Succinct Quadtrees for Road Data*

Heikki Saul

Abstract—Using conventional data structures for nearest neighbor searches on actual road map data is memory-intensive for both storage and working memory (RAM). In the article "Succinct Quadtrees for Road Data"[1] the authors propose using succinct quadtrees as a space-efficient data structure for road data, allowing for fast in-place queries for 2D nearest point and segment searches. These kind of searches are a critical part of GPS map matching for navigation applications.

I. INTRODUCTION

The article covered in this essay offers a new approach to a fundamental computational geometry problem: optimal data structures for nearest neighbor queries. Many currently popular data structures are optimized for speed and their memory usage does not scale well for large data, for example the nearly 55 million edges of the Japanese road network. There are a number of compressed data structures for 2D points, which to not support queries for line segments. There are also existing structures for line segments, but their memory usage is high. The algorithm proposed by the authors aims to fill a gap for a space-efficient data structure for line segments. Such a structure could be used in embedded devices for matching GPS coordinates to their nearest road to create more accurate navigation solutions.

II. PRELIMINARIES

A. Quadtrees

Quadtrees were proposed by Finkel and Bentley [2]. It is a 2D equivalent of octrees. In quadtrees, each internal node has exactly 4 children. The behavior of the tree is dependent on the bucket size. Each node can contain points up to the bucket value. If the bucket is exceeded, the node is split into 4 children. In Fig. 1 you can see the 2D representation of a quadtree based on some point data. It is easy to see that this is a fitting structure for map data, as the depth of the tree increases only in areas with more points.

B. Succinct data structures

Succinct data structures, as first proposed by Jacobson[3], are data structures, which use space close to the information-theoretic lower bound, but still retain their ability to be used in-place, unlike general lossless data compression algorithms. In the article being covered, the authors used one succinct data structure for sparse bit-vectors[4][5] and one for ordered trees[6][7]. They also used a variant of the DFUDS[8] representation of an ordered tree[5]. Due to the properties of quadtrees, the representation could be simplified to save space. Operations on the tree could be done in constant time[8].

III. SUCCINCT QUADTREES

Consider point set $P$ in a quadtree $T$. Assume all points are in a 2D universe $R = [0, W] \times [0, H]$. The root node of $T$ corresponds to $R$. If the number of points is greater, than the bucket capacity $C$, the root node $R$ is divided into four parts, so that $R_0 = [W/2, W] \times (H/2, H)$, $R_1 = [0, W/2] \times [H/2, H)$, $R_2 = [0, W/2] \times [0, H/2)$, and $R_3 = [W/2, W] \times [0, H/2)$. The point set $P$ is also divided among these child nodes. If any child node holds more than $C$ points, it is recursively split, until the number of points in any single leaf is $0, C$. The resulting tree $T$ is stored in a slightly modified version of DFUDS representation. Each node can be distinctly identified a preorder over the entire tree or just the leaves. Also, each node can be identified by the labels on the edges leading to the node from root. Due to the way quadtree subregions (nodes) are formed, each subregion is represented by the $W$, $H$ values and DFUDS representation. As points in a leaf node are stored separately, we must also maintain a pointer to the set of points in each leaf.

A. Storing points

Points inside a leaf are stored as base points and diff points. All base points are ordered by their relative $x$-coordinates in the region, which are then stored as differences from the previous points. $y$-coordinates are not sorted, so negative values need to be encoded. Diff points have a base point in the same subregion (leaf) and they are also sorted by their $x$-coordinates with only the coordinate difference being stored. The $x$-coordinate of a diff point is the summation of all the differences from the corresponding base point. Nearest
neighbor queries can be performed based on the distance between the query point and data points in the tree.

B. Storing polygonal chains

A polygonal chain is a set of line segments \((v_0, v_1), (v_1, v_2), \ldots, (v_{m-1}, v_m)\) where \(v_i\) are points. This kind of structure is often used to store road data. Polygonal chains allow for nearest neighbor queries based on the distance between the line segment (a segment of road) and the query point. Both ends of the line segment must be in one leaf, otherwise the line segment is cut on the node boundary. Points are divided into base and diff points greedily. We define \(v_0 = p\) as a base point, if a point \(v_i \ldots v_m\) is closer than a predefined distance \(L\), it is a diff point. Otherwise \(v_i\) is selected as the next base point. Points in line segments are encoded similarly to storing points, but as diff points are ordered in polygonal chains, they can not be sorted. Instead, they are kept in the same order as they appear in the polygonal chain. Long line segments are split by adding dummy nodes. This increases the number of segments, but queries are faster.

IV. DIGITAL ROAD MAP

Digital Road Map (DRM) is a standardized data format used for road maps in Japan. The universe is divided into primary meshes (1 degree longitude times 2/3 degrees latitude), which are then divided into \(8 \times 8\) secondary meshes (roughly 10 km by 10 km). In a secondary mesh, normalized coordinates from 0 to 10000 are used. For convenience, the quadtree was aligned with the secondary meshes covering the entirety of Japan. As roads have IDs, those were also stored, albeit changed in a way that they coincide with the polygonal chain order in the data structure.

V. EXPERIMENTAL RESULTS

The performance of the data structure was compared with ANN, a standard C++ library for nearest neighbor queries. ANN is based on k-d-trees.

A. Experiment setup

The comparisons were run on a workstation with 64GB RAM and an Intel Xeon processor, using only one core. ANN was set up to perform bounded radius, exact result nearest neighbor queries on points, as it does not support line segment queries. According to the authors, their algorithm has very similar performance on both point and segment queries. The DRM dataset has around 54 million points, which took 411 MB, if coordinates were stored uncompressed and 174 MB when compressed. The DFUDS structure took 2.5 MB.

B. Leaf cache

To speed up queries, a number of decoded leaves can be kept in memory in a decoded state. Following tests with multiple values, as seen on Fig. 2, the authors decided to keep the last 50 leaves in cache in decoded form. This offered a good balance between query speed and memory usage.