Memory-constrained Algorithm for Geometric Problems

Matsuo Konagaya (1010022)

School of Information Science,
Japan Advanced Institute of Science and Technology

February 6, 2012

Keywords: algorithm, constant-space algorithm, memory-constrained algorithm, computational geometry, workspace.

In this thesis we consider memory-constrained algorithms which are algorithms using restricted memory space in addition to read-only arrays storing input data. Recently, CPUs have got faster and memory devices have got cheaper than before. So, many applications use much storage for processing. On the other hand, intelligent devices such as iPad and digital camera have been popular. These devices can perform many high-functional applications in spite of their small sizes. Usually, thier work space is limited. So, we need space efficient algorithms.

We consider a situation that we can use only constant workspace to solve some problems. It is called a *constant-space algorithm*. This kind of algorithms have been studied for 30 years by the name of *log-space algorithm*. But, many results of log-space algorithm are theoretical, and too complicated to implement them. On the other hand, our purpose involves to design implementable algorithms with constant workspace.

Being able to use only constant-space is the hardest condition in memory-constrained algorithms. It is possible that we can achieve a faster algorithm if we can use more workspace. Therefore, we also try to design algorithms to use $O(s)$ space, where s is an adjustable parameter specifying the workspace. If we use algorithms with space-time tradeoffs, we can configure size of space used by the algorithms. Usually, algorithms run faster

using larger storage. On the contrary, they take a long time to perform if we save the memory space.

Throughout this paper, we assume that input data are stored in a read-only array. The array is not allowed to reorder elements in the array. We should not reorder and delete any elements in the input array on which different algorithms are performed in parallel. For example, a picture image is an input of digital camera. We should not rewrite the input data.

In this paper, we propose space-efficient algorithms for two geometric problems. One of them is a constant-space algorithm to compute farthest-point Voronoi diagram for a given set of n points. It runs in $O(n^2)$ time with constant workspace. The other is a space-efficient algorithm with space-time tradeoffs for the line segment intersection problem. We also show that there is a good algorithm with space-time tradeoffs if a given set consists of only vertical and horizontal line segments.

The farthest-point Voronoi diagram for a set of n points is a partition of the plane into convex regions, $FVR(p_1), \ldots, FVR(p_k)$, such that any point in $FVR(p_i)$ is farther from point p_i than from any other point in the given set. Known algorithms for the problem run in $O(n \log n)$ time with $O(n)$ workspace.

The farthest-point Voronoi diagram can be used to solve the other problems, such as smallest enclosing circle problem. This problem is to find the smallest circle enclosing all given points in the plane. To find the smallest enclosing circle for a point set S, we construct the farthest-point Voronoi diagram for S, which is denoted by $FV(S)$, and enumerate all Voronoi vertices and edges. Usually, $FV(S)$ is described by a doubly-connected edge list (DCEL) with $O(n)$ space. Once we construct the data structure, we can enumerate all Voronoi vertices in $O(1)$ time per vertex in S. However, the construction takes $O(n \log n)$ time using $O(n)$ workspace.

Our algorithm behaves just like DCEL. That is, we can enumerate all Voronoi vertices and Voronoi edges, and we can follow boundaries of a Voronoi region using constant workspace. The key fact is that only those points on the convex hull of S have Voronoi regions. Convex hull is the smallest polygon containing all the points in S. Therefore, for each vertex on the convex hull of S, we compute corresponding Voronoi vertices and edges. It takes $O(n^2)$ time to enumerate all the Voronoi vertices and edges

per vertex on the convex hull of S.

The line segment intersection problem is to find all the intersecting pairs for a given set of n line segments in the plane. The problem is one of the most fundamental problems in computational geometry. Thus, there are many researches for the problem. Bentley and Ottman's algorithm by plane sweep techinique runs in $O((n+K)\log n)$ time and $O(K)$ workspace, where K is the number of the intersecting line segment pairs. It is known as the first non-trivial algorithm for the problem. More efficient algorithm which runs in $O(n \log n + K)$ time with $O(n)$ space was obtained by Balaban. The space-efficient algorithm proposed by Chan and Chen works in $O((n+K)\log^2 n)$ time and $O(\log^2 n)$ extra space. The algorithm is based on an in-place algorithm, which uses an array storing input data as a workspace and $O(1)$ extra space. Note that a brute-force algorithm runs in $O(n^2)$ time using only $O(1)$ space.

The proposed algorithm consists of two steps. Let S be a given set of n line segments in the plane. In the first step, we partition S into subsets $S_1, \ldots, S_{\lceil n/s \rceil}$, each contains at most s line segments, and find the intersecting pairs within each subset S_i. Then, we take two subsets S_i and S_j, and find line segment pairs intersecting each other among $S_i \cup S_j$.

To find intersections within a given set, we use the Balaban's algorithm. It is similar to a plane sweep method, which a vertical line moves left to right in a plane. We can find all intersecting pairs in $O(s \log s + K_i)$ time, where K_i is the number of intersecting pairs within S_i.

But, our algorithm has a problem in the second step. It finds intersections not only among $S_i \cup S_j$, but also within S_i and S_j. Therefore, one intersection appears many times in our algorithm.

We modify our algorithm in the second step as follows. After the first step, it constructs a data structure for S_i. And then, for each segment $s_j \in S \setminus S_i$, it finds all the intersections with line segments in S_i.

The data structure is called CSW data structure, which is quasi-optimal data structure for simplex range search proposed by Chazell, Sharir, and Welzl. Once we have the data structure, algorithm can find K line segments in S_i intersecting with a line segment in $O(n^{1+\epsilon}/s^{1/2} + K)$ time. Its construction takes in $O(s^{(1+\epsilon)^2})$ time, where ϵ denotes a positive constant chosen arbitrary small.

If given line segments are either horizontal or vertical, then we can obtain an ideal space-efficent algorithm in $\mathrm{O}((n^2/s)\log s + K)$ using $\mathrm{O}(s)$ space. Even though the input contains line segments with only c kinds of slopes, we can design an algorithm with space-time tradeoffs in an analogous way.