

An Implicit Connection Graph Maze Routing Algorithm for ECO Routing

Jason Cong, Jie Fang and Kei-Yong Khoo
Computer Science Department, UCLA
Los Angeles, CA 90095
{cong, jfang, khoo}@cs.ucla.edu

Abstract—ECO routing is a very important design capability in advanced IC, MCM and PCB designs when additional routings need to be made at the latter stage of the physical design. ECO is difficult in two aspects: first, there are a large number of existing interconnects which become obstacles in the region. A hierarchical approach is not applicable in this situation, and we need to search a large, congested region thoroughly. Second, advances in circuit designs require variable width and variable spacing on interconnects. Thus, a gridless routing algorithm is needed. In this paper, we propose to use an implicit representation of a non-uniform grid graph for gridless maze routing algorithm. A novel slit-tree plus interval-tree data structure is developed, combined with a cache structure, to support efficient queries into the connection graph. Our experiments show that this data structure is very small in memory usage while very fast in answering maze expansion related queries. This makes the framework very useful in the ECO type of routing.

I. Introduction

An engineering change order (ECO) is a request to make design changes, typically late in the design process. The ECO support has become very important in managing large designs because such ECO changes are almost always inevitable and updating the design with an incremental change not only saves considerable effort in realizing the design but also reduce the likelihood of introducing new errors into the design. The ECO problem involves the ability to (i) specify the incremental design changes, (ii) implement the design changes and (iii) update all design databases to reflect the changes. This paper deals with the implementation of the design changes in detailed routing.

The difficulty of ECO routing depends on how “far” away the problem occurs from the “original” framework on which the routing was created. Clearly, if ECOs occur before the layout is compacted, and if the routing environment (such as routing tracks) is preserved, the original router that created the layout can readily handle any additional routings, ECO routings included, without much problem. However, the routing problem becomes considerably more difficult when the layout has been compacted and transferred to a different database (such as mask layout) so that all or most of the routing environment has been lost. Such problems can occur in layout designs migrated from a different technology using

mask-geometry transformations, or hard IP-blocks imported into the current design.

The loss of original routing environment, coupled with layout compaction, means that the routing problem may no longer be simplified using predefined routing grids. In addition, the effects of interconnect resistance and line-to-line coupling are more pronounced in deep sub-micron designs which make it necessary to control the interconnect delay and signal integrity by imposing variable-width and variable-spacing constraints on critical or sensitive nets in high-performance designs [1]. Therefore, the ECO routing problem requires a truly gridless detailed router that is capable of handling large designs.

The key operation in detailed routing is to seek an optimal design-rule-correct path between two given points in the routing space. This can be done using either the point-based or the tile-based approach. In the point-based approach, the routing area is conceptually populated with *feasible points* where the center-line of a path can pass through. The feasible points and their neighborhood information can be abstracted as a *connection* graph so that the optimal path can be found using the shortest path (Dijkstra) algorithm. In the tile-based approach, the available routing area is partitioned into tiles [2], and the path is searched from a tile to another. *Searching* for a tile-to-tile path is usually fast [3], [4] due to the smaller number of tiles and the use of corner-stitching data structure. However, tiles are more complex to manage: a tile-to-tile path needs post-processing to obtain a design-rule-correct detailed route and there are some difficulties in using the tile-based approach for multi-layer routing with more complex design rules. Therefore, we use the more flexible point-based routing approach for the ECO routing.

A straightforward method to realize a gridless detailed router is to use the very dense, *manufacturing* grid that is determined by the resolution of the technology or the design database. Directly allocating memory to represent this dense grid is often infeasible due to its large size. Previous works in general-area, gridless point-to-point routing therefore often attack the problem by aggressively optimizing the underlying connection graph [5], [6], [7]. Unfortunately, such aggressive optimizations often result in a complex connection graph that cannot be implemented efficiently in practice for large designs. Therefore, they are not applicable to the ECO routing.

In this paper, we show that it is feasible to implement a large connection graph using an *implicit* representation. In Section II, we propose a regular connection graph that is much smaller than the manufacturing grid but remains sim-

*This work is partially supported by DARPA/ETO under Contract DAAL01-96-K-3600 managed by the US Army Research Laboratory.

ple to represent. The graph is represented implicitly to save memory. The implicit representation requires an efficient data structure to perform fast query operations; that is discussed in Section III. Finally, the effectiveness of our algorithms is validated with experimental results shown in Section IV.

II. Implicit Connection Graph

A. Simplified Connection Graph

The first problem that we need to address in using the point-to-point approach to realizing a gridless ECO-router is how to construct a connection graph that is simpler than the manufacturing grids and yet contains the optimal route if one exists. Many algorithms simplify the connection graph [8], [9], [7], [5] at the expense of very costly pre-construction and representation. Therefore, their usefulness is limited for large designs, as in the case of ECO routing.

We now introduce a connection graph called *Non-Uniform Grid Graph* G_S , based on the expansion of rectangular obstacles in the routing region according to wire/via width and spacing rules. In the routing region, the pre-existing routings and objects are obstacles that current routing path must avoid. These obstacles can be most conveniently defined as a set of possibly overlapping rectangles at different layers $R = \{r_1, r_2, \dots, r_{N_R}\}$. The layout design rules create an obstruction zone [10] around each obstacle where the centerlines of wires and center of vias cannot be placed. That is, the centerline of a wire of width w must be at least $dw_i = (w/2 + ws_i)$ away from the edges of the obstacle r_i , where ws_i is the wire spacing between the current net and the obstacle r_i . We let \tilde{R} be the set of rectangles in R that are expanded by dw_i in each of the four rectilinear directions, as shown in Fig. 1(a). Please note that ws_i may not be the minimum wire to wire spacing and may vary from net to net due to aggressive optimizations in high-performance designs. Similarly, we can create the set of rectangles expanded according to via width and spacing rules, denoted as \tilde{R}^v . Our underlying routing graph is defined as follows:

Definition: Given a multi-layer routing problem with the obstacle set R , a source s and a sink t . A *Non-Uniform Grid Graph* G_S is a orthogonal grid graph where its x grid locations are the vertical boundary locations of \tilde{R} and \tilde{R}^v plus the x locations of s and t . Similarly, we can define the y grid locations, as shown in Fig. 1(b).

G_S is a strong connection graph; that is, it guarantees to contain a shortest path from s to t if any such connection exists among the set of obstacles R , according to design rules. It is easy to show that a strong connection graph, G_C , proposed by Zheng, et al. [7], using the same obstacle expansion strategy, is a sub-graph of our G_S , as shown in Fig. 1(c).

Compared to uniform grid graphs, where an ECO type of routing may generate very dense grids (as shown in

Fig. 1(d)) our non-uniform grid graph is much sparser. Comparing to previous non-uniform graphs, although our graph has more nodes, the gridded nature of G_S makes it very easy to come up with an implicit representation which is both highly compressed in storage and efficient in query.

B. Implicit Representation of Connection Graph

Instead of pre-computing the graph explicitly, we use an implicit representation of the connection graph. That is, we compute the graph nodes on-the-fly. The computation of a graph node, a *query*, consists of two steps: first, compute the possible position of the node, and second, determine the feasibility of the node.

Given the position of point p and a direction d , we need to find the position of the closest neighbor to p in the direction d quickly in order to support the implicit representation efficiently. Since our connection graph consists of non-uniform grids, we build an array X_S of sorted x -coordinates of the vertical grid-lines in G_S . If the x -coordinate of the current point corresponds to $X_S[i]$, the x -coordinate of the neighboring point is either $X_S[i+1]$ or $X_S[i-1]$, which can be found in constant time. The preprocessing time to compute such an array requires a $O(|R|)$ time for linear scanning of all rectangles and $O(|R|\log|R|)$ time for sorting. We need a similar array for computing y -coordinates. So the data structures to support implicit representation of non-uniform grid connection graphs are simply two arrays with a total memory requirement of $O(|G_h| + |G_v|)$ where $|G_h|$ and $|G_v|$ are the number of horizontal and vertical grid-lines in G_S , respectively.

A point is feasible for placing a wire or via if it is not enclosed by the applicable expanded rectangles in \tilde{R} or \tilde{R}^v . Therefore, finding the feasibility of a point requires a point enclosure query into the set of expanded rectangles:

Point enclosure problem: Given a set of rectangles $R = \{r_i \mid i = 1, 2, \dots, N_R\}$ and a point v , determine the set of rectangles that contain v .

We will discuss our data structure to support feasibility computation in the next section.

III. Query Data Structure

The feasibility check answers the simple question of whether a point falls into any expanded rectangle. However, the nature of ECO type of routing make this problem not trivial to solve. First, the data structure needs to represent a fairly large and congested region which contains a *huge* amount of rectangular objects. Second, the rectangular objects need to be expanded according to the width and spacing rules. However, ECO routing normally involves only a few nets. So the preprocessing time for the query data structure should not be long. Third, the query may be made many, many times during the routing, so it has to be answered *very* efficiently.

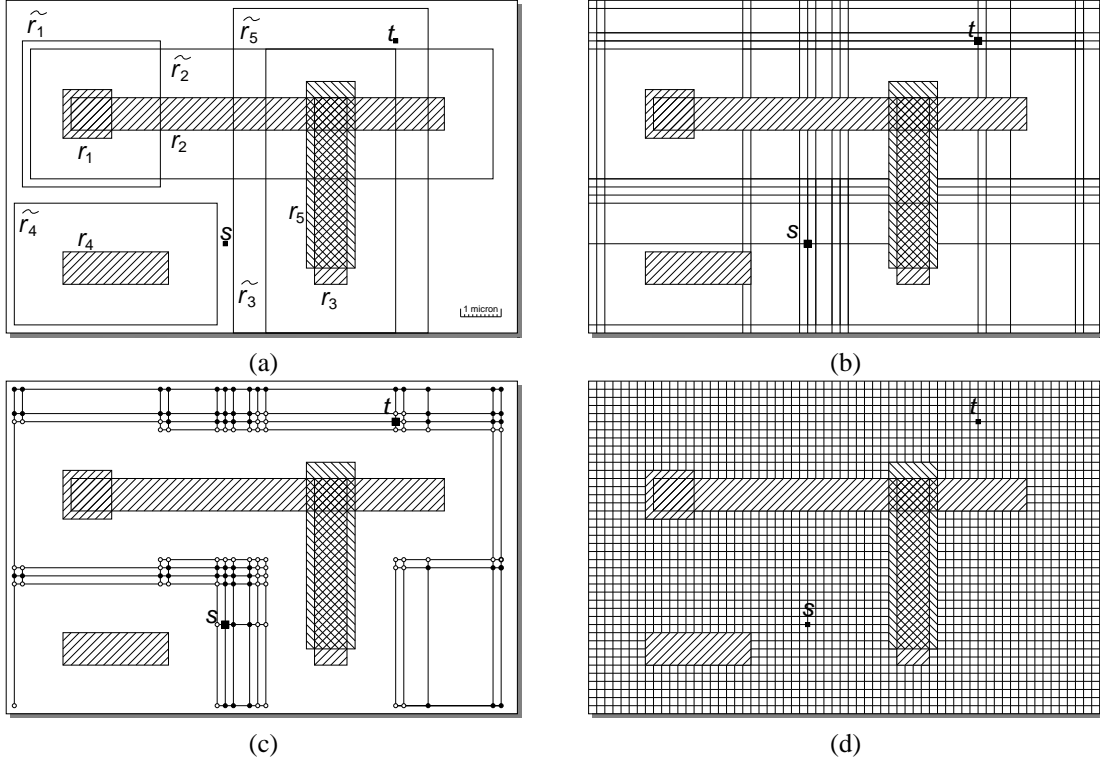


Fig. 1. Connection Graph Generation: (a) obstacle rectangles R and expanded rectangles \tilde{R} , (b) G_S constructed by x and y locations of \tilde{R} , \tilde{R}^v , s and t , (c) G_C constructed by boundaries and extension lines of \tilde{R} and \tilde{R}^v , (d) uniform grids which use very dense manufacturing grids.

The point enclosure problem is well studied in computational geometry, and it can be solved using segment trees in $O(\log n + k)$ time and $O(n \log n)$ space [11], or solved using priority search [12] trees in $O(\log^2 n + k)$ time and $O(n)$ space where k is the number of rectangles that enclose the point v . These tree-based algorithms, although good for static data, suffer from long preprocessing time (at least $n \log n$) due to dynamic updates, including expansion, insertion and deletion. More practical data structures have been proposed based on organizing the objects into one-dimensional buckets [13], [14], two-dimensional buckets [15], or two-dimensional data-oriented buckets called field blocks [16]. Extensive comparisons of the tree-based approach versus the two-dimensional buckets approach can be found in [15]. In preferred layer routing, the obstacles in a given layer are dominated by long rectangles in the preferred routing direction. This favors the one-dimensional buckets approach, such as the “slit-tree” in [13], [14] that recursively bisects the layer into slices in the preferred direction, and rectangles intersecting or overlapping a common slice are managed by a bidirectional linked list. The advantage of applying the slit-tree algorithm is that it requires linear memory space and linear preprocessing time while, in practice, the query takes constant time. However, applying the simple one-dimensional bucketing data structure to non-uniform

grid graph query in ECO routing has three drawbacks. First, there is still a very large number of rectangles in each slice. Second, each rectangle is expanded many time because the width and spacing rules may differ between wires and vias, from layer to layer and from net to net. Last, it does not exploit the locality of maze routing queries. In the remainder part of this section, we propose several techniques to improve the basic slit-tree algorithm.

The first drawback of a simple slit-tree structure is that it slows down when the number of objects is large in each slice. Although by further bisection we can reduce the number of objects in each slice, the number of “small” objects such as vias and short local wires can not be effectively reduced. Therefore, we propose a two-level data structure to solve this problem. The first level is a fixed height “slit-tree” and the second level is an auxiliary interval tree [17]. Notice that the interval-tree has predetermined, uniformly spaced cut-lines according to the size of the slice. The advantage of the interval-tree is that long rectangular objects along the preferred direction in each slice are stored at the highest level of interval node they cut while short objects which fall in between interval lines are stored at leaf nodes. We call them *cells*, as illustrated in Fig. 2. The storage space for such a data structure is still linear, while the number of rectangles a query needs to check can be significantly reduced by travers-

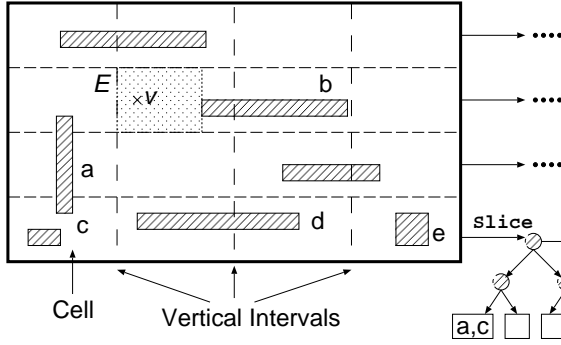


Fig. 2. Slit + Interval Tree: The horizontal slices are cut into cells by vertical intervals. Rectangular obstacles are stored on either cut-line nodes or leaf cells. The empty rectangle E is generated as a result of a query to point v .

ing the interval tree nodes top-down. Another advantage of this data structure is that its preprocessing time is still constant in practice.

The second problem is due to the expansion of rectangular objects. The algorithm requires expanding all the rectangles according to wire rule and via rule before checking for point enclosure. This is undesirable when the design rules vary frequently from net to net, since each set of design rule requires a new set of expanded rectangles. The result is multiple copies of expanded rectangles (wire rule and via rule) and frequent rebuilding of data structures (design rule changes). To solve this problem, we propose to store unexpanded rectangles R in the query data structure. Since the query involves a local search around the area of the query point, we can search for all the rectangles that are within the largest expansion distance ($\max_i dw_i$ for wire-feasibility or $\max_i dv_i$ for via-feasibility) to the query point and expand these rectangles on-the-fly. By paying the price of a slightly larger searching area, assuming the difference of width and spacing rules are much smaller than the size of “cells,” we are able to eliminate the need to pre-expand the rectangles, and thus allow easy implementation of flexible design rules.

Last, existing query data structures do not exploit locality of point enclosure queries due to the point-to-point expansion nature in maze routing. Each query into the data structure, although best optimized for trade-offs between the storage space and the speed of query, still requires multiple levels of tree traversal and a linear scan of each object in the cells. This is a very expensive operation because queries need to be made repeatedly in maze routing. In our implementation, we improve the query efficiency by exploiting the locality of the queries using the *cache* data structure, independent of the query data structure. The basic operation of maze routing is to expand node by node. So its queries have very good locality—the queries propagate from the source node location, and each time go to a neighboring location not far away. We can exploit this locality using *cache*—a small fixed-size vector of rectangles from recent query re-

TABLE I
ECO TEST EXAMPLES

Ex.	Block Dimension $x \times y (\mu\text{m})$	Layers	Pins	Cells	Rectangles
eco-1	1372.5 \times 1593.3	3	3	6417	232,309
eco-2	1372.5 \times 1593.3	3	2	6417	232,453
eco-3	1372.5 \times 1593.3	3	5	6417	232,004
eco-4	1372.5 \times 1608.6	3	2	6417	232,633
eco-5	1556.5 \times 1676.8	3	2	7542	196,947
eco-6	2926.1 \times 1676.8	3	2	13959	372,240
eco-7	2741.5 \times 3192.8	3	2	25668	701,172

sults. We keep two caches in our implementation: an obstacle cache and a “free” cache. If the query point v is *not* enclosed by a rectangle, then we compute a maximal “empty” rectangle around v , shown as E in Fig. 2. Notice that computing the *largest* empty rectangle is a *NP-hard* problem, so we use a simple heuristic: at the beginning, the empty rectangle R_e is set to be the same size of the cell(s) containing the query point. While the rectangular obstacles in the cell(s) are checked one-by-one, at each time we compute the maximal remaining empty rectangle with respect to the current obstacle R_k . This involves examination of all possible derivation (up to four) of new empty rectangle according to relative position of R_e and R_k . Such an operation takes constant time. Since we need to go through the rectangles to check for overlapping, we are able to find the empty rectangle with little extra effort. If the query point p is enclosed by an expanded rectangle, then the expanded rectangle is added to the obstacle cache. The rectangles in either cache are sorted according to the time they are generated and when the cache is full, the rectangle with “oldest” time stamp is swapped out. Our experiments show that adding in these two caches gives us $11\times$ reduction on average query time for our routing examples.

IV. Experimental Results

We have implemented our implicit connection graph and query data structures for ECO routing. A standard maze algorithm is used to search for the connection on the implicit graph. Several standard cell blocks with variable width and variable spacing design rules are used for ECO examples: route one random multi-terminal net. They were placed and routed by commercial tools and compacted. Only geometry information is passed to our router to search for the routes. Table I shows a summary of the examples used here. The results presented in this section were collected on a 168MHz Sun Ultra-1 workstation with 128MB of memory. We also compare our algorithm with *Iroute*, a tile-based interactive router in Magic layout systems [18], [19].

Table II shows a comparison of memory usage between explicit representation and implicit representation. Please note that the estimation of uniform grid uses the common

TABLE II
MEMORY USAGE OF DIFFERENT CONNECTION GRAPH (MB)

Ex.	Uniform Grids Explicit	Non-Uniform Grids Implicit	Iroute
eco-1	160.2	10.9	32.7
eco-2	160.2	10.9	32.7
eco-3	160.2	7.2	32.6
eco-4	161.7	10.9	32.6
eco-5	191.0	12.7	35.2
eco-6	359.4	15.9	52.6
eco-7	641.1	43.6	84.7
Ratio	14.3	1.00	3.0

TABLE III
ECO TEST RESULTS

Ex.	Non-Uniform Grid		Iroute	
	Runtime (sec.)	Wire/Via	Runtime (sec.)	Wire/Via
eco-1	19.1	17374/ 66	42.15	22629/89
eco-2	6.3	11332/ 42	26.58	13162/62
eco-3	34.5	34736/110	68.70	36593/103
eco-4	24.0	21760/122	57.39	24385/153
eco-5	12.3	27061/54	43.14	34543/12
eco-6	24.7	37858/70	74.29	56423/20
eco-7	38.2	35690/74	79.79	47591/20

divisor of wire/via width as the uniform grid distance. In these set of examples, it is $0.1\mu\text{m}$. The estimation of explicit memory usage assumes *minimum* memory requirement per grid. We use 2 bits per grid node, which is barely enough to distinguish wire/via obstacles and empty spaces, as suggested by [20]. Our results suggest that by using implicit representation, the average memory size is reduced by $14\times$ among seven of our test cases.

We also compare the run-time and routing quality with Iroute in Table III. Our experiments show that at a comparable routing quality, our algorithm uses 2 to $3\times$ less memory and get results 2 to $4\times$ faster than Iroute. This improvement over Iroute is significant, as tile-based algorithms are known for their memory and runtime efficiency because they store and search tile (area) instead of grids.

V. Conclusions

In this paper we have shown that ECO routing is difficult in that it needs to search large and congested routing regions under the variable width and variable spacing design rules imposed in advanced technologies. We proposed a non-uniform grid graph with its implicit representation. A novel slit-tree plus interval-tree data structure is developed, combined with a cache structure, to support efficient queries into the implicit graph. Our experiments compare our graph with an explicit uniform grid approach and Iroute, a well-known tile-based router for gridless routing. The results show that

not only this graph representation is very efficient in memory usage— $14\times$ smaller than explicit representation and 2 to $3\times$ smaller than Iroute, but the queries into the data structure are also very fast. The run-time of our maze routing algorithm is 2 to $4\times$ faster than Iroute. This makes the whole framework very suitable for the ECO type of routing.

References

- [1] J. Cong, C.-K. Koh, and P. Madden, "Performance optimization of vlsi interconnect layout," *Integration, the VLSI Journal*, vol. 21, pp. 1–94, 1996.
- [2] J. Ousterhout, "Corner stitching: a data-structuring technique for VLSI layout tools," *IEEE Trans. Computer-Aided Design*, vol. CAD-3, no. 1, pp. 87–100, 1984.
- [3] M. Sato, J. Sakanaka, and T. Ohtsuki, "A fast line-search method based on a tile plane," in *Int. Conf Circuits Systems*, pp. 588–591, 1987.
- [4] A. Margarino, A. Romano, A. D. Gloria, F. Curatelli, and P. Antognetti, "A tile-expansion router," *IEEE Trans. Computer-Aided Design*, vol. CAD-6, no. 4, pp. 507–517, 1987.
- [5] Y. Wu, P. Widmayer, M. Schlag, and C. Wong, "Rectilinear shortest paths and minimum spanning trees in the presence of rectilinear obstacles," *IEEE Trans. Computers*, vol. C-36, no. 1, pp. 321–311, 1987.
- [6] T. Ohtsuki, "Gridless routers—new wire routing algorithms based on computational geometry," in *Int. Conf Circuits Systems*, 1985.
- [7] S. Zheng, J. S. Lim, and S. Iyengar, "Finding obstacle-avoiding shortest paths using implicit connection graphs," *IEEE Trans. Computer-Aided Design*, vol. 15, no. 1, pp. 103–110, Jan. 1996.
- [8] J. Cohoon and D. Richards, "Optimal two-terminal $\alpha = \beta$ wire routing," *Integration, The VLSI J.*, vol. 6, no. 1, pp. 35–57, May 1988.
- [9] J. Jaja and S. Wu, "On routing two-terminal nets in the presence of obstacles," *IEEE Trans. Computer-Aided Design*, vol. 8, no. 5, pp. 563–570, May 1989.
- [10] W. Schiele, T. Kruger, K. Just, and F. Kirsch, "A gridless router for industrial design rules," in *27th ACM/IEEE Design Automation Conf.*, 24–28 June 1990.
- [11] V. Vaishnavi, "Computing point enclosures," *IEEE Trans. on Computers*, vol. C-31, no. 1, pp. 22–29, Jan. 1982.
- [12] E. McCreight, "Priority search trees," *SIAM J. Comp.*, vol. 14, no. 2, pp. 257–276, May 1985.
- [13] I. Kato, S. Ohhira, and Y. Hisatomi, "A method of pattern data management of PWB layout system," in *Proc. 35th Ann. Convention IPS Japan*, pp. 2429–2430, 1987.
- [14] E. Kuh and T. Ohtsuki, "Recent advances in VLSI layout," *Proc. IEEE*, vol. 78, no. 2, pp. 237–263, Feb. 1990.
- [15] M. Edahiro, K. Tanaka, T. Hoshino, *et al.*, "A bucketing algorithm for the orthogonal segment intersection search problem and its practical efficiency," *Algorithmica*, vol. 4, no. 1, pp. 61–76, 1989.
- [16] G. Suzuki and N. Hamada, "Practical data structure for incremental design rule checking and compaction," in *Proc. IEEE Int. Conf. Comp. Design*, pp. 442–447, 5–8 Oct. 1987.
- [17] H. Edelsbrunner, "A new approach to rectangle intersections," *Intl. Journal of Computer Mathematics*, no. 1, pp. 209–29, 1983.
- [18] M. Arnold and W. Scott, "An interactive maze router with hints," *Proc. 25th Design Automation Conference*, no. 1, pp. 672–76, 1988.
- [19] J. Ousterhout, G. Hamachi, R. Mayo, W. Scott, and G. Taylor, "Magic: a vlsi layout system," *Proc. 21st Design Automaton Conference*, no. 1, pp. 152–59, 1984.
- [20] J. Soukup, "Maze router without a grid map," in *IEEE/ACM Int. Conf. Computer-Aided Design Dig. Tech. Papers*, pp. 382–385, 8–12 Nov. 1992.