A "GREEDY" CHANNEL ROUTER *

by Ronald L. Rivest and Charles M. Fiduccia

MIT Laboratory for Computer Science, Cambridge, Mass. 02139, and
GE Research and Development Center, Schenectady, New York 12301

March 1981

Abstract

We present a new, "greedy", channel-router that is quick, simple, and highly effective. It *always succeeds*, usually using no more than one track more than required by channel density. (It may be forced in rare cases to make a few connections "off the end" of the channel, in order to succeed.) It assumes that all pins and wiring lie on a common grid, and that vertical wires are on one layer, horizontal on another.

The greedy router wires up the channel in a left-to-right, column-by-column manner, wiring each column completely before starting the next. Within each column the router tries to maximize the utility of the wiring produced, using simple, "greedy" heuristics. It may place a net on more than one track for a few columns, and "collapse" the net to a single track later on, using a vertical jog. It may also use a jog to move a net to a track closer to its pin in some future column. The router may occasionally add a new track to the channel, to avoid "getting stuck".

Introduction

Introduced in 1971 [Ha71], "channel routing" has become a very popular method of routing integrated circuits. (See [KSP73], [Hi74], [De76], [AKT76], [PDS77], [KK79], [Ri82].) Typically, the wiring area is first divided into disjoint rectangular "channels". A "global router" then determines which channels each net traverses. Finally a "channel router" computes a detailed routing for each channel. This approach is effective because it decomposes the overall problem into a number of simpler problems and simultaneously considers all nets traversing each channel.

The general channel-routing problem has been proven NP-Complete ([GJ79], [La80], [Sz81], [SB80]), although algorithms exist for highly-restricted cases ([DKSSU81], [LP81], [Pi81], [To80], [La80]). A slightly different wiring model permits one to come within a factor of 2 of channel density ([RBM81]). Useful methods also exist for computing lower bounds on channel widths ([BR81], [Le81]). These results highlight the need for good practical heuristics.

The algorithm presented here exploits a novel control structure: a left-to-right column-by-column scan of the channel, where the router completes the routing for one column before proceeding to the next. In each column the router acts in a "greedy" manner trying to maximize the utility of the wiring produced.

Our work is an extension of Alford's [Al80]; who also considered a left-to-right scan of the channel. His router did not guarantee success (because it did not allow nets to occupy more than one track in any column), ran quite slowly, and produced noticeably poorer results than our "greedy" algorithm.

Kawamoto and Kajitani [KK79] use a similar column-by-column approach, but not in left-to-right order. They also assume (as we do not) that between adjacent columns there is enough room to wire an arbitrary permutation.

The following paragraphs define what we mean by a "channel routing problem" and its solution.

A channel-routing problem is specified by giving:

(1) A "channel-length" λ. Most of the routing will lie within the channel whose "left end" is at $x = 0$, and "right end" is at $x = \lambda + 1$, on the vertical *columns* at x-coordinates $1, \ldots, \lambda$, although columns outside the channel may also be used.

(2) Top and bottom connection lists $T = (T_1, \ldots, T_\lambda)$ and $B = (B_1, \ldots, B_\lambda)$. T_i (resp. B_i) is the net number for the pin at the top (resp. bottom) of the i-th column (at $x = i$), or is 0 if no such pin exists.

(3) The left and right connection sets, L and R, specifying which nets must connect to the right and left ends of the channel. (They are *sets* since we assume that a net need connect at most once to an end of the channel, and that the relative ordering of such connections may be chosen by the channel router.)

A *solution* to a channel-routing problem specifies:

(1) The channel width w – the number of horizontal "tracks" used. These tracks are at y-coordinates $1, \ldots, w$. A channel router tries to minimize w.

(2) For each net n, a set of connected horizontal and vertical "wire segments" whose endpoints are grid points (x, y) with $1 \leq y \leq w$, except that segments with endpoints $(i, 0)$ or $(i, w + 1)$ must be included if $T_i = n$ or $B_i = n$. Endpoints with $x < 1$ or $x > w$ are legal but should be avoided. A net in L (resp. R) must have a segment touching the line $x = 0$ (resp. $x = \lambda + 1$). Two segments in the same direction are on the same layer, so they may not touch if they are for different nets. Two segments for the same net in different directions that touch at a grid point are said to be connected by a "contact" or "via" at that point. If the segments were for different nets we would have a "crossover".

* This research was supported by the General Electric Corporation, DARPA grant N00014-80-C0622, Air Force grant AFOSR-F49620-81-0054, and NSF grant MCS-8006938.

0420-0098/82/0000/0418$00.75 © 1982 IEEE

The channel density of a particular channel routing problem is defined to be the maximum number of nets which have pins on both sides of the line $x = \alpha$, for any α. (We don't count nets all of whose pins lie on a single vertical line.) The channel density is a lower bound on the width of any solution to that channel-routing problem.

If its "conflict graph" ([HS71]) contains cycles, a channel-routing problem may be unsolvable within the channel, for any w (e.g. $\lambda = 2$, $T = (1, 2)$ and $B = (2, 1)$.) Such problems can always be solved by using columns "outside" the channel.

The following factors are often used to evaluate the quality of a successful solution (in a typical order of priority): its width w, the number of columns "off the end" it uses, its total wire-length, and the number of vias it uses.

The Routing Algorithm

The greedy router scans the channel in a left-to-right, column-by-column manner, completing the wiring within a given column before proceeding to the next. In each column the router tries to maximize the utility of the wiring produced, in a simple "greedy" manner.

Its first step in a column is to make connections to any pins at the top and bottom of the column. These connections are *minimal*; no more vertical wiring is used than is needed to bring these nets safely into the channel, to the first track which is either empty or contains the desired net.

The second step in a column tries to free up as many tracks as possible by making vertical connecting jogs that "collapse" nets that currently occupy more than one track. This step may complete the job of bringing a connection from a pin over to a track that its net currently occupies (step 1 might have stopped at an intermediate empty track).

The third step tries to shrink the range of tracks occupied by nets still occupying more than one track, so collapsing these nets later will be less of a problem. Since freeing up tracks has high priority, jogs made here have priority over jogs made in the next step.

The fourth step makes "preference" jogs that move a net up if its next pin is on the top of the channel, and down if its next pin is on the bottom. The router chooses longer jogs over shorter ones if there is a conflict. This tends to maximize the amount of "useful" vertical wiring created. These jogs are effective at resolving upcoming "conflicts", even though no explicit consideration of these conflicts is made.

The fifth step is only needed if a pin could not be connected up in step one because the channel is "full". Then the router "adds a new track" to the channel between existing tracks, and connects the pin up to this track. (The old tracks are renumbered.)

When the processing for a column is complete, the router extends the wiring into the next column and repeats the same procedure. The following paragraphs make precise the algorithm just sketched.

The input for the greedy router consists of (1) a specification of a channel-routing problem, (2) three non-negative integer parameters: *initial-channel-width*, *minimum-jog-length*, and *steady-net-constant*.

The greedy router begins with the *initial-channel-width* given. A new track is added whenever the current channel-width becomes unworkable. The router does *not* begin over when a new track is added, so different initial widths may give different results. Good results are usually obtained with *initial-channel-width* just less than the best final channel width. One can run the router several times, with *initial-channel-width* set initially to the channel density and increased by one each time.

The router will make no "jogs" shorter than *minimum-jog-length*. A higher setting reduces the number of vias and thus produces more acceptable solutions, while a lower setting tends to reduce the number of tracks used. The best results are obtained with a setting of about $w/4$, where w is the best channel width obtainable. By running the router 2-4 times with different initial parameter settings we quickly determined the best solution obtainable.

Let $H(n)$ denote the highest column k for which $T_k = n$ or $B_k = n$ (except that $H(n) = \lambda + 1$ if $n \in R$). We say a net n "has its last pin in column k" if $H(n) = k$ and that it "has its last pin by column k" if $H(n) \leq k$.

When routing a given column, the greedy router classifies each net which has a pin to the right as either *rising*, *falling*, or *steady*. A net is *rising* if its next pin after the current column will be on the top of the channel (say in column k), and the net has no pin on the bottom of the channel before column $k +$ *steady-net-constant*. *Falling* nets are defined similarly. *Steady* nets are the remaining nets. We typically use a value of 10 for *steady-net-constant*. A larger value reduces the number of times a multi-pin net changes tracks.

The fundamental data structure for this router is the set $Y(n)$ for each net n of "tracks currently occupied" by net n. Each track is denoted by its y-coordinate, so $Y(n)$ is a subset of $\{1, \ldots, w\}$ for each n. If $Y(n) = \phi$ (the empty set), the net is not currently being routed (i.e. we have not yet reached the first column in which net n has a pin, or we have passed the last column in which net n has a pin and have completed all the routing for net n). Otherwise, suppose $Y(n) = \{y_1, \ldots, y_k\}$ when the router is working on column i. Then each point $(i, y_1), \ldots, (i, y_k)$ is a "dangling end" of some wiring already placed for net n. Exactly one such "dangling end" is listed in $Y(n)$ for each connected piece of wiring already placed for net n. The router is obligated to eventually connect together these "dangling ends" so that each net is finally implemented by a single connected piece of wire. When extending the routing from column i to column $i + 1$, horizontal wiring will used in every track y for which $y \in Y(n)$ for some n and either $|Y(n)| > 1$ (the dangling ends have yet to be connected together) or the last pin for net n occurs after column i.

We define a net to be *split* at any time that $|Y(n)| > 1$. We also call a split net "collapsible", since we may be

able to "collapse" it down to a single track (or zero tracks if we have passed the last pin for the net) by making an appropriate connecting jog.

We illustrate the operations of the router using a set of "before-after" figures for each step. These figures describe what happens in a single column, and should be interpreted as follows. Nets entering a column from the previous column are shown extended up to the current column. If the net has pins to the right of this column, the net is shown extended towards the next column with an arrowhead. Otherwise (if the net has no pins to the right), no arrowhead is shown. A "+", "—", or "+/ —" may be shown next to an arrowhead to denote rising, falling, or steady nets.

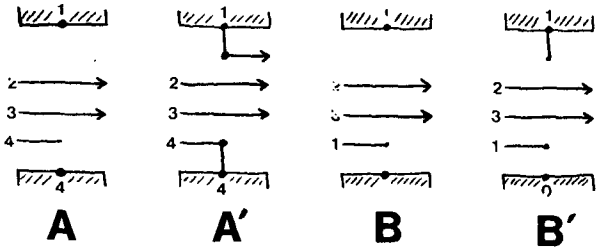
The Greedy Router

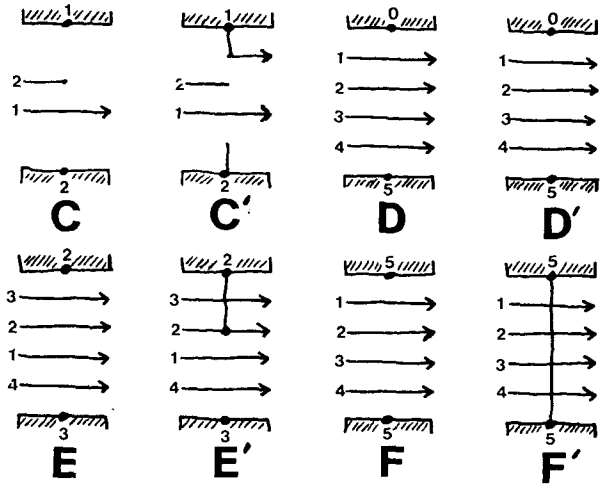
Let w denote the current channel width (initially $w =$ *initial-channel-width*).

Assign Tracks To Nets At Left End: For each net n in L give n a distinct value for $Y(n)$ (i.e. a distinct track in the range $1, \ldots,$ *initial-channel-width* on which to enter the channel from the left end.) Put the "rising" nets above the "steady" nets above the "falling" nets and generally group the nets at the center of the channel.

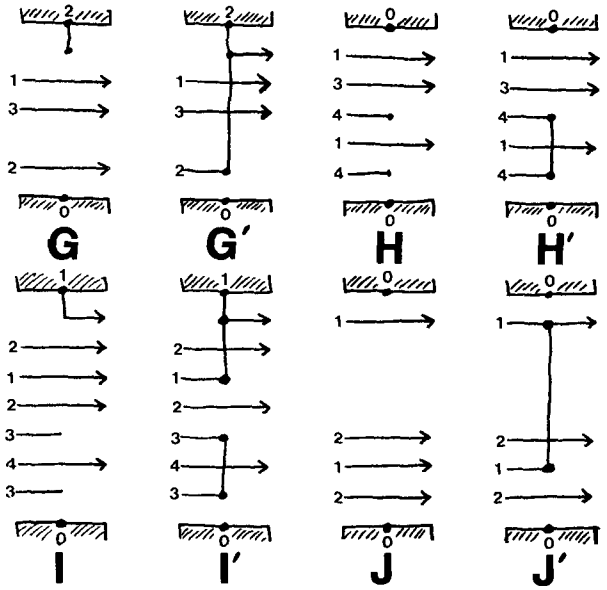
Route Channel From Left To Right: For each column i, for $i = 1, 2, \ldots,$ until $i \geq n$ and no split nets remain to be collapsed **do:**

(a) **Make Feasible Top and Bottom Connections in Minimal Manner:** If T_i or B_i is nonzero, "bring in" that net if possible to the to the *nearest possible* track which is either empty or already assigned to this net, by running a vertical wire from the edge of the channel to the desired track, and adding that track to $Y(T_i)$ or $Y(B_i)$. (Fig. A) Note that a net n is *not* routed to the nearest track in $Y(n)$ if there is a nearer empty track – leaving n temporarily assigned to an additional track. (Figs. B, C) Also note that a new net can not be brought into a "full" channel in this step (but see step (c)). (Fig. D) If T_i and B_i are both nonzero, try to "bring in" both nets but if $T_i \neq B_i$ and the vertical segments would conflict (overlap) then just bring in the net which can be brought in with the least wire, and leave the other net to be brought in at step (c). (Fig. E) As special case, if there are no empty tracks, and net $T_i = B_i \neq 0$ is a net which has connections in this column only, then run a vertical wire from top to bottom of this column. (Fig. F)





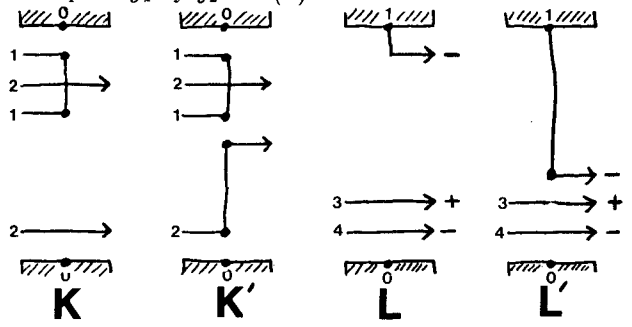
(b) **Free Up As Many Tracks As Possible By Collapsing Split Nets:** Add vertical segments in this column to collapse split nets in a pattern that will create the most empty tracks for use in the next column. Define a "collapsing jog" to be a "piece of vertical wire" which connects two tracks holding the same net without crossing another track holding that net. (So each split net n generates $|Y(n)| - 1$ such jogs.) Define a "pattern" to be any set of collapsing jogs for which jogs for different nets do not overlap and for which no jogs overlap any vertical wiring placed in step (a). The number of such patterns to consider may be exponential in the number of collapsing jogs there are to consider. Find the pattern which creates the most empty tracks by a small but complete combinatorial search. (Figs. G, H) A pattern will free up one track for every jog it contains, plus one additional track for every net it "finishes". (The pattern finishes a net n if it totally connects up the dangling ends for n and n has its last pin by column i.) Resolve any ties between patterns that free up the most tracks by choosing the pattern which leaves the



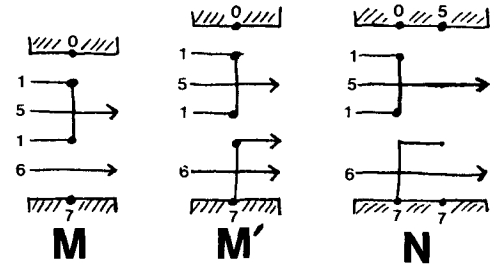
outermost uncollapsed split net as far as possible from the channel edge; if necessary consider the second outermost such net, etc. (Fig. I) Resolve any remaining ties by choosing the pattern with largest sum of jog lengths. (Fig. J) Add appropriate vertical wiring for each jog in the winning pattern, and for each such jog which connects a track y_1 to a track y_2 (assume $y_1 < y_2$) for some net n, delete y_1 from $Y(n)$. (This is an arbitrary choice that might get modified in steps (c) and (d).) Note that this step will typically collapse a net that was temporarily brought in to an empty track in step (a) when that net had a previously assigned but more distant track.

(c) **Add Jogs To Reduce The Range of Split Nets:** For each uncollapsed split net (i.e. for each net n with $Y(n) \geq 2$), try to reduce the range of tracks assigned to the net by adding vertical jogs that have the effect of moving the net: (i) from the maximum track in $Y(n)$ to the lowest possible empty track and (ii) from the minimum track in $Y(n)$ to the highest possible empty track. (Fig. K) Because of step (b), no collapsing will occur, but the difficulty of collapsing the the remaining split nets may be reduced. Make no jogs which are shorter than *minimum-jog-length* or which would be incompatible with vertical wiring already placed in this column by previous steps. If a jog for net n is made from track y_1 to track y_2, replace y_1 by y_2 in $Y(n)$.

(d) **Add Jogs to Raise Rising Nets and Lower Falling Nets:** Consider all the unsplit (i.e. $|Y(n)| = 1$) rising and falling nets being routed in order of *decreasing* distance from their track $y \in Y(n)$ to their "target edge" (e.g. the upper edge of the channel for rising nets). Try to add a vertical jog to move that net to an empty track which is as close as possible to its target edge. (Fig. L) Make no jogs which are shorter than *minimum-jog-length* or which would be incompatible with vertical wiring already placed in this column by previous steps. If a jog for net n is made from track y_1 to track y_2, replace y_1 by y_2 in $Y(n)$.



(e) **Widen Channel If Needed To Make Previously Infeasible Top Or Bottom Connection:** If a net T_i or B_i could not be brought in to a track in step (a), create a new track for this net and bring the net in to this track. Place this track as near the center of the channel as possible between existing tracks, subject only to the constraint that desired connection to the edge of the channel can be made. (Fig. M) (If the new track



lies between tracks previously numbered k and $k+1$, all old tracks at y-coordinates $k+1$ and greater now have their y-coordinates retroactively increased by one, and all $Y(n)$ referring to these tracks are appropriately modified.) Add the new track to $Y(T_i)$ or $Y(B_i)$ as appropriate.

(f) **Extend To Next Column.** For each net n such that $|Y(n)| = 1$ and n has no pins after column i, make $Y(n)$ be the empty set. (The routing for these nets is now finished.) Then for each track y which is in $Y(n)$ for some n, extend the "dangling end" for net n along track y into column $y+1$ with appropriate horizontal wiring. (Fig. N)

This completes the description of the greedy router. The router will always complete the routing successfully, although it may use a few additional columns beyond the natural right end of the channel to do so.

The algorithm takes about 10 seconds on a DEC KA-10 for moderate sized channels. The implementation was simple -- about 15 pages of LISP code, counting 10 pages for I/O and initialization.

Discussion

This algorithm is the result of long series of experimentation and evaluation of variations on the basic idea of scanning down the channel from left to right and routing everything as you go.

By "minimally" connecting a net in step (a) we separate the tasks of connecting up a pin and of deciding to use a column to jog all the way over to a track the net may already be on. Step (b) makes this latter decision; it might turn out that another such "collapsing pattern" frees up more tracks.

When collapsing nets the router tries to free up the most tracks, since it is hard to achieve optimal routings if nets are allowed to occupy more than one track for very long. Since we observed that it is very difficult in general to collapse a net which is in a track just next to the channel edge, due to the fact that other nets must cross this track to enter the channel, the collapsing algorithm will favor patterns that collapse these "difficult" nets.

The use of combinatorial search for the net collapsing phase was found to be acceptably fast since there were never more than four split nets in our examples. A "dynamic programming" approach can be used instead, if it is desired to avoid exponential worst-case running times.

We were surprised to find that the step (d) works so well, since it is very simple and takes no particular notice of

upcoming conflicts. Our initial implementation tried to first resolve upcoming conflicts in the order they were coming up, and then to jog the other nets as much as possible in the appropriate directions. The success of the current variation seems to be based on the fact that it tries to jog nets in tracks near the edges of the channel first – these are the most difficult places to move a net from, and also on the fact that the router will tends to maximize the amount of useful vertical wiring created.

One nice feature of the greedy router is that its control structure is very flexible and robust: it is easy to make variations in the heuristics employed to achieve special effects or to rearrange priorities. The particular algorithm presented here is merely our best suggestion based on our experimental evidence; other variations may turn out better in other situations. As an example, we have also considered a "gridless" variation where the track-to-track spacing can be reduced if a pair of adjacent tracks does not have contacts next to each other in some column. This variation uses more "intelligence" when selecting the jogs to make in a given column. It is also easy, for example, to restrict jogs for a net to those columns for which it has a top or bottom connection, etc. Another variation we have not yet tried is to scan outwards from a column of maximum density instead of using a left-to-right scan; we expect this variation may prove to be valuable in practice as well. We not very sure how one should best order the nets in L at the beginning – how should a set of rising nets be ordered?

One extension that is worth noting in particular is that is not too difficult to modify the router to handle the notorious "switchbox" problem – where a "channel" has a fixed length and width and terminals fixed on all four sides. (See [So81] for a discussion of the importance of this problem.) Two MIT students, Jim Koschella and David Christman, have performed this modification; their results are reported in Koschella's B.S. thesis, and their program is currently used in the MIT "PI" system. ([Ko81, Ri81b])

Experimental Results

We present three sorts of experimental results:

(a) Data on five chips routed at GE using previous algorithms,
(b) Data on Deutsch's "difficult example", and
(c) Data on program-generated standardized test examples taken from [Ri81a]

We considered five chips at GE that were designed using a poly-cell approach. All together they contained 26 channels, with an average channel density of 16.500, and a range of densities from 12 to 43. The greedy algorithm was able to route all of these channels successfully, with an average channel width of 16.654 (i.e. it routed 22 of the 26 channels using a number of tracks exactly equal to the channel density, and routed 4 of them using one more track than the channel density). This represents an average of an increase of 0.93 percent over channel density. The previous router used at GE averaged an increase of roughly 7.2 percent over channel density for these problems.

We tested the greedy router on the "difficult example" of Deutsch [De76]. This problem has a channel density of 19. To our knowledge no completely automatic algorithm has produced a routing in 19 tracks. Yoshimura and Kuh report an algorithm which achieved 20 tracks on this problem [YK80]. The greedy algorithm also produced a routing in only 20 tracks (although it did use more vias). This routing is given in the Appendix.

The paper [Ri81a] contains a "standard" set of benchmark channel-routing problems, described by a program that generates them. We ran the greedy router on many benchmark channels taken from this paper, and were generally unable to improve by hand any of the routings found.

Acknowledgements

We should like to thank Dan Schweikert, David Deutsch, Ernest Kuh, Phil Lewis, James Rumbaugh, Flavio Rose, Ron Pinter, Charles Leiserson, Alan Baratz, and Seth Alford for helpful discussions, information, and/or critical comments on an early draft of this paper.

REFERENCES

[Al80] Alford, S. "DYCHAR: A Channel Router which uses dynamic channel assignment," MIT Bachelor's thesis. (May 1980).

[AKT76] Asano, T., T. Kitahashi, and K. Tanaka, "On a Method of Realizing Minimum-Width Wiring," *Electronics and Communications in Japan*, Vol. J59-A, No. 2 (1976), 29-39.

[BR81] Brown, D. and R. L. Rivest, "New Lower Bounds on Channel Width", *Proc. CMU Conference on VLSI Systems and Computations*, (Computer Science Press 1981) 178-185.

[De76] Deutsch, D. N., "A 'Dogleg' Channel Router," *Proc. 13-th Design Automation Conference*, IEEE (1976), 425-433.

[DKSSU81] Dolev, D., K. Karplus, A. Siegel, A. Strong, and J. Ullman, "Optimal Wiring Between Rectangles," Proc. 13th Annual ACM Symposium on Theory of Computing (May 1981), 312-317.

[GJ79] Garey, M. and D. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, (Freeman, 1979).

[Ha71] Hashimoto, A. and J. Stevens, "Wire Routing by Optimizing Channel Assignment within Large Apertures," *Proc. 8-th Design Automation Workshop*, IEEE (1971), 214-224.

[Hi74] Hightower, D., "The Interconnection Problem: A Tutorial," *Computer* 7,4 (April 1974), 18-32.

[KK79] Kawamoto, T. and Y. Kajitani, "The Minimum Width Routing of of a 2-Row 2-Layer Polycell-Layout," *Proc. 16th Design Automation Conference*, IEEE (1979), 290-296.

[KSP73] Kernighan, B., D. Schweikert, and G. Persky. "An Optimal Channel-Routing Algorithm for Polycell Layouts of Integrated Circuits," *Proc. 10-th Design Automation Workshop*, IEEE (1973), 50-59.

[Ko81] Koschella, J., "A Placement/Interconnect Channel Router: Cutting your PI into Slices," Bachelor's Thesis. MIT Department of Electrical Engineering and Computer Science. (May 1981).

[La80] LaPaugh, A., "Algorithms for Integrated Circuit Layout: an Analytic Approach," Ph.D. Thesis, MIT Laboratory for Computer Science Report TR-248 (November 1980).

[Le81] Leighton, T., "New Lower Bounds for Channel Routing," To appear.

[LP81] Leiserson, C., and R. Pinter, "Optimal Placement for River Routing," *Proc. CMU Conference on VLSI Systems and Computations*, (Computer Science Press 1981), 126-143.

[PDS77] Persky, G., D. Deutsch, and D. Schweikert, "LTX – A Minicomputer-Based System for Automated LSI Layout," *Journal of Design Automation and Fault-Tolerant Computing* 1,3 (May 1977), 217-255.

[Pi81] Pinter, R., "Optimal Routing in Rectilinear Channels," *Proc. CMU Conference on VLSI Systems and Computations*, (Computer Science Press 1981), 160-177.

[Ri81a] Rivest, R. L., " 'Benchmark' Channel-Routing Problems," In preparation.

[Ri82] Rivest, R. L., "The 'PI' (Placement and Interconnect) System". *Proc. 19th Design Automation Conference*, (Las Vegas, 1982)

[RBM81] Rivest, R. L., A. E. Baratz, and G. Miller, "Provably Good Channel-Routing Algorithms," *Proc. CMU Conference on VLSI Systems and Computations*, (Computer Science Press 1981), 153-159.

[SB80] Sahni, S. and A. Bhatt, "The Complexity of Design Automation Problems," *Proc. 17th Design Automation Conference*, IEEE (June 1980), 402-411.

[So81] Soukup, J., "Circuit Layout," Proc. of the IEEE, Vol. 69, No. 10(Oct. 1981), 1281-1304.

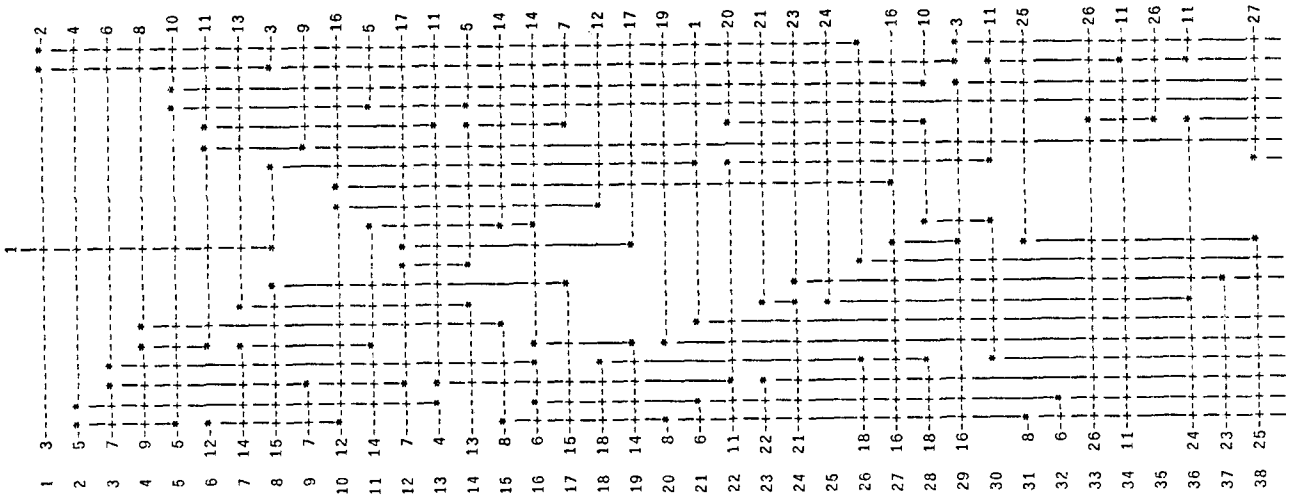
[Sz81] Szymanski, T., "Dogleg Channel Routing is NP-Complete," To appear.

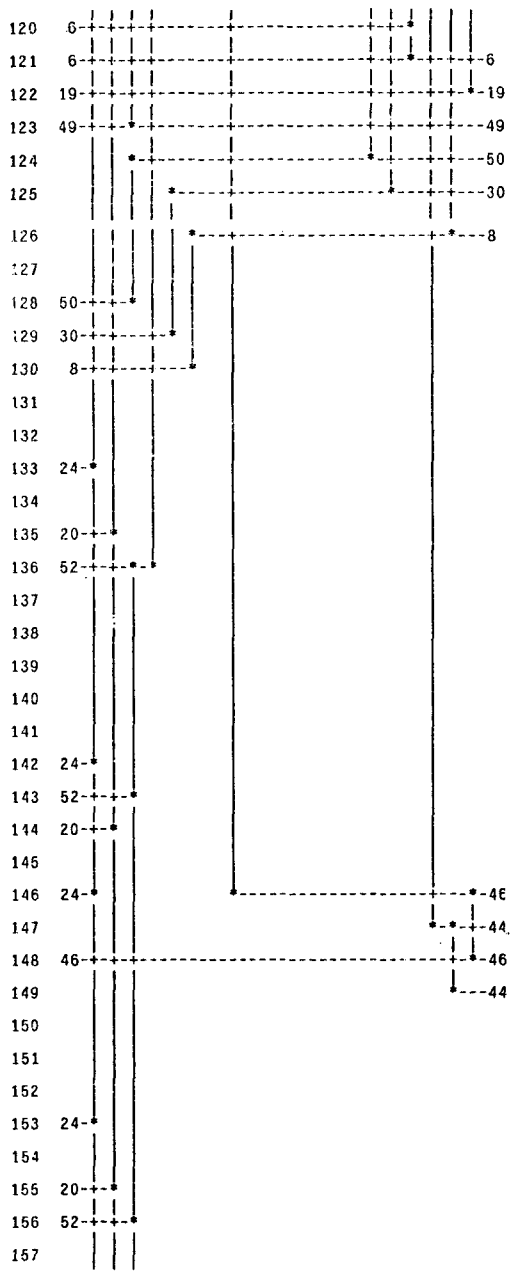
[To80] Tompa, M., "An Optimal Solution to a Wire-Routing Problem," *Proc. 12th Annual ACM Symposium on Theory of Computing*, (April 1980), 201-210.

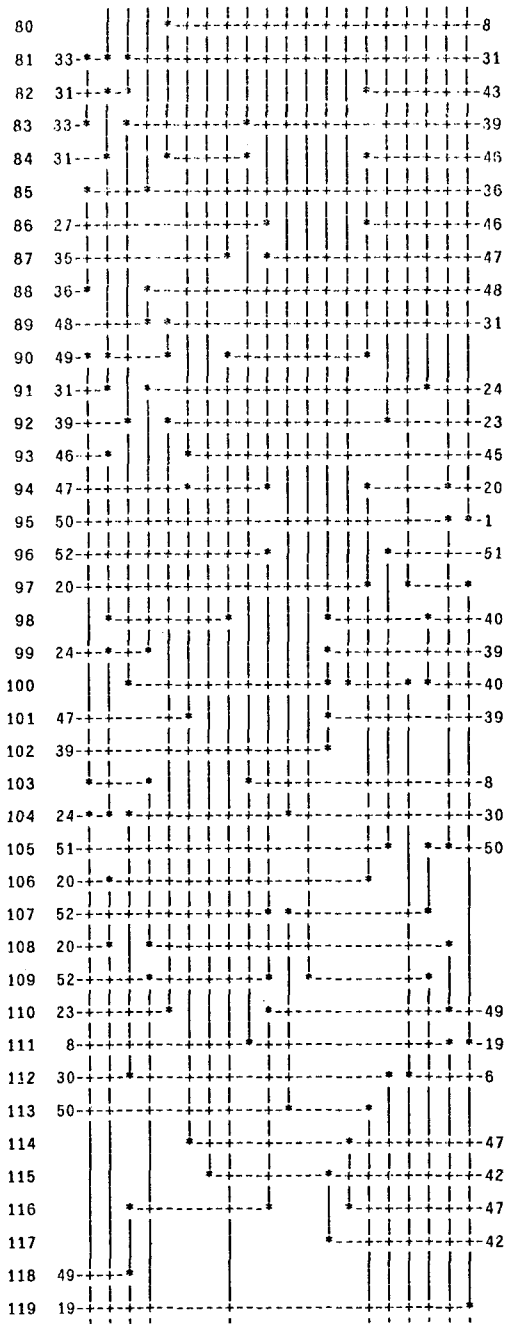
[YK80] Yoshimura, T. and E. Kuh, "Efficient Algorithms for Channel Routing," U. C. Berkeley Electronics Research Laboratory Memo. No. M80/43 (August 1980).

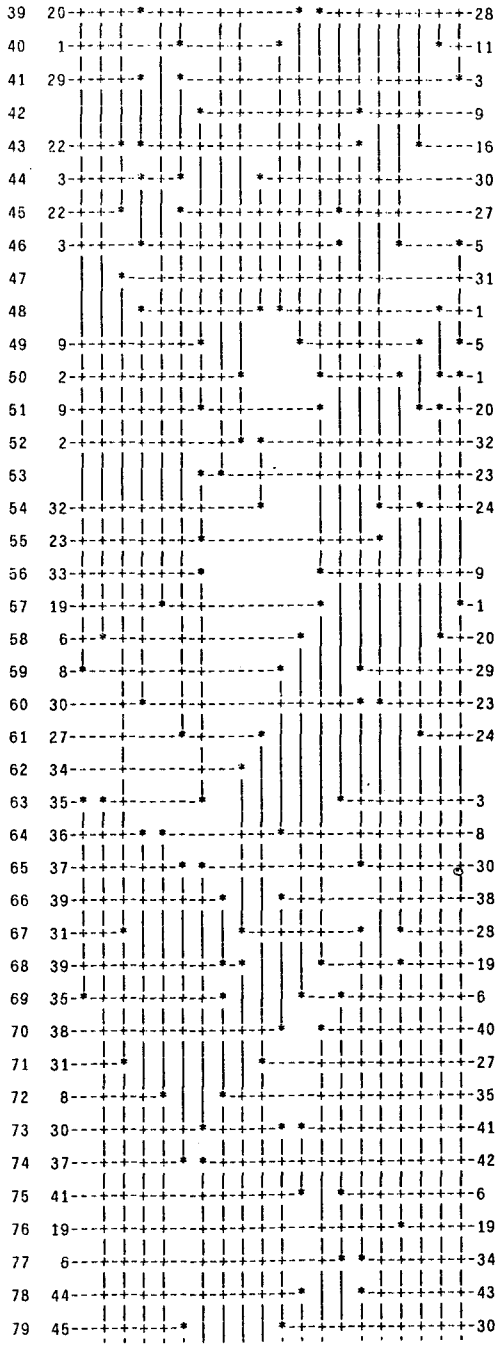
I. Appendix. Deutsch's "Difficult Example"

Parameters:	Initial-channel-width	=	20
	Minimum-jog-length	=	2
	Steady-net-constant	=	10
Results:	Channel-width	=	20 (Density = 19)
	Extra columns used	=	0
	Vias used	=	347
	Wire-length	=	4150
	Time	=	7.93 seconds




120 6
121 6 6
122 19 19
123 49 49
124 50
125 30
126 8
127
128 50
129 30
130 8
131
132
133 24
134
135 20
136 52
137
138
139
140
141
142 24
143 52
144 20
145
146 24 46
147 44
148 46 46
149 44
150
151
152
153 24
154
155 20
156 52
157


80 8
81 33 31
82 31 43
83 33 39
84 31 46
85 36
86 27 46
87 35 47
88 36 48
89 48 31
90 49
91 31 24
92 39 23
93 46 45
94 47 20
95 50 1
96 52 51
97 20
98 40
99 24 39
100 40
101 47 39
102 39
103 8
104 24 30
105 51 50
106 20
107 52
108 20
109 52
110 23 49
111 8 19
112 30 6
113 50
114 47
115 42
116 47
117 42
118 49
119 19


39 20 28
40 1 11
41 29 3
42 9
43 22 16
44 3 30
45 22 27
46 3 5
47 31
48 1
49 9 5
50 2 1
51 9 20
52 2 32
53 23
54 32 24
55 23
56 33 9
57 19 1
58 6 20
59 8 29
60 30 23
61 27 24
62 34
63 35 3
64 36 8
65 37 30
66 39 38
67 31 28
68 39 19
69 35 6
70 38 40
71 31 27
72 8 35
73 30 41
74 37 42
75 41 6
76 19 19
77 6 34
78 44 43
79 45 30