A Survey of Path Search Algorithms for VLSI Detailed Routing

Stèphano M. M. Gonçalves, Leomar S. da Rosa Jr., Felipe de S. Marques
Technology Development Center
Federal University of Pelotas
Pelotas, Brazil
{smmgoncalves, leomarjr, felipem}@inf.ufpel.edu.br

***Abstract*—The path search problem is very common in computing. One of its applications is in the VLSI routing. Since most of the path search algorithms used today are old, their evaluation took place in out of date scenarios, such as small 2D grid graphs. Thus, this work presents some path search algorithms, as well as an experiment comparing them in a scenario similar to detailed routing of integrated circuits. The presented algorithms are Soukup's, A*, LCS* and Hetzel's. We used ISPD 2008 benchmarks and FastRoute4 in order to create the input for the detailed routing experiment. The results show that Hetzel's algorithm is faster than the compared algorithms, and since it is optimal, it is a better choice for a path search algorithm in detailed routing.**

Keywords—path search; detailed routing; VLSI

I. INTRODUCTION

The task of building VLSI circuits consists in a large and complex procedure, containing many steps. One of these steps is called routing. The goal of routing is to find the routes of all wires that interconnect the circuit components, while minimizing the total wire length, power and delay. The routing is yet subdivided into 2 steps: global and detailed routing. While global routing aims to provide an orientation on where the wires should be placed, detailed routing determines the exact location of wires. In detailed routing, path search algorithms are commonly used to find the wire routes.

The path search problem can be generically defined by finding a path between two sets of points *S* and *T* (source and target sets) in a graph. Some algorithms can handle multiple source-target (s-t) points, while others can handle only a single s-t pair. In addition, the path cost is commonly desired to be minimized. In VLSI routing, the graph is either a 2D grid for some global routing methodologies or a 3D grid for other global routing approaches and for detailed routing.

There are two classes of path search algorithms in VLSI routing: the maze and the line probe algorithms. The maze algorithms work by expanding the search point by point, while the line probe algorithms use line segments. Usually, the maze algorithms guarantee the optimal path, but they are slow, while the line probe algorithms are usually fast but don't guarantee the optimal path. The first maze algorithm was Lee's algorithm [1]. Later, Rubin [2] used the A* search [3] technique to improve Lee's algorithm, making the search converge more quickly to the target point. Soukup [4] also proposed an algorithm that combined both breadth first search (BFS) and depth first search (DFS), aiming to improve the performance of Lee's algorithm.

The first line probe algorithm was proposed by Mikami and Tabuchi [5]. Later, Hightower [6] proposed a modification on this algorithm. Finally, Hetzel [7] united the advantages of both classes of algorithms, proposing a line probe algorithm based on the A* search. The algorithm was shown to be faster than A* in the experiments held in [7], and it is optimal, unlike [5] and [6]. Later, the algorithm was generalized in [8] and [9] to handle more generic scenarios.

Among the maze algorithms there is also the subclass of the bidirectional search. Basically, these algorithms perform two simultaneous searches, one starting from the source and the other starting from the target. Although this approach tends to result in a lesser search space, there are several drawbacks that compensate the search space reduction. There were several propositions of bidirectional algorithms [10][11][12] that tried to beat the A* search. However, A* was still better on average. Johann [13] proposed the LCS*, which is an improvement over BS* [12] that uses the information of both search fronts to improve the heuristic power. The experiments in [13] showed that LCS* was slightly faster than A* on average.

Although all these path search algorithms have been evaluated in some way, there was never a comparison of them using the same benchmarks. Moreover, since they are old, their evaluation involved the needs of their era, which are greatly different of today's needs. For example, LCS* was compared against A* in a 200x200 2D grid, and Hetzel compared his algorithm to A* using ISPD98 benchmarks, which are very small in comparison to the current benchmarks. Thus, in order to identify a good path search algorithm for detailed routing, it is indispensable the realization of an experiment that gathers the path search algorithms with more potential and evaluates them under the same benchmarks, according to the current needs, which consists in using global routing to restrict detailed routing

978-1-4673-6853-7/17/$31.00 ©2017 IEEE

search space in *huge tridimensional grid graphs*. The detailed routing search space reduction may interfere in their behavior, as well as the grid size, presenting different results of path searches performed in a different scope. Considering that no published work has ever performed such experiment, the goal of this work is to present it, using the ISPD2008 benchmarks [14] and FastRoute4 [15] as the global router.

II. Path Search Algorithms

This section presents the algorithms chosen for comparison. Concerning maze algorithms, the A* was chosen for been optimal and fast, and Soukup's algorithm was chosen for its speed. Of the line probe algorithms, Hetzel's algorithm was chosen for been optimal and for handling 3D grids, unlike [5] and [6]. Regarding the bidirectional algorithms, LCS* was chosen, since it achieved better results than A* in the experiments held in [13].

A. Soukup' Algorithm

Soukup's algorithm is a modification over [1]. Lee's algorithm starts the search in the source point, expanding the search to the neighboring nodes. These nodes form the frontier of the search, called wavefront. Then, each node in the wavefront is expanded, that is, its neighbors are added into the next wavefront. This BFS behavior, show in Fig. 1(a), makes the search expand in all directions, until the target point is reached.

Soukup's algorithm uses both BFS and DFS (Fig. 1(b)). When a node n is expanded, if a neighbor n' is closer to the target point than n, the algorithm starts a DFS in n' direction. All nodes generated in the DFS are put into the current wavefront. The neighbors of n that turn away from the target are added in the next wavefront. According to [4], Soukup's algorithm is 10-50 times faster than Lee's algorithm. It guarantees that a path is found if there is a feasible one, but it does not guarantee that the path found is optimal. The algorithm handles single source-target points.

B. A*

Lee's algorithm does not have any criterion to select nodes for expansion. It simply expands all nodes in the current wavefront, generating another wavefront. This behavior makes the search expand in all directions. However, this is very inefficient, since it may result in unnecessary effort. In order to avoid this behavior, the A* search can be used to direct the search towards the target point. This is illustrated in Fig. 1(c).

In the A* algorithm, each node n stores the cost of the known path from n to the source point, and an estimated cost from the node itself to the target point. These costs are denoted respectively by g(n) and h(n). The sum of this costs is denoted by f(n), and it represent n's potential cost. In order to direct the search to the target point, the nodes with the lower f(n) are expanded first. These nodes belong to the *open set*, and they are called open nodes, since they are available for expansion. The open set is analog to Lee's wavefront, differing that the nodes are ordered by f(n). When a node is expanded, it is removed from the open set and put into the *closed set*. The closed set holds the nodes already expanded, which are called closed nodes. The nodes generated by an expansion are only put into the open set if there is no closed node sharing the same point, or if there is no other open node of the same point with a better (or equal) cost. The procedure stops when the node of the target point is selected or when the open set becomes empty.

The A* algorithm guarantees that a path is found, if one exists, and it is optimal. The algorithm can handle multi sources and targets. In a grid with uniform cell costs, like in Fig. 1, this algorithm may behave similarly to Soukup's algorithm.

C. LCS*

The LCS* [13] algorithm, illustrated in Fig. 1(d), is an improvement over BS* [12], which is an improvement over BHPA [11]. The BHPA algorithm basically consists in two A* searches, one starting from the source point and the other starting from the target point. There is also a variable, called a_{min} that stores the cost of the best path that passes through a node closed in both searches. These nodes are called *meeting nodes*, since they represent the intersection of both searches. Note that in bidirectional search, when a meeting node is found, it does not mean that the best path has been found. The search ends when a_{min} is lesser or equals to any node's cost in both open sets. The path is obtained by the meeting node corresponding to a_{min}. The BS* algorithm introduces four new concepts: *nipping*, *pruning*, *trimming* and *screening*. When a node is selected for expansion, if it is closed in the opposed search, it is not necessary to expand it, since it does not hold a path that is better than any other already found. In this case,

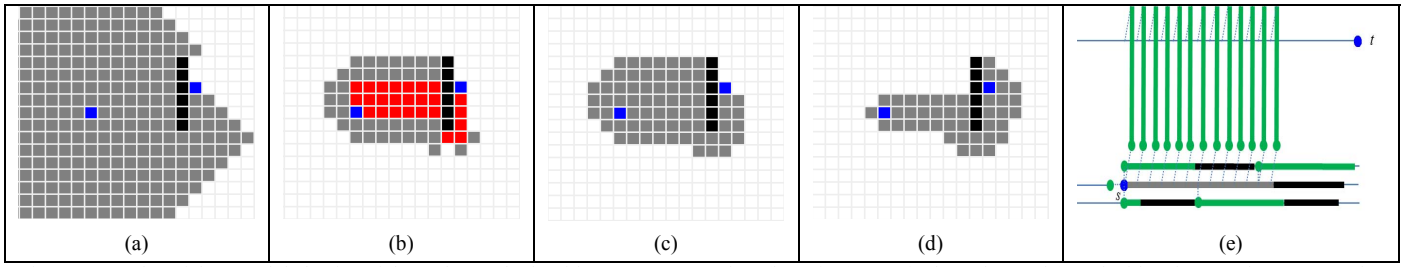

Fig. 1. Examples of the search behavior of the path search algorithms: Lee (a), Soukup (b), A* (c), LCS* (d), and Hetzel (e). The blue dots are the source and target points (from left to right, respectively). Grey represents the visited search space. Black represents obstacles. In (b), red denotes the DFS. (e) illustrates the expansion of the grey (closed) interval, generating the green (open) ones.

the node is closed and it is not expanded. This is called *nipping*. A nipped node in the opposed search may have successors in the open set that are unnecessary for the search. The removal of such nodes from the open set is called *pruning*. When a meeting node is found, it is possible that there are many nodes in the open sets that have a cost greater or equals to meeting node's cost. Since these nodes cannot lead the algorithm to find a path better than the meeting node's path, they can be removed from the search. This process is called *trimming*. *Screening* is a preemptive trimming. It is the act of do not add, in the open sets, the nodes that have costs greater than or equals to meeting node's cost.

The mechanics of the LCS* algorithm are the same of BS*, and the four introduced concepts in BS* are kept. The novelty of the algorithm is in the fact that it uses the information of the opposed search in order to improve the heuristic functions, making the algorithm terminate earlier. This is accomplished by having two variables, *Py* and Ω, for each search front. *Py* is the cost of the node of least cost in the bounds of the closed set (the parents of the open nodes). The variable Ω is the least penalty to be added to the cost of an estimation from a node outside the closed set, targeting the origin node of the closed set search tree. The node costs in the open sets are given just as the A* algorithm. These variables are used to make the algorithm stop early or to avoid unnecessary expansions. The LCS* guarantees to find the optimal path and handle single source-target points.

D. Hetzel's Algorithm

This algorithm has the same basic mechanic of A*. The main difference is that instead of using nodes associated with points, it uses line segments (or intervals). Thus, it can be viewed as an A* search on intervals. Fig. 1(e) illustrates an interval expansion.

In the A* algorithm, it is very common that a row of nodes in the same line share the same cost. Therefore, these nodes can be merged in a single interval that is labeled with the node's cost. In a generic graph, the only way to create such interval is to traverse a set of nodes. In this case, this interval labeling is not worthy. However, in a grid graph it is possible to know such intervals with a simple $O(1)$ geometric calculation. Thereby, Hetzel's algorithm is supposed to be applied only in grid graphs, which is the case of the detailed routing. Note that this algorithm does not store the graph explicitly. Instead, it stores sets of intervals used in the search. The grid obstacles are also stored as a set of line segments.

Hetzel's algorithm, proposed in [7], had some limitations, and thus, it was generalized in [8] and [9]. The resulting algorithm, called *GeneralizedDijkstra*, did not improve the algorithm speed in any way, but made it able to work not only on intervals, but also in generic sets of nodes in general graphs. Note that by mentioning "Hetzel's algorithm", we refer to GeneralizedDijkstra *applied on intervals*, which the pseudocode is presented in [16]. This algorithm is optimal, and supports multi source-target points.

III. Experiments

A. Methodology

The goal of the experiments is to evaluate the speed and the path quality of the presented algorithms in a detailed routing scenario. Since the comparison regards path search algorithms, we consider only the detailed routing peculiarities that are relevant for the comparison. The main detailed routing restriction is tunneling, since it drastically reduces the search space. Also, we consider specific costs for vias and jogs. We do not consider any other design rules, since they should be threated outside the core of the search algorithm (due to complexity and performance reasons). Thus, this should have little or no impact in the algorithmic comparison. Rip-up and Reroute is also not considered.

The experiments were run on the ISPD 2008 benchmarks [14], in a Linux machine with 132Gb RAM, CPU AMD Opteron 2.2 Ghz. The algorithms were implemented in Java. The experimental methodology was as follows. First, the FastRoute4 [15] global routing tool was run for each benchmark. The FastRoute4 output is a set of 3D rectilinear Steiner trees, each one representing the global net routes. The second step represents the detailed routing stage. Each net of a given benchmark was decomposed in two pin nets. For each two pin net, Hetzel's algorithm was run on the Steiner tree tunnels in order to find the route that interconnects the two pins. Notice that until now, the routes are in global routing coordinates (G-cells). This route connecting both pins is a tunnel, and it is used to restrict the search area of the path

TABLE I. Experimental Results

Benchmark	Grid Size	# Searches	Time (hours)				WL Over (%)
			Hetzel	A*	Soukup	LCS*	
adaptec1	11340 x 11340	674,203	1.6	24.9	14.2	18.1	1.21
adaptec2	14840 x 14840	741,761	1.8	15.9	15.2	11.9	0.98
adaptec3	23220 x 23220	1,298,090	3.1	71.9	54.5	37.6	0.57
adaptec4	23220 x 23370	1,270,890	2.6	60.1	71.6	34.6	0.50
bigblue1	11350 x 11350	762,031	3.8	99.2	78.4	67.3	0.69
bigblue2	18720 x 18840	1,365,883	4.6	63.6	58.3	48.3	3.02
newblue1	11970 x 11970	822,002	1.1	5.6	2.9	4.9	1.96
newblue2	27850 x 23150	1,199,989	4.3	58.4	25.3	31.3	1.09
newblue3	38920 x 50240	1,230,489	4.5	182.7	80.2	53	0.69

search algorithm used to find the actual route between the pins. Then, the chosen path search algorithm is run, and some search statistics are obtained. After that, a pre-computed path (see ahead) for this search is marked on the grid data structure. This procedure was performed for each search algorithm. The first execution was using Hetzel's algorithm. The resulting paths were added in the grid, as well as stored in disk. For the other three algorithms, the Hetzel's algorithm paths were loaded in memory and were used as the "pre-computed paths" mentioned before. Thus, for each search of each algorithm, they were all executed on the exact same grid. Since LCS* and Soukup's algorithms do not mention multi source-target treatment, single source-target was used.

The use of tunnels imposes a new lower bound in a path cost, because a tunnel usually has many detours. Therefore, there is no reason to not use the tunnel information to improve the heuristic function of the algorithms that use it (A*, LCS* and Hetzel's). When the tunnel is obtained, a simple preprocessing is done in the tunnel. When running, the algorithms use the stored information to calculate the *h* function. Hetzel's algorithm also uses the tunnel information to direct the expanded segments to the right way. Soukup's algorithm also uses this information to guide the depth first search.

B. Results

The Table 1 presents the results. Since all algorithms were run on the same grid, the total path cost of the optimal algorithms were the same. Thus, Table 1 do not show such information. Instead, the column "WL Over" shows the increment of Soukup's total path cost relative to the optimal. The "# Searches" column represents the total number of path searches (two pin nets). The "Grid Size" column represents the horizontal and vertical dimensions of the grid. All grids have 6 layers.

In almost all benchmarks, Soukup was faster than A. The DFS's tend to make the algorithm expand less nodes, at the cost of path quality. However, A* may be faster when avoiding a high cost area (e.g. an area only reached by jogs) that does not lead to any path, while Soukup exhaustively explores the area with DFS's. LCS* did better than A*, but A* can handle multiple sources and targets, which is a relevant factor in detailed routing, since the pins occupy multiple grid points. Finally, Hetzel's algorithm was very faster than LCS*, presenting an average speedup of 10.

IV. Conclusions and Future Works

This work presented a survey of some path search algorithms that can be applied in VLSI detailed routing. The algorithms were evaluated in a similar detailed routing scenario. Hetzel's algorithm was very faster than the other algorithms, presenting an average speedup of 10 with reference to LCS*, which was the second faster algorithm. At first, considering this algorithm is faster, optimal and can handle multiple sources and targets, there is no motive not to use it as a pathfinder in detailed routing. However, if the grid allows variable cell costs *by layer, in preferred direction*, then this algorithm may behave differently, since it was supposed to work on layers with uniform cell costs (although different layers may have different costs). LCS* has shown considerable better results against A* than the experiments in [13]. This shows why it is very relevant to perform new experiments with these old algorithms in order to identify their effectiveness in detailed routing nowadays.

As future works we intend to include other algorithms and variations in the comparison, such as Hightower algorithm, adapted to 3D space, and BS*, which is the predecessor of LCS*, aiming to confirm if LCS* variable updates are still worth the effort in the detailed routing scenario.

Acknowledgment

This research is partially supported by the following Brazilian funding agencies: CAPES and CNPq.

References

[1] C. Y. Lee, "An Algorithm for Path Connections and Its Applications" in IRE Trans. on Electronic Computers, vol. EC-10, pp. 346–365, Sept 1961.

[2] F. Rubin, "The Lee Path Connection Algorithm" in IEEE Trans. on Comput., vol. C-23, pp. 907-914, 1974.

[3] P. E. Hart, N. J. Nilsson, B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths" in IEEE Trans. on Syst. Sci. and Cybern., vol. 4, pp. 100–107, July 1968.

[4] J. Soukup. "Fast maze router" in Proc. ACM/IEEE Design Automation Conf., pp. 100–102, 1978.

[5] K. Mikami, K. Tabuchi, "A computer program for optimal routing of printed circuit connectors" in Proc. Int. Federation for Inform. Process., vol. H47, pp. 1475–1478, 1968.

[6] D. Hightower. 1969. A solution to line routing problems on the continuous plane. *Proc. ACM/IEEE Design Automation Conference.*

[7] A. Hetzel, "A sequential detailed router for huge grid graphs" in Design, Automation and Test in Europe, pp. 332–338, Feb 1998.

[8] J. Humpola. 2009. Schneller Algorithmus für kürzeste Wege in irregulären Gittergraphen. Doctoral Thesis, University of Bonn.

[9] S. Peyer, D. Rautenbach, J. Vygen. 2009. A generalization of Dijkstra's shortest path algorithm with applications to VLSI routing. *Journal of Discrete Algorithms*. (Dec. 2009), 377-390.

[10] T.A.J. Nicholson, "Finding the shortest route between two points in a network"., in *Comput. J. 9,* pp. 275-280, 1966.

[11] I. Pohl. "Bidirectional search", in: *Machine Intelligence 6*, pp. 127-140, 1971.

[12] J. B. H. Kwa, "BS*: An Admissible Bidirectional Staged Heuristic Search Algorithm", in Artificial Inteligence, pp. 95-109, 1989.

[13] M. Johann, R. Reis, "Net by Net Routing with a New Path Search Algorithm" in 13th Symposium on Integrated Circuits and Systems Design, pp. 144-149, 2000.

[14] ISPD Global Routing Contest 2008. http://archive.sigda.org /ispd2008/contests/ispd08rc.html.

[15] Y. Xu, Y. Zhang, C. Chu. "FastRoute 4.0: global router with efficient via minimization" in Proc. of the 2009 Asia and South Pacific Design Automation Conference (ASP-DAC), pp. 576-581, 2009.

[16] M. Gester, D. Müller, T. Nieberg, C. Panten, C. Schulte, J. Vygen. 2013. BonnRoute: Algorithms and data structures for fast and good VLSI routing. *ACM Trans. on Design Automation of Electron. Syst.*(Mar. 2013), 1-24.