Product Quantization for Nearest Neighbor Search

Joe Durie, CS 848



K-approximate nearest neighbor search

K-approximate nearest neighbor search:

- Given a set S of n vectors y in $\mathbb{R}^D$, and a query vector x, want to find an approximation of the K nearest neighbors of x in S

- We only consider the L_2 norm for $\mathbb{R}^D$, and we will consider minimizing the squared distance, since that is equivalent

$$d(x, y)^2 = \sum_i (x_i - y_i)^2$$

Quantization

Given a set S of n points in $\mathbb{R}^D$, want to create a set of k centroids, and assign each point y in S to a centroid $q(y)$ such that $q(y)$ can act as a representative of y

We want $k \ll n$, to reduce the time complexity and memory usage

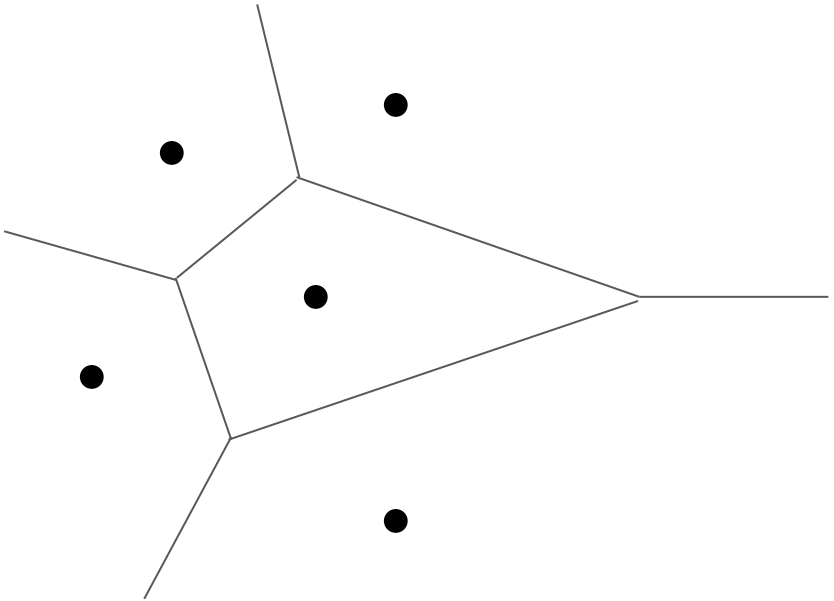
However, we want k to be large enough that the error is relatively low

We also enforce two conditions on the centroids

Lloyd Optimality Condition #1

Each point should be assigned to the nearest centroid

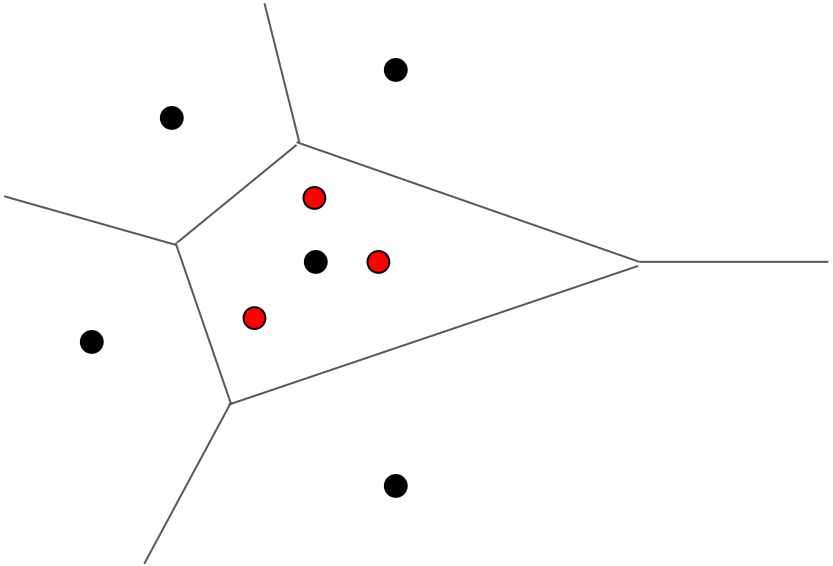
So the set of points assigned to each centroid is the Voronoi cell corresponding to that centroid



Lloyd Optimality Condition #2

Each centroid should be the expected value of all points in its cell

Given that we have a finite number of input points, we can interpret this as the centroid needing to be the average of the input points that are closest to it



Lloyd Quantizer (k-means clustering)

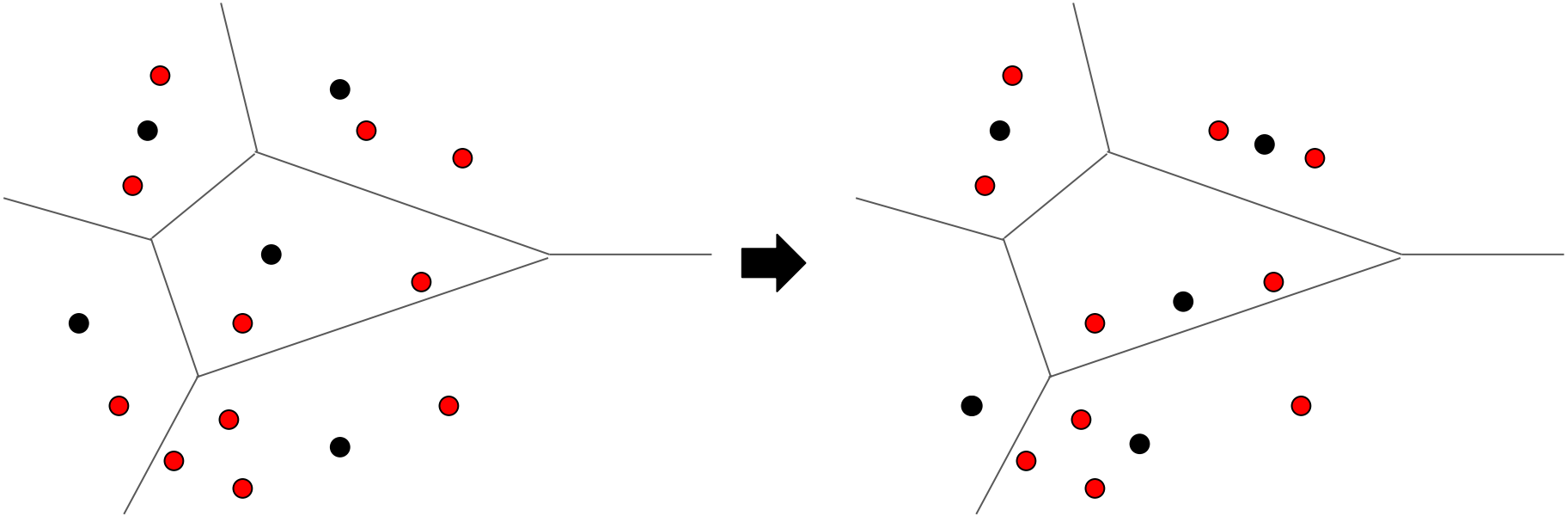
Explicitly computing centroids satisfying these constraints is difficult

We can approximate them by initially arbitrarily assigning the centroids

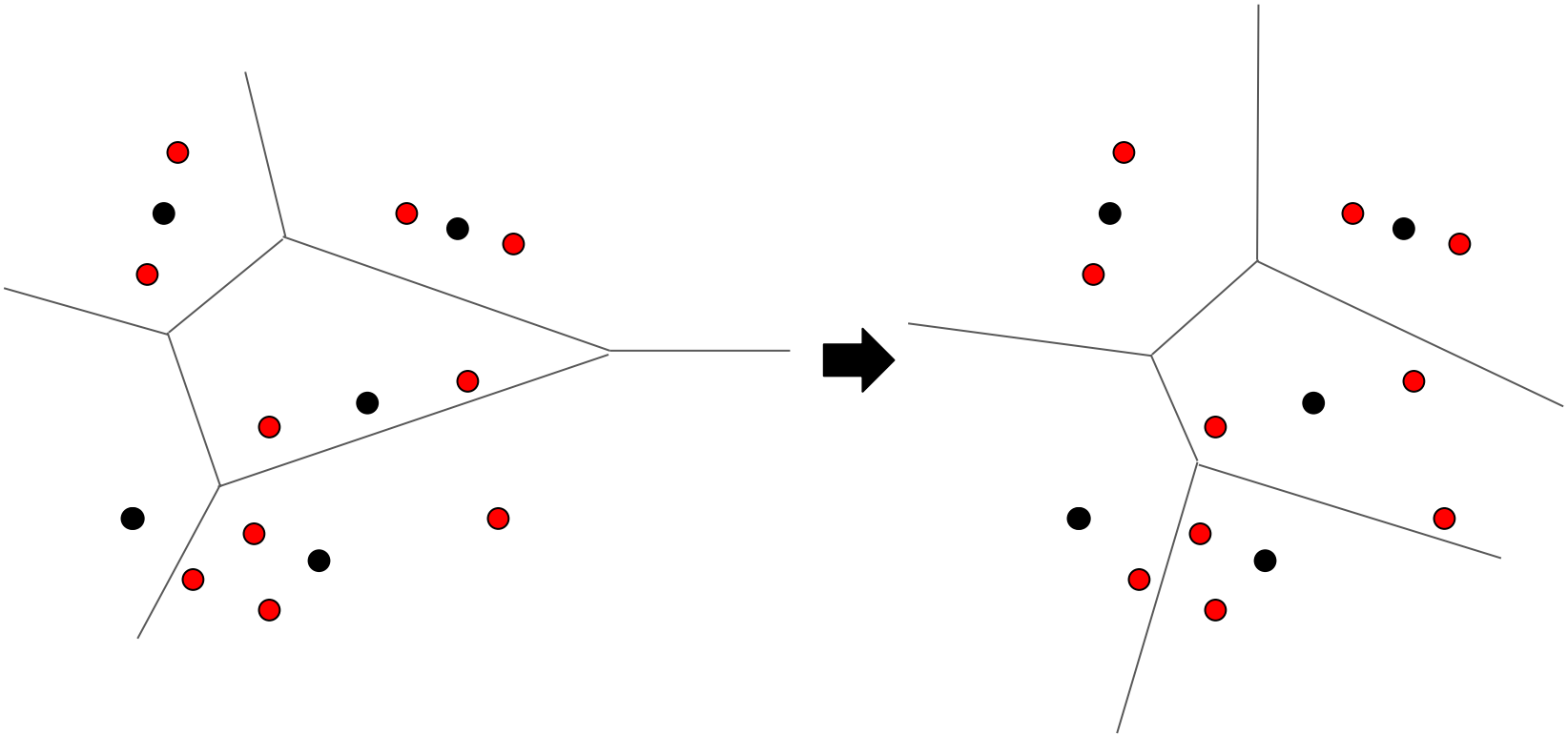
Then repeat the following process:

1. Find the subset of points closest to each centroid
2. Change the centroid to the average of those points

Lloyd Quantizer Example - Step 1



Lloyd Quantizer Example - Step 2



MSE

The quality of a quantizer is measured by its mean squared error (MSE)

$$\mathbf{MSE}(q) = \mathbb{E}_X\left[d(q(x), x)^2\right] = \int p(x)\, d\big(q(x), x\big)^2\, dx$$

This is the expected squared distance between a point and the centroid corresponding to it

Issues with basic quantizers

For high dimensions, often if you want a reasonably low MSE, the number of required centroids is prohibitively high

If $D = 128$ (as in SIFT), even if you only want half a bit of information per dimension, you need 2^{64} centroids!

This is impossible both to store in memory and to iterate through in a reasonable amount of time.

Product Quantization

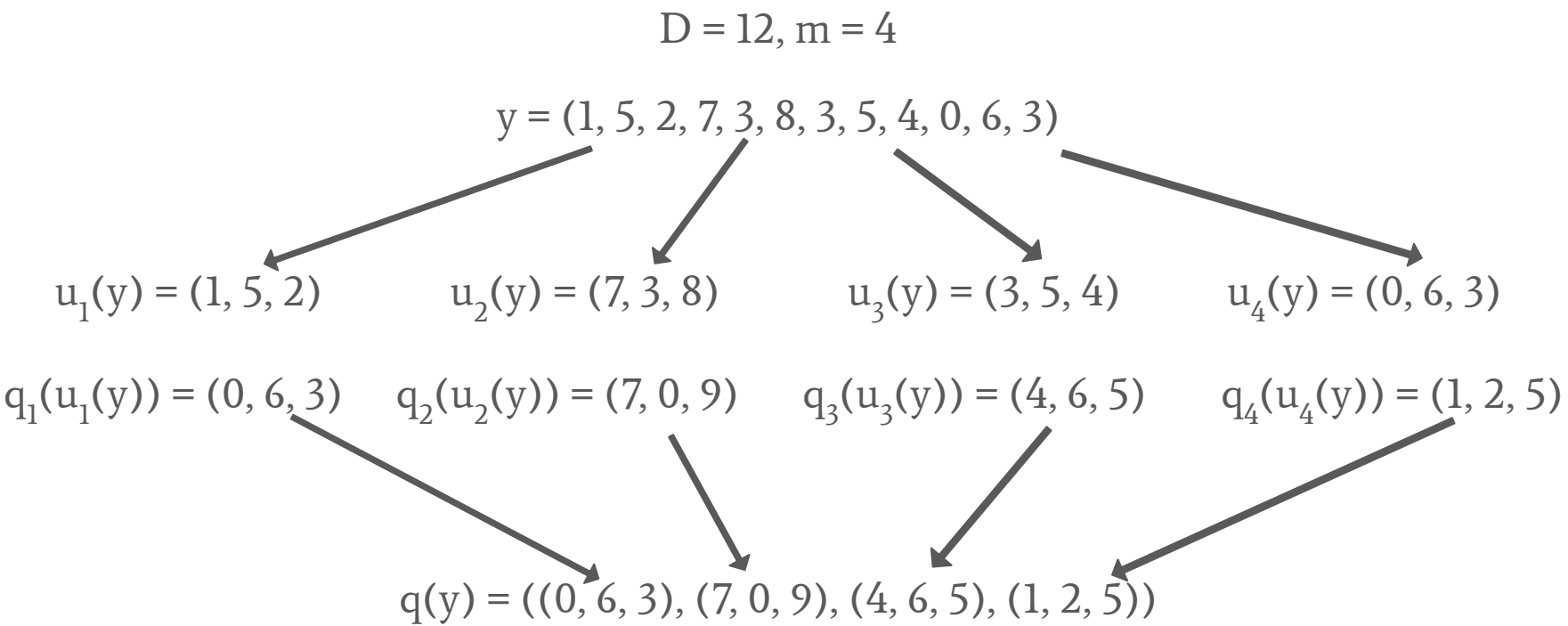
Instead of looking at all D dimensions at the same time, split the coordinates of the vector into m groups and create an independent quantizer q_j with k^* centroids for each group

- We guarantee that m is a divisor of D

The set of centroids in the product quantizer is the Cartesian product of centroids in the smaller quantizers

- Each centroid c is of the form $(c_1, c_2, \ldots c_m)$, where the c_i are centroids of the subquantizers

Product Quantization Example



Benefits of Product Quantization

By storing k^* centroids for each quantizer q_j, the original quantizer q can represent a total of $(k^*)^m$ centroids

However, the total space needed to store all centroids of the subquantizers is proportional to $(k^*)m$

This solves the memory issue discussed earlier

Benefits of Product Quantization

	Non-product quantization	Product Quantization
Total centroids represented	k	$(k^*)^m$
Dimension of stored centroids	D	D/m
Number of stored centroids	k	$(k^*)m$
Total memory of stored centroids	$O(kD)$	$O((k^*)D)$

MSE of Product Quantizer

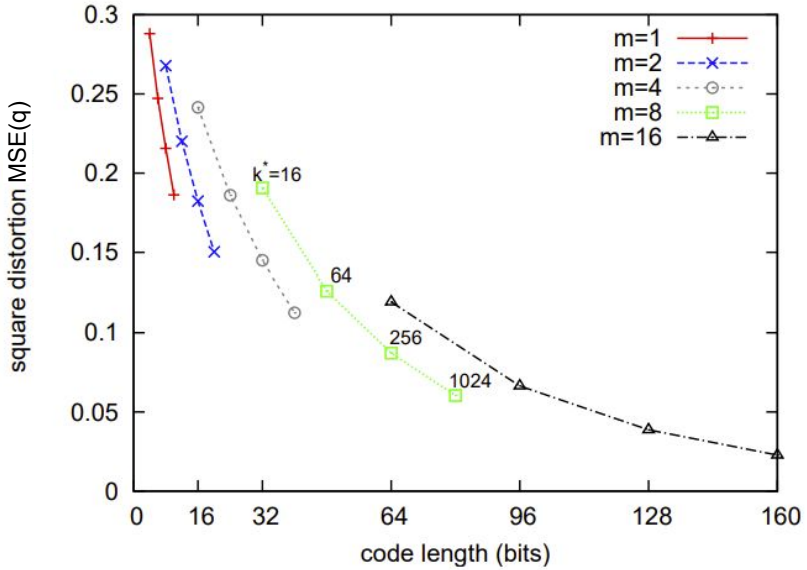
Using the distance formula, we have $d(y, q(y))^2 = \sum_j d(u_j(y), q_j(y))^2$, so the MSE of the product quantizer is simply

$$MSE(q) = \int p(x) d(q(x), x)^2 dx = \int p(x) \left(\sum_j d(q_j(x), u_j(x))^2 \right) dx = \sum_j \int p(x) d(q_j(x), u_j(x))^2 dx = \sum_j MSE(q_j)$$

MSE of Product Quantizer

Code length is the number of bits needed to identify a centroid $(m \log_2(k^*))$

- This is because we have $2^{(m\log 2(k^*))} = (k^*)^m$ possible centroid values

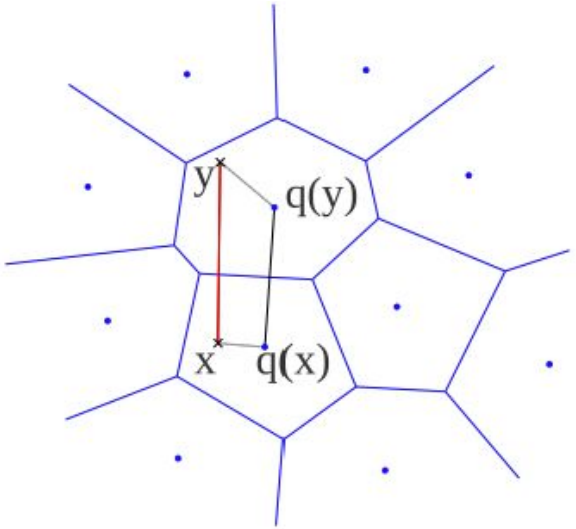


Symmetric Distance Computation (SDC)

Quantize all input points y to $q(y)$

When querying for the distance from query point x to y, quantize x to $q(x)$

Estimate the squared distance with $d(q(x),q(y))^2$



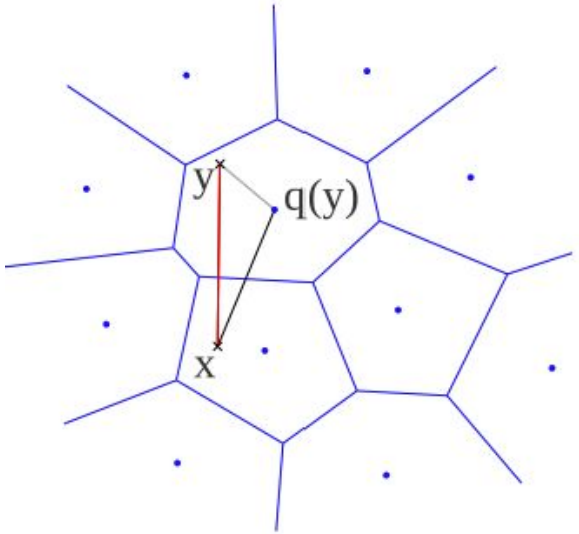
symmetric case

Asymmetric Distance Computation (ADC)

Quantize all input points y to $q(y)$

When querying for the distance from query point x to y, do not quantize x

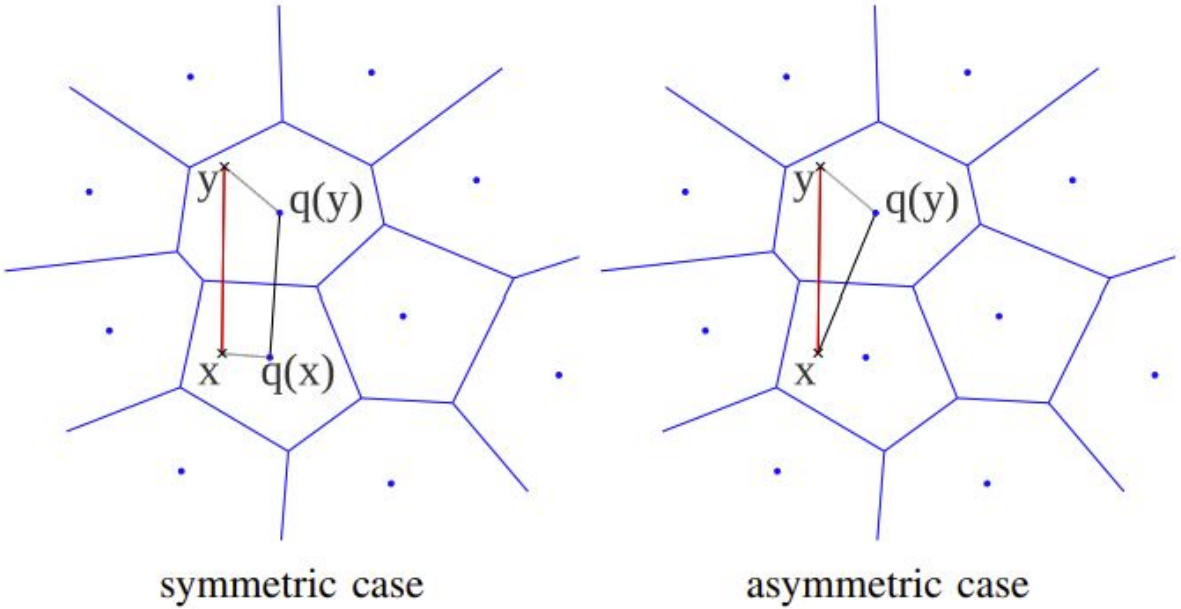
Estimate the squared distance with $d(x,q(y))^2$



asymmetric case

Comparing SDC and ADC (intuition)

SDC has error because of $d(x,q(x))$ and $d(y,q(y))$, ADC only has error from $d(y,q(y))$



symmetric case

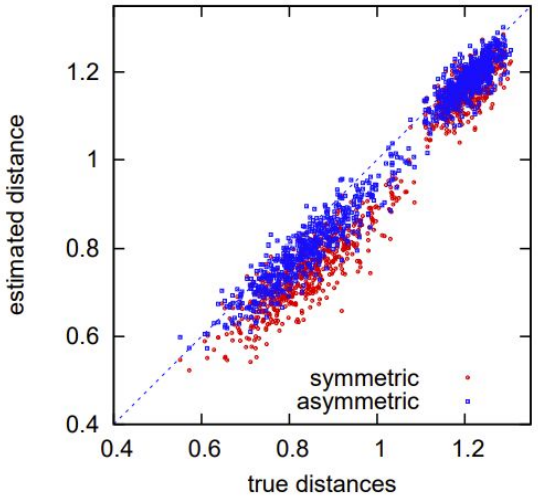
asymmetric case

Comparing SDC and ADC

- Error measured by mean squared distance error:

$$\mathrm{MSDE}(q) \triangleq \iint \left(d(x,y) - \tilde{d}(x,y)\right)^2 p(x)dx\, p(y)dy.$$

- MSDE(SDC) ≤ 2MSE(q)
- MSDE(ADC) ≤ MSE(q)

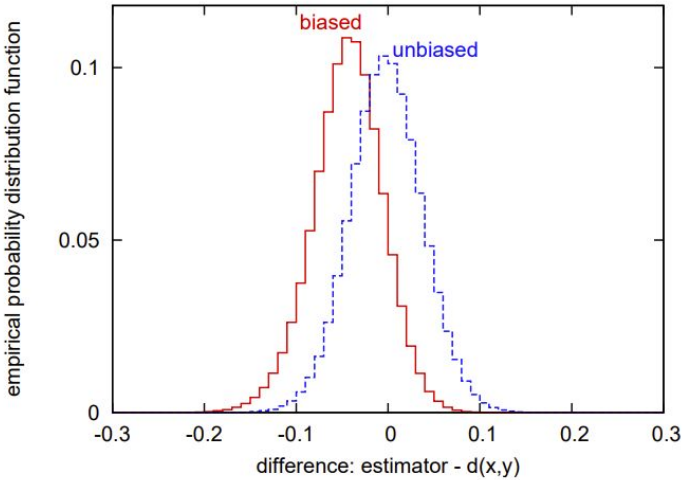


Comparing SDC and ADC

Overall, SDC and ADC both tend to underestimate distances

The authors also tested a version correcting for this, however the MSDE was slightly increased

Because all distances are underestimated by roughly similar amounts, the uncorrected version still produces reasonable nearest neighbors



Queries in SDC

Before the queries, compute a table of $d(c_{ji1}, c_{ji2})^2$ for all $1 \le j \le m, 1 \le i_1, i_2 \le k^*$

For each query point x, we need to find the distance from x to all $(k^*)m$ centroids to quantize x to $q(x)$

For each point y, we have $d(q(x), q(y))^2 = \sum_j d(q_j(x), q_j(y))^2$, so the distance estimate can be computed as the sum of m of the table entries

For each query, we can thus exhaustively get estimates for all $d(x, y)^2$ in $O((k^*)D + mn)$ with $O((k^*)^2D)$ precomputation

- Brute-force would take $O(Dn)$

Queries in ADC

Given a query vector x, compute a table of $d(u_j(x), c_{ji})^2$ for all $1 \le j \le m, 1 \le i \le k^*$

For each point y, we have $d(x, q(y))^2 = \sum_j d(u_j(x), q_j(y))^2$, so the distance estimate can be computed as the sum of m of these table entries

For each query, we can thus exhaustively get estimates for all $d(x, y)^2$ in $O((k^*)D + mn)$

- Brute-force would take $O(Dn)$

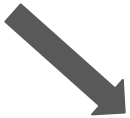
Similar performance to SDC, with less precomputation time and less error

ADC Example - Precomputation

$y = (8, 3, 1, 4), m = 2$

$u_1(y) = (8, 3)$ $\qquad$ $u_2(y) = (1, 4)$





Centroid	Location	Squared distance to $u_1(y)$
c_{11}	(0, 0)	73
c_{12}	(5, 1)	13
c_{13}	(3, 4)	26

Centroid	Location	Squared distance to $u_2(y)$
c_{21}	(7, 1)	45
c_{22}	(8, 3)	50
c_{23}	(0, 2)	5

So we have $q(y) = (c_{12}, c_{23}) = ((5, 1), (0, 2))$

ADC Example - Query

$x = (1, 2, 6, 0), m = 2$

$u_1(x) = (1, 2)$ $u_2(x) = (6, 0)$



Centroid	Location	Squared distance to $u_1(x)$
c_{11}	(0, 0)	5
c_{12}	(5, 1)	17
c_{13}	(3, 4)	8

Centroid	Location	Squared distance to $u_2(x)$
c_{21}	(7, 1)	2
c_{22}	(8, 3)	13
c_{23}	(0, 2)	40

ADC Example - Query

$$y = (8, 3, 1, 4),\ q(y) = (c_{12}, c_{23})$$

$$q_1(u_1(y)) = c_{12} \qquad q_2(u_2(y)) = c_{23}$$





Centroid	Location	Squared distance to $u_1(x)$
c_{11}	(0, 0)	5
c_{12}	(5, 1)	17
c_{13}	(3, 4)	8

Centroid	Location	Squared distance to $u_2(x)$
c_{21}	(7, 1)	2
c_{22}	(8, 3)	13
c_{23}	(0, 2)	40

This gives a squared distance estimate of 57

Problems with Exhaustive Search

This method exhaustively checks all possible y, which is obviously inefficient.

Ideally, we should only be checking y that are close to x, however this is difficult to do with product quantization

- y values that are close to x in one q_i may be very far away in another!

$$x = (1, 2, 3, 7) \qquad y = (1, 2, 1000, 5000)$$

$$u_1(x) = (1, 2), u_2(x) = (3, 7) \qquad u_1(y) = (1, 2), u_2(y) = (1000, 5000)$$

IVFADC - Basic Idea

We'll use two levels of quantization

The first level of quantization q_c is just used to partition the y into groups of nearby points so we can test a small subset of groups and not everything

- This level is not a product quantizer

The second level of quantization is a product quantizer q_p used to run ADC on the difference between points and their first-level centroid c_i

IVFADC - Coarse Quantizer q_c

This is a regular quantizer with a relatively small k' value of 10^3 to 10^6

Called "coarse" because it has much greater error than the product quantizer

For each centroid c_i in q_c, store a list L_i of all input points y that are closest to c_i

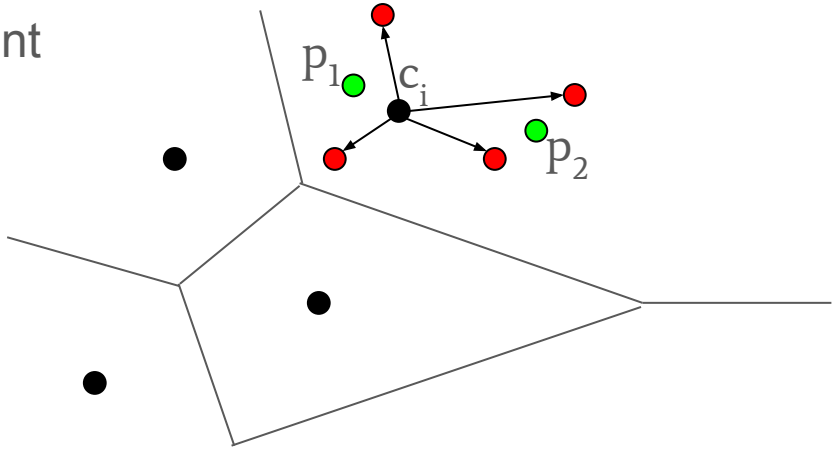
- This is an inverted file structure, hence the name IVFADC

IVFADC - Product Quantizer q_p

q_p quantizes the vectors $r(y) = (y-c_i)$, where $c_i = q_c(y)$

Much higher precision than q_c

To estimate nearest neighbor to an element of L_i, we will use ADC on q_p with the centroids p_j



Queries

Given a query vector x, find the distances $d(x, c_i)^2$ to all centroids of q_c

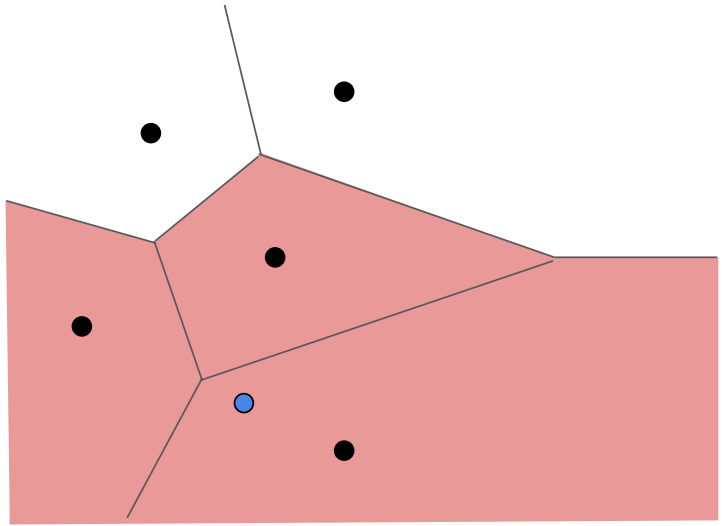
Consider the w closest centroids among these

For each one c_i, find the distance from $r(x) = (x-c_i)$ to $q_p(r(y))$ for all y in L_i

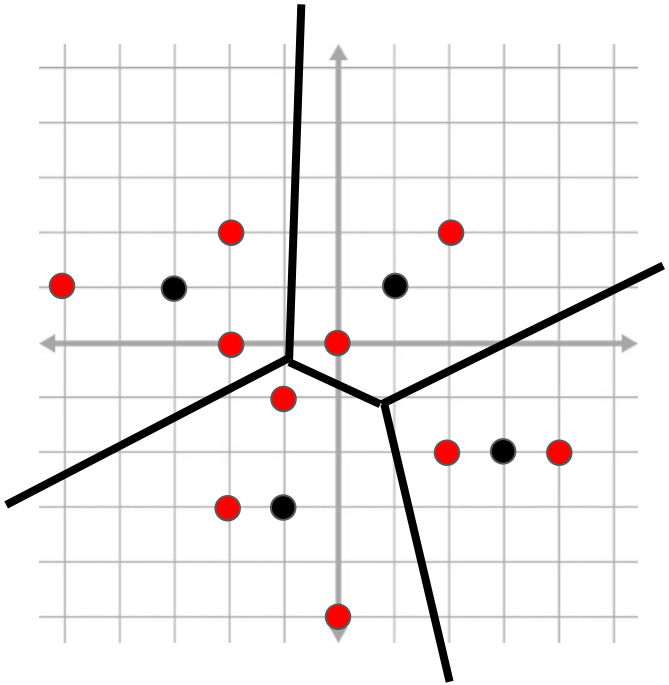
This is done by precomputing a lookup table for all $d(r_j(x), p_{ij})^2$ as in ADC

Then computing $d(r(x), q_p(r(y)))^2$ can be done in m additions, and we choose the lowest K estimates overall

Example with $w=3$:



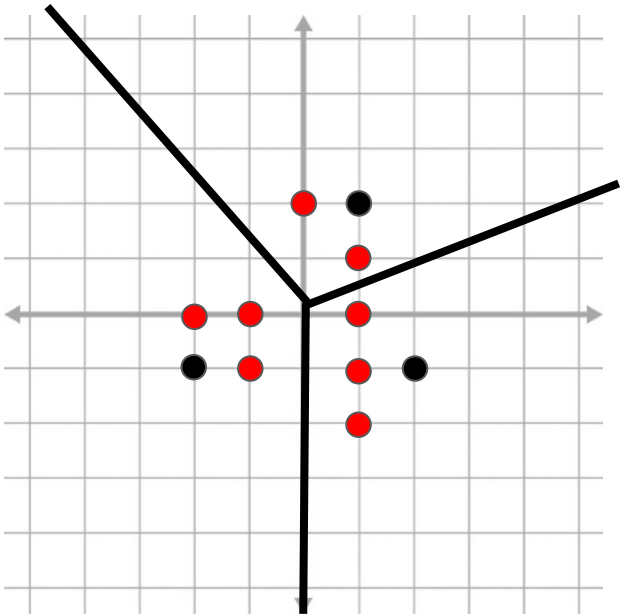
IVFADC Example - Coarse Quantization



y	$q_c(y)$	r(y)
(0,0)	(1, 1)	(-1,-1)
(2,2)	(1, 1)	(1,1)
(2,-2)	(3,-2)	(-1,0)
(4,-2)	(3,-2)	(1,0)
(-1,-1)	(-1,-3)	(0,2)

y	$q_c(y)$	r(y)
(0,-5)	(-1, -3)	(1,-2)
(-2,-3)	(-1, -3)	(-1,0)
(-2,0)	(-3,1)	(1,-1)
(-2,2)	(-3,1)	(1,1)
(-5,1)	(-3,1)	(2,0)

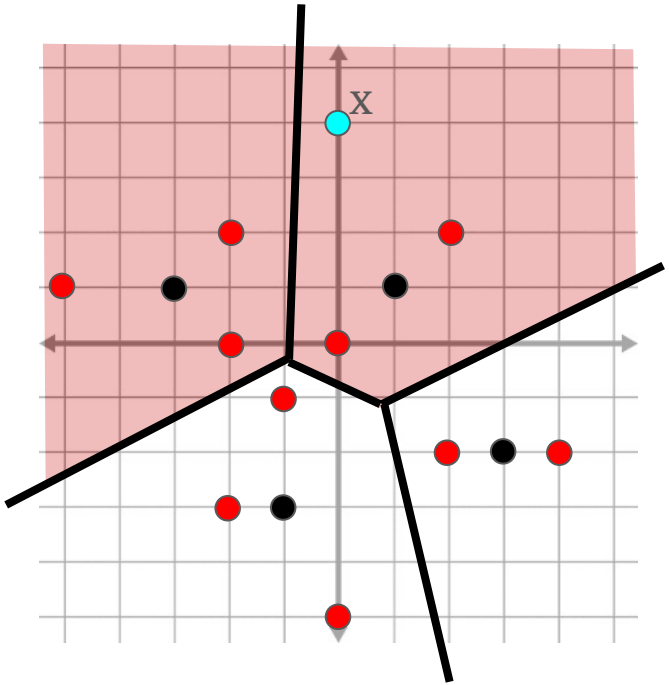
IVFADC Example - Product Quantization



r	$q_p(r)$
(0,2)	(1, 2)
(1,1)	(1, 2)
(1,0)	(2,-1)
(1,-1)	(2,-1)

r	$q_p(r)$
(1,-2)	(2, -1)
(-1,0)	(-2, -1)
(-1,-1)	(-2,-1)
(-2,0)	(-2,-1)

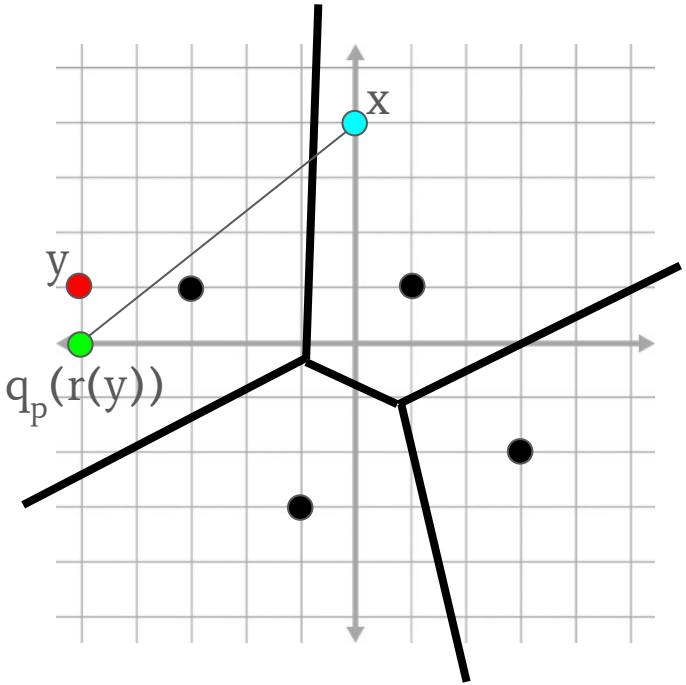
IVFADC Example - Queries



$w = 2$, x is closest to $(1, 1)$ and $(-3, 1)$

So we check all y in those regions

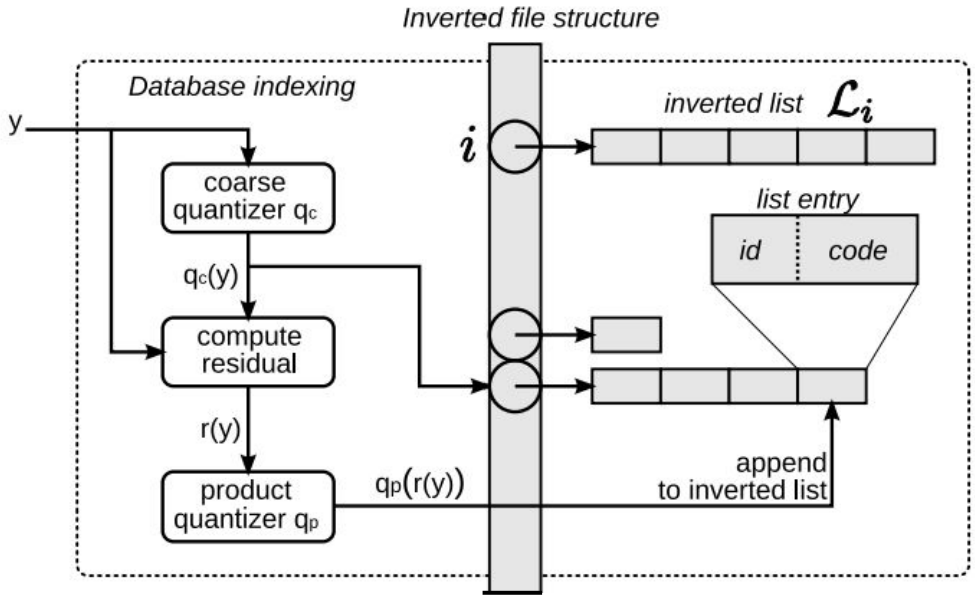
IVFADC Example - Queries



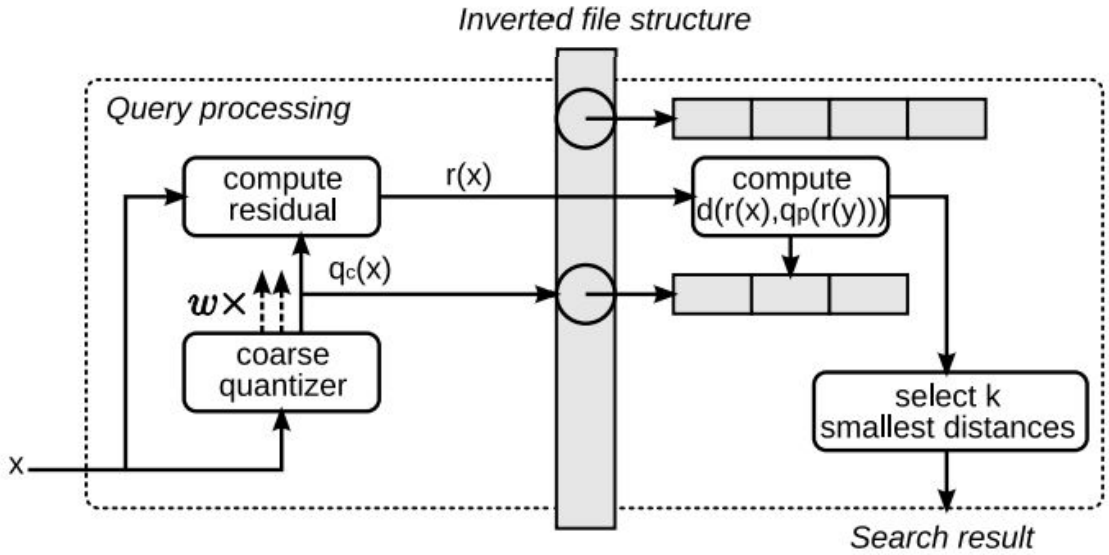
Our estimate of $d(x, y)^2$ is $d(r(x), q_p(r(y)))^2$

Imagine shifting the origin to $q_c(y)$

Visualizing IVFADC indexing

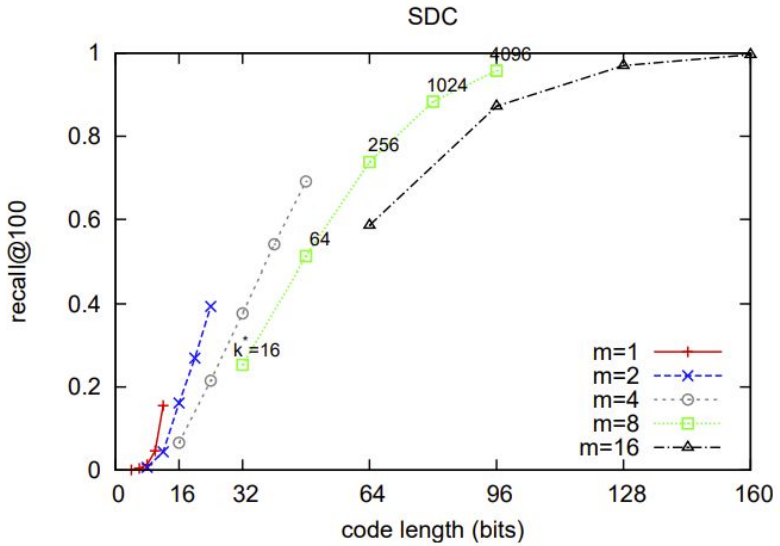


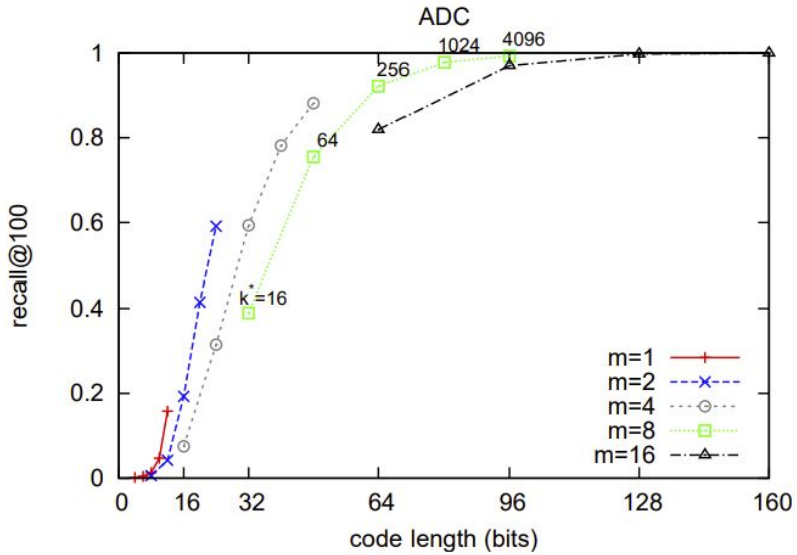
Visualizing IVFADC queries



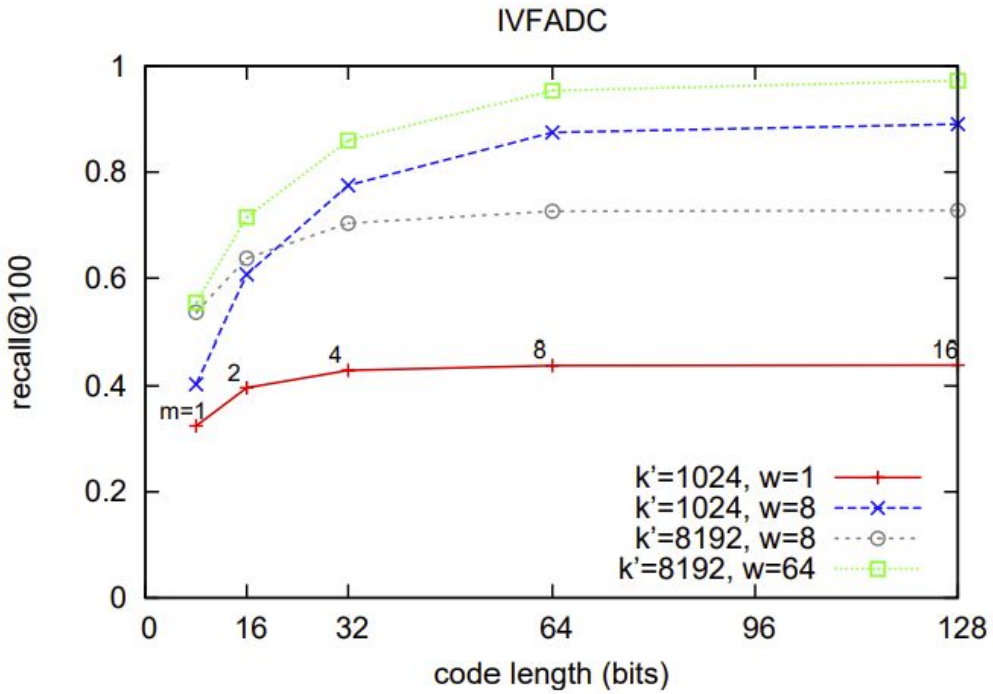
Experimental Results: Recall for SDC/ADC

recall@R is the percentage of queries where the true nearest neighbor was ranked in the top R results from the approximate query

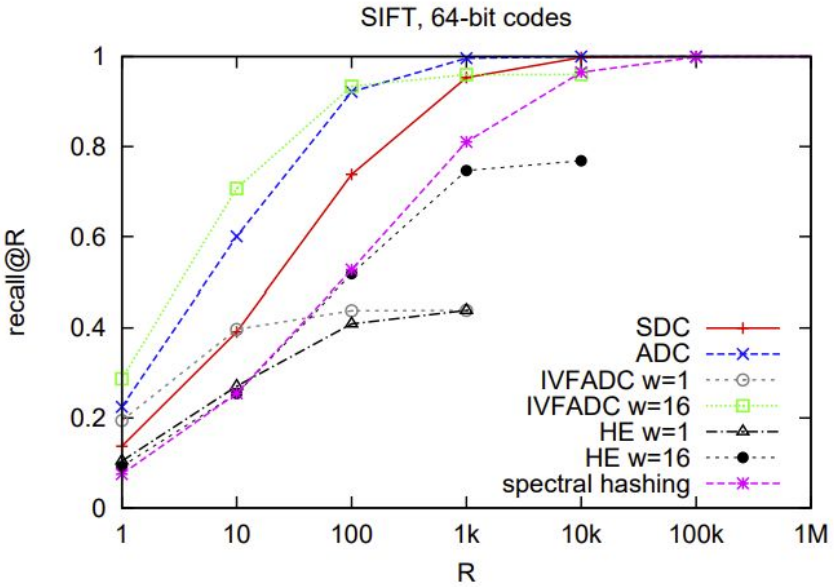


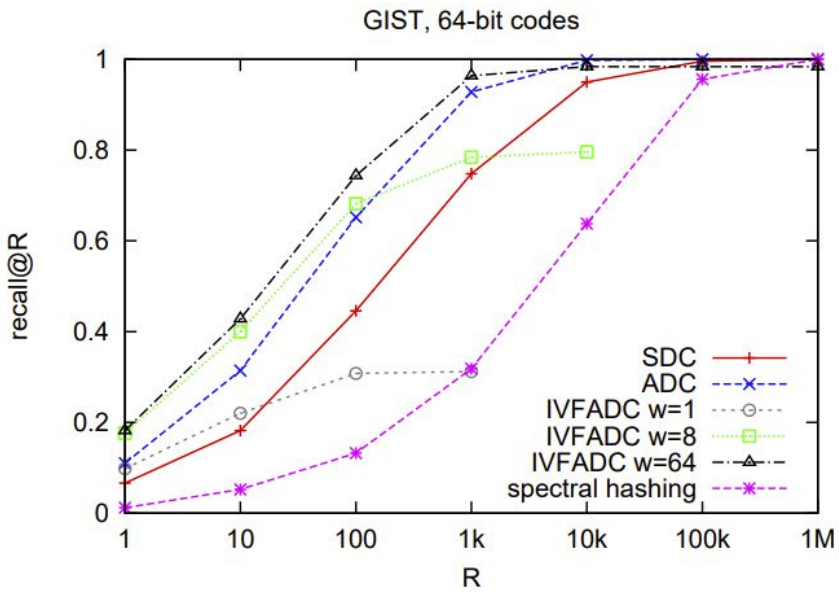


Experimental Results: Recall for IVFADC



Experimental Results: Recall Compared





Experimental Results: Recall + Time (GIST)

method	parameters	search time (ms)	average number of code comparisons	recall@100
SDC		16.8	1 000 991	0.446
ADC		17.2	1 000 991	0.652
IVFADC	k'= 1 024, w=1	1.5	1 947	0.308
	k'= 1 024, w=8	8.8	27 818	0.682
	k'= 1 024, w=64	65.9	101 158	0.744
	k'= 8 192, w=1	3.8	361	0.240
	k'= 8 192, w=8	10.2	2 709	0.516
	k'= 8 192, w=64	65.3	19 101	0.610
SH		22.7	1 000 991	0.132

Conclusion

Product quantization is a powerful tool for increasing the number of centroids with a low memory cost

It can be combined with non-product quantization to obtain the benefits of product quantization without the requirement of exhaustive search

Product quantization, via IVFADC, was competitive with and in many cases more effective than other K-ANN algorithms at the time