Searching for Pattern in Text

Goal: preprocess $T$ such that given $P$ one can quickly find one (all) occurrence(s) of $P$ in $T$

- Exact
- Approximate (occurrence of $P$ with some errors)

Locality Sensitive Hashing

A very popular solution (practical!)

- **Preprocessing**: hash database using a special family of hash functions
- **Query time**: hash query, return closest point in a bucket

The Paris Kanellakis Theory and Practice Award 2012

Andrei Broder
Google

Moses Charikar
Princeton

Piotr Indyk
MIT
Searching for Pattern in Text

Goal: preprocess $T$ such that given $P$ one can quickly find one (all) occurrence(s) of $P$ in $T$

- Case 0: known pattern length $m$
- Case 1: arbitrary pattern length $m$
- Case 2: $P$ has some errors

Case 0: Known pattern length

- Suppose that $m$ is fixed in advance. How do we preprocess $T$ so that we can quickly find $P$ in $T$?
- Hashing!
  - Each $m$-length substring $T_i$ of $T$ is an $m$-digit number
  - Build a hashing data structure on $T_1…T_n$
  - Given $P$, it suffices to evaluate $h(P)$ and lookup the answer
  - How quickly can we perform the lookup?
  - $O(m)$ time, using dot product hashing from before

Case 1: Unknown pattern length

- Suppose that $m$ is fixed in advance. How do we preprocess $T$ so that we can quickly find $P$ in $T$?
- Hashing!
  - Each $m$-length substring $T_i$ of $T$ is an $m$-digit number
  - Build a hashing data structure on $T_1…T_n$
  - Given $P$, it suffices to evaluate $h(P)$ and lookup the answer
  - How quickly can we perform the lookup?
  - $O(m)$ time, using dot product hashing from before
Definition: Suffix tree $ST$ for text $T$ (of length $n$)
- Rooted, directed tree $ST$, $n$ leaves, numbered $1..n$
- Path to leaf $i$ spells out the suffix $T[i..n]$, by concatenating edge labels
- Common prefixes share common paths, diverge to form internal nodes
**Suffix tree: definition**

\( T = \text{"xabxac"} \)

- **Common prefixes share common paths, diverge to form internal nodes**
- **Path to leaf**
- **Rooted, directed tree**

• **Definition:** Suffix tree \( ST \) for text \( T \) (of length \( n \))
  - Rooted, directed tree \( ST \), \( n \) leaves, numbered \( 1..n \)
  - Path to leaf \( i \) spells out the suffix \( T[i..] \), by concatenating edge labels
  - Common prefixes share common paths, diverge to form internal nodes

**Suffix tree: storage**

\( T = \text{"xabxac"} \)

• **How much space do we need to represent it?**
  - Only \( O(n) \)!
    - Each path corresponds to a substring of \( T \)
    - \( \Rightarrow \) we can represent it as a pair \([i..j]\)
    - We get binary tree with \( n \) leaves \( \Rightarrow O(n) \) nodes total

- **Definition:** Suffix tree \( ST \) for text \( T \) (of length \( n \))
  - Rooted, directed tree \( ST \), \( n \) leaves, numbered \( 1..n \)
  - Path to leaf \( i \) spells out the suffix \( T[i..] \), by concatenating edge labels
  - Common prefixes share common paths, diverge to form internal nodes
Exact string matching with suffix trees

- Given the suffix tree for text $T$
- Search pattern $P$ in $O(m)$ time
  - For every character in $P$, traverse the appropriate path of the tree, reading one character each time
  - If $P$ is not found in a path, $P$ does not occur in $T$
  - If $P$ is found in its entirety, then all occurrences of $P$ in $T$ are exactly the children of that node
    - Every child corresponds to exactly one occurrence
    - Simply list each of the leaf indices

Examples:

$P = \ldots x a b \ldots$
$P = \ldots x u g \ldots$

Case 2: Known pattern length but mismatches

Hamming distance $= 2$

- Suppose that we want to tolerate (about) $r$ mismatches
- What should we do?

Locality-Sensitive Hashing

- Idea: construct hash functions $g$ such that for any strings $p,q$:
  - If $D(p,q) \leq r$, then $\Pr[g(p) = g(q)]$ is “high”
  - If $D(p,q) > c r$, then $\Pr[g(p) = g(q)]$ is “small”
  - $c$ is an approximation factor
- Then we can solve the problem by hashing

Approximate Near Neighbor

- $c$-Approximate $r$-Near Neighbor: build data structure which, for any query $q$:
  - If there is a point $p \in P$, $D(p,q) \leq r$ then it returns $p' \in P$, $D(p',q) \leq cr$
- Algorithm is randomized
A family $H$ of functions $h$ is called $(P_1, P_2, r, c, r)$-sensitive, if for any $p, q$:
- If $D(p, q) < r$ then $Pr[h \{ h(p) = h(q) \}] > P_1$
- If $D(p, q) > c r$ then $Pr[h \{ h(p) = h(q) \}] < P_2$

Can we design LSH for Hamming distance such that $P_1 > P_2$?

Use hash functions of the form $h_i(p) = p_i$, i.e., sample the $i$-th bit of $p$.

Probabilities:
$$Pr[h(p) = h(q)] = 1 - D(p, q)/m$$
$$P_1 = 1 - r/m > 1 - c r/m = P_2$$

Analysis

Lemma 1: the algorithm solves $c$-approximate NN with:
- Number of hash functions:
  $$L = C n^\rho, \rho = \log(1/P1)/\log(1/P2)$$
  ($C = C(P1, P2)$ is a constant for $P1$ bounded away from 0)
- Constant success probability per query $q$

Lemma 2: for Hamming LSH functions, we have $\rho = 1/c$

Corollary: search time $O(m n^{1/c})$ for Hamming

Algorithm

We use functions of the form $g(p) = \langle h_{i1}(p), h_{i2}(p), \ldots, h_{ik}(p) \rangle$

I.e., sample $k$ bits at random

Preprocessing:
- Select $g_1 \ldots g_L$
- For all $p \in \mathbb{P}$, hash $p$ to buckets $g_1(p) \ldots g_L(p)$

Query:
- Retrieve the points from buckets $g_1(q), g_2(q), \ldots$, until
  - Either the points from all $L$ buckets have been retrieved, or
  - Total number of points retrieved exceeds $3L$
- Answer the query based on the retrieved points
- Total time: $O(mL)$

Proof of Lemma 1 by picture

Points in $\{0, 1\}^m$

Collision prob. for $k=1..3$, $L=1..3$ (recall: $L = \#indices, k = \#h's$)

Distance ranges from 0 to $m=10$

![Graph showing collision probability for different $L$ values](image)
Proof

• Define:
  – $p$: a point such that $||p-q|| \leq r$
  – $\text{FAR}(q) = \{ p' \in \mathbb{P} : ||p'-q|| > cr \}$
  – $B_i(q) = \{ p' \in \mathbb{P} : g_i(p') = g_i(q) \}$

• Will show that both events occur with $>0$ probability:
  – $E_1$: $g_i(p) = g_i(q)$ for some $i=1\ldots L$
  – $E_2$: $\Sigma_i |B_i(q) \cap \text{FAR}(q)| < 3L$

Proof, ctd.

• Set $k = \text{ceil}(\log_{1/p_2} n)$
• For $p' \in \text{FAR}(q)$,
  $$\Pr[g_i(p') = g_i(q)] \leq P_2^k \leq 1/n$$
• $\mathbb{E}[|B_i(q) \cap \text{FAR}(q)|] \leq 1$
• $\mathbb{E}[\Sigma_i |B_i(q) \cap \text{FAR}(q)|] \leq L$
• $\Pr[\Sigma_i |B_i(q) \cap \text{FAR}(q)| \geq 3L] \leq 1/3$

Proof, end

• $\Pr[ g_i(p) = g_i(q) ] \geq 1/P_1^k \geq P_1^{-\log_{1/p_2}(n)+1}$
  $ \geq P_1/n^p = 1/L$
• $\Pr[ g_i(p) \neq g_i(q) , i=1..L ] \leq (1-1/L)^L \leq 1/e$

• $\Pr[ E_1 \text{ not true} ] + \Pr[ E_2 \text{ not true} ] \leq 1/3 + 1/e = 0.7012.$
• $\Pr[ E_1 \cap E_2 ] \geq 1-(1/3+1/e) \approx 0.3$
Proof of Lemma 2

• Statement: for
  – \( P_1 = 1 - \frac{r}{d} \)
  – \( P_2 = 1 - \frac{cr}{d} \)
  we have \( \rho = \frac{\log(P_1)}{\log(P_2)} \leq \frac{1}{c} \)

• Proof:
  – Need \( P_1^c \geq P_2 \)
  – But \( (1-x)^c \geq (1-cx) \) for any \( 1 > x > 0, \ c > 1 \)