Sequence Nearest Neighbors
and
Block Edit Operations

S. Cenk Sahinalp
SFU
Sequence Nearest Neighbors

- **Input**: Set of sequences $S = \{S_1, S_2, \ldots, S_n\}$; from alphabet \{A, C, T, G\}

- **Output**: Data structure $D(S)$ which can compute the “nearest” $S_i$ to any on-line query sequence $Q$.

- **Nearest $S_i$**: $e(Q, S_i) \leq e(Q, S_j)$ for all $S_j$ in $S$;

- **$e(Q, R)$**: minimum number of “edit operations” to transform $Q$ to $R$. 
SNN in Computational Biology

- "Similar" biomolecular sequences (DNA, RNA, protein) are usually homologous:
  e.g. cancer causing gene ~ gene involved in normal growth
- Evolution reuses, duplicates, modifies successful structures: duplication with modification seems to be the main driving tool of protein evolution
- Genomic sequence archives (e.g. NCBI’s GenBank) compare each query sequence against every sequence in the archive (via dynamic programming or heuristical pruning e.g. BLAST, FASTA). Too slow for exponentially growing data. No tool for indexing sequences
Edit Operations of BioMolecular Interest

Character edits

- Insertion of a character: \( \ldots \text{AATG}\ldots \) to \( \ldots \text{AAC} \text{TG}\ldots \)
- Deletion of a character: \( \ldots \text{AA} \text{TG}\ldots \) to \( \ldots \text{AAG}\ldots \)
- Replacement of a character: \( \ldots \text{AA} \text{TG}\ldots \) to \( \ldots \text{AAG} \text{CG}\ldots \)

Block edits

- Copying a block: \( \ldots \text{CAATG}\ldots \) to \( \ldots \text{CAATG} \text{GAAT}\ldots \)
- Swapping blocks: \( \ldots \text{CAATG}\ldots \) to \( \ldots \text{TG} \text{CAGA}\ldots \)
- Removing one of two identical blocks: \( \ldots \text{CAAT} \text{CAGA}\ldots \) to \( \ldots \text{CAAT}\ldots \)
- Reversing a block: \( \ldots \text{ACGT}\ldots \) to \( \ldots \text{TGCA}\ldots \)
SNN Goals

Construction time: $O(\text{poly} (\Sigma |S_j|))$
Space: $O(\text{poly} (\Sigma |S_j|))$
Query time: $O(\text{poly} (|Q|) \cdot \text{polylog} (\Sigma |S_j|))$

Curse of dimensionality:

All known approaches are either exponential in query time or exponential in construction time and space.
Our Results: Approximate SNN

Construction time: \( O(\text{poly } (\sum |S_j|)) \)

Space: \( O(\text{poly } (\sum |S_j|)) \)

Query time: \( O(\text{poly } (|Q|).\text{polylog}(\sum |S_j|)) \)

If D returns \( S_i \) for \( Q \), we guarantee (whp) for all \( S_j \):

\[
e(S_i, Q) \leq \phi(d).e(S_j, Q)
\]

\[
d = \max(|S_k|)
\]

\[
\phi(d) = O(\log d .(\log^*d)^2)
\]

e(R,Q): minimum number of block and character edits.
Hardness of Block Edit Distance Computation

• It is NP-hard to compute any kind of edit distance that involves copying and/or deletion of blocks [LT96].

• Our scheme gives near logarithmic approximation.
Roadmap

- Transformation from Block Edit Distance to Hamming Distance
- Transformation from Hamming Distance to Prefix Distance
- Data structures for Prefix Distance
Distance Measures

Distance/similarity measures between sequences Q and R:

- **Prefix distance:**
  
  \[ p(Q,R) = |Q| - |lp(Q,R)| \]
  
  (lp: longest common prefix)

- **Hamming distance:**
  
  \[ h(Q,R) = \sum(Q_i \neq R_i) \]

- **Character edit distance:**
  
  \[ c(Q,R) = \text{minimum } \# \text{ character edits} \]

- **Block edit distance:**
  
  \[ e(Q,R) = \text{minimum } \# \text{ character/block edits} \]
Transforming Block Edit Distance to Hamming Distance

**Strategy:** Given Q, R, create bit-strings T(Q) and T(R) so that the Hamming Distance $h(T(Q), T(R))$ approximates the Block Edit Distance $e(Q, R)$.

Each bit in T(Q) corresponds to whether a particular core block (or core) exists in Q or not:
- If the bit is 1, its corresponding core is present in Q
- If the bit is 0, its corresponding core is **not** present in Q
- The cores arise from LCP-Locally Consistent Parsing [SV96]
Restricted Transformation

Let Q be a binary sequence of length $2^k$. Construct $T'(Q)$:

*Location based cores:* all $2^i$ bit blocks B that occur at $m \cdot 2^i$ for $m = 0..2^{k/i}$. Set the $B^{th}$ entry of $T'(Q)$ to 1; otherwise set it to 0.

Example: $Q = \underline{10111110}$

<table>
<thead>
<tr>
<th>blocks</th>
<th>0</th>
<th>1</th>
<th>00</th>
<th>01</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T'(Q)$</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

$T'(Q)$ is exponentially large but only $O(|Q|)$ entries are set to 1; Only these entries are explicitly listed in lexicographic order. e.g. $T'(Q) = 0, 1, 10, 11, 1011, 1110, 10111110$
Location Based Cores and Restricted Block Edit Distance

Location based cores restrict edit operations to *aligned blocks*:

e’(Q,R): transform Q into R via the following operations only:
  - *swap* “aligned” blocks of Q,
  - *copy* a block over an “aligned” block (replaced one must have another aligned copy)
  - *replace* any single character of Q
  - *reverse* any block of Q,

Two blocks W and Z are aligned if their length is the same, and their relative location to longer core blocks is the same.
Transforming $e'(.,.)$ into $h(.,.)$

**Claim:** For any $Q$ and $R$

$$h(T'(Q), T'(R)) = O(\log |Q| \cdot e'(Q, R))$$

**How does a restricted edit operation manifest itself on $T'$:**

- Swapping same size blocks do not affect non-intersecting cores.
- Aligned blocks are partitioned to small cores identically. Swapping aligned blocks do not change the internal cores.
- The only cores that change as a result of a swap are those which intersect but are not completely covered by swapped blocks.
- Because cores of length $2^i$ do not overlap, there are at most 4 blocks of size $2^i$ that can change as a result of a swap or any other restricted edit operation.
- Thus we can estimate $e'(.,.)$ within a factor of $O(\log|Q|)$ via $h(.,.)$
Extension to general block edit distance

Challenges/ideas:

- \( e(Q,R) \) allows non-aligned block edit operations – rather than aligned block operations.

- \( T(Q) \) is composed of position independent core blocks - obtained by Locally Consistent Parsing

Claim: \( h(T(Q), T(R)) = O(\log |Q| \cdot \log^*|Q| \cdot e(Q,R)) \)

\[ = \Omega\left(\frac{e(Q,R)}{\log^*|Q|}\right) \]
Locally Consistent Parsing - LCP

For the moment assume no pair of adjacent characters are identical.

Write each character as a bit string ie $A = 00000$, $C = 00001$

Generate a ‘tag’ for each character as:
Smallest bit location where it differs from its left neighbor
+ Bit value there

e.g. | Char  | C  | G  | A  |
    |-------|----|----|----|
    | Binary| 01 | 11 | 00 |
    | Location | - | 01 | 00 |
    | Tag    | -  | 011| 000|
Locally Consistent Parsing II

If the starting alphabet is $\Sigma$, the tags can take $2 \log |\Sigma|$ values - the alphabet will change-

Repeat the procedure on the tags iteratively until tags are 3 bits long.

Properties of the final tags:
- Final alphabet is $\{0,1,2,3,4,5\}$
- No adjacent pair is identical
- Takes $\log^* |\Sigma|$ iterations
- Each final tag depends on the $\log^* |\Sigma|$ characters to its left
Marking characters

Consider the final tags, and mark the following characters:
• Mark any tags that are local maxima (greater than left & right)
• Also mark any local minima if not adjacent to a marked char.

Clearly, no two adjacent characters are marked.
Also, successive marked tags are separated by at most five tags

Q C A G A T C T A G C
Tags - 10 01 00 11 10 01 00 11 10
Obtaining Core Blocks

Obtain a core block by concatenating each marked character and the five characters to its left and one character to its right.

Each character is included in at least one core.

There are at most $|Q|/2$ cores.

Now replace each core with a unique label to shrink the size of the input. Iterate on the shrunk input.
The cores are **consistently generated and labeled**: if a block is a core block somewhere it is always a core elsewhere.

**Claim:** \( h(T(Q), T(R)) \) is an \( O(\log|Q| \log^*|Q|) \) approximation to \( e(Q, R) \)
Upper bound

• Cores are consistently generated independent of location; no need to stick to *same size blocks*: arbitrary swapping do not affect cores which do not intersect.

• Because cores are consistently generated, if a block moves around, the cores within that block do not change: *no need for aligned cores for swapping* etc.

• Any edit operation can change at most $\log^*|Q|$ of cores of any level (they all intersect at the edited block); total cores affected are $O(\log|Q| (\log^*|Q|))$

• Thus an edit operation can only change $O(\log|Q| (\log^*|Q|))$ entries of $T(Q)$
Lower bound

Prove that $e(Q,R) = O(h(T(Q),T(R)))$

**Procedure:** start from $Q$, build $QR$, then remove $Q$ leaving $R$. Total number of edits : $O(h(T(Q),T(R)))$.

Build $R$ left-to-right by forming its fat-tree top down:
- If a core is not present in $Q$ or in the partially built $R$, give a credit to each descendant and recurse on each descendant.
- If a core is present in $Q$ or in the partially built $R$, copy it at unit cost.
- The only charges correspond to blocks that are present in $T(R)$ but not in $T(Q)$, and each is charged to at most once.
Finishing off

Once QR is built, remove Q - remember, all operations are reversible.

**Total cost:** number of cores that are present in one sequence but are not in the other; i.e., $h(T(Q), T(R))$.

**Verdict:** we can transform Block Edit Distance into Hamming Distance, with approximation factor $O(\log |Q| \log^* |Q|)$
Transforming Hamming Distance to Prefix Distance

- Consider the binary representation of sequence R.
- Randomly sample $\beta$ of R’s bits and calculate their parity [AMRT96]; the parity gives the first bit of R’s fingerprint.
- Iteratively calculate the parities of geometrically increasing samples, obtaining 2nd, 3rd... bits of R’s fingerprint.

**Details:** probability of under-estimation is $\epsilon$. Fix $\beta$ to be proportional to $-1/\log \epsilon$.

For $i = 1\ldots \log_\beta n$, sample $\beta^i$ locations $r_i(1..\beta^i)$ from R;
Build fingerprint $f(R)$ as $f_i(R) = \text{xor}_{j=1\ldots \beta^i}(R[r_i(j)])$
Estimating Hamming Distance

Compute $f(R)$ and $f(Q)$ using same $r$

Compute $p(f(Q), f(R))$.

Hamming distance estimate: $h'(Q,R) = \frac{3|Q|(\beta - 1)}{2p(f(Q), f(R))}$

Summary:

Size of fingerprint: $O(\log |Q|)$,

$h'(Q,R)$ estimates $h(Q,R)$ within a constant factor with fixed prob.

The fingerprints can be put in a binary trie for fast prefix search.
Converting Constant Probability to High Probability:

• Recent work on data structures for Hamming NNs.

**Input:** \( n \) bit sequences with \( d \) bits each.

• Preprocessing time: \( (d.n)^{O(1)} \)

• Space: \( n.d^{O(1)} \)

• Query time: \( O(d \text{ polylog } n) \)

Probability of failure in preprocessing: \( \gamma \)

Once correctly constructed, returns \((1+\varepsilon)\) approx-NN with probability \((1-\mu)\)

• Vector nearest neighbors - VNN:
  \( L_0, L_1, L_2, \text{Hausdorff} \ldots \text{[MI98, KOR98, I99]} \ldots \)
Open problems

- How do we focus on character edits only? Fundamental open problem.
- Is a factor of $O(\log d)$ inherent in approximating block edit distance?
- How can we give non-uniform weights to edit operations?