A Minimum Cost Path Search Algorithm Through Tile Obstacles

Zhaoyun Xing
Sun Microsystems Laboratories
901 San Antonio Road
Palo Alto, CA 94303
jason.xing@sun.com

Russell Kao
Sun Microsystems Laboratories
901 San Antonio Road
Palo Alto, CA 94303
russell.kao@sun.com

ABSTRACT

In this paper, based on tile connection graph, we propose an efficient minimum cost path search algorithm through tile obstacles. This search algorithm is faster than previous graph based algorithm and unlike previous tile based algorithms, this algorithm finds the minimum cost path.

Keywords

Shortest path search, VLSI routing.

1. INTRODUCTION

In this paper, we look at the problem of **minimum cost path searching through rectilinear obstacles.** Inside a rectangular area, there are several rectilinear objects, called obstacles. The space not occupied by obstacles is called clear space. For any given two points inside the clear space, the minimum cost path search problem is to find the minimum cost path inside the clear space that connects them. This problem has many engineering applications such as in VLSI routing.

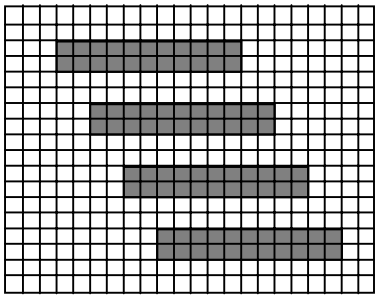


Figure 1. Maze search.

ISPD'01, April 1-4, 2001, Sonoma, California, USA

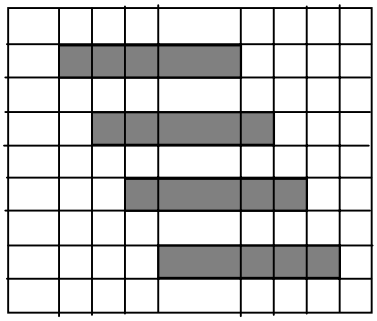


Figure 2. Non-uniform grid graph based search

There are several ways to solve this problem. The first searching method, as shown in Figure 1 is to embed the problem into a finer uniform grid space and use a maze searching to find the solution [7]. Because the grid graph could be very large in VLSI routing applications, this searching is very time consuming. Due to this reason, almost all the later works were aimed to reduce the searching graph. In [1], Cong et al proposed a reduced graph based on non-uniform grid. As shown in Figure 2, The boundaries of obstacles form the grid lines. An implicit searching method is used to explore the searching space. Though, compared with the searching graph used by maze search, the search graph is smaller, if there are large number of obstacles in the searching space, the search graph still can be very large. In [9], Wu et al built a smaller graph for the search. But the graph can not guarantee a solution even there is one. Furthermore, it is time consuming to build the graph. In [10], an implicit searching method is used for the searching through a graph, called connection graph. This graph is similar to the one used in [9]. But some expensive preprocessing is still needed. In [3][4][6][8], clear space is fractured into tiles. In the search, a tile graph is built. Each tile is a node and there is an edge between two tiles if they are neighbors. An estimate cost is given to an edge. The search is based on such a graph. The advantage of this approach is that such graph is smaller. The drawback is that the costs given to the edges are not accurate.

Figure 1 through Figure 4 shows the graph sizes of each searching approach. In Figure 2, the maze search grid graph is constructed. It has 437 nodes and 832 edges. The non-uniform graph in Figure 2 has 100 nodes and 180 edges. The connection graph in Figure 3 has 76 nodes and 132 edges. In Figure 4, there are 13 tiles and its corresponding graph has only 13 nodes and 16 edges. Those

examples show that the searching through tiles can be very efficient.

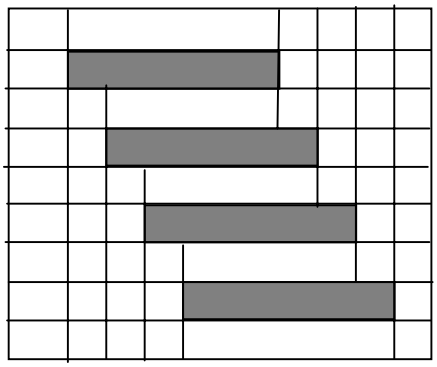
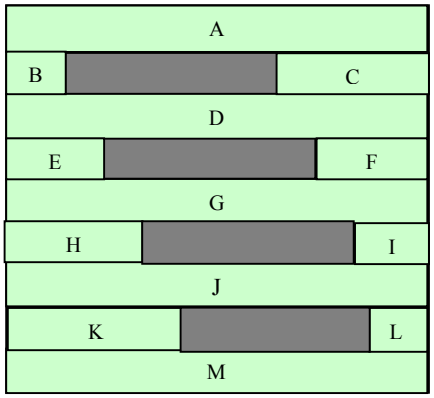


Figure 3. Connection graph



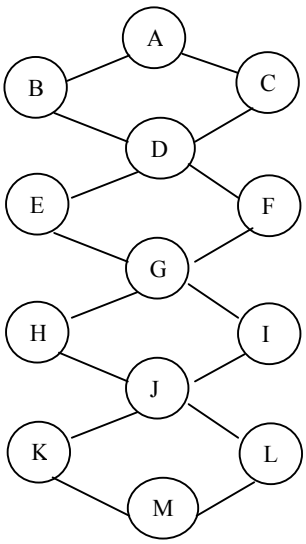


Figure 4. Partitioned obstacle tiles and space tiles.

In this paper, we propose an efficient minimum cost path search algorithm based on the small tile graph. Unlike previous tile based search algorithms that use cost estimation, our algorithm uses the technique of the minimum cost propagation. This algorithm will guarantee the minimum cost path after the search terminates.

2. PROBLEM FORMULATION

Assume on a rectangular design area, there are a list of rectilinear obstacles. As in Figure 5, The clear space has been fractured into nonoverlapping tiles (rectangles). Assume S and T are two tiles in the clear space and tiles A and B are obstacle tiles. We like to find a path from any point of S to any point on T such that the cost is the minimum. The dashed line path p in the center provides a path connecting S and T. The path p goes through 4 tiles.

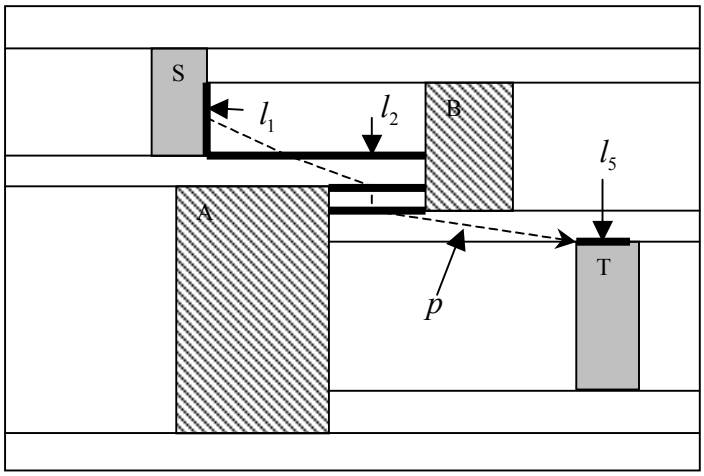


Figure 5 . VLSI routing through tiles.

Observe that to hop from one tile to the other, the path must pass through a line segment that lies on the boundaries of both tiles. We refer to this edge as the *boundary segment.* A path from source tile to target tile goes through a list of segments. Path p goes through 5 segments that are highlighted with thicker lines. Every two consecutive segments are on the boundary of the same tile. Inside each tile, assume the connection cost for any given two points (x_1, y_1) and (x_2, y_2) is defined as $c = \alpha \mid x_1 - x_2 \mid + \beta \mid y_1 - y_2 \mid, \alpha \geq 0, \beta \geq 0$, α is the unit horizontal cost, and β is the unit vertical cost. Assume the cost on the boundary of source tile S is 0. Thus the cost on segment l_1 is 0. Since segments l_1 and l_2 share the same tile, the cost function on l_1 is to be propagated to segment l_2. Note that the cost of a given point on l_2 is not constant and varies with its x coordinate. After we propagate the cost in this fashion through the rest of 5 segments, we can get the cost function on l_5, the boundary segment of destination tile T. By finding the minimum of cost function, we get the least cost from S to T through the same sequence of tiles path p passes. The problem we have to solve is how to propagate the cost function from one segment to the other.

3. COST PROPOGATION FROM ONE SEGMENT TO THE OTHER

In this section, we will study the minimum cost propagation pattern from one line segment to the other line segment. There are two cases. In the first case two line segments are perpendicular and in the second case they are parallel.

3.1 COST PROPAGATION FROM TWO PERPENDICULAR LINE SEGMENT

As shown in Figure 6, assume l_h and l_v are two perpendicular line segments. l_h runs horizontally and l_v runs vertically. Analytically assume $l_h = \{(x,h), x \in [a,b]\}$ and $l_v = \{(v,y), y \in [c,d]\}$. Assume $f(x)$ is a continuous piecewise linear function defined on interval $[a,b]$ and assume it has n knot points $(x_i, f(x_i)), 0 \le i \le n-1, a = x_0 \le x_1 \le \cdots \le x_{n-1} = b$. This function can be used to represent the cost on l_h. Though it is sufficient to consider the case l_h and l_v are on the boundaries of some tile, we consider a general case. We only assume l_h and l_v are inside the same tile. Thus the cost between two points $(x,h) \in l_h$ and $(v,y) \in l_v$ is given by $c = \alpha |v-x| + \beta |y-h|$. We like to find the least cost function $g(y), y \in [c,d]$ on l_v propagated from l_h. We will show that $g(y)$ is a piecewise linear function. For any given point (v,y) on l_v, the minimum cost function $g(y)$ can expressed as follows

$$\begin{aligned} g(y) &= \min_{x \in [a,b]} \{f(x) + \alpha |v-x| + \beta |y-h|\} \\ &= \beta |y-h| + \min_{x \in [a,b]} \{f(x) + \alpha |v-x|\} \\ &= \beta |y-h| + m \end{aligned}$$

where $m = \min_{x \in [a,b]} \{f(x) + \alpha |v-x|\}$. Since m is a constant, $g(y)$ is a piecewise linear function. We first show how to compute m. Function $p(x) = f(x) + \alpha |v-x|$ is piecewise linear. It has knot points $(x_i, f(x_i) + \alpha |v - x_i|), 0 \le i \le n-1$. If $a \le v \le b$ and v is not one of $x_i, 0 \le i \le n-1$ then $p(x)$ has an additional knot point $(v, f(v))$. Note that on any interval, a linear function always achieves its minimum at the end point of the interval. Thus a piecewise linear function achieves its minimum on its knot points. Therefore by finding the minimum y value of all knot points of $p(x)$, we can get m. If $h \notin [c,d]$, $g(y)$ is linear and it has only two knot points $(c, \beta |c-h| + m)$ and $(d, \beta |d-h| + m)$. If $c < h < d$, then $g(y)$ has an additional knot point (h,m). Apparently $g(y)$ is continuous on interval $[c,d]$.

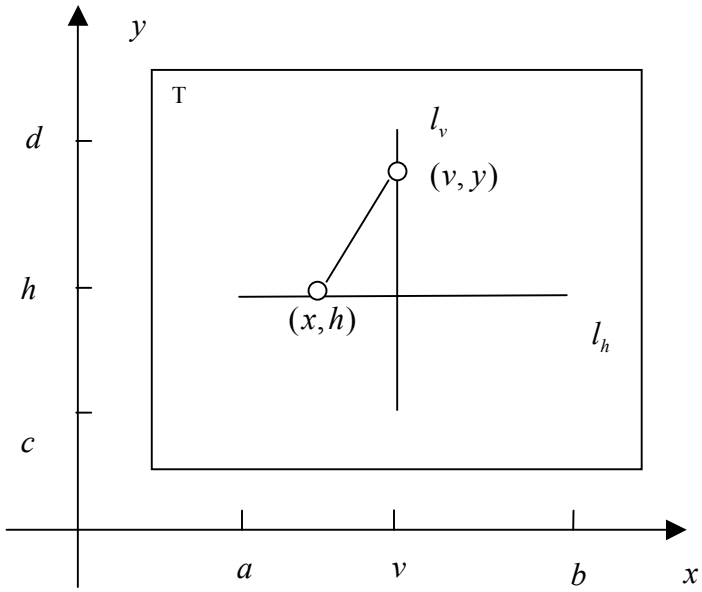


Figure 6. Least cost function propagation between two perpendicular line segments

3.2 MINIMUM COST FUNCTION PROPAGATION BETWEEN TWO OPPOSITE SIDES

As shown in

Figure 7, assume l_p and l_q are two parallel line segments and both run horizontally. Analytically assume $l_p = \{(x,p), x \in [a,b]\}$ and $l_q = \{(x,q), x \in [c,d]\}$. Assume $f(x)$ is a piecewise linear function defined on interval $[a,b]$ and assume it has n knot points $(x_i, f(x_i)), 0 \le i \le n-1, a = x_0 \le x_1 \le \cdots \le x_{n-1} = b$. This function can be used to represent the routing cost on l_p. Though it is sufficient to consider the case l_p and l_q are on the boundaries of some tile, we consider a general case. We only assume l_p and l_q are in the same tile. Thus the cost between two points $(x_p, p) \in l_p$ and $(x_q, q) \in l_q$ is given by $c = \alpha |x_p - x_q| + \beta |p-q|$. We like to find the least cost $g(x), x \in [c,d]$ on l_q propagated from l_p. For any given

point (x,q) on l_v, the minimum cost function $g(x)$ can expressed as follows

$$g(x) = \min_{\lambda \in [a,b]} \{f(\lambda) + \alpha |\lambda - x|\} + \beta |p - q|$$

Term $\beta |p-q|$ is a constant. Function $\min_{\lambda \in [a,b]} \{f(\lambda) + \alpha |\lambda - x|\}$ is a continuous function and can be computed by a process we will refer to as *linear minimum convolution*. How to efficiently compute the linear minimum convolution is given in the next section. We will show that $g(x)$ is a continuous piecewise linear function defined on interval $[c,d]$.

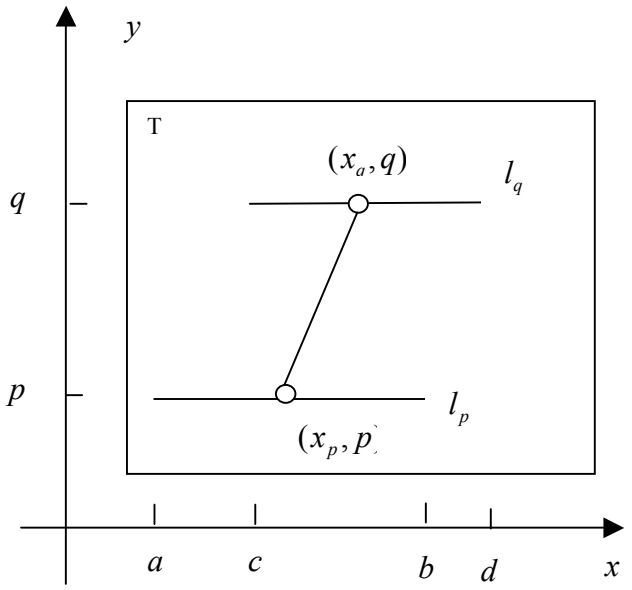


Figure 7 . Least cost function propagation between two parallel line segments.

4. LINEAR MINIMUM CONVOLUTION

In this section, we will define the linear minimum convolution, present an intuitive graphical view of the computation, and derive an efficient algorithm that takes a number of steps linear in terms of the number of knot points of the piecewise linear function.

The **linear minimum convolution** (LMC) of α and $f(x)$, and is written as $(\alpha * f)(x)$. It is given by expression

$$(\alpha * f)(x) = \min_{a \le \lambda \le b} (f(\lambda) + \alpha |\lambda - x|), x \in (-\infty, \infty)$$

To simplify the notation, we abbreviate linear minimum convolution as LMC. Suppose $f(x)$ is defined over the interval $[a,b]$. Thus, the domain of $f(x)$ is composed of an infinite number of points. Each of those points contributes an LMC Kernel. The result is the minimum of an infinite number of LMC kernels. This is shown in Figure 8. Here, the piecewise linear function $f(x)$ is shown with thicker lines, LMC kernels are formed with vertices on the curve $(x, f(x))$, and the minimum contour of the LMC is shown as the dashed line

It's interesting to note the resemblance between the LMC and the conventional convolution from linear systems theory [2].

$$Y(t) = \int_{-\infty}^{\infty} x(\lambda) h(t - \lambda) d\lambda$$

The LMC Kernel is analogous to the impulse response, $h(t)$. The decomposition of $x(t)$ into an infinite number of impulses is analogous to our decomposition of $f(x)$ into an infinite number of points. However instead of summing (integrating) the shifted and scaled impulse responses, we take the minimum of shifted and raised/lowered LMC kernels.

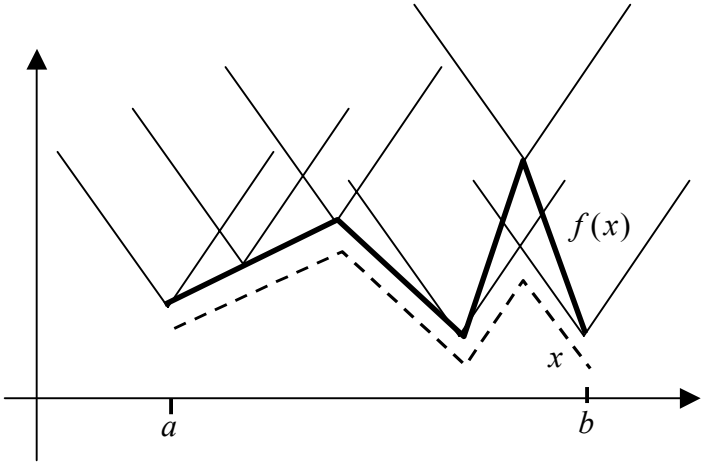


Figure 8 . The linear minimum convolution of a piecewise linear function.

In Figure 8, the piecewise linear function $f(x)$ is shown with thicker lines, wedges are formed with vertices on the curve $(x, f(x))$, and the contour of the LMC is shown as the dashed line.

LMC of a simple line segment can be trivially computed. LMC of the concatenation of two line segments is the minimum function of their LMCs.

4.1 LMC of a piecewise linear function

In this section, we consider the problem of finding the linear minimum convolution of a continuous piecewise linear function.

The LMC of a piecewise linear function can be found by finding the minimum of the LMCs of all its linear segments. Assume function $f(x)$ is piecewise linear defined as

$$(x_i, f(x_i)), 0 \le i \le n-1,$$
$$a = x_0 \le x_1 \le \cdots \le x_{n-1} \le x_n = b$$

The brute force approach of finding the LMC of $f(x)$ is to compute $(\alpha * f_i)(x)$ first. Then function $(\alpha * f)(x)$ can be computed by finding the minimum of all functions $(\alpha * f_i)(x)$. The fastest algorithm for finding the minimum function of two piecewise linear functions is linear in n, the number of segments. Therefore, the brute force algorithm to compute LMC of $f(x)$ is quadratic in the number of segments.

In this section, we propose a linear algorithm to compute the LMC of $f(x)$. For every linear segment $f_i(x)$, function $(\alpha * f)(x)$ has two infinite lines, a backward leg and a forward leg. The LMC of $f(x)$ can be found by clipping $f(x)$ using the all the legs and saving the lower line segments. Our algorithm has two sweeps, a forward sweep and backward sweep. We first give a definition. At any given point a_i not including b, a forward leg starting from $(a_i, f(a_i))$ is called *clipping* if the slope of $f_i(x)$ is greater than α. Likewise, a backward leg starting from $(a_i, f(a_i))$ is clipping if the slope of $f_{i-1}(x)$ is negative and less than $-\alpha$.

Assume we have sorted the segments. In the forward sweep, starting from a, first check whether forward segment is is clipping. If so then find the next linear segment $f_j(x)$ that intersects that leg. All points $a_k, k = i, ..., j$ will be ignored because they will not generate any linear segments lower than current leg. The process resumes from point a_{j+1}, continuing until b. The backward sweep is the opposite of the forward sweep. Starting from b, clipping the modified piecewise linear function using all clipping backward legs.

For line segment with a sloe that is positive and less than α, the forward leg at the left end point is not and f or line segment with a sloe that is positive and greater than α, the forward leg at the left end point is clipping. We know that the LMC of $f(x)$ consists only of parts of line segments of $f(x)$, forward legs, and backward legs. The clipping process chops out the parts of segments that are not part of the LMC. To prove our algorithm is correct, in the next lemma, we show that after a clipping forward leg intersects $f(x)$, the remaining line segment is no longer clipping.

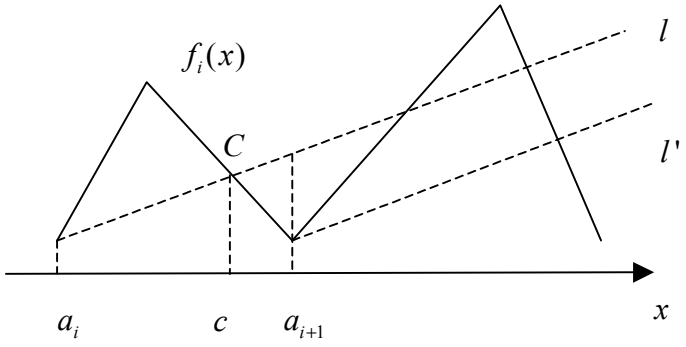


Figure 9. Forward leg clipping

After a clipping forward leg l intersects with a linear line segment $f_i(x)$, its remaining segment is not clipping. The same is true for backward leg. Based on this fact, we propose a linear running time algorithm to find the LMC of the piecewise linear function $f(x)$.

Algorithm 1 is the linear minimum convolution algorithm. Algorithm 2 is the forward leg sweep algorithm. The backward leg sweep algorithm is similar to the forward leg sweep algorithm; its listing is omitted.

Algorithm 1 (Linear Minimum Convolution Algorithm)

Input: α, a positive number and $L = \{l_0, l_1, \cdots, l_{n-1}\}$, a sorted list of line segments that represents the continuous piecewise linear function $f(x)$.

Output: L, the list of line segments that represents $(\alpha * f)(x)$, a continuous piecewise linear function.

1. Perform the forward leg sweep over *L*
2. Perform the backward leg sweep over *L*
3. The resulting *L* contains the segments of the piecewise linear function $(\alpha * f)(x)$.

Algorithm 2 (Forward Leg Sweep Algorithm)

1. Set index pointer $p = 0$.
2. Scan *L* from segment l_p to the end of *L* to find the first segment whose slope is greater than α. Let l_j stand for that segment. If no such segment is found, return.
3. Define a forward leg: $g(x) = \alpha(x - a_j) + f(a_j)$
4. Remove segments $l_j, l_{j+1}, \cdots$ from *L* until encountering a segment that intersects with $g(x)$. If no segment intersecting $g(x)$ is found, then go to step 10.

5. Let l_i be the segment found in step 4 that intersects with *g(x)*. Let $P(x_P, y_p)$ stand for the intersection point and let $R(x_R, y_R)$ stand for the right end point of l_i.
6. Insert a new segment $g(x), a_j < x < x_P$ into L at position j.
7. Insert into *L* at position *j+1* a new segment, that has P as its left end point and the right end point of l_i as its right end point.
8. Set $p = p + 2$
9. Go to step 2.
10. Insert a new segment: $g(x), a_j < x < \infty$ into *L* at position *j*. Return

5. LEAST COST PATH SEARCHING ALGORITHM

For any given line segment, the lower bound cost to a target tile can be easily evaluated by finding the minimum of the lower bounds to four sides of the tile. In this section, we propose a least cost path search algorithm using A* search method. Our algorithm works as follows. The boundary segments of source tile S are given 0 cost and their estimated costs to the target tile T are evaluated. Those segments are put into a priority queue with the estimated costs. Whenever there is a segment in the queue, the segment with the smallest cost is de-queued. First locate all the clear tiles that touch this segment. Then propagate the cost to all segments that border those clear tiles. Then evaluate the estimated cost for each of those segments and put them into the priority queue. After the queue is empty, check to see if T has any searched segments. If so, find the boundary segment of T with lowest cost and trace back to the source. Otherwise, report search failure. We describe the search as starting from source tile to target tile. However, this search can be bi-directional to reduce the search space.

Algorithm 3. Least cost searching algorithm

Input: source tile S and target tile T.

Output: a sequence of line segments that represents the least cost path from S to T

1. Initialization
    2. Set the cost of the boundary segments of S to 0
    3. For each boundary segment,
        4. Evaluate the estimated cost function to the target tile T.
        5. Find the estimated cost and assign the result as the cost of the segment.
        6. Push the segment to the priority queue.
7. While queue is not empty do
    8. De-queue AB, the segment with the min cost value
    9. For each segment CD that share the same tile with AB do
        10. Propagate the cost from AB to CD
        11. Find the estimated cost and use it as the cost of CD
        12. Push CD to the queue
13. If T has any searched segments. If so, find the boundary segment of T with lowest cost and trace back to the source. Otherwise, report search failure.

6. CONCLUSION

In this paper, based on small tile connection graph, we propose an efficient minimum cost path search algorithm through tile obstacles. This algorithm is faster than previous graph based algorithm. Unlike previous tile based search algorithms that use cost estimation, our algorithm uses the technique of the minimum cost propagation. This algorithm will guarantee the minimum cost path after the search terminates. Future works include the application of this algorithm to VLSI routing.

7. REFRENCES

[1] J. Cong, J. Fang, and K. Khoo, "An Implicit Connection Graph Maze Routing Algorithm for ECO Routing," Proceedings of ICCAD 1999, San Jose, CA, p163-167.

[2] L. Chua, C. Desoer, and E Kuh, *Linear and Nonlinear Circuits*, McGraw-Hill, Inc, 1987.

[3] Jeremy Dion and Louis M. Monier, "Contour: A Tile-based Gridless Router," Western Research Laboratory Research Report 95/3, Palo Alto, California.

[4] Margarino, A. Romano, A. De Gloria, F. Curatelli, and P. Antognetti, "A Tile-Expansion Router," IEEE Transactions on Computer-Aided Design CAD-6(4): 507-517, July, 1987.

[5] N. J. Nilsson, *Principles of Artificial Intelligence.* Englewood Cliffs, New Jersey,: 1980, pp. 53-94.

[6] John K. Ousterhout, "Corner Stitching: A Data-Structuring Technique for VLSI Layout Tools." IEEE Transactions on Computer-Aided Design, Vol. CAD-3, NO. 1, pp. 87-100, January 1984.

[7] S. Sait and H. Youssef, "VLSI Physical Design Automation – Theory and Practice," IEEE press, 1995.

[8] Tsai, S. Chen, and W. Feng, "An H-V Alternating Router," IEEE Transactions on Computer-Aided Design 11(8):976-991, August, 1992.

[9] Y. Wu, P. Widmayer, M. Schlag, and C. Wong, "Rectilinear Shortest Paths and Minimum Spanning Trees in the Presence of Rectilinear Obstacles," IEEE Transactions on Computers, Vol. C-36, NO. 3, March 1987.

[10] S. Zheng, J Lim, and S. Iyengar, "Finding Obstacle-Avoiding Shortest Paths Using Implicit Connection Graphs," IEEE Transactions on Computer-Aided Design, Vol. 15, No. 1, January 1996.