Congestion Router for Schematic Diagrams

Stephen T. Frezza & Steven P. Levitan

Department of Electrical Engineering
University of Pittsburgh
Pittsburgh, PA 15261
TR-CE-94-03

Abstract

This paper presents a new approach to routing schematic diagrams. Heuristic global and local algorithms are presented which focus on congestion and crossover issues in the development of aesthetically-pleasing diagrams. Congestion is the basis for the global, and crossover for the local router. This algorithm can work on a completed schematic, but also can be applied to schematics that are incrementally developed.

1 Introduction

Schematics are a widespread method of creating and documenting designs. With the increasing use of hardware description languages, there is a need to automatically-generate diagrams that represent these graphical representations of the design. The manual construction of a good schematic demands special skills, and is an error-prone process[SH90], due to the difficulties in producing a schematic that is useful to the designer. In this same manner, the automatic generation of schematic diagrams has proven to be difficult.

One of the uses of schematics as documentation is to trace out the signals in the design. In a well-constructed schematic, this should be a relatively easy task. However, the construction of a traceable schematic is neither easy to define, nor quickly implemented. In fact, one can characterize the success of a schematic generator by a measure of the traceability of the routes in the diagram created.

There are many *ad hoc* rules that can be used to characterize a good set of schematic routes. The most agreed upon rule is that the number of crossovers in the design should be minimized. While other schematic routers aim at reducing or minimizing crossovers, turns, etc. [Bre75, MMM83, SK89, BJS$^+$89, SH90], our view is that good routes are not made simply by minimizing the number of crossovers, but rather by controlling the *location* of the corners and crossovers, hence reducing the perceived congestion of the final schematic.

This paper presents the algorithms for a special-purpose schematics router developed as a part of an Automatic Schematic Generation (ASG) system, SPAR.[Fre91, FL92] We present the features that distinguish schematics routing from the general routing problem, and the previous work for routing in ASG systems. We then discuss how our global and local routing strategies address these issues. We then go on to describe the details of the global and local routing algorithms employed. We conclude with several examples, and a discussion of future work.

2 Background

Schematic routing differs from the general routing problem in many ways: In the general case the length of the route is a primary factor but is only of secondary importance in schematics. Minimizing the area used by the route is desirable in the general case, but is generally counter-productive for producing traceable schematics. Similarly, general-purpose routers do not consider the number of segments used, whereas the schematic router is interested in using as few segments as possible. The number and placement of corners is unimportant for general routers, but is very important for a schematic. Similarly, the existence of crossovers in a two-layer router is unimportant, but critical for a traceable schematic.

It is these difficulties that make the schematics routing problem sufficiently different from other routing problems to make much of the developed routing techniques inappropriate, hence the need for special-purpose routers such as the one described here.

Most schematic generation systems focus on developing good placements, making provisions in the placer to simplify the job of schematics router. This can be done in two ways: either completing the route incrementally *as* the schematic is developed, or by forming a placement that can be expected to result in a good routing solution. These techniques aim at dealing with the interdependence of placement and routing in the development of aesthetically-pleasing schematics diagrams.

The schematics routers in the literature tend to fall into one of three categories:

[1] **Modified standard-routing techniques**:

- Channel routers used in [MMM83, May85] and in AUTODRAFT [MMA86].
- Modified channel router, modified for incremental used in [AAM85].
- Modified line-expansion routers in Gems[VW86] and Eureka[SK89, KS89].

[2] **Expert-system techniques**:

- HALS [AHS83], CHITRALEKHA [BJS$^+$89] and the work presented in [SH90].

[3] **Special-purpose routers**:

- Segment/crossover minimization technique described in [Bre75].
- Congestion router described here.

These schematic routers have primarily focused on minimizing the segment count and solving the crossover problem, and tend to model the problem with a set of simple rules. Only the expert-system routers and the special-purpose routers have been designed to focus on other traceability issues more explicitly. Expert-system approaches have focused on developing a knowledge-base of rules that model what human designers do to recognize and solve routing problems[SH90, AHS83]. The earlier special-purpose algorithm focused on creating sets of heuristics for forming columns and rows of routes that resemble hand-drafted results[Bre75], and did not address the congestion of routes. We see traceability as the key figure of merit for schematics routers, and congestion as critical to finding the most traceable solution. We now discuss the relationship between traceability and congestion.

2.1 Traceability and Congestion

The proximity of nets, or rather the proximity of the features of a net, hinder the traceability of that net. This is an area phenomenon; as the number of corners in a specific area (the corner density) goes up, the traceability of the lines through that area goes down. The importance of this observation is that traceability of the diagram is a function not only of the crossovers in the net (which can be found using graph techniques[MMM83]) but also on the *density* of the diagram, which can only be found by managing the routing *regions*. This requires good space management techniques to be effective.

Figure 1 shows two regions with the same number of nets crossing them. In the two figures, the crossover and density counts are reversed. We maintain that the region with the high density count is more difficult to trace than the region with the high crossover count.

This indicates that the goal of traceability is better measured by the density of corners rather than by the number of crossovers.

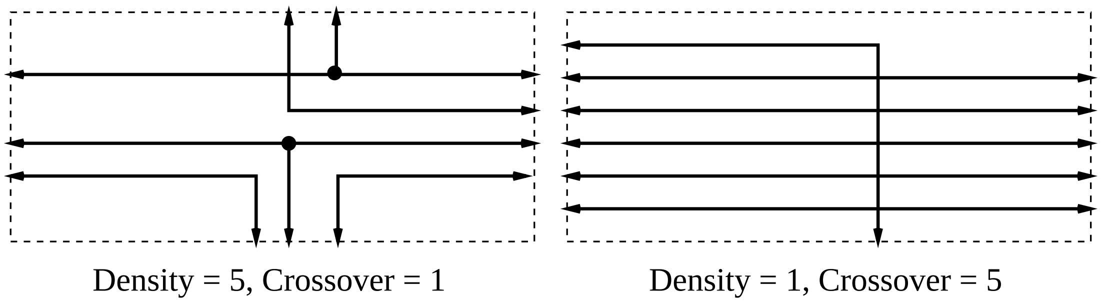

Figure 1: Density vs. crossover count as a measure of traceability

Therefore, simple rules for crossover, bend and connection counts completely ignore the density issue, as they have no tie to the areas in which they occur. Many such crossovers or bends, when spread over the diagram do not strongly hinder the traceability of the net. However, just a few such bends in a small area can make the given net much more difficult to follow. We refer to areas that have a high corner density as being *congested*, and hence the *congestion* of a given area is an inverse measure of the traceability of the lines which cross that area.

Though many ASG systems base their schematic routing on either minimizing the number of crossovers, or applying specific ordering rules that model what human designers do. We observe from Figure 1 that the minimum crossing solution is not *necessarily* the most traceable solution.

Even when the congestion issues have been adequately addressed, the crossover problems (as exemplified in Figure 2) still need to be addressed. The discussions presented [SH90, Bre75, MMM83, SK89] clearly illustrate the importance of line positioning to the readability of the schematic.

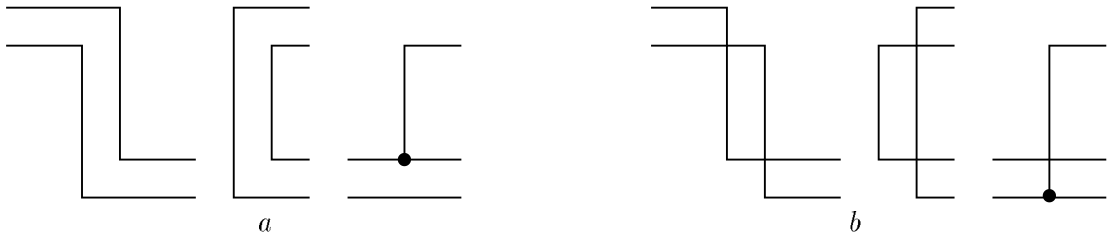

Figure 2: Good and bad line crossing solutions.

Figure 2 shows several instances where line crossing clearly detracts from the traceability of the lines. Three good combinations are shown on the left, juxtaposed with three bad combinations. The relative corner placements in Figure 2a are much better than those in 2b. There is a further complication to the problem of line positioning in addressing line–crossing issues, that being the linked nature of corners. Whenever a corner c is fixed, this necessarily fixes the lines connected to that corner. Fixing corner c restricts where the other corners linked to c can be placed, and although c may have an optimal placement, the restrictions that this placement has on the corners linked to it may make their placement sub-optimal. We view this as a problem in determining the order of evaluation, which is addressed in our local router.

Thus there are two traceability issues to address: congestion *and* crossover. We address these by dividing the router into two stages: A global router to discover a solution to the route that considers the congestion of the routes formed, and a local router that maps the given global route into a completed route with a satisfactory crossover solution.

Since congestion can only be addressed *after* a route has been proposed, we employ iterative-refinement combined with a search technique to discover the global route. We address the crossover problem in the local router by using a constraint-propagation technique along with some straight-forward ordering rules. By creating and linking the net constraints, we permit portions of nets to be easily ordered for the desired aesthetics, and the the final positions are adjusted to use the available routing space.

Given our formulation of the schematics routing problem, and the observations as to good approaches to the solutions of the issues raised, we go on to describe the details of the system.

3 Global Routing

Routing in SPAR is broken into two phases: global routing, where the general location of runs and bends in the nets is discovered, and local routing, where these overlapping areas are seen as combinations of constraints that need to be satisfied. The global router determines the number and location of bends in the broad sense, whereas the local router places the corners and makes connections among them. The combination of these two techniques aims at maximizing the readability of the resulting diagram.

The global route is required to connect all points of all nets, but also to yield an overall traceable result. We do this by performing a heuristic search that makes multi-point node expansions using cost metrics based on the *complexity of the route* (*i.e.* the number of bends) and the *congestion of the route* (meaning the number of other nets and corners that are grouped in areas that the net passes through).

To facilitate the evaluation of congestion for the different routes under construction, we employ corner-stitching techniques [Ous84] to represent the routing space. By developing the routes in data-structures that have a fixed area, we can evaluate the areas to determine congestion directly. Cornerstitching also removes the grid restrictions that are present in many existing ASG systems.

The use of cornerstitching, and the internal maintenance of the global route combine to make this router incremental. That is, new modules and nets can be incrementally added. We discuss this feature at length in the presentation of SPAR [FL92], the ASG System of which this router is a part.

Part of the area phenomena of congestion is that it is a function of the other nets in the circuit. This implies that congestion can only be evaluated after a set of global routes has been proposed. Once a route exists, measures for congestion are evaluated, and the nets are relaid using the existing congestion information. Thus the discovery of the global route is broken into two passes: the first pass aims at finding a good approximation to the best routes for each net, and the second selectively rips up and reroutes the diagram taking congestion into consideration. The mechanics of each pass operate in the same way, only the cost metrics are different.

We use a best-first search to discover the initial and final global routes. Rather than recalculate routes after the congestion is known, paths are maintained between phases so that only the cost calculations are redone.

3.1 The Search Process

The search process is actually a series of searches, one for each terminal of all nets. A partial route (*path*) is constructed until the path for one terminal connects to another path of the same net. A path that connects is said to be *complete.* The collection of paths associated with a given terminal of a net is referred to as an *expansion.* At each move, the best path seen is selected, and the last tile on that path is expanded, creating several new paths. When the best path selected is complete, then the search for that expansion is terminated. The search takes place in parallel, in that each expansion (there is one expansion for each terminal of each net) will make only one move until all expansions have completed.

Each move in the search consists of a single tile from the routing space being added to an existing path. This space is composed of two, superimposed 90°-rotated tile spaces, which contain the modules as obstacles to the route. The use of two tile spaces is to take advantage of one of the peculiarities of the cornerstitching algorithm, that being that free tiles (those without obstructing contents) extend as far in the horizontal (or vertical) direction as possible.

Each step in the search proceeds by selecting an expansion and choosing the lowest-cost path yet seen. The tile on the end of a path is used to select a set of perpendicularly-oriented tiles which are expanded into, thus creating more paths. Figure 3 shows one expansion of a net for the SN7474 example. The example depicts an expansion for a path for net *y* of the expansion associated with the input terminal of module *NAND.1.*

This path contains three tiles, two horizontal (*b* and *c*) and one vertical (*1*). The last horizontal tile on the path (tile *c*) is being expanded into the vertical plane, and can expand into the four vertical tiles (*2, 3, 4,* and *5*) marked by *?*. Thus the one path will have a vertical tile added to it, and three copies of the path will be made, with each of those having a different vertical tile added to it. The resulting paths to be considered again for expansion will be *b-1-c-2*, *b-1-c-3*, *b-1-c-4*, and *b-1-c-5.*

The following four steps summarize the actions taken at each search step:

1 *Select the least-expensive path for the given expansion.* This is the path that will be expanded. Of most importance is the last tile on the path, which is called the *current tile.*

2 *Check the current tile to see if this path should terminate.* Termination occurs when a terminal expansion (set of possible paths to the given point) contacts another active expansion.

3 *Use the current tile to expand into all connected, free, perpendicular tiles.* For each valid tile selected, a list of perpendicular tiles is collected, and for every tile not yet visited, the current path is copied, with new tile added to it. This 'not-visited' criterion is to avoid creating cycles, which would cause duplicate paths to compete for expansion.

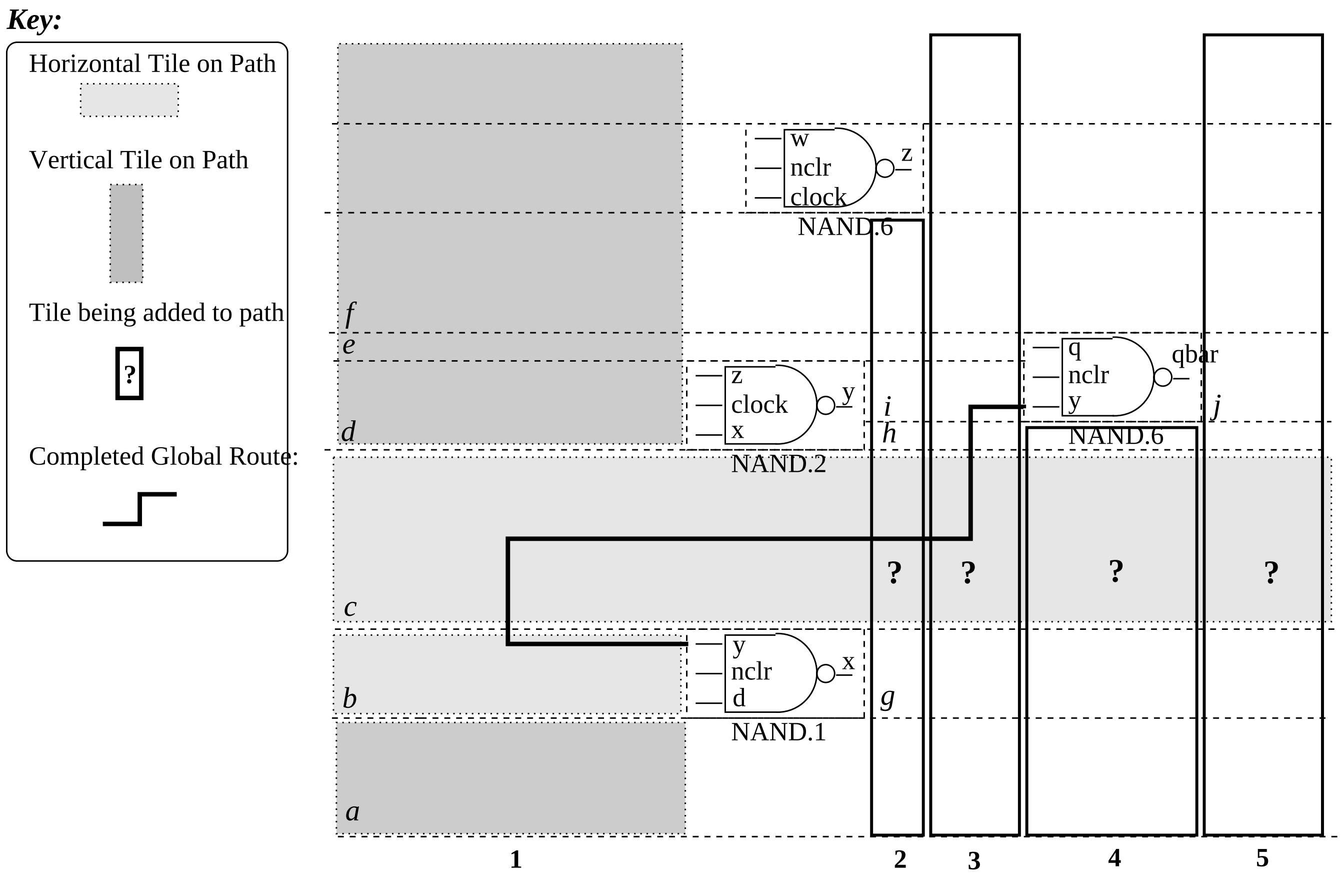

Figure 3: Path Expansion for one path of net y for the SN7474.

4 *Determine the cost of each of these new paths.* The active paths for a given expansion are ordered. Therefore, whenever this expansion is checked, its least-cost path can be quickly found.

When a tile of a path is going to be checked for expansion, it is first evaluated to see if it completes a connection to another expansion that is part of the same net. If this expansion contacts another, yet-incomplete expansion of the same net, it may terminate. All of its paths will be merged with those of the still-active terminal. The expansion process continues until a termination leaves *only* the contacted terminal active, which is then terminated. This ensures that all terminals on a net are connected. When all expansions on a net are terminated, then the net is completed. When all nets are complete, then the global router is complete.

3.2 The Cost Estimation Metric

The key to the success of the search process is the nature of the cost estimate used. There are two key factors to consider: the first is that the cost estimate models the relative importance of the various qualities of the finished route, such as low congestion and few turns. The second is that all costs are directly comparable. The addition of a low cost estimate of the distance-to-go to all partial routes insures that all paths are comparable. This estimate also insures that once a completed path is discovered to be the lowest-cost path, then there is no better path between the contacted terminals.

The path cost estimate is made up of several parts: a length cost, a corner cost, and a congestion cost. Here the length cost is a weighted measure of the path length, both known and estimated. The corner cost a weighted measure of the corner count, and the congestion cost is a weighted measure of the usage of the areas in which the corners fall.

$$Path\ Cost\ Estimate = Congestion\ Cost + Corner\ Count \times cw + Path\ Length \times lw \quad (1)$$

where:

$$Path\ Length = \overbrace{\sum_{i=1}^{no. of tiles} avg.\ distance(tile_i, tile_{i+1})}^{Avg.\ Known\ Path\ Length} + \overbrace{[|termX - \bar{x}| + |termY - \bar{y}| \times f]}^{Est.\ Length\ to\ Nearest\ Terminal} \quad (2)$$

Where $\bar{x}$ and $\bar{y}$ are the position of the last known corner in the average path, and $termX$ and $termY$ are the locations of the terminal nearest to ($\bar{x}$, $\bar{y}$).

The completion of the first search process creates a first–pass at the route: It should be a reasonable approximation to the best route, in that the paths selected have been optimized for the number of corners and the length of the route. The dark line in Figure 3 shows the completed global route for net y of the SN7474. This route consists of tiles *b, 1, c, 3,* and *i.*

Some parallel can be drawn between the modified line–expansion router utilized in *Eureka* [SK89, KS89] and the first pass of our global router. This similarity exists in that the selection of alternate tiles from the two tile spaces parallels the action of a line–expansion router avoiding an obstacle. We gain a significant advantage over the router used in *Eureka* in that we map the route to a tile space, from which our route is queried for information which is used to address congestion considerations, and when used incrementally, to accurately place other (as yet unplaced) modules.

Once the first-pass paths for all nets have been selected, the path costs for all nets are recalculated to include congestion. This includes both the completed nets, and those partial nets that were in competition. The overall congestion associated with the first-pass path for each net is evaluated and used to order the nets for the iterative refinement stage.

3.3 Using Iterative Refinement to Solve Congestion Problems

It is necessary to complete a set of routes before congestion can be considered; this is because congestion is only only meaningful for completed, rather than partial routes. This implies that to consider congestion requires an iterative refinement technique. The technique we employ consists of ordering the nets by their congestion values, and working from worst net to the best, ripping up the route and rerouting it using congestion considerations.

Iterative refinement starts by adding the first-pass routes back to the list of more expensive or incomplete paths associated with the expansions for that particular net. These paths are then recosted, reordered and finally the search process is restarted. As all of the routing information has been retained, it is simple to recost the paths contained in the net's expansions. Depending on the congestion problems, the old route might well be reselected, but this is by no means assured.

Our cost metric adds to the cost of a particular path based on the number of nets that actually pass through each tile on the path. Normally, the congestion cost is a weighted ratio of the number of tracks used, over the number of tracks available. The exception to this rule is when this ratio equals one. Such a tile is deemed *overused*, and warrants special treatment. The cost for overused tiles increases quickly, so that any path using such a tile is unlikely to be a low-cost path, and likely to be ripped up if it is part of a first-pass path. To summarize, the congestion cost is defined to be:

$$Congestion\ Cost = \begin{cases} \frac{Tracks\ Used}{Tracks\ Available} \times w & \text{if } Tracks\ Used < Tracks\ Available \\ 2w \times \frac{Tracks\ Used}{Tracks\ Available} + c & \text{otherwise} \end{cases} \tag{3}$$

where w is a constant weight which scales the congestion values to the other values in the path cost estimate. A constant c is used to insure that the penalty for tile overusage is sufficiently steep to prevent overused paths from being considered.

The value of re-routing the nets using congestion factors can be seen in the two examples of Figure 4. Figure 4a shows a completed function generator without congestion considerations, and Figure 4b shows the same schematic using congestion. The difference between the two figures is clearly seen in the more distributed placement of nets *rco, q_1, q_2, q_3,* and *q_4* among many others. The right-hand figure is clearly easier to trace. The placement for this example closely matches that depicted in [Tex86] for an application of the SN74AS850.

4 Local Routing

We use a constraint-propagation approach in our local router to address the previously discussed line-crossing issues. This approach is to arrange corners such that all corners of the same net at the same x or y position share the same value for that position; thus a single change to the value is automatically propagated to the necessary corners. This allows us to evaluate a single area at a time, as the changes made to one net in one area will automatically be propagated to the other corners of the net. Each area is evaluated by imposing an ordering relation to the corners in the given area. We place lines by applying our *Rules of Easy Reading* (Figure 5) to the lines in a given area. The horizontal tiles of Figure 5a are evaluated to determine the ordering for the vertical lines contained. Similarly vertical tiles are evaluated to determine the ordering of horizontal lines (Figure 5b). These rules insure that unnecessary line-crossings are avoided.

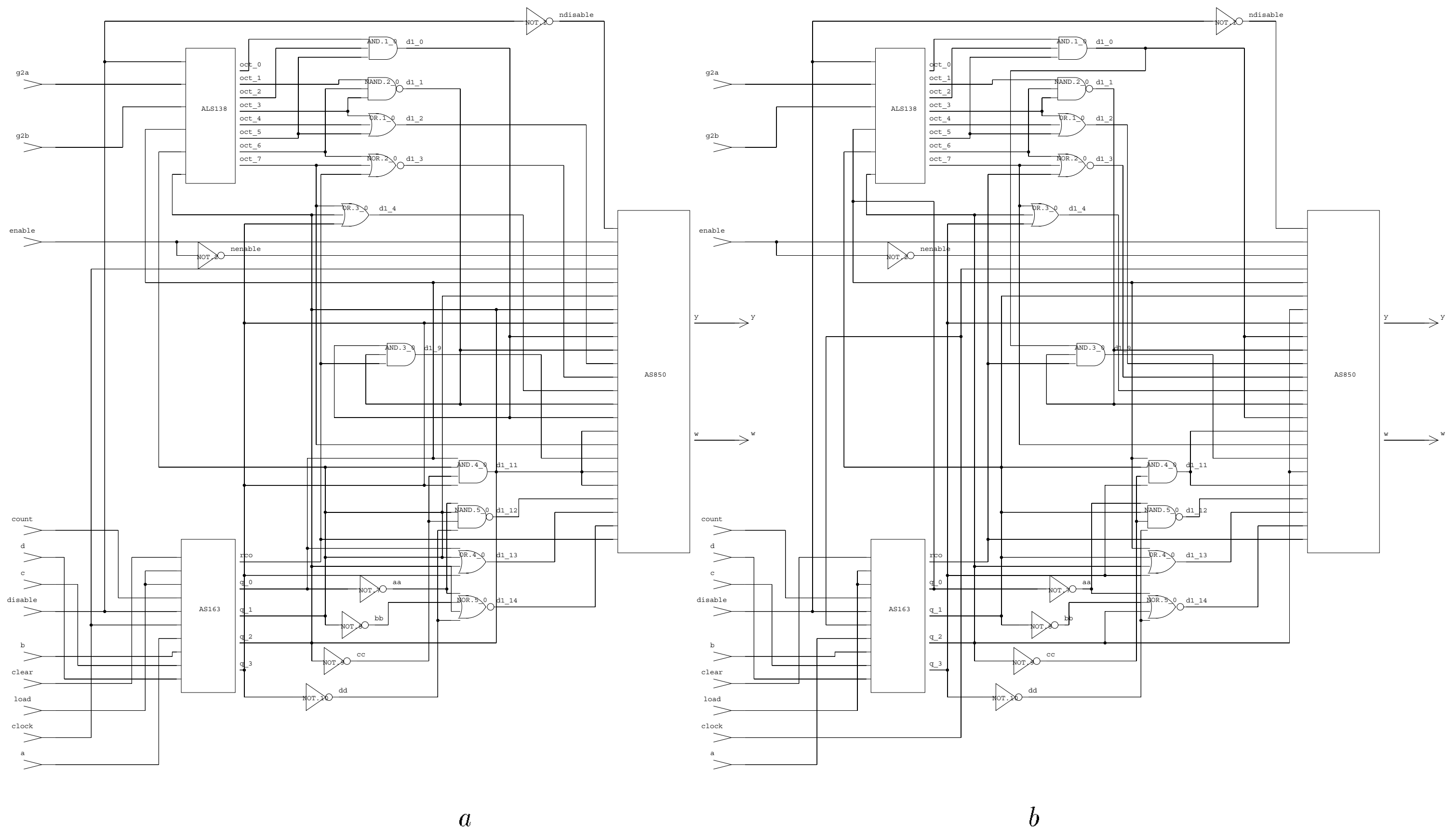

Figure 4: Routing for a function generator with congestion turned off (a) and on (b)

4.1 Mapping the global route into corners

The local routing process begins by first mapping the global routes of each net into a linked sequence of corners, where the actual (x,y) positions of the corners are not known. Rather, these positions consist of *ranges*, where a range is a continuous set of points that match the limits determined by the paths used in the global route. These ranges define the x and y positions where the corners may be placed, and are used to link attached corners. These ranges are initially set by the widths of the tiles in the global route, and are created as the global route is mapped into corners.

The results of the corner creation, linking, and collapsing process are illustrated in Figure 6. Here the global routing tiles for a three-terminal net are presented with a representation of the local route under construction. The net will have two floating corners, linked to the terminals of the nets. These two floating corners are linked by three ranges, two of which are vertically-oriented, and one of which is horizontal. *Ranges 1* and *3* initially take on the values $[y3..y4]$ and $[y1..y2]$, but as they are linked to terminals, their ranges are fixed to these y positions. *Range 2* will take on the range $[x1..x2]$, as fixed by the vertical tile in the net's global route.

The corners of a route are linked so that the x and y fields of the corners all point to a single

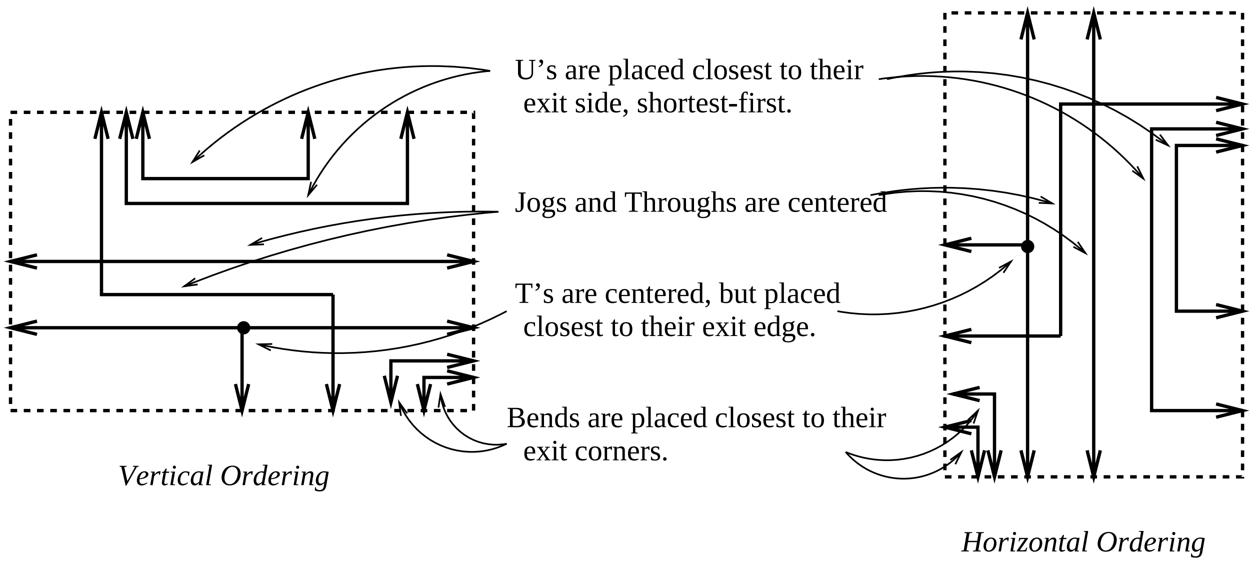
U's are placed closest to their exit side, shortest-first.
Jogs and Throughs are centered
T's are centered, but placed closest to their exit edge.
Bends are placed closest to their exit corners.
Vertical Ordering
Horizontal Ordering

Figure 5: *Ordering Rules for Easy Reading*: The relative placement of segments

range that defines their values. In this way, any restriction made to one corner will immediately be reflected in the linked corners. For example, any change to *Range 2* of Figure 6 will affect *Corner 1* and *Corner 2* equally. All modifications that relax/restrict the initial ranges are communicated to all corners equally. Figure 7 shows several corners with various range/link combinations.

4.2 Separating Overlapped Ranges

The task of the separation algorithms is to determine the final values that all ranges are to take on. Insuring that routes take on legal, traceable values implies that the ranges may not be left overlapping, and should be selected so as to place corners relative to each other in an appropriate way. This process is applied to each tile in the routing space, starting with the most congested tile, and is based on the judicious positioning of overlapped ranges.

In the process of evaluating these tiles, the local router separates the ranges in question. This separation process has an exactly-opposite effect to that of a standard channel-router: the routes are spread evenly within the available space thus limiting the congestion of the individual tile, and making the routes through it as traceable as possible.

The use of location information to solve the range overlap problem is based on the use of the tile space. Vertical ranges specify the y location of at least two corners, and thus define where a horizontal line will fall. Similarly, horizontal ranges define the x location of two or more corners, and those corners that may overlap in the x dimension will be found in the same vertical tile. Thus the horizontal ranges that may overlap are found in the vertical tiles. The case holds true for vertical ranges and the horizontal tiles, which greatly simplifies the separation process.

Given the dependence of one corner position on another, there arises an ordering problem as to which corner should be fixed first. As mentioned, the most-congested tile is selected first for separation, and then the next most congested and so on, until all tiles containing corners

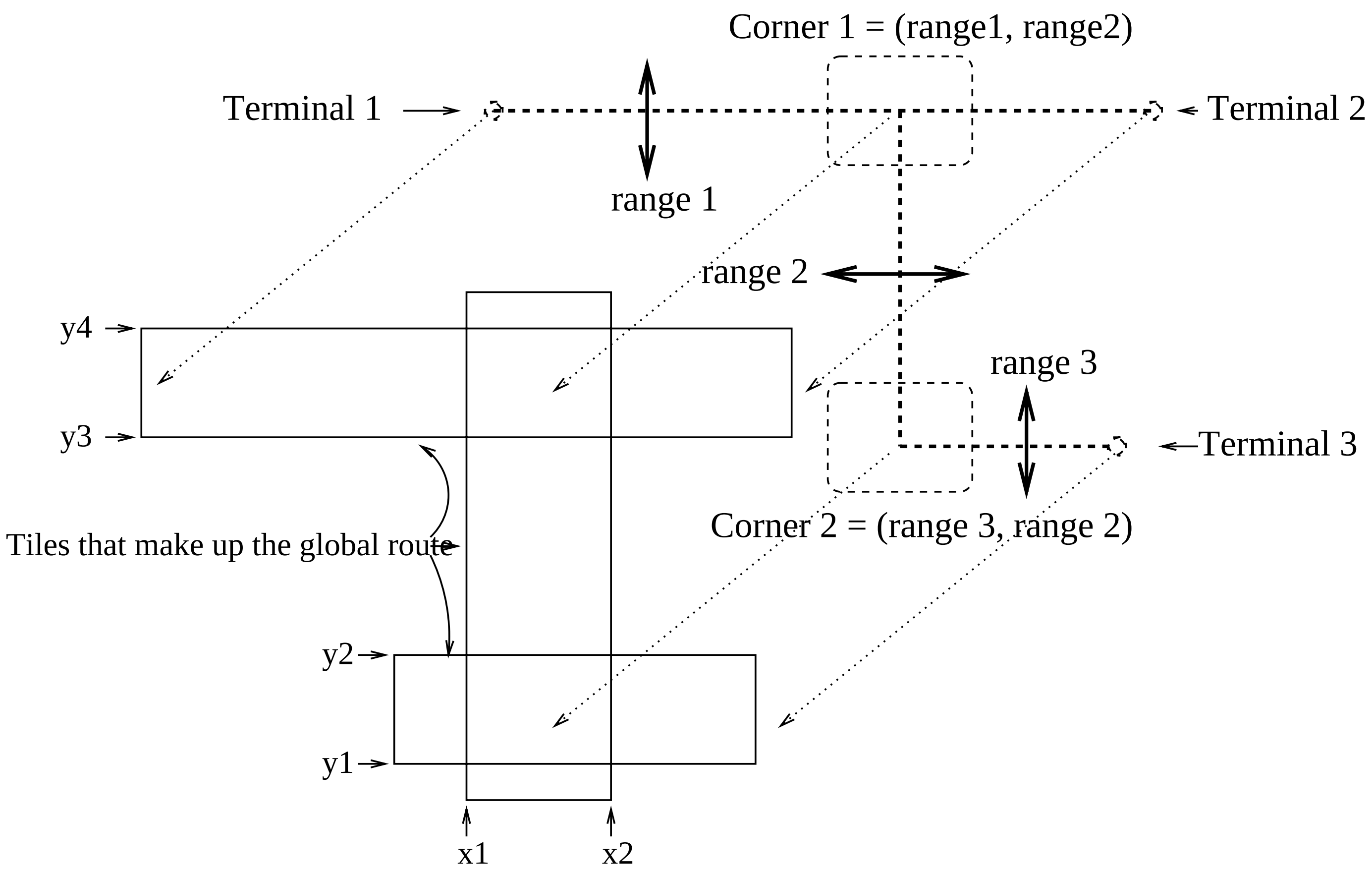
Corner 1 = (range1, range2)
Terminal 1
Terminal 2
range 1
range 2
range 3
Terminal 3
Corner 2 = (range 3, range 2)
Tiles that make up the global route
y4
y3
y2
y1
x1
x2

Figure 6: Mapping a global route to a set of corners.

have been separated. This forms a natural ordering, as the most congested tiles are given preference for having the cleanest route.

The range separation operations performed on each tile breaks down into three steps, *range-ordering*, *dependency-graph creation*, and *space distribution*. *Range ordering* is a mapping of aesthetic rules to the given ranges such that the desired relative placement of each range is known. A *dependency graph* is a list of the interfering ranges, sequenced by the given range-ordering. *Space distribution* is the mapping of a given ordered list of ranges to the overall space which they can occupy such that each range takes on a unique (non-overlapping) value.

Ranges within a tile are ordered by examining one corner for each range, classifying it, and then using the classification along with location and connection information to make the ordering comparisons. Corners are classified as jogs, bends, T's, or U's, depending on whether the other corners to which they are linked fall inside or outside of the tile being examined. This ordering follows the *Rules of Easy Reading* of Figure 5.

The horizontal line segments of Figure 8 illustrate the vertical ranges selected for ordering in a single horizontal tile. Each segment has a unique range which determines the legal y-values that position the corners of the segments. There are four segments, contained (one for each net), and therefore four vertical ranges to be ordered. Two ranges(*v.range3* and *v.range4*) are classified as a jogs, as both corners that use the vertical range exit on opposite sides of the tile.

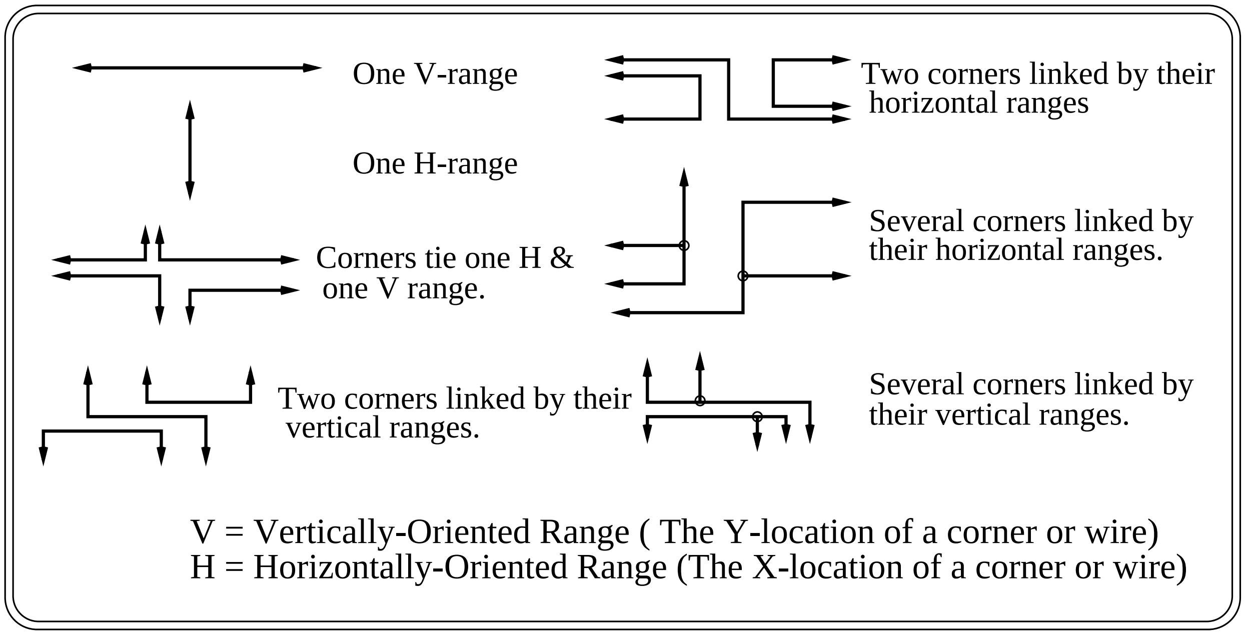
One V-range
One H-range
Corners tie one H & one V range.
Two corners linked by their vertical ranges.
Two corners linked by their horizontal ranges
Several corners linked by their horizontal ranges.
Several corners linked by their vertical ranges.
V = Vertically-Oriented Range (The Y-location of a corner or wire)
H = Horizontally-Oriented Range (The X-location of a corner or wire)

Figure 7: Examples of range/link combinations

The ranges *v.range1* and *v.range2* are classified as U's, as they have two corners that exit the tile on the same side. From this classification, and the arrangement of the corners, the range ordering is determined.

In this example, *v.range2* is placed lowest, as it is the shortest U, and exits the bottom of the tile. Similarly, *v.range1* is the next lowest, it also being classified as a U. The two jogs are more interesting, as their vertical ordering is somewhat arbitrary. As net q exits the tile furthest to the left, its range is placed above that of net *qbar*.

Once the given set of corners has been ordered, the dependency graphs that indicate which ranges need to be compared must be formed. For the corner-placement problem, a dependency graph is a set of ranges that must be separated.

A range z is said to be dependent on another range x ($z \Rightarrow x$) if x should be placed before z, and the line segment associated with the x overlaps any portion of the line segment associated with z. This relationship is transitive, if the line segment of x overlaps y which overlaps z, then z is still dependent on x. Given that $\Rightarrow$ denotes the dependency relation described, x is said to be a parent of the graph $\{x, y, z\}$ if $z \Rightarrow y \Rightarrow x$, and there are no ranges in the tile that x depends on.

It is clearer to see the implications of a range that is *not* dependent on another range: such ranges do not need to be compared for overlap resolution, as the line segments positioned by these ranges can share the same column or row. The purpose of the dependency graph is to divide the set of ranges into subsets containing groups of ranges that cannot overlap. The important implication is that all ranges that are *not* part of a particular graph may overlap with all of the ranges in the graph. This implies that ranges may occur in more than one dependency graph. By paying attention to what ranges need to be separated, the use of dependency graphs in the separation process ensures that the maximum available space is

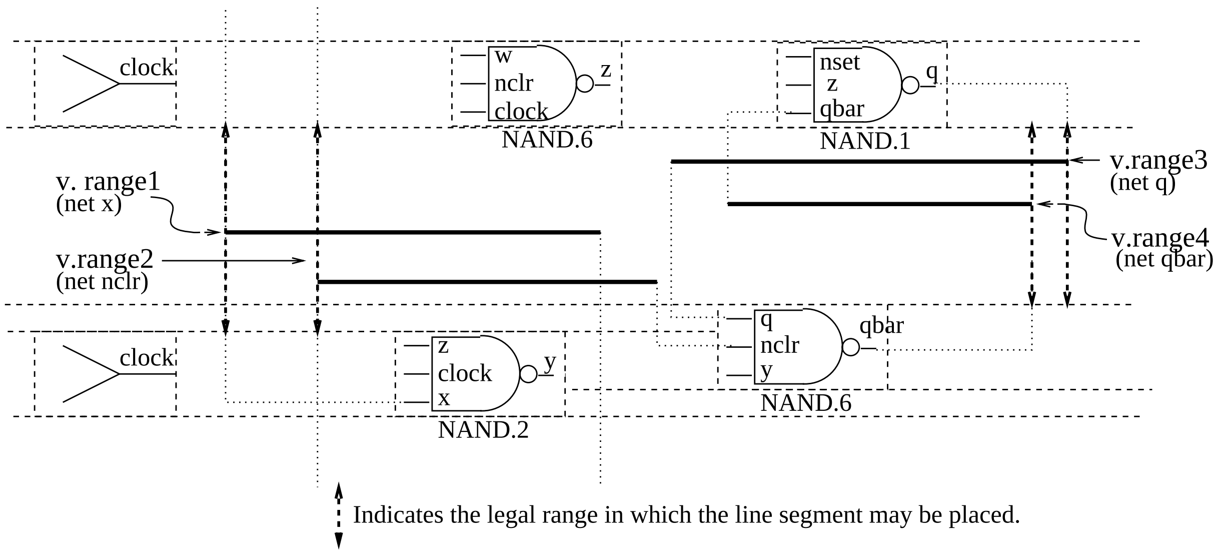

Figure 8: Separation of vertical ranges showing range ordering

allocated to each range.

The last step of the separation process is to use the dependency graphs formed to remove the overlap from among the ranges being compared. This evaluation distributes the available space among the ranges of a dependency graph so that none will overlap. The longest dependency graph in the tile is evaluated first, and then the next longest and so on, so that any repeated ranges are adjusted to match the tightest constraints first. The order of the ranges within the dependency graph matches the ordering done in the first step of the separation process. This ordering is crucial, as the space available is distributed in this same order.

For each graph, the range that the entire graph has to fit in is determined by the minimum and maximum of the ranges at the ends of the graph. The space between these two points is divided evenly among all of the ranges of the graph, such that each receives a slot, and care is taken to insure that the values assigned to the slot are legal restrictions on the range values. Figure 9 shows the completed SN7474 flip-flop. Note how the nets in the input column are evenly spaced to fit within the given space.

The separation process for an individual tile completes when all dependency graphs for the tile have had any overlap removed. The result of applying this separation process to all tiles results in a complete removal of all overlap among the ranges that are used to locate the corners of the route within the diagram. The final task remaining to the local router is to center these ranges to the middle of their available ranges. This provides the maximum spacing between lines, so as to further enhance the traceability of the diagram.

5 Discussion and Future Work

In this paper, we have presented a new set of algorithms that route schematic diagrams, and have demonstrated how these are useful for routing traceable schematic diagrams. The algorithms model the traceability of a schematic net by searching for global routes in a routing

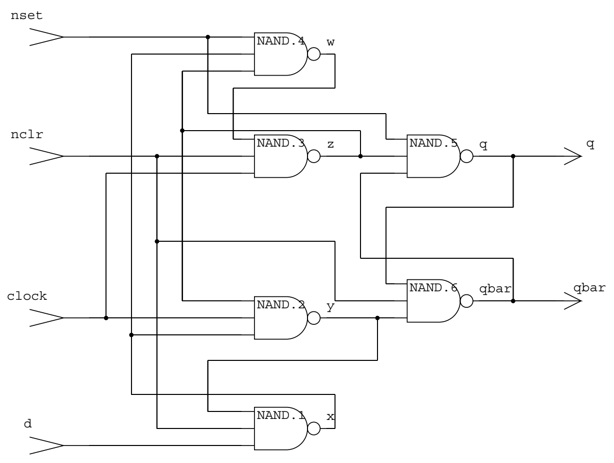

Figure 9: Completed SN7474 flip-flop.

space where congestion can be estimated, and by addressing crossover issues in a local router based on constraint-propagation. The techniques employed can be used without the restriction of an underlying grid, and can be used in an incremental environment.

Figure 10 shows a the results of congestion routing applied to a typical multi-row schematic. The placement for this example comes from [SH90].

Figure 11 shows the same schematic as placed in [May85]. Here our router takes advantage of the more optimized placement. Both examples illustrate how the application of a congestion-based global router and crossover-based local router combine to produce traceable schematics.

The most interesting results of this work have been in two areas: the successful application of two-dimensional space-management techniques employed in the global router, and the constraint-propagation techniques used in the local router. Most notable is the application of the space management techniques to allow for incremental placement and routing. There are other area-phenomena to which the space-management techniques could be applied, as well as further developments of geometric constraint-propagation to be explored. Proposed extensions to the router developed include extending the router to handle bus structures, 45° angle corners, and reordering of pins within modules where applicable.

The results presented in this paper demonstrate that an algorithmic approach to schematic routing is both feasible, and can yield good results, especially for smaller schematics. Our experience in developing these shows that routing for schematic generation is not a trivial problem, and requires special-purpose algorithms and data-structures to yield good results.

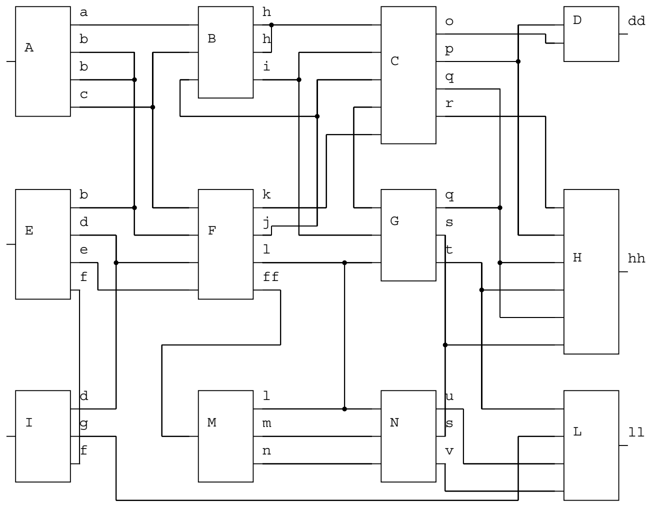

Figure 10: Congestion router applied to a multi-row schematic from [SH90].

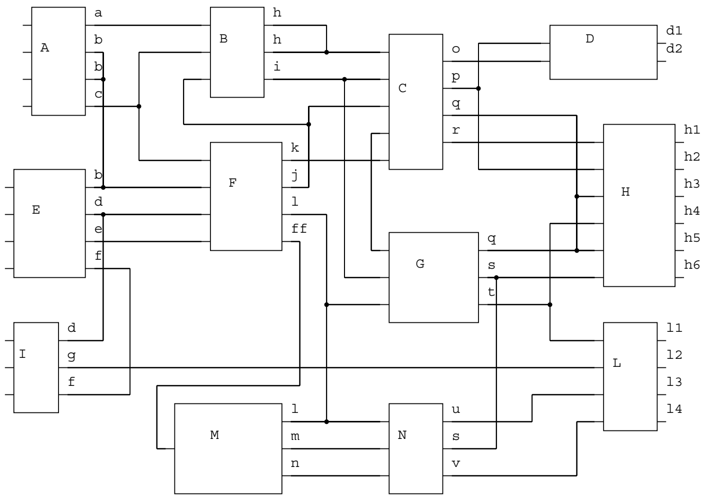

Figure 11: A Typical Multi-row Schematic from [May85].

References

[AAM85] V. Swaminathan A. Arya, A. Kumar and A. Misra. Automatic Generation of Digital System Schematic Diagrams. *22nd Design Automation Conf.*, pages 388–395, 1985.

[AHS83] M. L. Ahlstrom, G. D. Hadden, and G. R. Stroick. HAL: A Heuristic Approach to Schematic Generation. *IEEE International Conf. on Comp. Aided Design*, pages 83–86, 1983.

[BJS$^+$89] S. K. Baruah, M. R. Jerath, S. Sundaresan, S. Banerjee, S. Kumar, A. Kumar, and P. C. P. Bhatt. A Blackboard Architecture to Support Generation of Schematics for Design Automation. *Proc. IFIP TC 10 / WG 10.2 Working Conf. on CAD Systems Using AI Techniques*, pages 51–58, June 1989. North Holland Press, G. Odawara Ed.

[Bre75] R. J. Brennan. An Algorithm for Automatic Line Routing on Schematic Diagrams. *12th Design Automation Conf.*, pages 324–330, June 1975.

[FL92] S. T. Frezza and S. P. Levitan. SPAR: A Schematic Place And Route System. *IEEE Transactions on Comp. Aided Design*, Expected 1992. Submitted July, 1991.

[Fre91] Stephen T. Frezza. SPAR: A Schematic Place And Route System. Technical Report TR-CE-91-02, Deptartment of Electrical Engineering, University of Pittsburgh, Pittsburgh, Pennsylvania, April 1991.

[KS89] G.J.P. Koster and L. Stok. From Network to Artwork: Automatic Schematic Diagram Generation. Technical Report EUT 89-E-219, Eindhoven University of Technology, Eindhoven, Netherlands, April 1989.

[May85] M. May. Computer-Generated Multi-Row Schematics. *IEEE Transactions on Comp. Aided Design*, 17(1):25–29, January 1985.

[MMA86] T. Fuhrman M. Majewski, F. Krull and P. Ainslie. AUTODRAFT: Automatic Synthesis of Circuit Schematics. *IEEE International Conf. on Comp. Aided Design*, pages 435–438, November 1986.

[MMM83] A. Iwainsky M. May and P. Mennecke. Placement and Routing for Logic Schematics. *IEEE Transactions on Comp. Aided Design*, 15(3):89–101, May 1983.

[Ous84] John K. Ousterhout. Corner Stitching: A Data-Structuring Technique for VLSI Layout Tools. *IEEE Transactions on Comp. Aided Design*, CAD-3(1):87–100, 1984.

[SH90] G. M. Swinkels and Lou Hafer. Schematic Generation with an Expert System. *IEEE Transactions on Comp. Aided Design*, 9(12):1289–1306, December 1990.

[SK89] L. Stok and G.J.P. Koster. From Network to Artwork. *26th Design Automation Conf.*, pages 686–689, June 1989.

[Tex86] Texas Instruments, Dallas, Texas. *LSI Logic Data Book*, 1986.

[VW86] V.V. Venkataraman and C.D. Wilcox. GEMS: An Automatic Layout Tool for MIMOLA Schematics. *23rd Design Automation Conf.*, pages 131–137, 1986.