IT University of Copenhagen

Software Development MSc

Algorithm Design Project

QuadSketch

Authors:
Anders Edelbo Lillie
Emil Refsgaard Middelboe
Fredrik Emil Bakke
Stefán Björn Pétursson

Supervisors:
Samuel McCauley
Johan von Tangen Sivertsen

April 11, 2018

Abstract

Contents

1 Introduction

1.1 Motivation

1.2 Case

1.3 Problem definition

2 Contribution

3 Background

4 Analysis

4.1 Why randomly shift hypercube?

The notion of randomly shifting a hypercube in each dimension centered on a single point may seem confusing. The action is however necessary for the `QuadSketch` implementation to maintain guarantees on arbitrary datasets. We hope to obtain a cube in which data points are shifted into the same quad leaves / buckets and avoid data points near to each other end up in different leaves. The shifting can have varying practical effects given the randomness of the shifting.

If the dataset is of a known format, randomly shifting it in each dimension could turn out to be less effective than taking a more specific approach. An example could be a dataset in which all the points are very close to each other. A large amount of the points might end up in the same leaves and as such defeat the purpose of using the *Quad Tree* to begin with. Randomly shifting over these points will not be of any significant gain to the problem at hand. One approach could be to spread out the dataset in the sense of scaling the points given some constant to increase the distance between them and hereby obtain a *Quad Tree* of a better quality. The original relative distance is preserved by remembering the scaling constant.

4.2 Building the sketch

The sketch is a compressed representation of a point set `X`. The points of `X` have `d` dimensions and are in *euclidean* space. Each coordinate of a point is represented by `B` bits. From this data `QuadSketch` can be run. `QuadSketch` will take as input the point set `X` and two additional parameters `L` and Λ. These two parameters are used to specify the compression of the point set where `L` is the depth of the *Quad Tree* that is built and Λ is the amount of nodes from the *Quad Tree* which can be removed. There are three steps for building the sketch: randomly shifted grid, *Quad Tree* construction and pruning. These steps will be explained briefly and there will be given an example of how a sketch is built on a small example.

The first step is randomly shifted grid. Here there is constructed a hypercube `H` which contains all points of the given point set `X`. `H` will then be set up to be centered around a point of `X`. Then choose a random value σ for each dimension of the hypercube. Each dimension `j` of `H` will be shifted with σ_j.

The next step is creating the *Quad Tree*. There is created a *2d-ary Quad Tree* by starting at the root of `H`. There is then added the children which contain a point from `X`, thus child nodes that do not contain a point from `X` are ignored. On the edges to child nodes there is a label `D` long of the bit string for the i'th position of the points bit string formatted as the bit for the first dimension then the second and so forth. This step is then done recursively until the level `L` is reached. An example of a constructed *Quad Tree* is given in the following figure:

TODO ADD FIGURE

After construction of the *Quad Tree* the tree is pruned. Here there is found any downward of length `k` where the nodes only have one child. If `k` $> \Lambda + 1$ then there are removed nodes from the *Quad Tree* and there is inserted a long edge from labeled with the length of the path it replaces. This is given an example of the three after a pruning with $\Lambda = 1$:

TODO ADD FIGURE

From this the sketch can be built. For this there is used the "Eulerian Tour Technique"[1]. It starts at the root an searches down to the leftmost leaf and the to the new leaf by going upwards again. When there a edge is explored downwards there is stored a 0 and the label of the edge either a bit string or the length of an long edge with and a bit specifying if the given edge is a long or short edge. If an upwards edge is explored then a 1 is stored. Furthermore, there is stored for each point a child index for the child node which contains it.

4.3 Theorems

The paper introduces three theorems, where *Theorem 1* is the main theorem covering the basic variant of `QuadSketch`. *Theorem 2* covers an advanced variant, and *Theorem 3* is the full version. Comparing *Theorem 1* and *Theorem 2*, it is difficult to immediately understand the difference. The former makes guarantees for a `QuadSketch` variant, where "for each point, the distances from that point to all other points are preserved up to a factor of 1+- ϵ with a constant probability", whereas the latter "makes it possible to approximately preserve the distances between all pairs of points". It seems like *Theorem 1* thus allows us to find distances between one single point and to all other points, but not the other way around, whereas with *Theorem 2* we can find the distance both from node 1 to node 2 and 3, but also from node 2 to node 3. This comes with a trade off, as with *Theorem 1* you are able to recover any points by decompressing the sketch, which is infeasible with *Theorem 2*, where `QuadSketch` is recursively applied.

[1]See e.g., https://en.wikipedia.org/wiki/Euler_tour_technique

5 Methods

This section will cover which methods have been used throughout the project and why the methods have been chosen. Each method is motivated to support the problem definition and the analysis of the `QuadSketch` implementation.

5.1 Experiment verification

5.1.1 Baseline comparison

The paper compares the `QuadSketch` performance with other compression algorithms, including *Product Quantization* (`PQ`) and a simple baseline algorithm they call Grid. In order to verify their results, a version of Grid has been implemented as baseline to use for replication experiments. The basic concept is as follows... And is implemented as follows...

5.1.2 Other datasets

To ensure proper verification of the results for the `QuadSketch`, the implementation and baseline must be tested on another dataset not included in the paper. If the results from another dataset somewhat match the results given in the paper [Indyk et al., 2017], it will strengthen the credibility of the practical efficiency of the `QuadSketch` implementation.

5.2 Testing the improvements

6 Results

6.1 Practical improvements

7 Discussion

7.1 Threats to validity

8 Conclusion

9 References

[Indyk et al., 2017] Indyk, P., Razenshteyn, I., and Wagner, T. (2017). Practical Data-Dependent Metric Compression with Provable Guarantees. Technical report, MIT, Long Beach, CA, USA. 31st Conference on Neural Information Processing Systems (NIPS 2017).

A Example