Fajie Li and Reinhard Klette

Euclidean Shortest Paths

Exact or Approximate Algorithms

Springer

Contents

Preface

A Euclidean shortest path connects a source with a destination, avoids some places (called *obstacles*), visits some places (called *attractions*), possibly in a defined order, and is of minimum length. Euclidean shortest-path problems are defined in the Euclidean plane or in Euclidean 3-dimensional space. The calculation of a convex hull in the plane is an example for finding a shortest path (around the given set of planar obstacles). Polyhedral obstacles and polyhedral attractions, a start and an endpoint define a general Euclidean shortest-path problem in 3-dimensional space.

The book presents selected algorithms (i.e., not aiming at a general overview) for the exact or approximate solution of shortest-path problems. Subjects in the first chapters of the book also include fundamental algorithms. Graph theory offers shortest-path algorithms for discrete problems. Convex hulls (and to a less extent also constrained convex hulls) have been discussed in computational geometry. Seidel's triangulation and Chazelle's triangulation method for a simple polygon, and Mitchell's solution of the continuous Dijkstra problem have also been selected for a detailed presentation, just to name three examples of important work in the area.

The book also covers a class of algorithms (called *rubberband algorithms*), which originated from a proposal for calculating minimum-length polygonal curves in cube-curves; Thomas Bülow was a co-author of the initiating publication, and he coined the name 'rubberband algorithm' in 2000 for the first time for this approach.

Subsequent work between 2000 and now shows that the basic ideas of this algorithm generalized for solving a range of problems. In a sequence of publications between 2003 and 2010 we, the authors of this book, describe a class of rubberband algorithms with proofs of their correctness and time-efficiency. Those algorithms can be used to solve different Euclidean shortest-path (ESP) problems, such as calculating the ESP inside of a simple cube-arc (the initial problem), inside of a simple polygon, on the surface of a convex polytope, or inside of a simple polyhedron, but also ESP problems such as touring a finite sequence of polygons, cutting parts, or the safari, zookeeper, or watchman route problems.

We aimed at writing a book that might be useful for a second or third-year algorithms course at university level. It should also contain sufficient details for students and researchers in the field who are keen to understand the correctness proofs, the

analysis of time complexities and related topics and not just the algorithms and their pseudocodes. The book discusses selected subjects and algorithms at some depth, including mathematical proofs for most of the given statements. (This is different from books which aim at a representative coverage of areas in algorithm design.)

Each chapter closes with theoretical or programming exercises, giving students various opportunities to learn the subject by solving problems or doing their own experiments. Tasks are (intentionally) only sketched in the given programming exercises, not described exactly in all their details (say, as it is typically when a costumer specifies a problem to an IT consultant), and identical solutions to such vaguely described projects do not exist, leaving space for the creativity of the student.

The audience for the book could be students in computer science, IT, mathematics, or engineering at a university, or academics being involved in research or teaching of efficient algorithms. The book could also be useful for programmers, mathematicians, or engineers which have to deal with shortest-path problems in practical applications, such as in robotics (e.g., when programming an industrial robot), in routing (i.e., when selecting a path in a network), in gene technology (e.g., when studying structures of genes), or in game programming (e.g., when optimizing paths for moves of players) - just to cite four of such application areas.

The authors thank (in alphabetical order) Tetsuo Asano, Donald Bailey, Chanderjit Bajaj, Partha Bhowmick, Alfred (Freddy) Bruckstein, Thomas Bülow, Xia Chen, Yewang Chen, David Coeurjolly, Eduardo Destefanis, Michael J. Dinneen, David Eppstein, Claudia Esteves Jaramillo, David Gauld, Jean-Bernard Hayet, David Kirkpatrick, Wladimir Kovalevski, Norbert Krüger, Jacques-Olivier Lachaud, Joe Mitchell, Akira Nakamura, Xiuxia Pan, Henrik G. Petersen, Nicolai Petkov, Fridrich Sloboda, Gerald Sommer, Mutsuhiro Terauchi, Ivan Reilly, the late Azriel Rosenfeld, the late Klaus Voss, Jinlong Wang, and Joviša Žunić for discussions or comments that were of relevance for this book.

The authors thank Chengle Huang (ChingLok Wong) for discussions on rubberband algorithms; he also wrote C++ programs for testing Algorithms 7 and 8. We thank Jinling Zhang and Xinbo Fu for improving C++ programs for testing Algorithm 7. The authors acknowledge computer support by Wei Chen, Wenze Chen, Yongqian Du, Wenxian Jiang, Yanmin Luo, Shujuan Peng, Huijuan Pi, Huazhen Wang, and Jian Yu.

The first author thanks dean Weibin Chen at Huaqiao University for supporting the project of writing this book. The second author thanks José L. Marroquín at CIMAT Guanajuato for an invitation to this institute, thus providing excellent conditions for working on this book project.

Parts of Chapter 4 (on relative convex hulls) are co-authored by Gisela Klette, who also contributed comments, ideas and criticisms throughout the book project.

We are grateful to Garry Tee for corrections and valuable comments, often adding important mathematical or historic details.

Huaqiao and Auckland, *Fajie Li*
July 2011 *Reinhard Klette*

Part I
Discrete or Continuous Shortest Paths

The road system of the historic city of Guanajuato is mostly underground, forming a network of tunnels. Optimizing routes from one part of the city to another part is an exciting task not only for visitors of the city, but even for locals. In 2010, there was not yet a "3D route planner" available for calculating shortest connections in this historic city.

The first part of the book defines shortest paths in the geometry that we practice in our daily life, known as *Euclidean geometry*. Finding a shortest path between two given points and avoiding existing obstacles can be a challenge. We provide basic definitions and we propose the class of *rubberband algorithms* for solving various Euclidean shortest-path problems, either defined in the plane or in the 3-dimensional space.

Chapter 1
Euclidean Shortest Paths

Ptolemy once asked Euclid whether there was any shorter way to a knowledge of geometry than by a study of the Elements, whereupon Euclid answered that there was no royal road to geometry.

Proclus Diadochus (410...412 – 485)

This introductory chapter explains the difference between shortest paths in finite graphs and shortest paths in Euclidean geometry, which is also called 'the common geometry of our world'. The chapter demonstrates the diversity of such problems, defined between points in a plane, on a surface, or in the 3-dimensional space.

1.1 Arithmetic Algorithms

Technology is evaluated by applying various *measures*, which are mapping components of the technology into real numbers, such as kilometres per hour, a maximum error in millimetres, or a shape descriptor. We describe algorithms with respect to run time, deviations of results from being optimal, or necessary constraints on input data.

The shortest path algorithms in this book are designed to be fast, accurate, and for a wide range of input data.

The time for calculating a shortest path on a computer depends on parameters such as available memory space or execution time per applied operation. It is meaningless to express the required time as an absolute value in, say, micro- or nanoseconds, because computers differ in their parameters, they change frequently, and your computer is certainly not identical to one used by someone else.

Measures for computing time need to be independent from the configuration of computers for expressing the quality of an algorithm. For example, we apply an abstract measure to estimate the running time of algorithms, also called *time complexity* or *computational complexity*. This is a common approach in computer science for more than 50 years. We define a set of *basic operations*, thus specifying a particular *abstract computer*, and we assign *one time unit* to each basic operation. Such a 'unit of time' is not measured in a physical scale; it is an abstract unit for each

basic operation. For those abstract units, the estimation of run time of an algorithm is independent of progress in computer technology.

Definition 1.1. An *algorithm* is a finite sequence of basic operations.

We consider basic operations that are executable on 'a normal sequential computer', and not, for example, on some kind of specialized processor. Basic operations are classified into numerical operations, logical tests, and control instructions; alphanumerical operations on letters or other symbols are not a subject in this book. Logical tests are comparisons in magnitude (such as "$length < threshold$"?). Control instructions are of the type if-then-else, while-do, or do-until. It only remains to define a particular computer model by specifying the set of numerical operations.

Definition 1.2. An *arithmetic algorithm* is defined by having addition, subtraction, multiplication, division, and the n-th root (for any integer $n > 1$) as its basic operations.

A *scientific algorithm* expands an arithmetic algorithm by the following additional basic operations: trigonometric and inverse trigonometric functions, exponential and logarithmic functions, factorials, and conversions of numbers between different representation schemes (binary, decimal, or hexadecimal).

This book applies arithmetic algorithms for solving shortest-path problems.

(There is just one case in the book where trigonometric functions are used when optimizing a point location in an ellipse.)

Arithmetic algorithms are further specified by the range of numbers they are working on. For example, assuming the unbounded range of *rational numbers* a/b (for integers a and b) goes beyond the capabilities of existing computers, which can only represent rational numbers up to some finite number of bits per number. Even if divisions a/b are not performed, and assuming that there are no roots (i.e., all calculations remain in the area of integers) then there is still the problem of a limited range, having only a finite number of bits for representing all available integers.

Definition 1.3. An *arithmetic algorithm over the rational numbers* uses the basic operations as available in an arithmetic algorithm and operates on the set of all rational numbers.

1.2 Upper Time Bounds

We assume, as is common in algorithmic complexity theory, that the performance of each basic operation requires one time unit.