BMI/CS 776 Lecture #18 Pattern matching - Locality-sensitive hashing

Colin Dewey

March 27, 2008

Exact vs inexact matching

- Exact matching
  - Good for highly-similar sequences, or for locating highly-conserved short substrings
  - As sequences diverge, exact matching must use shorter seeds, resulting in low specificity
  - Can be done very efficiently - suffix trees/arrays
- Inexact matching
  - Better for diverged sequences
  - Much harder than exact matching

Inexact matching problem

today

- Given a set S of sequences
- Find all pairs of d-mers that differ in at most r positions
- Or, find all pairs of d-mers that have edit distance less than ε

Locality-sensitive hashing

- Problem:
  - Given set of high-dimensional data points
  - Want to find all similar points, or find closest points to a given query point
- Locality-sensitive hashing (Indyk & Motwani, 1998):
  - Hashing scheme with similar points more likely to hash to the same bin
  - Randomized algorithm

(r, ε)-Neighbor problem

- Given:
  - P: set of elements from set S
  - D: distance function on set S
  - q: query element
- Determine whether:
  - exists p in P such that $D(q,p) \leq r$
    - return a point p' such that $D(q, p') < r(1 + \varepsilon)$
  - or all p in P have $D(q,p) \geq r(1 + \varepsilon)$

(r1,r2,p1,p2)-sensitive hash families

Definition 1 *A family $\mathcal{H}$ of functions $h : S \to U$ is (r_1, r_2, p_1, p_2)-sensitive for $D(\cdot,\cdot)$ if $\forall p, q \in S$*

1. *if $p \in \mathcal{B}(q, r_1)$ then $\mathbb{P}_{\mathcal{H}}[h(q) = h(p)] \geq p_1$*
2. *if $p \notin \mathcal{B}(q, r_2)$ then $\mathbb{P}_{\mathcal{H}}[h(q) = h(p)] \leq p_2$*

where $\mathcal{B}(q, r) = \{p : D(p, q) \leq r\}$

- Useful families have $p_1 > p_2$ and $r_1 < r_2$
- The closer the points, the higher the chance of collision via the hash function

(r1,r2,p1,p2)-sensitive hash family example

$$S = H^{d'} \text{ (d'-dimensional Hamming cube)}$$

$$D(p,q) = d_H(p,q) \text{ (Hamming distance)}$$

$$\mathcal{H}_{d'} = \{h_i : h_i((b_1, \ldots, b_{d'})) = b_i, \text{for } i = 1, \ldots, d'\}$$

$$\mathcal{H}_{d'} \text{ is } \left(r, r(1+\epsilon), 1 - \frac{r}{d'}, 1 - \frac{r(1+\epsilon)}{d'}\right)\text{-sensitive, } \forall r, \epsilon$$

LSH functions

Choose l functions $g_1, \ldots, g_l$, where g are of the form:

$$g_i(p) = (h_{i_1}(p), h_{i_2}(p), \ldots, h_{i_k}(p))$$

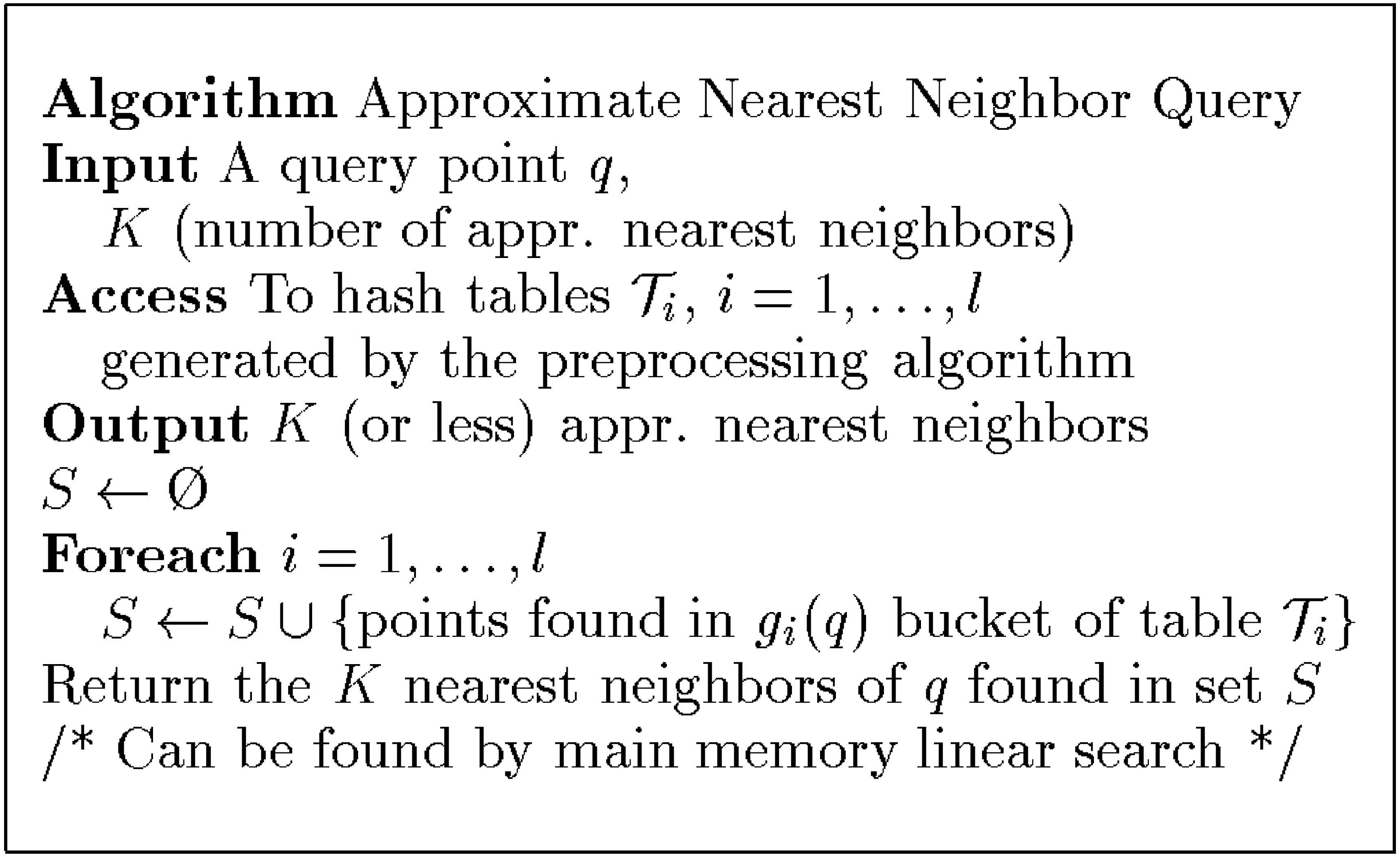
where $h_{i_1}, \ldots, h_{i_k}$ chosen at random from $\mathcal{H}$ with replacement

LSH preprocessing

```
**Algorithm** Preprocessing
**Input** A set of points $P$,
  $l$ (number of hash tables),
**Output** Hash tables $\mathcal{T}_i$, $i = 1, \ldots, l$
**Foreach** $i = 1, \ldots, l$
  Initialize hash table $\mathcal{T}_i$ by generating
  a random hash function $g_i(\cdot)$
**Foreach** $i = 1, \ldots, l$
  **Foreach** $j = 1, \ldots, n$
    Store point $p_j$ on bucket $g_i(p_j)$ of hash table $\mathcal{T}_i$
```

(Gionis, 1999)

LSH approximate nearest neighbor



```
**Algorithm** Approximate Nearest Neighbor Query
**Input** A query point $q$,
    $K$ (number of appr. nearest neighbors)
**Access** To hash tables $\mathcal{T}_i$, $i = 1, \ldots, l$
    generated by the preprocessing algorithm
**Output** $K$ (or less) appr. nearest neighbors
$S \leftarrow \emptyset$
**Foreach** $i = 1, \ldots, l$
    $S \leftarrow S \cup \{$points found in $g_i(q)$ bucket of table $\mathcal{T}_i\}$
Return the $K$ nearest neighbors of $q$ found in set $S$
/* Can be found by main memory linear search */
```

(Gionis, 1999)

LSH (r, ε)-Neighbor correctness conditions

$$P' = \{p' : p' \in P, d(q, p') > r_2 = r(1+\epsilon)\}$$

LSH algorithm solves (r, ϵ)-Neighbor problem if both:

P1 If there exists p^* s.t. $p^* \in \mathcal{B}(q, r_1)$, then $g_j(p^*) = g_j(q)$ for some $j = 1, \ldots, l$

P2 The total number of hash table blocks referenced by q and containing only points from P' is less than cl, for some constant c.

LSH (r, ε)-Neighbor correctness

Theorem 1 *For a (r_1, r_2, p_1, p_2)-sensitive family $\mathcal{H}$, if we set $\rho = \frac{\ln 1/p_1}{\ln 1/p2}$, $k = \log_{1/p_2}(n/B)$ and $l = (\frac{n}{B})^\rho$, then* **P1** *and* **P2** *hold with probability at least* $\frac{1}{2} - \frac{1}{e} > 0.132$

"constant probability" - does not change with input size n

For proof, see (Gionis, 1999)

Randomized algorithms

- Given an algorithm A_1 that succeeds with probability p_1
- Algorithm A_2, which runs A_1 t times, succeeds with probability $p_2 = 1 - (1 - p_1)^t$
- Can make p_2 as big as we like
- For $t > 32$, LSH (r, ε)-Neighbor succeeds with probability > 0.99

Complexity results

- LSH (r, ε)-Neighbor used to solve ε-Nearest Neighbor Search (ε-NNS) problem
- $O(dn^{1/(1+\varepsilon)})$ query time (sublinear!) for all ε
- $O(n^{1+1/(1+\varepsilon)} + nd)$ preprocessing time

LSH for sequence comparison

- Buhler, 2001
- Points are d-mers over some alphabet
- Comparing all d-mers at once, not just one query d-mer against all others
- Hash function *f*:
    - pick *k* indices $i_1, \ldots, i_k$ from {1,...,d}
    - $f(s) = (s[i_1], s[i_2], \ldots, s[i_k])$

(r1,r2,p1,p2)-sensitive property

- If s_1 and s_2 differ by at most $r_1 = r$ positions then,

$$\mathbb{P}[f(s_1) = f(s_2)] \geq p_1 = \left(1 - \frac{r}{d}\right)^k$$

- If not, s_1 and s_2 differ by at least $r_2 = r + 1$ positions and

$$\mathbb{P}[f(s_1) = f(s_2)] \leq p_2 = \left(1 - \frac{r+1}{d}\right)^k$$

LSH-ALL-PAIRS

- Input: Set *C* of sequences of total length *N*
- Output: All pairs of *d*-mers that differ by no more than *r* substitutions
- Algorithm: Iterate ℓ times:
  - Choose random LSH function (choose *k* indices)
  - Partition *d*-mers by hash value
  - In each partition, compare all *d*-mers, output those that differ in no more than *r* positions

False-negative rate

- Typically set ℓ and *k* such that expected false negative rate is sufficiently small (e.g., 0.05)

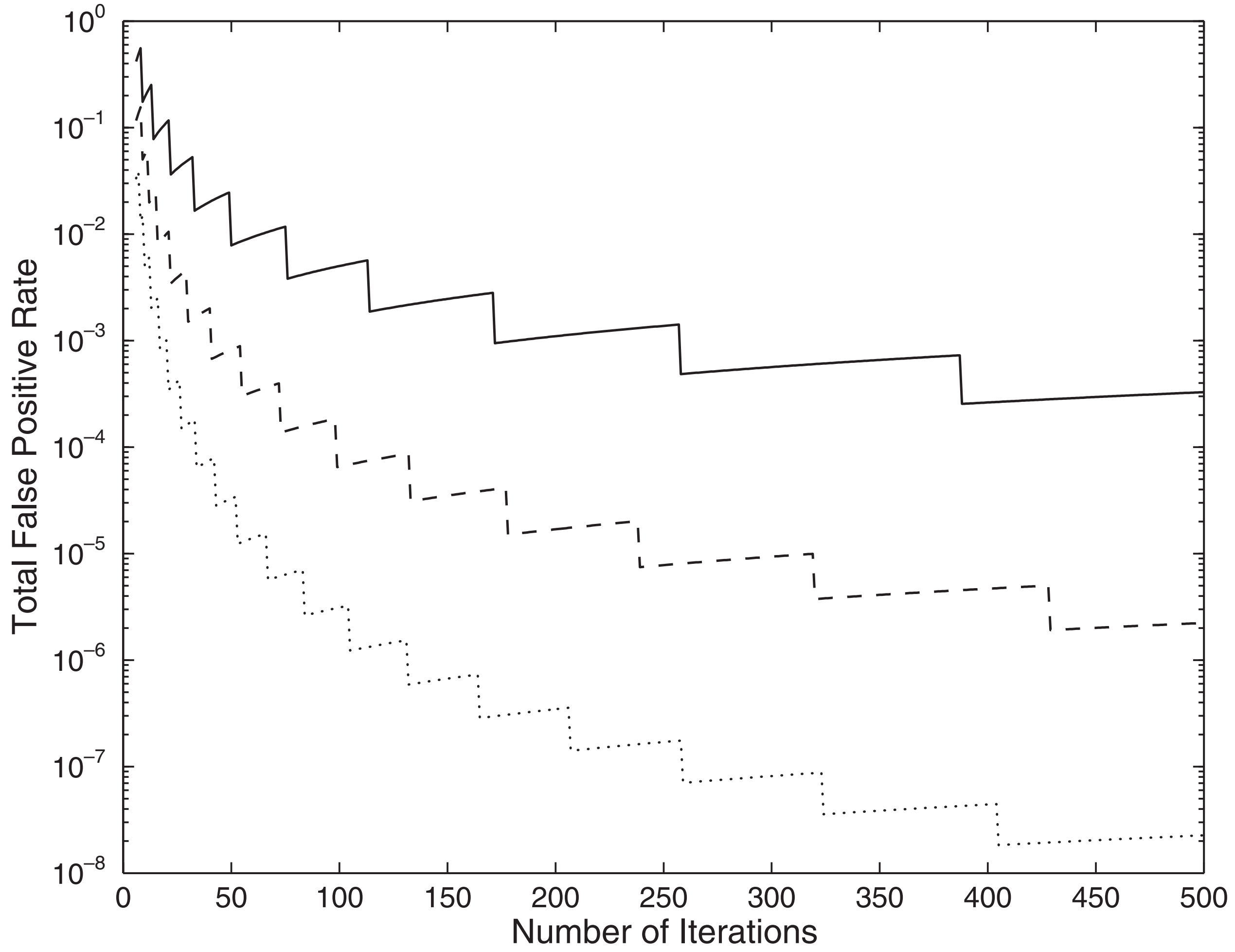
$$\rho_{fn} \le \left[1 - \left(1 - \frac{r}{d}\right)^k\right]^\ell$$

$$k \le \frac{\log\left(1 - \rho_{fn}^{1/\ell}\right)}{\log\left(1 - \frac{r}{d}\right)}$$

False positive rate

- False positive rate: fraction of *d*-mers that we compare (because they hash to the same value) that are not similar enough
- For two unrelated random d-mers, assume chance of match at any position is Φ
- Chance that unrelated d-mers differ by t substitutions: $$\beta_{1-\phi,d}(t) = \binom{d}{t}(1-\phi)^t\phi^{d-t}$$
- False positive rate: $$\rho_{fp} = \ell \sum_{t=r+1}^{d} \beta_{1-\phi,d}(t)\left(1-\frac{t}{d}\right)^k$$

Tradeoffs



Fixed $\rho_{fn} = 0.05$ (i.e. k is changing). $d = 75$.

Curves for three values of r (25, 19, 15)

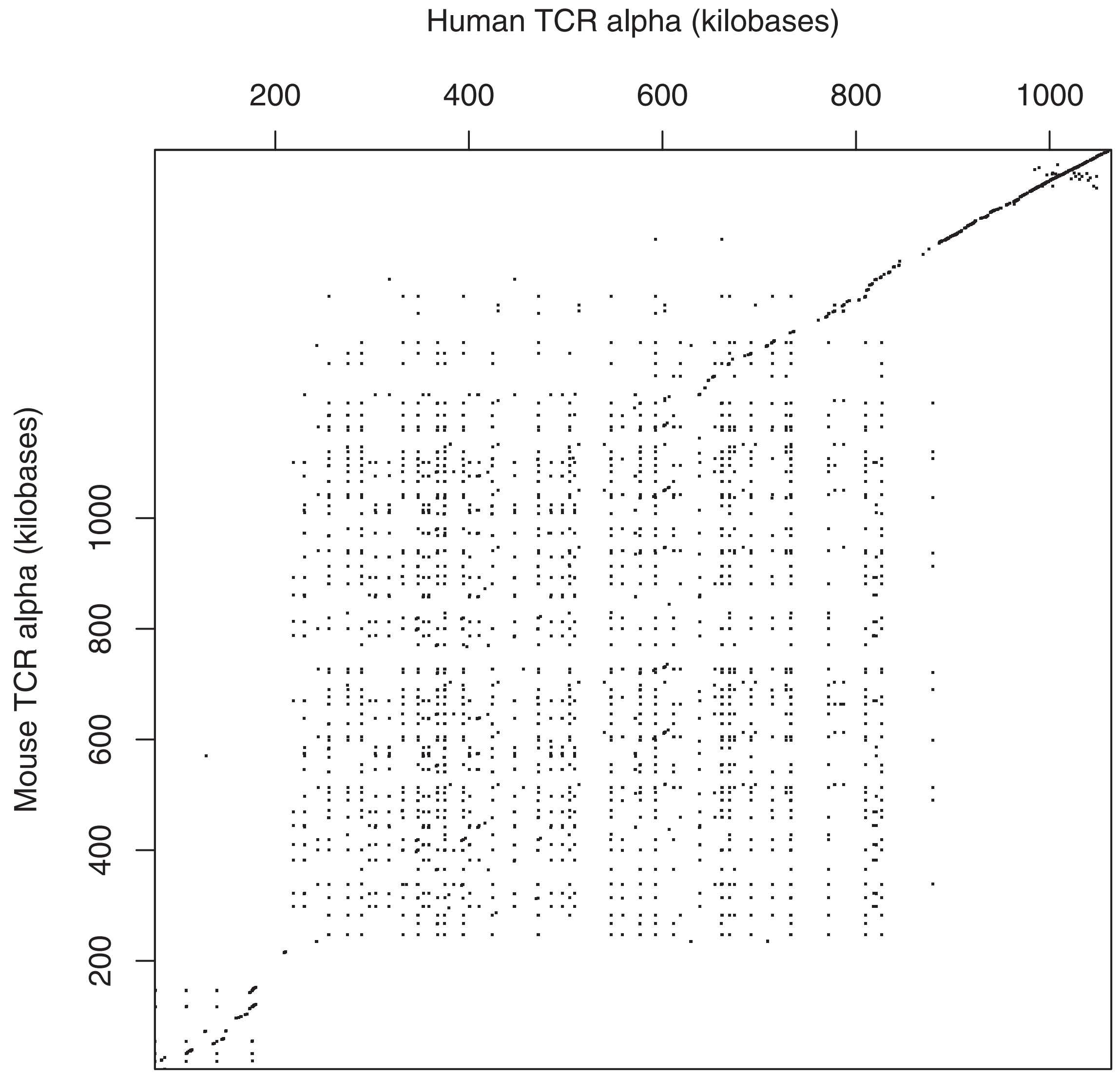
Buhler, 2001

Running time

$$\underbrace{O(\ell k N)}_{\text{hashing/partitioning time}} + \underbrace{O(\rho_{fp} d N^2)}_{\text{comparison time}}$$

- Trick is to balance the two terms
  - ρ_{fp} decreases with increasing *k*
  - *k* depends on ℓ

Testing



Buhler, 2001