Rectilinear Paths among Rectilinear Obstacles[*]

D.T. Lee, C. D. Yang[†]
Department of Electrical Engineering and Computer Science
Northwestern University
Evanston, IL 60208
e-mail: dtlee@eecs.nwu.edu

C. K. Wong[‡]
Department of Computer Science
Chinese University of Hong Kong
Shatin, New Territories
Hong Kong
e-mail: wongck@cs.cuhk.hk

ABSTRACT

Given a set of obstacles and two distinguished points in the plane the problem of finding a collision free path subject to a certain optimization function is a fundamental problem that arises in many fields, such as motion planning in robotics, wire routing in VLSI and logistics in operations research. In this survey we emphasize its applications to VLSI design and limit ourselves to the rectilinear domain in which the goal path to be computed and the underlying obstacles are all rectilinearly oriented, i.e., the segments are either horizontal or vertical. We consider different routing environments, and various optimization criteria pertaining to VLSI design, and provide a survey of results that have been developed in the past, present current results and give open problems for future research.

1 Introduction

Given a set of obstacles and two distinguished points in the plane, the problem of finding a collision free path subject to a certain optimization function is a fundamental problem that arises in many fields, such as motion planning in robotics, wire routing in VLSI and logistics in operations research. In this survey we emphasize its applications to VLSI design and limit ourselves to the rectilinear domain in which the goal path to be computed and the underlying obstacles are all rectilinearly oriented, i.e., the segments are either horizontal or vertical. We provide a survey of

[*]Supported in part by the National Science Foundation under Grants CCR-8901815 and CCR-9309743.
[†]Present address: ArcSys Inc. Cupertino, CA 95014. E-mail: yangcd@arcsys.com
[‡]On leave from IBM T. J. Watson Research Center, P.O. Box 218, Yorktown Heights, NY 10598

results pertaining to routing in VLSI according to routing environment, optimization criterion, and problem solving approach.

In VLSI layout, a basic function unit or circuit module, is represented by a rectilinear polygon, whose shape may or may not be (rectilinearly) convex. The *pins* or *terminals* of the same *net* that lie on different modules need to be interconnected by wires that mostly run either horizontally or vertically. In a single layer model, the total *length* of wires in the interconnection is often used as an objective function, as it affects cost of wiring and total delay. On the other hand, the *number of bends* on a path affects the resistance and hence the accuracy of expected timing and voltage in chips. In a two layer model in which horizontal wires are restricted to be on a *horizontal* layer, and vertical wires are restricted to be on a *vertical* layer, the number of *bends* used in the interconnection becomes an important measure as well, as each bend corresponds to a *via*, i.e., a connection between the two layers. Having more *vias* not only introduces unexpected resistance and higher fabrication costs, but also decreases the reliabibility of the interconnection. It is desirable to have these routing measures, i.e., length and the number of bends, minimized. (In the geometry literature it is referred to as *bicriteria* optimization problem. See [51] for details.) Unfortunately they cannot be both optimized *simultaneously* in general. Therefore a "best path" can be categorized by either minimizing each of these measures individually, giving them different optimizing priorities, or giving one a certain bound and minimizing the other. In addition to these optimization factors of rectilinear paths, the routing models and the *types* of obstacles also affect the complexity of the problems.

Most of the previous results [4, 11, 15, 22, 23, 34, 36, 37, 43, 47, 48, 49, 73, 75, 77] deal with problems under the assumption that paths do not cross any obstacles, i.e., the path is *collision free* (Figure 1a). Among these results, many aim at finding a collision free *shortest* path among a set of *non-intersecting* obstacles in the plane[11, 15, 34, 47, 48, 49, 57, 75, 77]. In the hierarchical representation of VLSI layout, some pre-routed modules or lines may be re-designed or re-routed if it is subsequently found to be more beneficial to have some paths cross over the area previously occupied by them. Under these circumstances, such modules can be considered *penetrable* and assigned *weights* properly representing the cost to re-construct them (Figure 1b). In this setting, the total routing cost is measured by adding the extra cost (due to penetration) to the original measure.

As to the models where routing is performed, researchers have considered two or more layers with restrictions reflecting the needs or limitations of VLSI design. In a *two layer model*, obstacles are scattered on two layers separately and the routing orientation in each of the layers is fixed, e.g., allowing horizontal routing in one layer and vertical routing in the other. Paths can switch from one layer to the other when they need to change directions. Assorted paths with respect to the criteria mentioned above, i.e., *length* and *number of bends*, are also sought in this model. In addition, finding paths by minimizing the total length of their *subpaths* in one of the layers has also been studied. This is motivated from practical considerations that different layers are fabricated using different materials, e.g., silicon and metal, and that electric current runs at a different speed in each separate layer. Since electric current runs much faster in metal layer than in silicon layer, minimizing only the path length in silicon layer may be sufficient to give an approximation to the actual delay. More importantly this might better minimize total delay than minimizing total path length.

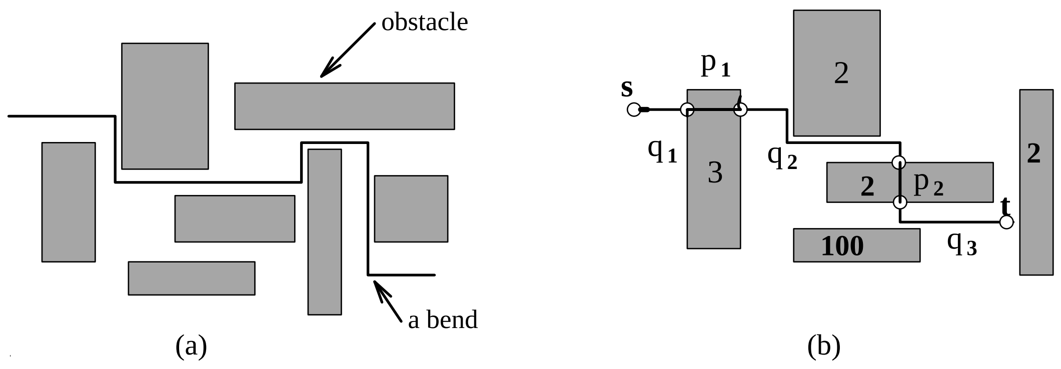

Figure 1: Finding paths among obstacles

Although algorithms finding a path between two terminals are adopted in practice as a basic routine for iteratively routing multiple two-terminal nets, finding more than one path at a time and minimizing the total cost of these paths could yield better overall results. Since the general problems that call for an overall optimal solution for arbitrary k pairs of paths are intractable[20], it is practical to settle for routing of a limited number of pairs of paths at a time. We thus consider the problem of finding k non-crossing paths, where k is a small fixed integer. These paths can be adjacent to each other but must be *non-crossing*. This has applications to topological routing in VLSI, for example, the rubber-band routing [12].

In the next section, we formally define the problems. In section 3, we briefly discuss previous and current results, and summarize them in two tables. In section 4, we discuss several general problem-solving approaches used in previous works. In section 5, we summarize some interesting problems.

2 Preliminaries

A *rectilinear path* is a path consisting of only vertical and horizontal segments. The *length* of a rectilinear path is the sum of the lengths of all its segments. An obstacle is said to be *rectilinear* if all its boundary edges are either horizontal or vertical. The point adjacent to a horizontal segment and a vertical segment is called a *bend*. Let $d(\pi)$ and $b(\pi)$ denote, respectively, the length and the number of bends of a path, π. The *rectilinear distance* between two points p and q equals $|p.x - q.x| + |p.y - q.y|$ where $p.x$ and $p.y$ are the abscissa and the ordinate of point p, respectively.

From here on, 'path' refers to rectilinear path and all the obstacles are rectilinear unless otherwise specified.

Definition 1 Given a set of disjoint (non-penetrable) obstacles, a source point s and a destination point t, we define the following problems:

Shortest Path (SP) Problem: Find a shortest (collision free) path from s to t, denoted as *sp*.

Minimum Bend Path (MBP) Problem: Find a path from s to t with a minimum number of bends, denoted as *mbp*.

Smallest Path (SMALLP) Problem: Find a path from s to t such that it is shortest and has a minimum number of bends *simultaneously*. Such a path is called a *smallest path* and denoted as *smallp*. Note that such a path may not exist.

Minimum Bend Shortest Path (MBSP) Problem: Find a path with the minimum number of bends among all the shortest paths from s to t. Such a path is called a *minimum bend shortest path* and denoted as *mbsp*.

Shortest Minimum Bend Path (SMBP) Problem: Find a shortest path among all the minimum bend paths from s to t. Such a path is called a *shortest minimum bend path* and denoted as *smbp*.

Minimum Cost Path (MCP) Problem: Find a minimum cost path from s to t, where the cost is a non-decreasing function of the length and the number of bends of a path, denoted as *mcp*.

Bounded Bend Shortest Path (BBSP) Problem: Find a shortest path among all the paths from s to t with the number of bends no greater than a given bound. Such a path is called a *bounded bend shortest path* and denoted as *bbsp*.

Bounded Length Minimum Bend Path (BLMBP) Problem: Find a path with the minimum number of bends among all the paths from s to t with length no greater than a given bound. Such a path is called a *bounded length minimum bend path* and denoted as *blmbp*.

As mentioned before, obstacles may be penetrated at an *extra* cost. We represent the extra cost of penetrating an obstacle as the *weight* of the obstacle, and define the *weighted* path length as follows.

Definition 2 A path Π_{st} from point s to t in the presence of *weighted* obstacles can be decomposed into subpaths $q_1, p_1,\ q_2, p_2, \ldots, q_k, p_k$, where q_i is a subpath without intersecting the interior of any obstacle and p_i is a subpath completely within some obstacle R_{j_i} (Figure 1b). Subpaths q_1 and p_k may be empty. The *weighted length* of Π_{st}, denoted $dw(\Pi_{st})$, is defined as $dw(\Pi_{st}) = \sum_{i=1}^{k}(d(q_i) + d(p_i)) + \sum_{i=1}^{k} d(p_i) * R_{j_i}.w$, where $R_{j_i}.w$ is the weight of obstacle R_{j_i}.

Note that the first term contributing to the weighted distance is the (unweighted) path length $d(\Pi_{st})$, and the second term reflects the extra cost of going through the interior of the obstacles. We have the following problems with respect to different types of obstacles. Let X $\in \{sp, mbp, smallp, mbsp, smbp, mcp, bbsp, blmbp\}$.

Definition 3 Given a problem instance as in Definition 1, we define the following problems.

Weighted X Problem: Find an X path when the obstacles are weighted. The distance measure is the weighted distance of a path.

(Weighted) $\mathbf{X}^{\square}$ Problem: Solve the (weighted) X problems when the given obstacles are all rectangular.

The *two layer model* is defined as follows. We are given two layers as the routing environment. On the *horizontal routing layer* (denoted as H_{layer}), only horizontal segments are allowed, and

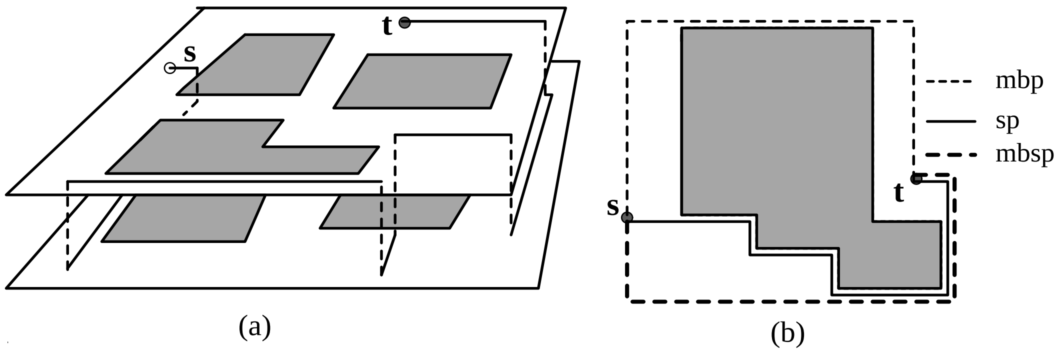

Figure 2: Finding paths on multi-layer and single layer.

on the *vertical routing layer* (denoted as V_{layer}) only vertical segments are allowed. Obstacles are given on each layer and they are disjoint on the same layer. A path in this model consists of horizontal segments on H_{layer} and vertical segments on V_{layer}. When a path switches layers, a bend or *via* is introduced (Figure 2a.) The total length of the segments on H_{layer} is called H-distance or x-distance of the path. V-distance or y-distance of a path is similarly defined.

Definition 4 Given a problem instance as in Definition 1, we define the following problems.

2-Layer$_{r|ur}$ X Problem: Find an X path in a two layer model where each obstacle is specified to be in one of the two layers. There are two variants: the one with subscript r restricts the routings on the two layers to be horizontal and vertical respectively; the other with subscript ur ("*u*n*r*estricted") allows both vertical and horizontal routing in each layer. In this paper, we consider only the restricted problems (with subscript r) in the two layer model.

2-Layer$_{r|ur}$ 1D-X Problem: Find an X path in a two layer model measuring only the distance of path in *one* specified layer. As before, routings in each layer may be restricted to be either vertical or horizontal (subscript r) or may be unrestricted (subscript ur).

1D-X Problem: Find an X path in the single layer model measuring only the x- or y-distance of the path. It can be viewed as solving a 2-Layer 1D-X problem when the obstacles are considered present on both layers.

Multi-Layer (mD-)X Problem: Find an X path in a multi-layer model where routing in each layer may or may not be restricted to some directions. The distance measure may refer to the total length of subpaths on some specified layers.

Z^{pol} Problem: Solve Z problem within a given simple rectilinear polygon, where Z is any problem name defined above.

Two paths are called *non-crossing* if they either are disjoint or may overlap but never *cross* each other (See [81] for a formal definition.)

Definition 5 **kNCP-Z Problem:** Find k non-crossing paths between k pairs of given points, where Z is as defined above. For example, 2NCP-SP is to find two non-crossing paths with shortest total length between two given pairs of points. 2NCP-SMALLP is to find two non-crossing paths with shortest total length and fewest total number of bends between two given pairs of points, referred to as the *smallest pair*.

Another variation is the *query version* of these problems, in which the obstacles are given a priori (i.e., fixed), pre-processing of the obstacles is allowed, and one or both of the source and the destination are specified on-line. We have two cases depending on whether the source is given a priori or not.

Definition 6 Query$_{1|2}$ Z Problem: Find paths designated by Z from a given source to any query destination (subscript 1) or between two query points (subscript 2) among a given set of obstacles, with preprocessing allowed.

3 Previous and Current Results

We give a survey of previous and current results based on the problem classification defined above. Let n denote the number of obstacles, e the number of obstacle edges and V the set of obstacle vertices including the source and the destination.

3.1 Problems SP and SP$^{\square}$

In conventional Computed-Aided Design for PCB or VLSI circuit routing, wires and modules are usually restricted to lie on certain grids embedded on the board. This eases the complexity of design and the fabrication as well. When there is an embedded grid consisting of vertical and horizontal lines, the path finding problem is to find a rectilinear path along the grid lines from a designated starting grid point to another destination grid point while avoiding all obstacles, represented as a set of marked grid points in this grid structure. The Lee algorithm [35] was the first one finding a shortest path between two points on a routing plane with grid. It applies the algorithm due to Moore [54] on a planar grid structure. It simply runs over the grid step by step starting from the source point till the destination is reached (Figure 3a). The Lee's and Moore's algorithm are called the *grid expansion* or *maze running* algorithm. Though it has the merit of simplicity, it consumes too much memory and time. It needs $O(m^2)$ space to represent an $m \times m$ grid structure and has a worst case time complexity $O(l^2)$ for finding a path crossing l grid points. Many algorithms were derived to improve the maze running algorithm [1, 2, 3, 21, 24, 28, 62, 66]. They achieved certain amount of speed-up but did not improve the worst case time complexity. See section 4.1 for more details about this approach. To reduce the large amount of memory required for maze running algorithms, a *gridless model* was introduced, where obstacles are each represented by a set of line segments on the plane.

Mikami and Tabuchi [46], and independently, Hightower [27] proposed a *line searching* approach to finding a path connecting the source and the destination. It first emits vertical and horizontal lines from the source. If these lines have intersections with other lines that pass through some boundary edges of the obstacles, the searching *jumps* over to those lines and continues. See

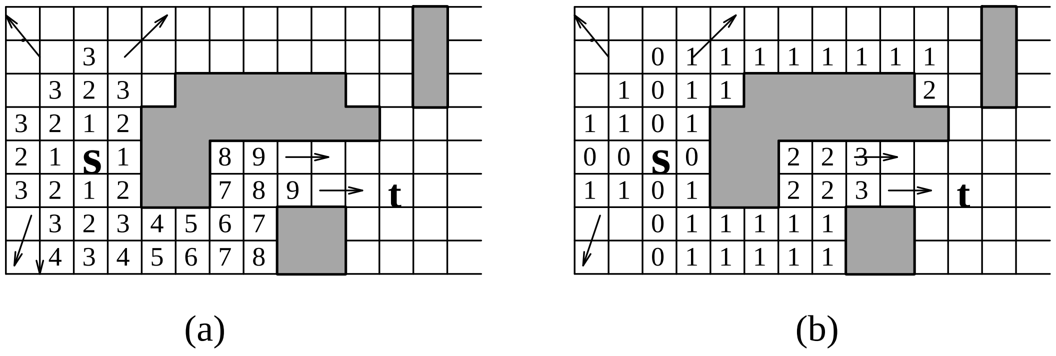

Figure 3: Maze Running approach

section 4.2 for more details of this approach. Though their algorithms aimed only at finding connections between source and destination (in fact, it finds a *mbp* instead of a *sp*), they introduced a more efficient representation for path finding problems.

Larson and Li [34] studied the problem of finding all shortest paths among a set of m source-destination pairs with polygonal obstacles. By transforming the problem into a network problem, they solved it in $O((m+e)^2)$ time. deRezende *et al.* [15] proposed a $\theta(n \log n)$ time algorithm for $\mathrm{SP}^{\square}$ along with a proof of its optimality. They utilized the monotonicity of the paths and used a so-called *plane sweep* approach[61] to find the path. Mitchell [47] proposed a *45-degree wave front* approach and presented an $O(e \log^2 e)$ algorithm for SP.[1] He used 45-degree line segments, called *wave fronts*, to carry distance information as they travel away from the source. A wave front can make turns around obstacles, can split to two wave fronts if it hits an obstacle and can be deleted if it is subsumed by other wave fronts. Wave fronts are processed in ascending order of the distances they carry. The distance of a shortest path can be reported when a wave front first hits the destination and the actual path can be constructed by tracing back the history of the movements of wave fronts. More of this approach is described in section 4.5. For SP, two of the best results[2] were due to Clarkson, Kapoor and Vaidya [11] and Wu, Widmayer, Schlag and Wong [77]. One runs in $O(e \log^{3/2} e)$ time and space[11] or alternatively in $O(e \log^2 e)$ time using $O(e \log e)$ space[10], and the other in $O(e \log m + m^2 \log m)$ time[77], where m is the number of extreme edges among those given obstacles. An obstacle edge is *extreme* if its two adjacent edges lie on the same side of the line containing the edge. Both the algorithms[47] and [11] can also find shortest paths among simple polygonal obstacles. The algorithm by Wu, *et al.*[77] may have better performance than the one by Clarkson, *et al.*[11], if m is much less than e. Both algorithms use a *graph-theoretic* approach where a so-called *path preserving* graph is constructed which provides sufficient distance information for one to apply graph searching algorithms to find the goal path. More details of this approach are given in section 4.4.[3]

[1]The time complexity is $O(E \log e)$, where E is the number of events while dragging wave fronts on the plane. That $E = O(e \log e)$ was proved but $E = O(e)$ was conjectured. The conjecture was later proved true in [50].

[2]Mitchell in a recent paper [50] presented an $O(e + n \log n)$ time algorithm, which is asymptotically optimal.

[3]Yang, *et al.* [82] recently proposed a method combining the features from these two path preserving graphs and obtained a better algorithm which runs in $O(e \log m + m \log^{3/2} m)$ time using $O(e + m \log^{3/2} m)$ space.

3.2 Problems MBP and SMALLP^{pol}

The problem of finding a minimum bend path has been studied and gained more attention recently. In Euclidean case, several minimum bend path [4] algorithms have been proposed (e.g., [33, 53, 67]). These algorithms for Euclidean cases may provide approximations for rectilinear case [13, 33, 42, 45, 53, 57, 67]. Although the Lee algorithm [35] can be modified to find a minimum bend path, (Figure 3b) it has the drawback that it is neither time nor space efficient. The Mikami-Tabuchi line searching algorithm [46] finds an *mbp* in a plane or in the two layer model. Hightower's algorithm [27] is similar to the Mikami-Tabuchi algorithm, but it does not always find an *mbp*. Combining both maze running approach and line searching approach, Heyns *et al.* [25] proposed a *line expansion* algorithm. (See section 4 for details.) The algorithm [25] has advantages of assuring a connection and preventing duplicated searching, but has a drawback that it does not always find a minimum bend path. Ohtsuki [58] provided suitable data structures and analyzed their algorithms [25, 27, 46] showing that they all have quadratic time complexities.

Ohtsuki [57] proposed an algorithm for finding an *mbp*, which is modified from line searching algorithms. The algorithm runs in $O(e \log^2 e)$ time using $O(e)$ space. Alternatively, one can use the data structure given by Imai and Asano [30] and apply Ohtsuki's algorithm to solve MBP in $\theta(e \log e)$ time and space. Sato *et al.* [64] independently presented a line searching algorithm which finds an *mbp* in $O(e \log e)$ time. Their algorithm has an advantage that iterative path searching can be done efficiently. A tile expansion router obtained by Margarino, *et al.* [44] and derived from line expansion approach, finds an *mbp* but no analysis of its time complexity is given.

Other variants of SP and MBP are also studied. de Berg [14] showed that there always exists a smallest path within a simple rectilinear polygon and presented an algorithm to solve SMALLP^{pol} (or alternatively SP^{pol} or MCP^{pol}) in $O(e \log e)$ time and space. His algorithm was basically designed for the query version of the problem (see section 3.6). McDonald and Peters [45] later presented a linear time algorithm for SMALLP^{pol}. Lipski proposed a *Manhattan Path* problem asking for a rectilinear path (not necessarily an *sp*) connecting two specified segments such that the path is covered (constructed) by a given set of n horizontal or vertical line segments. He presented an $O(n \log n \log \log n)$ time algorithm [40] in which an *mbp* tree from the source segment to every other reachable segment is constructed, which is later improved to run in $\theta(n \log n)$ time and space[41]. For a generalized Manhattan Path problem where the segment intersection of the path is required to lie in a given rectangular area, Asano [5] presented an algorithm that runs in $O(n \log n)$ time and space or alternatively $O(n \log^2 n)$ time using $O(n)$ space.

3.3 Problems MBSP$^{\square}$, MBSP, SMBP, MCP, BBSP and BLMBP

Algorithms which find the shortest paths do not necessarily produce a path with a minimum number of bends, and vice versa. For example, the algorithm due to Wu, *et al.* [77] finds a shortest path but leaves the number of bends in the solution path uncertain. The shortest path obtained by Clarkson, *et al.* [11] tends to have more bends than necessary, as shown by the solid lines in Figure 2b. Similarly, the minimum bend path obtained by Ohtsuki[57] is by no means close to being the shortest.

[4] Sometimes it is referred to as *minimum link path* in the literature.

Using a *graph-theoretic approach*, Yang, Lee and Wong [79] constructed a *path preserving* graph which is assured to retain desired goal paths, *mbsp, smbp, mcp, bbsp* and *blmbp* (see section 4.4). By applying the shortest path searching algorithm of Fredman and Tarjan [18] to such a graph, an $O(K + e \log e)$ algorithm is derived for MBSP, SMBP and MCP, where K, bounded by $O(em)$, is the number of intersections between *tracks*, i.e., vertical or horizontal lines passing through extreme points and convex vertices. (Recall that m is the number of extreme edges of the obstacles.) The time complexity can be improved to $O(K + \frac{e \log e}{\log \log e})$, if one applies the *trans-dichotomous* algorithm of Fredman and Willard [19].[5] For BBSP an $O(B(K + e \log e))$ time algorithm was given in [79] based on the same graph, where $B = \min\{b^*, b\}$, and b^* and b are, respectively, the bend number of the *mbsp* and the given bound for the number of bends. As for BLMBP, a similar algorithm, which runs in $O(b^*(K + e \log e))$ time was given [79]. When obstacles are rectangular, the *mbsp* can be found in optimal $\theta(e \log e)$ time using plane sweep approach (see Yang, *et al.* [83].) A new *dynamic searching* approach is devised by Yang, Lee and Wong [82] by which MBSP and MCP can be solved in $O(e \log^{3/2} e)$ time and space. Instead of pursuing a smaller path preserving graph, the authors [82] construct a graph just for guidance purpose. The so-called *guidance graph* is assured to contain the homotopy information [6] of the goal path and allows one to perform searching on it and compute the goal path on the fly. It is a combination of both graph-theoretic and continuous searching (e.g., wave front approach) approaches. More about this approach can be found in section 4.6. For SMBP, an optimal $\theta(e \log e)$ time algorithm was also obtained by Yang, *et al.* [80] using a *horizontal wave front* approach. More of it is discussed and compared with 45-degree wave front approach in section 4.5.

3.4 Weighted X and Weighted X$^\square$ Problems

In [52], Mitchell and Papadimitriou first introduced the Euclidean *weighted region problem* with applications to motion planning in robotics. They proposed an algorithm to find the shortest *weighted distance* path among weighted regions that may be adjacent. Their algorithm runs in $O(n^8 L)$ time, where L is the precision of the problem instance (including the number of bits required to specify the largest integer among the weights and the coordinates of vertices) and n is the number of the weighted regions.

The version where paths are rectilinear and obstacles are non-adjacent was studied by Lee, Yang and Chen [38], who extended the $O(e \log^{3/2} e)$ result of Clarkson, Kapoor and Vaidya [11] for SP to Weighted SP. For Weighted SP$^\square$ problem Yang, Chen and Lee [78] proposed an optimal $\theta(n \log n)$ time algorithm using plane sweep approach and a weighted segment tree structure.[7]

3.5 2-Layer$_{r|ur}$ X Problems and 1D-X Problems

Ohtsuki and Sato [56] studied the 2-Layer 1D-SP problem and presented an $O(e \log e)$ algorithm. They used graph-theoretic approach to construct a linear size graph providing suitable intercon-

[5] The trans-dichotomous algorithm utilizes addressing of random access machine and speeds up computation in a way falling outside the framework of comparison based algorithms.

[6] The homotopy of a path is its topology with respect to the obstacles. Two paths are homotopic to each other if and only if either one can be dragged or transformed to the other without crossing any obstacles.

[7] Weighted MBSP$^\square$ is also solved in $\theta(n \log n)$ by Yang, Lee and Wong [83].

nections in the routable region and then applied the Dijkstra's algorithm [16] to solve the 1D-SP problem. They also studied 2-Layer$_{ur}$ MBP problem. For this problem, a bipartite graph is constructed to represent the interconnections in the routable regions on both layers, and then searching is applied to the graph. Their algorithm runs in $O(e \log^2 e)$ time using $O(e)$ space or alternatively, in $O(e \log e)$ time and space if one adopts the data structure of Imai and Asano [30] (see also [6]). A simpler version of the problem is to consider the x-distance or the y-distance of the paths in a plane. Yang, Lee and Wong [80] presented optimal $\theta(e \log e)$ time and $O(e \log e)$ space algorithms for 1D-MBSP, 1D-SMBP and 1D-MCP by using the horizontal wave front approach. Their results can be modified to solve 2-Layer 1D-MBSP, 2-Layer SMBP and 2-Layer 1D-MCP by adopting a two-layer-to-one-layer problem transformation [39]. The problem transformation proposed by Lee, Yang and Wong [39] provides a general solution for most of the problems in the two layer model. They defined a *skeleton* which is a set of segments that can be placed in a single goal layer and which represents sufficiently the blocking behavior of all the obstacles on two layers. After making the skeleton the new obstacle set, most of the previous path finding algorithms in the plane, i.e., in the one layer routing enviroment, can be applied and the resultant path can be easily transformed to the goal path in the two layer model. The transformation can be done in $O(e \log e)$ time, which is dominated by the time complexities of most path finding algorithms. The correctness of the path transformation depends on a property that is required of the path finding algorithms. The property, which is called the *alignment property*, requires each segment of the goal path to be aligned with obstacle edges. As mentioned in [39], most algorithms have the property or can be modified to have it by post-processing. In [39], an $O(e \log^2 e)$ algorithm is also presented to solve 2-Layer$_r$ SP without using the problem transformation.

3.6 Query$_{1|2}$ Z Problems

Along with solving SP$^{\square}$, deRezende, Lee and Wu [15] also solved Query$_1$ SP$^{\square}$ in $O(\log n)$ time with $O(n \log n)$ preprocessing. Recently, Atallah and Chen [7] (independently ElGindy and Mitra[17]) presented an algorithm for solving Query$_2$ SP$^{\square}$ in $O(\log e)$ time after $O(e^2)$ preprocessing. More details about this approach can be found in section 4.7. Variations where query points are restricted to be on the obstacle boundary or on the boundary of a polygon which encloses all obstacles are also discussed in [7].

While solving SP, Wu, *et al.*[77] also solved Query$_1$ SP in $O(\log e)$ time after $O((m^2+e) \log m)$ time and $O(m^2 + e)$ space preprocessing for constructing a planar subdivision from their track graph. In [47], Mitchell constructed a *shortest path map* structure (originally defined in [37]) of size $O(e \log e)$, with which the shortest distance from the source to a query point can be found in $O(\log e)$ time.

de Berg, van Kreveld, Nilsson and Overmars [13] considered the case where the obstacles are all horizontal or vertical segments and a *mcp* from a fixed source to a given destination is queried. The input segments are allowed to intersect each other.

As for preprocessing, their algorithm spends $O(n^2)$ time and $O(n \log n)$ space to compute one-to-all *mcp*'s from the source to every endpoint, where n is the number of given segments. They used a *slab tree* data structure (built in $O(n \log n)$ time[13]) and reported an *mcp* from the fixed source to the query destination in $O(\log n)$ time. The slab tree structure can also be used as a pre-constructed structure for query version of other path finding problems. Once all the goal paths from the fixed source to all obstacle vertices are found, one can construct a slab tree that partitions the plane and supports iterative queries. For example, Query$_1$ SP can be solved in $O(\log e)$ time after $O((m^2+e)\log m)$ preprocessing time if the SP algorithm by Wu, *et al.* [77] is used to find the one-to-all shortest paths, or $O(e \log^{3/2} e)$ preprocessing time if the SP algorithm by Clarkson, *et al.* [11] is used. More details of the slab tree can be found in section 4.8.

de Berg [14] also presented algorithms that solve Query$_2$ SP^{pol} and Query$_2$ MBP^{pol} in $O(\log e)$ time after $O(e \log e)$ time and space preprocessing.[8] He also considered the variant when the query points are vertices of the polygon.

3.7 kNCP-Z Problem

Even for planar graphs, the problem of finding k disjoint paths for k pairs of vertices for an arbitrary k, is known to be NP-complete [20]. When k equals 2 and the graph, $G = (V, E)$, is planar or chordal, $O(|E|)$ time suffices, due to a result by Perl and Shiloach [60]. Ohtsuki [55] and Shiloach [65] have also presented $O(|E| \times |V|)$ algorithms to find two disjoint paths in an undirected graph $G = (V, E)$. The problem of finding two *shortest* vertex disjoint paths between a pair of vertices in a directed graph can be solved in $O(|V|^2 \log |V|)$ time [69] and in $O(|E| \log_{(1+|E|/|V|)} |V|)$ time [31, 72].

When the k pairs of vertices lie on two specified face boundaries (one on the outer and one on the inner face) of a plane grapah with n vertices, Takahashi *et al.*[70] presented an $O(n \log n)$ time algorithm for the kNCP-SP problem, i.e., minimizing total path length. A similar problem, in which the routing region is a rectangular area containing r rectangular modules and the k terminal pairs lie on the boundaries of a specific module and the outer rectangle, (the shaded area in Figure 4), is also solved in time $O(n \log n)$, where $n = k + r$, by Takahashi *et al.*[71]. 2NCP-SMALLP^{pol} problem is considered by Yang, Lee and Wong [81]. The authors presented a linear time algorithm for finding the smallest pair inside a simple polygon if one exists. When a given problem instance does not contain a smallest pair, they find the *minimum bend shortest pair* or the *shortest minimum bend pair* using linear time and space.

3.8 Tables of Problems

In this section, we summarize in two tables all the *recent* results by the problem factors under consideration. For algorithms with alternative time complexities due to time-space trade-offs, only the best time complexity is listed in the table. Blank entries indicate that no result is known to the authors, and N/A stands for *not applicable*. The notation $\{\mathrm{x}\}_f^{(special\ case)}$ refers to item x described below with *special case*, if any, and time complexity f.

Numbers correspond to entries in Tables I and II.

[8]The query problem that finds the shortest geodesic (respectively minimum link distance) path in a simple polygon between two query points is solved in $O(\log e + k)$ time by Guibas and Hershberger [22] (respectively Suri [68]).

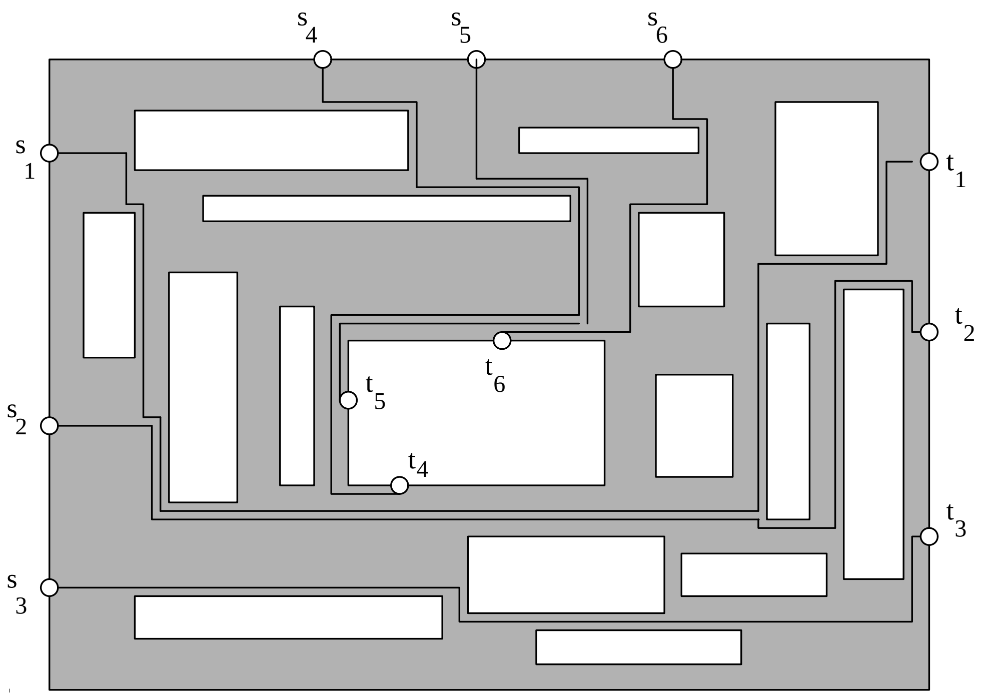

Figure 4: Shortest non-crossing paths in a plane region.

{**1**} SP$^\square$. $\theta(n \log n)$ time and $O(n)$ space algorithm by deRezende, Lee and Wu [15] using plane sweep approach due to the monotonicity of the goal path. Query$_1$ SP$^\square$ is also solved in $O(\log n)$ time with $O(n \log n)$ preprocessing time.

{**2**} SP. $O(e \log^{3/2} e)$ time and space (alternatively $O(e \log^2 e)$ time and $O(e \log e)$ space) algorithm by Clarkson, Kapoor and Vaidya [11] using graph-theoretic approach. Query$_1$ SP can be solved in $O(\log e)$ time with $O(e \log^{3/2} e)$ preprocessing time.

{**3**} SP. $O((e + m^2) \log m)$ time and $O(e + m^2)$ space algorithm by Wu, Widmayer, Schlag and Wong [77] using graph-theoretic approach, where m is the number of extreme edges of obstacles.

{**4**} SP. $O(e \log e)$ time and $O(e \log e)$ space algorithm by Mitchell [50] using 45-degree wave front approach.

{**5**} SP. $O(e \log m + m \log^{3/2} m)$ time and $O(e + m \log^{3/2} m)$ space (alternatively $O(e \log m + m \log^2 m)$ time and $O(e + m \log m)$ space) algorithm by Yang, Lee and Wong [82] using graph-theoretic approach, where m is the number of extreme edges of obstacles.

{**6**} MBP. $O(e \log^2 e)$ time and $O(e)$ space algorithm by Ohtsuki [57] using line searching approach. Adopting the data structure from Imai and Asano [30], the algorithm runs in $\theta(e \log e)$ time and space.

{**7**} MBSP$^\square$ and Weighted MBSP$^\square$. $\theta(n \log n)$ time and $O(n)$ space algorithm by Yang, Lee and Wong [83] using plane sweep approach and a monotonicity property of the goal paths.

Table I (objectives $\times$ problem models)

	rectilinear obstacles	weighted obstacles	on two layers 2-lyr distance	on two layers 1-lyr distance
sp	$\{1\}^{\square}_{\theta(n\log n)}$ $\{2\}_{O(e\log^{3/2} e)}$ $\{3\}_{O((e+m^2)\log m)}$ $\{4\}_{O(e\log e)}$ $\{5\}_{O(e\log m+m\log^{3/2} m)}$	$\{12\}^{\square}_{\theta(n\log n)}$ $\{13\}_{O(e\log^{3/2} e)}$	$\{14\}_{O(e\log^{3/2} e)}$	$\{15\}_{\theta(e\log e)}$
mbp	$\{6\}_{\theta(e\log e)}$	N/A	$\{6{,}14\}_{\theta(e\log e)}$	$\{6{,}14\}_{\theta(e\log e)}$
mbsp	$\{7\}^{\square}_{\theta(n\log n)}$ $\{8\}_{O(em+e\log e)}$ $\{10\}_{O(e\log^{3/2} e)}$	$\{7\}^{\square}_{\theta(n\log n)}$	$\{10{,}14\}_{O(e\log^{3/2} e)}$	$\{11{,}14\}_{\theta(e\log e)}$
smbp	$\{8\}_{O(em+e\log e)}$ $\{11\}_{\theta(e\log e)}$	N/A	$\{11{,}14\}_{\theta(e\log e)}$	$\{11{,}14\}_{\theta(e\log e)}$
mcp	$\{8\}_{O(em+e\log e)}$		$\{8{,}14\}_{O(e^2)}$	
bbsp	$\{8\}_{O(B(em+e\log e))}$		$\{8{,}14\}_{O(Be^2)}$	
blmbp	$\{8\}_{O(b^*(em+e\log e))}$		$\{8{,}14\}_{O(b^*e^2)}$	
kNCP-*sp*	$\{9\}^{\square *}_{O(n\log n)}$			

$*$ The k source-destination pairs lie on the boundaries of two rectangles.

{8} MBSP, SMBP, MCP, BBSP and BLMBP. $O(em+e\log e)$ time and $O(em)$ space algorithm by Yang, Lee and Wong [79] for finding *mbsp, smbp* and *mcp* using graph-theoretic approach, where m is the number of extreme edges of obstacles. $O(B(em+e\log e))$ time algorithm for BBSP, where $B = min\{b^*, b\}$, b is the given bound and b^* is the bend number of an *mbsp*. $O(b^*(em+e\log e))$ time algorithm for BLMBP. Adopting the slab tree structure from de Berg, *et al.*[13], Query$_1$ (MBSP|SMBP|MCP) can also be solved in $O(\log e)$ query time with $O(e^2)$ preprocessing.

{9} kNCP-*sp*$^{\square}$. $O(n\log n)$ time algorithm by Takahashi *et al.*[71], where $n = k + r$, and r denotes the number of rectangular modules inside a rectangle. (Figure 4.)

{10} MBSP, SMBP, MCP. $O(e\log^{3/2} e)$ time and space (alternatively $O(e\log^2 e)$ time and $O(e\log e)$ space) algorithm by Yang, Lee and Wong [82] using dynamic searching approach. Adopting the slab tree structure from de Berg, *et al.*[13], Query$_1$ (MBSP|SMBP|MCP) can also be solved in $O(\log e)$ query time with $O(e\log^{3/2} e)$ preprocessing.

{11} MBP, SMBP, 1D-SMBP, 1D-SP, and 1D-MBSP. $\theta(e\log e)$ time and $O(e\log e)$ space algorithm by Yang, Lee and Wong [80] using horizontal wave front approach.

{12} Weighted SP$^{\square}$. $\theta(n\log n)$ time and $O(n)$ space algorithm by Yang, Chen and Lee [78] using plane sweep approach and the monotonicity of the path.

{13} Weighted SP. $O(e\log^{3/2} e)$ time and space (alternatively $O(e\log^2 e)$ time and $O(e\log e)$ space) algorithm by Lee, Yang and Chen [38] using graph-theoretic approach.

{14} 2-Layer$_r$ X Problems. $O(e\log e)$ time algorithm by Lee, *et al.* [39] transforming problem

Table II (objectives $\times$ problem models)

Superscripts *pre*: preprocessing time, $\square$: rectangular obstacles, $Q_{1|2}$: query types.

	Query version given obstacles	within a simple rectilinear polygon	paths on given n segments (Manhattan Path)
sp	$\{1\}^{Q_1,\square,pre:O(n\log n)}_{O(\log n)}$ $\{2,16\}^{Q_1,pre:O(e\log^{3/2} e)}_{O(\log e)}$ $\{21\}^{Q_2,\square,pre:O(n^2)}_{O(\log n)}$	$\{17\}_{O(e)}$ $\{19\}^{Q_2,pre:O(e\log e)}_{O(\log e)}$	
mbp		$\{17\}_{O(e)}$ $\{19\}^{Q_2,pre:O(e\log e)}_{O(\log e)}$	$\{18\}_{\theta(n\log n)}$
smallest p		$\{17\}_{O(e)}$ $\{19\}^{Q_2,pre:O(e\log e)}_{O(\log e)}$	
mcp	$\{16\}^{Q_1,pre:O(e^2)}_{O(\log e)}$ $\{8,16\}^{Q_1,pre:O(e^2)}_{O(\log e)}$ $\{10,16\}^{Q_1,pre:O(e\log^{3/2} e)}_{O(\log e)}$		
mbsp or *smbp*	$\{8,16\}^{Q_1,pre:O(e^2)}_{O(\log e)}$ $\{10,16\}^{Q_1,pre:O(e\log^{3/2} e)}_{O(\log e)}$		
smallest 2NCP		$\{20\}_{O(e)}$	

instances in the two layer model into ones in one layer model. $O(e\log^2 e)$ time and $O(e\log e)$ space algorithm[39] for solving 2-Layer SP using graph-theoretic approach without problem transformation.

{15} 2-Layer$_r$ 1D-SP. $\theta(n\log n)$ time and $O(e)$ space algorithm by Ohtsuki and Sato [56] using graph-theoretic approach.

{16} Query$_1$ MCP. $O(\log e)$ query time by de Berg, van Kreveld, Nilsson and Overmars [13]. Obstacles are segments that may have intersection. It needs $O(e^2)$ preprocessing time and uses $O(e\log e)$ space for the slab tree data structure, which can be used to solve other query problems.

{17} SMALLP^{pol}. Linear time and space algorithm by McDonald and Peters [45]. It relies on the linear time polygon triangulation algorithm of Chazelle [8].

{18} Manhattan Path. $\theta(n\log e)$ time algorithm by Lipski [41].

{19} SMALLP^{pol}. $O(e\log e)$ time and space algorithm by de Berg [14], in which Query$_2$-SMALLP^{pol} is solved in $O(\log e)$ time.

{20} 2NCP-SMALLP^{pol}. Linear time and space algorithm by Yang, Lee and Wong [81].

{21} Query$_2$ SP$^{\square}$. $O(\log e)$ query time with $O(e^2)$ preprocessing time by Atallah and Chen [7] and by ElGindy and Mitra[17].

4 Problem-Solving Approaches

There are different approaches to solving rectilinear shortest paths in the presence of obstacles, such as maze running, line searching, graph-theoretic, wave front and dynamic searching, reported in the literature. We discuss them below in more details. Two methods applicable to many path finding problems, the 2-Layer to 1-Layer problem transformation and the slab tree structure construction, are also briefly discussed in this section.

4.1 Maze Running Approach

This approach, first proposed by Lee[35], basically propagates distance information from the source point by point along the grid constructed from the problem instance. While obstacles may block some grid lines, it is like searching for a path in a maze, and there comes the name 'maze running'. The propagation creates waves carrying the shortest distance from the source (see Figure 3). When the destination is hit, the shortest distance is known and the shortest path can be traced back. It is simple, easy to implement and easy to modify to find paths achieving other criteria, e.g. minimizing the number of bends. However, it needs $O(m^2)$ space to represent an $m \times m$ grid graph and has a worst case time complexity $O(l^2)$ for finding a path crossing l grid points.

Many improvements [1, 2, 3, 21, 28, 24, 62, 66] of the Lee algorithm were proposed but they only achieved a certain amount of speed-up for some cases without improving the asymptotic worse case time complexity. One of the improvements propagates simultaneously from both the source and destination points, and fuses the information when the waves from both sides meet. Another restricts the propagating area and enlarges it only when the path cannot be found within the area. Others give preferences to certain propagating directions so that the searching toward the destination may be sped up. We remark that another disadvantage of maze running algorithms is that modules and paths are restricted to be placed on a grid, resulting in a huge use of memory when high precision placement is required. The Lee algorithm [35] and these improvements were discussed in details in the survey of maze running algorithms and the line searching algorithms due to Ohtsuki [58]. Compared with the other approaches introduced in the following, maze running algorithms perform fairly well only when the goal paths are short.

4.2 Line Searching Approach

The line searching approach [46] is designed to speed up the maze running approach (see [58] for a survey). Since the maze running approach always marches point by point on grid, which is most time consuming, the line searching approach attempts to march along projecting lines passing through the obstacle boundaries [27, 57]. An apparent improvement over the maze running approach is that no underlying grid graph is constructed and therefore, the precision of modules and line placement do not affect the memory used by the algorithm. Furthermore when the obstacles are not very dense or of complex shape, the paths can be found much faster than running on grid.

Ohtsuki [57] used this approach to solve MBP in optimal time with the data structure by

Imai and Asano [30]. In his algorithm, we first generate so called *escape lines*, which are vertical or horizontal, from the source as those labeled '1' in Figure 5. In the next step any lines which go along the obstacle boundaries and intersect the existing escape lines will be considered new escape lines. As more escape lines are included, searching is performed by advancing from escape lines with smaller labels as in depth first search. When a line passing through the destination point is found to have intersection with any escape line, a *mbp* is found (the dotted path shown in Figure 5). The reason that it is sufficient to consider only those escape lines going along obstacle boundaries is that there always exists a goal path consisting only of segments aligning with obstacle boundaries.

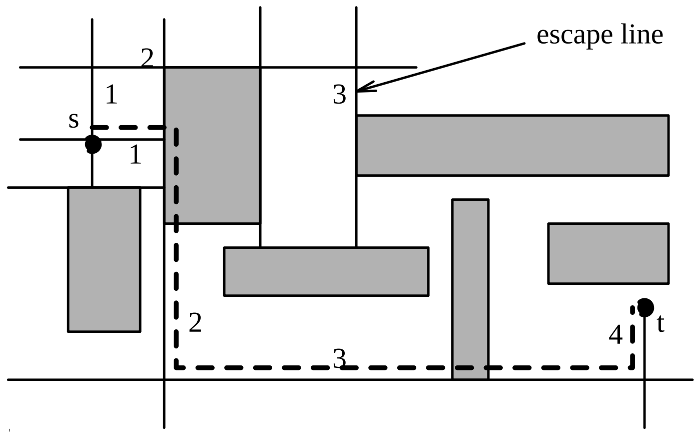

Figure 5: Line Searching approach.

Compared to the maze running algorithm, it does not run any faster if the number of obstacles is large. Furthermore, the data structure needed is more complex than a grid graph. Another drawback is that the escape lines carry no distance information and thus the algorithm cannot be used to find a shortest path.

4.3 Plane Sweep Approach

When the goal paths possess a special property called *monotonicity*, the plane sweep approach is very efficient to use. A path π is said to be $X(Y)$-*monotone* if there is no vertical (horizontal) line which has disjoint intersections with π. A *monotone* path is an X- or Y-monotone path [15].

When the goal path is monotone, one can use a sweep line to sweep over all the obstacles from s to t in the direction of its monotonicity and record necessary bends and length information on such a line. A data structure is associated with the sweep line so that queries about the shortest path from s to any point on the sweep line can be efficiently obtained. The data structure is usually a height balanced tree that provides logarithmic time for data accessing and updating. When the sweep line sweeps over an obstacle, the information on the data structure is updated. Finally when the destination is reached, the shortest distance can be obtained by performing a query on the data structure associated with the sweep line. When the obstacles are rectangular, many paths mentioned have such property. Utilizing the monotonicity property and the plane sweep approach, deRezende, Lee and Wu [15] and Yang, Chen and Lee [78] gave optimal $\theta(n \log n)$ time algorithms for $SP^{\square}$, and for weighted $SP^{\square}$ respectively. Recently Yang, Lee and Wong [83] also obtained optimal $\theta(n \log n)$ time algorithms for both $MBSP^{\square}$ and weighted $MBSP^{\square}$ using this approach.

4.4 Graph-Theoretic Approach

The graph-theoretic approach is based on a construction of a graph from the given obstacles in such a way that it is guaranteed that at least one goal path is embedded. Such a graph is called a *path preserving graph* [11]. A widely used path preserving graph, known as *visibility graph*, preserves the Euclidean shortest path among obstacles. Most of the Euclidean shortest path algorithms are based on the visibility graph (e.g., [4, 32, 59, 73]). After constructing such a graph, one can apply any suitable graph searching algorithm to find the goal path. Therefore, finding a path preserving graph as small in size as possible is the key to the efficiency of the algorithm. An advantage of the graph-theoretic approach is its generality. In addition to finding paths, one can use any graph based algorithms to find, e.g., spanning trees or other interesting structures in the graph with respect to various criteria without having to deal with obstacles [77, 82]. Three path preserving graphs by Clarkson, Kapoor and Vaidya [11], by Wu, Widmayer, Schlag and Wong [77] and by Lee, Yang and Wong [79] are introduced in the following.

In [10], Clarkson, *et al.* construct a graph for SP, which contains $O(e \log e)$ vertices and edges (Figure 6). (See also Widmayer[74].) The graph is for providing connections between pairs of points, p and q, in V (Figure 6a), whenever the closed rectangle, denoted R_{pq}, with $\overline{pq}$ as its diagonal contains neither the interior of any obstacle nor any other vertices of V, except p and q. It is shown that the graph constructed in this manner preserves a shortest path between any two points in V. A graph constructed by explicitly having an edge for every such pair of vertices has in the worst case $O(e^2)$ edges, implying a quadratic time algorithm for this method. A smaller graph, in terms of the numbers of vertices and edges, needs to be constructed to obtain a more efficient algorithm.

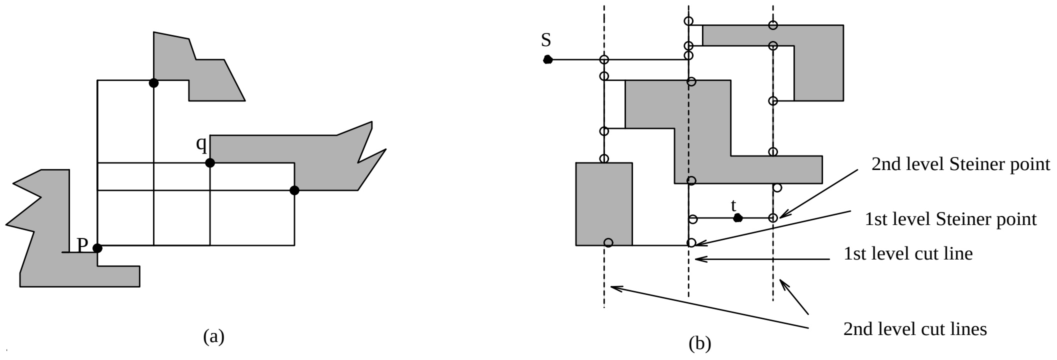

Figure 6: The path preserving graph by Clarkson, et al.

To reduce the size of the graph, *Steiner points* are introduced. As in Figure 6b, a vertical cut line is first placed at the median of the abscissae of all vertices. Vertices in V (recall that V is the set of obstacle vertices, the source and the destination) that are *visible* from the cut line generate projection points on the line (shown as hollow circles in Figure 6b), which are the *first level* Steiner points. By adding edges connecting these vertices to their projection points and between every two consecutive Steiner points on the cut line, it provides the needed *connection* between every pair of points p and q that lie on different sides of the cut line and whose associated R_{pq} is empty. Recursively cutting the vertices respectively on the left and right sides of the cut line, producing Steiner points at subsequent levels, will eventually provide all the necessary connections as needed. Note that there are $O(\log e)$ levels of recursion. Since each vertex can produce at most $O(\log e)$ Steiner points on the cut lines, and since edges are created along with the introduction of the Steiner points, there are $O(e \log e)$ edges and vertices in this path preserving graph.

Applying the $O(|E_G| + |V_G| \log |V_G|)$ time algorithm by Fredman and Tarjan [18] for finding a shortest path in $G = (V_G, E_G)$, one obtains an algorithm that runs in $O(e \log^2 e)$ time using $O(e \log e)$ space.[10]

Clarkson, *et al.*[11] later refined the graph further by decreasing the number of vertices at the expense of the number of edges so as to balance the time complexity in the searching algorithm. While generating Steiner points on a vertical cut line, they partition the plane into $O(\frac{e}{\sqrt{\log e}})$ horizontal strips such that each strip contains $O(\sqrt{\log e})$ vertices in V. In each strip, only two Steiner points (the highest and the lowest), which act as bridges for connections with vertices above and below that strip, are included in the vertex set of the graph. For any other pair of points, p, q in the same strip satisfying the conditions as described above, i.e., they are visible from the cut line and the associated R_{pq} is empty, an edge connecting them directly is included in the edge set of the graph. Consequently, since there are overall $O(e\sqrt{\log e})$ strips, we have $O(e\sqrt{\log e})$ vertices in this reduced path preserving graph. But the total number of edges becomes $O(e \log^{3/2} e)$ since each strip contains $O(\log e)$ edges. Such a path preserving graph thus yields an algorithm which runs in $O(e \log^{3/2} e)$ time and space. The same graph construction is also adopted by Lee, *et al.* [38] and the same time and space complexities were achieved for weighted SP.

On the other hand, Wu, *et al.* [77] derived an $O(e \log m + m^2 \log m)$ algorithm for finding a *sp* by constructing a *track graph.*

The *track graph* is constructed by generating *tracks* from the vertices on extreme edges, as in Figure 7. For each endpoint of an extreme edge, a horizontal and vertical track away from the obstacle corner are generated. The projected endpoints of these tracks when they hit an obstacle edge and the intersections among these tracks are made vertices of the track graph. The edges are the track segments between the intersections. Tracks from the source and the destination are then generated, and are stopped and glued to the graph at the first intersection with the tracks generated from extreme edges. The track graph construction is finally completed by adding edges connecting two consecutive projection points along the obstacle boundary. Consequently, the track graph contains $O(m^2 + e)$ edges and vertices and hence yields an $O(e \log m + m^2 \log m)$ algorithm for SP.

Recently, Yang, *et al.* [79] generated a graph preserving both the length and number of bends among unweighted obstacles (Figure 8). Their graph contains $O(em)$ edges and $O(e)$ vertices

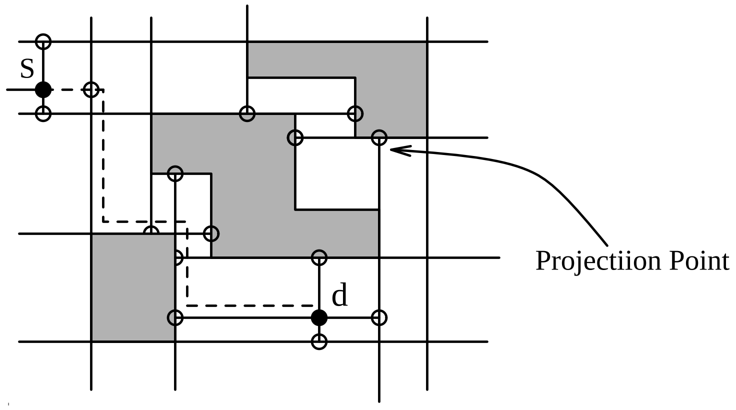

Figure 7: The track graph by Wu, et al.

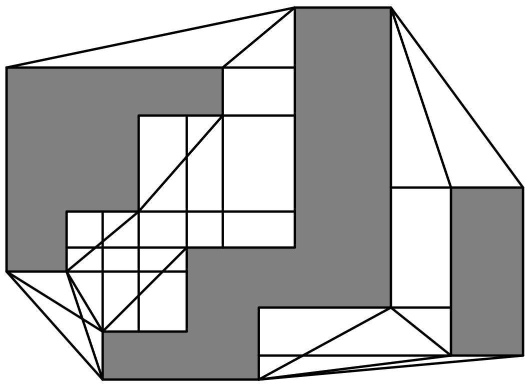

Figure 8: The path preserving graph by Lee, et al.

which yields an $O(em + e\log e)$ time algorithm for *sp, mbp, mbsp, smbp, mcp.* Basically, they generate tracks as in Wu *et al.*'s algorithm from all convex obstacle vertices. Instead of making all intersections the graph vertices, they directly make a connection between two convex obstacle vertices if the tracks from them intersect. Besides V, the projection points are the only extra graph vertices. With the edges connecting a convex obstacle vertex to its projected point and with the edges connecting two consecutive vertices and projection points along the obstacle boundary, the graph contains only $O(e)$ vertices and $O(em)$ edges. As shown in Figure 8, there are no vertices placed at the intersections of the tracks.

Ohtsuki [56] also solved 2-Layer$_r$ 1D-SP in $O(e\log e)$ time using this approach. Recently, Yang, *et al.* [82] presented a uniformly better $O(e\log m + m\log^{3/2} m)$ algorithm for SP, where they generate a smaller path preserving graph by combining the features of the path preserving graphs of Clarkson, *et al.* [11] and of Wu, *et al.* [77].

4.5 Wave Front Approach

In contrast to the graph-theoretic approach, which constructs an abstract model before searching, the 45-degree wave front approach proposed by Mitchell [47] tries to retain all the local information while searching and extend the process in a *continuous* manner. It is similar to the maze running approach, yet the wave front propagating structure is more subtle. It carries more information and invokes fewer and faster wave front propagations. The wave front propagation structure contains a set of segments with slopes of 1 or -1 (Figure 9). The algorithm keeps a number of growing loci of equal distances from the source and marches until the destination is reached. In the beginning, as in Figure 9a, the locus of points equidistant from the source is a diamond-shaped boundary, which can be represented by four segments going in different directions. Any point w on the segments has the same rectilinear distance from s. These segments are called the wave fronts.

Each wave front carries the distance information (an increasing entity) when it travels. Traveling of wave fronts is done by *segment dragging* operations, which find the first hit point and drag the wave front to it. An algorithm of Chazelle [9] for dragging vertical or horizontal segments is modified for dragging the 45-degree wave fronts. Wave fronts are subject to different changes; some become longer while dragging (as the wave fronts just start from s in Figure 9b), some become shorter (as the one toward r) and some remain the same (as the one heading right after passing through p); some can generate new wave fronts when they hit a boundary vertex (as q in Figure 9b) or can split into two different wave fronts when hitting corner vertices (as p in Figure 9b). A priority queue is used to store all wave fronts so that the wave front carrying the shortest distance from s among all wave fronts can be selected and dragged next. The shortest path is found when some wave front reaches the destination.

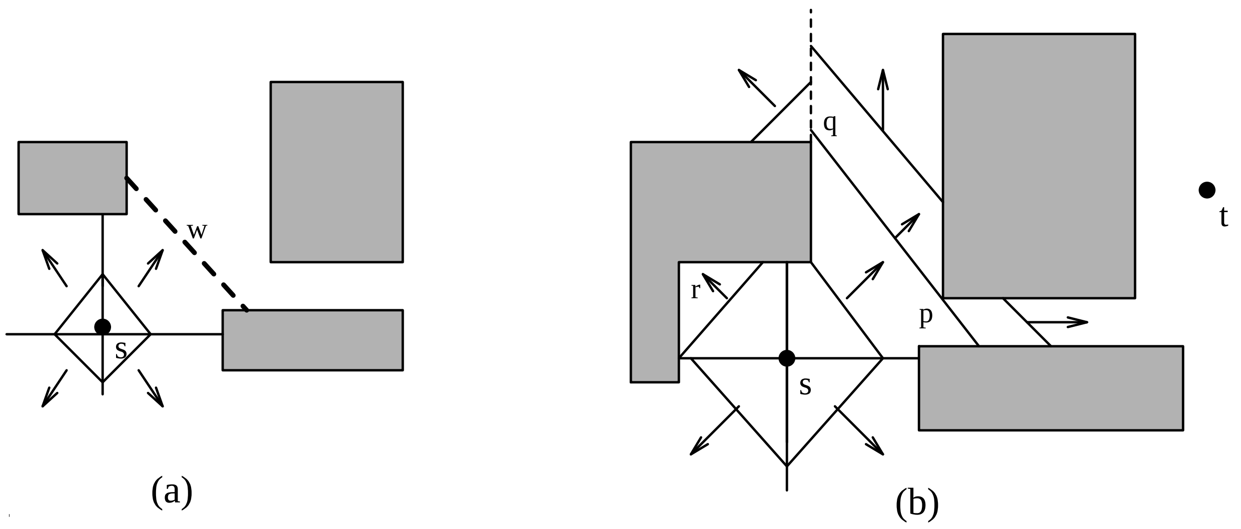

Figure 9: (a) Wave front segments launch from the source. (b) Motions of wave fronts.

Using this approach Mitchell [47] derived an $O(e \log^2 e)$ algorithm (and later an $O(e \log e)$ algorithm[50]) to solve SP. One advantage of this approach is due to its continuous searching, which can find the path quickly if s and t are close to each other. However, unless the paths between different pairs of points are queried repeatedly among the same set of obstacles, the advantage is almost overshadowed by the complex preprocessing step and dragging operations.

The complexity of the 45-degree wave front approach basically comes from its 45-degree shape. One has to partition the free space (routable region) into subdivisions so that the wave front movements can be simulated by segment dragging. For some problems, especially when the distance measure is not the primary factor to minimize (e.g., smbp) , one can adopt a simpler shape for the wave fronts, for example, horizontal wave fronts. A horizontal (vertical) wave front is the locus of points that can be reached from the source (refer to Figure 10) by a path with an equal number of bends. One can find an *mbp* path obviously from dragging these kinds of wave fronts. Distance information, if considered as a secondary factor other than bends, will be *fragmented* on the wave fronts. Dragging a horizontal (vertical) wave front needs no partitioning of routing regions. In [80], Yang, Lee and Wong first compute the horizontal or vertical projection points from all obstacle vertices and then implement the dragging by simply tracing those projection points along obstacle boundaries, as they are the *event* points where *critical* information on the wave fronts is modified. When the number of bends is the only or the primary factor to minimize, this approach is more attractive for its simplicity.

Yang, *et al.* [80] used this approach to solve SMBP in optimal $\theta(e \log e)$ time. They also applied this approach to solve assorted 1D-X problems where either the number of bends or the y-distance can all be carried properly by propagating the horizontal wave fronts. Note that a horizontal wave front can also be considered the locus of points with equal y-distance from s (Figure 10). Optimal $\theta(e \log e)$ algorithms are also obtained in [80] for 1D-SP, 1D-SMBP and 1D-MBSP.

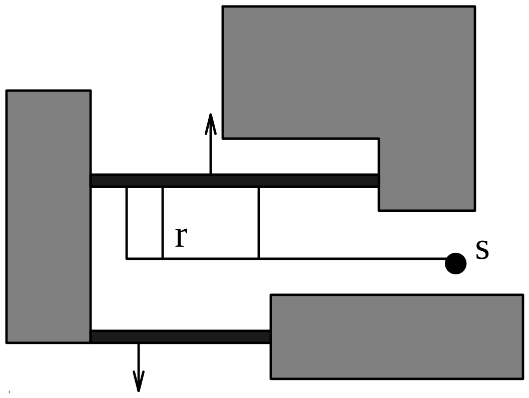

Figure 10: Interpretation of horizontal wave fronts.

4.6 Dynamic Searching Approach

A dynamic searching approach was proposed by Yang, *et al.* recently for solving MBSP, SMBP and MCP more efficiently [82] than their previous $O(em + e \log e)$ result given in [79].

Dynamic searching is not '*continuous*' as in a wave front approach, since it does not keep all the local information, a task that is much too complex. It is also not '*abstract*' as in graph-theoretic approach, since it does not compute a path preserving graph, a task that is too costly to do in terms of the size of the graph for all the assorted paths. A so-called *guidance graph* is generated, which, instead of preserving goal paths, is guaranteed to contain at least a path *homotopic* to a goal path from s to t (Figure 11). In other words, instead of preserving exactly the goal path, the guidance graph is constructed to preserve only the *homotopy* of at least one of the goal paths and is used just for navigating the searching task to be performed. By abandoning the path preserving property, one obtains a graph smaller in size than that constructed in [79] and hence one that can be generated more efficiently.

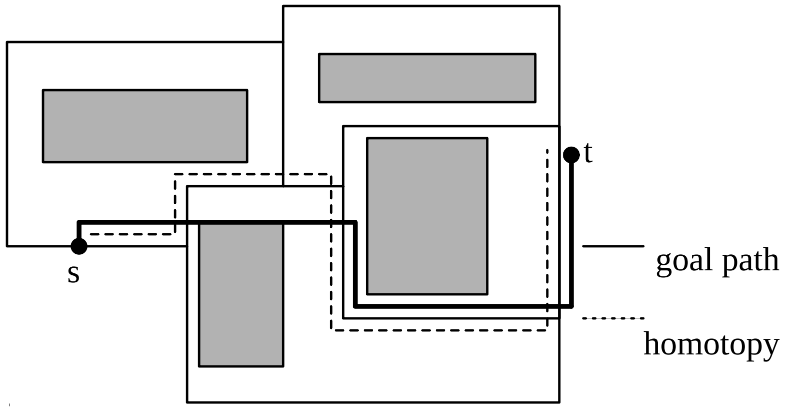

Figure 11: A guidance graph preserves the homotopy of a goal path.

The searching routine following the construction of the guidance graph has to find a goal path by altering on the fly a path in the graph, (shown in dotted lines in Figure 11) which is homotopic to the goal path (shown in thick lines in Figure 11). The path altering is done by *segment dragging* operations. Each path searched on the guidance graph is altered to be a partial goal path according to the search criteria. This is done by dragging segments on the path to be aligned to obstcles and eliminating unnecessary U-shaped subpaths. While searching on the graph, more than one partial goal path may arrive at a node. They are compared with the ones that are already stored at the node and those that are *dominated* by any existing partial path are discarded, and the *undominated* one is retained at the node. When the destination is reached, one of the partial goal paths retained is computed to be the solution. The kernel of the searching algorithm is Dijkstra's algorithm [16].

The *shortest* path preserving graph generated by Clarkson, *et al.* [11] can be used as the guidance graph to solve MBSP, SMBP, and MCP in $O(e \log^{3/2} e)$ time and space [82].

4.7 2-Layer to 1-Layer Problem Transformation

For corresponding problems in the two layer model, all the algorithms using the approaches mentioned previously, e.g., the graph-theoretic or the plane sweep approach, can be modified to work in this environment. However, the adjustments can be complicated and different from case to case. It would be very helpful if there were a common scheme that could solve two layer problems by adopting single layer algorithms. To this end, Lee, Yang and Wong [39] derived an inter-model transformation which transforms a problem instance (obstacles and points) in the two layer model to a problem instance in the single layer model regardless of the type of the goal path. With this transformation, most of the approaches and algorithms for single layer cases can be applied.

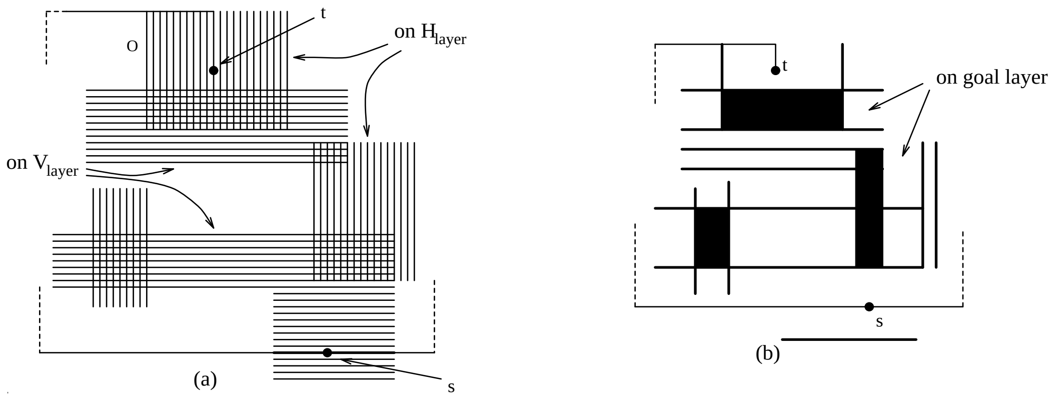

Figure 12: Skeleton (b) obtained from transformation of a problem instance (a) in the two layer model.

Recall that the two layer model considered by the authors assumes fixed routing orientation on each layer. They obtain a representation of obstacles on different layers which can be combined and put in a goal layer while the behavior (the blocking effect) of the obstacles is preserved. The representation, which is called the *skeleton* (Figure 12b) is a set of line segments loosely covering all the obstacles as defined below (refer to [39]).

First observe that an obstacle in H_{layer} is equivalent to a set of infinitely many *vertical* segments of *zero*-width covering the obstacle area (as obstacle *o* in Figure 12a). If one *copies* such a representation onto V_{layer}, it does not affect anything since all vertical segments of a path will not be blocked by these zero-width vertical segments. That is, such a representation of an obstacle on H_{layer} can be placed exactly at the same place in V_{layer} and it behaves exactly the same as it does on H_{layer}. Similarly one can represent each obstacle in V_{layer} by an infinite number of *zero*-width horizontal line segments covering its area. Now if we do this transformation to each obstacle and place all of those representations on a common layer, called the *goal layer* (denoted as G_{layer}), we will have an instance (called a *sketch*) that is equivalent to the original one (as in Figure 12a). Since one obviously cannot afford to represent an obstacle in the two layer model by an *infinite* number of obstacle segments in one layer model, an equivalent representation is further sought from the *sketch*. It turns out that a *finite* representation, called the *skeleton*, can be obtained if certain properties (*alignment property* — see [39] for details) of the goal paths can be enforced.

The *skeleton* of a set of obstacles given in the two layer model is defined as follows:

(1) The *skeleton* is a subset of the sketch.
(2) Any maximal vertical (respectively horizontal) segment which is covered by an obstacle on H_{layer} (respectively V_{layer}) and contains an obstacle vertex is in the *skeleton*.
(3) Any maximal vertical segment covered by an H_{layer} obstacle or maximal horizontal segment covered by a V_{layer} obstacle, that passes through an endpoint of a skeleton segment is in the *skeleton*.
(4) No other segment is in the *skeleton*.

The skeleton is a set of line segments. Its contour, which can be found in $O(n \log n)$ time for n line segments as in [61], has been shown to represent those obstacles adequately for most of the path finding problems. The existing algorithms can then be applied to the representation of the obstacles on the combined goal layer, and the resultant path transformed back to the goal path.

4.8 Slab Tree Structure for Query Problems

Given a set of n horizontal or vertical line segments, de Berg, van Kreveld, Nilsson and Overmars [13] showed that one can first find an *mcp* from the source to all endpoints of the line segments and then construct a slab tree data structure for subsequent path queries. The slab tree structure will be described below. The structure is of size $O(n \log n)$ and can thereafter in $O(\log n)$ time support queries to find an *mcp* from a fixed source to any query destination point. The only necessary condition for applying the slab tree as a preprocessed data structure for later queries is that there *exists* a goal path, π_{st}, from the source, s, to the query destination, t, such that π_{st} can be decomposed into two parts, π_{sp} and π_{pt}, with p being one of the endpoints of the line segments, and with π_{pt} being a vertical, horizontal or L-shaped path segment. In other words, on π_{st}, t can connect to one of the endpoints by a segment with no more than one bend. This is true for *sp, mp, mcp, mbsp* and *smbp.* One can therefore adopt this useful data structure as a post-processed structure for many path finding algorithms and use it as the preprocessed structure to support queries in logarithmic time.

The last one-bend-at-most segment from a vertex to t is called a *simple step.* There are only a constant number of possible shapes for a simple step. For example, it can go left and then turn downward from t to reach a vertex (called *left-down* simple step). The algorithm first partitions the space in a way such that a goal path from s to t ending with a simple step of a certain shape can be answered in one query. After creating different partitions for all possible shapes of the last segment, the solution can be obtained after finding the goal paths from the source to all the vertices, creating the slab tree and then answering a constant number of queries, each for one possible shape of the last segment.

Consider the case for finding the best one among paths with a left-down simple step. As in the lower portion of Figure 13, the space is conceptually cut into vertical slabs by drawing vertical lines through every endpoint such that each slab defined by two consucutive vertical lines corresponds to a leaf of the slab tree[9].

[9]This *vertical* slab tree is used for the cases where the simple step has a horizonal segment followed possibly by a vertical segment. For other cases in which the simple step has a vertical segment followed possibly by a horizontal segment we'll use a *horizontal* slab tree.

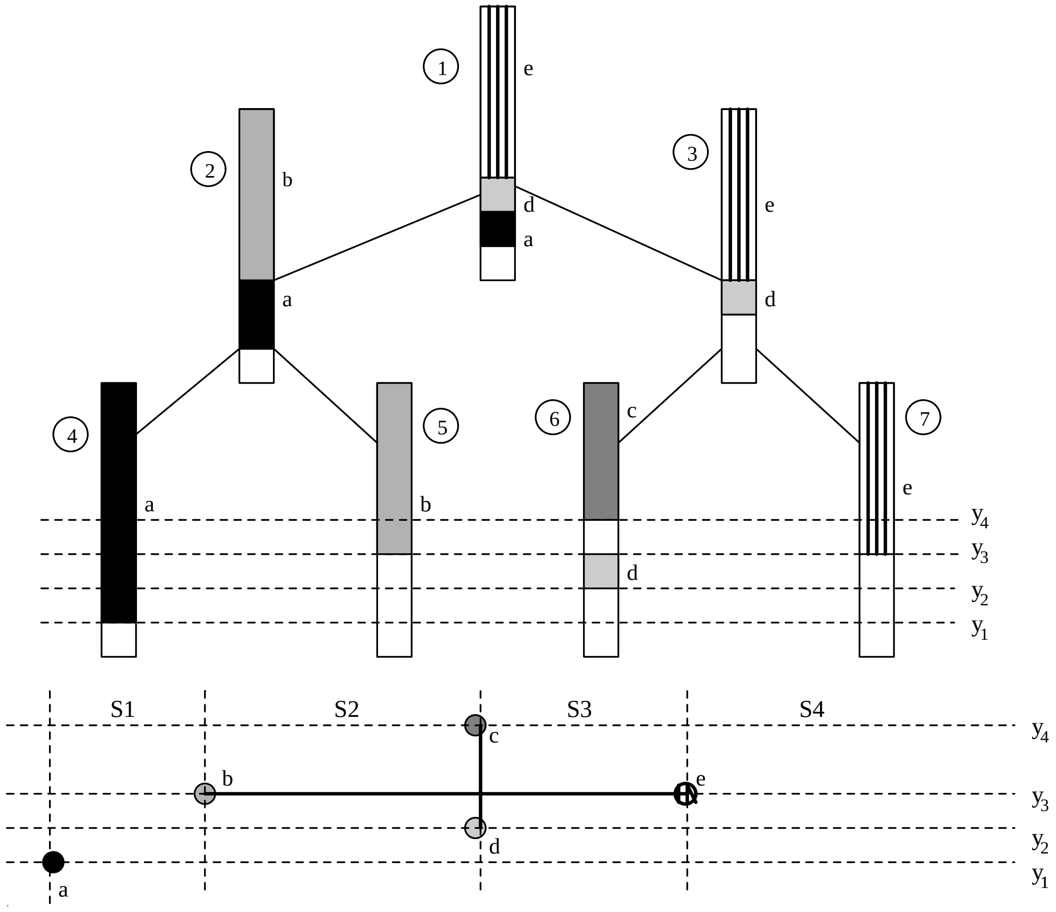

Figure 13: A slab tree for the left-down simple step.

The slab tree is a data structure where each node of the tree represents the combined slab of its two children. Associated with each node in the slab tree is a list of ordinates, called the *associated list* which stores the best vertex to connect to, if t falls in different ordinate ranges in this slab. In Figure 13, lists are represented by different shading patterns inside the bar. The white shading indicates that no such path with a left-down simple step exists.

Consider the example shown in Figure 13. There are four slabs corresponding to four leaves labeled 4,5,6 and 7. The list in leaf 4, which contains only two elements shaded black and white, shows that if the query t is in slab 1 with ordinate greater than y_1, then the best path is from t to a and then follows a pre-computed best path from a to s. At a non-leaf node we compute and store a list obtained from the associated lists of its children. This is to store the best vertex over the slabs corresponding to its children. For example, node 3 contains a list by comparing the lists in node 6 and node 7. Comparisons are made based on the criterion that the better vertex for the path with the simple step is recorded.

Therefore for a query point we first identify a maximal horizontal segment emanating from the query leftward to the first segment that it hits. This horizontal segment defines the range of the x-coordinates of all possible endpoints that the desired path may visit last. We then use the horizonal segment as a query interval to traverse down the slab tree along the root-to-leaf path leading to the leaf slabs containing the two endpoints of the segment. For those non-leaf nodes whose associated slabs are totally contained in the query interval, the associated lists are consulted to find the best vertex. As there are at most $O(\log n)$ candidate vertices, the solution can be obtained in $O(\log n)$ time.

de Berg, *et al* [13] showed that such a data structure takes only $O(n \log n)$ space and can be built in $O(n \log n)$ time once all the necessary information (i.e., the best path from s to all the obstacle vertices) is computed.

5 Research Directions and Open Problems

Whether or not there is a $\theta(e \log e)$ time algorithm for finding *mbsp* and *mcp* remains to be seen. Algorithms with less preprocessing time for Query_1 SP or Query_1 MCP are worth searching for. Solving Query_2 SP in linear or sublinear query time would be very useful and practical. Algorithms solving these problems dynamically where obstacles can be deleted or inserted are crucial in iterative routing and would be of great importance. This type of problem, called the *dynamic version* of the path finding problem, has not been addressed.

Weighted rectilinear obstacles in previous studies (namely in [78]) are non-adjacent. That is, two obstacles can be as close to each other as possible and yet they always admit a path going in between without paying any costs. It would be nice to eliminate this assumption to allow adjacent weighted obstacles, as in the case studied by Mitchell and Papadimitriou[52].

Finding paths in multi-layers is of interest since multiple layers have been used also as a routing model. The paths on each layer have different resistance and can be modeled by assigning different weights to the layers. (Recall that we have studied the case where paths on one of the two layers in the two layer model have *zero* weight, e.g., the y-distance.) In this model, some of the layers may have fixed orientation routing and others may allow free routing (e.g., for a ground net). A general scheme for solving problems on multi-layers will be very useful.

Although the general problem of solving kNCP-SP for arbitrary k is NP-hard, a solution to the problem for fixed $k \geq 3$ would have both theoretical and practical values. Note that the time and space requirements for finding k paths between k pairs of given points, $k \leq 2$, within a simple polygon are linear, as shown in [81]. The problem is to determine how many paths at a time, i.e., the value of k, such that k paths can be routed and the objective function optimized within reasonable time and space bounds, either inside a simple polygon or among a set of obstacles. Generalization of the results of Takahashi *et al.*[71] to cases where the k source-destination pairs can lie on multiple modules, instead of just on the outermost module and *one* inner module, will be of great interest.

We list some of these interesting open problems as follows.

Optimal Algorithms for MBSP and MCP Find $\theta(e \log e)$ algorithms for MBSP and MCP.

Adjacent Obstacles in Weighted Case Find assorted paths among weighted obstacles where obstacles can be adjacent in such a way that a path going in between two adjacent obstacles pays a certain amount of cost.

Linear or Sublinear Algorithms for Query$_2$ Versions With preprocessing allowed find the assorted paths between two query points in linear or sublinear time.

Improved Algorithms for BBSP and BLMBP Find more efficient algorithms for BBSP or BLMBP.

Multi-layer nD-X Problem Formulate and solve path finding problems in multi-layer model where the routing orientation may or may not be fixed on different layers and the paths on different layers may have different weights.

kNCP-Z Problem Find k non-crossing paths among obstacles or within a polygon with assorted optimization factors of the paths, where k is a small constant.

6 Remarks and Conclusion

In this paper, we have addressed a class of problems pertaining to finding rectilinear paths between two points in the presence of rectilinear obstacles with respect to different criteria. Finding paths between two terminals is a basic building block in finding more complicated geometric structures among modules in VLSI design. The minimum spanning tree of a set of terminals in the presence of modules, for example, can be constructed by adopting various path finding algorithms [77, 82]. Furthermore the minimum spanning tree is widely used for constructing a Steiner tree which plays an important role in global or detailed routing for multi-terminal nets (see [76, 29] for Steiner tree surveys). More complex Steiner trees are studied where more factors are considered [63]. The more one can control the parameters involved in the single path routing, the more likely one can construct a routing scheme where all parameters are well balanced.

Acknowledgment

The authors would like to thank K. P. Bogart and the anonymous referees for their careful reading and comments of the earlier version of the manuscript. Their suggestions have helped improve the readability of the manuscript.

References

[1] L. C. Abel, "On the ordering of connections for automatic wire routing," *IEEE Trans. on Comput.*, Vol.C-21, 1972, 1227-1233.

[2] S. B. Akers, "A modification of Lee's path connection algorithms," *IEEE Trans. on Electronic Computers*, Vol. EC-16, 1967, 97-98.

[3] S. B. Akers, "Routing," in *Design Automation of Digital Systems*, Vol.1, Breuer, M.A. (Ed), 283-333, Prentice Hall, NJ 1972.

[4] T. Asano, T. Asano, L. Guibas, J. Hershberger, H. Imai, "Visibility of disjoint polygons," *Algorithmica,* 1, 1986, 49-63.

[5] T. Asano, "Generalized Manhattan path algorithm with applications," *IEEE Trans. Computer-Aided Design*, 7, 1988, 797-804.

[6] T. Asano, M. Sato and T. Ohtsuki, "Computational geometry algorithms," in Layout Design and Verification, (T. Ohtsuki Ed.), North-Holland, 1986, 295-347.

[7] M. J. Atallah and D. Z. Chen, "Parallel rectilinear shortest paths with rectangular obstacles," *Computational Geometry: Theory and Applications*, V.1, No.2, 1991, 79-113.

[8] B. Chazelle, "Triangulating a simple polygon in linear time," *Discrete & Computational Geometry*, 6 1991, 485-524.

[9] B. Chazelle, "An algorithm for segment dragging and its implementation," *Algorithmica*, 3, 1988, 205-221.

[10] K. L. Clarkson, S. Kapoor and P. M. Vaidya, "Rectilinear shortest paths through polygonal obstacle in $O(n \log^2 n)$ time" *Proc. 3rd ACM Symp. on Computational Geometry*, 1987, 251-257.

[11] K. L. Clarkson, S. Kapoor and P. M. Vaidya, "Rectilinear shortest paths through polygonal obstacles in $O(n \log^{3/2} n)$ time," unpublished manuscript.

[12] W. W. Dai, R. Kong, J. Jue and M.Sato, "Rubber band routing and dynamic data representation," *IEEE International Conf. on Computer Aided Design*, 1991, 52-55.

[13] M. de Berg, M. van Kreveld, B. J. Nilsson, M. H. Overmars, "Shortest path queries in rectilinear worlds," *Int'l J. Computational Geometry & Applications*, 3,2 Sept. 1992, 287-309.

[14] M. de Berg, "On rectilinear link distance," *Computational Geometry: Theory and Applications*," 1,1, 1991, 13-34.

[15] P. J. deRezende, D. T. Lee and Y. F. Wu, "Rectilinear shortest paths with rectangular barriers," *Discrete & Computational Geometry*, 4, 1989, 41-53.

[16] E. W. Dijkstra, "A note on two problems in connection with graphs," *Numerische Mathematik*, 1, 269.

[17] H. ElGindy and P. Mitram "Orthogonal shortest route queries among axes parallel rectangular obstacles," *Int'l J. Computational Geometry & Applications,* 4,1 March 1994, 3-24.

[18] M. L. Fredman and R. E. Tarjan, "Fibonacci heaps and their uses in improved network optimization algorithms," *J. ACM*, 34,3, July 1987, 596-615.

[19] M. L. Fredman and D. E. Willard, "Trans-dichotomous algorithms for minimum spanning trees and shortest paths," *Proc. 31st IEEE Symp. on Foundations of Computer Science*, 1990, 719-725.

[20] M. R. Garey and D. S. Johnson, Computers and Intractability. A Guide to the Theory of NP-Completeness, W. H. Freeman, San Francisco, 1979, 214.

[21] J. M. Geyer, "Connection routing algorithm for printed circuit boards," *IEEE Trans on Circuit Theory*, Vol.CT-18, 1971, 95-100.

[22] L. J. Guibas and J. Hershberger, "Optimal shortest path queries in a simple polygon," *J. Comput. Syst.Sci.*, 39, 1989, 126-152.

[23] Guibas, L., J. Hershberger, D. Leven, M Sharir and R. Tarjan, "Linear time algorithms for visibility and shortest path problems inside simple polygons," *Algorithmica* 2, 1987, 209-233.

[24] F. O. Hadlock, "A shortest path algorithm for grid graphs," *Networks*, Vol.7, 1977, 323-334.

[25] W. Heyns, W. Sansen, and H. Beke, "A line-expansion algorithm for the general routing problem with a guaranteed solution," *Proc. 17th Design Automation Conf.*, 1980, 243-249.

[26] J. Hershberger and J. Snoeyink, "Computing minimum length paths of a given homotopy class," *Computational Geometry: Theory and Applications,* 4,2 June 1994, 63-97.

[27] D. W. Hightower, "A solution to line-routing problem on the continuous plane," *Proc. 6th Design Automation Workshop,*, 1969, 1-24.

[28] J. H. Hoel, "Some variations of Lee's algorithm," *IEEE Trans. on Comput.*, Vol.C-25, 1976, 19-24.

[29] F. K. Hwang, D. S. Richards and P. Winter, Steiner tree problems, North-Holland, 1992.

[30] H. Imai and T. Asano, "Dynamic orthogonal segment intersection search," *J. Algorithms,* 8 1987, 1-18.

[31] D. B. Johnson, "Efficient algorithms for shortest paths in sparse networks," *J. ACM*, 24, 1977, 1-13.

[32] S. Kapoor and S. N. Maheshwari, "Efficient algorithms for Euclidean shortest path and visibility problems with polygonal obstacles," *Proc. Fourth Annual ACM Symp. on Computational Geometry*, 1988, 172-182.

[33] Y. Ke, "An efficient algorithm for link-distance problems," *Proc. 5th ACM Sympo. on Computational Geometry*, 1989, 69-78.

[34] R. C. Larson and V. O. Li, "Finding minimum rectilinear distance paths in the presence of barriers," *Networks*, 11, 1981, 285-304.

[35] C. Y. Lee, "An algorithm for path connections and its application," *IRE Trans. on Electronic Computers*, V.EC-10, 1961, 346-365.

[36] D. T. Lee, "Proximity and reachability in the plane," PhD. Dissertation, University of Illinois, 1978.

[37] D. T. Lee and F. P. Preparata, "Euclidean shortest paths in the presence of rectilinear barriers," *Networks*, 14, 1984, 393-410.

[38] D. T. Lee, C. D. Yang and T. H. Chen, "Shortest rectilinear paths among weighted obstacles," *Int'l J. Computational Geometry & Applications*, 1,2, 1991, 109-124.

[39] D. T. Lee, C. D. Yang and C. K. Wong, "Problem transformation for finding rectilinear paths among obstacles in two-layer interconnection model", Tech. Report 92-AC-104, Dept. of EECS, Northwestern Univ., Jan. 1992.

[40] W. Lipski, "Finding a Manhattan path and related problems," *Networks*, 13, 1983, 399-409.

[41] W. Lipski, "An $O(n \log n)$ Manhattan path algorithm," *Inform. Process. Lett.*, 19, 1984, 99-102.

[42] Lenhart, R. Pollack, J. Sack, R. Seidel, M. Sharir, S. Suri, G. Toussaint, S. Whitesides and C. Yap, "Computing the link center of a simple polygon," *Discrete & Computational Geometry*, 3,3 1988, 281-293.

[43] T. Lozano-Perez and M. A. Wesley, "An algorithm for planning collision-free paths among polyhedral obstacles," *Comm. ACM*, 22, 1979, 560-570.

[44] A. Margarino, A. Romano, A. De Gloria, F. Curatelli and P. Antognetti, "A tile-expansion router," *IEEE Trans. CAD*, Vol.CAD-6, no.4, 1987, 507-517.

[45] K. M. McDonald and J. G. Peters, "Smallest paths in simple rectilinear polygons," *IEEE Trans. CAD*, 11,7 July 1992, 864-875.

[46] K. Mikami and K. Tabuchi, "A computer program for optimal routing of printed circuit connectors," *IFIPS Proc.*, Vol. 47, 1968, 1475-1478.

[47] J. S. B. Mitchell, "L_1 Shortest paths among polygon obstacles in the plane". *Algorithmica*, 8,1, 1992, 55-88.

[48] J. S. B. Mitchell, "A new algorithm for shortest paths among obstacles in the plane," *Annals of Math. and Artificial Intelligence*, 3, 1991, 83-106.

[49] J. S. B. Mitchell, "An optimal algorithm for shortest rectilinear paths among obstacles," *Abstracts of First Canadian Conference on Computational Geometry*, 1989, 22.

[50] J. S. B. Mitchell, "Shortest paths among obstacles in the plane", *Proc. 9th ACM Symp. on Comput. Geometry,*, May 1993, 308-317.

[51] J. S. B. Mitchell, C. Piatko, and E. M. Arkin, "Computing a shortest k-link path in a polygon," *Proc. 33rd Annual Symp. Foundations of Computer Science,* Oct. 1992, 573-582.

[52] J. S. B. Mitchell and C. H. Papadimitriou, "The weighted region problem: finding shortest paths through a weighted planar subdivision", *J. ACM*, 38,1, January 1991, 18-73.

[53] J. S. B. Mitchell, G. Rote and G. Wōginger, "Minimum link path among obstacles in the planes," *Algorithmica*, 8 1992, 431-459.

[54] E. F. Moore, "The shortest path through a maze," *Annals of the Harvard Computation Laboratory*, Vol.30, Pt.II, 1959, 185-292.

[55] T. Ohtsuki, "The two disjoint path problem and wire routing design," *Lecture Notes in Computer Science*, 108, Springer-Verlag, 1981, 207-216.

[56] T. Ohtsuki and M. Sato, "Gridless routers for two layer interconnection," *IEEE International Conference on Computer Aided Design*, 1984, 76-78.

[57] T. Ohtsuki, "Gridless routers — new wire routing algorithm based on computational geometry," *Int'l Conference on Circuits and Systems*, China, 1985.

[58] T. Ohtsuki, "Maze running and line-search algorithms," in *Layout Design and Verification*, (T. Ohtsuki Ed.), North-Holland, 1986, 99-131.

[59] M. H. Overmars and E. Welzl, "New methods for computing visibility graphs," *Proc. Fourth Annual ACM on Computational Geometry*, 1988, 164-171.

[60] Y. Perl and Y. Shiloach, "Finding two disjoint paths between two pairs of vertices in a graph," *J. ACM*, 25,1, 1978, 1-9.

[61] F. P. Preparata and M. I. Shamos, Computational Geometry, Springer-Verlag, NY, 1985.

[62] F. Rubin, "The Lee path connection algorithm," *IEEE Trans. on Comput.*, C-23, 1974, 907-914.

[63] M. Sarrafzadeh and C. K. Wong, An Introduction to VLSI Physical Design, McGraw Hill, 1995.

[64] M. Sato, J. Sakanaka and T. Ohtsuki, "A fast line-search method based on a tile plane," in *Proc. IEEE ISCAS*, 1987, 588-591.

[65] Y. Shiloach, "A polynomial solution to the undirected two paths problem," *J. ACM.*, 27,3 1980, 445-456.

[66] J. Soukup, "Fast maze router," *Proc. 15th Design Automation Conf.*, 100-102, 1978.

[67] S. Suri, "A linear time algorithm for minimum link paths inside a simple polygon," *Computer Vision, Graphics and Image Processing,* 35, 1986, 99-110.

[68] S. Suri, "On some link distance problems in a simple polygon," *IEEE Trans. on Robotics and Automation*, 6, 1990, 108-113.

[69] J. W. Suurballe, "The single-source, all-terminals problem for disjoint paths," Unpublished technical memorandum, Bell Laboratories, 1982.

[70] J. Y. Takahashi, H. Suzuki and T. Nishizeki, "Algorithms for finding non-crossing paths with minimum total length in plane graphs" *Proc. Int'l Symp. on Algorithms and Computation*, Dec. 1992, 400-409.

[71] J. Y. Takahashi, H. Suzuki and T. Nishizeki, "Finding shortest non-crossing rectilinear paths in plane regions," *Proc. Int'l Symp. on Algorithms and Computation*, Dec. 1993, 98-107.

[72] R. E. Tarjan, Data Structures and Network Algorithms, Soc. Ind. Appl. Math. 1983.

[73] E. Welzl, "Constructing the visibility graph for n line segments in $O(n^2)$ time," *Info. Proc. Lett.*, 1985, 167-171.

[74] P. Widmayer, "Network design issues in VLSI", unpublished manuscript, 1989.

[75] P. Widmayer, Y. F. Wu, C. K. Wong, "On some distance problems in fixed orientations," *SIAM J. on Computing*, 16, 1987, 728-746.

[76] P. Winter, "Steiner problem in networks: A survey," *Networks*, Vol.17, 1987, 129-167.

[77] Y. F. Wu, P. Widmayer, M. D. F. Schlag, and C. K. Wong, "Rectilinear shortest paths and minimum spanning trees in the presence of rectilinear obstacles," *IEEE Trans. on Comput.*, 1987, 321-331.

[78] C. D. Yang, T. H. Chen and D. T. Lee, "Shortest rectilinear paths among weighted rectangles," *J. of Information Processing*, 13,4, 1990, 456-462.

[79] C. D. Yang, D. T. Lee and C. K. Wong, "On bends and lengths of rectilinear paths: A graph-theoretic approach," *Int'l J. of Computational Geometry & Applications*, 2,1, March 1992, 61-74.

[80] C. D. Yang, D. T. Lee and C. K. Wong, "On bends and distances of paths among obstacles in two-layer interconnection model," *IEEE Trans. on Computers,* Vol 43, No.1, June 1994, 711-724.

[81] C. D. Yang, D. T. Lee and C. K. Wong, "Smallest pair of non-crossing paths in a simple rectilinear polygon", Tech. Report 92-AC-109, Dept. of EECS, Northwestern Univ., March 1992.

[82] C. D. Yang, D. T. Lee and C. K. Wong, "Rectilinear path problems among rectilinear obstacles revisited," *SIAM J. on Computing,*, 24,3 June 1995, 457-472.

[83] C. D. Yang, D. T. Lee and C. K. Wong, "On minimum-bend shortest rectilinear path among weighted rectangles," Tech. Report 92-AC-122, Dept. of EECS, Northwestern Univ., Aug. 1992.