..., Eds. *Handbooks in OR & MS, Vol. 1*

Chapter 1

A Survey of Computational Geometry

Joseph S. B. Mitchell

Applied Math, SUNY, Stony Brook, NY 11794-3600. Email: jsbm@ams.sunysb.edu.

Subhash Suri

Bellcore, 445 South Street, Morristown, NJ 07960. Email: suri@bellcore.com

1. Introduction

Computational geometry takes an algorithmic approach to the study of geometrical problems. The principal motivation in this study is a quest for "good" algorithms for solving geometrical problems. Of course, several practical and aesthetic factors determine what one means by a good algorithm, but the general trend has been to associate "goodness" with the asymptotic efficiency of an algorithm in terms of the time and space complexity. Lately, however, the implementational ease and robustness also are becoming increasingly important considerations in the algorithm design.

Although many geometrical problems and algorithms were known before, computational geometry evolved into a cohesive discipline only in the mid to late seventies. An important event in this development was the publication of the Ph.D. thesis of M. Shamos [227] in 1978. During its first decade, the field of computational geometry grew enormously as its fundamental structures were applied to a vast variety of problems in diverse disciplines, and many new tools and techniques were developed. In the process, new insights were gained into inter-relationships among some of these fundamental structures, which also led to a unification and consolidation of several disparate sets of ideas. In the last five or so years, the field has matured significantly, both in mathematical depth as well as in algorithmic ideas.

Computational geometry has had strong interactions with other fields, in mathematics as well as in applied computer science. A particularly fruitful interplay has taken place between computational geometry and combinatorial geometry. The latter is a branch of mathematics concerned primarily with the "counting" of certain geometric structures. Examples include counting the maximum possible number of

incidences between a set of lines and a set of points, and counting the number of lines that bisect a set of points. Both fields seem to have benefited from each other: combinatorial bounds for certain structures have been obtained by analyzing an algorithm that enumerates them and, conversely, the analysis of algorithms often depends crucially on the combinatorial bound on some geometric objects.

The field of computational geometry has also benefited from its interactions with other disciplines within computer science such as VLSI, database theory, robotics, computer vision, computer graphics, pattern recognition and learning theory. These areas offer a rich variety of problems that are inherently geometrical.

Due to its interconnections with many applications areas, the variety of problems studied in computational geometry is truly enormous. Our goal is this paper is quite modest: we survey state of the art in some selected areas of computational geometry, with a strong bias towards problems with an optimization component. In the process, we also hope to acquaint the reader with some of the fundamental techniques and structures in computational geometry.

Our paper has seven main sections. The survey proper begins in Section 3, while Section 2 introduces some foundational material. In particular, we briefly describe five key concepts and fundamental structures that permeate much of computational geometry, and therefore are somewhat essential to a proper understanding of the material in later sections. The structures covered are convex hulls, arrangements, geometric duality, Voronoi diagram, and point location data structures. The main body of our survey begins with Section 3, where we describe four popular geometric graphs: minimum and maximum spanning trees, relative neighborhood graphs, and Gabriel graphs. Section 4 is devoted to algorithms in path planning. The topic of path planning is a vast one, with problems ranging from finding shortest paths in a discrete graph to deciding the feasible motion of a complex robot in an environment full of complex obstacles. We briefly mention most of the major developments in path planning research over the last two decades, but to a large extent limit ourselves to issues related to shortest paths in a planar domain. In Section 5, we discuss the matching and the traveling salesman type problems in computational geometry. Section 6 describes results on a variety of problems related to shape analysis and pattern recognition. We close with some concluding remarks in Section 7. In each section, we also pose what in our opinion are the most important and interesting open problems on the topic. There are altogether twenty open problems in this survey.

2. Fundamental Structures

2.1. Convex Hulls

The *convex hull* of a finite set of points S is the smallest convex set containing S. In two dimensions, for instance, the convex hull is the smallest convex polygon containing all the points of S; see Figure 2.1.1 for an example. In higher dimensions, the convex hull is a polytope. Before we discuss the algorithms for computing

a convex hull, we must address the question of representing it. There are several representations of a convex hull, depending upon how many features of the corresponding polytope are described. In the simplest representation, we may only store the vertices of the convex hull. The other extreme of the representation is the *face lattice*, which stores all the faces of the convex hull as well as the incidence relationships among the faces. The intermediate forms of representation may store faces of only certain dimensions, such as the $(d-1)$-dimensional faces, also called the *facets*. The differences among these representations become significant only in dimensions $d \geq 4$, where the full lattice may have size $\Theta(n^{\lfloor d/2 \rfloor})$ while the number of vertices is obviously at most n. (Grübaum's book [112] is an excellent source for information on polytopes.)

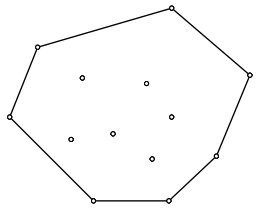

Fig. 2.1.1. A planar convex hull.

In two dimensions, several $O(n \log n)$ time algorithms are known. Almost every algorithmic paradigm in computational geometry has been applied successfully to the planar convex hull algorithm: for instance, divide-and-conquer, incremental construction, planar sweep, and randomization have all been utilized to obtain $O(n \log n)$ time algorithms for planar convex hulls; see the textbook by Preparata and Shamos [211]. The best theoretical bound for the planar convex hull problem is achieved by an algorithm of Kirkpatrick and Seidel [151], which runs in time $O(n \log h)$, where h is the number of convex hull vertices.

In three dimensions, Preparata and Hong [210] proposed an $O(n \log n)$ time algorithm based on the divide-and-conquer paradigm. Theirs was the only known optimal algorithm in three dimensions, until Clarkson and Shor [65] developed a randomized incremental algorithm that achieved an expected running time $O(n \log n)$. A variant of Clarkson-Shor algorithm was proposed by Guibas, Knuth and Sharir [116], which admits a simpler implementation as well as an easier analysis. (These randomized algorithms are quite practical and considerably easier to implement than the divide-and-conquer algorithm.) Very recently, Chazelle and Matousek [54] have settled a long-standing open problem by announcing a deterministic $O(n \log h)$ time algorithm for the three-dimensional convex hull problem.

In higher dimensions, Chazelle [50] recently proposed an algorithm whose worst-case running time matches the worst-case bound on the facet complexity of the convex hull in any dimension $d \geq 4$. Chazelle's algorithm builds on earlier ideas of Kallay [144] and Seidel [225], and runs in worst-case time $O(n \log n + n^{\lfloor d/2 \rfloor})$. The algorithm in [50] achieves the best worst-case performance, but it does not depend on the actual size of the face lattice. An algorithm by Seidel [226] runs in time proportional to the size of the face lattice. In particular, the algorithm in [226] takes $O(n^2 + F \log n)$ time for facet enumeration and $O(n^2 + L \log n)$ time for producing the face lattice, where F is the number of facets in the convex hull and L is the size of the face lattice; Seidel's algorithm uses a method called "gift-wrapping" and builds upon the earlier work by Chand and Kapur [47] and Swart [237].

There is a vast literature on convex hulls, and the presentation above has hardly scratched the surface. We have left out whole lines of investigation on the convex hull problem, such as the expected case analysis of algorithms and the average size of the convex hull; we refer the reader to Dwyer [83], Devroye and Toussaint [76], and Golin and Sedgewick [107].

2.2. Arrangements

Arrangements refer to space partitions induced by lines, hyperplanes, or other algebraic varieties. A finite set of lines L partitions the plane into convex regions, called "cells," which are bounded by straight line edges and vertices. The *arrangement of lines* $\mathcal{A}(\mathrm{L})$ refers to this collection of cells, along with their incidence relations. Similarly, a set of hyperplanes (or other surfaces, such as spheres) induce arrangements in higher dimensions.

An arrangement encodes the sign pattern for its generating set. In other words, an arrangement is a manifestation of equivalence classes induced by a set of lines or hyperplanes — all points of a particular cell have the same "sign vector" with respect to all the hyperplanes in the set. This property is a key to solving many geometric problems in computational geometry. It often turns out that solving a geometric problem requires computing a particular cell or a family of cells in an arrangement of hyperplanes. We will say more about this in the section on Voronoi diagrams.

Combinatorial aspects of arrangements have been investigated for a long time; the arrangements in two and three dimensions were studied by Steiner [112]. The interested reader may consult Grübaum's book [112] for a detailed discussion on arrangements. We will focus mainly on the computational aspects of arrangements. An arrangement of n hyperplanes in d-space can be computed in time $O(n^d)$ by an algorithm due to Edelsbrunner et al. [91]. This bound is asymptotically tight since a simple arrangement (where no more than d hyperplanes meet in a point) has $\Theta(n^d)$ complexity.

Although a single cell in an arrangement of hyperplanes can have complexity $\Theta(n^{\lfloor d/2 \rfloor})$, not many cells can be large: afterall, the combined complexity of $\Theta(n^d)$ cells is only $O(n^d)$. There has been a considerable amount of work on bounding

the complexity of a family of cells in an arrangement. For instance, in two dimensions, the total complexity of any m cells in an arrangement of n lines is roughly $O(m^{2/3}n^{2/3}+m+n)$, up to some logarithmic factors [89, 25]. Extensions to higher dimensions and related results can be found in [27, 93, 87, 208]. Arrangements of bounded objects, such as line segments, triangles and tetrahedra, have also been studied, see [52, 166].

2.3. Geometric Duality

Duality plays an important role in geometric algorithms, and it often provides a tool for transforming an unfamiliar problem into a familiar setting. In this section, we will give a brief description of two most frequently used transformations in computational geometry.

The first transform $\mathcal{D}$ maps a point p to a hyperplane $\mathcal{D}(p)$ and vice versa:

$$p : (p_1, p_2, \ldots, p_d) \quad \Longleftrightarrow \quad \mathcal{D}(p) : \; x_d \; = \; 2p_1x_1 + 2p_2x_2 + \cdots + 2p_{d-1}x_{d-1} - p_d \tag{2.1}$$

Thus, in the plane the point (a, b) maps to the line $y = 2ax - b$, and the line $y = mx + c$ maps to the point $(m/2, -c)$. This transformation preserves incidence and order: (1) a point p is incident to hyperplane h if and only if the dual point $\mathcal{D}(h)$ is incident to dual hyperplane $\mathcal{D}(p)$, and (2) a point p lies below hyperplane h if and only if the dual point $\mathcal{D}(h)$ lies below the dual hyperplane $\mathcal{D}(p)$.

The second transform, also called the "lifting transform," maps a points in R^d to a point in R^{d+1}. It maps a point $p = (p_1, p_2, \ldots, p_d)$ in R^d to the point $p^+ = (p_1, p_2, \ldots, p_d, p_1^2+p_2^2+\cdots+p_d^2)$. If we treat R^d as the hyperplane $x_{d+1} = 0$ embedded in R^{d+1}, then the lifting transform maps a point $p \in R^d$ onto its vertical projection on the unit paraboloid $U : \; x_{d+1} \; = \; x_1^2 + x_2^2 + \cdots + x_d^2$. The combination of the lifting transform and the duality map $\mathcal{D}$ maps a point $p \in R^d$ to the hyperplane

$$\mathcal{D}(p^+) : \; x_{d+1} \; = \; 2p_1x_1 + 2p_2x_2 + \cdots + 2p_dx_d - (p_1^2 + p_2^2 + \cdots + p_d^2) \tag{2.2}$$

The hyperplane $\mathcal{D}(p^+)$ is tangent to the paraboloid U at the point p^+. It turns out that this mapping is especially useful for computing Voronoi diagrams, the topic of our next section.

2.4. Voronoi Diagram and Delaunay Triangulation

The Voronoi diagram is perhaps the most versatile data structure in all of computational geometry. This diagram, along with its graph-theoretical dual, the *Delaunay Triangulation*, finds applications in problems ranging from associative file

searching and motion planning to crystallography and clustering. In this section, we give a brief survey of some of the key ideas and results on these structures; for further details, consult Edelsbrunner's book [86] or a survey by Aurenhammer [32].

A Voronoi diagram encodes the "nearest-neighbor" information for a set of "sites." We begin by explaining the concept in two dimensions. Given a set of n "sites" or points $S = \{s_1, s_2, \ldots, s_n\}$ in the two-dimensional Euclidean plane, the Voronoi diagram of S partitions the plane into n convex polygons $V_1, V_2, \ldots, V_n$ such that any point in V_i is closer to s_i than to any other site:

$$V_i = \{x \mid d(x, s_i) \leq d(x, s_j), \text{ for all } j \neq i\},$$

where $d(x, y)$ is the Euclidean distance between the points x and y. An interesting fact about Voronoi diagrams in the plane is their linear complexity: $O(n)$ vertices and edges.

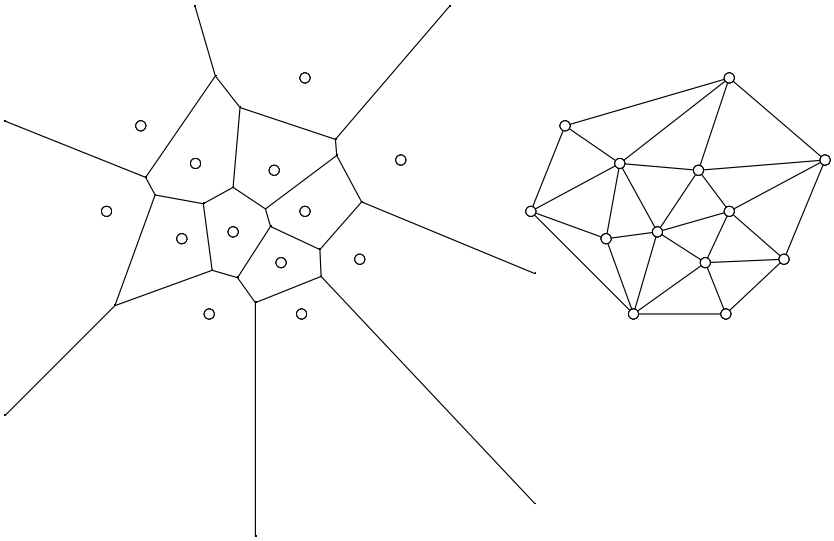

Fig. 2.4.1. The Voronoi diagram (left) and the Delaunay triangulation (right) of a set of points in the plane.

The Delaunay triangulation of S is the graph-theoretic dual of its Voronoi diagram: two sites s_i and s_j are joined by an edge if the Voronoi polygons V_i and V_j share a common edge. Under a non-degeneracy assumption that no four points of

S are co-circular, the dual graph is always a triangulation of S. Figure 2.4.1 shows an example of a Voronoi diagram and the corresponding Delaunay triangulation.

Just like convex hulls, algorithms based on several different paradigms are known for the construction of planar Voronoi diagrams and Delaunay triangulations, such as divide-and-conquer [82, 117], plane sweep [97], and randomized incremental methods [65, 116]. They all run in $O(n \log n)$ time (worst-case for deterministic, and expected for randomized).

The concepts of Voronoi diagram and Delaunay triangulation extend naturally to higher dimensions, as well as to other metrics. In d dimensions, the Voronoi diagram of a set of points S is a tessellation of E^d by convex polyhedra. The polyhedral cell V_i consists of all those points that are closer to s_i than to any other site in S. The Delaunay triangulation of S is the geometric dual of the Voronoi diagram: there is a k-face for every $(d-k)$-face of the Voronoi diagram. In particular, there is an edge between s_i and s_j if the Voronoi polyhedra V_i and V_j share a common $(d-1)$-dimensional face. An equivalent way of defining the Delaunay triangulation is via the *empty-sphere* test: a $(d+1)$-tuple $(s^1, s^2, \ldots, s^{d+1})$ is a simplex (triangle) of the Delaunay triangulation of S if and only if the sphere determined by $(s^1, s^2, \ldots, s^{d+1})$ does not contain any other point of S. The Voronoi diagram of n points in d dimensions, $d \geq 3$, can have super-linear size: $\Theta(n^{\lceil d/2 \rceil})$ [86].

It turns out that Voronoi diagrams and Delaunay triangulations are intimately related to convex hulls and arrangements via duality transforms. This relationship was first discovered by Brown [42], who showed using an inversion map that Voronoi diagram of a set $S \in R^d$ corresponds to the convex hull of a transformed set in R^{d+1}. Later Edelsbrunner and Seidel [92] extended and simplified this idea, using the paraboloid transforms mentioned in the previous section. We now sketch their idea.

Let $S = (s_1, s_2, \ldots, s_n\}$ be a set of n points in R^d. We map S to a set of hyperplanes $\mathcal{D}(S^+)$ in R^{d+1}, using the combination of lifting and duality maps mentioned in Section 2.3. In particular, a point $s = (a_1, a_2, \ldots, a_d)$ maps to the hyperplane $\mathcal{D}(s^+)$ whose equation is

$$x_{d+1} = 2a_1x_1 + 2a_2x_2 + \cdots + 2a_dx_d - (a_1^2 + a_2^2 + \cdots + a_d^2).$$

Let $\mathcal{P}$ be the polyhedron defined by the intersection of "upper" halfspaces bounded by these hyperplanes. Then, the vertical projection of $\mathcal{P}$ onto the hyperplane $x_{d+1} = 0$ gives precisely the Voronoi diagram of S in R^d.

A similar (and perhaps easier to visualize) relationship exists between convex hulls and Delaunay triangulations, using only the lifting transform. We map the points $S = \{s_1, s_2, \ldots, s_n\}$ to their "lifted" counterpart $S^+ = \{s_1^+, s_2^+, \ldots, s_n^+\}$. Now compute the convex hull of S^+. The triangles in the Delaunay triangulation of S correspond precisely to the facets of $CH(S^+)$ with downward normal.

Thus, both the Voronoi diagram and the Delaunay triangulation of a set of points in R^d may be computed using a convex hull algorithm in R^{d+1}. This relationship also explains why the worst-case size of both a Voronoi diagram in R^d and a convex hull in R^{d+1} is $\Theta(n^{\lfloor (d+1)/2 \rfloor})$.

2.5. Point Location

Many problems in computational geometry often require solving a so-called point location problem. Given a partition of space into polyhedral cells and a query point q, the problem is to identify the cell containing q. For instance, if Voronoi diagrams are used for answering "nearest neighbor" queries, one needs to locate the Voronoi polyhedron containing the query point. Typically, a large number of queries are asked with respect to the same cell complex; thus, it makes sense to preprocess the cell complex in order to speed up queries.

The problem has been studied intensely in two dimensions, where several optimal algorithms and data structures are now known. These algorithms can preprocess a planar map on n vertices into a linear space data structure and answer a point location query in $O(\log n)$ time (see [163, 150, 90, 108]).

In higher dimensions, the point location problem is less well-understood, and no algorithm simultaneously achieves optimal bounds for preprocessing time, storage space, and query time. We give a brief summary of results and give pointers to relevant literature. We denote the performance of an algorithm by the triple $\{P, S, Q\}$, which refer to preprocessing time, storage space, and query time.

Preparata and Tamassia [238] give an $\{O(n \log^2 n), O(n \log^2 n), O(\log^2 n)\}$ algorithm for point location in a convex cell complex of n facets in three dimensions. Using randomization, Clarkson [63] gives an $\{O(n^{d+\varepsilon}), O(n^{d+\varepsilon}), O(\log n)\}$ algorithm for point location in an arrangement of n hyperplanes in d dimensions; the space and query bounds are worst-case, but the preprocessing time is expected. Chazelle and Friedman [53] were later able to make Clarkson's algorithm deterministic, albeit at an increased preprocessing cost, resulting in an algorithm with resource bounds $\{O(n^{d(d+3)/2+2}), O(n^d), O(\log n)\}$.

3. Geometric Graphs

3.1. Minimum Spanning Trees

The minimum spanning tree (MST) problem is one of the best-known problems of combinatorial optimization, and it has received a considerable attention in computational geometry as well. Given a graph $G = (V, E)$, with non-negative real-valued weights on edges, a minimum spanning tree of G is an acyclic subgraph of G that spans all the nodes in V and has minimum total edge weight. An MST has obvious applications in the design of computer and communication networks, transportation systems, and other kinds of networks. But applications of the minimum spanning tree extend far beyond network design problems. They are used in problems as diverse as network reliability, computer vision, automatic speech recognition, clustering and classification, matching and traveling salesman problems, and surface homogeneity tests.

Efficient algorithms for computing an MST have been known for a long time; a survey by Graham and Hell [111] traces the history of MST and cites algo-

rithms dating back to the beginning of the century. Although the algorithms of Kruskal [154] and Prim [212] are among the best known, an algorithm by Borøuvka preceded them by almost thirty years [111]. Using suitable data structures, these algorithms can be implemented in $O(|E|\log|V|)$ or $O(|V|^2)$ time. In the last two decades, several new implementations and variants of these basic algorithms have been proposed, and the fastest ones run in almost linear time in the size of the graph [98, 100].

The interest of computational geometry researchers in MST stems from the observation that in many applications the underlying graph is Euclidean: we want to compute a minimum spanning tree for a set of n points in R^d, for $d \geq 1$. The set of n points in this case completely specifies the graph, without an explicit enumeration of the edges. Since the edge-weights in this geometric graph are not entirely arbitrary, a natural question is if one can compute an MST in (asymptotically) less than n^2 steps, that is, without inspecting every edge.

Surprisingly, it turns out that for a set of n points in the plane, an MST can be computed in $O(n\log n)$ time. A key observation is the following lemma, which states that the edges of an MST are contained in the Delaunay triangulation graph; we omit the proof, but an interested reader may consult the book of Preparata-Shamos [211].

Lemma 3.1. *Let S be a set of n points in the plane, and let $DT(S)$ denote the Delaunay triangulation of S. Then, $MST(S) \subseteq DT(S)$.*

We recall from Section 2.4 that the Delaunay triangulation in two dimensions is a planar graph. Thus, by running an efficient graph MST algorithm on $DT(S)$, we can find a minimum spanning tree of S in $O(n\log n)$ time. In fact, given the Delaunay triangulation, a minimum spanning tree of S can be extracted in linear time, by using an algorithm of Cheriton and Tarjan [56], which computes an MST in linear time for planar graphs.

Lemma 3.1 holds in any dimension; however, it no longer serves a useful purpose for computing minimum spanning trees in higher dimensions since the Delaunay graph can have size $\Omega(n^2)$ in dimensions $d \geq 3$ [211]. Nevertheless, the underlying geometry can be exploited to compute an MST in subquadratic worst-case time. Yao [259] has proposed a general method for computing geometric minimum spanning trees in time $O(n^{2-\alpha_d}(\log n)^{1-\alpha_d})$, where α_d is a dimension-dependent constant. Yao's algorithm is based on the following idea: if we partition the space around a point p into polyhedral cones of sufficiently small angular diameter, then there is at most one MST edge incident to p per cone, and this edge joins p to its nearest neighbor in that cone. In order to find these nearest neighbors efficiently, Yao utilizes a data structure that, after a polynomial-time preprocessing, can determine a nearest neighbor in logarithmic time. In the original paper of Yao [259], the constant α_d had value of $2^{-(d+1)}$, thus, making his algorithm only slightly subquadratic; however, the interesting conclusion is that an MST can be computed without checking all the edges.

The exponent in the general algorithm of Yao has steadily improved, as better data structures have been developed for solving the nearest-neighbor problem. Re-

cently, Agarwal et al. [2] have shown also that computationally the twin problems of computing a minimum spanning tree and computing a bi-chromatic nearest neighbor are roughly equivalent. The constant α_d in the running time their algorithm is roughly $\frac{2}{\lceil d/2 \rceil + 1}$ [2]. In three dimensions, the algorithm of [2] computes an MST in $O(n^{4/3} \log^{4/3} n)$ time. An alternative and somewhat simpler, albeit randomized, algorithm of the same complexity is given by Agarwal, Matoušek and Suri [4].

There also has been work on computing an approximation of the MST. Vaidya [245] constructs in $O(\varepsilon^{-d} n \log n)$ time a spanning tree with length at most $1 + \varepsilon$ times the length of an MST. If the n points are independently and uniformly distributed in the unit d-cube, then the expected time complexity of Vaidya's algorithm is $O(n\alpha(cn, n))$, where α is the inverse Ackermann function.

The best lower bound known for the MST problem is $\Omega(n \log n)$, in any fixed dimension $d \geq 1$. (The lower bound holds in the algebraic tree model of computation for any input consisting of an unordered set of n points; $o(n \log n)$ time algorithms are possible for special configurations of points, such as the vertices of a convex polygon if the points are given in order along the boundary of the polygon.) It is an outstanding open problem in computational geometry to settle the asymptotic time complexity of computing a geometric minimum spanning tree in d-space.

Open Problem 1. *Given a set S of n unordered points in E^d, compute its Euclidean minimum spanning tree in $O(c_d\, n \log n)$ time, where c_d is a constant depending only on the dimension d. Alternatively, prove a lower bound that is better than $\Omega(n \log n)$.*

There is an obvious connection between MST and nearest neighbors: the MST neighbors of a points s include a nearest neighbor of s. Thus, the *all-nearest-neighbors* problems, which asks for a nearest neighbor for each of the points of S, is no harder than computing the $MST(S)$. A few years ago, Vaidya [248] gave an $O(c_d\, n \log n)$ time algorithm for the all-nearest-neighbors problems for any fixed dimension; the constant c_d in Vaidya's algorithm is of the order of 2^d.

Unfortunately, no reduction in the converse direction (given all nearest neighbors, compute MST) is known. However, the result of Agarwal et al. [2] points out an equivalence between the MST and the *bi-chromatic closest pair* problem. The bi-chromatic closest pair problem is defined for two d-dimensional sets of points R and B, and it asks for a pair $r \in R$ and $b \in B$ that minimizes the distance over all such pairs. It is shown in [2] that the asymptotic time complexities of the two problems are the same if they have the form $\Theta(n^{1+\varepsilon})$, for any $\varepsilon > 0$, otherwise, they are within a polylogarithmic factor of each other. This leads to the following open problem.

Open Problem 2. *Given two unordered sets of points $B, R \subset E^d$, compute a bi-chromatic closest pair of B and R in time $O(c_d\, n \log n)$, where $n = |B| + |R|$ and c_d is a constant depending only on the dimension d. Alternatively, prove a lower bound better than $\Omega(n \log n)$.*

3.2. Maximum Spanning Trees

A maximum spanning tree is the other extreme of the minimum spanning tree: it *maximizes* the total edge weight. In graphs, a maximum spanning tree can be computed using any minimum spanning tree algorithm, by simply negating all the edge weights. But what about a *geometric* maximum spanning tree? Is it possible to compute the maximum spanning tree of a set of points in less then quadratic time?

As a first attempt, we could try to generalize Lemma 3.1. Instead of a Delaunay triangulation, we would consider the so-called *furthest-point* Delaunay triangulation, which is the graph-theoretic dual of the *furthest-point Voronoi diagram*. (In a furthest-point Voronoi diagram of a set of points S, the region associated with a site $s_i \in S$ consists of all the points x that satisfy $d(x, s_i) \geq d(x, s_j)$, for all $s_i \neq s_j$; see Preparata-Shamos [211] for details.) Unfortunately, the maximum spanning tree edges do not necessarily lie in the furthest-point Delaunay triangulation. One of the reasons why this relationship does not hold is trivial: the furthest-point Delaunay triangulation only triangulates the convex hull vertices of S; the interior points of S have an empty furthest-point Voronoi polygon. The trouble in fact goes deeper: even if all points of S were to lie on its convex hull, the maximum spanning tree does not always lie in the Delaunay triangulation. Consider the example in Figure 3.2.1, which is due to Bhattacharya and Toussaint [40]. In this figure, $\triangle ACD$ is an equilateral triangle and B lies on the line joining D with the center O of the circle ACD such that $2d(D,O) > d(D,B) > d(D,A)$. It is easy to check that the furthest-point Delaunay triangulation consists of triangles $\triangle ABC$ and $\triangle ACD$, and does not include the diagonal BD. On the other hand, the maximum spanning tree of $\{A, B, C, D\}$ necessarily contains the edge BD; the other two edges can be any two of the three edges of the equilateral triangle $\triangle ACD$.

An optimal $O(n \log n)$ time algorithm for computing a maximum spanning tree of n points in the plane was proposed a few years ago by Monma, Paterson, Suri and Yao [184]. Their algorithm starts by computing the *furthest neighbor* graph: connect each point to its furthest neighbor. This results in a forest, whose components are called *clusters* in [184]. Monma et al. show that these clusters can be cyclically ordered around their convex hull, and that the final tree can be computed by merging *adjacent* clusters, where merging two clusters means adding a longest edge between them. The algorithm in [184] runs in $O(n)$ time if all the points lie on their convex hull.

Subquadratic algorithms for higher dimensional maximum spanning trees were obtained a little later by Agarwal, Matoušek, and Suri [4], who proposed randomized algorithms of expected time complexity $O(n^{4/3} \log^{7/3} n)$ for dimension $d = 3$, and $O(n^{2-\alpha_d})$ for dimension $d \geq 4$, where α_d is roughly $\frac{2}{\lceil d/2 \rceil + 1}$. Agarwal, Matoušek, Suri [4] also present a simple approximation algorithm that computes in $O(\varepsilon^{(1-d)/2} n \log^2 n)$ time a spanning tree with total length at least $(1 - \varepsilon)$ times the optimal.

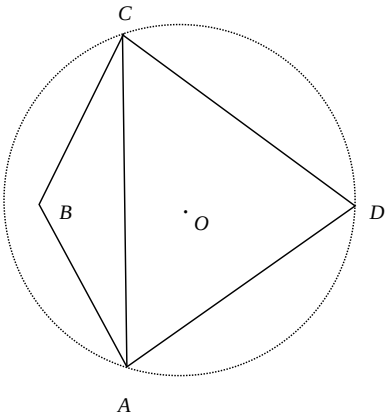

Fig. 3.2.1. MXST is not a subset of farthest-point Voronoi diagram.

3.3. Applications of Minimum and Maximum Spanning Trees

We said earlier that minimum spanning trees have several applications; some are obvious, such as network design problems, and some are less obvious, such as pattern recognition, traveling salesman, and clustering problems. In this section, we mention some constrained clustering problems that can be solved efficiently using minimum and maximum spanning trees.

Given a set of points S in the plane, define a k-partition of S as a decomposition of S into k disjoint subsets $\{C_1, C_2, \ldots, C_k\}$. We want to find a k-partition that *maximizes* the minimum intercluster distance: $\min_{i,j} \min\{d(s,t) \mid s \in C_i, t \in C_j\}$. Asano et al. [30] show that an optimal k-partition is found by deleting the $(k-1)$ *longest* edges from the minimum spanning tree of S.

Next, consider the problem of partitioning a point set S into two clusters subject to the condition that the larger of the two diameters is minimized; recall that the *diameter* of a finite set of points is that maximum distance between any two points in the set. An $O(n \log n)$ time solution of this problem was proposed by Asano et al. [30], and also by Monma and Suri [185], based on the maximum spanning tree. The method of Monma and Suri is particularly simple: compute a maximum spanning tree of S and 2-color its nodes (points). The partition induced by the 2-coloring is an optimal minimum diameter 2-partition.

A related bi-partition problem is to minimize the *sum* of measures of the two subsets. Monma and Suri [185] gave an $O(n^2)$ time algorithm for computing a bi-partition of n points minimizing the sum of diameters. This result was subsequently improved to $O(n \log^2 n)$ time by Hershberger [123]. An interesting problem in this class is to find a sub-quadratic algorithm for the two-disk covering of a point set with a minimum radius. The relevant results on this problem appear in Hershberger

and Suri [129] and Agarwal and Sharir [5]; the former gives an $O(n^2 \log n)$ time algorithm to check the feasibility of a covering by two disks of given radii, and the latter gives an $O(n^2 \log^3 n)$ time algorithm for finding the minimum radius. It is an open problem whether a minimum radius two-disk covering of n points can be computed in sub-quadratic time.

Open Problem 3. *Given n points in the plane, give an $o(n^2)$ time algorithm for computing the minimum radius r such that all the points can be covered with two disks of radius r; also, find the corresponding disks.*

3.4. Gabriel and Relative Neighborhood Graphs

Nearest neighbor relationships play an important role in pattern recognition problems. One of the simplest graphs encoding these relationships is the *nearest neighbor* graph, which has a (directed) edge from point a to point b if b is a nearest neighbor of a. The minimum spanning tree is a next step, which repeatedly applies the nearest neighbor rule until we obtain a connected graph. Building on this theme, several other classes of graphs have been introduced. We discuss two such graphs in this section: the Gabriel graph and the relative neighborhood graph. The Gabriel graph was introduced by Gabriel and Sokal [101] in the context of geographical variation analysis, while the relative neighborhood graph was introduced by Toussaint [244] in a graph-theoretical context. Matula and Sokal [167] studied several properties of the Gabriel graphs, with applications to zoology and geography. A recent survey paper by Jaromczyk and Toussaint [142] is a good source for additional information on these graphs. Let us first describe the Gabriel graph.

Let $S = \{s_1, s_2, \ldots, s_n\}$ be a set of points in the plane, and define the *circle of influence* of a pair $s_i, s_j \in S$ as

$$C(s_i, s_j) \;=\; \{x \in R^2 \mid d^2(x, s_i) + d^2(x, s_j) \;=\; d^2(s_i, s_j)\}.$$

We observe that $C(s_i, s_j)$ is the circle with diameter (s_i, s_j). The *Gabriel* graph $GG(S)$ has the set of points S as its vertex set, and two vertices s_i and s_j have an edge between them if the circle $C(s_i, s_j)$ does not include any other point of S. In other words, (s_i, s_j) is an edge of $GG(S)$ if and only if $d^2(s_i, s_k) + d^2(s_j, s_k) \geq d^2(s_i, s_j)$, for all s_k. This definition immediately implies that the Gabriel graph of S is a subgraph of the Delaunay triangulation $DT(S)$; recall the empty circle definition of Delaunay triangulations (cf. Sec. 2.4).

Matula and Sokal [167] give an alternative definition of the Gabriel graph: an edge (s_i, s_j) of $DT(S)$ is in $GG(S)$ if and only if (s_i, s_j) intersects its dual edge in the Voronoi diagram of S. This latter characterization leads immediately to an $O(n \log n)$ time algorithm for computing the Gabriel graph: first compute the Delaunay triangulation $DT(S)$ and then delete all those edges that do not intersect their dual edges in the corresponding Voronoi diagram.

In dimensions $d \geq 3$, the complexity of the Gabriel graph depends on whether or not many points are co-spherical. (Note that this is not the case for Delaunay

triangulation.) If no more than a constant number of points lie on a common $(d-1)$-sphere, then GG has only a linear number of edges. Computing this graph in less than quadratic time is still quite non-trivial. Slightly sub-quadratic algorithms are presented in [3]. Without the non-degeneracy assumption, the Gabriel graph can have $\Omega(n^2)$ edges even in three dimensions. The following example gives a lower bound construction for $d = 3$. Take two orthogonal, interlocking circles of radius 2, each passing through the center of the other. In particular, let the first circle lie in the xy-plane with $(0, -1, 0)$ as the center, while the second circle lies in the yz-plane with $(0, 1, 0)$ as the center. Place $n/2$ points on the first circle very close to the point $(0, 1, 0)$, and $n/2$ points on the second circle close to the point $(0, -1, 0)$. Then, it is easy to see that the Gabriel graph of these n points contains the complete bipartite graph between the two sets of $n/2$ points.

Open Problem 4. *Given n points in E^d such that only $O(d)$ points lie on a common sphere, give an $O(c_d n \log n)$ time algorithm to construct their Gabriel graph, where c_d is dimension-dependent constant. Alternatively, prove a lower bound better than $\Omega(n \log n)$.*

The basic construct in the definition of a relative neighborhood graph is the "lune of influence." Given two points $s_i, s_j \in S$, their *lune of influence* $L(s_i, s_j)$ is defined as follows:

$$L(s_i, s_j) \;=\; \{x \in R^2 \mid \max\{d(x, s_i),\, d(x, s_j)\} \leq d(s_i, s_j)\}.$$

Thus, the lune $L(s_i, s_j)$ is the common intersection of two disks of radius $d(s_i, s_j)$ centered on s_i and s_j. The *relative neighborhood* graph $RNG(S)$ has an edge between s_i and s_j if and only if the lune $L(s_i, s_j)$ does not contain any other point of S. Again, it easily follows that $RNG(S) \subseteq DT(S)$; in fact, the relative neighborhood graph is also a subgraph of the Gabriel graph, since the circle of influence is a subset of the lune of influence. Thus, we have the following ordered relations among the four graphs we have discussed in this section:

$$MST \subseteq RNG \subseteq GG \subseteq DT.$$

Characterization of the DT edges not in RNG, however, is not so easy as it was for the Gabriel graph. In two dimensions, Supowit [232] presents an $O(n \log n)$ time algorithm for extracting the RNG from the Delaunay triangulation. If the points form the vertices of a convex polygon, then the minimum spanning tree, relative neighbor graph, Gabriel graph, and Delaunay triangulation can each be computed in linear time. The bound for MST, GG, and DT is implied by a linear time algorithm for computing the Voronoi diagram of a convex polygon [8], and the result on RNG is due to Supowit [232].

In dimensions $d \geq 3$, the size of the relative neighborhood graphs depends on whether or not the points are co-spherical. If only a constant number of points lie on a common $(d-1)$-sphere, then the RNG has $O(n)$ edges, but without this restriction, it is easy to construct examples where RNG has $\Omega(n^2)$ edges in any dimension

$d \geq 4$. In R^3, the best upper bound on the size of the relative neighborhood graph currently known is $O(n^{4/3})$ [3].

Open Problem 5. *Given a set S of n points in E^3 such that only a constant number of points lie on a common sphere, show that the relative neighborhood graph of S has only $O(n)$ edges. Alternatively, prove a super-linear lower bound on the size of the relative neighborhood graph.*

The size of the relative neighborhood graph is related to the number of bichromatic closest pairs [3].

Open Problem 6. *Given two unordered sets of points $B, R \subset E^3$, what is the maximum number of pairs (b, r) such that $r \in R$ is a closest neighbor of $b \in B$?*

4. Path Planning

4.1. Introduction

The shortest path problem is a familiar problem in algorithmic graph theory. Given a graph $G = (V, E)$, whose edges have non-negative, real-valued weights associated with them, a shortest path between two nodes s and t is a path in G from s to t having the minimum possible total edge weight. The shortest path problem is to find such a path. Generalizations of this basic shortest path problem include the *single source* and the *all-pairs* shortest paths problems; the former asks for shortest paths to all the vertices of G from a specified *source* vertex s, and the latter asks for shortest paths between all pairs of vertices. The best-known algorithm for computing shortest paths is due to Dijkstra [77]. If properly implemented, his algorithm can find a shortest path between two vertices in time $O(\min(n^2, m \log n))$; here n and m denote the number of vertices and edges of G. A considerable amount of research has been invested in improving this time complexity for sparse graphs, that is, graphs with $m \ll n^2$. Only a few years ago, Fredman and Tarjan [98] succeeded in devising an implementation of Dijkstra's algorithm, using their Fibonnaci heap data structure, that achieved a worst-case running time of $O(m + n \log n)$; this time bound is optimal in a comparison-based model of computation.

The shortest path problem acquires a new richness when transported to a geometric domain. Unlike a graph, an instance of the geometric shortest path problem is typically specified through the description of some geometric objects that implicitly encode the graph. This raises the following rather interesting question: is it possible to compute a shortest path without explicitly constructing the entire graph? There are some intriguing possibilities associated with this question. For instance, a set of geometric objects can encode some very large, super-polynomial or even exponential, size graphs, implying that an efficient shortest path algorithm must necessarily avoid building the entire graph. Even if the graph is polynomial-size, considerable efficiency gain is possible if the shortest path problem can be solved by constructing only a small, relevant subset of the edges. We will address

these issues in more detail later, but for now let us just say that there is a diverse variety of shortest path problems, depending upon the type of geometric objects considered, the metric used, and the dimension of the underlying geometric space. We start with a discussion of some common basic concepts.

4.2. Basic Concepts

The most commonly studied shortest path problems in computational geometry typically involve a set of polyhedral objects, called *obstacles*, in Euclidean d-space, $d \geq 2$, and the goal is to find an obstacle-avoiding path of minimum length between two points. Much of our discussion will be limited to shortest paths in the plane ($d = 2$). A connected subset of the plane whose boundary consists of a union of a finite number of straight line segments will be called a *polygonal domain*. The boundary segments are called *edges*; their endpoints are called *vertices*. A polygonal domain P is called a *simple polygon* if it is simply-connected, that is, it is homeomorphic to a disk. A multiply-connected polygonal domain P is also called a *polygon with holes*.

4.2.1. Triangulation

A triangulation of a polygonal domain P is a decomposition of P into triangles with disjoint interiors, with each triangle having its corners among the vertices of P. (If we allow triangles whose corners are *not* among the vertices of P, the triangulation is called a *Steiner* triangulation; we do not use Steiner triangulations in this section.) It is a well-known fact that a polygonal domain can always be triangulated (without using Steiner points). Since a triangulation is a planar graph, the number of triangles is linearly related to the number of vertices of P.

A polygonal domain with n vertices can be triangulated in $O(n \log n)$ time [211]. This time complexity is worst-case optimal in the algebraic tree model of computation. The lower bound, however, does not apply if P is a simple polygon, raising the possibility than a better algorithm might be possible for triangulating a simple polygon. Indeed, the problem of triangulating a simple polygon became one of the most notorious problems in computational geometry in the eighties. Despite the discovery of numerous algorithms, the $O(n \log n)$ time bound remained unbeaten in the worst-case performance. Then in 1988, a breakthrough result by Tarjan and van Wyk [240] produced an $O(n \log\log n)$ time triangulation algorithm. Finally, Chazelle [51] recently managed to devise a linear-time algorithm for triangulating a simple polygon, thus settling the theoretical complexity of the problem.

For a polygon with holes, it is possible to perform a triangulation in running time dependent on the number of holes or the number of reflex, i.e., non-convex, vertices. In particular, a polygonal domain P can be triangulated in $O(n \log r)$ time, where r is the number of reflex vertices of P, or in time $O(n + h \log^{1+\varepsilon} n)$, where h is the number of holes in P and ε is an arbitrarily small positive constant [37].

4.2.2. Visibility

Visibility is a key concept in geometric shortest paths. We say that points s and t are (mutually) *visible* if the line segment joining them lies within the polygonal domain P. The relevance to shortest path planning is clear: If points s and t are visible to one another, then the shortest obstacle-avoiding path between them is simply the line segment joining them.

The *visibility polygon*, $V(s)$, with respect to a point $s \in P$ is defined to be the set of points that are visible to s. A visibility polygon can be found in time $O(n \log n)$ by applying the *sweep-line* paradigm of computational geometry: We simulate the sweeping of a ray $r(\theta)$ angularly about s, keeping track of the ordered *crossing list* of edges of P intersecting $r(\theta)$. When the ray $r(\theta)$ encounters a vertex of P, we insert and/or delete an edge from the crossing list, and we make any necessary updates to the visibility profile; the cost per update is $O(\log n)$. We can always know which vertex is encountered next by the sweeping ray if we sort the vertices of P angularly about s, in $O(n \log n)$ time. If P has h holes, then a recent result of Heffernan and Mitchell [120] shows that one can compute a visibility polygon in optimal time, $O(n + h \log h)$.

The *visibility graph* (VG) of P is defined to be the graph whose nodes are the set of vertices of P and whose edges join pairs of vertices that are mutually visible. Refer to Figure 4.2.1. We let E_{VG} denote the number of edges in VG; note that $E_{VG} \leq \binom{n}{2}$ for an n-vertex domain P. Visibility graphs were first introduced in the work of Nilsson [190], who used them for computing shortest paths for a mobile robot.

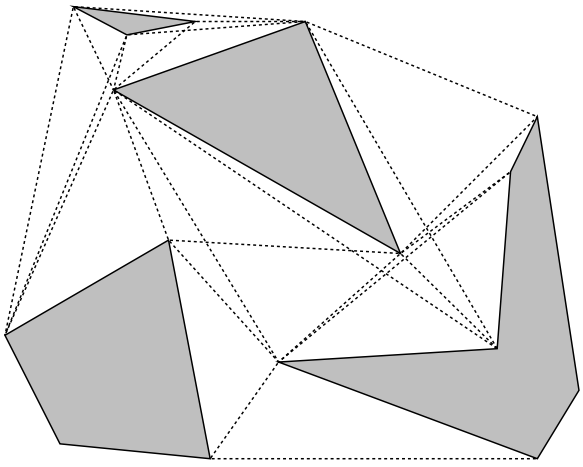

Fig. 4.2.1. A visibility graph.

The most naive algorithm for computing the VG runs in time $O(n^3)$, checking each pair of vertices (u, v) for visibility by testing against all edges of P. A substantially improved algorithm is possible based on the idea of computing visibility

polygon of each vertex. The visibility graph edges incident to a vertex v can be found in $O(n \log n)$ time by first constructing the visibility polygon of v, and hence the entire visibility graph can be computed in $O(n^2 \log n)$, using only $O(n)$ working space.

The state of the art in VG construction remained at the $O(n^2 \log n)$ level until 1985, when Welzl [252] (and, independently, Asano et al [29]) obtained algorithms whose worst-case running times were $O(n^2)$. These new algorithms rely on the trick of mapping the vertices of P to their *dual* lines, building the *arrangement* of these lines (in time $O(n^2)$ [91]), and then using the information present in the arrangement to read off the sorted order of vertices about each vertex v in total time $O(n^2)$. Thus, the $O(n)$ angular sorts are not independent of each other, as they can be done collectively in total time $O(n^2)$. Once the angular order is known for vertices about every other vertex, a further trick is necessary to produce the VG without the logarithmic overhead per pair — for example, Welzl [252] uses a topological sort (available from the arrangement) to guide the construction of the visibility profiles about every vertex. Edelsbrunner and Guibas [88] have shown how to use a method of "topological sweep" to compute the VG in time $O(n^2)$ using only $O(n)$ working storage (i.e., avoiding the need to keep the entire line arrangement in memory during VG construction).

In the worst case, we know that it takes quadratic time ($O(n^2)$) to compute a visibility graph, since visibility graphs exist with this size. In some cases, however, the visibility graph is very sparse (linear in size). Thus, ideally, we would like an algorithm whose running time is *output-sensitive*, taking time proportional to the size (E_{VG}) of the output.

Ghosh and Mount [106] have developed such an *output-sensitive* algorithm, achieving a time bound of $O(n \log n + E_{VG})$, with a working storage requirement of $O(E_{VG})$. Their algorithm begins with a triangulation of P and constructs VG edges by a careful analysis of the properties of "funnel sequences". Independently, Kapoor and Maheshwari [146] obtained a similar bound and also show how one can compute the *subgraph of VG relevant for shortest path planning* in time $O(n \log n + E_{SP})$ and space $O(E_{SP})$, where E_{SP} is the size of the resulting subgraph. (In other words, only those edges of the VG that appear along some nontrivial shortest path are actually discovered and output.)

Overmars and Welzl [200] give two very simple (easily implementable) algorithms for computing the visibility graph that use only $O(n)$ space. The first algorithm runs in time $O(n^2)$ and is based on "rotation trees"; the second is output-sensitive, requiring time $O(E_{VG} \log n)$. See also Alt and Welzl [18]. The main open problem in visibility graph construction is summarized below:

Open Problem 7. *Given a polygonal domain with n vertices and h holes, compute the visibility graph in time $O(h \log h + E_{VG})$, where E_{VG} is the number of edges of the resulting graph. Ideally, do this computation using only $O(n)$ working storage.*

Mitchell and Welzl [170] have developed an on-line algorithm to construct a VG, by showing that one can update a VG when a new obstacle is inserted in time

$O(n+k)$, where k is the number of VG edges that must be removed when the new obstacle is inserted. (Note that k may be as large as $\Omega(n^2)$.) Vegter [249, 250] shows that a VG can be maintained under both insertions and deletions, in time $O(\log^2 n + K \log n)$ per update, where K is the size of the change in the VG. We are left with an interesting open question:

Open Problem 8. *Devise a dynamic algorithm for maintaining a visibility graph in $O(\log n + K)$ time per insertion or deletion of an obstacle, where K denotes the number of changes in the visibility graph.*

4.3. Shortest Obstacle-Avoiding Paths

The most basic kind of geometric shortest path problem is that of finding a shortest path from s to t for a point robot that is confined to the interior of a polygonal domain P. We assume that P has h holes (which can be thought of as the *obstacles*) and a total of n vertices. In this subsection, we measure path length according to the (usual) Euclidean metric; in the following subsections, we discuss variants on the objective function.

4.3.1. Paths in a Simple Polygon

Assume that there are no holes in P (i.e., $h = 0$). Then, there is a unique *homotopy* class of any path from s to t, and the shortest path from s to t will be the unique "taut string" path. If we triangulate polygon P, then there is a unique path in the triangulation dual graph (which is a tree in this case), from the triangle containing s to the triangle containing t. This gives us a *sleeve* within P that is known to contain the shortest s-t path. Chazelle [48] and Lee and Preparata [160] show that, in time linear in the number of triangles defining the sleeve, one can "pull taut" the sleeve, producing the unique shortest s-t path.

The algorithm proceeds incrementally, considering the effect of adding the triangles in order along the sleeve. At a general step of the algorithm, when we are about to add triangle Δabc, we know the shortest path from s to a vertex r (of the sleeve), and the (concave) shortest subpaths from r to a and from r to b, which define a region called the *funnel* with base ab and apex r. Refer to Figure 4.3.1. In order to add Δabc, we must "split" the funnel according to the taut-string path from r to c, which will, in general, include a segment, uc, joining c to some vertex of tangency u along one of the concave chains of the funnel. We need to keep only one of the two funnels (based on ac and ab), since only one can lead through the sleeve to t, which allows us to charge off the work of searching for u to vertices that can be discarded.

Since a simple polygon can be triangulated in linear time (Chazelle [51]), the result of [48, 160] establishes that shortest paths in a simple polygon can be found in $O(n)$ time, which is worst-case optimal. This result has been generalized in several directions:

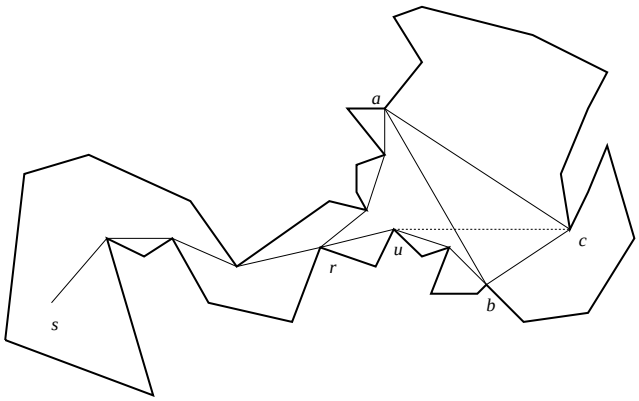

Fig. 4.3.1. Splitting a funnel.

– Guibas et al. [114] show that one can construct the shortest path tree (and its extension into a "shortest path map") rooted at a point s in $O(n)$ time, after which the length of a shortest path to any query point t can be reported in $O(\log n)$ time (and the shortest path can be output in time proportional to its size). Their result relies on using "finger search trees" to do funnel splitting, which now must be done *without* discarding either of the two new funnels. Hershberger and Snoeyink [126] have given a considerably simpler algorithm to compute shortest path trees without any special data structures.

– Guibas and Hershberger [113] show that a simple polygon can be preprocessed in time $O(n)$, into a data structure of size $O(n)$, such that one can answer shortest path length queries between a pair of points in $O(\log n)$ time. In fact, within the $O(\log n)$ query time, one can construct an implicit representation of the shortest path, so that one can output the path explicitly in time proportional to its length (number of vertices).

– ElGindy and Goodrich [95] give a parallel algorithm to compute shortest paths in time $O(\log n)$, using $O(n)$ processors (in the CREW PRAM model). Goodrich, Shauck, and Guha [109] show how, with $O(n/\log n)$ processors and $O(\log n)$ time, one can compute a data structure that supports $O(\log n)$ (sequential) time shortest path queries between pairs of points in a simple polygon. They also give an $O(\log n)$ time algorithm using $O(n)$ processors to compute a shortest path tree. Hershberger [124] builds on the results of [109] and gives an algorithm for shortest path trees requiring only $O(\log n)$ time and $O(n/\log n)$ processors (CREW); he also obtains optimal parallel algorithms for related visibility and geodesic problems.

– Many other problems have been studied with respect to shortest path (*geodesic*) distances within a simple polygon. Aronov [24] shows how to compute, in time $O(n \log^2 n)$, the Voronoi diagram of a set of point sites in a simple polygon if the metric is the geodesic distance. The *geodesic diameter* is the length of

the longest shortest path between a pair of vertices; it can be computed in time $O(n \log n)$ ([235, 113]). The *geodesic center* is the point within P that minimizes the maximum of the shortest path lengths to any other point in P; Pollack, Sharir, and Rote [209] give an $O(n \log^2 n)$ algorithm. Suri [235] studies problems of computing geodesic furthest neighbors. The furthest-site Voronoi diagram for geodesic distance is computed in time $O(n \log n)$ by Aronov, Fortune, and Wilfong [26].

All of the above linear-time algorithms rely on a triangulation of a simple polygon. It's an interesting open problem whether a shortest path inside a simple polygon can be computed optimally without a triangulation.

Open Problem 9. *Given a simple polygon with n vertices, devise an $O(n)$ time algorithm for computing the shortest path between two points without triangulating the polygon.*

4.3.2. Paths in General Polygonal Spaces

In the general case in which P contains holes (*obstacles*), shortest paths can be computed using the visibility graph, as justified in the following straightforward lemma (proved in [159, 171]).

Lemma 4.1. *Any shortest path from $s \in P$ to $t \in P$ in a polygonal domain P must lie on the visibility graph, VG, of P (where VG includes s and t, in addition to vertices of P, as nodes).*

This lemma implies that, after constructing the VG, we can search for shortest paths in time $O(E_{VG} + n \log n)$, using Dijkstra's algorithm with appropriate data structures (e.g., Fibonacci heaps [98] or relaxed heaps [81]). The result of Dijkstra's algorithm is a shortest path tree, SPT(s). In practice, it may be faster to apply the A* heuristic search algorithm (e.g., see Pearl [207]), using the straight-line Euclidean distance as heuristic function, $h(\cdot)$ (which is a lower bound, so it implies an "admissible" algorithm).

Since VG can be computed in time $O(E_{VG} + n \log n)$ ([106, 146]), we conclude that Euclidean shortest paths among obstacles in the plane can be computed in time $O(E_{VG} + n \log n) = O(n^2)$.

Special cases of these results are possible when the obstacles are *convex*, in which case the quadratic term can be written in terms of h (the number of obstacles) rather than n (the number of vertices); see Mitchell [171] and Rohnert [217, 216].

Another special case of relevance to VLSI routing problems (see [66, 161, 102]) is to compute shortest paths among obstacles *of a given homotopy type*. Hershberger and Snoeyink [126] generalize the shortest path algorithm for simple polygons to show that one can compute a shortest path among obstacles of a particular "threading" in time proportional to the "size" of the description of the homotopy type.

4.3.2.1. Shortest Path Maps A *shortest path map*, SPM(s), is an implicit representation of the set of shortest paths from s to *all* points of P. The utility of SPM(s) is

that it is a planar subdivision (of size $O(n)$) such that once we perform an $O(\log n)$ time point location query for t, the map tells us the length of a shortest s-t path and allows a path to be reported in time proportional to its size (number of bends). The general concept of shortest path maps applies to all metrics; here, we mention some facts relevant to Euclidean shortest paths among polygonal obstacles in the plane.

If our final goal is to compute a shortest path map, SPM(s), then we can obtain it in $O(n \log n)$ time, *given* the shortest path tree obtained by searching VG with Dijkstra's algorithm [174].

An alternative approach is to build the (linear-size) SPM(s) directly, and avoid altogether the construction of the (quadratic-size) VG. Lee and Preparata [160] use this approach to construct a shortest path map in optimal $O(n \log n)$ time for the case of obstacles that are parallel line segments (implying monotonicity of shortest paths with respect to the direction perpendicular to the segments). This approach also leads Reif and Storer [214] to an $O(hn + n \log n)$ time, $O(n)$ space, algorithm for general polygonal obstacles based on adding the obstacles one-at-a-time, updating the SPM(s) at each step using a shortest path algorithm for simple polygons (without holes).

Mitchell [174] shows how the Euclidean SPM(s) can be built in $O(kn \log^2 n)$ time, $O(n)$ space, where k is a quantity called the "illumination depth" (and is bounded above by the number of obstacles touched by a shortest path). This algorithm is based on a general technique for solving geometric shortest path problems, called the *continuous Dijkstra* paradigm (see [171, 176, 177, 174, 175, 181, 173]). The main idea is to simulate the "wavefront propagation" that occurs when running Dijkstra's algorithm in the continuum. The continuous Dijkstra paradigm has led to efficient algorithms for a variety of shortest path problems, as we mention later, including shortest paths on polyhedral surfaces [176], shortest paths through "weighted regions" [177], maximum "flows" in the continuum [173], and rectilinear paths among obstacles in the plane [175, 181].

A major open question in planar computational geometry is to devise a subquadratic-time algorithm for Euclidean shortest obstacle-avoiding paths. The only known lower bound is the trivial ($\Omega(n + h \log h)$) one.

Open Problem 10. *Given a polygonal domain with n vertices, compute a Euclidean shortest path between two points in $O(n \log n)$ time.*

4.4. Other Notions of "Short"

Instead of measuring the length of a path as its Euclidean length, several other objective functions are possible, as we describe below.

4.4.1. Rectilinear Metric

If we measure path length by the L_1 (or L_∞) metric ($d_1(p,q) = |p_x - q_x| + |p_y - q_y|$ or $d_\infty(p,q) = \max\{|p_x - q_x|, |p_y - q_y|\}$), or require that paths be *rectilinear*

(with edges parallel to the coordinate axes), then subquadratic-time algorithms for shortest paths in the plane are known.

For the general case of a polygonal domain P, Mitchell [175, 181] shows how to apply the continuous Dijkstra paradigm to build the L_1 (or L_∞) shortest path map in time $O(n \log n)$ (and space $O(n)$).

Clarkson, Kapoor, and Vaidya [64] develop a method based on principles similar the use of visibility graphs in searching for L_2 optimal paths: They construct a sparse graph (with $O(n \log n)$ nodes and edges) that is *path preserving*, meaning that it suffices for searching for shortest paths. This allows them to apply Dijkstra's algorithm, obtaining an $O(n \log^2 n)$ time ($O(n \log n)$ space) algorithm for L_1 shortest paths. Alternatively, this approach yields an $O(n \log^{1.5} n)$ time ($O(n \log^{1.5} n)$ space) algorithm [64, 253].

4.4.1.1. Fixed Orientations and Approximations. Methods for finding L_1 shortest paths generalize immediately to the case of *fixed orientation metrics* in which distances are measured in terms of the length of the shortest polygonal path whose links are restricted to a set of k fixed orientations (see Widmayer, Wu, and Wong [254]). (The L_1 and L_∞ metrics are special cases in which there are four fixed orientations, equally spaced by 90 degrees.) The result is an algorithm for finding shortest obstacle-avoiding paths in time $O(kn \log n)$ [175, 181].

We can apply the above result to get an approximation algorithm for Euclidean shortest paths by noting that the Euclidean metric is approximated to within accuracy $O(1/k^2)$ by the fixed orientation metric with k equally spaced orientations. The result is an algorithm that runs in time $O((n/\sqrt{\epsilon}) \log n)$ to produce a path guaranteed to have length within factor $(1 + \epsilon)$ of the Euclidean shortest path length [181]. Clarkson [62] also gives an approximation algorithm, using a related method, that computes an ϵ-optimal path in time $O(n/\epsilon + n \log n)$, after spending $O((n/\epsilon) \log n)$ time to build a data structure of size $O(n/\epsilon)$.

4.4.2. Minimum Link Paths

In some applications, the number of edges in a path, and not its length, is a more appropriate measure of the path complexity. In real life, for instance, while traveling in an unfamiliar territory, we tend to prefer directions with fewer turns, even at the expense of a slight increase in travel time. The technical motivation for minimizing the number of edges in a path arises from applications in robot motion planning, graph layouts, and telecommunication networks, where straight-line routing is often cheaper and preferable while "turning" is an expensive operation [189, 215, 233, 236]. One also encounters minimum link paths in solid modeling, where they are used for curve-compression and the approximation of univariate functions [188, 140, 169, 179, 115].

With this background, let us now formally define the notion of a minimum link path. We concentrate on two dimensions, but extensions to higher dimensions will be obvious. Given a polygonal domain P, a *minimum link path* between two points

s and t is a polygonal path with fewest possible number of edges that connects s to t while staying inside P. The *link distance* between s and t, denoted $d_L(s,t)$, is the number of edges in a minimum link path from s to t. (It is possible that there is no path from s to t avoiding all the obstacles, in which case the link distance is defined to be infinite.)

Most of the results on minimum link paths are for the case of a simple polygon, and so we discuss that first.

Minimum link paths in a simple polygon

Like other shortest path problems, a considerable effort has been spent on the simple polygon. In this case, the obstacle space consists of the boundary of a simple polygon P and the free-space consists of the interior of the polygon. Evidently, the notion of link distance is closely related to the notion of visibility. After all, the visibility polygon $V(s)$ consists of precisely the set of points whose link distance to s is one. Building upon this idea, Suri [233, 236] introduced the concept of a *window partition* of a polygon. The window partition of P with respect to s is a partition of the interior of P into cells over which the link distance to s is constant. Clearly, $V(s)$ is the cell with link distance 1. The cells with link distance 2 are the regions of $P - V(s)$ that are visible from the *windows* of $V(s)$; a window of $V(s)$ is an edge that forms a boundary between $V(s)$ and $P - V(s)$. The cells with larger link distance are obtained by iterating this procedure. Figure 4.4.1 shows an example of a window partition.

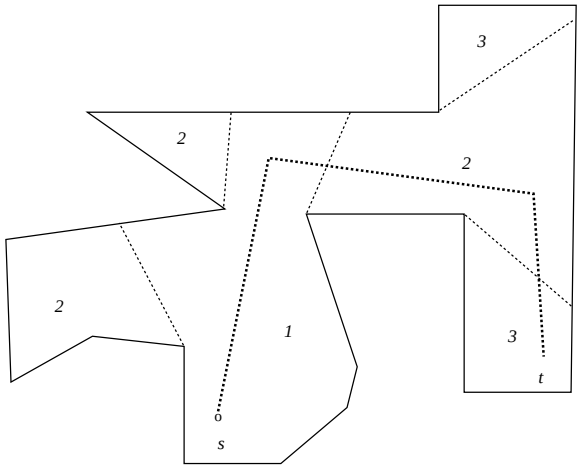

Fig. 4.4.1. The window partition of a polygon from point s. Numbers in regions denote their link distance from s. A minimum link path from s to t has three links.

Window partitions turn out to be a powerful tool for solving a number of minimum link path problems, both optimally as well as approximately. In [233], Suri

presents a linear time algorithm for computing the window partition of a triangulated simple polygon. Based on this construction, he derives the following results:

(i) The link distance from a fixed point s to all the vertices of P can be computed easily once the window partition from s has been computed: the link distance of a vertex v is k if the cell containing v has label k.

(ii) The window partition is a planar subdivision, which can be preprocessed in linear additional time to allow point location queries in logarithmic time (cf. Section2.5). With this preprocessing, the link distance from s to a query point t can be determined in $O(\log n)$ time.

(iii) The graph-theoretic dual of the window partition is a tree, called the *window tree*. Suri [236] observes that distances in the window tree nicely approximate the link distance between points. In particular, he shows how to calculate the link diameter of the polygon within ± 2 links in linear time; the *link diameter* is the maximum link distance between any two points of the polygon. More generally, the link-farthest neighbor of each vertex of P can also be computed within ± 2 links in linear time.

In all the cases above, a minimum link path can always be extracted in time proportional to the link distance. Suri [234] and Ke [147] propose $O(n \log n)$ time algorithms for computing the link diameter exactly. Another link-distance related concept that may have applications in shape analysis is *link center*: it is the set of points from which the maximum link distance to any point of P is minimized. Lenhart et al. [162] proposed an $O(n^2)$ time algorithm, based on window partitions, for computing the link center. This was subsequently improved to $O(n \log n)$, independently by Ke [147] and Djidjev, Lingas and Sack [79].

Recently, Arkin, Mitchell and Suri [22] have developed an $O(n^3)$ space data structure for answering link-distance queries in a simple polygon when *both* s and t are query points. Their data structure stores $O(n^2)$ window partitions. The query algorithm exploits information about the geodesic path between s and t. If it detects that the geodesic path has an *inflection edge* (i.e., the predecessor and the successor edges lie on opposite sides of the edge) or a *rotationally pinned edge* (i.e., the polygon touches the edges from both sides), then the link distance $d_L(s,t)$ is computed by searching the window partitions of the two polygon vertices that are associated with the inflection or pinned edge. If the path has neither an inflection nor a pinned edge, then it must be a *spiral* path, and this is the most complicated case. The query algorithm in this case uses *projection functions*, which are fractional linear forms, to track the other endpoint of a constant-turning path as its first endpoint moves linearly along an edge of P. The query algorithm of [22] works even if s and t are convex polygons instead of just points, however, the query time becomes $O(\log k \log n)$ if the two polygons have a total of k edges. In particular, if the polygons have a fixed number of edges, the query time is asymptotically optimal.

Open Problem 11. *Devise a data structure to answer 2-point link distance queries in a simple polygon. The data structure should use no more than* $O(n^2)$ *time and space for its construction and answer queries in* $O(\log n)$ *time.*

Minimum link paths among obstacles

With multiple obstacles, determining the "homotopy class" of an optimal path becomes a critical problem. Of course, the basic idea behind the window partition still holds: repeatedly compute visibility polygons until the destination point t is reached. However, unlike the simple polygon, where t is always separated from s by a unique window, there are multiple homotopically distinct paths in the general case. It requires a careful pruning technique to overcome a combinatorial explosion. There is essentially one result on link distance among general polygonal obstacles. Mitchell, Rote and Wöginger [178] present an $O(E_{VG} \log^2 n)$ time algorithm for finding a minimum link path between two fixed points among a set of polygonal obstacles with a total of n edges, where $E_{VG} = O(n^2)$ is the size of the visibility graph. The result of Mitchell et al. is only a first step; the problem of computing link distances among obstacles is far from solved. The only lower bound known is $\Omega(n \log n)$ [178].

Open Problem 12. *Given a polygonal domain having n vertices, compute a minimum-link path between two given points in time $O(n \log n)$ (or any subquadratic bound).*

The assumption of orthogonal obstacles and rectilinear paths results in significant simplifications. In [72], de Berg shows how to preprocess a rectilinear simple polygon in $O(n \log n)$ time and space to support $O(\log n)$ time *rectilinear* link distance between two arbitrary query points. In [74], de Berg et al. develop an $O(n \log n)$ space data structure for answering fixed-source link distance queries among orthogonal obstacles, with a total of n edges. The data structure requires $O(n^2)$ preprocessing time, and can answer a link distance query in $O(\log n)$ time. In fact, their data structure allows for the minimization of a *combined metric*, based on a fixed linear combination of the L_1 length and the link length: the cost of a rectilinear path is its L_1 length plus C times the number of turns, for some pre-specified constant $C \geq 0$. Subsequently, de Berg, Kreveld, and Nilsson [73] generalized the result of [74] to arbitrary dimensions. In d dimensions, their data structure requires $O((n \log n)^{d-1})$ space, $O(n^d \log n)$ preprocessing time, and supports fixed-source link distance queries in $O(\log^{d-1} n)$ time [73].

For the general link distance problem in higher dimensions, the only results known are approximations: Mitchell and Piatko [182] show that one can get within a constant factor (2) of the link distance in polynomial time (for any fixed d).

4.4.3. Weighted Regions

A natural generalization of the standard shortest obstacle-avoiding path problem is to consider varied terrain in which each region of the plane is assigned a weight that represents the cost per unit distance of traveling in that region. Clearly, the standard problem fits within this framework if we let obstacles have weight ∞ while free space has weight 1.

We can think of the "weighted plane" as being a network with an (uncountably) infinite number of nodes, one per point of the plane. We join every pair of points

with an edge, assigning a weight equal to the line integral of the weight function along the straight line segment joining the two points.

More formally, we consider the problem in which a planar polygonal subdivision $\mathcal{S}$ is given, with a weight $\alpha \in \{0, 1, \ldots, W, +\infty\}$ assigned to each face of the subdivision. We let n denote the total number of vertices describing the subdivision. Our objective is to find a path π from s to t that has minimum weighted length over all paths from s to t. (The weighted length of a path is given by the path integral of the weight function — it equals the weighted sum of its Euclidean lengths within each region.) This problem of finding an optimal path within a varied terrain is called the *Weighted Region Problem* (WRP), and was introduced by Mitchell and Papadimitriou [171, 177].

There are many potential applications of the WRP. The original motivation was to solve the minimum-time path problem for a point robot (without dynamic constraints) moving in a terrain of varied types: grassland, brushland, blacktop, marshland, bodies of water (obstacles to overland travel), and other types of terrain can each be assigned a weight according to the maximum speed at which a mobile robot can traverse the region. In this sense, the weights α denote a "traversability index," or the reciprocal of maximum speed.

Mitchell and Papadimitriou [177] present a polynomial-time solution to the WRP, based on the continuous Dijkstra paradigm, that finds a path guaranteed to be within a factor of $(1+\epsilon)$ of the optimal weighted length, where $\epsilon > 0$ is any user-specified degree of precision. The time complexity of the algorithm is $O(E \cdot S)$, where E is the number of "events" in the simulation of Dijkstra's algorithm, and S is the complexity of performing a numerical search to solve the following subproblem: Find a $(1+\epsilon)$-shortest path from s to t that goes through a given sequence of k edges of $\mathcal{S}$. It is known that $E = O(n^4)$ and that there are examples where E can actually achieve this upper bound (so that no better bound is possible) [177]. Mitchell and Papadimitriou also show that the numerical search can be done with a form of binary search that exploits the local optimality condition that an optimal path bends according to Snell's Law of Refraction when crossing a region boundary. This leads to a bound of $S = O(k^2 \log(nNW/\epsilon))$ on the time needed to perform a search on a k-edge sequence, where N is the largest integer coordinate of any vertex of the subdivision $\mathcal{S}$. Since one can show that $k = O(n^2)$, this yields an overall time bound of $O(n^8 L)$, where $L = \log(nNW/\epsilon)$ can be thought of as the bit complexity of the problem instance. Although the exponent looks particularly bad, we note that these are truly *worst-case* bounds; in the average case, we might expect that E behaves like n or n^2, and that k is effectively constant.

Many other papers are written on the WRP problem and its special cases; e.g., see [104, 231, 11–13]. A recent pair of papers by Kindl, Shing, and Rowe [148, 149] reports practical experience with a simulated annealing approach to the WRP. Papadakis and Perakis [202, 201] have generalized the WRP to the case of time-varying maps, where both the weights and the region boundaries may change over time; they obtain generalized local optimality conditions for this case and propose a search algorithm to find good paths.

4.5. Bicriteria Shortest Paths

The shortest path problem asks for paths that minimize some *one* objective function that measures "length" or "cost". Frequently, however, our application actually wants us to find paths to minimize *two or more* different costs. For example, in mobile robotics applications, we may wish to find a path that simultaneously is short in (Euclidean) length and has few turns.

Multi-criteria optimization problems tend to be hard. Even the bicriteria path problem in a graph is NP-hard [103]: Does there exist a path from s to t whose length is less than L and whose weight is less than W? Pseudo-polynomial time algorithms are known, and many heuristics have been devised (e.g., see [118, 122]).

Several geometric versions of bicriteria shortest path problems have recently been investigated. Various optimality criteria are of interest, including any pair from the following list: Euclidean (L_2) length, rectilinear (L_1) length, other L_p metrics, the number of turns in a path (its link length), the total amount of integrated turning done by a path, etc.

For example, applications in robot motion planning problems may require us to find a shortest (L_2) path constrained to have at most k links. To date, no exact method is known for this problem. Part of the difficulty is that a minimum-link path will not, in general, lie on the visibility graph (or any simple discrete graph).

Arkin et al. [22] show that, in a simple polygon, one can always find an s-t path whose link length is within a factor of 2 of the link distance from s to t, while *simultaneously* having Euclidean length within a factor of $\sqrt{2}$ of the Euclidean shortest path length. (A corresponding result is not possible for polygons with holes.)

Mitchell et al. [183] study the problem of finding shortest k-link paths in a simple polygon, P. They exploit the local optimality condition on the turning angles at consecutive bends of a shortest k-link path in order to devise a binary search scheme, tracing paths according to this local optimality criterion, in order to find the turning angle at the first bend point. The results of these searches are then combined via dynamic programming recursions to yield an algorithm that produces a path whose length is guaranteed to be within a factor $(1+\epsilon)$ of the length of shortest k-link path, for any user-specified tolerance ϵ. The algorithm runs in time polynomial in n, k and logarithmic in $1/\epsilon$ and the largest integer coordinate of any vertex of P.

For polygons with holes, we pose an interesting open question:

Open Problem 13. *Given a polygonal domain (with holes), what is the complexity of computing a shortest k-link path between two given points? Is it NP-complete to decide if there exists a path with at most k links and Euclidean length at most L?*

Several recent papers have addressed the bicriteria path problem for a combination of *rectilinear* link distance and L_1 length, in an environment of rectilinear obstacles. In de Berg et al. [74, 73], efficient algorithms are given in two and higher dimensions for computing optimal paths according to a "combined metric," which takes a linear combination of rectilinear distance and L_1 path length. (Note that this is not the same as solving the problem of computing Pareto-optimal solutions.)

Yang, Lee, and Wong [258, 257] give an $O(n \log^2 n)$ algorithm for computing a shortest k-bend path, a minimum-bend shortest path, or any combined objective that uses a monotonic function of rectilinear link length and L_1 length in a planar rectilinear environment. In all of these rectilinear problems, there is an underlying grid graph which can serve as a "path preserving graph". This immediately implies the existence of polynomial-time solutions to the various problems studied by [74, 73, 258, 257]; the contributions of these papers lie in their clever methods to solve the problems very efficiently.

Some lower bounds on bicriteria path problems have been established by Arkin at al. [21]. In particular, they show that the following geometric versions are NP-hard: (1) Given a polygonal domain, find a path whose L_2 length is at most L, and whose "total turn" is at most T; (2) Given a polygonal domain, find a path whose L_p length is at most λ_p and whose L_q length is at most λ_q $(p \neq q)$; and (3) Given a subdivision of the plane into red and blue polygonal regions, find a path whose travel through blue (resp. red) is bound by B (resp. R).

4.6. Higher Dimensions

While the shortest obstacle-avoiding path problem is solved efficiently in the plane, Canny and Reif [43, 46] show that the problem of finding shortest obstacle-avoiding paths according to any L_p $(1 \leq p \leq \infty)$ metric in three dimensions is NP-hard, even when all of the obstacles are convex polytopes.

The difficulty lies in the structure of shortest paths in three dimensions: They do not (necessarily) lie on any kind of discrete visibility graph. In general, shortest paths in a three-dimensional polyhedral domain P will be polygonal, with bend points that lie *interior* to edges of obstacles. The manner in which a shortest path bends at an edge is well constrained: It must enter and leave at the same angle to the edge. This implies that any locally optimal subpath joining two consecutive obstacle vertices can be "unfolded" at each obstacle edge that it touches, in such a way that the subpath becomes a straight segment.

The unfolding property of optimal paths can be exploited to yield polynomial-time algorithms in the special case in which the path must stay on a polyhedral *surface*. For the case of a *convex* surface, Sharir and Schor [230] give an $O(n^3 \log n)$ time algorithm for computing shortest paths. Their algorithm has been improved by Mount [186], who gives an $O(n^2 \log n)$ time algorithm for the same problem and shows how to use only $O(n \log n)$ space. For the case of shortest paths on a *nonconvex* polyhedral surface, O'Rourke, Suri, and Booth [199] give an $O(n^5)$ time algorithm. Mitchell, Mount, and Papadimitriou [176] improved the time bound to $O(n^2 \log n)$, giving an algorithm based on the continuous Dijkstra paradigm to construct a shortest path map for any given source point on an arbitrary polyhedral surface having n facets. Chen and Han [55] improve the algorithm of [176], obtaining an $O(n^2)$ time (and $O(n)$ space) bound. (See Aronov and O'Rourke [28] for the proof of the nonoverlap of the "star unfolding," required by [55].)

For the case when the domain P has only a few convex obstacles, Sharir [228]

has given an $n^{O(k)}$ algorithm for shortest paths, based on a careful analysis of the structure of shortest paths, and a bound of $O(n^7)$ on the number of distinct edge sequences that correspond to shortest paths on the surface of a convex polytope. Mount [187] has improved the bound on edge sequences to $O(n^4)$, which he shows to be tight. Schevon and O'Rourke [198] show a tight bound of $\Theta(n^3)$ on the number of *maximal* edge sequences for shortest paths. Agarwal et al.[1] give an $O(n^7 \log n)$ algorithm for computing all $O(n^4)$ edge sequences that correspond to shortest paths on a convex polytope.

For general three-dimensional polyhedral domains P, the best algorithmic results known are approximation algorithms. Papadimitriou [204] gives a fully polynomial approximation scheme that produces a path guaranteed to be no longer than $(1+\epsilon)$ times the length of a shortest path. His algorithm requires time $O(n^3(L+\log(n/\epsilon))^2/\epsilon)$, where L is the number of bits in an integer coordinate of vertices of P. Clarkson [62] also gives a fully polynomial approximation scheme, which improves upon that of [204] in the case that $n\epsilon^3$ is large.

While three-dimensional shortest path problems are known already to be hard, the proof (Canny and Reif [46]) is based upon a construction in which the size of the SPM is exponential. This leaves open an interesting algorithmic question of a potentially practical nature, since we may hope that "in practice" such huge SPM's will not arise:

Open Problem 14. *Given a polyhedral domain in 3 dimensions, compute a shortest path map in output-sensitive time.*

4.7. Kinetics and Other Constraints

4.7.0.1. Minimum Time Paths. Any real mobile robot has a bounded acceleration vector and a maximum speed. If we include these constraints in our model for path planning, then an appropriate objective is to minimize the *time* necessary for a (point) robot to travel from one point of free space to another, with the velocity vector known at the start and possibly constrained at the destination. In general, this kinodynamic planning problem is a very difficult optimal control problem. We are no longer in the nice situation of having optimal paths that are "taut string" paths, lying on a visibility graph. Instead, the paths will be complicated curves in free space, and the complexity of finding such optimal paths remains open.

In a first step towards understanding the algorithmic complexity of computing time-optimal trajectories under dynamic constraints, Canny et al [44] have produced a polynomial-time procedure for finding a *provably good* approximating time-optimal trajectory that is within a factor of $(1+\epsilon)$ of being a minimum-time trajectory. Their method is fairly straightforward — they discretize the four-dimensional phase space that represents position and velocity. Special care is needed, however, to ensure that the size of the grid is bounded by a polynomial in $1/\epsilon$ and n and the analysis to prove the effectiveness of the resulting paths is quite tedious.

Canny, Rege, and Reif [45] give an *exact* algorithm for computing an optimal path when there is an upper bound on the L_∞ norm of the velocity and acceleration

vectors. Their algorithm is based on characterizing a set of "canonical solutions" (related to "bang-bang" controls in one dimension) that are guaranteed to include an optimal solution path. Then, by writing an appropriate expression in the first-order theory of the reals, they obtain an exponential time, but polynomial space, algorithm. It remains an open question whether or not a polynomial-time algorithm exists.

4.7.0.2. Bounded Turning Radius. Related to the general problem of handling dynamic constraints, is the important problem of finding shortest paths subject to a bound on their *curvature*. Placing a lower bound on the curvature can be thought of as a means of handling an upper bound on the acceleration vector of a point robot whose speed is constant, or can be thought of as the realistic constraint imposed by the fact that many mobile robots have a bounded steering angle.

Fortune and Wilfong [96] gave an exponential-time decision procedure to determine whether or not it was possible for a robot to move from a start to a goal among a set of given obstacles, while obeying a lower bound on the curvature of its path (and not allowing reversals). If the point following the path *is* allowed to reverse direction, then Laumond [157] has shown that it is always possible to obtain a bounded curvature path if a feasible path exists.

Since the general problem seems to be extremely difficult, a restricted version has been studied: Wilfong [255, 256] considers the case in which the robot is to follow a given *network* of lanes, with the robot allowed to turn from one segment to another along a (bounded curvature) circular arc if the two lanes intersect. In Wilfong [255], a polynomial-time algorithm is given for producing *some* feasible path; in Wilfong [256], the problem of finding a shortest feasible path is shown to be NP-complete, while a polynomial-time method is given for deforming a given feasible path into a shortest equivalent feasible path. (The time bound is $O(k^3n^2)$, where n is the number of vertices describing the obstacles, and k is the number of turns in the path.)

4.8. Optimal Robot Motion

Most of our discussion is focused on the case of point robots. When the robot is not a point, the problem usually becomes much harder. An exception is the case of a circular robot (which is often a very good assumption anyhow) or a non-rotating convex robot. In the case of a circular robot, the problem of finding a shortest path among obstacles is solved almost as in the point robot case — we simply must "grow" the obstacles by the radius of the robot and "shrink" the robot to a point. This is the standard "configuration space" approach in motion planning, and leads to shortest path algorithms with time bounds comparable to the point robot case [57, 125, 171].

Optimal motion of rotating non-circular robots is a very hard problem. Consider the simplest case of moving a line segment ("ladder") in the plane. The motion

planning problem, which ignores any measure of "cost" of motion, is solvable in time $O(n^2 \log n)$ [261]. A natural definition of cost of motion for a ladder is to consider the *work* necessary to move the ladder from one place to another, assuming that there is a uniform coefficient of kinetic friction. Optimal motion of a ladder is an open problem at this point: Papadimitriou and Silverberg [205] and O'Rourke [196] give solutions for restricted cases of moving a ladder among obstacles, and Icking et al. [137] have characterized the solution for the general case *without* obstacles.

Open Problem 15. *Given a polygonal domain, compute an optimal motion of a ladder from one position to another.*

4.9. On-line Algorithms and Navigation Without Maps

In all of the path planning problems we have discussed so far, we have assumed that we know in advance the exact layout of the environment in which the robot moves — i.e., we assume we are given a perfect *map*. In most real problems, we cannot make this assumption. Indeed, if we are given a map or floorplan of where walls and obstacles are located, the map will invariably contain inaccuracies, and we may be interested also in being able to navigate around obstacles that may not be in the map. For example, for a robot moving in an office building, while the floorplan and desk layouts may be considered accurate and fixed, the location of a chair or a trashcan is something that we usually cannot assume to be known in advance.

When planning paths in the absence of perfect map information, we must have some model of *sensory inputs* that enable the robot to sense the local structure of its environment. Many different assumptions are possible here: visual sensors, range sensors (perhaps from sonar or computed from stereo imagery), touch sensors, etc. While numerous heuristic methods have been devised for sensor-based autonomous vehicle navigation (see [141]), only recently has there been interest in these questions from the theory of algorithms community.

Necessarily the theoretical results require stringent assumptions before anything can be claimed and proven. One of the first papers was by Lumelsky and Stepanov [164], who show that if a point robot is endowed only with a contact ("tactile") sensor, which can determine when it is in contact with an obstacle, then there is a strategy for "feeling" one's way from a start to goal such that the resulting path length is at most 1.5 times the total perimeter length of the set of obstacles. (The strategy, called "BUG2," is closely related to the strategy of keeping one's hand on the wall when going through a maze.) No assumptions have to be made about the shapes of the obstacles. Lumelsky and Stepanov show that this ratio is (essentially) best possible for this model; see [71] for some further work on an extension of the Lumelsky-Stepanov model.

An obvious complaint with the model of [164] is that it does not bound the *competitive ratio* — the worst-case ratio of the length of the actual path to that of an optimal path. Among the first results that bound the competitive ratio is

that of Papadimitriou and Yannakakis [206], who show that if the obstacles are assumed to be squares, one can achieve a competitive ratio of $\frac{\sqrt{26}}{3}$, and no strategy can achieve a ratio better than $\frac{3}{2}$. Further, by an adversary argument, they show that, for arbitrary (e.g., "thin") aligned rectangular obstacles and a robot that has perfect line-of-sight vision, there is no strategy with a bounded competitive ratio. See also [85].

Blum, Raghavan, and Schieber [41] show that if the obstacles are aligned (disjoint) rectangles in a square, n-by-n room, then there is a strategy using a tactile sensor that achieves competitive ratio $n2^{O(\sqrt{\ln n})}$. Bar-Eli, Berman, Fiat, and Yan [35] give a strategy that achieves competitive ratio $O(n \ln n)$, and show that no deterministic algorithm can yield an asymptotically better ratio (even if the robot is endowed with perfect vision).

Klein [152] has shown that if one is navigating in a simple polygon of a special structure (called a "street," in which it is possible for two guards to traverse the boundary of the polygon, while staying mutually visible and never backing up), then there is a strategy for a robot with perfect visibility sensing to achieve competitive ratio $1 + \frac{3}{2}\pi$.

For the problem of finding a short path from s to t among arbitrary unknown obstacles, Mitchell [172] has given a method of computing the best possible *local strategy*, assuming that the robot has perfect vision and can remember everything that has been seen so far, and assuming that one assigns a cost per unit distance of some fixed constant, α, for travel in terrain that has not yet been seen.

If, instead of simply asking for a path from s to t, our objective is to traverse a path that allows the entire space to be mapped out, then Deng, Kameda, and Papadimitriou [75] have shown that no competitive strategy exists, in general. If the number of obstacles is bounded, then they give a competitive strategy.

4.10. Motion Planning

There is a vast literature on the *motion planning problem* of finding any *feasible* path for a "robot" moving in a geometrically constrained environment; see for instance [156] and [131], and two survey articles [261] and [224]. A general paradigm in this field is to think of the motion of a d-degree-of-freedom robot as described by the motion of a single point in a d-dimensional *configuration space*, $\mathcal{C}$, in which the set of points representing feasible configurations of the system constitute "free space," $\mathcal{FP} \subset \mathcal{C}$.

Example: A simple example of this concept is given by the planar problem of planning the motion of a circular robot among a set of polygonal obstacles: We think of "shrinking" the robot to a point, while expanding the obstacles by the radius of the robot. The complement of the resulting "fattened" obstacles represents the free space for the disk. One can use the Voronoi diagram of the set of polygonal obstacles (treating the polygons as the "sources") to define a graph of size $O(n)$ (computable in time $O(n \log n)$ [262]) that can be searched

for a feasible path for a disk of any given size. This method, known as the "retraction" method of motion planning [261], solves this particular instance of the problem in time $O(n \log n)$.

Abstractly, the motion planning problem is that of computing a path between two points in the topological space $\mathcal{FP}$. In the first two of five seminal papers on the "Piano Movers' Problem" ([220–222, 229, 223], collected in the book [131]), Schwartz and Sharir show that the boundary of $\mathcal{FP}$ is a semialgebraic set (assuming the original constraints of the problem are semialgebraic). This then allows the motion planning problem to be written as a decision question in the theory of real closed fields (see Tarski [241]), which can be solved by adding appropriate adjacency information to the *cylindrical decomposition* that is (symbolically) computed by the algorithm of Collins [67]. For any fixed d and fixed degree of the polynomials describing the constraints, the complexity of the resulting motion planning algorithm is polynomial in n, the combinatorial size of the problem description.

Instead of computing a cell decomposition of $\mathcal{FP}$, an alternative paradigm in motion planning is to compute a lower dimensional subspace, $\mathcal{FP}' \subseteq \mathcal{FP}$, and to define a "retraction function" that maps $\mathcal{FP}$ onto $\mathcal{FP}'$. O'Dùnlaing and Yap [194] and O'Dùnlaing, Sharir, and Yap [195, 192, 193] have computed such retractions on the basis of Voronoi diagrams, obtaining efficient solutions to several low-dimensional motion planning problems.

Most recently, Canny [43] has described a method of reducing the motion planning problem to a (one-dimensional) graph search problem, by means of a "roadmap"; this is the currently best known method for general motion planning problems. The bottom line is that the motion planning problem can be solved in polynomial time (polynomial, that is, in the combinatorial complexity of the set of obstacles), for any *fixed* number of degrees of freedom of the robot.

Many lower bounds have also been established on motion planning problems. The first such results were by Reif [213], who showed that the generalized movers' problem (with many independently movable objects) is PSPACE-hard. Hopcroft, Joseph, and Whitesides [130] give PSPACE-hardness and NP-hardness results for several planar motion planning problems. See also the recent lower bounds paper by Canny and Reif [46].

5. Matching, Traveling Salesman, and Watchman Routes

Matching and traveling salesman are among the best known problems in combinatorial optimization. In this section, we survey some results on these problems where the underlying graph is induced by a geometric input.

5.1. Matching

5.1.1. Graph Matching

By a classical result of Edmonds, an optimal weighted matching in a general graph can be computed in polynomial time. Specifically, if $G = (V, E)$ is a graph with real-valued edge weights, then a minimum-weight maximum-cardinality matching in G can be found in polynomial time. Edmonds' is a primal-dual algorithm, which works by growing and shrinking the so-called "blossoms." Exactly how these blossoms are maintained and manipulated critically determines the running time of the algorithm. The original algorithm proposed by Edmonds could be implemented to run in worst-case time $O(n^4)$, where $n = |V|$ [94]; this was later improved to $O(n^3)$ by Lawler [158]. The last two decades have witnessed a flurry of research on further improving this time complexity, in particular, for sparse graphs. The latest result on this problem is due to H. Gabow, who presents an algorithm with the worst-case time complexity $O(n(m + n \log n))$, where the graph has n nodes and m edges [99].

5.1.2. Matching in the Euclidean Plane

A natural question from our point of view is this: can the $O(n^3)$ time bound for matching be improved if the graph is induced by a set of points in the plane? In other words, let S be a set of $2n$ points in the plane, and let G be the complete graph on the vertex set S, with the weight of an edge (u, v) being equal to the Euclidean distance between u and v. Does the geometry of the plane constrain an optimal matching sufficiently to admit a faster algorithm?

In the late seventies and early eighties, several conjectures were made regarding the relationship of minimum weight matching and other familiar geometric graphs, such as the Delaunay triangulation or minimum spanning tree [227]. In particular, it was conjectured that a minimum weight perfect matching of a set of points is a subset of the Delaunay triangulation of the points. Since triangulations are planar graphs, the validity of these conjectures would have immediately led to an order-of-magnitude improvement in the running time of the matching algorithm for the geometric case. Unfortunately, these conjectures all turned out to be false. Akl [10] showed that none of the following graphs is guaranteed to contain a minimum-weight perfect matching: Delaunay triangulation, minimum-weight triangulation, greedy triangulation, minimum-weight spanning tree.

Nevertheless, it turns out that a faster algorithm is possible for the matching of points in the plane. Vaidya [247] was able to improve the running time of Edmonds' algorithm from $O(n^3)$ to $O(n^{2.5} \log^4 n)$, using geometric data structures and a more careful choice of slack variables. He also gave improvements for bipartite matching and other metrics [247]. Vaidya's method depends on efficient solution to a bichromatic closest pair problem, where points may be deleted from one set and added to the other. Any improvements to the latter's solution would also improve the matching algorithm's running time.

Marcotte and Suri [165] considered a special case where all the points are in a convex position, i.e., they form the vertices of a convex polygon. The problem retains

much of its complexity even for this restricted class of input, as it can be shown that all the counterexamples of Akl [10] still hold. But, surprisingly, Marcotte and Suri were able to devise a much simpler and significantly faster algorithm for matching. Their algorithm is based on divide-and-conquer and runs in time $O(n \log n)$. There are two key ideas in their algorithm: an extensibility lemma and vertex weights. The extensibility lemma establishes a geometric condition under which a certain subset of the edges can be immediately added to the optimal matching. The vertex weights are real numbers carefully chosen in such a way that we can invoke the extensibility lemma on the weighted nearest-neighbor graph. The algorithm in [165] also solves the assignment problem in the same time bound, and it also extends to the case where the points lie on the boundary of a simple polygon and the weight of an edge (u, v) is the length of the shortest path from u to v inside the polygon. X. He [119] gives a parallel version of the Marcotte-Suri algorithm that runs in $O(\log^2 n)$ time with $O(n)$ processors on a PRAM.

There also are numerous approximation algorithms for matching. For uniform distributions of points, Bartholdi-Platzman [36] and Dyer-Frieze [84] describe fast heuristics that give matchings with total weight close to optimal as $n \to \infty$. Vaidya [246] describes an approximation algorithm that has a guaranteed performance for any input and works for any fixed dimension. His algorithm produces a matching with weight at most $(1+\varepsilon)$ times the weight of a minimum weight matching, and runs in time roughly $O(n^{1.5} \log^{2.5} n)$; the constant of proportionality is of the order of $(d/\varepsilon)^{1.5d}$, where d is the dimension of input space.

Despite the failure of earlier conjectures relating an optimal matching to other well-known geometric graphs, such as the Delaunay triangulation, it remains a reasonable question whether one can define certain easily constructed, sparse graphs that are guaranteed to contain an optimal geometric matching. The ultimate question, of course, is to determine the extent to which the geometry of the plane can be exploited in the matching problem.

Open Problem 16. *Give an $o(n^2)$ time algorithm for computing a minimum-weight complete matching for a set of $2n$ points in the plane.*

Interestingly, a result of Marcotte and Suri [165] shows that, for the vertices of a convex polygon, finding a maximum-weight matching is substantially easier than finding a minimum-weight matching. A natural question then is: does the same hold for a general set of points.

Open Problem 17. *Give an $o(n^2)$ time algorithm for computing a maximum-weight complete matching for a set of $2n$ points in the plane.*

5.1.3. Non-crossing matching

There is a celebrated Putnam Competition problem on non-crossing matching (see [155]). Given two sets of points R (red) and B (blue) in the plane, with n points each, find a matching of R and B using straight line segments so that no two segments cross; clearly, we must assume that the points are in general position.

There are several proofs of the fact that a non-crossing matching always exists. We give just one: pick a matching that minimizes the sum of all line segment lengths in the matching; by the triangle inequality, no two segments in this matching can cross. Akiyama and Alon [9] extend this result to arbitrary dimensions: given d sets of points in d-space, each set containing n points, we can always find n pairwise disjoint simplices, each with one vertex from each set.

The algorithmic problem of finding such a matching was first considered by Atallah [31], who gave an $O(n \log^2 n)$ time algorithm for the two-dimensional problem. Later, Hershberger and Suri [128] were able to obtain an $O(n \log n)$ time algorithm for the same problem; this time bound is also optimal in the algebraic tree model of computation. Finding a non-intersecting simplex matching in d-dimensions, for $d \geq 3$, remains an open problem.

A minimum-weight matching in the plane is always non-crossing. On the other hand, a maximum-weight matching generally has many crossing edges. An interesting question is to compute a maximum-weight matching with *no crossings*. To the best of our knowledge, no polynomial time algorithm is known for this problem.

Open Problem 18. *Given $2n$ points in general position the plane, find a non-crossing maximum-weight matching.*

A very recent result of Alon, Rajagopalan and Suri [14] gives a simple and efficient approximation algorithm for the above problem. Their algorithm produces a non-crossing matching of length at least $2/\pi$ times the longest matching, and takes $o(n^{5/2} \log n)$ time. Alternatively, they can find a non-crossing matching of length at least $\frac{2}{\pi}(1-\varepsilon)$ times the optimal in $O(n \log n/\sqrt{\varepsilon})$, for any $\varepsilon > 0$. Somewhat surprisingly, Alon et al. [14] show that their approximate matching is within $2/\pi$ factor of even the longest *crossing* matching. Similar results are also obtained for the non-crossing *Hamiltonian path* problem and the non-crossing *spanning tree* problem.

5.2. Traveling Salesman and Watchman Routes

It is well-known that the traveling salesman problem remains NP-complete even when restricted to the Euclidean plane [203]. The best heuristics known for approximating a Euclidean TSP are the same that work for graphs whose edge weights obey the triangle inequality. In particular, a performance ratio of 2 is achieved by double-traversing the MST, and a ratio of 1.5 is achieved by the heuristic of Christofides [61]. It remains an outstanding open problem whether the ratio of 1.5 can be improved for the geometric problem.

Open Problem 19. *Give a polynomial time algorithm for approximating the Euclidean TSP of n points with a performance ratio strictly less than 1.5.*

Within computational geometry, the TSP has not received much consideration. A slightly related problem that elicited some interest was the question: Does the

Delaunay triangulation of a set of points contain its traveling salesman tour? This, not too surprisingly, was answered negatively, first by Kantabutra [145] for degenerate set of points, and later by Dillencourt [78] for general-position points. The question of the "Hamiltonicity" of Delaunay triangulations also arose in the context of pattern recognition and shape representation, in a paper by O'Rourke et al. [197]. Dillencourt [78] shows that Delaunay triangulation graphs are 1-tough,[1] partly explaining why in many practical cases the triangulations turned out to be Hamiltonian.

A problem that ties together the traveling salesman type issues with visibility issues is the *watchman route* problem. Given a polygonal region of the plane (possibly with holes), the problem is to compute a shortest cycle such that every point on the boundary of the region is visible from *some* point on the cycle. If the region is the interior of a simple polygon (without holes), and we are given a starting point through which the route must pass, then Chin and Ntafos [60] give an $O(n^4)$ algorithm to compute an optimal route; for orthogonal polygons, the time bound improves to $O(n)$. Tan, Hirata, and Inagaki [239] have recently given an $O(n^3)$ algorithm for finding an optimal watchman route through a given point in a simple polygon. However, the problem becomes NP-complete for a polygon with holes or for a simple three-dimensional polyhedron [59]. Other results on watchman route type problems can be found in [191, 153, 180, 104].

6. Shape Analysis, Computer Vision, and Pattern Matching

Applications in computer-aided design, machine vision, and pattern matching all have need to describe, represent, and reason about "shapes". Computational geometry has addressed many questions regarding shapes, such as: How can we compare two shapes? How can we detect when a shape is present within a digital image? How can a shape be represented efficiently in order to expedite basic geometric queries (such as intersection detection)?

Here, we think of a *shape* as being the image ("orbit") of some collection of points (countable or uncountable) under the action of some group of transformations, $\mathcal{T}$ (e.g., translation, rotation, rigid motions, etc.). Thus, a shape may be represented by a finite collection of points in d-space, or by a polygon, etc. A shape may be given to us in any of a number of forms, including a binary array (of "pixels" that comprise the shape), a discrete set of points, a boundary description of a solid, a CSG (Constructive Solid Geometry) representation in terms of boolean set operations on primitive solids (halfspaces), a Binary Space Partition tree, etc. When we speak of the "complexity" of a shape, we mean the combinatorial size of its representation (e.g., the number of vertices defining the boundary of a polygon).

The fields of computer vision and pattern matching have motivated the study of many shape analysis problems in computational geometry over the last decade.

[1] A connected graph G is called *1-tough* if deletion of any k nodes splits G in to at most k connected components.

Early work on shape analysis focussed on the use of planar convex hulls ([39, 243]), decompositions of simple polygons into convex pieces ([49]), etc.

In the last five years, effort has concentrated on several problems in shape comparison based on precisely defined notions of distance functions and geometric matching. The goal has been to define a meaningful notion of shape resemblance that is efficiently computable.

6.1. Shape Comparison

A very natural and general definition of shape distance can be based on the *Hausdorff metric*, which we now define precisely. Let A and B denote two given shapes, and let $\tau \in \mathcal{T}$ denote a transformation (in group $\mathcal{T}$), such as translation and/or rotation, under which we consider shapes to be equivalent. Then, the Hausdorff distance between shapes A and B is defined to be

$$d^{(H)}(A, B) = \min_{\tau \in \mathcal{T}} d_H(A, \tau(B)),$$

where d_H denotes standard Hausdorff distance

$$d_H(A, B) = \max\{\sup_{a \in A} \inf_{b \in B} \delta(a, b), \sup_{b \in B} \inf_{a \in A} \delta(a, b)\},$$

for some underlying distance function δ defined on pairs of points.

The problem of computing the Hausdorff distance between sets of points or between polygons, under various allowed transformations, has been addressed in several recent papers [6, 15, 133, 135, 134, 218].

Rote [218] shows that the Hausdorff distance between two sets of points on the real line can be computed in time $O(n \log n)$, and this is best possible. Huttenlocher and Kedem [133] show how to compute the Hausdorff distance between two sets of points (of sizes m and n) in the plane under translations τ in time $O((mn)^2 \alpha(mn))$, where $\alpha(\cdot)$ denotes the inverse Ackermann function. Huttenlocher et al. [135] improve the time bound to $O(mn(m+n)\alpha(mn) \log mn)$. They also show how to compute the Hausdorff distance between sets of (disjoint) line segments (under translation) in time $O((mn)^2 \log mn)$, assuming the underlying metric δ is L_1 or L_∞. Chew and Kedem [58] have recently shown that the Hausdorff distance between two point sets can be computed in time $O(n^2 \log^2 n)$, assuming the underlying metric δ is L_1 or L_∞.

Alt, Behrends, and Blömer [15] study the problem of computing Hausdorff distance between simple polygons in the plane, under a variety of possible transformations τ with underlying metric $\delta = L_2$. They give an $O(n \log n)$ algorithm for computing Hausdorff distance between two simple polygons (without transformation), an $O((mn)^3(m+n)\log(m+n))$ algorithm for Hausdorff distance under translation, several algorithms with high-degree polynomial time bounds for various types of transformations, and *approximate* algorithms for these cases that require time $O(nm \log^2 nm)$.

Agarwal, Sharir, and Toledo [6] show how parametric search can be used to improve the complexity to $O((mn)^2 \log^3(mn))$ for the case of comparing two simple polygons under translation (and $\delta = L_2$).

Most recently, Huttenlocher, Kedem, and Kleinberg [134] examine the case of rigid body motions (translation and rotation) of point sets in the plane, and obtain an algorithm with time complexity of $O((m+n)^6 \log(mn))$.

One drawback of the Hausdorff metric for shape comparison is that it measures only the "outliers" — points that are worst-case.

The polygon metric defined by Arkin et al. [19] avoids some of the problems associated with the Hausdorff metric. Basically, [19] give an efficient ($O(n^2 \log n)$) means of computing the (L_2) distance between the "turning functions" of two simple polygons (scaled to have the same perimeter), under all possible shifts of the origins of the parameterizations. (The *turning function* of a polygon measures the accumulated angle of the counterclockwise tangent as a function of the arc-length, starting from some reference point on the boundary.) Rote [219] has suggested the use of the bounded Lipschitz norm for comparing two single-variable functions (e.g., turning functions), and gives an $O(n \log n)$ time method to compute it. The metrics given by [19, 219] have the disadvantage of not applying as generally as does the Hausdorff; for example, neither metric extends readily to the case of polygons with holes or to higher dimensions.

Alt and Godau [16] study the so-called *Fréchet-Metric* between curves, and give an $O(mn)$ algorithm to decide if the Fréchet distance between two fixed polygonal chains (of sizes m and n) is less than a given $\epsilon > 0$, and, using this with parametric search, they compute the Fréchet distance between two fixed chains in time $O(mn \log^2 mn)$. They do not optimize over a transformation group; it would be interesting to devise an efficient method to do so.

6.2. Point Pattern Matching

A special case of the shape comparison problem is that of matching two discrete sets of points: Find a transformation of a set of points B that makes it "match" most closely a set of points A. Matching problems are present throughout the computer vision literature, since their solution forms a fundamental component in many object recognition systems (e.g., see [132]).

More precisely, the *point matching* problem in computer vision can be stated as follows: Given a set of n *image* points $A = \{a_1, \ldots, a_n\} \subset \mathbb{R}^d$ and a set of m *model points* $B = \{b_1, \ldots, b_m\} \subset \mathbb{R}^{d'}$, determine a *matching* μ (i.e., a list of pairs (a_i, b_j) such that no two pairs share the same first element or the same second element) and a transformation $\tau : \mathbb{R}^d \to \mathbb{R}^{d'}$, within an allowable class of transformations $\mathcal{T}$, such that the application of τ to point a_i brings it into "correspondence" to point b_j, for each pair $(a_i, b_j) \in \mu$. The "value" of a matching can be taken to be the number of pairs (a_i, b_j), or possibly a sum of weights.

The term "correspondence" can take on several different meanings. In the *exact point matching problem* (also known as the "image registration problem"), we re-

quire that $\tau(a_i) = b_j$ for every pair $(a_i, b_j) \in \mu$ of the matching. In the *inexact point matching problem* (also known as the "approximate congruence problem"), we only require that $\tau(a_i)$ be *close* to b_j, for each $(a_i, b_j) \in \mu$. A natural definition of closeness is to define for each model point b_j, a "noise region" B_j, and to say that $\tau(a_i)$ is "close" to b_j if $\tau(a_i) \in B_j$. We let $\mathcal{R} = \{B_1, \ldots, B_m\}$ denote the set of noise regions. Refer to Figure 6.2.

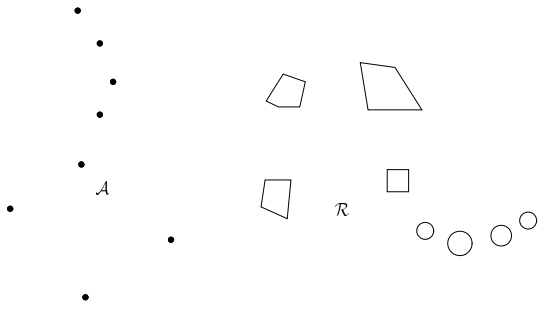

Fig. 6.2.1. A point matching problem

The exact point matching problem has been solved in time $O(n^{d-2} \log n)$ for $d = d'$ and $\mathcal{T}$ the set of congruences (translations and rotations) (see [17]).

Baird [34] formalizes the inexact point matching problem and provides algorithms for the case of similarity transformations and convex polygonal noise regions; his algorithms are worst-case exponential, and he leaves open the question of solving the problem in polynomial time. This open question is resolved in the work of Alt et al [17] and the work of Arkin et al. [23], where it is shown that many versions of the inexact matching problem can be solved in polynomial time, for various assumptions about the allowed transformations and the shapes of the noise regions. Arkin et al. also give lower bounds on the number of possible matches and generalize the problem to allow arbitrary piecewise-linear cost functions for the matching. Arkin et al. [20] give improved algorithms and combinatorial bounds on the number of matches for several special cases of the inexact point matching problem in which the noise regions are assumed to be disjoint.

A major obstacle to making the existing methods of point matching practical is the very high degree polynomial time bounds. For example, even for the case of point matching under translation, the algorithm of [17] requires time $O(n^6)$. One possible direction for an improvement has been suggested by Heffernan and Schirra [121], who show one can get low-degree polynomials (in n) if one allows an *approximate* decision procedure, which is allowed to give an "I don't know" answer in response to situations in which the data is particularly "bad".

Zikan [263] and Aurenhammer et al. [33] have studied problems in which the objective function is based on least-squares.

6.3. Shape Approximation

A requirement for any system that analyzes physical models is the representation of geometric data, such as points, polygons, polyhedra, and general solids. One would like to have as compact a representation as possible, while still capturing the degree of precision required by the problem at hand. In particular, this issue is important for real-time systems whose algorithms have running times that depend on the size of the data according to some high degree polynomial (e.g., vision systems, motion planning systems, etc.).

For example, cartographers are interested in the question of approximating general planar objects, such as polygons with holes, sets of polygons, or general planar maps. A geographic terrain map may have millions of polygonal cells, some of which are large and open, others of which are tiny or quite contorted. Such would be the case if we were to look at an agricultural use map of the United States or at a segmentation of a digitized image. But, if we were to put on a pair of "ϵ-blurring eyeglasses", what we would see in such a map is a subdivision with a few "large" (in comparison with ϵ) cells, and blurred "gray masses" where the cell structure is quite fine (in comparison with ϵ). Refer to Figure 6.3.1. We would like to replace the original subdivision with a new one of lower resolution (or perhaps a hierarchy of many different resolutions).

A standard approach to the map simplification problem is to take each polygonal curve that defines a boundary in the map and replace it by a simpler one, subject to the new curve being "close" to the original curve. Cartographers have been interested in this "line simplification problem" for some time ([80, 168]). Computational geometers have defined and solved several instances of the problem; see [115, 127, 138–140, 169]. The general method has been to model the problem as an "ordered stabbing" question, in which one wants to pass a polygonal curve through an ordered set of "fattened" boundary elements (e.g., disks centered on vertices) from the original curve.

Guibas et al. [115] have noted that simplifying each boundary curve of a map individually can cause all kinds of topological inconsistencies, such as islands becoming inland, intersections among boundaries that were previously disjoint, etc. Even the special case of the cartographer's problem in which one wants to approximate a *single* simple polygon, P, suffers from the potential problem that the approximating curve is not simple.

In particular, consider an "ϵ-fattening" of the boundary of P to be the set of all ("gray") points within distance ϵ of the boundary of P. The boundary of the gray region can be computed in time $O(n \log n)$ by finding the Voronoi diagram of P. If the fattened gray region is an annulus, then we are lucky: The minimum-link cycle algorithms of Aggarwal et al. [7], Wang and Chan [251], or Ghosh et al. [105] can be applied to give an exact answer to the problem in $O(n \log n)$ or $O(n)$ time. For

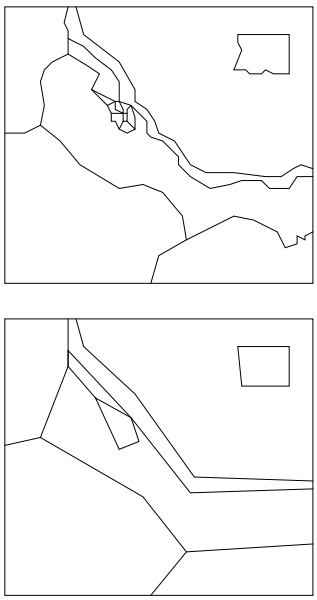

Fig. 6.3.1. The original map (top) and its simplification (bottom).

larger values of ϵ, however, the fattening may create more holes, in which case, one wants a minimum-vertex *simple* polygon surrounding all the holes of the fattened region. Guibas et al. [115] give an $O(n \log n)$ time algorithm to compute a simple polygon with at most $O(h)$ vertices more than optimal, where h is the number of holes in the fattening; they conjecture that the exact solution of the problem is NP-hard.

Mitchell and Suri [179] have studied a related problem of finding a minimum-link subdivision that separates a given set of polygons. They give an $O(n \log n)$ time algorithm, based on computing minimum-link paths in the "moats" between polygons, that produces an approximating subdivision (or a *separating family*) that is guaranteed to be within a constant factor of optimality. The exact solution of the problem has been shown to be NP-hard by Das and Joseph [70].

Polyhedral Separation/Approximation

The generalization of the boundary approximation problem to three dimensions is of primary importance for any real CAD applications. If we are given a polyhedral surface, how can we approximate it with a significantly simpler polyhedral surface?

One approach is to "ϵ-fatten" the original surface and then look at simplifying surfaces that lie within the fattened region. Thus, we ask the following polyhedral separation question: Given two polyhedral surfaces, P and Q, find a polyhedral surface Σ of minimum facet complexity that separates P from Q.

Das and Joseph [68, 70, 69] have shown that this problem is NP-hard, even for convex surfaces P and Q. Mitchell and Suri [179] have shown that if P and Q are convex, one can, in time $O(n^3)$, compute a separating surface whose facet complexity is guaranteed to be within a small, logarithmic, factor of the size of an optimal separator. While the preliminary results of [179] are interesting as a first step, many questions remain to be addressed, particularly with respect to nonconvex surfaces.

Open Problem 20. *Given two nonconvex polyhedra P and Q, with a total of n faces, find a polyhedral surface of $f(n)$ faces that separates P from Q such that $f(n)$ is within a small factor of the optimal.*

7. Conclusion

In this survey, we touched upon some of the major problem areas and techniques of computational geometry. Our emphasis was on optimization problems that should be of most interest to the Operations Research community. We certainly have not done justice to the field of computational geometry as a whole, and have left out entire subareas of intense research. But we hope to have supplied sufficient pointers to the literature that an interested reader can track down more detailed information on any particular subtopic.

Computational geometry is a very young discipline, and while it has matured extremely rapidly in the last ten years, we expect a steady stream of new results to

continue. Particularly, as the interaction between more applied fields and computational geometry grows, entire new lines of investigation are expected to evolve.

8. Acknowledgement

We thank Joseph O'Rourke and Godfried Toussaint for several helpful comments that have improved the presentation of this survey.

References

[1] P. K. Agarwal, B. Aronov, J. O'Rourke, and C. Schevon. Star unfolding of a polytope with applications. In J. R. Gilbert and R. Karlsson, editors, *Proc. of 2nd Scandanavian Workshop on Algorithm Theory*, pages 251–263. Springer-Verlag, July 1990. Lecture Notes in Computer Science, Vol. 447.

[2] P. K. Agarwal, H. Edelsbrunner, O. Schwarzkopf, and E. Welzl. Euclidean minimum spanning trees and bichromatic closest pairs. *Discrete Comput. Geom.*, 6:407–422, 1991.

[3] P. K. Agarwal and J. Matoušek. Relative neighborhood graphs in three dimensions. In *Proc. 3rd ACM-SIAM Sympos. Discrete Algorithms*, pages 58–65, 1992.

[4] P. K. Agarwal, J. Matoušek, and S. Suri. Farthest neighbors, maximum spanning trees and related problems in higher dimensions. *Computational Geometry: Theory and Applications*, 1, pp. 189–201, 1992.

[5] P. K. Agarwal and M. Sharir. Planar geometric location problems and maintaining the width of a planar set. In *Proc. 2nd ACM-SIAM Sympos. Discrete Algorithms*, pages 449–458, 1991.

[6] P. K. Agarwal, M. Sharir, and S. Toledo. Applications of parametric searching in geometric optimization. In *Proc. 3rd ACM-SIAM Sympos. Discrete Algorithms*, pages 72–82, 1992.

[7] A. Aggarwal, H. Booth, J. O'Rourke, S. Suri, and C. K. Yap. Finding minimal convex nested polygons. *Inform. Comput.*, 83(1):98–110, October 1989.

[8] A. Aggarwal, L. J. Guibas, J. Saxe, and P. W. Shor. A linear-time algorithm for computing the Voronoi diagram of a convex polygon. *Discrete Comput. Geom.*, 4:591–604, 1989.

[9] J. Akiyama and N. Alon. Disjoint simplices and geometric hypergraphs. *Annals New York Academy of Science*, pages 1–3, 1989.

[10] S. Akl. A note on Euclidean matchings, triangulations and spanning trees. *J. of Combnatorics, Information and System Sciences*, pages 169–174, 1983.

[11] R. Alexander. Construction of optimal-path maps for homogeneous-cost-region path-planning problems. Ph.D. Thesis, Computer Science, U.S. Naval Postgraduate School, Monterey, CA, 1989.

[12] R. Alexander and N. Rowe. Geometrical principles for path planning by optimal-path-map construction for linear and polygonal homogeneous-region terrain. Technical report, Computer Science, U.S. Naval Postgraduate School, Monterey, CA, 1989.

[13] R. Alexander and N. Rowe. Path planning by optimal-path-map construction for homogeneous-cost two-dimensional regions. In *IEEE Int. Conf. Robotics and Automation*, 1990.

[14] N. Alon, S. Rajagopalan and S. Suri. Long Non-Crossing Configurations in the Plane. Technical Report, Bell Communications Research, 1992.

[15] H. Alt, B. Behrends, and J. Blömer. Approximate matching of polygonal shapes. In *Proc. 7th Annu. ACM Sympos. Comput. Geom.*, pages 186–193, 1991.

[16] H. Alt and M. Godau. Measuring the resemblance of polygonal curves. In *Proc. 8th Annu. ACM Sympos. Comput. Geom.*, pages 102–109, 1992.

[17] H. Alt, K. Mehlhorn, H. Wagener, and E. Welzl. Congruence, similarity and symmetries of geometric objects. *Discrete Comput. Geom.*, 3:237–256, 1988.

[18] H. Alt and E. Welzl. Visibility graphs and obstacle-avoiding shortest paths. *Zeitschrift für Operations Research*, 32:145–164, 1988.

[19] E. M. Arkin, L. P. Chew, D. P. Huttenlocher, K. Kedem, and J. S. B. Mitchell. An efficiently computable metric for comparing polygonal shapes. *IEEE Trans. Pattern Anal. Mach. Intell.*, 13(3):138–148, 1991.

[20] E. M. Arkin, K. Kedem, J. S. B. Mitchell, J. Sprinzak, and M. Werman. Matching points into pairwise-disjoint noise regions: combinatorial bounds and algorithms. *ORSA Journal on Computing*, 4(4):375–386, 1992.

[21] E. M. Arkin, J. S. B. Mitchell, and C. D. Piatko. Bicriteria shortest path problems in the plane. In *Proc. 3rd Canad. Conf. Comput. Geom.*, pages 153–156, 1991.

[22] E. M. Arkin, J. S. B. Mitchell, and S. Suri. Optimal link path queries in a simple polygon. In *Proc. 3rd ACM-SIAM Sympos. Discrete Algorithms*, pages 269–279, 1992.

[23] E. M. Arkin, J. S. B. Mitchell, and K. Zikan. Algorithms for point matching problems. Manuscript, School Oper. Res. Indust. Engrg., Cornell Univ., Ithaca, NY, 1989.

[24] B. Aronov. On the geodesic Voronoi diagram of point sites in a simple polygon. *Algorithmica*, 4:109–140, 1989.

[25] B. Aronov, H. Edelsbrunner, L. Guibas, and M. Sharir. Improved bounds on the complexity of many faces in arrangements of segments. Report 459, Dept. Comput. Sci., New York Univ., New York, NY, July 1989.

[26] B. Aronov, S. J. Fortune, and G. Wilfong. The furthest-site geodesic Voronoi diagram. In *Proc. 4th Annu. ACM Sympos. Comput. Geom.*, pages 229–240, 1988.

[27] B. Aronov, J. Matoušek, and M. Sharir. On the sum of squares of cell complexities in hyperplane arrangements. In *Proc. 7th Annu. ACM Sympos. Comput. Geom.*, pages 307–313, 1991.

[28] B. Aronov and J. O'Rourke. Nonoverlap of the star unfolding. In *Proc. 7th Annu. ACM Sympos. Comput. Geom.*, pages 105–114, 1991.

[29] Ta. Asano, Te. Asano, L. J. Guibas, J. Hershberger, and H. Imai. Visibility of disjoint polygons. *Algorithmica*, 1:49–63, 1986.

[30] Te. Asano, B. Bhattacharya, J. M. Keil, and F. Yao. Clustering algorithms based on minimum and maximum spanning trees. In *Proc. 4th Annu. ACM Sympos. Comput. Geom.*, pages 252–257, 1988.

[31] M. Atallah. A matching problem in the plane. *Journal of Computer and System Sciences*, 31:63–70, 1985.

[32] F. Aurenhammer. Voronoi diagrams: a survey of a fundamental geometric data structure. *ACM Comput. Surv.*, 23:345–405, 1991.

[33] F. Aurenhammer, F. Hoffmann, and B. Aronov. Minkowski-type theorems and least-squares partitioning. In *Proc. 8th Annu. ACM Sympos. Comput. Geom.*, pages 350–357, 1992.

[34] H.S. Baird. *Model-Based Image Matching Using Location.* Distinguished Dissertation Series. MIT Press, 1984.

[35] E. Bar-Eli, P. Berman, A. Fiat, and P. Yan. On-line navigation in a room. In *Proc. 3rd ACM-SIAM Sympos. Discrete Algorithms*, pages 237–249, Orlando, FL, 1992.

[36] J. J. Bartholdi, III and L. K. Platzman. A fast heuristic based on spacefilling curves for minimum-weight matching in the plane. *Inform. Process. Lett.*, 17:177–180, 1983.

[37] R. Bar-Yehuda and B. Chazelle. Triangulating a set of non-intersecting and simple polygonal chains. Manuscript, Computer Science, Tel-Aviv University, 1992.

[38] M. Ben-Or. Lower bounds for algebraic computation trees. In *Proc. 15th Annu. ACM Sympos. Theory Comput.*, pages 80–86, 1983.

[39] B. K. Bhattacharya. Applications of computational geometry to pattern recognition problems. Ph.D. Thesis, School Comput. Sci., McGill Univ., Montreal, PQ, 1980.

[40] B. K. Bhattacharya and G. T. Toussaint. On geometric algorithms that use the furthest-point Voronoi diagram. In G. T. Toussaint, editor, *Computational Geometry*, pages 43–61. North-Holland, Amsterdam, Netherlands, 1985.

[41] A. Blum, P. Raghavan, and B. Schieber. Navigating in unfamiliar geometric terrain. In *Proc.*

23rd Annu. ACM Sympos. Theory Comput., pages 494–503, 1991.

[42] K. Q. Brown. Geometric transforms for fast geometric algorithms. Ph.D. Thesis and Report CMU-CS-80-101, Dept. Comput. Sci., Carnegie-Mellon Univ., Pittsburgh, PA, 1980.

[43] J. Canny. The complexity of robot motion planning. Ph.D. Thesis, Electrical Engineering and Computer Science, Massachusetts Institute of Technology, 1987.

[44] J. Canny, B. R. Donald, J. Reif, and P. Xavier. On the complexity of kinodynamic planning. In *Proc. 29th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 306–316, 1988.

[45] J. Canny, A. Rege, and J. Reif. An exact algorithm for kinodynamic planning in the plane. *Discrete Comput. Geom.*, 6:461–484, 1991.

[46] J. Canny and J. H. Reif. New lower bound techinques for robot motion planning problems. In *Proc. 28th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 49–60, 1987.

[47] D. R. Chand and S. S. Kapur. An algorithm for convex polytopes. *J. ACM*, 17:78–86, 1970.

[48] B. Chazelle. A theorem on polygon cutting with applications. In *Proc. 23rd Annu. IEEE Sympos. Found. Comput. Sci.*, pages 339–349, 1982.

[49] B. Chazelle. Approximation and decomposition of shapes. In *Advances in Robotics 1: Algorithmic and Geometric Aspects of Robotics*, pages 145–185, J. T. Schwartz and C.-K. Yap, editors. Lawrence Erlbaum Associates, Hillsdale, NJ, 1987.

[50] B. Chazelle. An optimal convex hull algorithm and new results on cuttings. In *Proc. 32nd Annu. IEEE Sympos. Found. Comput. Sci.*, pages 29–38, 1991.

[51] B. Chazelle. Triangulating a simple polygon in linear time. *Discrete Comput. Geom.*, 6:485–524, 1991.

[52] B. Chazelle and H. Edelsbrunner. An optimal algorithm for intersecting line segments in the plane. *J. ACM*, 39:1–54, 1992.

[53] B. Chazelle and J. Friedman. A deterministic view of random sampling and its use in geometry. *Combinatorica*, 10:229–249, 1990.

[54] B. Chazelle and J. Matoušek. Derandomizing an output-sensitive convex hull algorithm in three dimensions. Technical report, Dept. Comput. Sci., Princeton Univ., 1992.

[55] J. Chen and Y. Han. Shortest paths on a polyhedron. In *Proc. 6th Annu. ACM Sympos. Comput. Geom.*, pages 360–369, 1990.

[56] D. Cheriton and R. E. Tarjan. Finding minimum spanning trees. *SIAM J. of Computing*, 5:724–742, 1976.

[57] L. P. Chew. Planning the shortest path for a disc in $O(n^2 \log n)$ time. In *Proc. 1st Annu. ACM Sympos. Comput. Geom.*, pages 214–220, 1985.

[58] L. P. Chew and K. Kedem. Improvements on approximate pattern matching problems. To appear in *Proc. 3rd Scand. Workshop Algorithm Theory*, volume ?? of *Lecture Notes in Computer Science*, pages ??–?? Springer-Verlag, 1992.

[59] W. Chin and S. Ntafos. Optimum watchman routes. *Inform. Process. Lett.*, 28:39–44, 1988.

[60] W.-P. Chin and S. Ntafos. Watchman routes in simple polygons. *Discrete Comput. Geom.*, 6(1):9–31, 1991.

[61] N. Christofides. Worst-case analysis of a new heuristic for the traveling salesman problem. In J. F. Traub, editor, *Sympos. on New Directions and Recent Results in Algorithms and Complexity*, New York, NY, 1976. Academic Press.

[62] K. L. Clarkson. Approximation algorithms for shortest path motion planning. In *Proc. 19th Annu. ACM Sympos. Theory Comput.*, pages 55–65, 1987.

[63] K. L. Clarkson. New applications of random sampling in computational geometry. *Discrete Comput. Geom.*, 2:195–222, 1987.

[64] K. L. Clarkson, S. Kapoor, and P. M. Vaidya. Rectilinear shortest paths through polygonal obstacles in $O(n(\log n)^2)$ time. In *Proc. 3rd Annu. ACM Sympos. Comput. Geom.*, pages 251–257, 1987.

[65] K. L. Clarkson and P. W. Shor. Applications of random sampling in computational geometry, II. *Discrete Comput. Geom.*, 4:387–421, 1989.

[66] R. Cole and A. Siegel. River routing every which way but loose. In *Proc. 25th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 65–73, 1984.

[67] G. E. Collins. Quantifier elimination for real closed fields by cylindric algebraic decomposition. In *Proc. Second GI Conference on Automata Theory and Formal Languages*, volume 33 of *Lecture Notes in Computer Science*, pages 134–183, Berlin, 1975. Springer-Verlag.

[68] G. Das. Approximation schemes in computational geometry. Ph.D. Thesis, University of Wisconsin, 1990.

[69] G. Das and D. Joseph. The complexity of minimum convex nested polyhedra. In *Proc. 2nd Canad. Conf. Comput. Geom.*, pages 296–301, 1990.

[70] G. Das and D. Joseph. Minimum vertex hulls for polyhedral domains. In *Proc. 7th Sympos. Theoret. Aspects Comput. Sci.*, Lecture Notes in Computer Science. Springer-Verlag, 1990.

[71] A. Datta and K. Krithivasan. Path planning with local information. In *Proc. Foundations of Software Technology and Theoretical Computer Science*, New Delhi, India, December 1988.

[72] M. de Berg. On rectilinear link distance. *Comput. Geom. Theory Appl.*, 1:13–34, 1991.

[73] M. de Berg, M. van Kreveld, and B. J. Nilsson. Shortest path queries in rectilinear worlds of higher dimension. In *Proc. 7th Annu. ACM Sympos. Comput. Geom.*, pages 51–60, 1991.

[74] M. de Berg, M. van Kreveld, B. J. Nilsson, and M. H. Overmars. Finding shortest paths in the presence of orthogonal obstacles using a combined L^1 and link metric. Technical report, 1990.

[75] X. Deng, T. Kameda, and C. Papadimitriou. How to learn an unknown environment. In *Proc. 32nd Annu. IEEE Sympos. Found. Comput. Sci.*, pages 298–303, 1991.

[76] L. Devroye and G. T. Toussaint. A note on linear expected time algorithm for finding convex hulls. *Computing*, Vol. 26, pp. 361–366, 1981.

[77] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[78] M. B. Dillencourt. A non-Hamiltonian, nondegenerate Delaunay triangulation. *Information Processing Letters*, 25:149–151, 1987.

[79] H. N. Djidjev, A. Lingas, and J. Sack. An $O(n \log n)$ algorithm for computing the link center of a simple polygon. *Discrete Comput. Geom.*, To appear, 1992.

[80] D. H. Douglas and T.K. Peuker. Algorithms for the reduction of the number of points required to represent a line or its caricature. *The Canadian Cartographer*, 10(2):112–122, 1973.

[81] J. R. Driscoll, H. N. Gabow, R. Shrairaman, and R. E. Tarjan. Relaxed heaps: An alternative to Fibonacci heaps with applications to parallel computation. *Commun. ACM*, 31:1343–1354, 1988.

[82] R. A. Dwyer. A faster divide-and-conquer algorithm for constructing Delaunay triangulations. *Algorithmica*, 2:137–151, 1987.

[83] R. A. Dwyer. Average-case analysis of algorithms for convex hulls and Voronoi diagrams. Ph.D. Thesis, Carnegie-Mellon University, 1988.

[84] M. E. Dyer and A. M. Frieze. A partitioning algorithm for minimum weighted Euclidean matching. *Inform. Process. Lett.*, 18:59–62, 1984.

[85] P. Eades, X. Lin, and N. C. Wormald. Performance guarantees for motion planning with temporal uncertainty. Technical report, Dept. of Computer Science, Univ. of Queensland, St. Lucia, Queensland, 1989.

[86] H. Edelsbrunner. *Algorithms in Combinatorial Geometry*. Springer-Verlag, Heidelberg, Germany, 1987.

[87] H. Edelsbrunner, L. Guibas, and M. Sharir. The complexity of many cells in arrangements of planes and related problems. *Discrete Comput. Geom.*, 5:197–216, 1990.

[88] H. Edelsbrunner and L. J. Guibas. Topologically sweeping an arrangement. *J. Comput. Syst. Sci*, 38:165–194, 1989. Corrigendum in 42(1991)249–251.

[89] H. Edelsbrunner, L. J. Guibas, and M. Sharir. The complexity and construction of many faces in arrangements of lines and of segments. *Discrete Comput. Geom.*, 5:161–196, 1990.

[90] H. Edelsbrunner, L. J. Guibas, and J. Stolfi. Optimal point location in a monotone subdivision. *SIAM J. Comput.*, 15:317–340, 1986.

[91] H. Edelsbrunner, J. O'Rourke, and R. Seidel. Constructing arrangements of lines and hy-

perplanes with applications. *SIAM J. Comput.*, 15:341–363, 1986.
[92] H. Edelsbrunner and R. Seidel. Voronoi diagrams and arrangements. *Discrete Comput. Geom.*, 1:25–44, 1986.
[93] H. Edelsbrunner and E. Welzl. On the maximal number of edges of many faces in an arrangement. *J. Combin. Theory Ser. A*, 41:159–166, 1986.
[94] J. Edmonds. Maximum matching and a polyhedron with 0,1 vertices. *J. of Research NBS*, 69B:125–130, 1965.
[95] H. ElGindy and M. T. Goodrich. Parallel algorithms for shortest path problems in polygons. *Visual Comput.*, 3:371–378, 1988.
[96] S. Fortune and G. Wilfong. Planning constrained motion. In *Proc. 20th Annu. ACM Sympos. Theory Comput.*, pages 445–459, 1988.
[97] S. J. Fortune. A sweepline algorithm for Voronoi diagrams. *Algorithmica*, 2:153–174, 1987.
[98] M. Fredman and R. E. Tarjan. Fibonacci heaps and their uses in improved network optimization problems. *JACM*, 34:596–615, 1987.
[99] H. Gabow. Data structures for weighted matching and nearest common ancestors with linking. In *Proc. of First ACM-SIAM Symposium on Discrete Algorithms*, pages 434–443, 1990.
[100] H. Gabow, Z. Galil, T. Spencer, and R. E. Tarjan. Efficient algorithms for finding minimum spanning trees in undirected and directed graphs. *Combinatorica*, 6:109–122, 1986.
[101] K. R. Gabriel and R. R. Sokal. A new statistical approach to geographic variation analysis. *Systematic Zoology*, 18:259–278, 1969.
[102] S. Gao, M. Jerrum, M. Kaufmann, K. Mehlhorn, W. Rülling, and C. Storb. On continuous homotopic one layer routing. In *Computational Geometry and its Applications*, volume 333 of *Lecture Notes in Computer Science*, pages 55–70. Springer-Verlag, 1988.
[103] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, New York, NY, 1979.
[104] L. Gewali, A. Meng, J. S. B. Mitchell, and S. Ntafos. Path planning in $0/1/\infty$ weighted regions with applications. *ORSA J. Comput.*, 2(3):253–272, 1990.
[105] S. K. Ghosh and A. Maheshwari. An optimal algorithm for computing a minimum nested nonconvex polygon. Technical Report CS-90/2, Tata Inst., 1990.
[106] S. K. Ghosh and D. M. Mount. An output-sensitive algorithm for computing visibility graphs. *SIAM J. Comput.*, 20:888–910, 1991.
[107] M. Golin and R. Sedgewick. Analysis of a simple yet efficient convex hull algorithm. In *Proc. of 4th Annual Sympoisum on Computational Geometry*, pp. 153–163, 1988.
[108] M. Goodrich and R. Tamassia. Dynamic trees and dynamic point location. In *Proc. 23rd Annu. ACM Sympos. Theory Comput.*, pages 523–533, 1991.
[109] M. T. Goodrich, S. B. Shauck, and S. Guha. Parallel methods for visibility and shortest path problems in simple polygons. In *Proc. 6th Annu. ACM Sympos. Comput. Geom.*, pages 73–82, 1990.
[110] R. L. Graham. An efficient algorithm for determining the convex hull of a finite planar set. *Inform. Process. Lett.*, 1:132–133, 1972.
[111] R. L. Graham and P. Hell. On the history of minimum spanning tree problem. *Annals of History of Computing*, 7:43–57, 1985.
[112] B. Grünbaum. *Convex Polytopes*. Wiley, New York, NY, 1967.
[113] L. J. Guibas and J. Hershberger. Optimal shortest path queries in a simple polygon. *J. Comput. Syst. Sci.*, 39:126–152, 1989.
[114] L. J. Guibas, J. Hershberger, D. Leven, M. Sharir, and R. E. Tarjan. Linear-time algorithms for visibility and shortest path problems inside triangulated simple polygons. *Algorithmica*, 2:209–233, 1987.
[115] L. J. Guibas, J. E. Hershberger, J. S. B. Mitchell, and J. S. Snoeyink. Approximating polygons and subdivisions with minimum link paths. In *Proc. 2nd Annu. SIGAL Internat. Sympos. Algorithms*, volume 557 of *Lecture Notes in Computer Science*, pages 151–162. Springer-Verlag, 1991.
[116] L. J. Guibas, D. E. Knuth, and M. Sharir. Randomized incremental construction of Delaunay

and Voronoi diagrams. *Algorithmica*, 7:381–413, 1992.

[117] L. J. Guibas and J. Stolfi. Primitives for the manipulation of general subdivisions and the computation of Voronoi diagrams. *ACM Trans. Graph.*, 4:74–123, 1985.

[118] G. Y. Handler and I. Zang. A dual algorithm for the constrained shortest path problem. *Networks*, 10:293–310, 1980.

[119] Xin He. An efficient parallel algorithm for finding minimum weight matching for points on a convex polygon. *Inform. Process. Lett.*, 37(2):111–116, 1991.

[120] P. J. Heffernan and J. S. B. Mitchell. An optimal algorithm for computing visibility in the plane. In *Proc. 2nd Workshop Algorithms Data Struct.*, volume 519 of *Lecture Notes in Computer Science*, pages 437–448. Springer-Verlag, 1991.

[121] P. J. Heffernan and S. Schirra. Approximate decision algorithms for point set congruence. In *Proc. 8th Annu. ACM Sympos. Comput. Geom.*, pages 93–101, 1992.

[122] M.I. Henig. The shortest path problem with two objective functions. *European J. of Operational Research*, 25:281–291, 1985.

[123] J. Hershberger. Minimizing the sum of diameters efficiently. In *Proc. 3rd Canad. Conf. Comput. Geom.*, pages 62–65, 1991.

[124] J. Hershberger. Optimal parallel algorithms for triangulated simple polygons. In *Proc. 8th Annu. ACM Sympos. Comput. Geom.*, pages 33–42, 1992.

[125] J. Hershberger and L. J. Guibas. An $O(n^2)$ shortest path algorithm for a non-rotating convex body. *J. Algorithms*, 9:18–46, 1988.

[126] J. Hershberger and J. Snoeyink. Computing minimum length paths of a given homotopy class. In *Proc. 2nd Workshop Algorithms Data Struct.*, volume 519 of *Lecture Notes in Computer Science*, pages 331–342. Springer-Verlag, 1991.

[127] J. Hershberger and J. Snoeyink. An implementation of the Douglas-Peucker line simplification algorithm using at most $cn \log n$ operations. Technical report, 1991.

[128] J. Hershberger and S. Suri. Applications of a semi-dynamic convex hull algorithm. *BIT*, 32, pp. 249–267, 1992.

[129] J. Hershberger and S. Suri. Finding tailored partitions. *J. Algorithms*, 12:431–463, 1991.

[130] J. E. Hopcroft, D. A. Joseph, and S. H. Whitesides. Movement problems for 2-dimensional linkages. *SIAM J. Comput.*, 13:610–629, 1984.

[131] J. E. Hopcroft, J. T. Schwartz, and M. Sharir. *Planning, Geometry, and Complexity of Robot Motion*. Ablex Publishing, Norwood, NJ, 1987.

[132] D. P. Huttenlocher. Three-dimensional recognition of solid objects from a two-dimensional image. Ph.D. Thesis and Report TR-1045, Electrical Engineering and Computer Science, Massachusetts Institute of Technology, 1988.

[133] D. P. Huttenlocher and K. Kedem. Computing the minimum Hausdorff distance for point sets under translation. In *Proc. 6th Annu. ACM Sympos. Comput. Geom.*, pages 340–349, 1990.

[134] D. P. Huttenlocher, K. Kedem, and J. M. Kleinberg. On dynamic Voronoi diagrams and the minimum Hausdorff distance for point sets under Euclidean motion in the plane. Technical Report TR 92-1271, Dept. Comput. Sci., Cornell Univ., Ithaca, NY, March 1992.

[135] D. P. Huttenlocher, K. Kedem, and M. Sharir. The upper envelope of Voronoi surfaces and its applications. In *Proc. 7th Annu. ACM Sympos. Comput. Geom.*, pages 194–203, 1991.

[136] Y.-H. Hwang, R.-C. Chang, and H.-Y. Tu. Finding all shortest path edge sequences on a convex polyhedron. In *Proc. 1st Workshop Algorithms Data Struct.*, volume 382 of *Lecture Notes in Computer Science*, pages 251–266. Springer-Verlag, 1989.

[137] C. Icking, G. Rote, E. Welzl, and C. Yap. Shortest paths for line segments. Technical Report, Fachbereich Mathematik, Freie Universität, Berlin, 1989.

[138] H. Imai and M. Iri. Computational-geometric methods for polygonal approximations of a curve. *Computer Vision, Graphics, and Image Processing*, 36:31–41, 1986.

[139] H. Imai and M. Iri. An optimal algorithm for approximating a piecewise linear function. *Journal of Information Processing*, 9(3):159–162, 1986.

[140] H. Imai and M. Iri. Polygonal approximations of a curve-formulations and algorithms. In

G. T. Toussaint, editor, *Computational Morphology*, pages 71–86. North-Holland, Amsterdam, Netherlands, 1988.

[141] S. Sitharama Iyengar and Alberto Elfes, editors. *Autonomous Mobile Robots: Perception, Mapping, and Navigation*. IEEE Computer Society Press, Los Alamitos, CA, 1991.

[142] J. Jaromczyk and G. T. Toussaint. Relative neighborhood graphs and their relatives. *Proceedings of IEEE, Special Issue on Computational Geometry*, 1992.

[143] R. A. Jarvis. On the identification of the convex hull of a finite set of points in the plane. *Inform. Process. Lett.*, 2:18–21, 1973.

[144] M. Kallay. The complexity of incremental convex hull algorithms in R^d. *Inform. Process. Lett.*, 19:197, 1984.

[145] V. Kantabutra. Traveling salesman cycles are not always subgraphs of Voronoi duals. *Information Processing Letters*, 16:11–12, 1983.

[146] S. Kapoor and S. N. Maheshwari. Efficient algorithms for Euclidean shortest path and visibility problems with polygonal obstacles. In *Proc. 4th Annu. ACM Sympos. Comput. Geom.*, pages 172–182, 1988.

[147] Y. Ke. An efficient algorithm for link-distance problems. In *Proc. 5th Annu. ACM Sympos. Comput. Geom.*, pages 69–78, 1989.

[148] M. Kindl, M. Shing, and N. Rowe. A stochastic approach to the weighted-region problem: I. the design of the path annealing algorithm. Technical report, Computer Science, U.S. Naval Postgraduate School, Monterey, CA, 1991.

[149] M. Kindl, M. Shing, and N. Rowe. A stochastic approach to the weighted-region problem: Ii. performance enhancement techniques and experimental results. Technical report, Computer Science, U.S. Naval Postgraduate School, Monterey, CA, 1991.

[150] D. G. Kirkpatrick. Optimal search in planar subdivisions. *SIAM J. Comput.*, 12:28–35, 1983.

[151] D. G. Kirkpatrick and R. Seidel. The ultimate planar convex hull algorithm? *SIAM J. Comput.*, 15:287–299, 1986.

[152] R. Klein. Walking an unknown street with bounded detour. In *Proc. 32nd Annu. IEEE Sympos. Found. Comput. Sci.*, pages 304–313, 1991.

[153] E. Kranakis, D. Krizanc, and L. Meertens. Link length of rectilinear watchman tours in grids. In *Proc. 2nd Canad. Conf. Comput. Geom.*, pages 328–331, 1990.

[154] J. B. Kruskal. On the shortest spanning tree of a graph and the traveling salesman problem. *Proc. of American Math. Soc.*, 7:48–50, 1956.

[155] L. C. Larson. *Problem-Solving Through Problems*. Springer Verlag, New York, 1983.

[156] J. C. Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, 1991.

[157] J. Laumond. Feasible trajectories for mobile robots with kinematic and environment constraints. *Intelligent Autonomous Systems*, ??:346–354, 1986.

[158] E. L. Lawler. *Combinatorial Optimization: Networks and Matroids*. New York: Holt, Rinehart and Winston, 1976.

[159] D. T. Lee. Proximity and reachability in the plane. Report R-831, Dept. Elect. Engrg., Univ. Illinois, Urbana, IL, 1978.

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

[161] C. E. Leiserson and F. M. Maley. Algorithms for routing and testing routability of planar vlsi layouts. In *Proc. 17th Annu. ACM Sympos. Theory Comput.*, pages 69–78, 1985.

[162] W. Lenhart, R. Pollack, J.-R. Sack, R. Seidel, M. Sharir, S. Suri, G. T. Toussaint, S. Whitesides, and C. K. Yap. Computing the link center of a simple polygon. *Discrete Comput. Geom.*, 3:281–293, 1988.

[163] R. J. Lipton and R. E. Tarjan. Applications of a planar separator theorem. *SIAM J. Comput.*, 9:615–627, 1980.

[164] V. J. Lumelsky and A. A. Stepanov. Path-planning strategies for a point mobile automaton moving amidst unknown obstacles of arbitrary shape. *Algorithmica*, 2:403–430, 1987.

[165] O. Marcotte and S. Suri. Fast matching algorithms for points on a polygon. *SIAM J. Comput.*, 20:405–422, 1991.

[166] J. Matoušek, N. Miller, J. Pach, M. Sharir, S. Sifrony, and E. Welzl. Fat triangles determine linearly many holes. In *Proc. 32nd Annu. IEEE Sympos. Found. Comput. Sci.*, pages 49–58, 1991.

[167] D. W. Matula and R. R. Sokal. Properties of Gabriel graphs relevant to geographic variation research and clustering of points in the plane. *Geogr. Anal.*, 12:205–222, 1980.

[168] R.B. McMaster. Automated line generation. *Cartographica*, 24(2):74–111, 1987.

[169] A. Melkman and J. O'Rourke. On polygonal chain approximation. In G. T. Toussaint, editor, *Computational Morphology*, pages 87–95. North-Holland, Amsterdam, Netherlands, 1988.

[170] J. Mitchell and E. Welzl. Dynamically maintaining a visibility graph under insertions of new obstacles. Manuscript, School Oper. Res. Indust. Engrg., Cornell Univ., Ithaca, NY, 1990.

[171] J. S. B. Mitchell. Planning shortest paths. Ph.D. Thesis, Stanford Univ., Stanford, CA, 1986.

[172] J. S. B. Mitchell. Algorithmic approaches to optimal route planning. In *Proc. SPIE Conference on Mobile Robots*, Boston, MA, 4-9 November 1990.

[173] J. S. B. Mitchell. On maximum flows in polyhedral domains. *J. Comput. Syst. Sci.*, 40:88–123, 1990.

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

[175] J. S. B. Mitchell. L_1 shortest paths among polygonal obstacles in the plane. *Algorithmica*, 8:55–88, 1992.

[176] J. S. B. Mitchell, D. M. Mount, and C. H. Papadimitriou. The discrete geodesic problem. *SIAM J. Comput.*, 16:647–668, 1987.

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

[178] J. S. B. Mitchell, G. Rote, and G. Woeginger. Minimum-link paths among obstacles in the plane. *Algorithmica*, 8:431–459, 1992.

[179] J. S. B. Mitchell and S. Suri. Separation and approximation of polyhedral surfaces. In *Proc. 3rd ACM-SIAM Sympos. Discrete Algorithms*, pages 296–306, 1992.

[180] J. S. B. Mitchell and E. L. Wynters. Watchman routes for multiple guards. In *Proc. 3rd Canad. Conf. Comput. Geom.*, pages 126–129, 1991.

[181] J.S.B. Mitchell. An optimal algorithm for shortest rectilinear paths among obstacles. Manuscript, School Oper. Res. Indust. Engrg., Cornell Univ., Ithaca, NY, 1989.

[182] J.S.B. Mitchell and C. Piatko. Approximation methods for link distances in higher dimensions. Manuscript, School Oper. Res. Indust. Engrg., Cornell Univ., Ithaca, NY, 1992.

[183] J.S.B. Mitchell, C.D. Piatko, and E.M. Arkin. Computing a shortest k-link path in a simple polygon. In *Proc. 33rd Annu. IEEE Sympos. Found. Comput. Sci.*, pages 573–582, 1992.

[184] C. Monma, M. Paterson, S. Suri, and F. Yao. Computing Euclidean maximum spanning trees. *Algorithmica*, 5:407–419, 1990.

[185] C. Monma and S. Suri. Partitioning points and graphs to minimize the maximum or the sum of diameters. In *Graph Theory, Combinatorics and Applications (Proc. 6th Internat. Conf. Theory Appl. Graphs)*, volume 2, pages 899–912, New York, NY, 1991. Wiley.

[186] D. Mount. On finding shortest paths on convex polyhedra. Technical Report 1495, Department of Computer Science, University of Maryland, 1985.

[187] D. M. Mount. The number of shortest paths on the surface of a polyhedron. *SIAM J. Comput.*, 19:593–611, 1990.

[188] B. K. Natarajan. On comparing and compressing piece-wise linear curves. Technical report, Hewlett Packard, 1991.

[189] W. P. Niedringhaus. Scheduling with queueing: the space factory problem. Technical report, Princeton University, 1979.

[190] N. Nilsson. A mobile automaton: An application of artificial intelligence techniques. In *Proc. IJCAI*, pages 509–520, 1969.

[191] S. Ntafos. Watchman routes under limited visibility. In *Proc. 2nd Canad. Conf. Comput. Geom.*, pages 89–92, 1990.

[192] C. Ó'Dúnlaing, M. Sharir, and C. K. Yap. Generalized Voronoi diagrams for moving a ladder:

I. topological analysis. *Commun. Pure Appl. Math.*, 39:423–483, 1986.

[193] C. Ó'Dúnlaing, M. Sharir, and C. K. Yap. Generalized Voronoi diagrams for moving a ladder: II. efficient construction of the diagram. *Algorithmica*, 2:27–59, 1987.

[194] C. Ó'Dúnlaing and C. K. Yap. A "retraction" method for planning the motion of a disk. *J. Algorithms*, 6:104–111, 1985.

[195] C. Ó'Dúnlaing and M. Sharir C. K. Yap. Retraction: a new approach to motion-planning. In *Proc. 15th Annu. ACM Sympos. Theory Comput.*, pages 207–220, 1983.

[196] J. O'Rourke. Finding a shortest ladder path: a special case. IMA Preprint Series 353, Inst. Math. Appl., Univ. Minnesota, Minneapolis, MN, 1987.

[197] J. O'Rourke, H. Booth and R. Washington. Connect-the-dots: a new heuristic. *Computer Vision, Graphics, and Image Processing*, Vol. 39, pp. 258–266, 1987.

[198] J. O'Rourke and C. Schevon. Computing the geodesic diameter of a 3-polytope. In *Proc. 5th Annu. ACM Sympos. Comput. Geom.*, pages 370–379, 1989.

[199] J. O'Rourke, S. Suri, and H. Booth. Shortest paths on polyhedral surfaces. In *Proc. 2nd Sympos. Theoret. Aspects Comput. Sci.*, volume 182 of *Lecture Notes in Computer Science*, pages 243–254. Springer-Verlag, 1985.

[200] M. H. Overmars and E. Welzl. New methods for computing visibility graphs. In *Proc. 4th Annu. ACM Sympos. Comput. Geom.*, pages 164–171, 1988.

[201] N. Papadakis and A. Perakis. Minimal time vessel routing in a time-dependent environment. *Transportation Science*, 23(4):266–276, 1989.

[202] N. Papadakis and A. Perakis. Deterministic minimal time vessel routing. *Operations Research*, 38(3):426–438, 1990.

[203] C. H. Papadimitriou. The Euclidean traveling salesman problem is *np*-complete. *J. of Theoretical Computer Science*, pages 237–244, 1977.

[204] C. H. Papadimitriou. An algorithm for shortest-path motion in three dimensions. *Inform. Process. Lett.*, 20:259–263, 1985.

[205] C. H. Papadimitriou and E. B. Silverberg. Optimal piecewise linear motion of an object among obstacles. *Algorithmica*, 2:523–539, 1987.

[206] C.H. Papadimitriou and M. Yannakakis. Shortest paths without a map. In *Proc. ICALP*, 1989.

[207] J. Pearl. *Heuristics: Intelligent Search Strategies for Computer Problem Solving.* Addison-Wesley, Reading, MA, 1984.

[208] M. Pellegrini. On the zone of a co-dimension p surface in a hyperplane arrangement. In *Proc. 3rd Canad. Conf. Comput. Geom.*, pages 233–238, 1991.

[209] R. Pollack, M. Sharir, and G. Rote. Computing of the geodesic center of a simple polygon. *Discrete Comput. Geom.*, 4:611–626, 1989.

[210] F. P. Preparata and S. J. Hong. Convex hulls of finite sets of points in two and three dimensions. *Commun. ACM*, 20:87–93, 1977.

[211] F. P. Preparata and M. I. Shamos. *Computational Geometry: an Introduction.* Springer-Verlag, New York, NY, 1985.

[212] R. C. Prim. Shortest connection networks and some generalizations. *Bell Systems Technical Journal*, 36:1389–1401, 1957.

[213] J. H. Reif. Complexity of the generalized movers problem. In J. Hopcroft, J. Schwartz, and M. Sharir, editors, *Planning, Geometry and Complexity of Robot Motion*, pages 267–281. Ablex Pub. Corp., Norwood, NJ, 1987.

[214] J. H. Reif and J. A. Storer. Shortest paths in Euclidean spaces with polyhedral obstacles. Report CS-85-121, Dept. Comput. Sci., Brandeis Univ., Waltham, MA, 1985.

[215] J. H. Reif and J. A. Storer. Minimizing turns for discrete movement in the interior of a polygon. *IEEE J. on Robotics and Automation*, pages 182–193, 1987.

[216] H. Rohnert. A new algorithm for shortest paths avoiding convex polygonal obstacles. Report A86/02, Fachber. Inform., Univ. Saarlandes, Saarbrücken, Germany, 1986.

[217] H. Rohnert. Shortest paths in the plane with convex polygonal obstacles. *Inform. Process. Lett.*, 23:71–76, 1986.

[218] G. Rote. Computing the Hausdorff distance between point sets on a line. *IPL*, 1992.
[219] G. Rote. A new metric between polygons, and how to compute it. In *Proc. ICALP*, 1992.
[220] J. T. Schwartz and M. Sharir. On the "piano movers" problem I: the case of a two-dimensional rigid polygonal body moving amidst polygonal barriers. *Commun. Pure Appl. Math.*, 36:345–398, 1983.
[221] J. T. Schwartz and M. Sharir. On the "piano movers" problem II: general techniques for computing topological properties of real algebraic manifolds. *Adv. Appl. Math.*, 4:298–351, 1983.
[222] J. T. Schwartz and M. Sharir. On the "piano movers" problem III: coordinating the motion of several independent bodies: the special case of circular bodies moving amidst polygonal barriers. *Internat. J. Rob. Res.*, 2(3):46–75, 1983.
[223] J. T. Schwartz and M. Sharir. On the "piano movers" problem V: the case of a rod moving in three-dimensional space amidst polyhedral obstacles. *Commun. Pure Appl. Math.*, 37:815–848, 1984.
[224] J. T. Schwartz and M. Sharir. Algorithmic motion planning in robotics. In J. van Leeuwen, editor, *Algorithms and Complexity*, volume A of *Handbook of Theoretical Computer Science*, pages 391–430. Elsevier, Amsterdam, Netherlands, 1990.
[225] R. Seidel. A convex hull algorithm optimal for point sets in even dimensions. Report 81/14, Dept. Comput. Sci., Univ. British Columbia, Vancouver, BC, 1981.
[226] R. Seidel. Constructing higher-dimensional convex hulls at logarithmic cost by face. In *Proc. 18th Annu. ACM Sympos. Theory Comput.*, pages 404–413, 1986.
[227] M. I. Shamos. Computational geometry. Ph.D. Thesis, Dept. of Computer Science, Yale University, 1978.
[228] M. Sharir. On shortest paths amidst convex polyhedra. *SIAM J. Comput.*, 16:561–572, 1987.
[229] M. Sharir and E. Ariel-Sheffi. On the "piano movers" problem IV: various decomposable two-dimensional motion planning problems. *Commun. Pure Appl. Math.*, 37:479–493, 1984.
[230] M. Sharir and A. Schorr. On shortest paths in polyhedral spaces. *SIAM J. Comput.*, 15:193–215, 1986.
[231] T. Smith, G. Peng, and P. Gahinet. A family of local, asynchronous, iterative, and parallel procedures for solving the weighted region least cost path problem. Technical report, Department of Computer Science, University of California, Santa Barbara, 1988.
[232] K. J. Supowit. The relative neighborhood graph with an application to minimum spanning trees. *J. ACM*, 30:428–448, 1983.
[233] S. Suri. A linear time algorithm for minimum link paths inside a simple polygon. *Comput. Vision Graph. Image Process.*, 35:99–110, 1986.
[234] S. Suri. Minimum link paths in polygons and related problems. Ph.D. Thesis, Dept. Comput. Sci., Johns Hopkins Univ., Baltimore, MD, 1987.
[235] S. Suri. Computing geodesic furthest neighbors in simple polygons. *J. Comput. Syst. Sci.*, 39:220–235, 1989.
[236] S. Suri. On some link distance problems in a simple polygon. *IEEE Transactions on Robotics and Automation*, 6:108–113, 1990.
[237] G. F. Swart. Finding the convex hull facet by facet. *J. Algorithms*, 6:17–48, 1985.
[238] R. Tamassia and F. P. Preparata. Dynamic maintenance of planar digraphs, with applications. *Algorithmica*, 5:509–527, 1990.
[239] X. H. Tan, T. Hirata, and Y. Inagaki. An incremental algorithm for constructing shortest watchman routes. In *Proc. 2nd Annu. SIGAL Internat. Sympos. Algorithms*, volume 557 of *Lecture Notes in Computer Science*, pages 163–175. Springer-Verlag, 1991.
[240] R. E. Tarjan and C. J. Van Wyk. An $O(n \log\log n)$-time algorithm for triangulating a simple polygon. *SIAM J. Comput.*, 17:143–178, 1988. Erratum in 17(1988)106.
[241] A. Tarski. *A decision method for elementary algebra and geometry.* Univ. of California Press, Berkeley, CA, 1951.
[242] G. T. Toussaint. Pattern recognition and geometrical complexity. In *Proc. 5th Internat. Conf. Pattern Recogn.*, pages 1324–1347, 1980.

[243] G. T. Toussaint. The relative neighborhood graph of a finite planar set. *Pattern Recognition*, Vol. 12, pp. 261–268, 1980.

[244] G. Toussaint. Computing geodesic properties inside a simple polygon. Technical report, School of Computer Science, McGill University, 1990.

[245] P. M. Vaidya. Minimum spanning trees in k-dimensional space. *SIAM J. Comput.*, 17:572–582, 1988.

[246] P. M. Vaidya. Approximate minimum weight matching on points in k-dimensional space. *Algorithmica*, 4:569–583, 1989.

[247] P. M. Vaidya. Geometry helps in matching. *SIAM J. Comput.*, (18):1201–1225, 1989.

[248] P. M. Vaidya. An $O(n \log n)$ algorithm for the all-nearest-neighbors problem. *Discrete Comput. Geom.*, 4:101–115, 1989.

[249] G. Vegter. The visibility diagram: A data structure for visibility problems and motion planning. In *Proc. 2nd Scand. Workshop Algorithm Theory*, volume 447 of *Lecture Notes in Computer Science*, pages 97–110. Springer-Verlag, 1990.

[250] G. Vegter. Dynamically maintaining the visibility graph. In *Proc. 2nd Workshop Algorithms Data Struct.*, volume 519 of *Lecture Notes in Computer Science*, pages 425–436. Springer-Verlag, 1991.

[251] C. A. Wang and E. P. F. Chan. Finding the minimum visible vertex distance between two nonintersecting simple polygons. In *Proc. 2nd Annu. ACM Sympos. Comput. Geom.*, pages 34–42, 1986.

[252] E. Welzl. Constructing the visibility graph for n line segments in $O(n^2)$ time. *Inform. Process. Lett.*, 20:167–171, 1985.

[253] P. Widmayer. Network design issues in vlsi. Technical report, Institut für Informatik, University Freiburg, Rheinstrasse 10-12, 7800, Freiburg, West Germany, 1989.

[254] P. Widmayer, Y. F. Wu, and C. K. Wong. On some distance problems in fixed orientations. *SIAM J. Comput.*, 16:728–746, 1987.

[255] G. Wilfong. Motion planning for an autonomous vehicle. In *IEEE Int. Conf. Robotics and Automation*, pages 529–533, 1988.

[256] G. Wilfong. Shortest paths for autonomous vehicles. Technical Report, AT& T Bell Labs, 1988.

[257] C. Yang, D. Lee, and C. Wong. Rectilinear paths among rectilinear obstacles revisited. Technical report, Dept. of EE & CS, Northwestern Univ., 1992.

[258] C. D. Yang, D. T. Lee, and C. K. Wong. On bends and lengths of rectilinear paths: a graph-theoretic approach. In *Proc. 2nd Workshop Algorithms Data Struct.*, volume 519 of *Lecture Notes in Computer Science*, pages 320–330. Springer-Verlag, 1991.

[259] A. Yao. On constructing minimum spanning trees in k-dimensional spaces and related problems. *SIAM J. Computing*, 11:721–736, 1982.

[260] A. C. Yao. A lower bound to finding convex hulls. *J. ACM*, 28:780–787, 1981.

[261] C. K. Yap. Algorithmic motion planning. In *Advances in Robotics 1: Algorithmic and Geometric Aspects of Robotics*, pages 95–143, J. T. Schwartz and C.-K. Yap, editors. Lawrence Erlbaum Associates, Hillsdale, NJ, 1987.

[262] C. K. Yap. An $O(n \log n)$ algorithm for the Voronoi diagram of a set of simple curve segments. *Discrete Comput. Geom.*, 2:365–393, 1987.

[263] K. Zikan. Least-squares image registration. *ORSA Journal on Computing*, 3:169–172, 1991.