Equi-Voronoi Polytopes: A Geometric Basis for 2-bit Quantisation

Richard Connor†, Alan Dearle†, Ben Claydon†

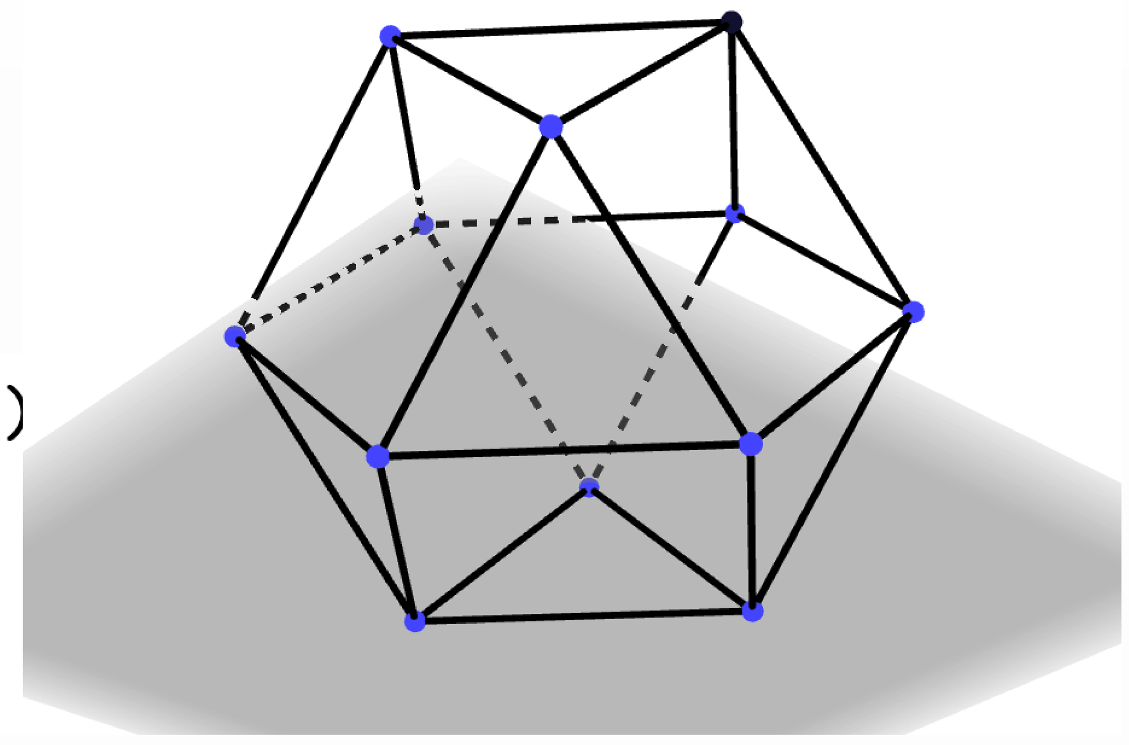
† School of Computer Science, University of St Andrews, Scotland

Overview

The cuboctahedron

- tessellation within a sphere
- not regular, but vertices are congruent
- all vertex coordinates are drawn from $\{-1, 0, 1\}$, in all permutations with a single zero

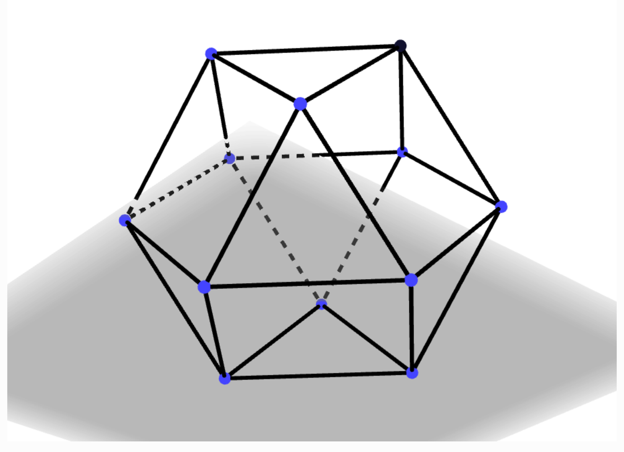
```
(-1,-1,0) (-1,0,-1) (0,-1,-1)
(-1,1,0) (-1,0,1) (0,-1,1)
(1,-1,0) (1,0,-1) (0,1,-1)
(1,1,0) (1,0,1) (0,1,1)
```



Generalisation to many dimensions

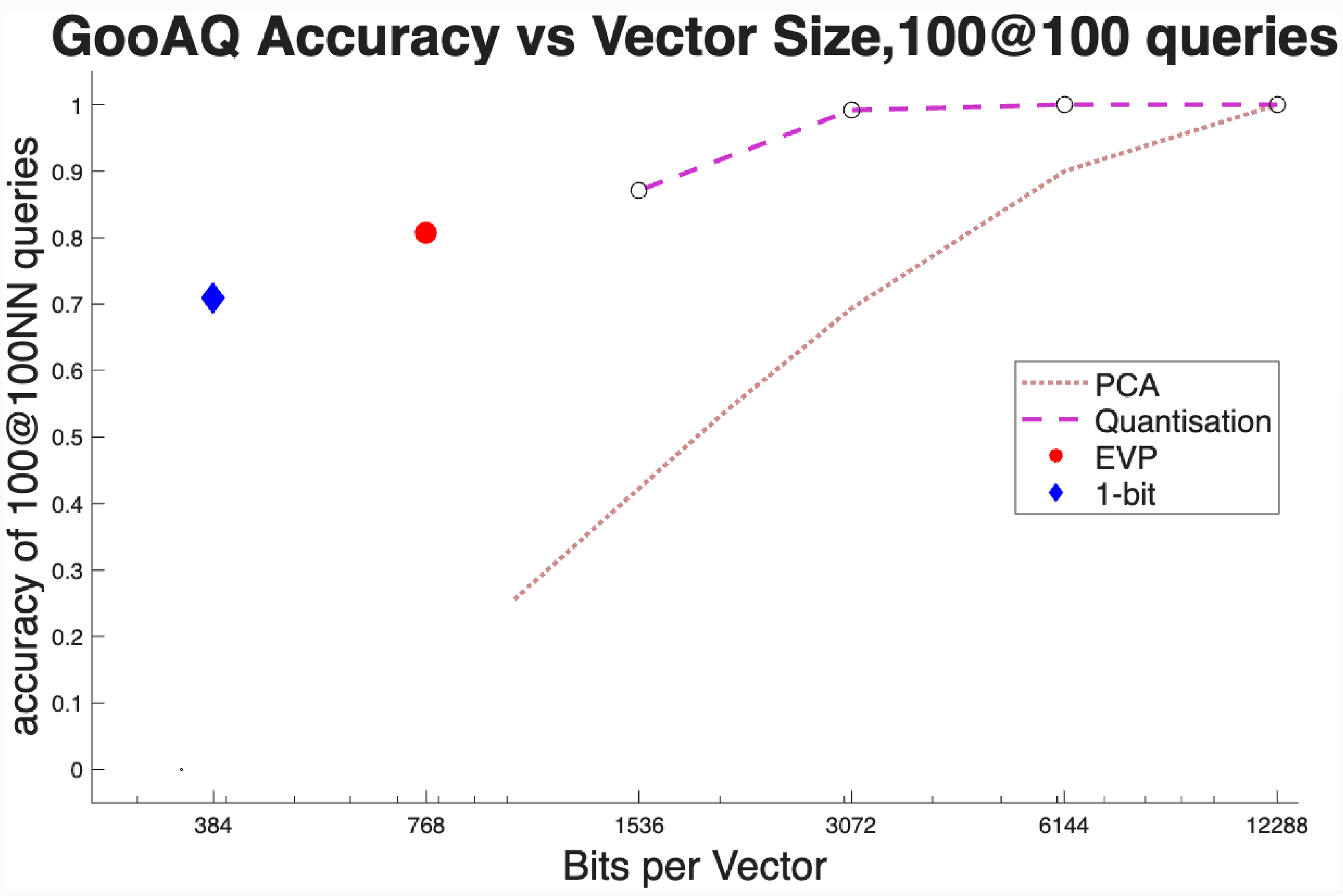
- The $\{x, d\}$ equi-Voronoi polytope: in d dimensions, with x non-zero coordinate elements
- we find, for any two points in the d-dimensional hypersphere, the distance between their nearest EVP vertices is a good approximation
- finding the nearest vertex, and calculating vertex distances, are *very* efficient operations
- storage: 2 bits per dimension
- number of vertices: $\binom{d}{x} \cdot 2^x$ (i.e. huge!!)

Finding the nearest vertex



- maximise scalar product of datum and vertex
- which is easy:
  - find x largest absolute values
  - set to $\{-1, 1\}$according to sign
  - set others to 0

Accuracy



Calculating vertex distance

- fixed x: all vertex norms are the same ($\sqrt{x}$)
- so can use scalar product as similarity function
- all element values are in $\{-1, 0, 1\}$!
- so worst case is a masked addition operation
- but we have a better way: b_2sp over binary strings

b_2sp

u_1	float	0.32	0.4	-0.38	-0.19	0.29	0.45	0.44	-0.16	0.23	-0.02
u_2	float	-0.16	-0.4	0.38	0.45	0.14	0.19	-0.38	-0.04	0.4	-0.35
v_1^+	binary	1	1	0	0	0	1	1	0	0	0
v_1^-	binary	0	0	1	0	0	0	0	0	0	0
v_2^+	binary	0	0	1	1	0	0	0	0	0	0
v_2^-	binary	0	1	0	0	0	0	1	0	1	0

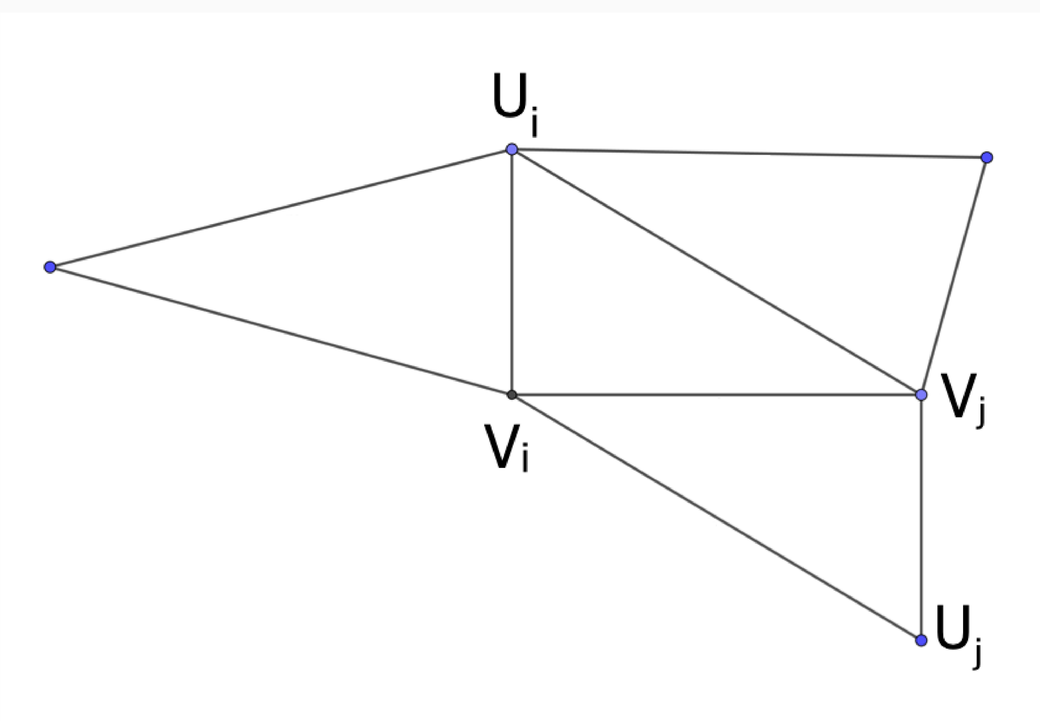
$$\begin{aligned} b_2sp(v,w) &= (bsp(v^+,w^+) + bsp(v^-,w^-)) \\ &\quad - (bsp(v^+,w^-) + bsp(v^-,w^+)) \end{aligned}$$

where

$$bsp(v,w) = \textit{bitcount}(v \wedge w)$$

- multi-dimensional data stored in two contiguous binary blocks, each of d bits
- scalar product proxy function b_2sp is massively parallelisable on commodity hardware
- combination for e.g. hundreds of dimensions, 100 × speedup

Why it works...



- for general points U_i, U_j and their nearest vertices V_i, V_j:
- we are in a Hilbert space, so can construct a tetrahedron in 3D
- distances $d(U_i, V_i)$ are very *small*, and very *constrained* - because there are a huge number of vertices
- angles $\angle V_i, V_j, U_j$ are very constrained - because we are in a high-dimensional space

- $d(V_i, V_j)$ is a **very good** estimator for $d(U_i, U_j)$