

# Optimal hash functions for approximate closest pairs on the $n$ -cube

Daniel Gordon and Victor Miller and Peter Ostapenko

IDA/CCR

January 30, 2009



# Outline

- 1 Introduction
- 2 Optimal Regions and Hash Functions
- 3 Hashing with Projection
- 4 Hashing with Codes
- 5 Computing Optimal Regions



## Closest Pair Problem

Given a set of  $n$ -bit vectors  $v_1, v_2, \dots, v_M$ , find a pair with minimal distance.



## Closest Pair Problem

Given a set of  $n$ -bit vectors  $v_1, v_2, \dots, v_M$ , find a pair with minimal distance.

## Applications

- DNA sequence comparison
- Information retrieval
- *GET MORE EXAMPLES*



# Finding Close Vectors

01101010100110010110110001100101001101100100100110  
01011100010110001001100001100011101001010101100100  
1011100111011110010000001000100000010011100111001  
01100101100001000010101111011000001001011000111000  
11101111000010011101000000000111010111000100110111  
10100100010101011000110011010100101110000011010000  
00001001000111111101011001110110010000111001111011  
00110001011110011101001110100001111001100110011110  
11010010110111010111011110000001011110001111010011  
11011100000110001001100001100010101001010101110100  
10001101000100110000000101101010110100110001001000  
01111011111110111010100010010001010100001000011000  
11000000001010010010111100100000100010100011000001



# Finding Close Vectors

```
01101010100110010110110001100101001101100100100110  
01011100010110001001100001100011101001010101100100  
1011100111011110010000001000100000010011100111001  
0110010110000100001010111011000001001011000111000  
11101111000010011101000000000111010111000100110111  
10100100010101011000110011010100101110000011010000  
00001001000111111101011001110110010000111001111011  
00110001011110011101001110100001111001100110011110  
11010010110111010111011110000001011110001111010011  
11011100000110001001100001100010101001010101110100  
10001101000100110000000101101010110100110001001000  
01111011111110111010100010010001010100001000011000  
11000000001010010010111100100000100010100011000001
```



## Strategy 0: Check Every Pair

For lists of size  $M$ , work is  $O(M^2)$ .

Simple, but this becomes too expensive for large  $M$ .



## Strategy 0: Check Every Pair

```
01101010100110010110110001100101001101100100100110  
01011100010110001001100001100011101001010101100100  
1011100111011110010000001000100000010011100111001  
01100101100001000010101111011000001001011000111000  
11101111000010011101000000000111010111000100110111  
10100100010101011000110011010100101110000011010000  
00001001000111111101011001110110010000111001111011  
00110001011110011101001110100001111001100110011110  
11010010110111010111011110000001011110001111010011  
11011100000110001001100001100010101001010101110100  
10001101000100110000000101101010110100110001001000  
01111011111110111010100010010001010100001000011000  
11000000001010010010111100100000100010100011000001
```



## Strategy 0: Check Every Pair

```
01101010100110010110110001100101001101100100100110  
01011100010110001001100001100011101001010101100100  
1011100111011110010000001000100000010011100111001  
01100101100001000010101111011000001001011000111000  
11101111000010011101000000000111010111000100110111  
10100100010101011000110011010100101110000011010000  
00001001000111111101011001110110010000111001111011  
00110001011110011101001110100001111001100110011110  
11010010110111010111011110000001011110001111010011  
11011100000110001001100001100010101001010101110100  
10001101000100110000000101101010110100110001001000  
01111011111110111010100010010001010100001000011000  
11000000001010010010111100100000100010100011000001
```



## Strategy 0: Check Every Pair

```
01101010100110010110110001100101001101100100100110  
01011100010110001001100001100011101001010101100100  
10111001110111100100000010001000000010011100111001  
01100101100001000010101111011000001001011000111000  
11101111000010011101000000000111010111000100110111  
10100100010101011000110011010100101110000011010000  
00001001000111111101011001110110010000111001111011  
00110001011110011101001110100001111001100110011110  
11010010110111010111011110000001011110001111010011  
11011100000110001001100001100010101001010101110100  
10001101000100110000000101101010110100110001001000  
01111011111110111010100010010001010100001000011000  
11000000001010010010111100100000100010100011000001
```



## Strategy 0: Check Every Pair

```
01101010100110010110110001100101001101100100100110  
01011100010110001001100001100011101001010101100100  
10111001110111100100000010001000000010011100111001  
01100101100001000010101111011000001001011000111000  
11101111000010011101000000000111010111000100110111  
10100100010101011000110011010100101110000011010000  
00001001000111111101011001110110010000111001111011  
00110001011110011101001110100001111001100110011110  
11010010110111010111011110000001011110001111010011  
11011100000110001001100001100010101001010101110100  
10001101000100110000000101101010110100110001001000  
01111011111110111010100010010001010100001000011000  
11000000001010010010111100100000100010100011000001
```



# Strategy 1: Projection

Hash on  $k$  bits, check for collisions.  
If there's an error in those bits, this will fail.



# Strategy 1: Projection

Hash on  $k$  bits, check for collisions.  
If there's an error in those bits, this will fail.

Work per Success:

$$\frac{M \cdot C_{\text{Hash}} + M^2/2^{k+1} \cdot C_{\text{Test}}}{(1 - p^k)}$$



# Strategy 1: Projection

```
01101010100110010110110001100101001101100100100110
01011100010110001001100001100011101001010101100100
1011100111011110010000001000100000010011100111001
01100101100001000010101111011000001001011000111000
1110111100001001110100000000111010111000100110111
10100100010101011000110011010100101110000011010000
00001001000111111101011001110110010000111001111011
00110001011110011101001110100001111001100110011110
11010010110111010111011110000001011110001111010011
11011100000110001001100001100010101001010101110100
10001101000100110000000101101010110100110001001000
0111101111110111010100010010001010100001000011000
11000000001010010010111100100000100010100011000001
```



# Strategy 1: Projection

```
01101010100110010110110001100101001101100100100110
01011100010110001001100001100011101001010101100100
101110011110111100100000010001000000010011100111001
01100101100001000010101111011000001001011000111000
11101111000010011101000000000111010111000100110111
10100100010101011000110011010100101110000011010000
00001001000111111101011001110110010000111001111011
00110001011110011101001110100001111001100110011110
11010010110111010111011110000001011110001111010011
11011100000110001001100001100010101001010101110100
10001101000100110000000101101010110100110001001000
01111011111110111010100010010001010100001000011000
1100000000101001001010111100100000100010100011000001
```



# Strategy 1: Projection

```
01101010100110010110110001100101001101100100100110
01011100010110001001100001100011101001010101100100
1011100111011110010000010001000000010011100111001
0110010110000100001010111011000001001011000111000
1110111100001001110100000000111010111000100110111
101001000101010110011010100101110000011010000
00001001000111111101011001110110010000111001111011
00110001011110011101001110100001111001100110011110
110100101101110101110000001011110001111010011
11011100000110001001100001100010101001010101110100
10001101000100110000000101101010110100110001001000
011110111111101110101000100100100001000011000
1100000000101001001011100100000100010100011000001
```



## Strategy 2: Other Hash Functions

### Alternate Idea

Use a different hash function, such as mapping  $n$  bits to codewords of an  $[n, k]$  error-correcting code.



## Strategy 2: Other Hash Functions

### Alternate Idea

Use a different hash function, such as mapping  $n$  bits to codewords of an  $[n, k]$  error-correcting code.

This uses more bits, but error may not be fatal.

This idea has occurred independently many times, and been patented twice.



Work per Success:

$$(M \cdot C_{\text{Hash}} + M^2/2^{k+1} \cdot C_{\text{Test}})/P^h$$

where

$$P^h = P^h(p) = \text{Prob}(h(\mathbf{v}_i) = h(\mathbf{v}_i + \mathbf{e}))$$



## Work per Success:

$$(M \cdot C_{\text{Hash}} + M^2/2^{k+1} \cdot C_{\text{Test}})/P^h$$

where

$$P^h = P^h(p) = \text{Prob}(h(\mathbf{v}_i) = h(\mathbf{v}_i + \mathbf{e}))$$

## The Big Question

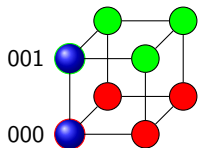
What hash function minimizes work/success?



Example:  $n = 3$ ,  $k = 1$

Project on one bit

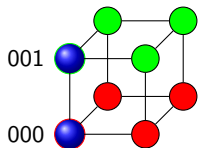
Region  $Q_2$  maps to a point.



Example:  $n = 3, k = 1$

Project on one bit

Region  $Q_2$  maps to a point.



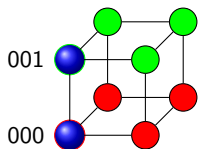
$$P^h = (1 - p)$$



Example:  $n = 3, k = 1$

Project on one bit

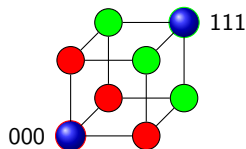
Region  $Q_2$  maps to a point.



$$P^h = (1 - p)$$

Code  $C = \{000, 111\}$

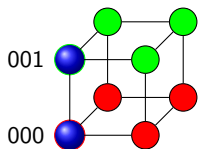
Region  $B_3(1)$  maps to a point.



Example:  $n = 3, k = 1$

Project on one bit

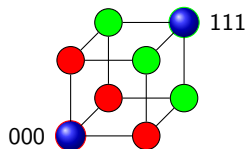
Region  $Q_2$  maps to a point.



$$P^h = (1 - p)$$

Code  $C = \{000, 111\}$

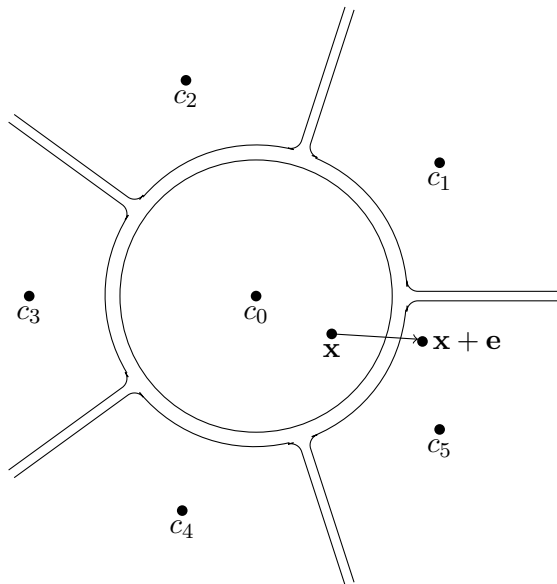
Region  $B_3(1)$  maps to a point.



$$P^h = (1 - p)(1 - p(\frac{1}{2} - p))$$



# Structure of Hamming space around codewords



# Standard Coding Theory vs. Hashing with Codes I

## Coding Theory

Correct codewords with errors.



# Standard Coding Theory vs. Hashing with Codes I

## Coding Theory

Correct codewords with errors.

## Hashing with codes

Correct **anything** with errors.



Let  $\mathcal{S}$  be the points in  $\mathcal{V}$  that hash to  $\mathbf{0}$ .

$h(\mathbf{x}) = h(\mathbf{x} + \mathbf{e})$  with probability

$$P_{\mathcal{S}}(p) = \frac{1}{|\mathcal{S}|} \sum_{\mathbf{x}, \mathbf{y} \in \mathcal{S}} p^{d(\mathbf{x}, \mathbf{y})} (1 - p)^{n - d(\mathbf{x}, \mathbf{y})}.$$



# Optimal Regions

Let  $\mathcal{S}$  be the points in  $\mathcal{V}$  that hash to  $\mathbf{0}$ .

$h(\mathbf{x}) = h(\mathbf{x} + \mathbf{e})$  with probability

$$P_{\mathcal{S}}(p) = \frac{1}{|\mathcal{S}|} \sum_{\mathbf{x}, \mathbf{y} \in \mathcal{S}} p^{d(\mathbf{x}, \mathbf{y})} (1 - p)^{n - d(\mathbf{x}, \mathbf{y})}.$$

## Definition

$\mathcal{S}$  is an optimal region if it maximizes this probability for any region of size  $|\mathcal{S}|$ .



## Definition

If  $\mathcal{S}$  is a code, the *probability of undetected error* is

$$\mathcal{P}(\mathcal{S}, p) = \frac{1}{|\mathcal{S}|} \sum_{\mathbf{x}, \mathbf{y} \in \mathcal{S}} p^{d(\mathbf{x}, \mathbf{y})} (1 - p)^{n - d(\mathbf{x}, \mathbf{y})}.$$



## Definition

If  $\mathcal{S}$  is a code, the *probability of undetected error* is

$$\mathcal{P}(\mathcal{S}, p) = \frac{1}{|\mathcal{S}|} \sum_{\mathbf{x}, \mathbf{y} \in \mathcal{S}} p^{d(\mathbf{x}, \mathbf{y})} (1 - p)^{n - d(\mathbf{x}, \mathbf{y})}.$$

## Coding Theory

$\mathcal{S}$  is a code. Minimize this probability.



## Definition

If  $\mathcal{S}$  is a code, the *probability of undetected error* is

$$\mathcal{P}(\mathcal{S}, p) = \frac{1}{|\mathcal{S}|} \sum_{\mathbf{x}, \mathbf{y} \in \mathcal{S}} p^{d(\mathbf{x}, \mathbf{y})} (1 - p)^{n - d(\mathbf{x}, \mathbf{y})}.$$

## Coding Theory

$\mathcal{S}$  is a code. Minimize this probability.

## Hashing with Codes

$\mathcal{S}$  is the sphere around a codeword. Maximize this probability.

Let

$$A_i = \#\{(\mathbf{x}, \mathbf{y}) : \mathbf{x}, \mathbf{y} \in \mathcal{S} \text{ and } d(\mathbf{x}, \mathbf{y}) = i\}$$

## Distance Distribution Function

$$A(\mathcal{S}, \zeta) := \sum_{i=0}^n A_i \zeta^i$$



Let

$$A_i = \#\{(\mathbf{x}, \mathbf{y}) : \mathbf{x}, \mathbf{y} \in \mathcal{S} \text{ and } d(\mathbf{x}, \mathbf{y}) = i\}$$

## Distance Distribution Function

$$A(\mathcal{S}, \zeta) := \sum_{i=0}^n A_i \zeta^i$$

$$\begin{aligned} P_{\mathcal{S}}(p) &:= \frac{1}{|\mathcal{S}|} \sum_{\mathbf{x}, \mathbf{y} \in \mathcal{S}} p^{d(\mathbf{x}, \mathbf{y})} (1-p)^{n-d(\mathbf{x}, \mathbf{y})} \\ &= \frac{1}{|\mathcal{S}|} \sum_{i=0}^n A_i p^i (1-p)^{n-i} = \frac{(1-p)^n}{|\mathcal{S}|} A\left(\mathcal{S}, \frac{p}{1-p}\right). \end{aligned}$$

## Project $x$ onto $k$ coordinates

- $\mathcal{S}$  is an  $n - k$  subcube.
- DD function is  $A(\mathcal{S}, \zeta) = (2(1 + \zeta))^{n-k}$
- Probability of collision is

$$\begin{aligned} P^{\mathcal{P}_{n,k}}(p) &= \frac{(1-p)^n}{2^{n-k}} \left( \frac{2}{1-p} \right)^{n-k} \\ &= (1-p)^k. \end{aligned}$$



## Projection $\mathcal{P}_{n,k}$ (cont'd)

For small error rates, projection is optimal:

### Theorem

Let  $\mathcal{S}$  be the  $2^{n-k}$ -subcube of  $\mathcal{V}$ . For any error rate  $p \in (0, 2^{-2(n-k)})$ ,  $\mathcal{S}$  is an optimal region, and so  $k$ -projection is an optimal hash.



## Perfect Codes

A code is **perfect** if every vertex is distance  $\leq e$  from *exactly* one codeword.

## Perfect Binary Codes

- $[n, n, 1]$  Repetition Codes
- $[2^m - 1, 2^m - m - 1, 3]$  Hamming Codes  $\mathcal{H}_m$
- $[23, 12, 7]$  binary Golay Code  $\mathcal{G}$



$\mathcal{S} = 3$ -sphere

The 3-sphere's DD function is

$$2048 + 11684\zeta + 128524\zeta^2 + 226688\zeta^3 \\ + 1133440\zeta^4 + 672980\zeta^5 + 2018940\zeta^6.$$



$\mathcal{S} = 3$ -sphere

The 3-sphere's DD function is

$$2048 + 11684\zeta + 128524\zeta^2 + 226688\zeta^3 \\ + 1133440\zeta^4 + 672980\zeta^5 + 2018940\zeta^6.$$

Corollary

This beats projection  $\mathcal{P}_{23,12}$  for  $p > 0.2555$ .



$\mathcal{S} = 1$  - sphere

The 1-sphere's DD function is

$$2^m + 2(2^m - 1)\zeta + (2^m - 1)(2^m - 2)\zeta^2,$$



$\mathcal{S} = 1$  - sphere

The 1-sphere's DD function is

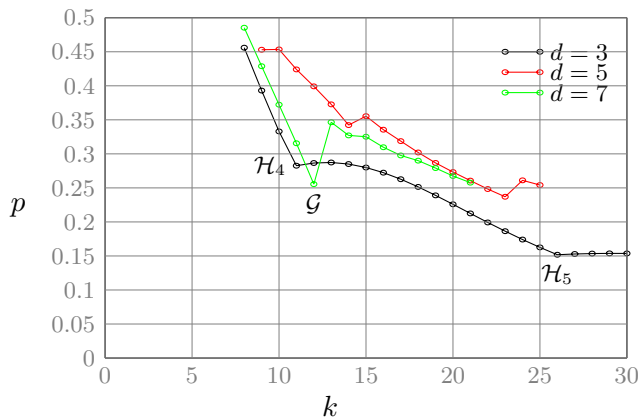
$$2^m + 2(2^m - 1)\zeta + (2^m - 1)(2^m - 2)\zeta^2,$$

Corollary

This beats projection for  $m \geq 4$  and  $p > \alpha_m \approx (m - 2)/2^m$



# Other Linear Codes



## Alternate Formulation

What region of size  $2^t$  in  $\mathbb{F}_{2^n}$  has the best  $P(p)$ ?



## Alternate Formulation

What region of size  $2^t$  in  $\mathbb{F}_{2^n}$  has the best  $P(p)$ ?

## Previous Results

- $2^{n-1}$ -subcube is optimal for all  $n, p$ .
- $2^t$ -subcube is optimal for  $t \leq 3$  for all  $n, p$ .
- A subcube is optimal for any  $t, n$  if  $p$  is small enough.



# Structure of Optimal Regions

## Definition

For  $\mathbf{x} = (x_1, \dots, x_n) \in \mathcal{V}$ , let

$$\rho_i(\mathbf{x}) := (x_1, x_2, \dots, x_{i-1}, 0, x_{i+1}, \dots, x_n)$$

and

$$\sigma_{ij}(\mathbf{x}) := (x_1, \dots, \min(x_i, x_j), \dots, \max(x_i, x_j), \dots, x_n).$$



# Structure of Optimal Regions

## Definition

For  $\mathbf{x} = (x_1, \dots, x_n) \in \mathcal{V}$ , let

$$\rho_i(\mathbf{x}) := (x_1, x_2, \dots, x_{i-1}, 0, x_{i+1}, \dots, x_n)$$

and

$$\sigma_{ij}(\mathbf{x}) := (x_1, \dots, \min(x_i, x_j), \dots, \max(x_i, x_j), \dots, x_n).$$

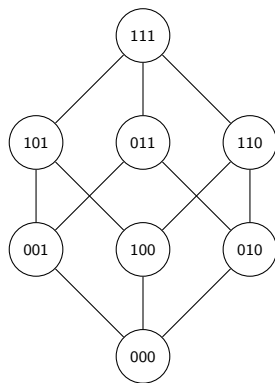
## Definition

A set  $\mathcal{S} \subset \mathcal{V}$  is a *down-set* if  $\rho_i(\mathcal{S}) \subset \mathcal{S}$  for all  $i \leq n$ .

## Definition

A set  $\mathcal{S} \subset \mathcal{V}$  is *right-shifted* if  $\sigma_{ij}(\mathcal{S}) \subset \mathcal{S}$  for all  $i, j \leq n$ .

## Structure of Optimal Regions (cont'd)



## Theorem

If a set  $\mathcal{S}$  is optimal, then it is isomorphic to a right-shifted down-set.



## Theorem

If a set  $\mathcal{S}$  is optimal, then it is isomorphic to a right-shifted down-set.

## Computing Right-shifted Downsets

- We may find all right-shifted downsets, and look for optimal regions.
- For size 64, there are 4384627.
- We have compiled tables of optimal regions of up to size 64.
- Unfortunately, they don't tile the cube.



We would expect that for large  $n$ , a random code would do well.



We would expect that for large  $n$ , a random code would do well.

## Theorem

For a fixed error rate  $p \in (0, 1/2)$ , rate  $R = k/n$ , and  $n$  sufficiently large, a random code of rate  $R$  will beat projection.

