Computational Geometry 46 (2013) 959–969

Contents lists available at SciVerse ScienceDirect

Computational Geometry: Theory and Applications

www.elsevier.com/locate/comgeo

Memory-constrained algorithms for simple polygons ☆

Tetsuo Asano [a], Kevin Buchin [b,1], Maike Buchin [b], Matias Korman [c,*,2], Wolfgang Mulzer [d], Günter Rote [d], André Schulz [e]

[a] JAIST, Japan
[b] TU Eindhoven, The Netherlands
[c] UPC, Barcelona, Spain
[d] Freie Universität Berlin, Germany
[e] WWU Münster, Germany

ARTICLE INFO

Article history:
Received 15 June 2012
Accepted 30 April 2013
Available online 9 May 2013
Communicated by Beppe Liotta

Keywords:
Space–time trade-off
Constant workspace
Triangulation
Shortest path
Simple polygon

ABSTRACT

A constant-work-space algorithm has read-only access to an input array and may use only $O(1)$ additional words of $O(\log n)$ bits, where n is the input size. We show how to triangulate a plane straight-line graph with n vertices in $O(n^2)$ time and constant work-space. We also consider the problem of preprocessing a simple polygon P for shortest path queries, where P is given by the ordered sequence of its n vertices. For this, we relax the space constraint to allow s words of work-space. After quadratic preprocessing, the shortest path between any two points inside P can be found in $O(n^2/s)$ time.

© 2013 Elsevier B.V. All rights reserved.

1. Introduction

In algorithm development and computer technology, we observe two opposing trends: on the one hand, there are vast amounts of computational resources at our fingertips. Alas, this often leads to bloated software that is written without regard to resources and efficiency. On the other hand, we see a proliferation of specialized tiny devices that have a limited supply of memory or power. Software that is oblivious to space resources is not suitable for such a setting. Moreover, even if a small device features a fairly large memory, it may still be preferable to limit the number of write operations. For one, writing to flash memory is slow and costly, and it also reduces the lifetime of the memory. Furthermore, if the input is stored on a removable medium, write-access may be limited for technical or security reasons.

With this situation in mind, it makes sense to focus on algorithms that need only a limited amount of work-space, while the input resides in read-only memory. In this paper, we will develop such algorithms for geometric problems in planar polygons or, more generally, plane straight-line graphs (PSLGs).

☆ This work was initiated at the Dagstuhl Workshop on Memory-Constrained Algorithms and Applications, November 21–23, 2011. We are deeply grateful to the organizers as well as the participants of the workshop for helpful discussions during the meeting. A preliminary version was presented as T. Asano, K. Buchin, M. Buchin, M. Korman, W. Mulzer, G. Rote, A. Schulz, Memory-constrained algorithms for simple polygons, in: Abstracts of the 28th EWCG, 2012, pp. 49–52.

* Corresponding author.
E-mail addresses: t-asano@jaist.ac.jp (T. Asano), k.a.buchin@tue.nl (K. Buchin), m.e.buchin@tue.nl (M. Buchin), matias.korman@upc.edu (M. Korman), mulzer@inf.fu-berlin.de (W. Mulzer), rote@inf.fu-berlin.de (G. Rote), andre.schulz@uni-muenster.de (A. Schulz).

[1] M. Buchin is supported by the Netherlands Organisation for Scientific Research (NWO) under project no. 612.001.106.
[2] With the support of the Secretary for Universities and Research of the Ministry of Economy and Knowledge of the Government of Catalonia and the European Union.

0925-7721/$ – see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.comgeo.2013.04.005

In particular, we consider two fundamental problems from computational geometry [6]: first, we are given a PSLG $G = (V, E)$ with n vertices, and we would like to find a *triangulation* for G, i.e., a PSLG with vertex set V that contains all the edges in E and to which no edge can be added without violating planarity. We show how to find such a triangulation in $O(n^2)$ time with $O(1)$ words of work-space (Section 4). Since our model does not allow storing the output, our algorithm outputs the triangles of the triangulation one after another.

Then, we apply this result in order to construct a *memory-adjustable* data structure for shortest path queries in simple polygons (Section 5). Given a simple polygon P with n vertices and a parameter $s \in \{1, \ldots, n\}$, we build a data structure for P that requires $O(s)$ words of storage and that lets us output the edges of a shortest path between any two points inside P in $O(n^2/s)$ time using $O(s)$ work-space. The preprocessing time is $O(n^2)$.

Model assumptions. The input to our algorithms is either a simple polygon P or a PSLG G with n vertices, stored in a read-only data structure.[3] In case of a PSLG, we assume that G is given in a way that allows us to enumerate all edges of G in $O(n)$ time and to find the incident vertices of a given edge in constant time (a standard adjacency list representation will do). In case of a polygon, we require that the vertices of P are stored according to their counterclockwise order along the boundary, so that we can obtain the (clockwise and counterclockwise) neighbor of any given vertex in constant time. We also assume that is takes constant time to access the x- and y-coordinates of any vertex and to perform basic geometric operations, such as determining whether a point lies above or below a given line.

Storage is counted in terms of *cells* or *words*. As usual, a word is assumed to be large enough to contain either an input item (such as a point coordinate) or a pointer into the input structure (of $\Theta(\log n)$ bits). Thus, in order to convert our storage bounds into bits, we have to multiply them by a factor of $\log n$. In addition to the input, which can only be read, the algorithm has $O(s)$ words of *work-space* at its disposal for reading and writing. Here, s is a parameter of the model and can range between 1 and n. We will consider both the case where s is a fixed constant and the case where s can be chosen by the user. Since there is no way to store the result, we use an additional operation `output` in order to generate output data. We require that every feature of the desired structure is `output` exactly once.

For simplicity, we will make the usual general position assumption: no three input vertices are on a line and no two input vertices have the same x-coordinate.

Related work. Given the many applications of memory-constrained algorithms, a significant amount of research has been devoted to them, even as early as in the 1980s [23]. One of the most studied problems in this setting is that of selection in an unsorted array with elements from a totally ordered universe [11,18,23,24,26].

In computational complexity theory, the constant-work-space model is represented by the complexity class LOGSPACE [1]. There are several algorithmic results for this class, most prominently Reingold's celebrated method for finding a path between two vertices in an undirected graph [27]. However, complexity theorists typically do not try to optimize the running time of their constant-work-space algorithms, whereas one of our objectives is to solve a given problem as quickly as possible under the memory constraint.

There are other models that allow only read-access to the input, such as the *streaming* model [25] or the *multi-pass* model [10]. In these models, the input can be read only a bounded number of times in sequential order, whereas we allow the input to be accessed in any order and as often as necessary. Other memory-constrained models are *succinct* data structures [22] and *in-place* algorithms [7–9,12]. The aim of succinct data structures is to use the minimum number of bits to represent a given input. Although this kind of approach significantly reduces the memory requirement, in many cases $\Omega(n)$ bits are still necessary. For in-place algorithms, we also assume that only $O(1)$ cells of work-space are available. However, in this model we are allowed to reorder and sometimes rewrite the input data. This makes it possible to encode our data structures through appropriate permutations of the input and often to achieve the best possible running time. Several classic geometric problems, such as convex hull computation or nearest-neighbor search, have been considered in this model [7–9,12]. Note that the improved running times in the in-place model come at the expense of requiring more powerful operations from the computational environment, making the results less widely applicable. Moreover, in most cases the input values are reordered and the original ordering cannot be recovered after the algorithm is executed. In our problem definition, the input polygon is given as a list of vertices in (say) clockwise order. Hence, losing the ordering of the input is equivalent to having the input destroyed.

A classic algorithm from computational geometry that fits into our model is the gift-wrapping method (also known as Jarvis' march): given n points in the plane, we can report the h points on the convex hull in $O(nh)$ time using $O(1)$ cells of work-space [28]. More recently, Asano et al. [2] initiated the systematic study of constant-work-space algorithms in a geometric context. They describe algorithms to compute well-known geometric structures (such as the Delaunay triangulation, the Voronoi diagram, and the Euclidean minimum spanning tree) using $O(1)$ cells of work-space. They also show how to obtain a triangulation of a planar point set and how to find the edges of the shortest path between any two points inside a simple polygon. These algorithms use constant work-space and run in quadratic time. We emphasize once again that since the output may have linear size, it is not stored, but reported piece by piece. Recently, Barba et al. [5] gave a constant-work-space algorithm for computing the visibility region of a point inside a simple polygon.

[3] A *plane straight-line graph* (PSLG) consists of a planar point set V (vertices) and a set E of non-crossing line segments with endpoints in V (edges). By planarity, we have $|E| = O(|V|)$.

Download English Version:

https://daneshyari.com/en/article/10327393

Download Persian Version:

https://daneshyari.com/article/10327393

Daneshyari.com