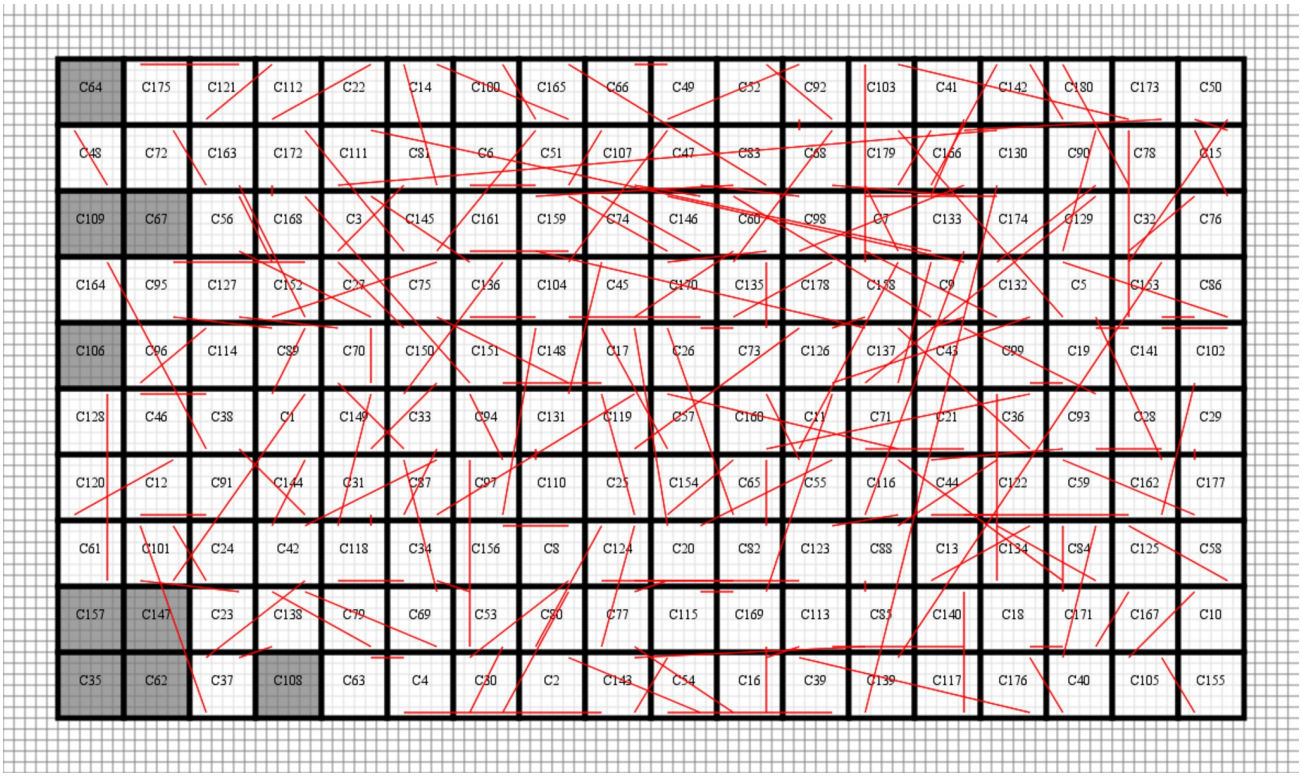

Figure 43: Benchmark 3 - Step 2/7 - Force directed placement

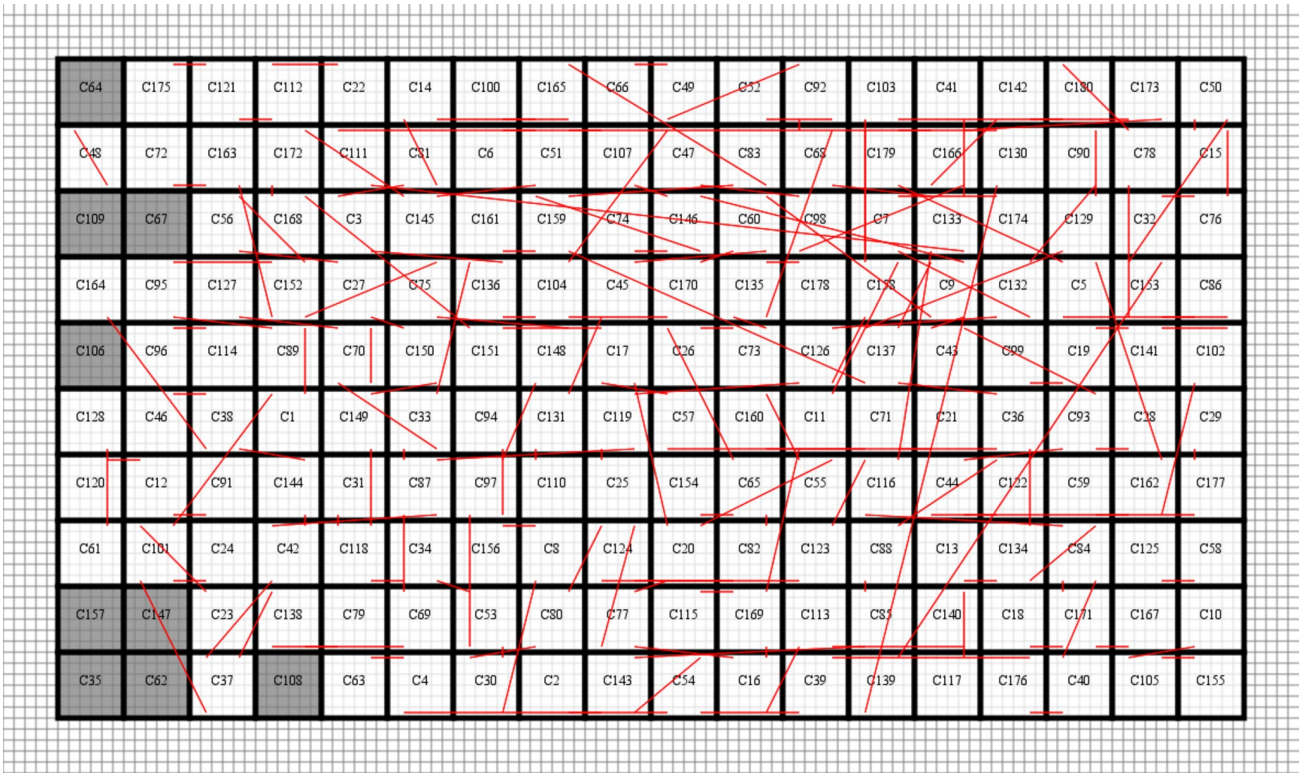

Figure 44: Benchmark 3 - Step 3/7 - Flip cells

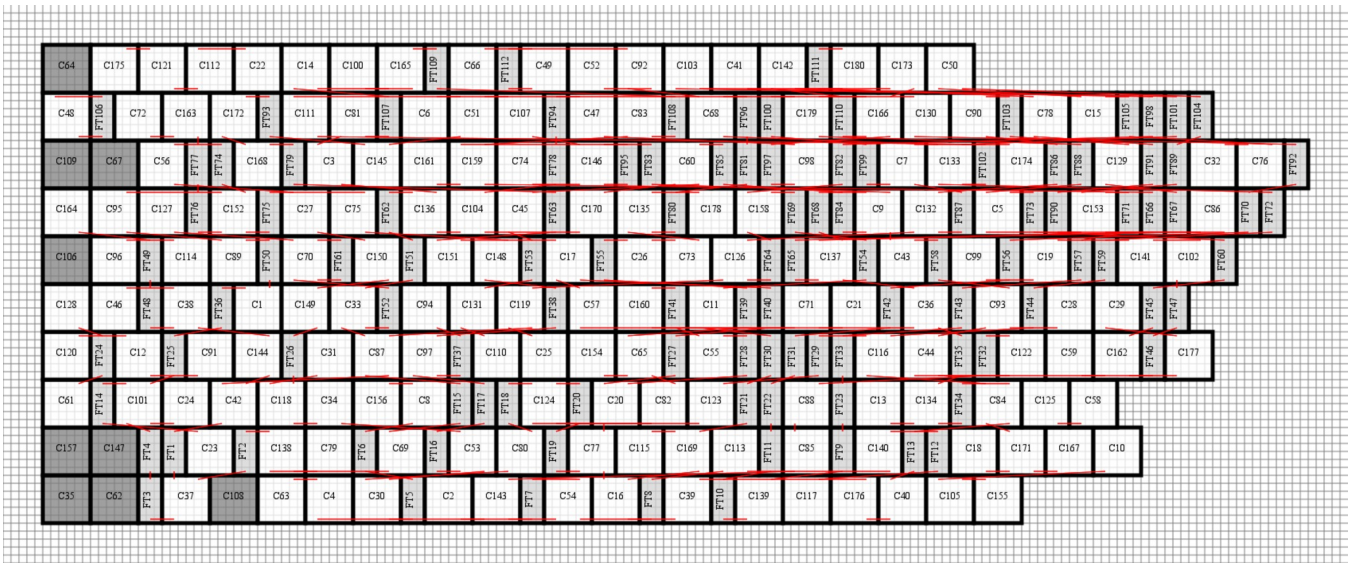

Figure 45: Benchmark 3 - Step 4/7 - Add feed-throughs

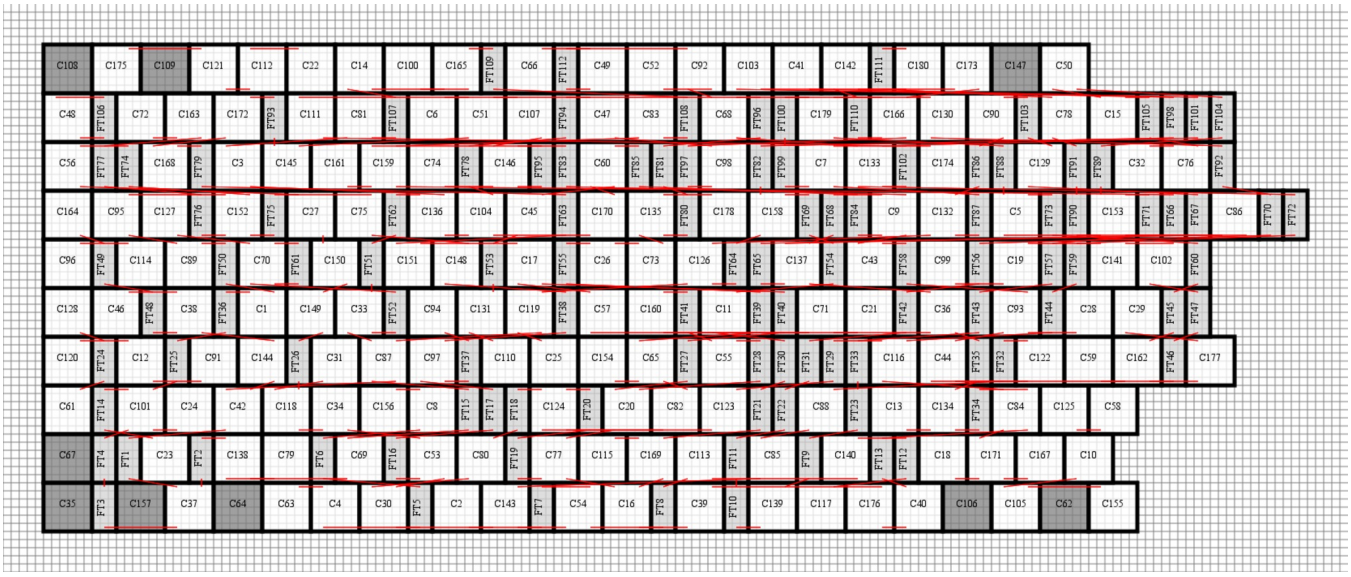

Figure 46: Benchmark 3 - Step 5/7 - Even out the row length

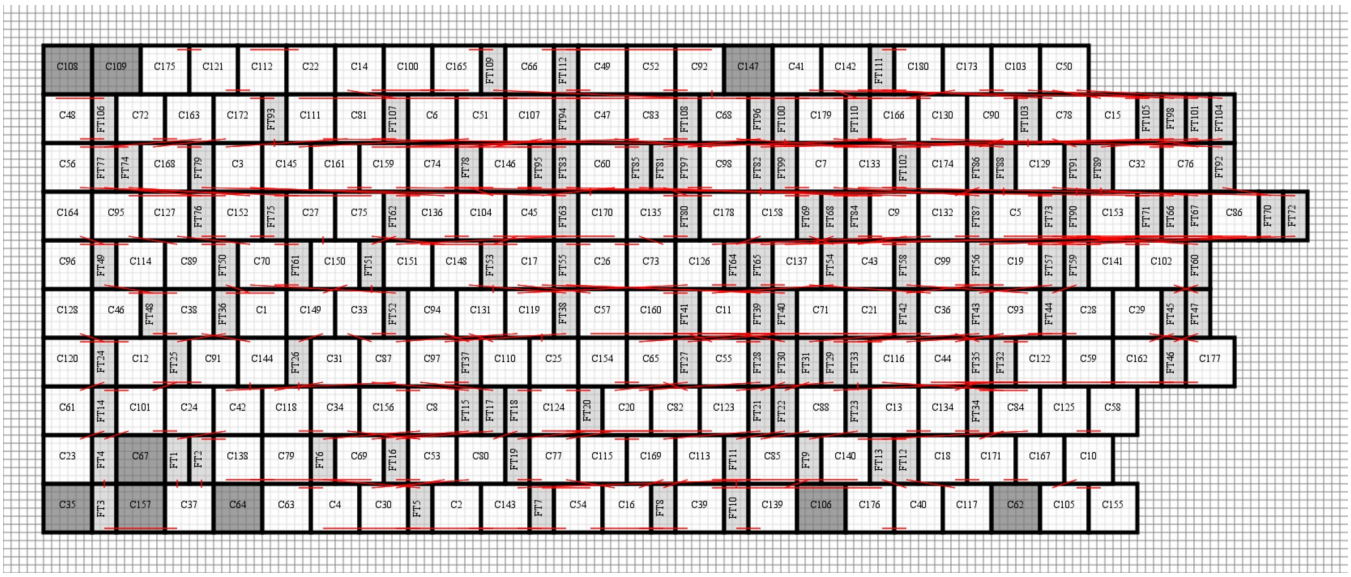

Figure 47: Benchmark 3 - Step 6/7 - Pull cells in the rows closer together

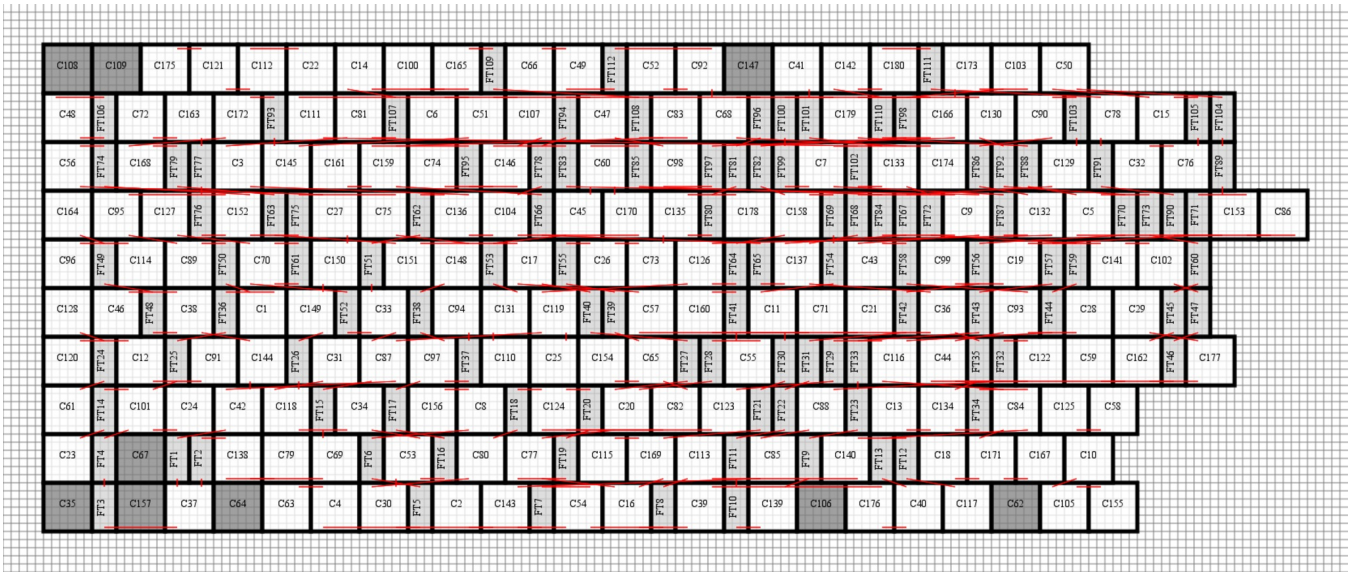

Figure 48: Benchmark 3 - Step 7/7 - Move feed-throughs to optimal location

4.10 Benchmark 3 - Final Routing

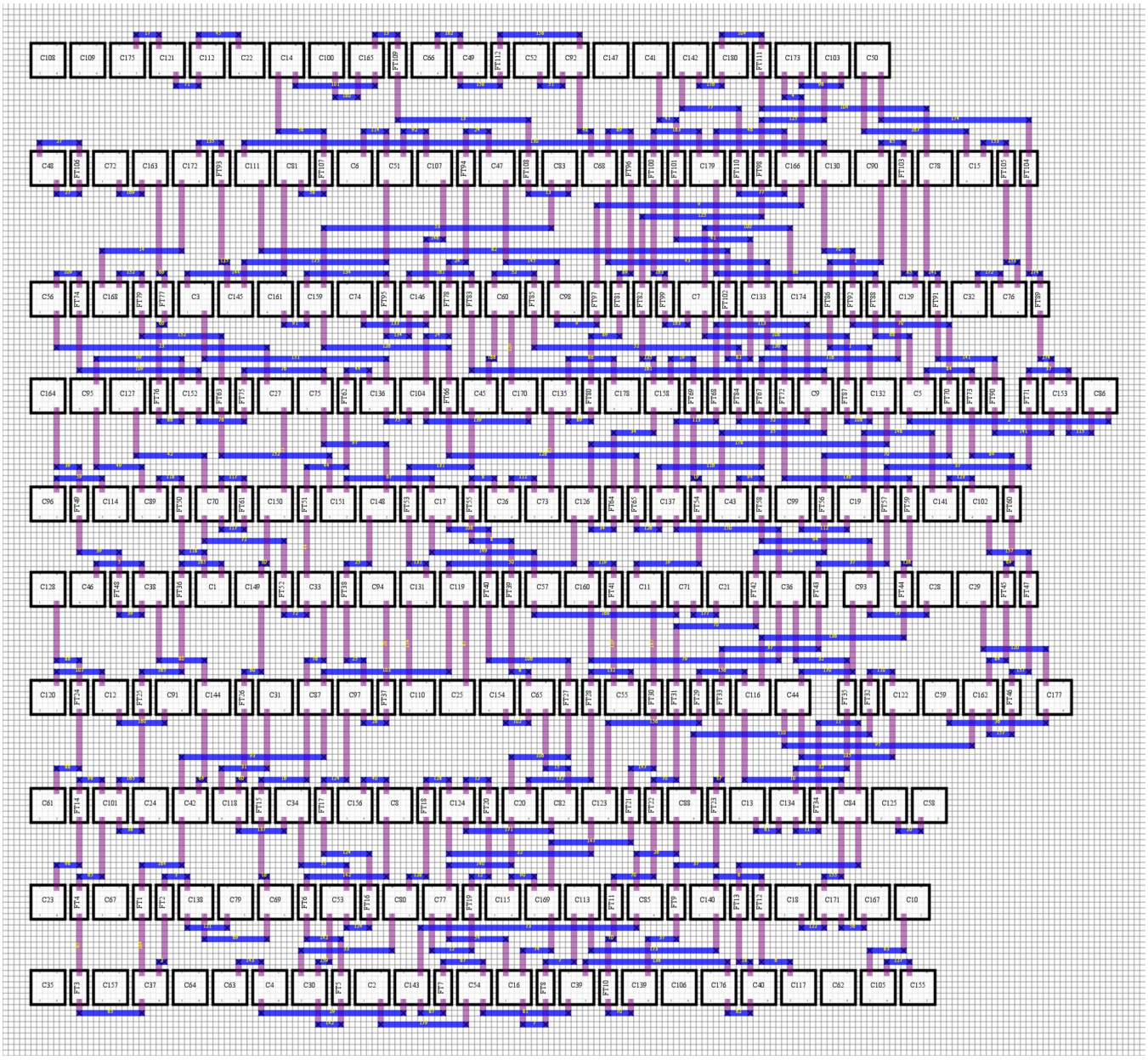

Figure 49: Benchmark 3 - Final routing

4.11 Benchmark 3 - Magic Screenshot

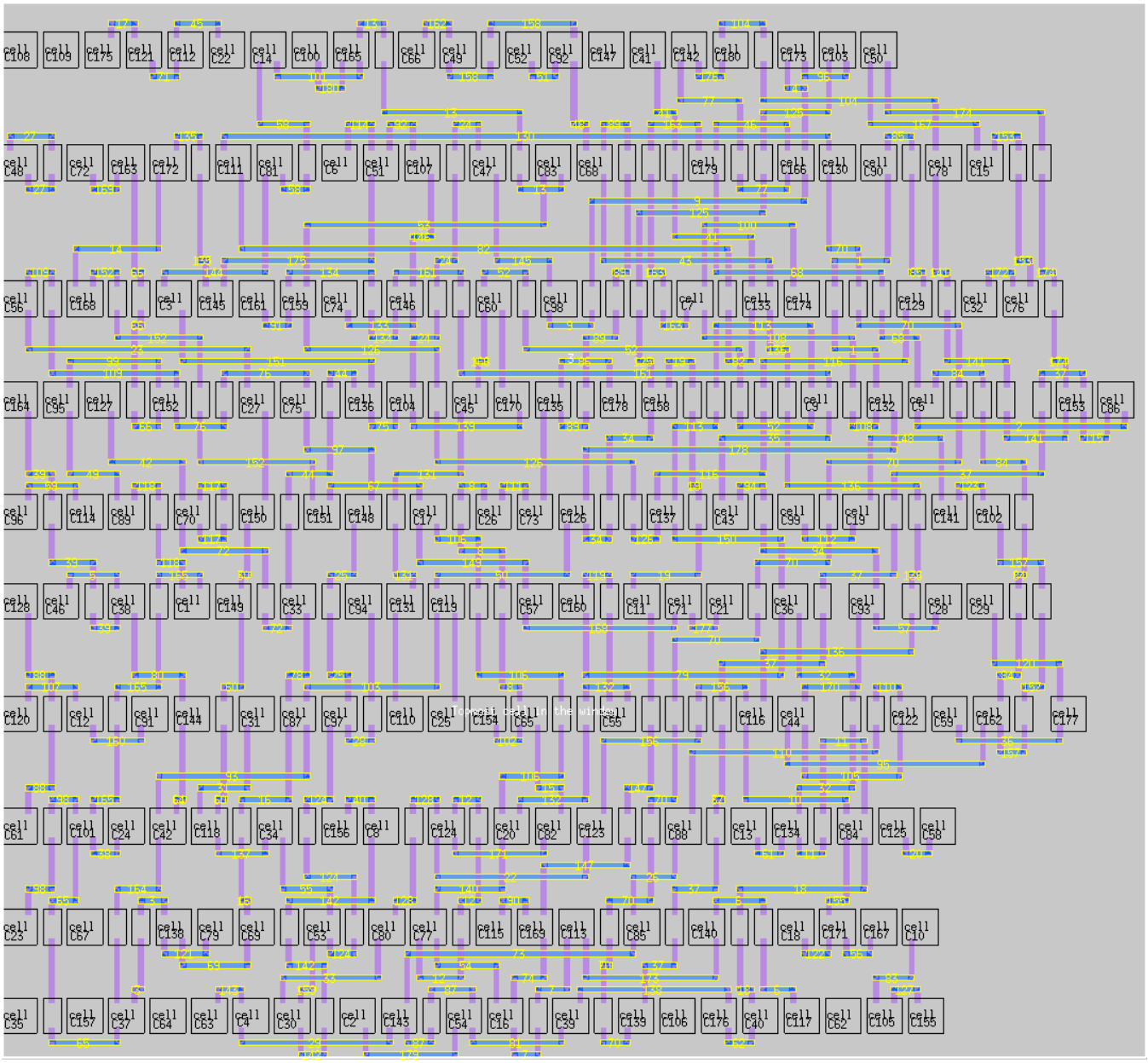

Figure 50: Benchmark 3 - Magic Screenshot

4.12 Benchmark 3 - Log file

```
Number of cell: 180
Number of nets: 180

Placement
---------
Initial grid size: 18 10
Number of feed through cells: 112

Routing
-------
Number of vias: 562
Total number of tracks: 53
Total wirelength: 6114

Size
----
Total width: 191
Total height: 175
Total area: 33425
Squareness (width/height): 1.091429

Time
----
Place time: 0.023764s
Route time: 0.007911s
Total time: 0.044868s

Memory used: 432.000000 kB
```

Listing 3: Benchmark 3 - Log

4.13 Benchmark 4 - Placement Steps

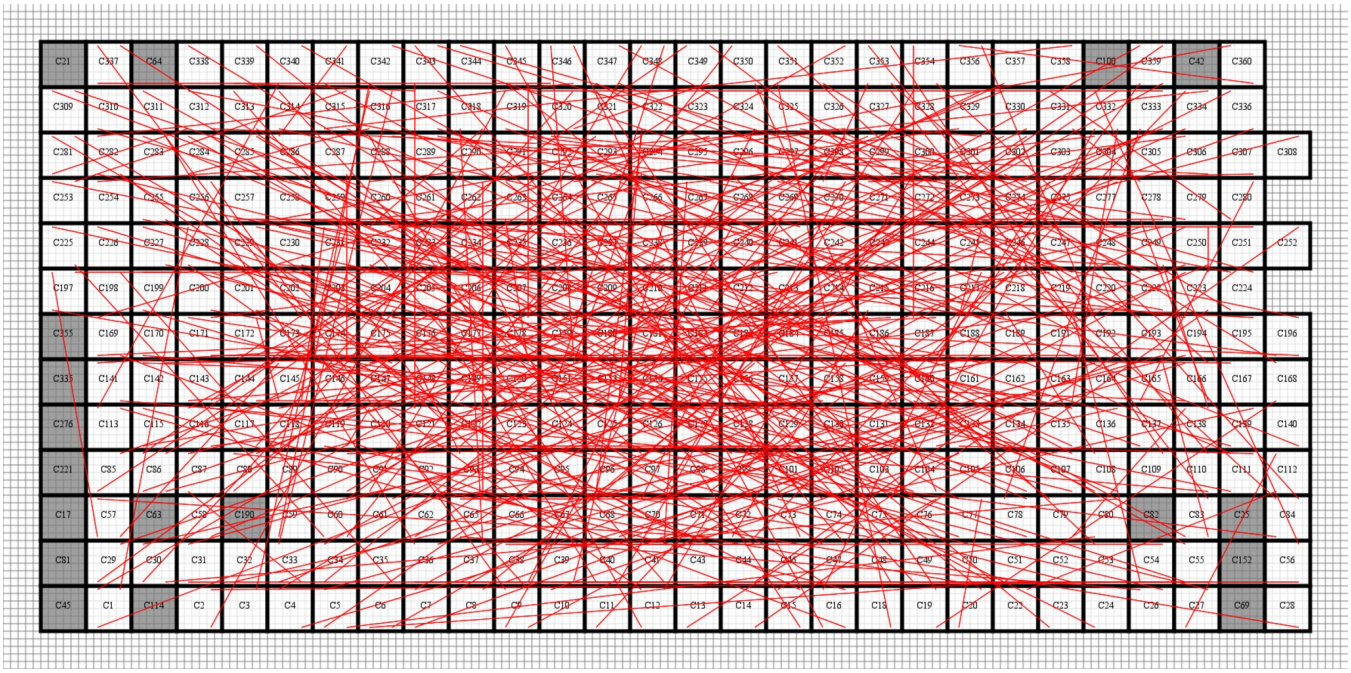

Figure 51: Benchmark 4 - Step 1/7 - Initial placement

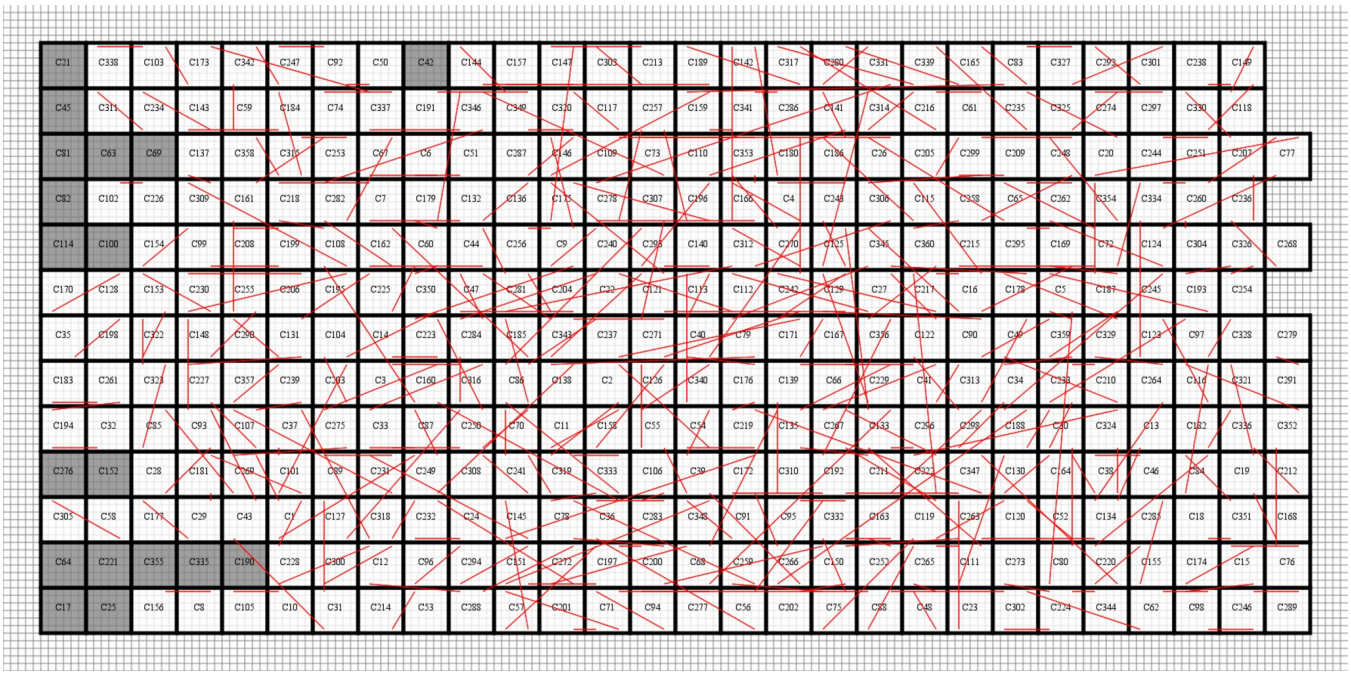

Figure 52: Benchmark 4 - Step 2/7 - Force directed placement

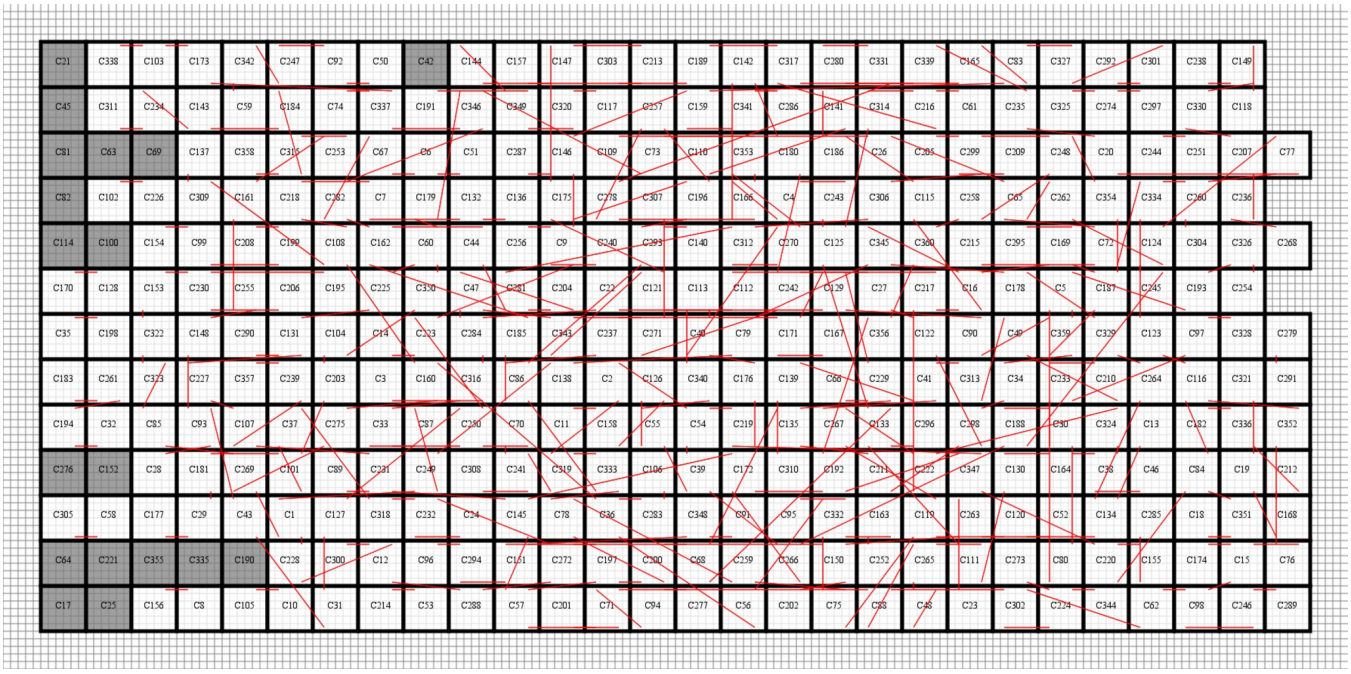

Figure 53: Benchmark 4 - Step 3/7 - Flip cells

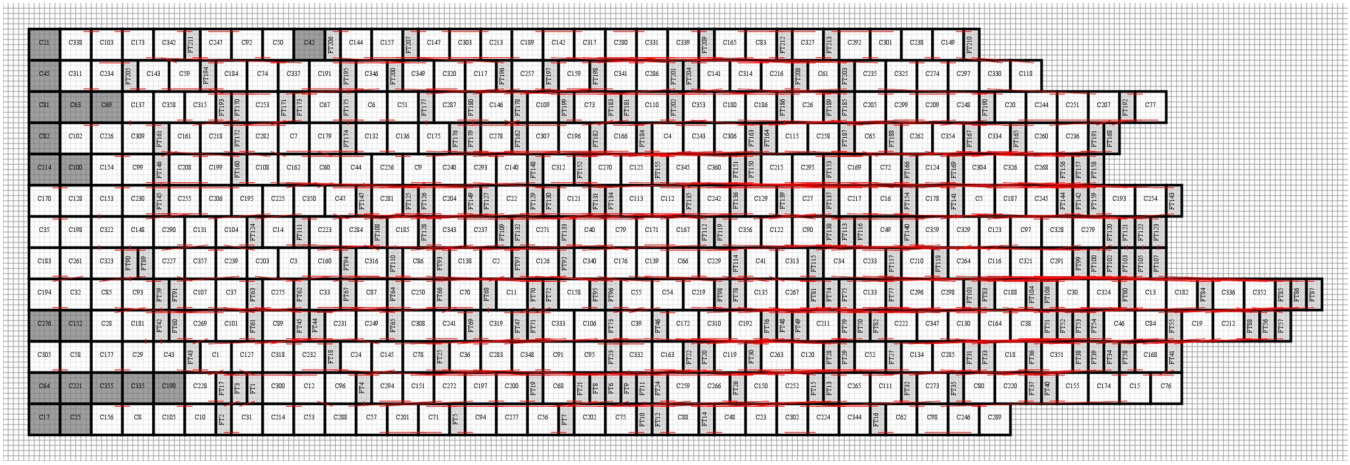

Figure 54: Benchmark 4 - Step 4/7 - Add feed-throughs

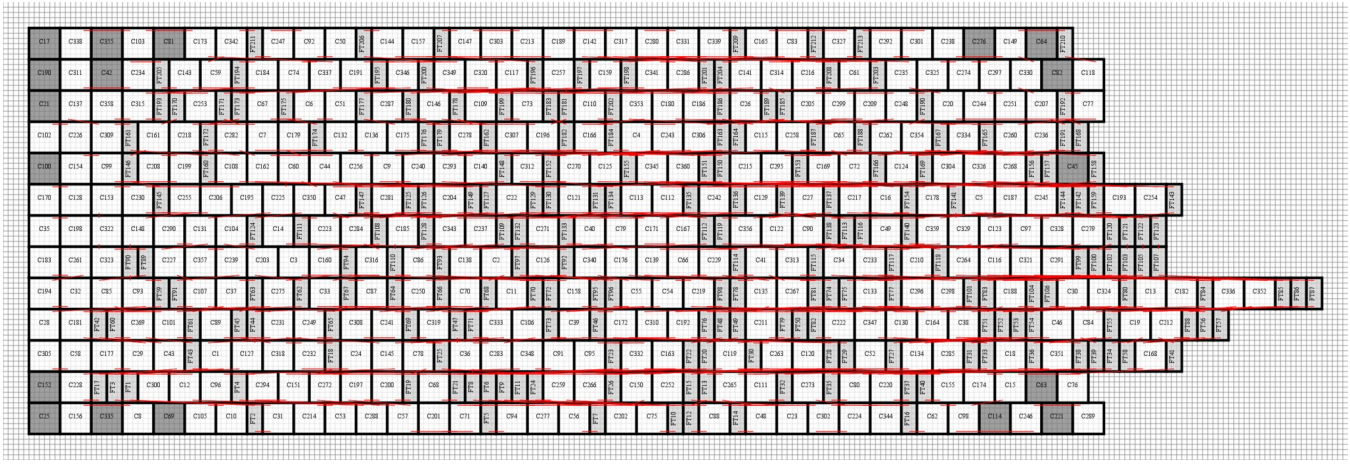

Figure 55: Benchmark 4 - Step 5/7 - Even out the row length

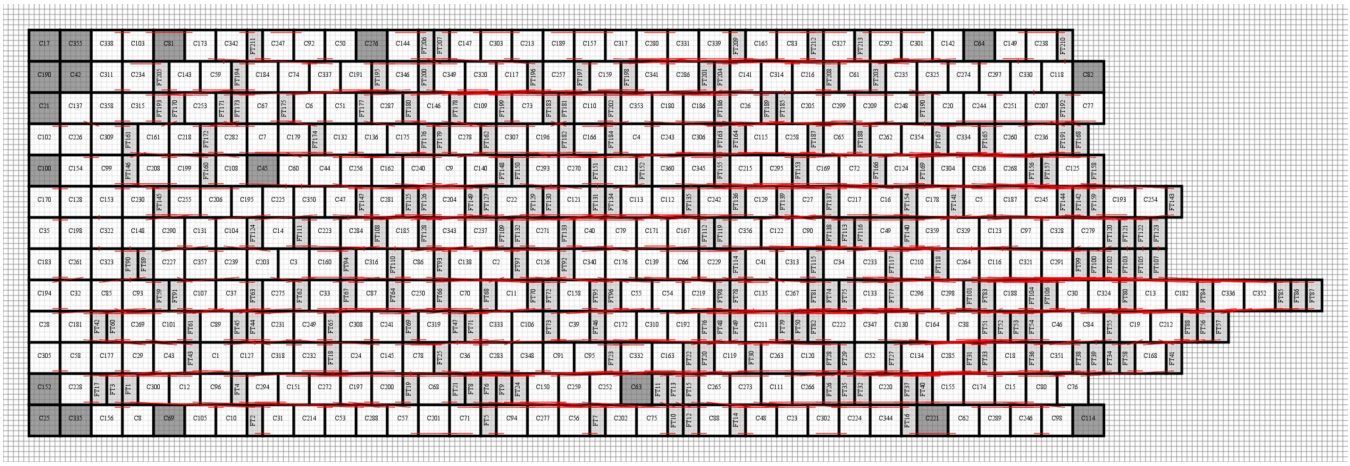

Figure 56: Benchmark 4 - Step 6/7 - Pull cells in the rows closer together

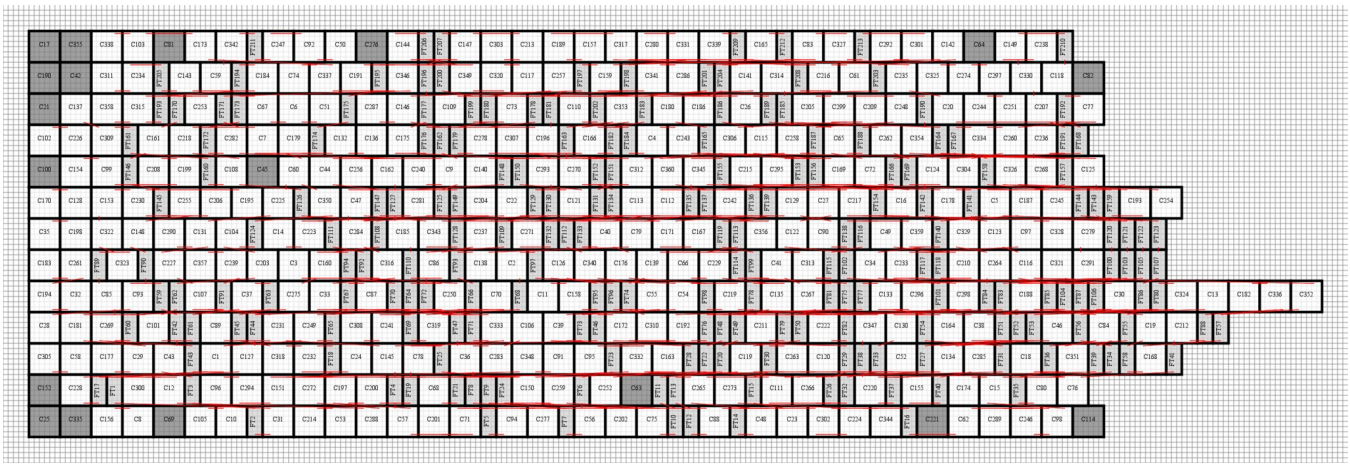

Figure 57: Benchmark 4 - Step 7/7 - Move feed-throughs to optimal location

4.14 Benchmark 4 - Final Routing

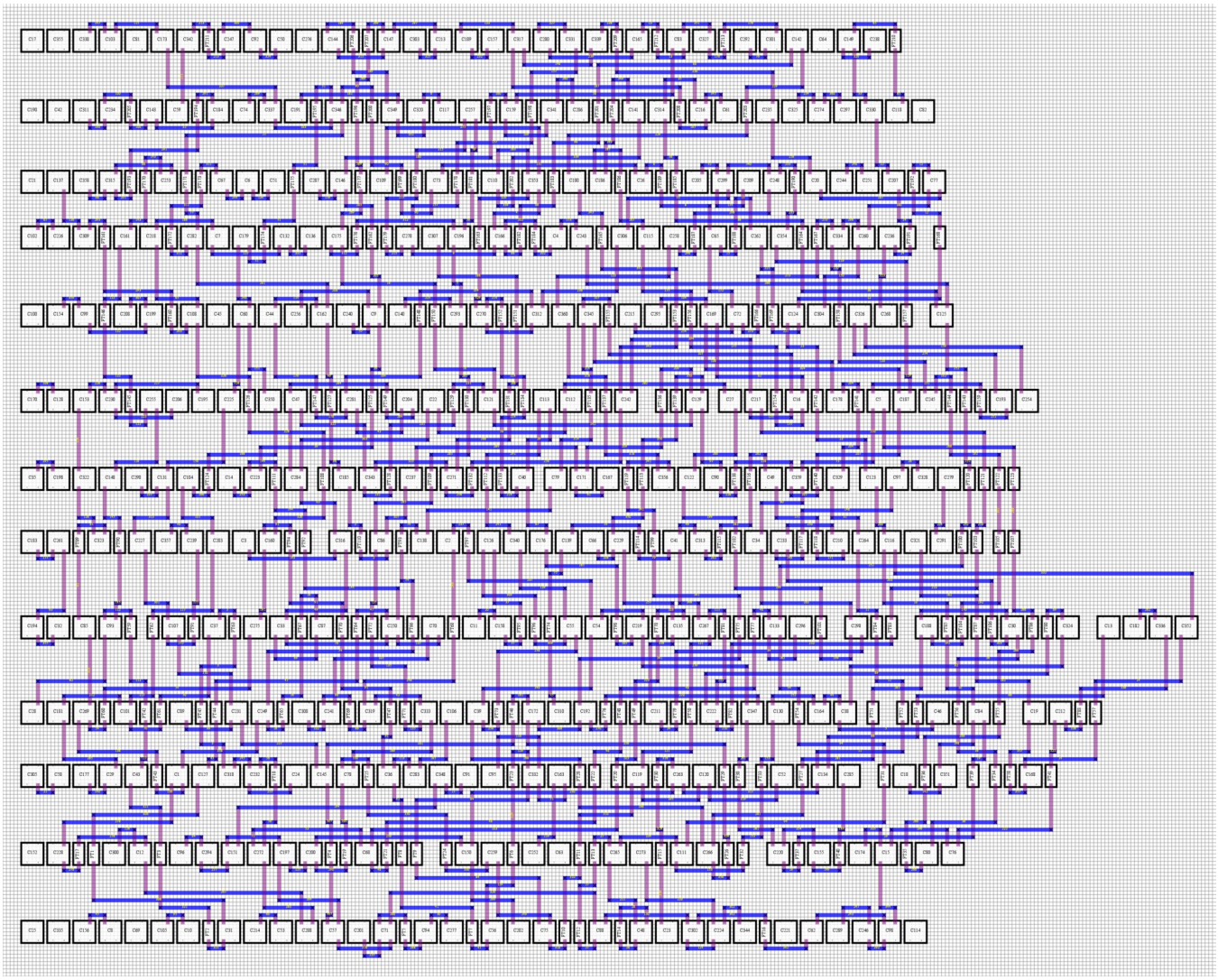

Figure 58: Benchmark 4 - Final routing

4.15 Benchmark 4 - Magic Screenshot

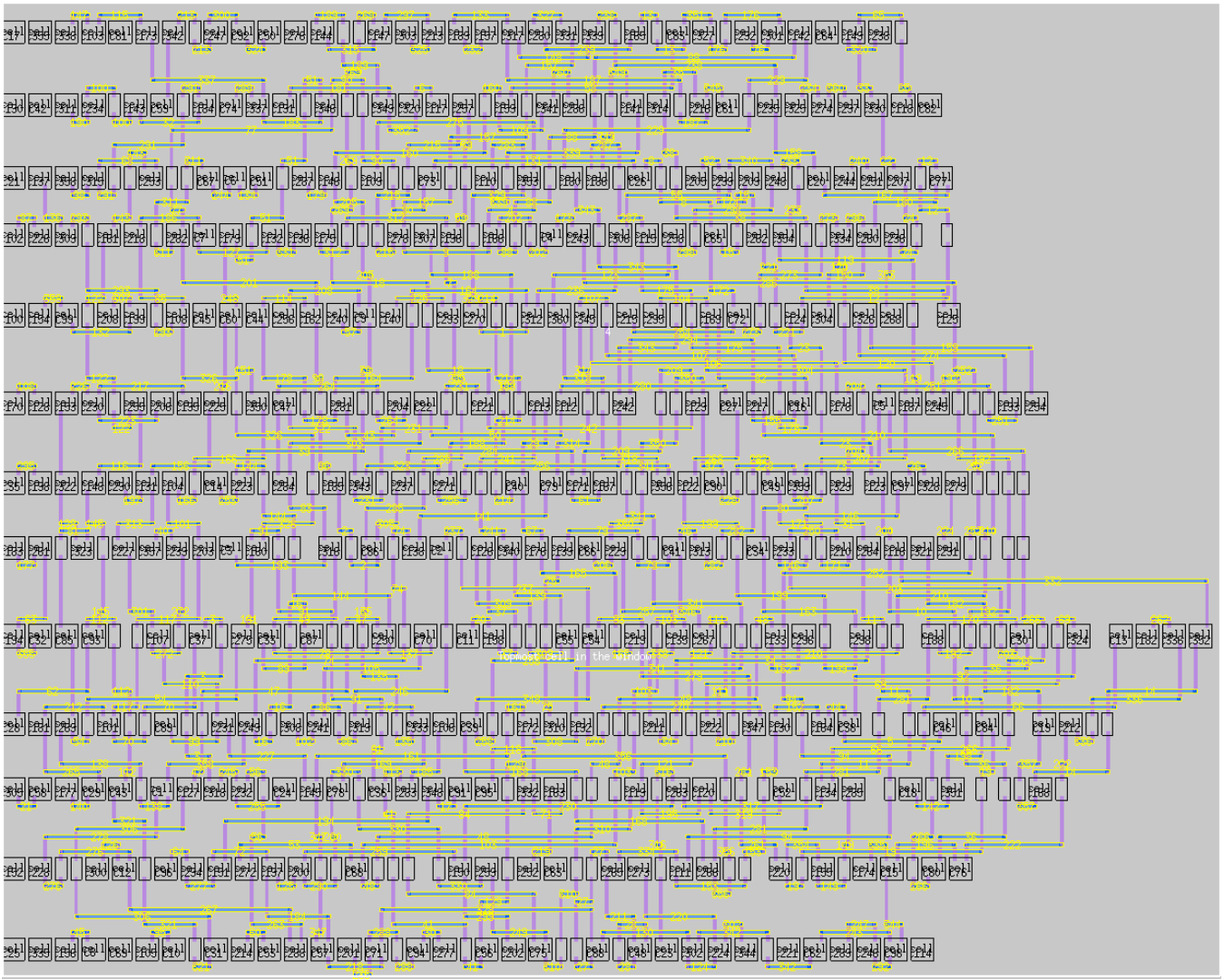

Figure 59: Benchmark 4 - Magic Screenshot

4.16 Benchmark 4 - Log file

```
Number of cell: 360
Number of nets: 360

Placement
---------
Initial grid size: 28 13
Number of feed through cells: 213

Routing
-------
Number of vias: 1126
Total number of tracks: 81
Total wirelength: 14108

Size
----
Total width: 317
Total height: 252
Total area: 79884
Squareness (width/height): 1.257937

Time
----
Place time: 0.061769s
Route time: 0.023148s
Total time: 0.110694s

Memory used: 584.000000 kB
```

Listing 4: Benchmark 4 - Log

4.17 Benchmark 5 - Placement Steps

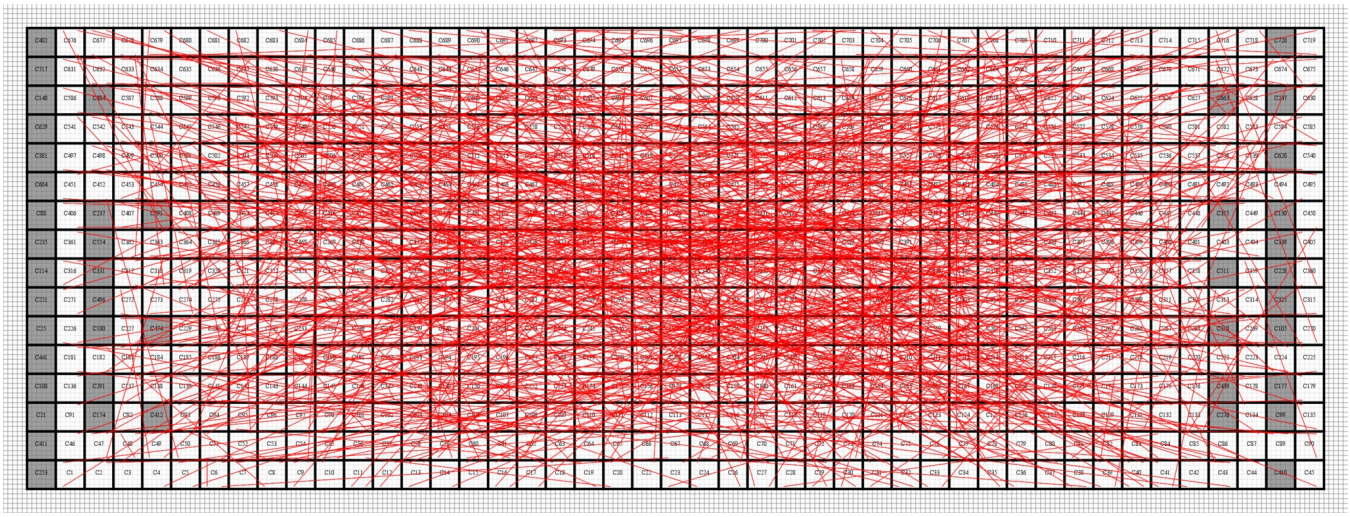

Figure 60: Benchmark 5 - Step 1/7 - Initial placement

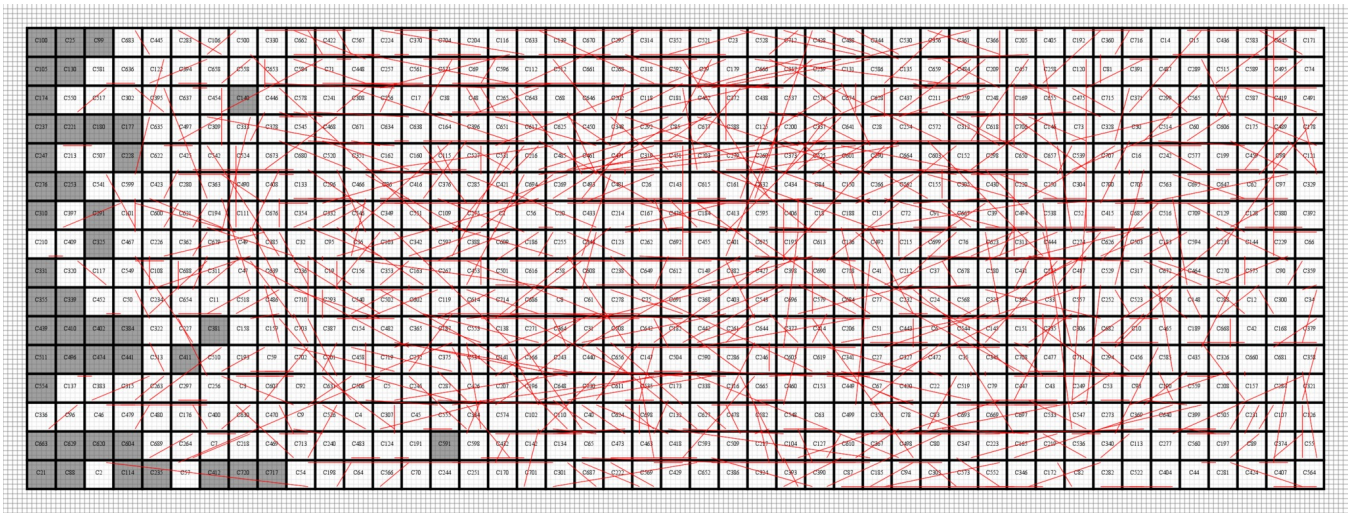

Figure 61: Benchmark 5 - Step 2/7 - Force directed placement

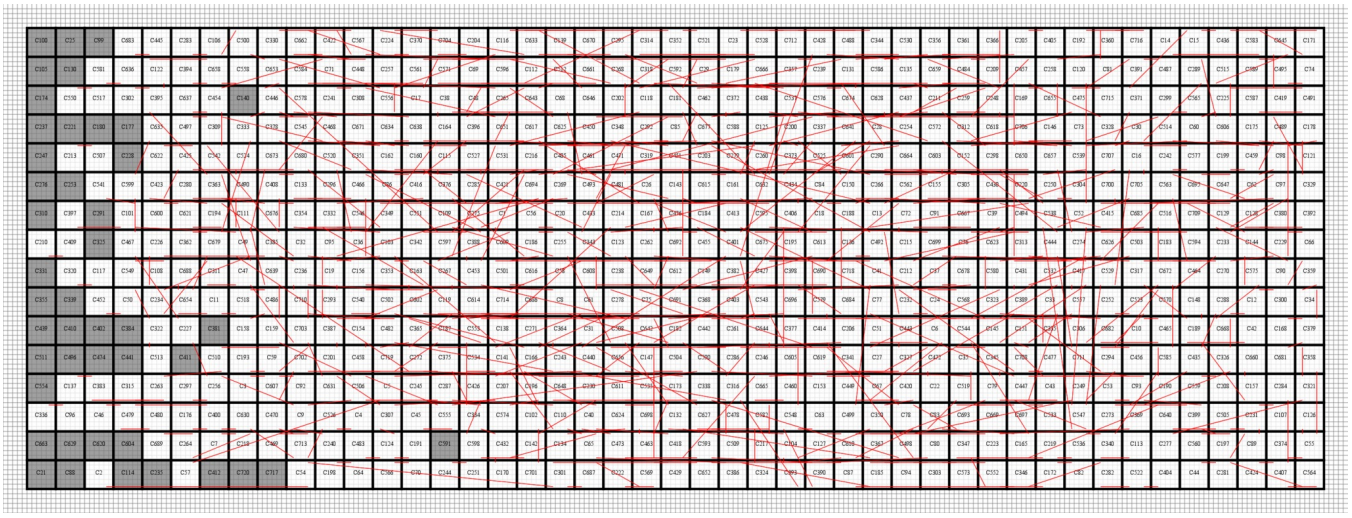

Figure 62: Benchmark 5 - Step 3/7 - Flip cells

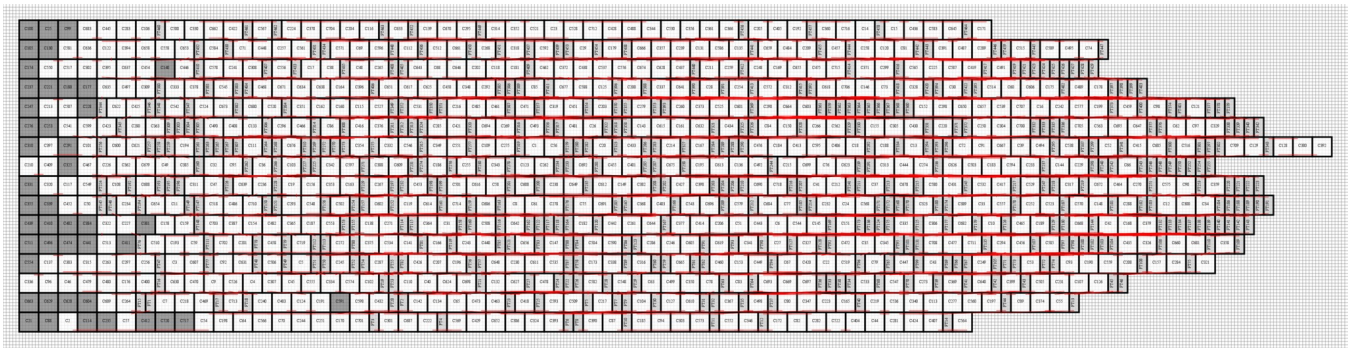

Figure 63: Benchmark 5 - Step 4/7 - Add feed-throughs

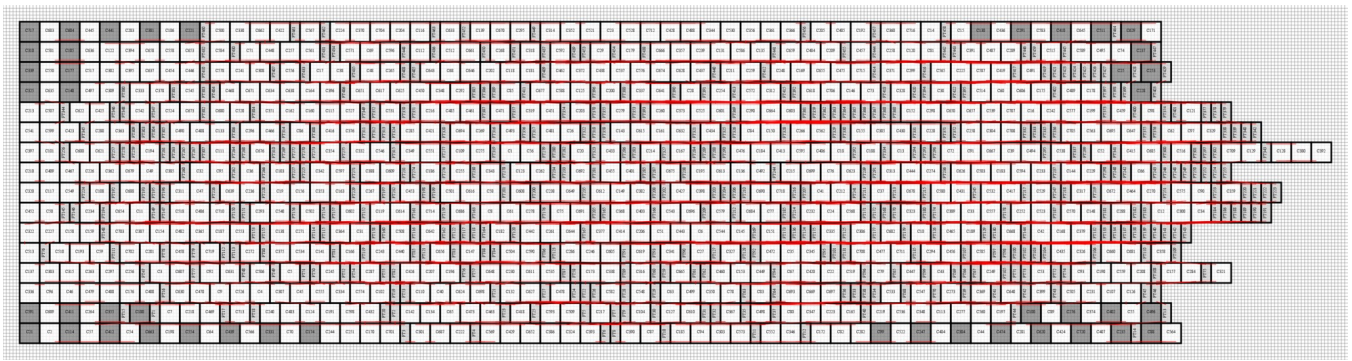

Figure 64: Benchmark 5 - Step 5/7 - Even out the row length

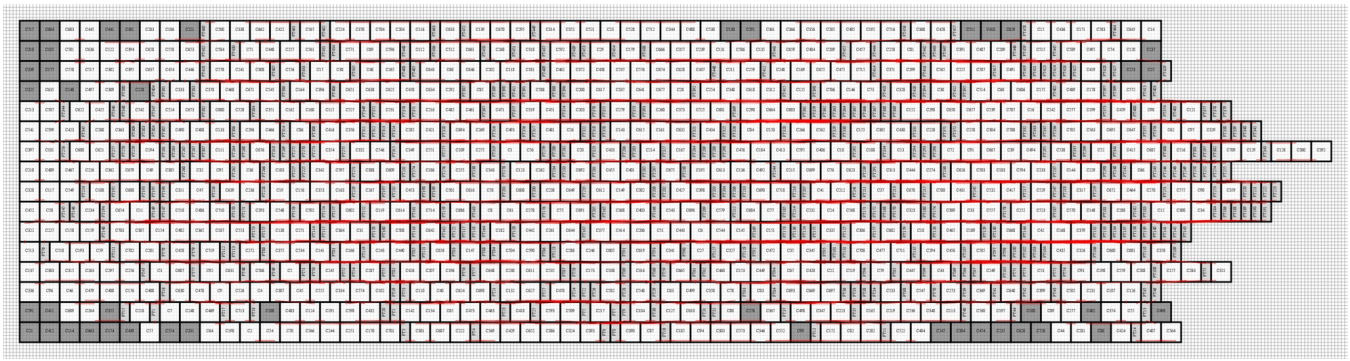

Figure 65: Benchmark 5 - Step 6/7 - Pull cells in the rows closer together

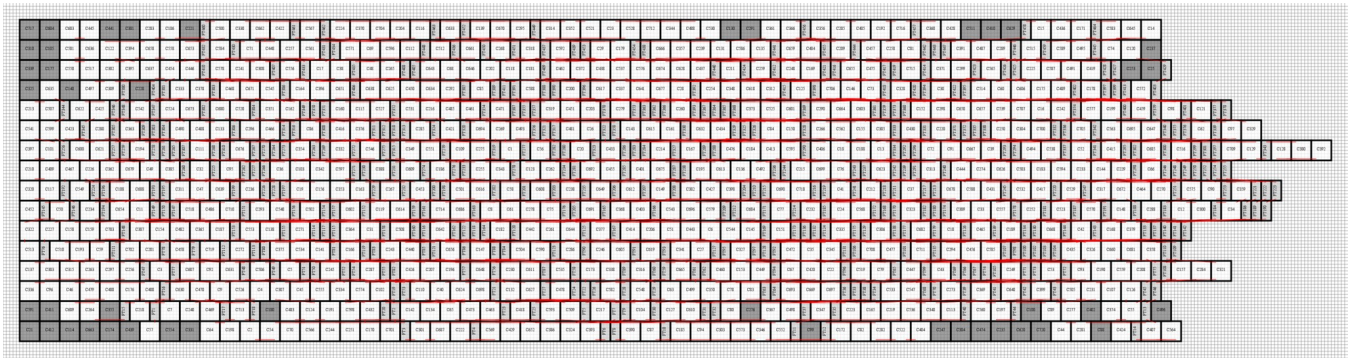

Figure 66: Benchmark 5 - Step 7/7 - Move feed-throughs to optimal location

4.18 Benchmark 5 - Final Routing

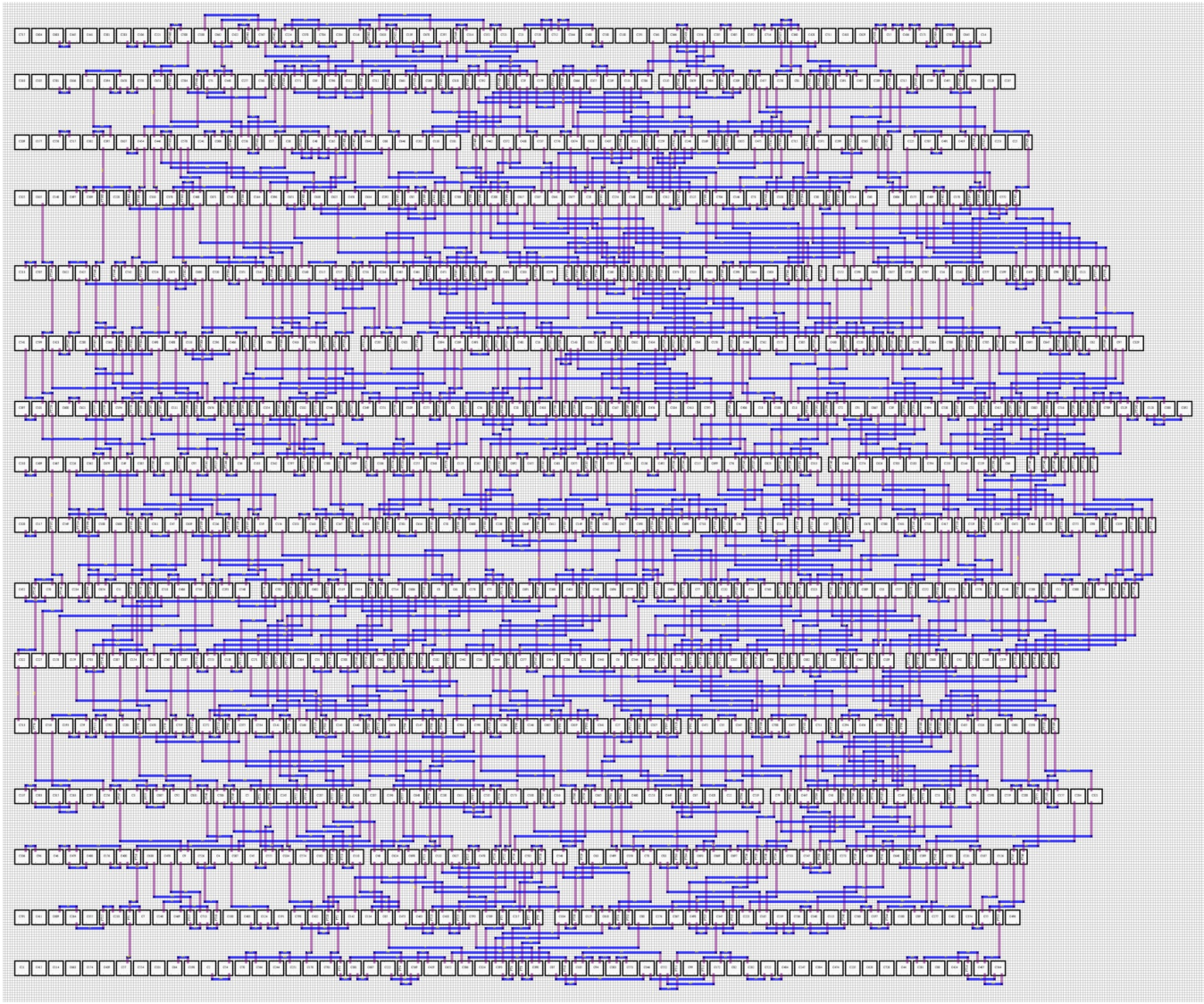

Figure 67: Benchmark 5 - Final routing

4.19 Benchmark 5 - Magic Screenshot

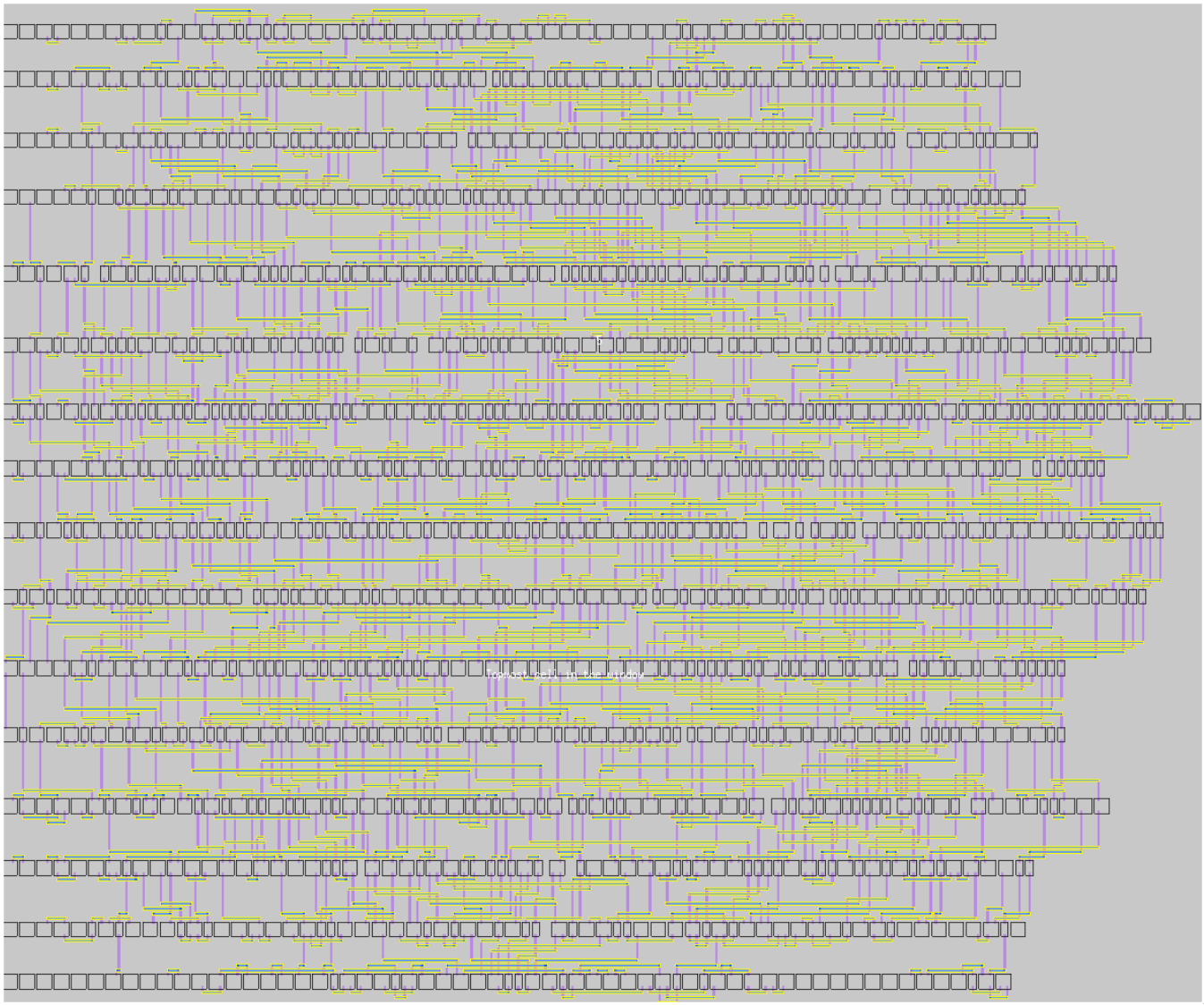

Figure 68: Benchmark 5 - Magic Screenshot

4.20 Benchmark 5 - Log file

```
Number of cell: 720
Number of nets: 720

Placement
---------
Initial grid size: 45 16
Number of feed through cells: 464

Routing
-------
Number of vias: 2336
Total number of tracks: 146
Total wirelength: 41469

Size
----
Total width: 486
Total height: 403
Total area: 195858
Squareness (width/height): 1.205955

Time
----
Place time: 0.216536s
Route time: 0.083907s
Total time: 0.353846s

Memory used: 916.000000 kB
```

Listing 5: Benchmark 5 - Log

4.21 Benchmark 6 - Placement Steps

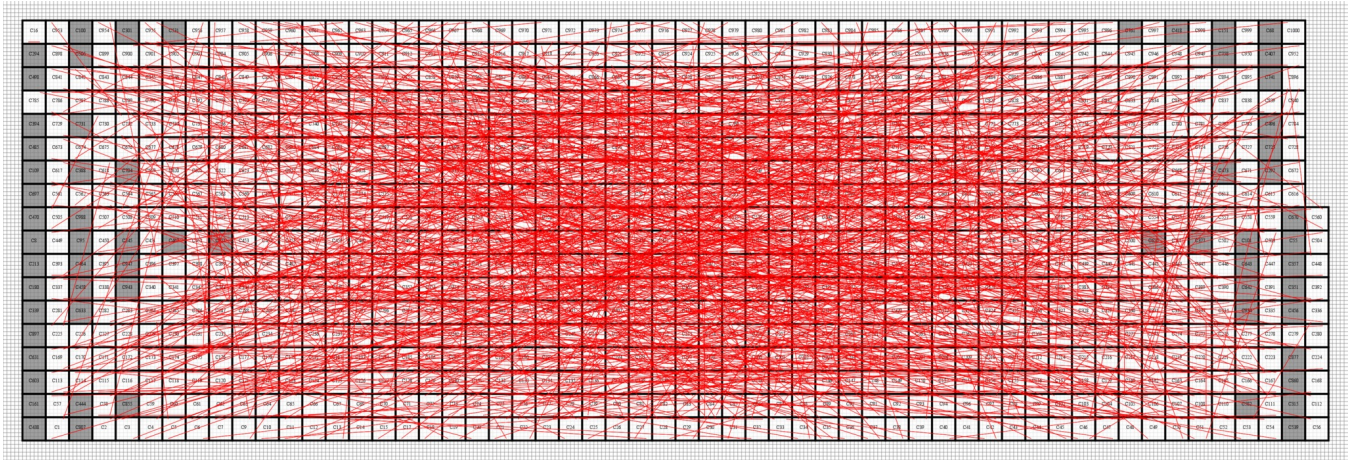

Figure 69: Benchmark 6 - Step 1/7 - Initial placement

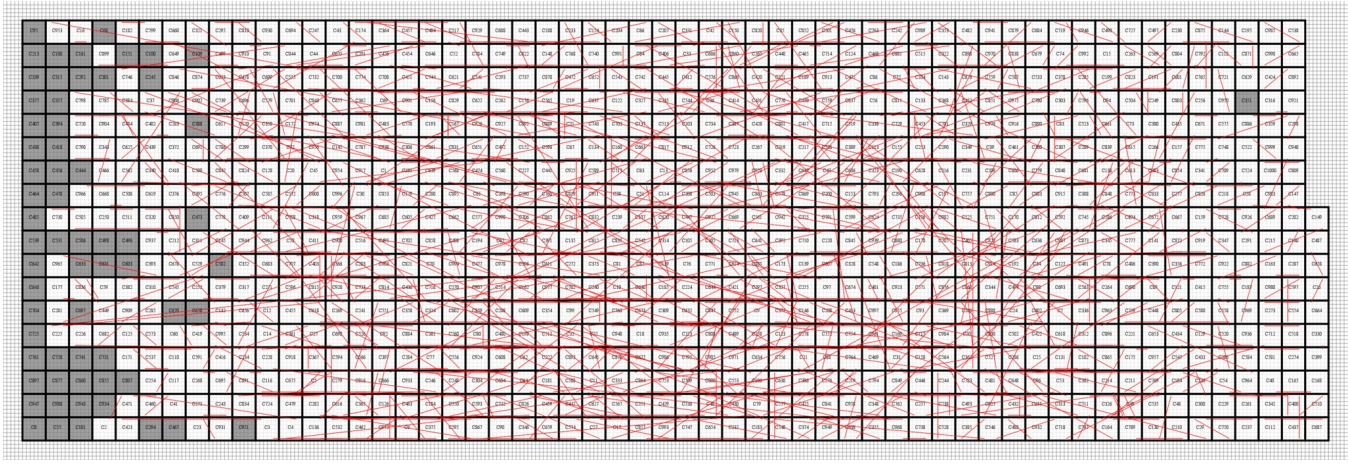

Figure 70: Benchmark 6 - Step 2/7 - Force directed placement

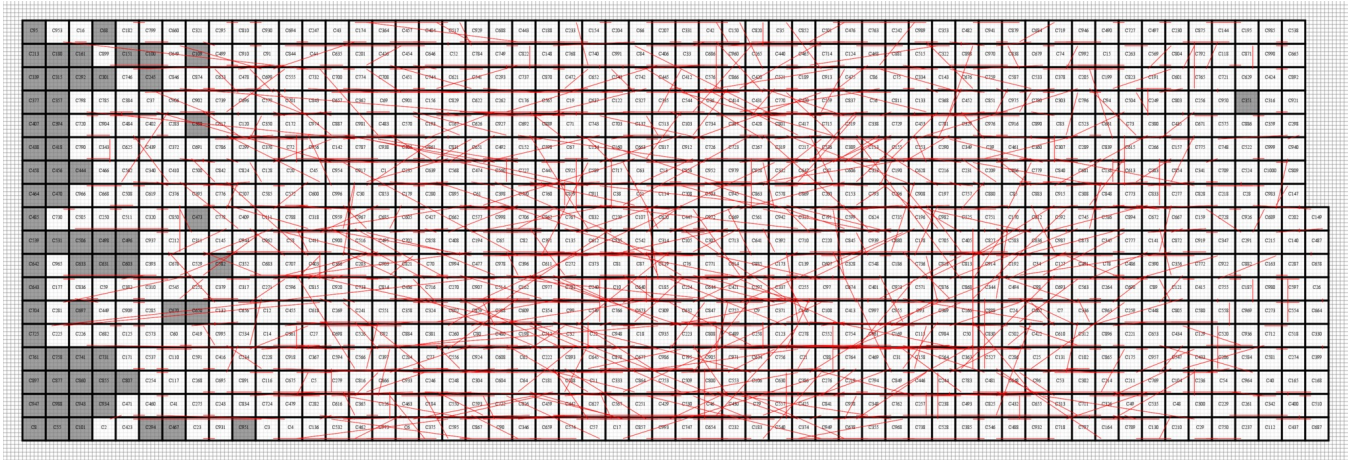

Figure 71: Benchmark 6 - Step 3/7 - Flip cells

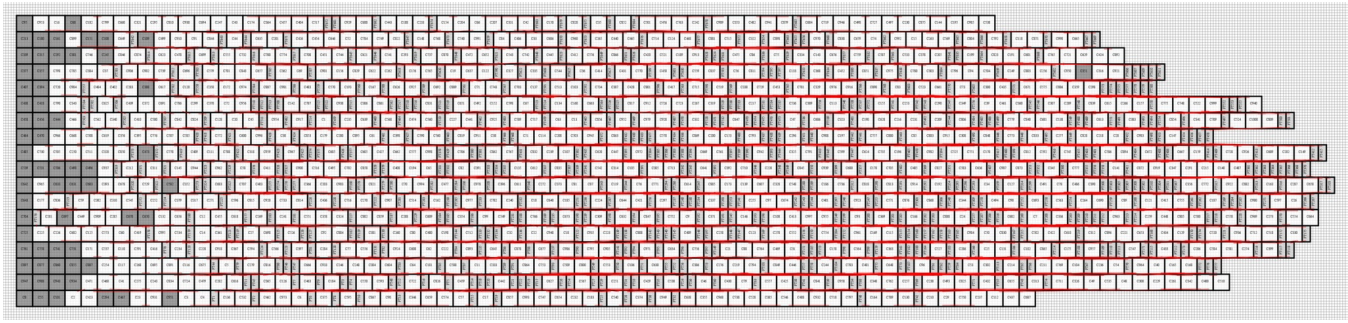

Figure 72: Benchmark 6 - Step 4/7 - Add feed-throughs

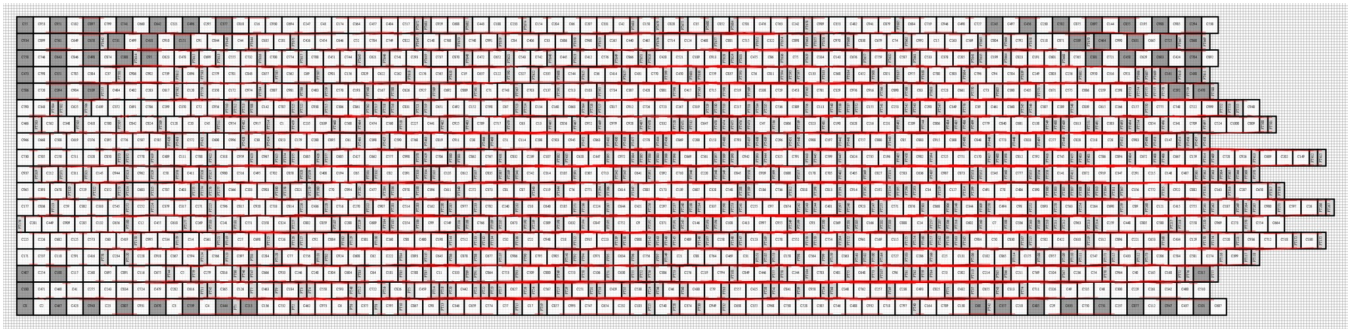

Figure 73: Benchmark 6 - Step 5/7 - Even out the row length

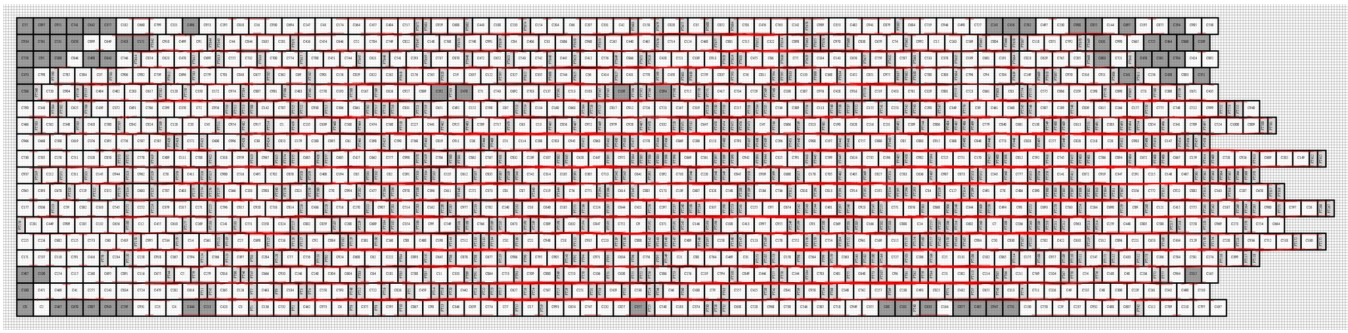

Figure 74: Benchmark 6 - Step 6/7 - Pull cells in the rows closer together

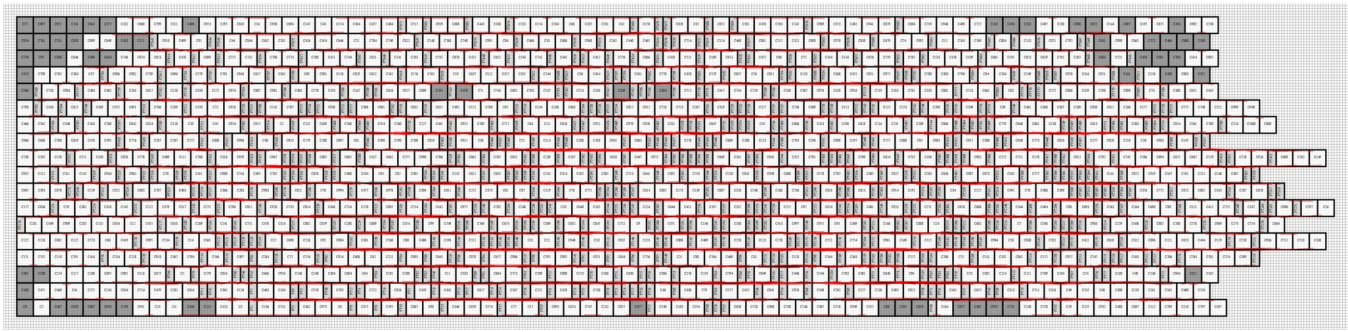

Figure 75: Benchmark 6 - Step 7/7 - Move feed-throughs to optimal location

4.22 Benchmark 6 - Final Routing

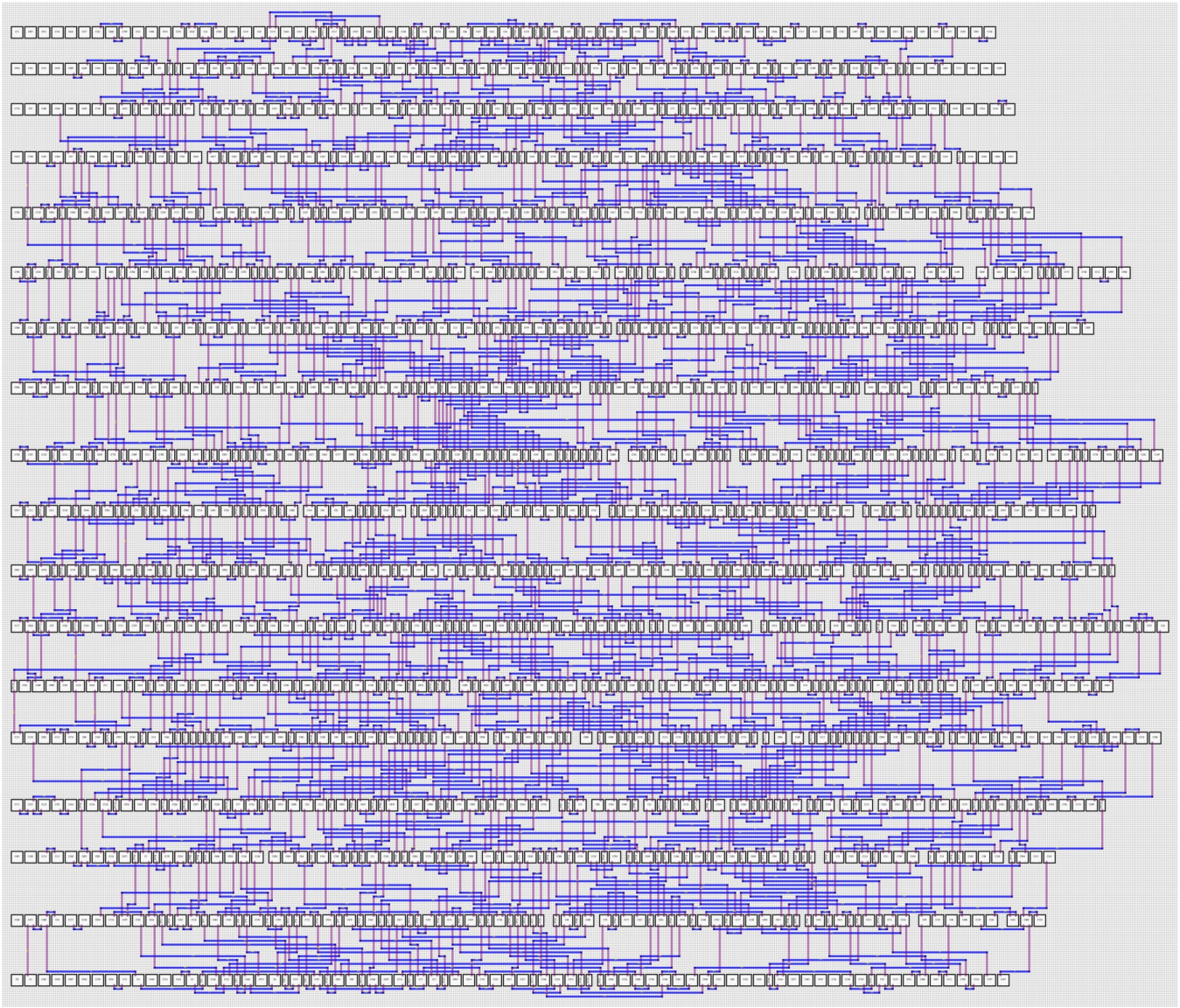

Figure 76: Benchmark 6 - Final routing

4.23 Benchmark 6 - Magic Screenshot

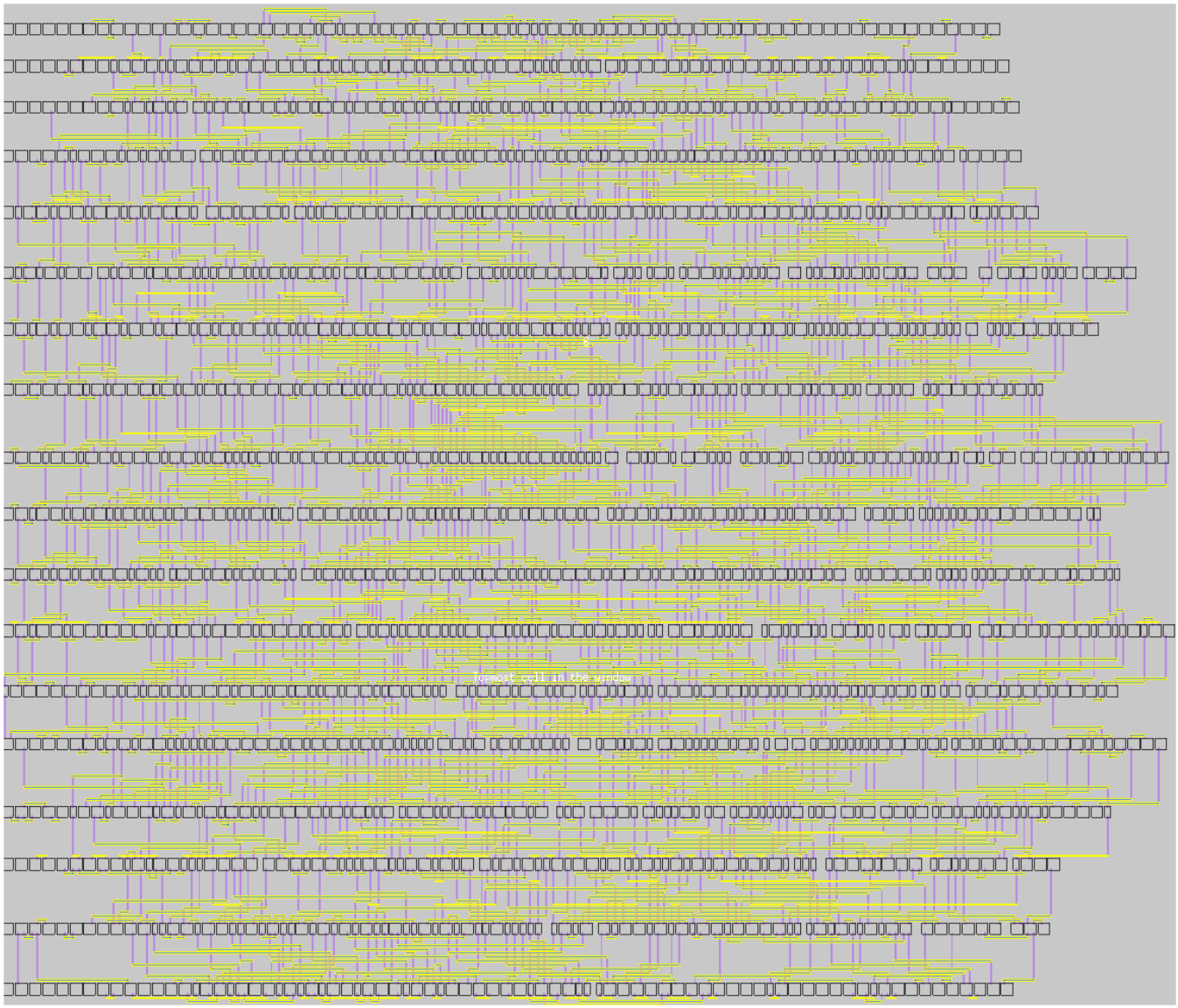

Figure 77: Benchmark 6 - Magic Screenshot

4.24 Benchmark 6 - Log file

```
Number of cell: 1000
Number of nets: 1000

Placement
---------
Initial grid size: 56 18
Number of feed through cells: 685

Routing
-------
Number of vias: 3322
Total number of tracks: 194
Total wirelength: 70536

Size
----
Total width: 602
Total height: 513
Total area: 308826
Squareness (width/height): 1.173489

Time
----
Place time: 0.424019s
Route time: 0.168323s
Total time: 0.668167s

Memory used: 1208.000000 kB
```

Listing 6: Benchmark 6 - Log

4.25 Benchmark 7 - Placement Steps

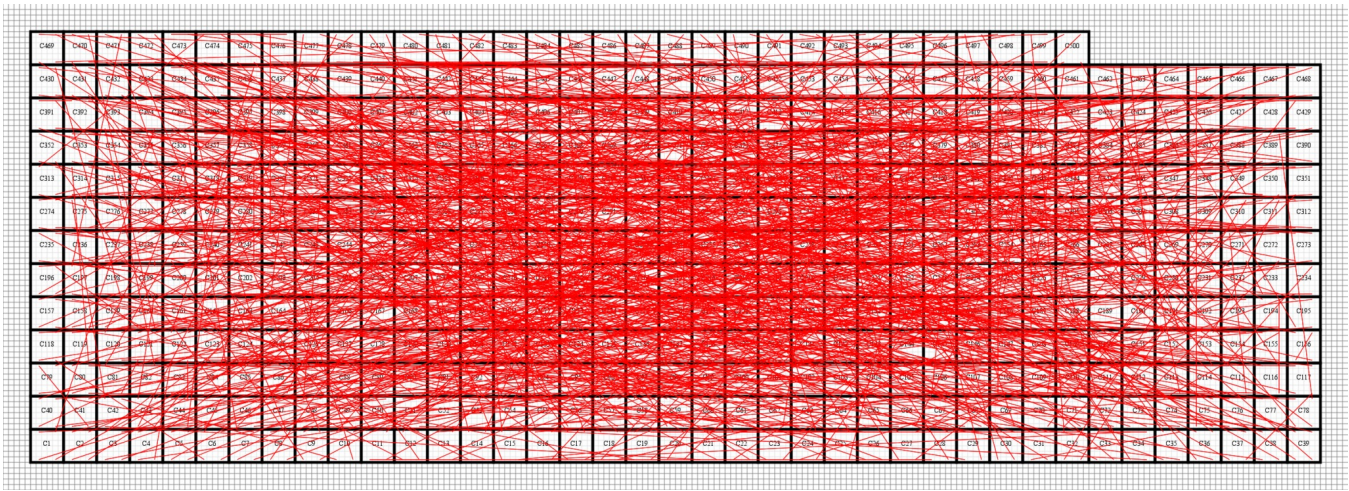

Figure 78: Benchmark 7 - Step 1/7 - Initial placement

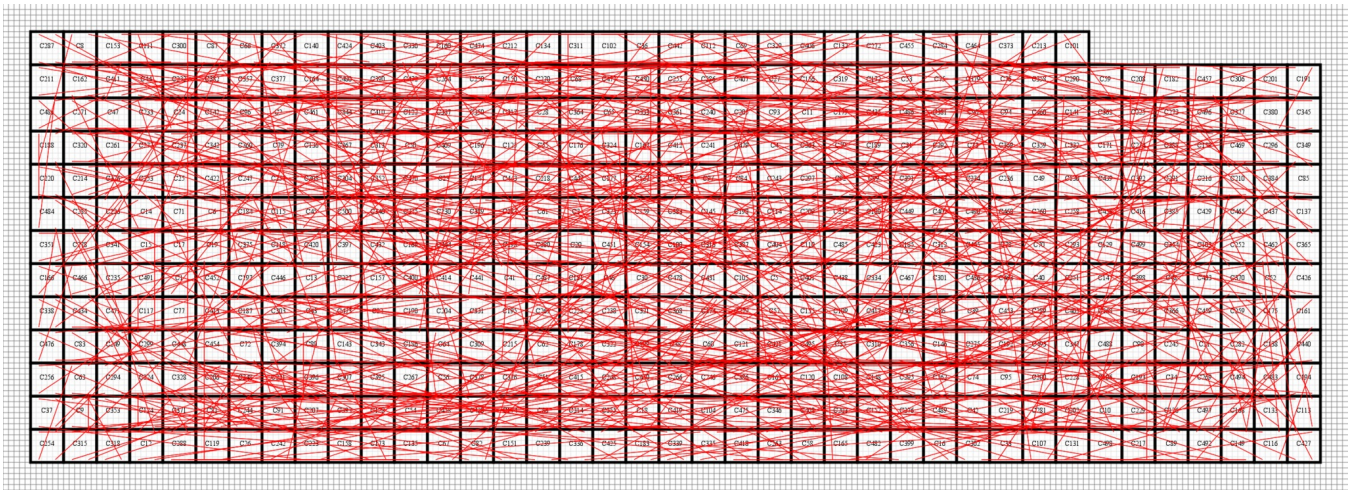

Figure 79: Benchmark 7 - Step 2/7 - Force directed placement

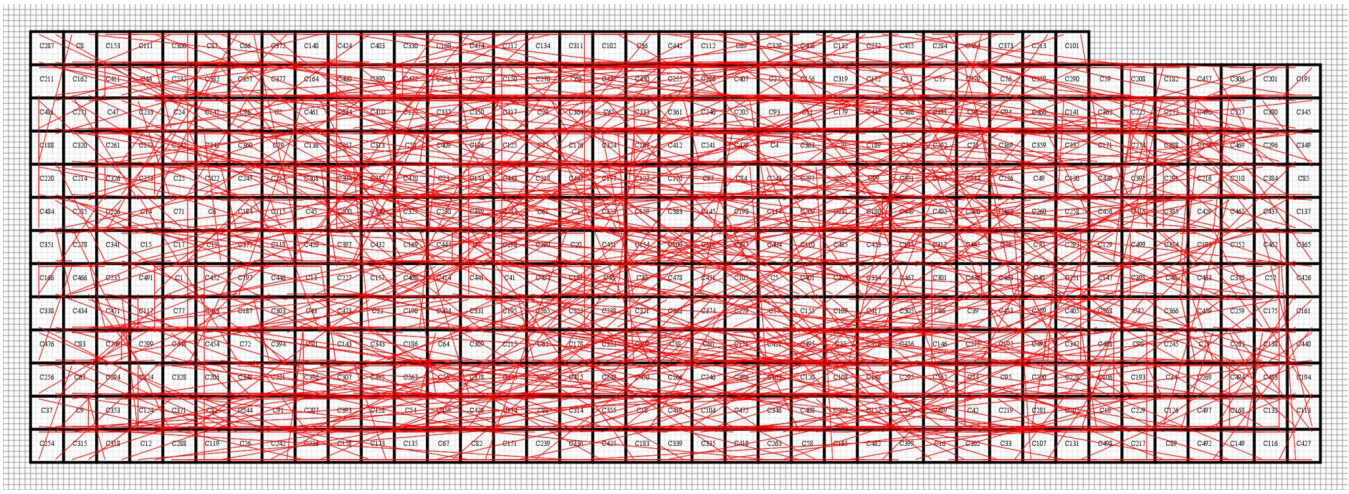

Figure 80: Benchmark 7 - Step 3/7 - Flip cells

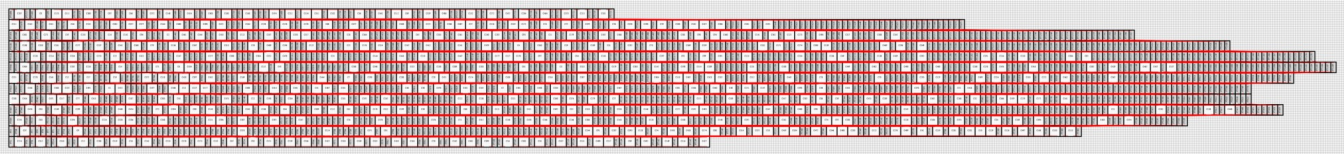

Figure 81: Benchmark 7 - Step 4/7 - Add feed-throughs

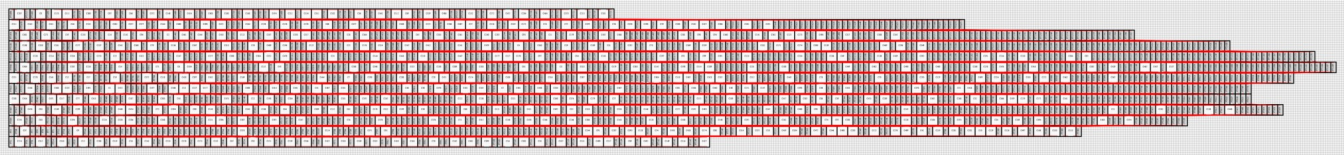

Figure 82: Benchmark 7 - Step 5/7 - Even out the row length

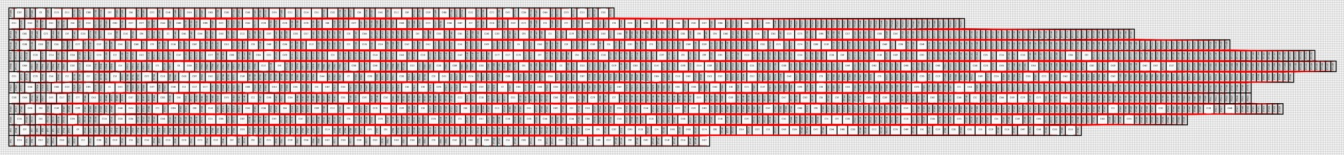

Figure 83: Benchmark 7 - Step 6/7 - Pull cells in the rows closer together

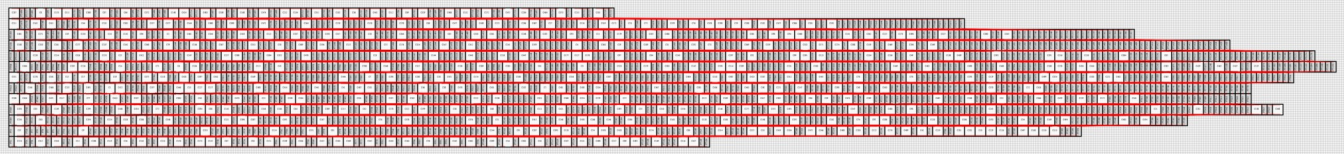

Figure 84: Benchmark 7 - Step 7/7 - Move feed-throughs to optimal location

4.26 Benchmark 7 - Final Routing

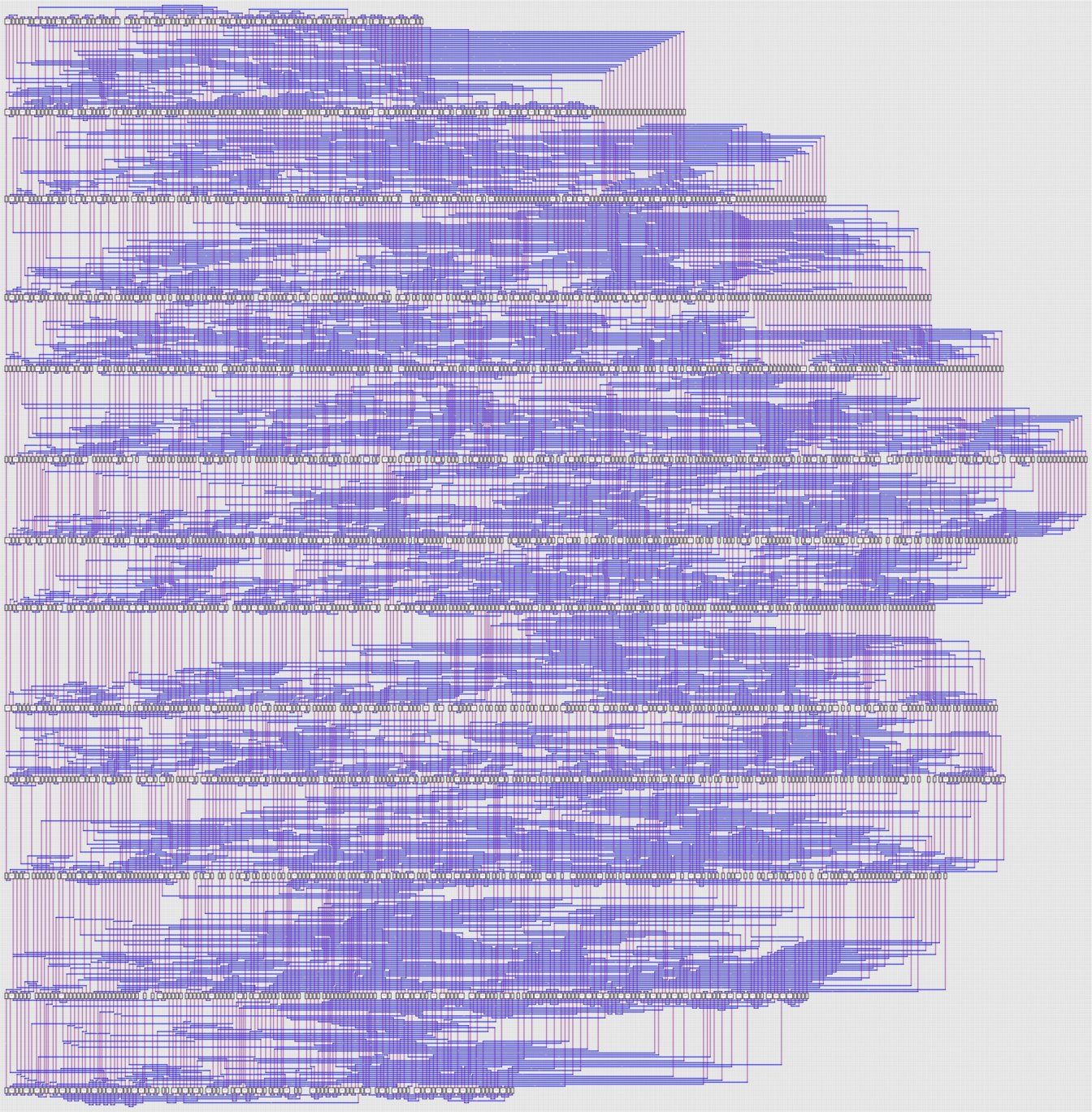

Figure 85: Benchmark 7 - Final routing

4.27 Benchmark 7 - Magic Screenshot

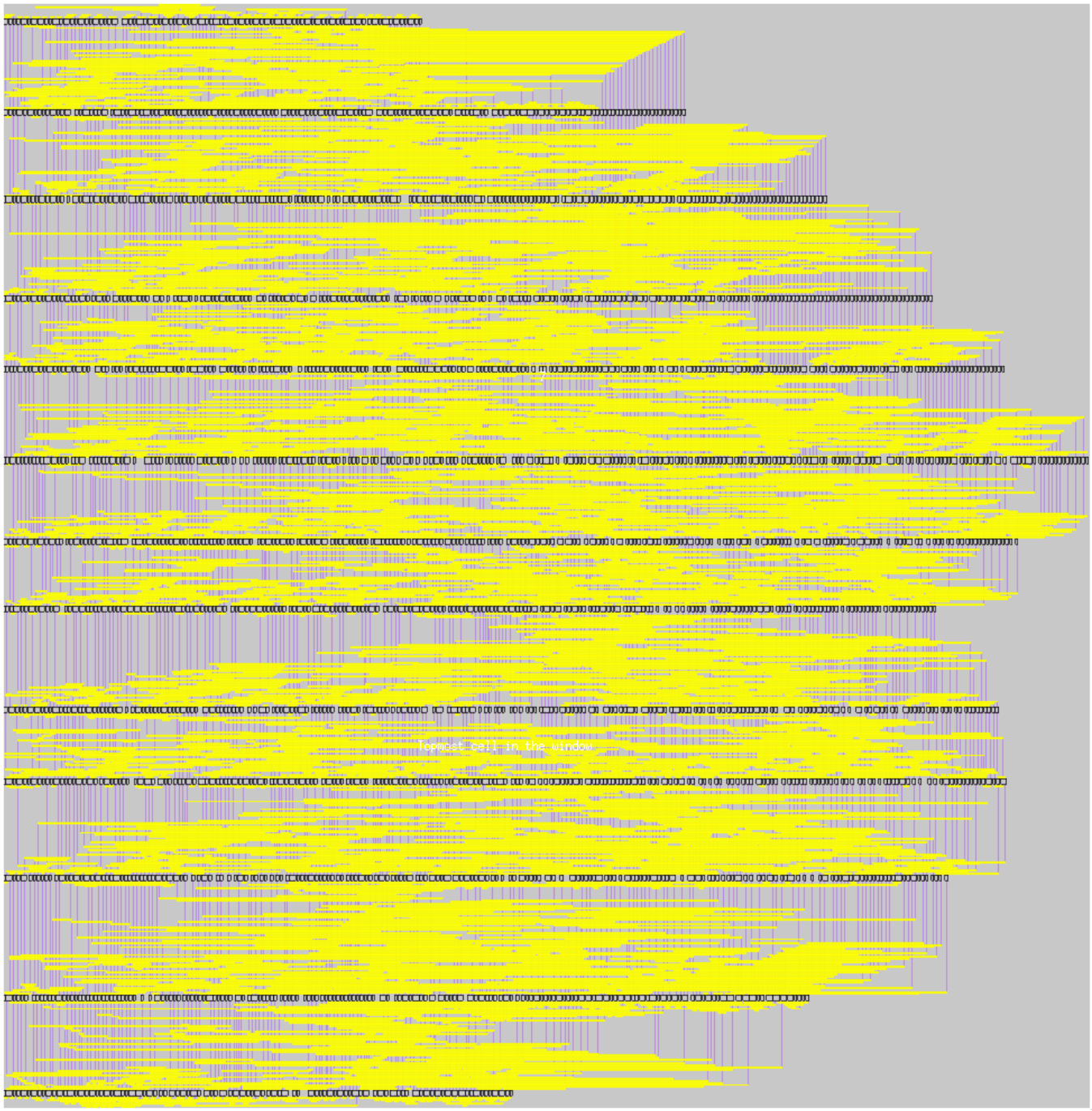

Figure 86: Benchmark 7 - Magic Screenshot

4.28 Benchmark 7 - Log file

```
Number of cell: 500
Number of nets: 1000

Placement
---------
Initial grid size: 39 13
Number of feed through cells: 1738

Routing
-------
Number of vias: 5448
Total number of tracks: 520
Total wirelength: 443767

Size
----
Total width: 1110
Total height: 1130
Total area: 1254300
Squareness (width/height): 0.982301

Time
----
Place time: 0.430179s
Route time: 1.023264s
Total time: 1.569695s

Memory used: 1512.000000 kB
```

Listing 7: Benchmark 7 - Log

4.29 Benchmark 8 - Placement Steps

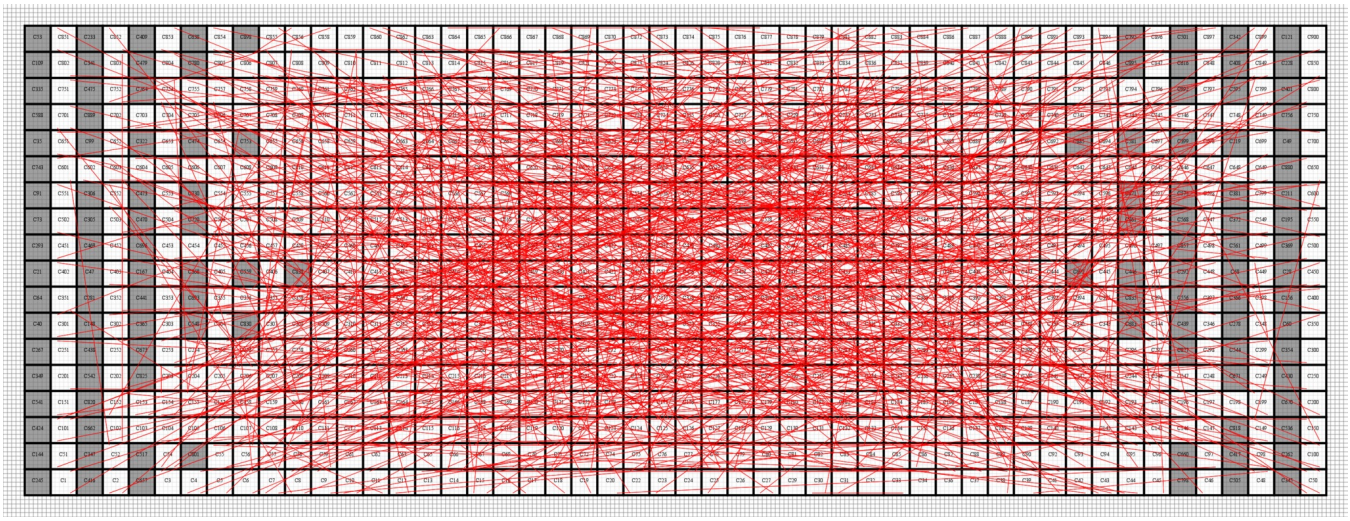

Figure 87: Benchmark 8 - Step 1/7 - Initial placement

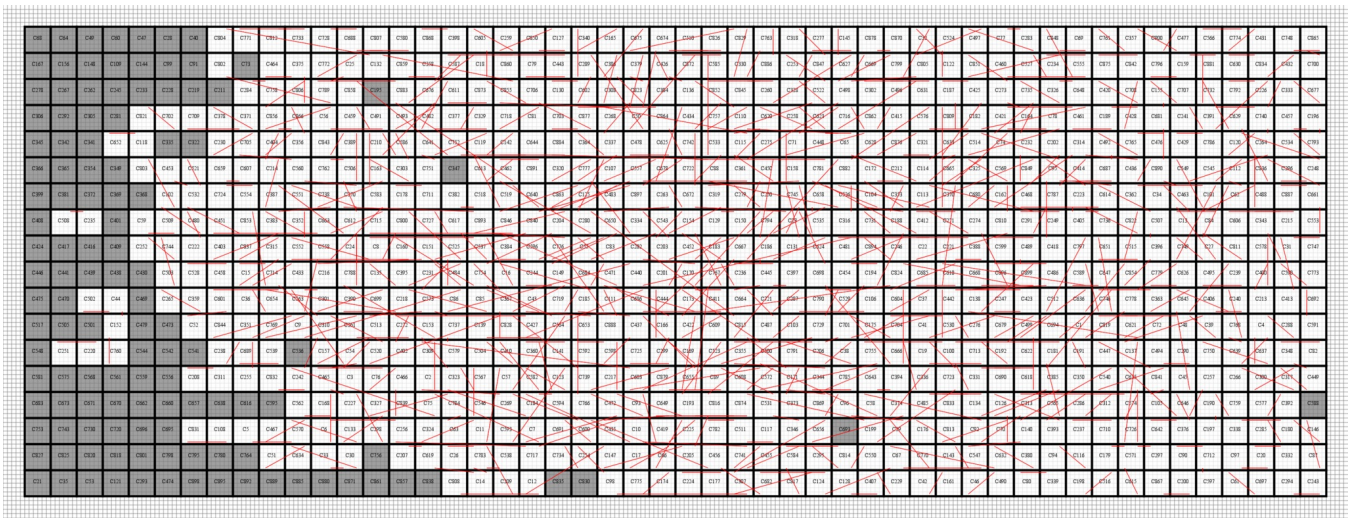

Figure 88: Benchmark 8 - Step 2/7 - Force directed placement

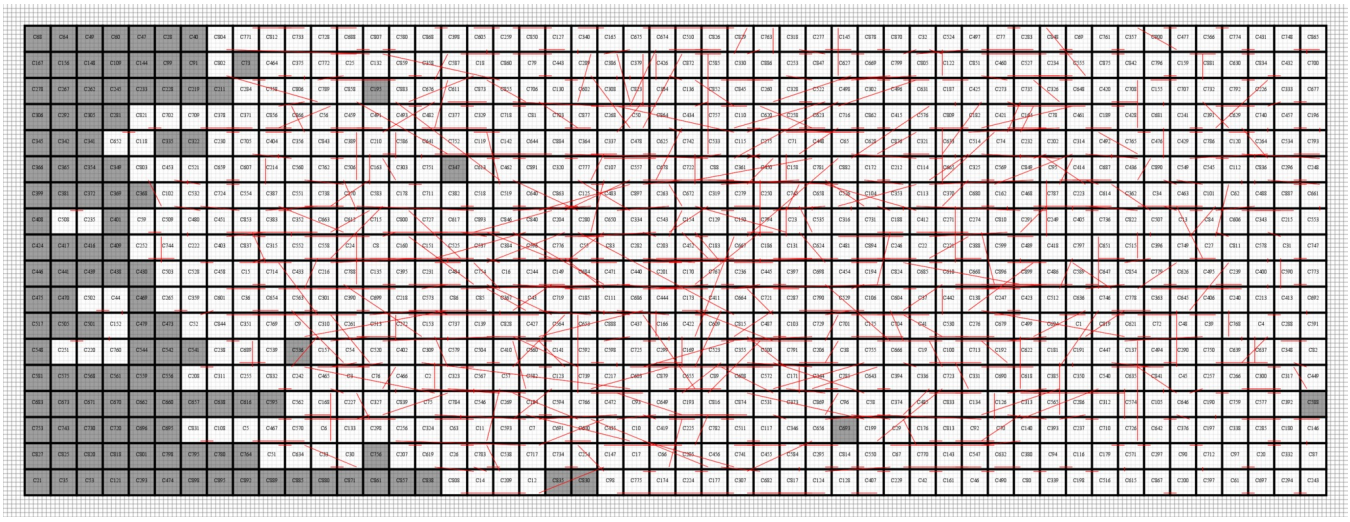

Figure 89: Benchmark 8 - Step 3/7 - Flip cells

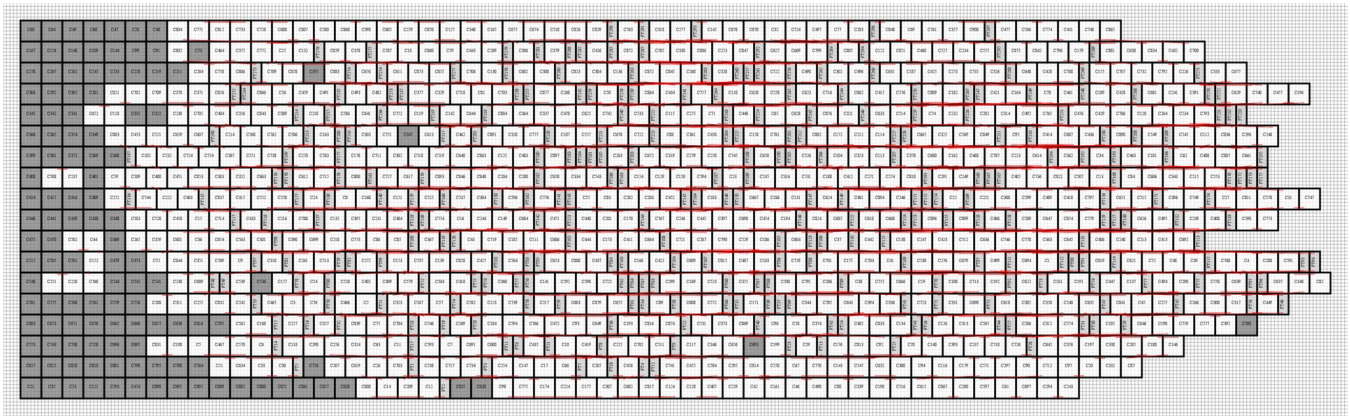

Figure 90: Benchmark 8 - Step 4/7 - Add feed-throughs

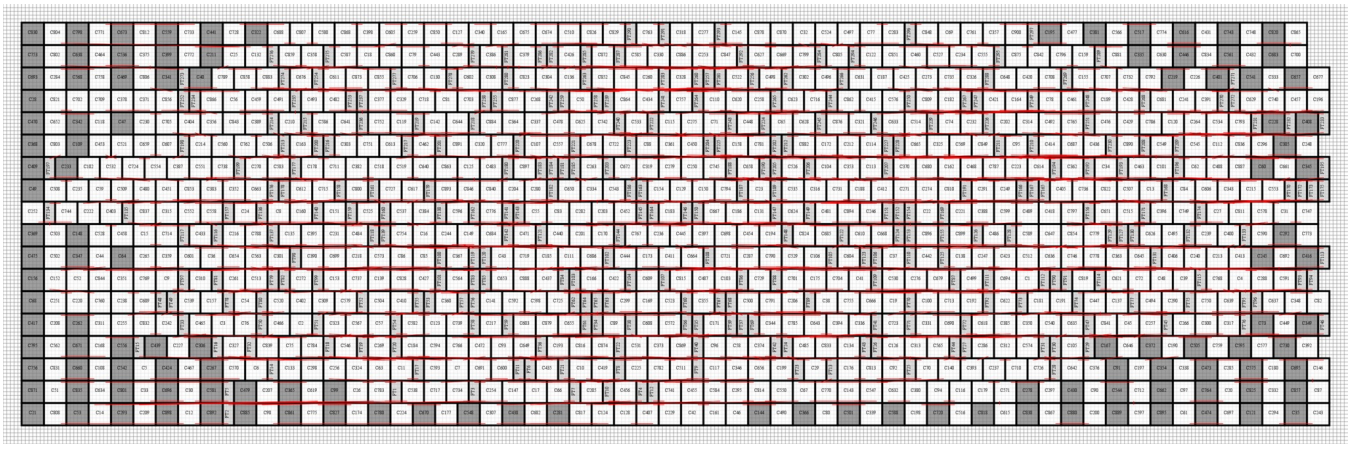

Figure 91: Benchmark 8 - Step 5/7 - Even out the row length

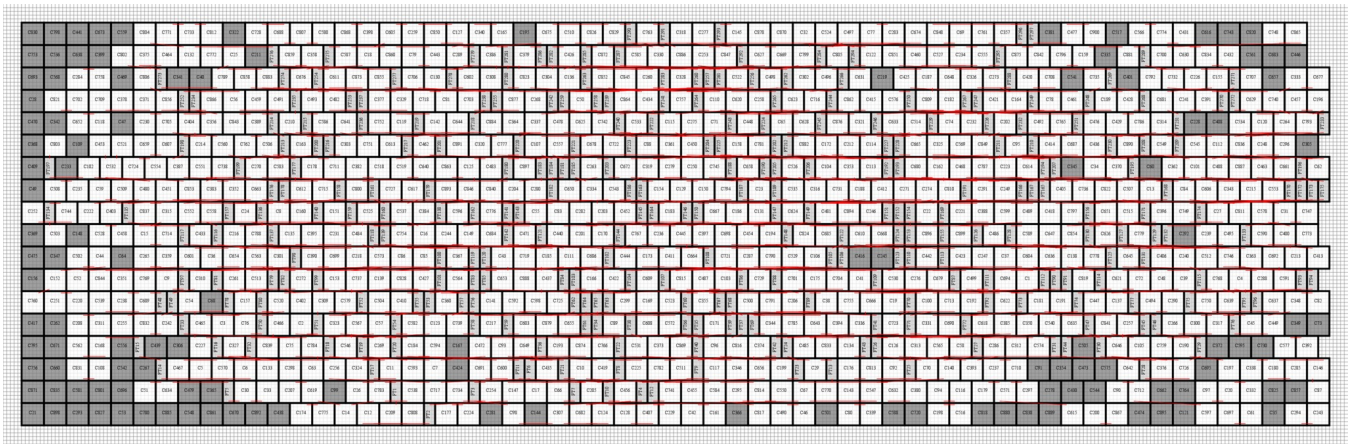

Figure 92: Benchmark 8 - Step 6/7 - Pull cells in the rows closer together

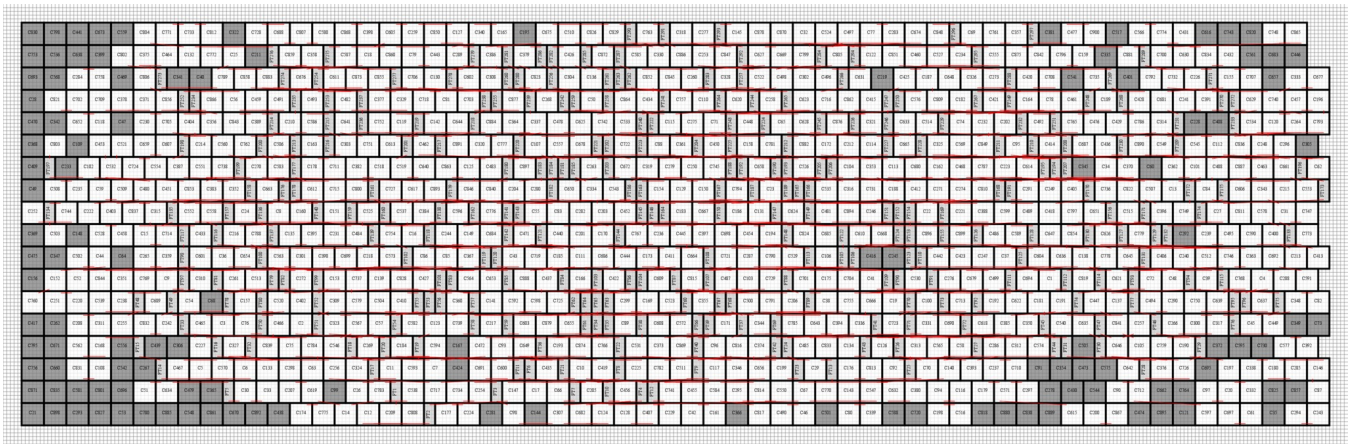

Figure 93: Benchmark 8 - Step 7/7 - Move feed-throughs to optimal location

4.30 Benchmark 8 - Final Routing

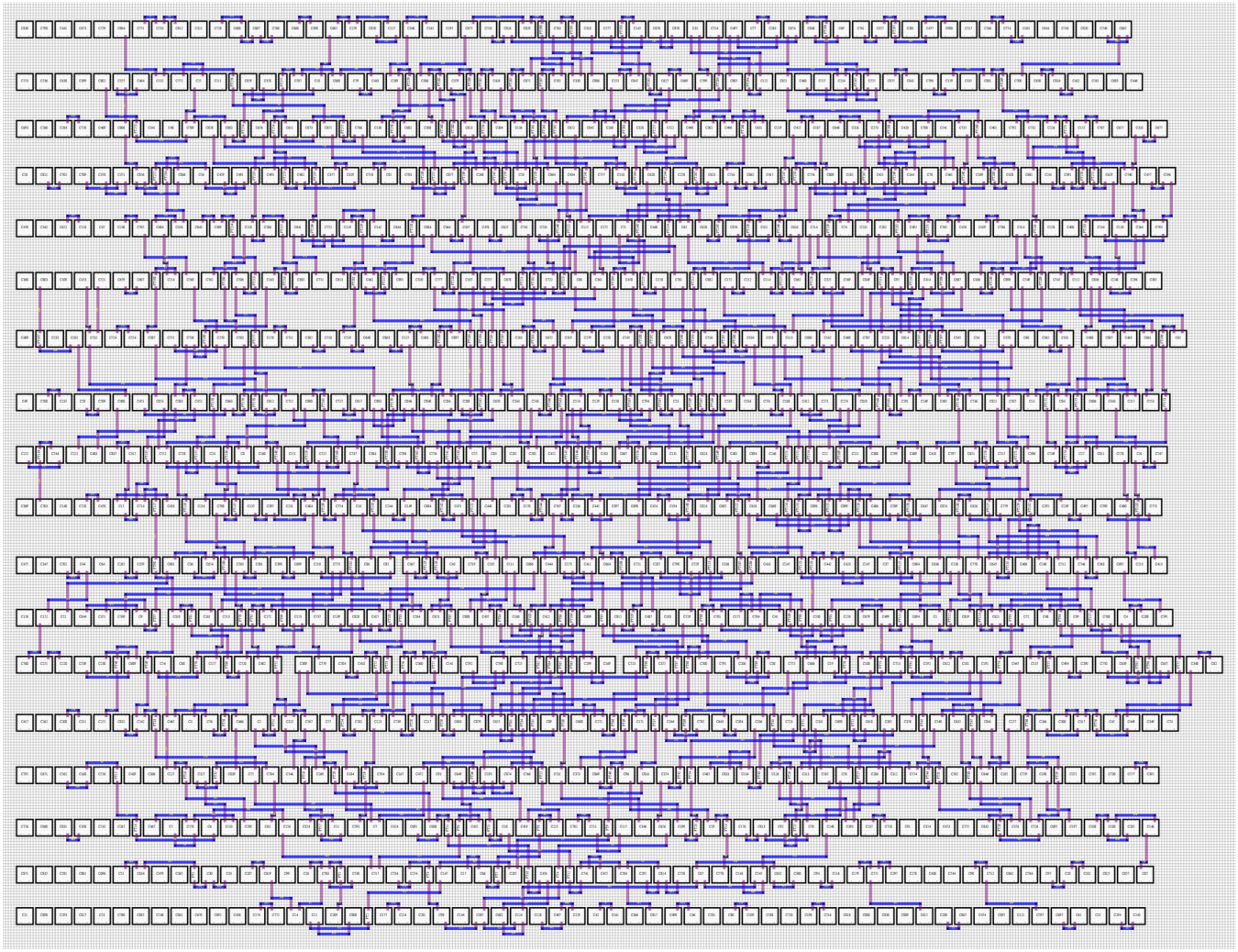

Figure 94: Benchmark 8 - Final routing

4.31 Benchmark 8 - Magic Screenshot

Figure 95: Benchmark 8 - Magic Screenshot

4.32 Benchmark 8 - Log file

```
Number of cell: 900
Number of nets: 720

Placement
---------
Initial grid size: 50 18
Number of feed through cells: 297

Routing
-------
Number of vias: 1994
Total number of tracks: 104
Total wirelength: 22907

Size
----
Total width: 437
Total height: 333
Total area: 145521
Squareness (width/height): 1.312312

Time
----
Place time: 0.238644s
Route time: 0.051257s
Total time: 0.339161s

Memory used: 916.000000 kB
```

Listing 8: Benchmark 8 - Log

4.33 Benchmark 9 - Placement Steps

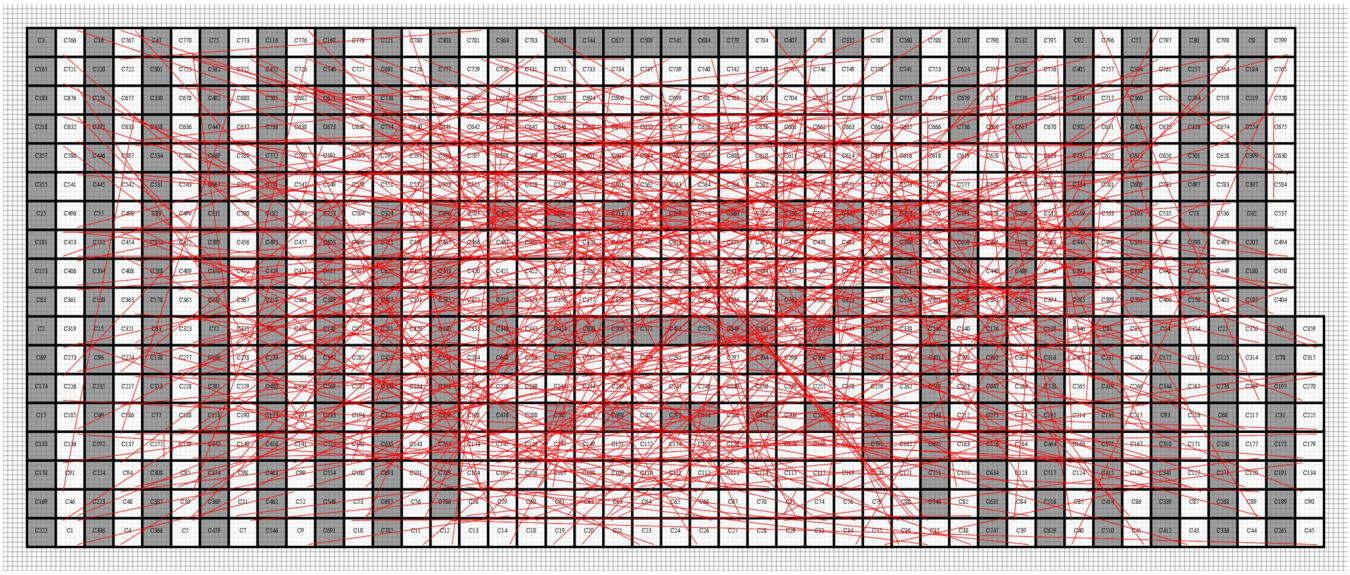

Figure 96: Benchmark 9 - Step 1/7 - Initial placement

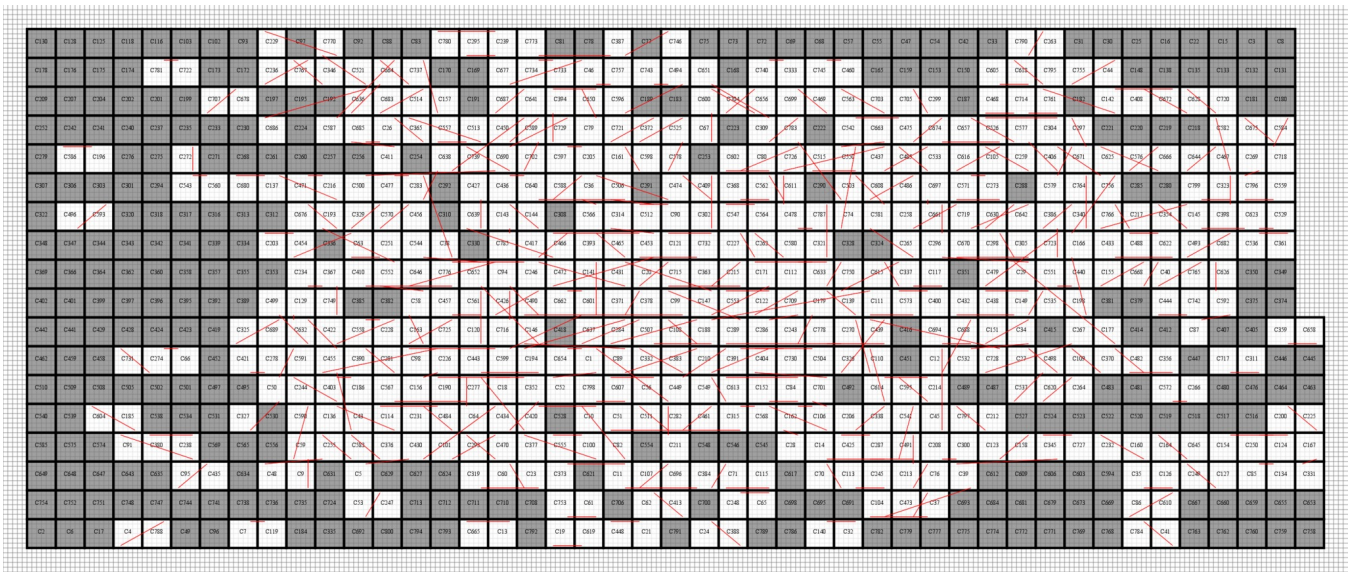

Figure 97: Benchmark 9 - Step 2/7 - Force directed placement

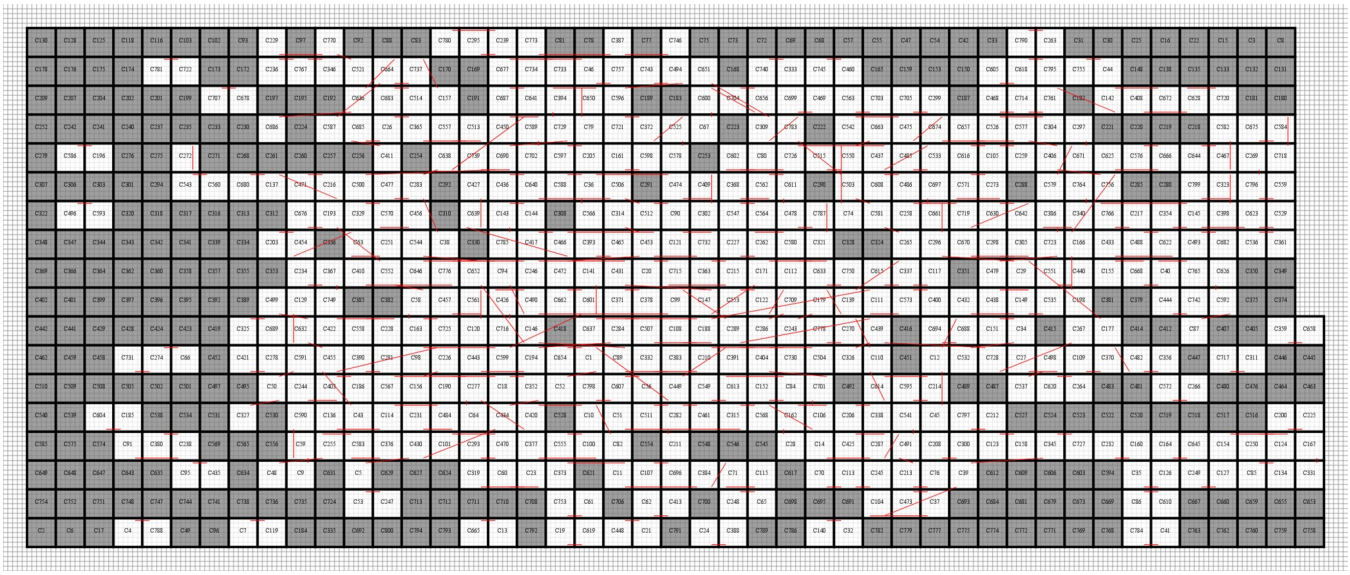

Figure 98: Benchmark 9 - Step 3/7 - Flip cells

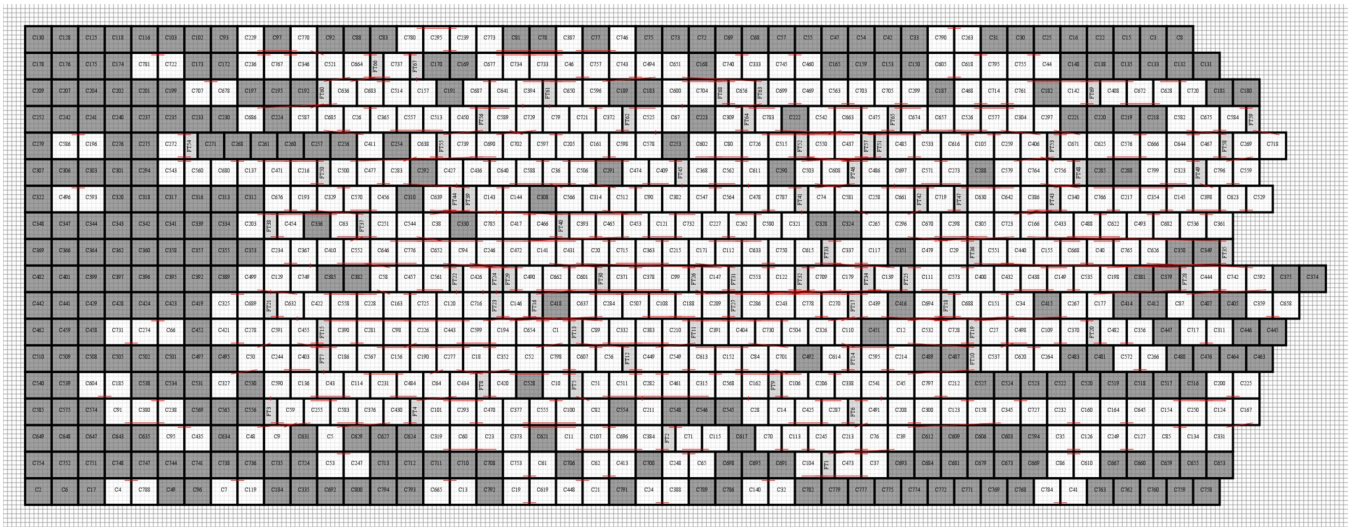

Figure 99: Benchmark 9 - Step 4/7 - Add feed-throughs

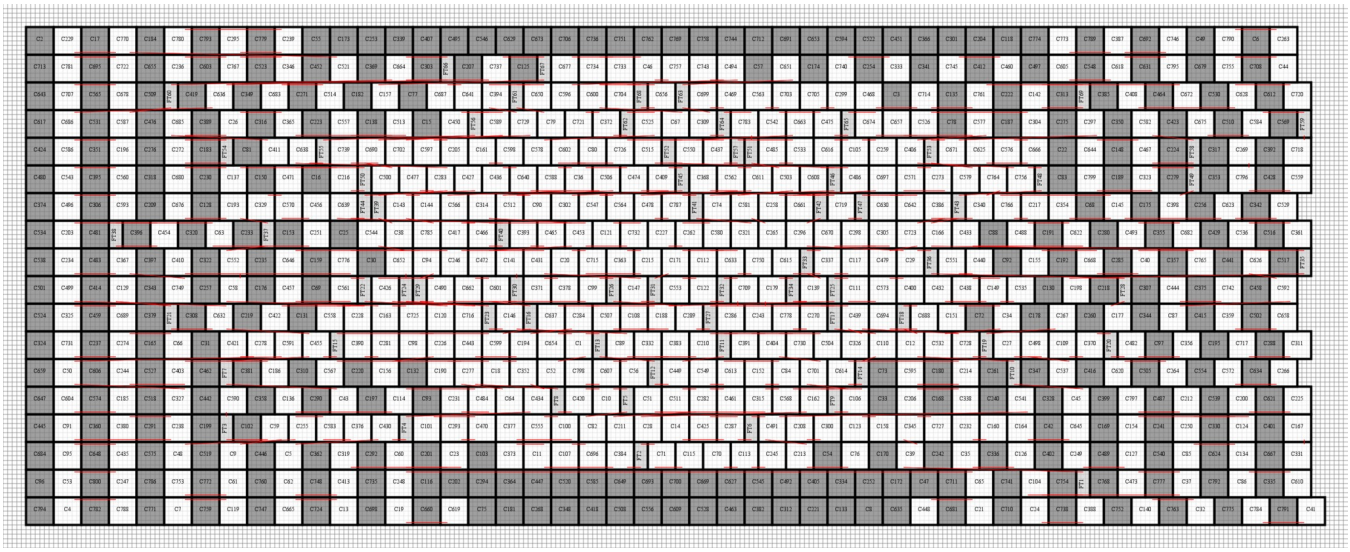

Figure 100: Benchmark 9 - Step 5/7 - Even out the row length

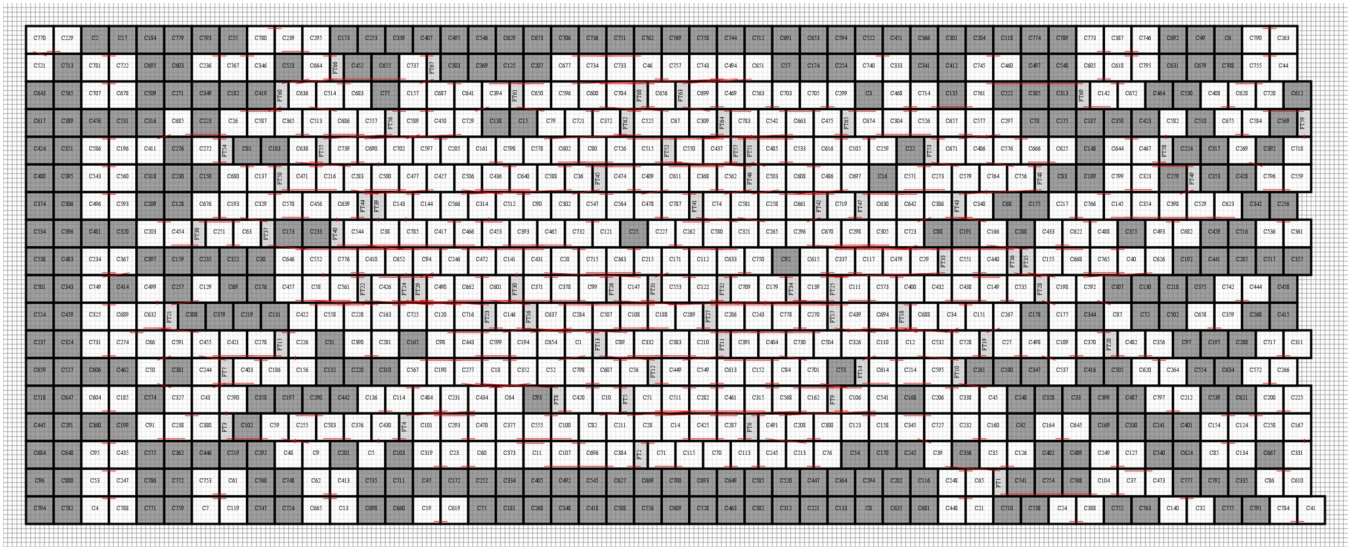

Figure 101: Benchmark 9 - Step 6/7 - Pull cells in the rows closer together

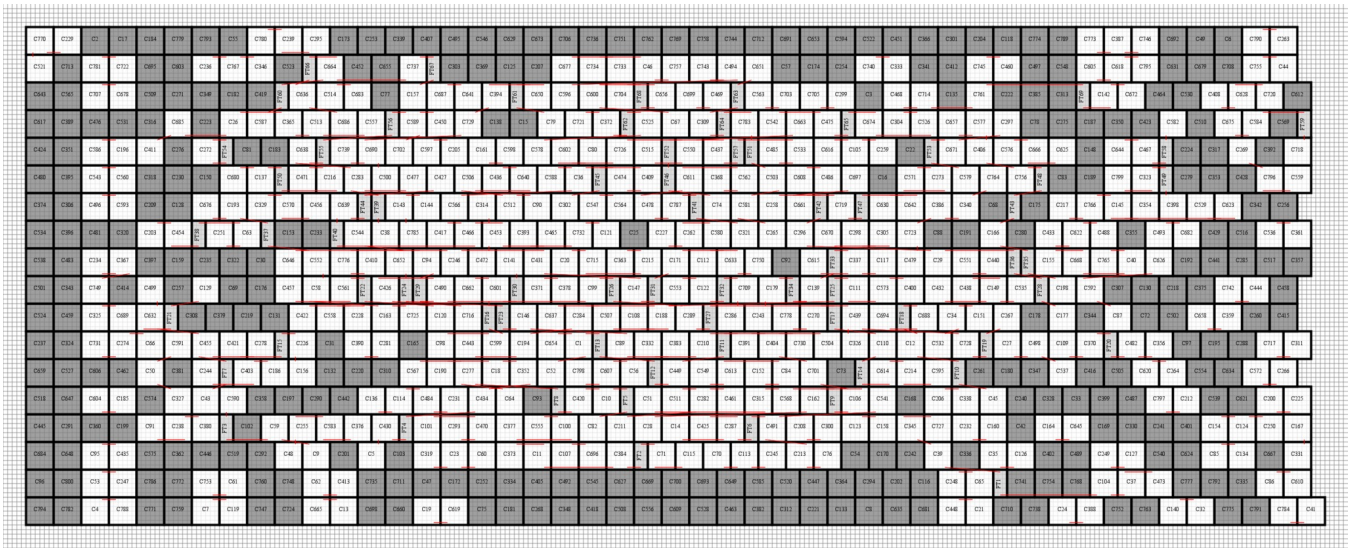

Figure 102: Benchmark 9 - Step 7/7 - Move feed-throughs to optimal location

4.34 Benchmark 9 - Final Routing

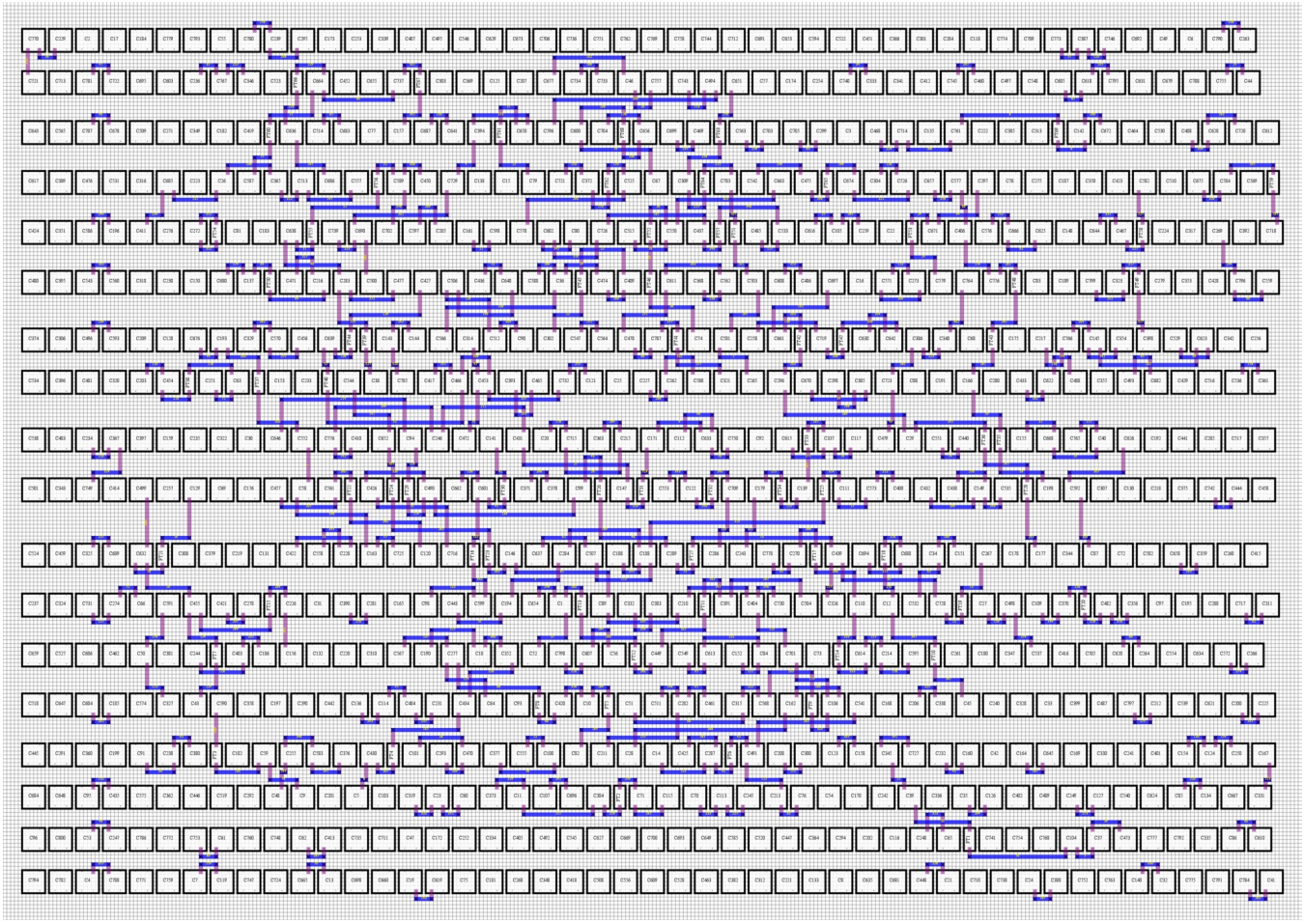

Figure 103: Benchmark 9 - Final routing

4.35 Benchmark 9 - Magic Screenshot

Figure 104: Benchmark 9 - Magic Screenshot

4.36 Benchmark 9 - Log file

```
Number of cell: 800
Number of nets: 360

Placement
---------
Initial grid size: 45 18
Number of feed through cells: 69

Routing
-------
Number of vias: 842
Total number of tracks: 52
Total wirelength: 6097

Size
----
Total width: 328
Total height: 229
Total area: 75112
Squareness (width/height): 1.432314

Time
----
Place time: 0.165468s
Route time: 0.011609s
Total time: 0.204302s

Memory used: 668.000000 kB
```

Listing 9: Benchmark 9 - Log

4.37 Benchmark 10 - Placement Steps

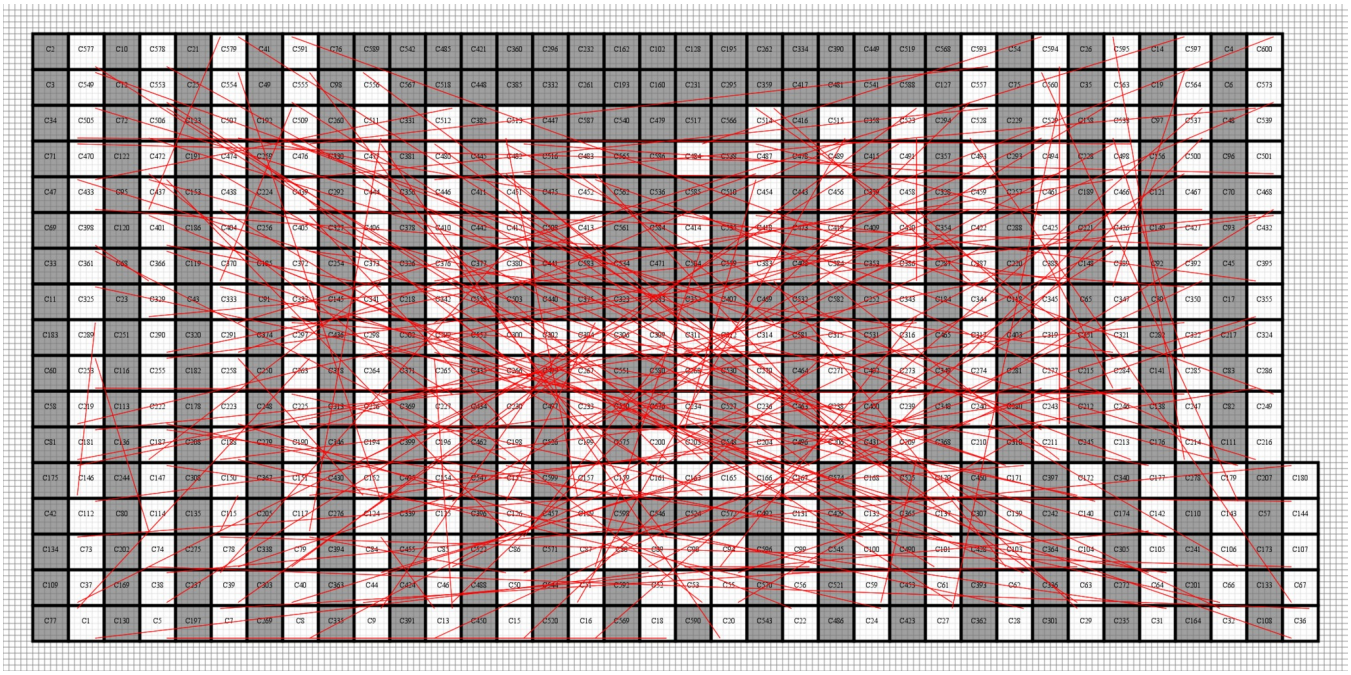

Figure 105: Benchmark 10 - Step 1/7 - Initial placement

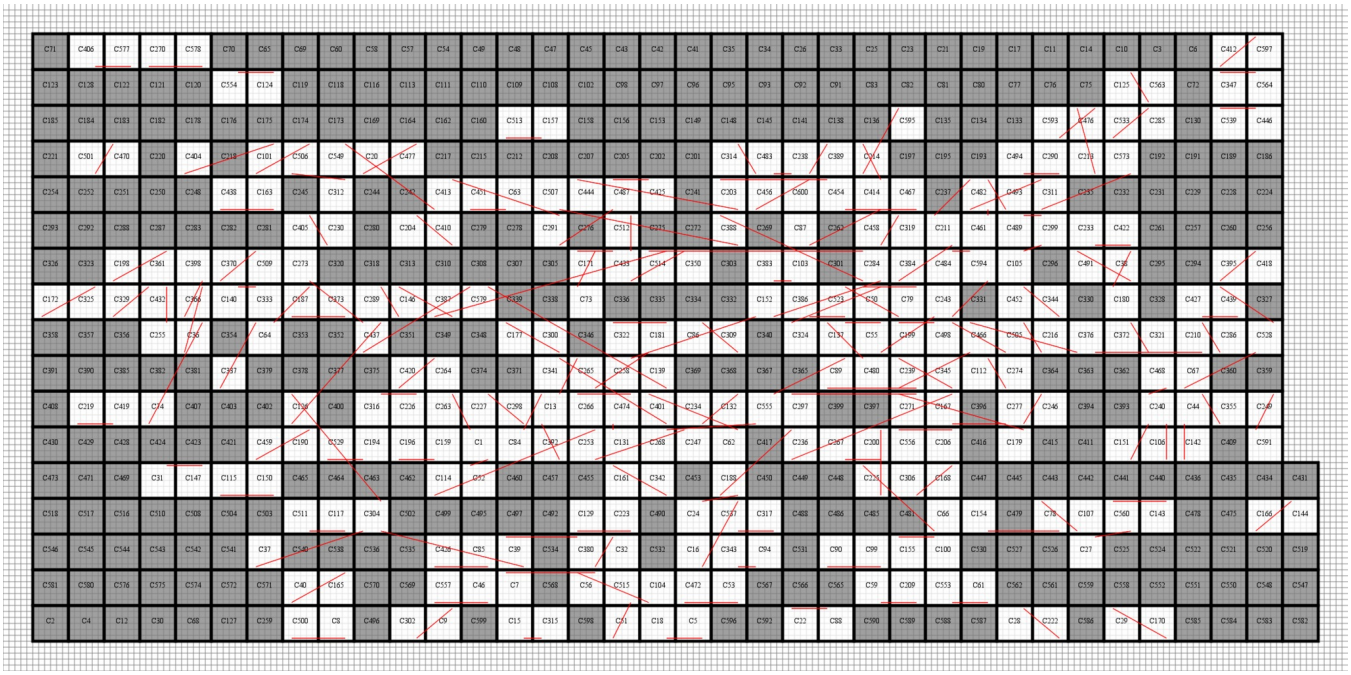

Figure 106: Benchmark 10 - Step 2/7 - Force directed placement

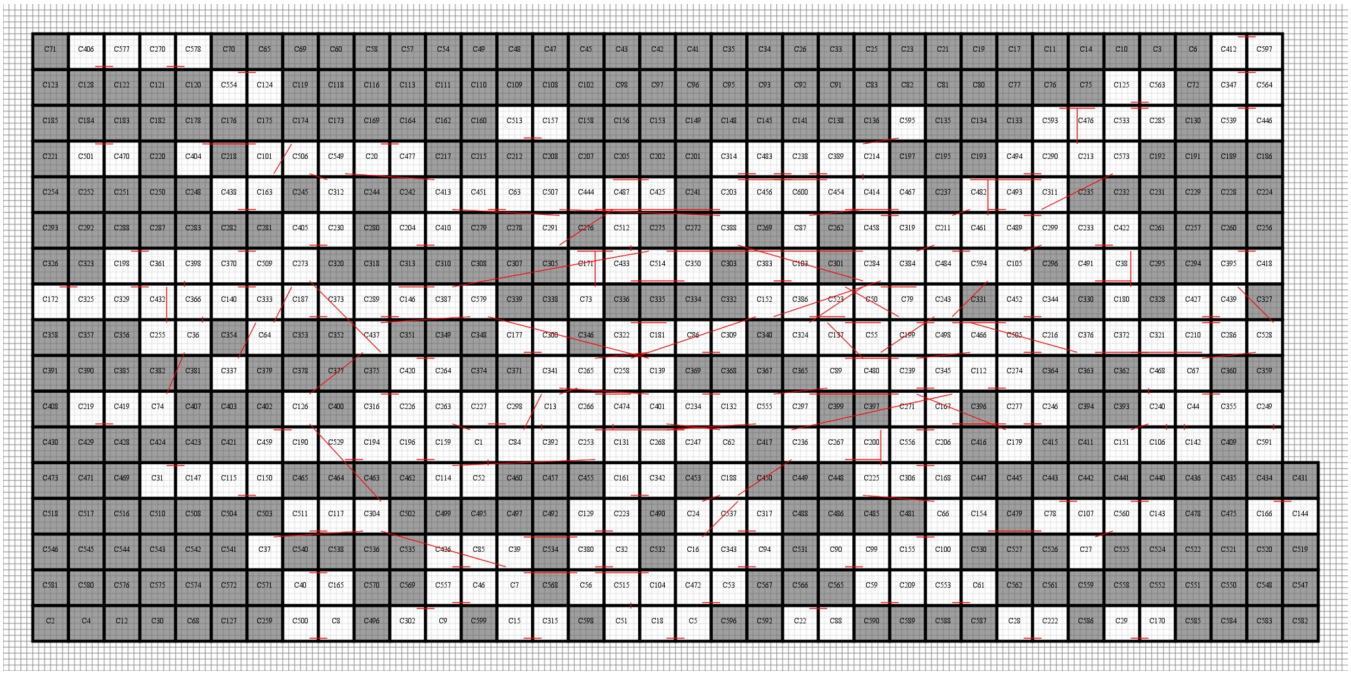

Figure 107: Benchmark 10 - Step 3/7 - Flip cells

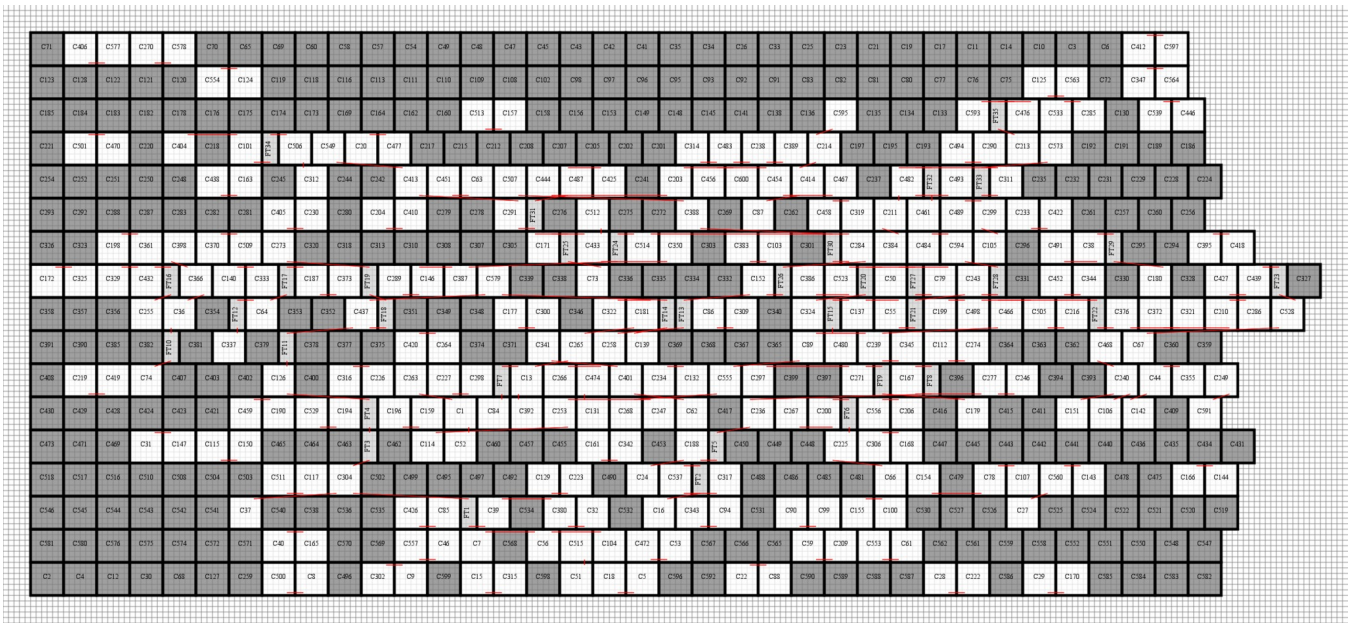

Figure 108: Benchmark 10 - Step 4/7 - Add feed-throughs

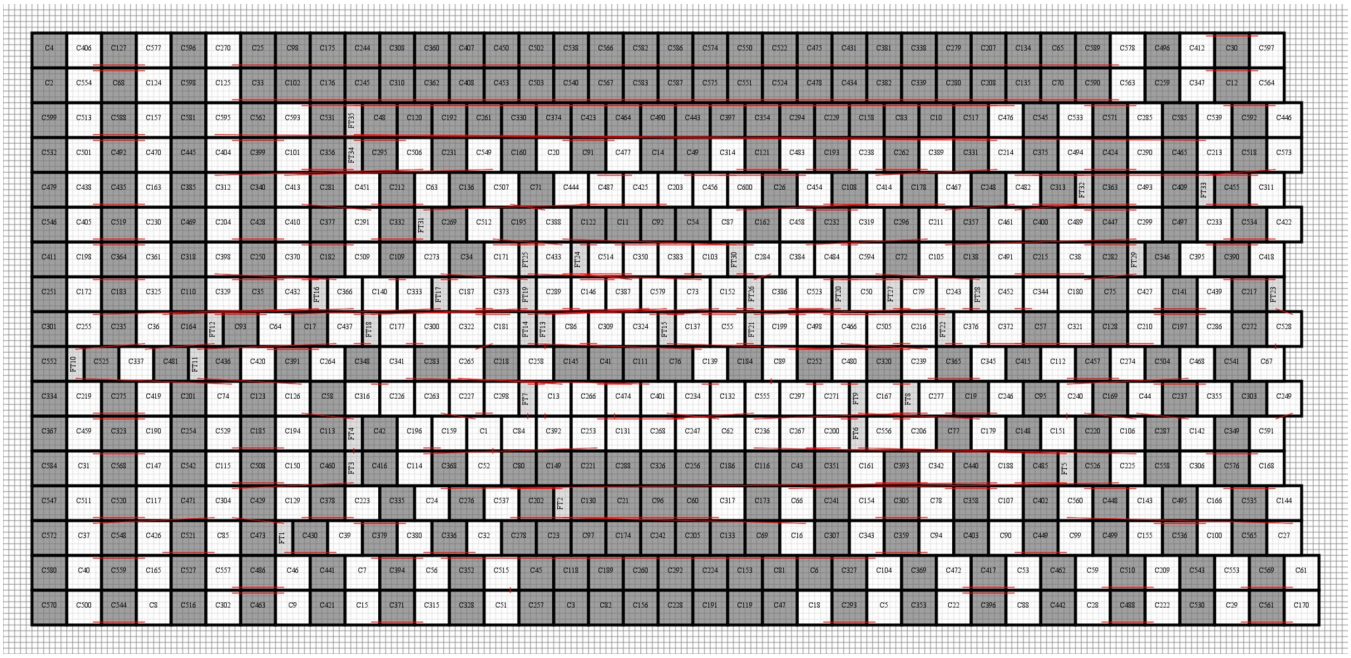

Figure 109: Benchmark 10 - Step 5/7 - Even out the row length

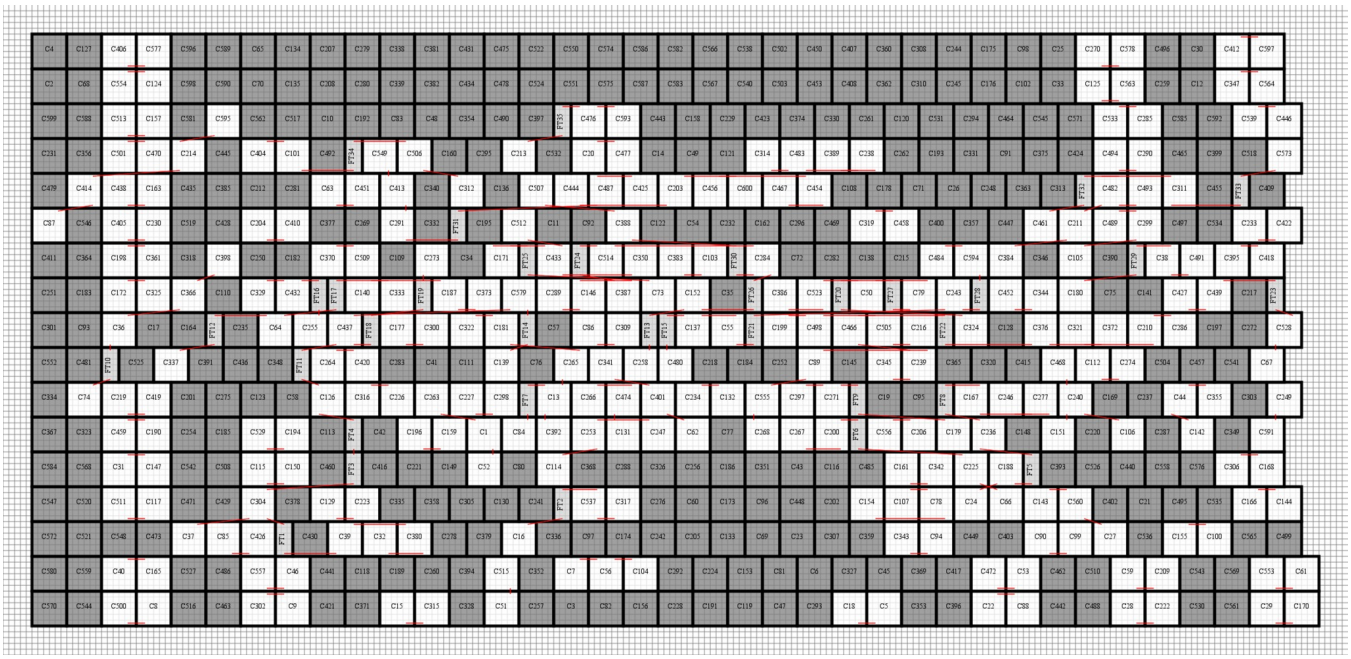

Figure 110: Benchmark 10 - Step 6/7 - Pull cells in the rows closer together

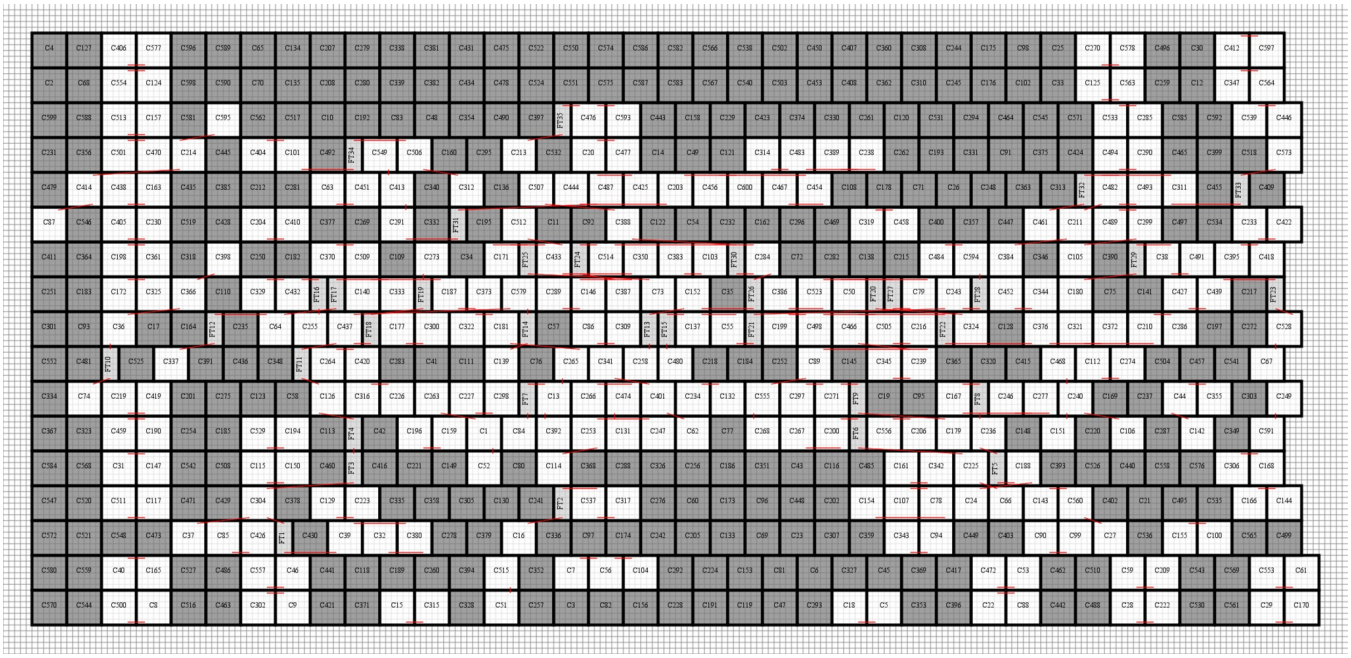

Figure 111: Benchmark 10 - Step 7/7 - Move feed-throughs to optimal location

4.38 Benchmark 10 - Final Routing

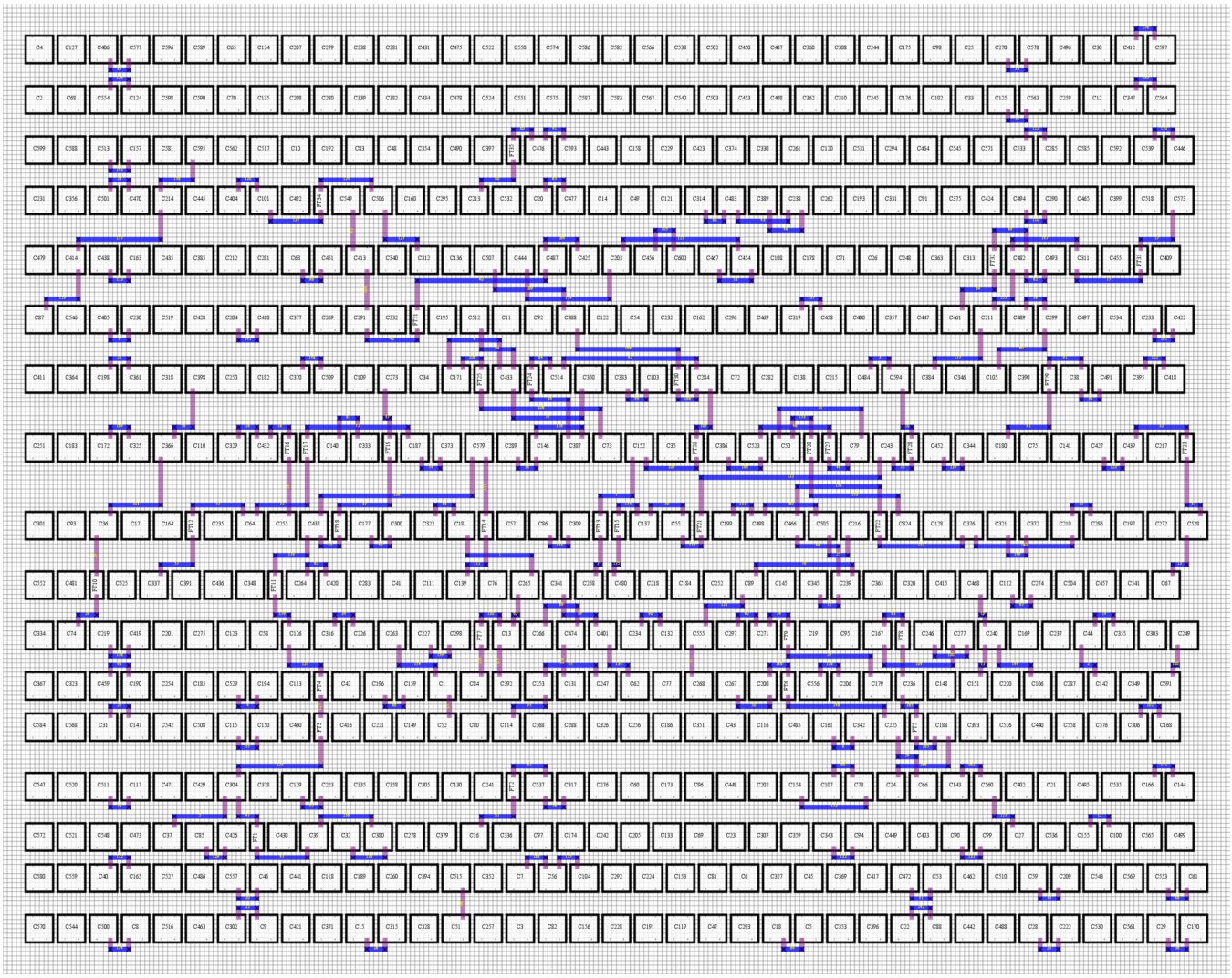

Figure 112: Benchmark 10 - Final routing

4.39 Benchmark 10 - Magic Screenshot

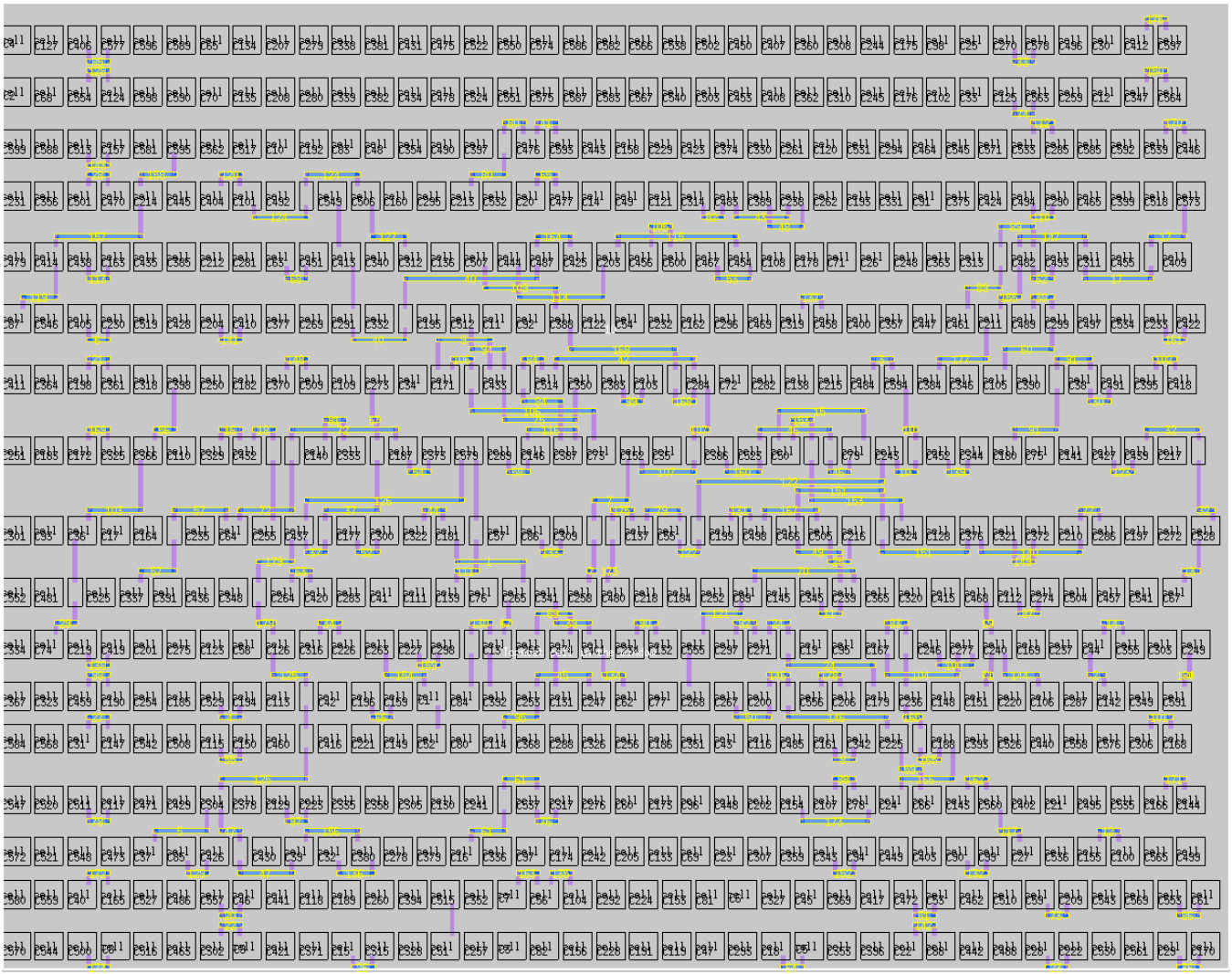

Figure 113: Benchmark 10 - Magic Screenshot

4.40 Benchmark 10 - Log file

```
Number of cell: 600
Number of nets: 180

Placement
---------
Initial grid size: 36 17
Number of feed through cells: 35

Routing
-------
Number of vias: 404
Total number of tracks: 42
Total wirelength: 2881

Size
----
Total width: 258
Total height: 202
Total area: 52116
Squareness (width/height): 1.277228

Time
----
Place time: 0.138171s
Route time: 0.004349s
Total time: 0.159355s

Memory used: 544.000000 kB
```

Listing 10: Benchmark 10 - Log

5 Retrospective

Overall, we are pretty satisfied with both the execution speed and quality of our results. All of our execution times with the exception of benchmark 7 are under half a second. We also feel that our memory usage is minimal as we only store one vector of actual cell objects and all other data structures just contain pointers back to the original objects. We also make heavy use of pass by reference to avoid needlessly copying big data structures into functions.

In terms of placement and routing quality there is always room for improvement. It's easy to visually look at almost any result and see ways that it could be improved but how to translate these improvements into algorithms is not always so obvious. We believe, however, that we were are able to handle a lot of specific cases (like moving the feed-throughs to the other side of the cell) which in the end resulted in superior layouts. For the force directed placement algorithm it is hard to tell if the results you get are truly the best, especially when the circuit is so large you cannot visually comprehend it. We feel like we did our best analyzing it empirically and tuned is to best suit our benchmark files. One of the main problems with the force directed algorithm is that it tends to pull all the cells into the center of the layout. We tried to fix this by redistributing the unconnected cells evenly through the layout and then re-force directing the rows but it is still not always ideal. One interesting method to pursue would be using a combination of min-cut / quadratic placement and then force directing the sub divisions (or vice versa). If we were to min cut horizontally we might be able to reduce some nets that currently span multiple rows.

6 Appendix

Benchmark	Num Cells	Num Nets	Grid Width	Grid Height	Squareness
1	45	45	7	7	0.72449
2	90	90	10	9	0.736486
3	180	180	14	13	0.676724
4	360	360	19	19	0.702857
5	720	720	27	27	0.612457
6	1000	1000	32	32	0.605988
7	500	1000	23	22	0.645203
8	900	720	30	30	0.630648
9	800	360	29	28	0.634349
10	600	180	25	24	0.67148
1	45	45	8	6	0.820225
2	90	90	27	8	0.984496
3	180	180	12	12	0.755981
4	360	360	15	18	0.740413
5	720	720	20	26	0.595577
6	1000	1000	28	31	0.561298
7	500	1000	33	21	0.614935
8	900	720	24	29	0.637652
9	800	360	32	27	0.685294
10	600	180	30	23	0.75
1	45	45	9	5	1.078947
2	90	90	13	7	1.20339
3	180	180	17	11	0.921875
4	360	360	22	17	0.833333
5	720	720	29	25	0.681979
6	1000	1000	34	30	0.583232
7	500	1000	25	20	0.646465
8	900	720	33	28	0.672032
9	800	360	31	26	0.731629
10	600	180	28	22	0.773946
1	45	45	12	4	1.779661
2	90	90	15	6	1.623762
3	180	180	18	10	1.091429
4	360	360	23	16	0.84127
5	720	720	30	24	0.742911
6	1000	1000	35	29	0.624372
7	500	1000	27	19	0.692828
8	900	720	34	27	0.711864
9	800	360	32	25	0.790323
10	600	180	29	21	0.869748
1	45	45	15	3	2.923077
2	90	90	18	5	1.864583
3	180	180	20	9	1.322368
4	360	360	24	15	0.920382
5	720	720	32	23	0.75098
6	1000	1000	36	28	0.699068
7	500	1000	28	18	0.715323
8	900	720	35	26	0.694501
9	800	360	34	24	0.810289
10	600	180	30	20	0.991111
1	45	45	23	2	4.945946
2	90	90	23	4	2.626667
3	180	180	23	8	1.391608
4	360	360	26	14	0.986532
5	720	720	33	22	0.848671
6	1000	1000	38	27	0.674667
7	500	1000	30	17	0.763196
8	900	720	36	25	0.832599
9	800	360	35	23	0.883333
10	600	180	32	19	1.074074

Table 3: Squareness Trial Data