An Interactive Maze Router with Hints

Michael H. Arnold
Walter S. Scott

Computer Aided Design Group, O Division
Lawrence Livermore National Laboratory
University of California
Livermore, CA 94550

Abstract

This paper presents the Irouter, an interactive maze router for the Magic IC layout editor that takes hints. The Irouter is a flexible tool intended to be useful wherever tight or unusual constraints do not permit automatic routing. It has already been used in the layout of an 80,000 transistor CMOS chip developed by our group at LLNL, and is currently being used to route the control signals of a 100,000 transistor, high performance, FPU chip being developed by the Berkeley SPUR project.

Several novel ideas for maze routing have been developed in the Irouter. Hint layers permit the user to map out the general path of a route and pull the route in desired directions, while leaving details, such as obeying the design rules, to the router. The gross structure of the layout is preprocessed to facilitate accurate estimates of cost to completion during routing and hence effective pruning of misdirected partial routes. A windowed search strategy slowly shifts the focus from the start point towards the goal. This permits the consideration of alternatives at all stages of routing without blowing up into an exhaustive search.

1. Introduction

Manual routing is still often necessary. Nets that the automatic routers could not complete, off-grid routing in tight spots, nets with critical timing constraints, and other special considerations, can all require hand treatment. Manual routing involves two distinct steps. The first: factoring in special considerations and determining the best overall path for a route, is usually easy for a human user, but difficult for automatic systems. The second step: the detailed implementation of the route, requires the user to painstakingly "crawl" along the path of the route at a scale sufficiently fine to observe the relevant width and spacing rules, a very slow and error-prone process, which should be automatable. This suggests a sort of "power tool" approach to routing, midway between manual and fully automatic routing, where the user describes the general path of a route with "hints", but leaves the details of the route: obeying the design rules, placing contacts, and jogging around minor obstacles, to the computer.

The work described here was supported in part by the Department of Energy under Contract No. W-7405-Eng-8 and by the Department of Defense VHSIC program under Contract No. FY1175-85-N-5695.

This paper presents the Irouter, an interactive maze router for the Magic layout editor [1] that is intended to be useful whenever special requirements do not permit fully automatic routing. The Irouter proceeds one point-to-point connection at a time. The user suggests the overall path of the route via graphic hints, and controls characteristics of the route such as width and preferred horizontal and vertical routing layers via parameters. The router automatically takes care of the detailed implementation of the route, constructing a design-rule-correct path, that attempts to minimize cost factors including the overall length of the route, the number of contacts, and the use of nonpreferred layers. The Irouter has been used in the layout of a 80,000-transistor CMOS chip developed internally by our group at LLNL, and is currently being used to route the control signals for a 100,000-transistor high-performance floating-point chip being developed by the SPUR project at Berkeley. Cross-chip routes on the floating-point chip typically take 5 seconds on a SUN 3/110 (an approximately 2 MIPS machine). The Irouter will be included in the 1988 Magic release, available from U.C. Berkeley by publication time.

To meet its objective of being useful when fully automatic routing fails, the Irouter must be flexible and controllable enough to handle the special circumstances that thwart automatic systems. In addition, the Irouter must be fast enough for the interactive style of use it is designed for. The mutual satisfaction of these three key requirements: flexibility, controllability, and interactive performance, has required several inventions.

The flexibility requirement dictates a maze routing approach. Only maze routing permits free-form routing, without restriction to tracks, two layers, or other stylizations. The Irouter computes maze routing blockages directly from the edge-based design rules in the DRC-section of the Magic technology files. This permits more complex design rules than other approaches.

Controllability was achieved by augmenting traditional cost parameters with two styles of hints: *fences* and *magnets*. Fences permit the user to designate areas to route inside of as well as areas to stay clear of: the router will not cross a fence boundary. Magnets are used to pull routes in desired directions. Hints can be used to specify the overall path of a route, to cause routes to hug up against existing wiring, or to reserve space for future routing. They have proven effective for quickly communicating the overall structure of a route, without being bothered by the details.

CH2540-3/88/0000/0672$01.00 © 1988 IEEE

Much effort has gone into making the Irouter fast. The price of the flexibility obtained by maze routing is the concomitant huge search space that makes maze routers notoriously slow. Several novel techniques were pioneered in the Irouter to obtain acceptable performance despite the use of general maze routing. A feature-based *interesting point* expansion technique eliminates much wasted processing time on neighboring points where nothing has changed. A global cost estimation prepass allows the router to distinguish between promising and hopeless paths based on the gross structure of the layout. Finally a windowed search progressively shifts the focus of the search nearer the goal. This allows alternatives to be pursued at all stages of routing, without exploding into an exhaustive search.

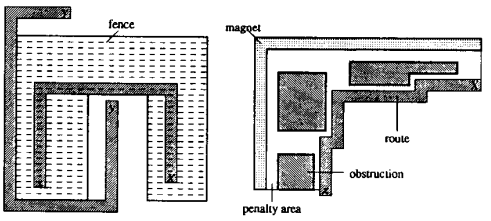
Section 2 discusses control of the Irouter, presenting the hint mechanisms in detail. Section 3 explains how the Irouter handles design rules. Finally the routing algorithms themselves are given, in Section 4, with emphasis on the new performance optimization techniques. Section 5 gives a brief conclusion.

2. Control of the Router

The success of the Irouter as a replacement for manual routing hinges on its controllability: the user must be able to obtain precisely the routes he wants, as if he were routing them by hand. This kind of control is obtained through an extensive set of parameters (discussed at the end of this section) and two hint mechanisms: *fences* and *magnets*.

Fences are rectilinear regions, on a special **fence** layer. The Irouter will not create routes crossing fence boundaries, thus they are useful both for designating a routing region that a route must stay inside of and for establishing regions that the router is not to trespass on; see Figure 1(a). The primary use of fences is for establishing the path of a route, or sequence of routes on a gross scale. An important side benefit is their restriction of the search space for the maze route, which improves the routers performance, often dramatically.

Magnets, also a special layer, serve to pull routes in desired directions; see Figure 1(b). They are typically used to cause a route to hug one side of a routing region. The area between the actual route and the nearest magnet is factored into the cost of a route, so that the further and longer a route stays away from a magnet the more it is penalized. The strength of magnets is controlled by the weight on the penalty areas. If a low weight is chosen, magnets will be weak, and can be used to prefer a route that goes up and then right over one that goes right and then up, say, but will not induce extra jogs. On the other hand if a large enough weight is chosen, even a route on an unfavorable layer will be preferred, if it follows a magnet more closely. The best strength for magnets depends on exactly how and where they are used, but generally an intermediate strength is indicated: one that will cause routes to jog toward magnets they parallel for a significant distance, but will not cause them to change to a nonpreferred layer. Unlike fences, the use of magnets, particularly strong magnets, tends to degrade the performance of the router. This is because they can cause a severe discrepancy between estimated and actual route costs, damaging the router's ability to distinguish between promising and unpromising partial paths. Factoring magnets into cost estimation could eliminate this problem.



(a) Fence (b) Magnet

Figure 1 - Hints. Two hint layers, *fence* and *magnet*, provide graphical control over the Irouter. The Irouter does not cross fence boundaries (a). Hence fences can designate routing regions, as in the route from x to X, or regions where routing is precluded, as in the route from y to Y. Magnets (b) pull routes in desired directions. In order to minimize the penalty areas, the Irouter will route as close to magnets as is possible without violating design rules or incurring uncompensated cost from other factors.

The use of fences and magnets together allows the user to easily specify the general path of a route. Typically fences are used to establish "channels", and magnets to define fill directions within the channels.

Parameters control the relative importance of the various cost factors used to choose between paths during routing. The primary cost factors are the costs per unit-length for each routing layer. Separate factors for horizontal and vertical segments permit the user to setup preferred routing directions for each layer. Other factors are a cost per jog, a cost per contact, and a penalty for resisting magnets. Routing widths and spacings, initially derived from the technology design rules, can also be varied interactively. Through cost, width, and spacing parameters, the user can greatly influence the nature of the routes created by the Irouter. Different parameter settings are appropriate to different situations. Parameter values can be saved and reloaded, enabling the user to load appropriate parameters from a library, whenever dictated by the routing situation.

3. Handling Design Rules

Many routers ensure design rule correct routes by limiting routing to a grid of tracks of sufficient pitch so violations can not arise between tracks. The Irouter uses a more flexible approach, which does not restrict routing to predefined tracks and readily handles obstacles from preexisting layout. In the Irouter, *blockage planes* are constructed in such a way that zero-width paths that do not impinge on blocked areas will be design rule correct when they are filled out to full width routes; see Figure 2. A separate blockage plane for each route layer and contact type identifies the locations where that layer may and may not be placed.

Blockage planes are constructed, prior to routing, by applying the design rules to the existing layout. Simple spacing rules are handled with blocks generated by growing the relevant layout features by the minimum spacing plus half the wire width (less 1 unit) in each direction. The blockage plane approach can be readily extended to more complex rules. In Magic, design rules more complex than simple spacings are

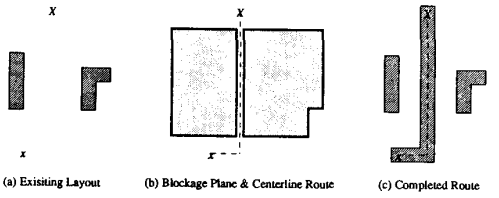


Figure 2 - Blockage Planes. The existing layout (a) is processed prior to routing to obtain blockage planes (b) for each route-layer and contact type. The blockage planes are constructed in such a way that any center-line route that does not impinge on blocked areas will yield a design-rule correct route when flushed out to full width. Blockage planes decouple design rule handling from the routing algorithm.

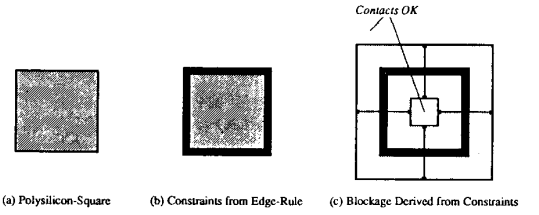


Figure 3 - Block Generation from Edge Rules. Constraints generated from edge rules, can be expanded to obtain appropriate blocks for complex rules. For example, an edge rule prohibiting inter-metal contacts on top of polysilicon edges, as in the MOSIS SCMOS process, generates constraints at polysilicon edges disallowing metal1-metal2 contacts (a and b). By expanding these constraints to account for contact width, appropriate blocks for the metal1-metal2 contact blockage plane are obtained (c).

expressed as edge rules [2]. Each edge rule defines an edge type by giving the layers present on the inside and outside, and specifies a constraint region, extending for a specified distance to the outside of the edge, where only certain layers are allowed. Edge rules excluding route layers from constraint areas can be handled by determining the constraint regions, as is done during design rule checking, and then adding them, suitably grown, to the blockage planes. For example, Figure 3 shows how blockage areas would be generated near the edges of a square of polysilicon for a technology that does not allow inter-metal contacts over polysilicon edges.

The blockage plane approach does have some shortcomings. It is not clear how to guard against rule violations related to *new* edges created during routing. For example the above rule disallowing inter-metal contacts over polysilicon edges implies restrictions on polysilicon routing that are not handled by the approach. (This particular rule can be satisfactorily approximated by requiring one-unit spacing to metal1-metal2 contacts when routing on polysilicon.) Another problem is that design rule violations, such as stacked contacts, can arise from self interactions in a route. It is too expensive to avoid this problem by continuously modifying the blockage planes in accordance with the layout of partial routes being considered; instead special case code is used to prevent common inter-route violations such as stacked contacts. Another problem with the current implementation of the blockage planes is that per-route incremental generation accounts for about 40% of the routing time. However, this overhead could be greatly reduced by preserving blockage information between routes, discarding it only in areas where the layout has been modified or when routing width is changed.

The blockage plane approach is working out well. With the extension described above (not yet fully implemented) more complex rules can be handled than in other approaches, and the rules that can't be dealt with appear to be unimportant in practice. The computational overhead, though large, has not precluded interactive performance, and seems well worth the added routing flexibility it provides.

4. Routing Algorithms and Performance

Maze routing was first described by Moore [3]. Moore-routing is done on a rectangular grid of cells, with some cells free and others blocked. The algorithm finds a shortest path between designated start and goal cells that does not pass through any blocked regions. The algorithm begins at the start cell, repeatedly expanding outward to neighboring free cells until the destination is reached. A backpointer is stored in each cell giving the location from which the cell was first reached, so that when the goal is reached the complete path can be traced back to the start point.

Maze routing is easily generalized. The notion of expanding to neighbors is not restricted to rectangular arrays, but works on any graph: the graph nodes correspond to cells, and the edges define neighbors. Thus maze routing is equally applicable to such diverse problems as many-layer printed circuit board routing, an almost 3-dimensional problem, and global routing of custom integrated circuits, where the "cells" are irregularly shaped and connected channels; for examples and a bibliography see Soukup's overview paper [4]. In addition both global and detailed maze routers have been built on top of Ousterhout's corner-stitched data representation [5, 6, 7].

Moore's algorithm is almost always augmented with a cost function [8], incorporating such factors as preferred routing directions and the cost of vias. Instead of expanding uniformly outward from the start point, only the least-cost partial path is expanded at each iteration. In place of the shortest path solutions of the unembellished Moore-algorithm, this provides more general "least-cost" solutions.

Cost-based maze routing is characterized by its range of application: appropriate choice of cells, neighbors, and cost-function readily adapts it to many diverse problems, by its thoroughness: it chooses between all possible routes not just a stylized subset, and by its slowness. Slowness is a consequence of the many cells that must be processed: a single Lee-route across a VLSI chip requires the examination of on the order of 100 million, minimum resolution wide, cells. The big challenge in the development of the Irouter was to reduce the number of cells that needed to be considered to a manageable number without sacrificing the power and flexibility of general maze routing. This challenge has been largely met; the Irouter finds optimal solutions to typical cross-chip routes after examining only approximately 1000 cells. The following subsections explain how this is possible.

4.1. Interesting Points

The first key performance optimization employed by the Irouter is expansion directly to the next *interesting point* where a change in direction or routing layer might be contemplated. This saves time by skipping over uninteresting pixels, and also by eliminating the need to process data on the cumbersome level of pixels at all: processing is done directly on Magic's feature oriented, corner-stitched, data representation [5].

A point is *interesting* because of alignment with the goal or hints, or because the available routing space shrinks or expands there; see Figure 4. Dual blockage planes are maintained for each routing layer, one sorted into maximal vertical strips and one into maximal horizontal strips. The next point where the available routing space changes can be found by simply noting the edge locations of the tile containing the current point, on the blockage plane organized in strips perpendicular to the routing direction; this is illustrated in Figure 5.

The performance advantage of the interesting point approach is most dramatic in situations involving few blockages. For example, a two-layer 100-unit long route with no obstacles required over 100,000 path expansions to Lee-route in the traditional manner, but only 6 using interesting points. On the other extreme, a short (60 micron) route through a dense area of a two-layer-metal CMOS design was measured. It required 36,807 path expansions to Lee-route in the traditional manner and 1,858 expansions using interesting points, still about a factor of 20 less.

4.2. Global Cost Estimation

Another performance optimization in the Irouter is the accurate estimation of the cost of completing partial paths. Recall that cost-based maze routers expand the least-cost partial path. Incorporating an estimated cost to completion into the cost of partial paths avoids the further expansion of paths that are headed in entirely the wrong direction, and speeds up routing immensely.

Previous systems have used cost estimates based on the distance to the goal [9]. This approach is illustrated in Figure 6. Unfortunately, such estimates are inaccurate and ineffective in situations involving large obstacles such as subcells; see Figure 6(b). Since almost all routes are made in the

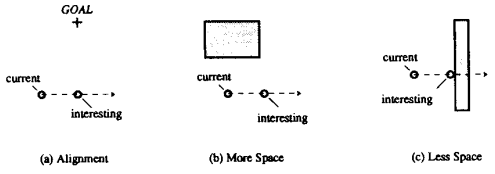


Figure 4 - Interesting Points. The Irouter saves time by identifying the next point, in each direction, where a jog or layer change might make sense, and proceeding directly to these *interesting points*. A point can be *interesting* because of alignment with hints or the goal (a), or because the amount of routing space in the perpendicular direction changes (b). A block (c) is a special case of change in the available routing space.

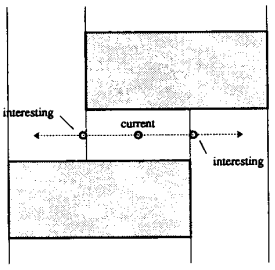


Figure 5 - Using Maximal Strips to Locate Changes in the Routing Space. Dual blockage planes are kept for each layer: one sorted into maximal vertical strips, the other into maximal horizontal strips. When routing horizontally (as above), the left and right edges of the current tile in the vertical strip representation determine where the routing space changes. The horizontal strip representation serves similarly for vertical routing.

presence of subcells and other large obstacles, the Irouter factors major obstacles into its cost estimates. This is made possible by a cost estimation plane containing only large scale obstacles (currently subcells and fences), constructed in a global prepass. Dijkstra's shortest path algorithm [10] is used to compute the cost to the goal from the corners of all tiles in the estimation plane, avoiding only obstacles in the estimation plane. Then, *estimators* are computed for each tile in the estimation plane, giving formulas for the estimated cost to the goal for points interior to the tile. Estimators are derived from the minimum horizontal and vertical routing costs within the tiles and the precomputed estimated cost from the tile corners to the goal, and are based on hypothetical paths from the interior of the tile to a given tile corner and then on to the goal; they have the form $Ax + By + C$.

Global cost estimation is effective. The estimation plane technique was compared with straight-line distance estimation on two routes taken from a semi-custom chip. One route

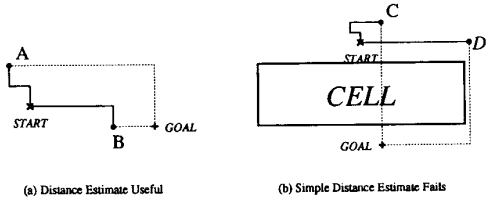


Figure 6 - Using Distance to the Goal. Incorporating distance to the goal into the cost of partial paths can speed up routing by discriminating between paths that are headed entirely in the wrong direction, such as **A**, and those that are more or less on the right track, such as **B**. However, in the presence of large obstacles, this approach is too simple, and actually degrades performance. The problem is that paths such as **C**, relatively close to the goal in straight line distance, are preferred to paths such as **D**, which are making the necessary jog around the obstacle. The Irouter solves this problem by considering major obstacles when estimating the cost-to-completion for partial paths.

required 10,830 partial path expansions using straight-line distance and 1,114 expansions using the global estimation plane. A second, larger route, required 63,045 expansions using straight-line distance, and only 2,431 using the estimation plane. The computational overhead for using the estimation plane accounts for about 10% of the total route time.

4.3. Goal Biased Searching

Minor obstacles, which are not factored into the global estimates, drive the actual cost of routes above the estimated cost. This causes the router to favor shorter paths that have not yet incurred the cost of unanticipated obstacles. In typical situations, involving several minor obstacles, this results in something approaching a parallel search of all alternative paths and a very long search time.

Such behavior can be avoided by biasing the search in favor of partial paths that are nearer to completion, i.e., by giving the estimated cost-to-completion greater weight than the actual cost of the part of the path already routed. This is very effective. A 10% bias on cost to completion reduces the number of path expansions in the above examples from 1,114 to 142 and from 2,431 to 235 respectively (recall that without the estimation plane these routes require 10,830 and 63,045 expansions, respectively!). The problem with this approach is that it makes the router insensitive to minor cost factors: paths that make early progress to the goal are preferred over those that minimize the number of contacts and jogs or follow hints more closely. What is needed is a middle ground between an unbiased search, which explores all alternatives to find a guaranteed minimal solution and the simple biased search described above, which chooses the first approximately acceptable path with essentially no exploration of alternatives.

The Irouter employs a shifting search window to explore alternatives at all stages of routing in a controlled fashion that won't explode into an exhaustive search. The window, which bounds the estimated cost-to-completion of the partial paths to be explored, begins centered on the start point, and is slowly shifted toward the goal as time goes on. Partial paths within the window are chosen between based on unbiased estimated total cost. Paths closer to the goal than the window are deferred until the window shifts to include them. Paths further from the goal than the window are penalized proportinately to their distance from the window, so that as they get further behind they must be significantly better than the paths within the window in order to be chosen for expansion. The idea is to consider alternative paths in an unbiased fashion, but still shift the focus towards the goal as time goes on.

The windowed search is implemented with three heaps. A maximum cost-to-go heap holds paths closer to the goal than the window, with the path farthest from the goal on the top. A minimum cost heap, holds the paths within the window sorted by unbiased cost. Finally a minimum adjusted-cost heap holds the paths behind the window. Although the penalties on the paths behind the window grow as the window is shifted forward, the ordering of the paths is not affected, thus it is sufficient to use a fixed standard window position to compute costs for paths on the adjusted-cost heap. Whenever the window position moves, the top elements of the maximum cost-to-go heap are shifted to the minimum cost heap until a path outside the window is encountered, and similarly elements from the minimum cost heap are shifted to the minimum adjusted-cost heap until a path still within the window is found. The next path to expand from is determined by correcting the cost of the top path on the adjusted-cost heap to take into account the current window position and comparing it with the cost of the path on top of the minimum cost heap: the least cost path is chosen for expansion.

In practice the three heaps are augmented by a *bloom stack* that gives priority to continued expansion of the last selected path up to a given cost increment. This results in the early exploration of straight line paths, an important heuristic in maze routing [11], that is necessary for windowed searching to be effective.

Windowed searching works. In a four-signal benchmark, derived from the control-signal routing for the SPUR FPU chip, windowed searching routed all four signals optimally in 13 seconds (on a 2 MIPS SUN 3/110), only 10% longer than to generate imperfect routes with a simple biased search. An unbiased unwindowed search did not complete in the several minutes that were allowed it.

5. Conclusion

The Irouter has been used on one VLSI chip, is currently being used on another, and will soon be released to a large number of users. It has demonstrated maze routing performance optimizations that yield interactive performance on VLSI chips, and hint mechanisms that permit the user to specify and obtain the routes he wants. With tools like the Irouter to complement the rapidly advancing fully-automatic routers, unassisted manual routing should soon be obsolete.

6. References

[1] J.K. Ousterhout, et al., "The Magic VLSI Layout System", *IEEE Design and Test of Computers*, Vol. 2, No. 1, Feb. 1985, pp. 19-30.

[2] G.S. Taylor and J.K. Ousterhout, "Magic's Incremental Design-Rule Checker", *Proc. of the 21st Design Automation Conf.*, June 1984, pp. 160-165.

[3] E.F. Moore, "Shortest Path Through a Maze", *Annals of the Computation Laboratory of Harvard University*, Harvard Univ. Press, Cambridge, Mass., Vol. 30, 1959, pp. 285-292.

[4] J. Soukup, "Circuit Layout", *Proc. of the IEEE*, Oct. 1981, Vol. 69, No. 10, pp. 1281-1304.

[5] J.K. Ousterhout, "Corner Stitching: A Data-Structuring Technique for VLSI Layout Tools", *IEEE Trans. on CAD/ICAS*, Jan. 1984,. Vol. CAD-3, No. 1.

[6] G.T. Hamachi and J.K. Ousterhout, "Magic's Obstacle-Avoiding Global Router", *Chapel Hill Conference on VLSI*, H. Fuchs, ed., Computer Science Press, 1985, pp. 145-164.

[7] A. Margarino, et al., "A Tile-Expansion Router", *IEEE Trans. on CAD/ICAS*, Vol. CAD-6, No. 4, pp. 507-517.

[8] C.Y. Lee, "An Algorithm for Path Connections and Its Applications", *IRE Trans. On Electronic. Computers*, Sept. 1961,

[9] F. Rubin, "The Lee Connection Algorithm", *IEEE Trans. on Computing*, 1974, Vol C-23, pp. 907-914.

[10] E.W. Dijkstra, "A Note on Two Problems in Connexion with Graphs.", *Numerische Mathematik*, Vol 1, 1959, pp. 269-271.

[11] J. Soukup, "Fast Maze Router", *Proc. of the 15th Design Automation Conference*, June 1978, pp. 100-102.