Fast, Compact NN-table build using Equi-Voronoi Polytopes

Alan Dearle, Richard Connor, Ben Claydon, and Ferdia McKeogh

University of St Andrews

Highlights

We have submitted solutions to both challenges using our newly-invented technique of 2-bit quantisation based on Equi-Voronoi Polytopes (EVPs)

This gives two very clear advantages

1. The data is compressed to 2 bits per vector element requiring only around 6% of the 32-bit floating point space This representation is paired with binary similarity metric, *b2sp*, which is highly parallelisable on SIMD processors
2. The compression maintains sufficient accuracy to give reasonable results over the original Euclidean space

For both challenges we build a near-neighbour (NN) table using:

- the ultra-quantised data
- the b2sp metric
- a variant of the NN-Descent algorithm,

which has the advantage of using fixed-size memory

The combination of these techniques along with a parallel, lock-free Rust implementation allows us to build a reasonably accurate near-neighbour table very quickly

Written in Rust

Code may be found at https://github.com/MetricSearch/metric_space_rust

The build Algorithm

```
Algorithm 1: Our adaptation of NN Descent
Data: Dataset V, similarity oracle σ, K, δ
Result: K-NN list B
begin
    B[v] ← RANDOM(V,K) × {⟨∞,true⟩}   ∀v ∈ V
    loop
        parallel for v ∈ V do
            old[v] ← all items in B[v] with a false flag
            new[v] ← all items in B[v] with a true flag
            mark sampled items in B[v] as false
        end
        rev = REVERSE(B)
        c ← 0   //update counter
        parallel for v ∈ V do
            newRev[v] ← new[v] ∪ rev[v]
            for u1,u2 ∈ newRev[v], u1 < u2
            or u1 ∈ new[v], u2 ∈ old[v] do
                l ← σ(u1,u2)
                // c and B[.] are synchronised
                c ← c + UPDATENN(B[u1], ⟨u2,l,true⟩)
                c ← c + UPDATENN(B[u2], ⟨u1,l,true⟩)
            end
        end
    end
    return B;
end
```

Build times

On the challenge platform provided

- Low levels thread parallelisation
- Very old hardware
- Space limited context

Challenge 1 PUBMED23 23 million vectors (384 dimensions) perform k=30 nearest neighbour queries

Build time 2980s

Challenge 2 GOOAQ 3 million vectors (384 dimensions) build *k*-nearest neighbour graph for *k=15*

Build time 528s

Parallelism

The key to performance is the amount of parallelism which can be achieved

The update loop is highly parallelisable (way more than the thread limit provided in the challenge), but with the major caveat of requiring random write access to:

- the near-neighbour table
- the similarities
- and flags tables

Furthermore, there are a very large number of updates made, for example with the GOOAQ dataset in the second iteration there are 257,949,066 updates made to each of these data structures

Update conflict to any of them could result in undefined behaviour and erroneous results, and use of locking would be prone to deadlocks and incur a major performance penalty

An atomic 64 bit value consisting of a (id, distance) pair called a *Nality* is used

This permits the use of atomic compare and exchange (single instruction) update in a lock free manner

What slowed queries down

The tables are very narrow to keep within the space budget

Therefore, we could **not** meet the Challenge threshold of 70% recall for 30@30 queries

We therefore perform 100NN approximate queries in the quantised space

We **reorder** results by incrementally loading the **original 32-bit float data** of the returned ids which gives around 80% accuracy

This requires random access to the hd5 files which is **SLOW**!

{al, rchc, bc89, fm208}@st-andrews.ac.uk