# Permutation Editing and Matching via Embeddings

Graham Cormode, S. Muthukrishnan,
Cenk Sahinalp

(grahamc@dcs.warwick.ac.uk)

## Permutation Editing and Matching

- Why study permutations?
- Distances between permutations
- Approximating permutation distances with embeddings into well studied spaces
- Some applications — Pattern Matching, Approximate Nearest / Furthest Neighbors

## Why study Permutations?

- Permutations model (simplistically) biological sequences
- Distances on permutations give explanations of evolutionary behaviour
- Problems on permutations give insight onto problems on strings
- Nice, clean combinatorial problems: we treat all permutations as an arrangement of integers 1 to n

# Distances between permutations

How “close” are two permutations P and Q?

One approach: allow certain editing operations.
The number of editing operations needed to turn P into Q is their distance.

Consider the transposition distance:

$1\ 3\ \underbrace{5\ 6\ 8}\underbrace{4\ 2}\ 7$ $\qquad$ $1\ 3\ \underbrace{4\ 2}\underbrace{5\ 6\ 8}\ 7$

The minimum number of transpositions needed to turn P into Q is their Transposition Distance, t(P,Q).

## Approximating Transposition Distance

No efficient way to find Transposition Distance is known

Try to find a guaranteed approximation instead:

- Extend every permutation so that the first element is 0, the last is n+1
- Count the number of “transposition breakpoints”: when j immediately follows i in Q but not in P

P: **0** 3 6 5 1 2 4 **7**   Q: $\mathbf{0} \underset{\uparrow}{\,} 5\ 1\ 2 \underset{\uparrow}{\,} 3\ 6 \underset{\uparrow}{\,} 4\ \mathbf{7}$

The number of Transposition Breakpoints gives a 3-approximation for the Transposition Distance

# Approximating Transposition Distance II

Proof of the claim:

— Any transposition can remove at most 3 transposition breakpoints (because only 3 adjacencies change)

— Can remove at least one breakpoint per transposition

**Q**: $0\ Q_1 \ldots Q_i\ Q_{i+1} \ldots \ldots \ldots Q_n\ n{+}1$

**P**: $0\ Q_1 \ldots Q_i\ \underbrace{P_j \ldots}\underbrace{Q_{i+1} \ldots}\ P_n\ n{+}1$

Therefore, the true transposition distance is at most the # of breakpoints, and at least 1/3 the # of breakpoints

## Transforming into Hamming Distance

We can embed into the Hamming space of binary strings, H

The transform: Build a binary matrix for a permutation, **T(P)**
so that T(P)[i,j] = 1 if j immediately follows i in P
and T(P)[i,j] = 0 otherwise

Each Transposition breakpoint between P and Q corresponds to a place where T(P) = 1 and T(Q) = 0, and vice-versa.

$\therefore$ The Hamming distance of these matrices leads to a 3-approximation for the Transposition distance. ❑

Can improve this to 9/4 approx using Walter, Dias, Meidanis '00

## The value of the embedding approach

• The transformation of P is completely independent of Q

• Hamming distance is a well understood, well studied distance

• Many interesting problems have been solved for Hamming distance:

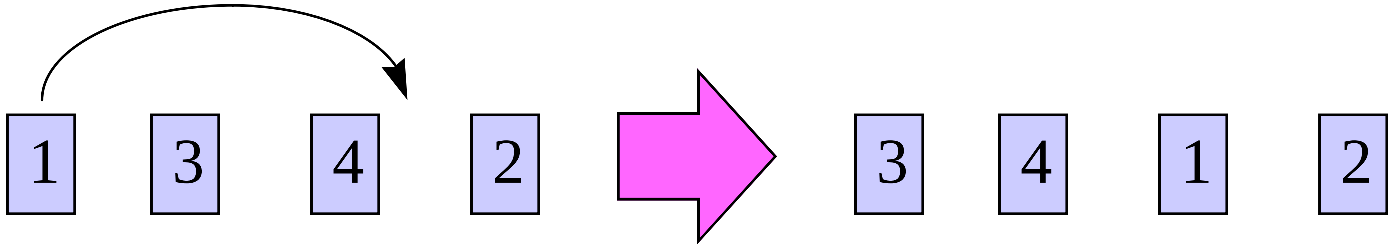
- Approximation using O(log 1/$\delta$) bits of communication
- “Sketching” using a sublinear size sketch
- Approximate Nearest Neighbors, sublinear query cost
- Clustering problems
- Many more…

Instead of solving these problems directly for Transposition Distance, we can use the solutions for Hamming Distance to immediately give solutions for Transposition Distance.

# Permutation Edit Distance

Permutation Edit Distance, e(P,Q):

Permitted operation is to move a single symbol at a time



This can be much larger than the Transposition Distance

eg P = 1 2 3 … 2n Q = n+1 n+2… 2n 1 2 … n

t(P,Q) = 1 but e(P,Q) = n

Similar to character based (string) edit distances

# Embedding into Intersection

**Define:**

A(P)[i,j] = 1 if i occurs exactly $2^k$ before j in P (for some k)
A(P)[i,j] = 0 otherwise

B(Q)[i,j] = 1 if j occurs before i in Q
B(Q)[i,j] = 0 otherwise

Intersection Size between two bit vectors, X and Y
I(X,Y) = number of places where X & Y are both 1

Claim: $e(P,Q) \leq I(A(P),B(Q)) \leq \log n \cdot e(P,Q)$

That is, the intersection size of A(P) and B(Q) is a log n-approximation for Permutation Edit Distance

# Example of permutation editing

$P = 5 \;\; \underline{2} \;\; 3 \;\; \underline{4} \;\; 1 \;\; 7 \;\; 6 \;\; \underline{8}$

$Q = 5 \;\; 8 \;\; 3 \;\; 1 \;\; 2 \;\; 7 \;\; 6 \;\; 4$

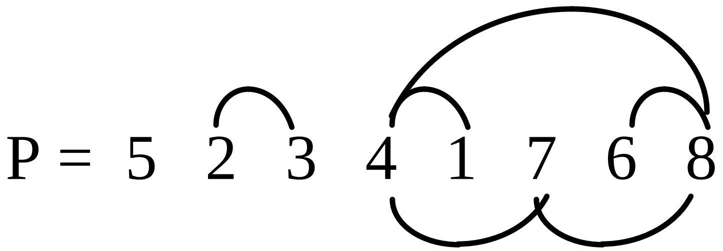
What does I(A(P),B(Q)) tell us?

- that we should count one for every pair i,j

where i occurs $2^k$ before j in P but the **other way round** in Q.

Each "intersecting" pair tells us that one of them must be moved.

Mark on P which pairs contribute to I(A(P),B(Q))



Here, I(A(P),B(Q)) = 6, e(P,Q) = 3, log n = 3 so

$$e(P,Q) \leq I(A(P),B(Q)) \leq \log n \cdot e(P,Q)$$

## Upper bound

$I(A(P),B(Q)) \leq \log n \; e(P,Q)$

Suppose one move picks up j and puts it in a new place.
There are at most log n i's for which A(P)[i,j] = 1
Hence I(A(P),B(Q)) changes by at most log n for any move.

When we have finished, we have made Q, and I(A(Q),B(Q))=0
So overall, we have to reduce I(A(P),B(Q)) to zero
It can reduce by at most log n per move
So $\log n \cdot e(P,Q)$ must be at least I(A(P),B(Q)). ❑

# Lower bound

$e(P,Q) \leq I(A(P),B(Q))$

Notionally relabel Q so it is **1 … n**, and apply the relabelling to P

$$\begin{array}{lcccccccclcccccccc}
Q = & 5 & 8 & 3 & 1 & 2 & 7 & 6 & 4 & P = & 5 & 2 & 3 & 4 & 1 & 7 & 6 & 8 \\
 & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\
Q' = & \mathbf{1} & \mathbf{2} & \mathbf{3} & \mathbf{4} & \mathbf{5} & \mathbf{6} & \mathbf{7} & \mathbf{8} & P' = & \mathbf{1} & \mathbf{5} & \mathbf{3} & \mathbf{8} & \mathbf{4} & \mathbf{6} & \mathbf{7} & \mathbf{2}
\end{array}$$

To transform P' into Q', have to move everything that is **not** in a Longest Increasing Subsequence (LIS).
So $e(P,Q) = e(P',Q') = n - LIS(P')$

Also note that $I(A(P'),B(Q'))$ counts one for each pair in P' where $P'[i] > P'[i + 2^k]$ for some k.

# Lower bound

Consider only the adjacent items:

$$\begin{tikzcd}[column sep=small] \mathbf{1} & \mathbf{5} \arrow[r, bend left] & \mathbf{3} & \mathbf{8} \arrow[r, bend left] & \mathbf{4} & \mathbf{6} & \mathbf{7} \arrow[r, bend left] & \mathbf{2} \end{tikzcd}$$

Count the number of “breaks” as b(P') — here, b(P') = 3

Split P' two interleaved parts:

$P'_{odd}$ = **1 3 4 7**

$P'_{even}$ = **5 8 6 2**

Consider extending LIS of $P'_{odd}$ to be an increasing sequence of P'. Betwen 2 consecutive members of LIS($P'_{odd}$), either we can include a member of $P'_{even}$, or else there is a failed comparison.

This results in an Increasing Subsequence, whose length is at most LIS(P'), by definition.

$$\text{So } LIS(P') \geq LIS(P'_{odd}) + (LIS(P'_{odd}) - b(P'))$$

# Lower bound

| | |
|---|---|
| So | $LIS(P') \geq 2\,LIS(P'_{odd}) - b(P')$ |
| Symmetrically | $LIS(P') \geq 2\,LIS(P'_{even}) - b(P')$ |
| Sum and halve these | $LIS(P') \geq LIS(P'_{even}) + LIS(P'_{odd}) - b(P')$ * |

Now split $P'_{even}$ and $P'_{odd}$ into odd and even halves, and repeat the argument… keep going until we reach unit length sequences. The LIS of a unit length sequence is trivially 1.

Substitute back into *:

$$LIS(P') \geq \underbrace{1 + 1 + \ldots + 1}_{= n} - b(P') - \underbrace{b(P'_{even}) - b(P'_{odd}) - \ldots}_{= -I(A(P'),B(Q'))}$$

Hence $I(A(P),B(Q)) \geq n - LIS(P') = e(P',Q') = e(P,Q)$ ❑

## Consequences

Permutation Edit Distance can be approximated by comparing independent binary matrices.

Intersection size is harder to deal with than Hamming distance, so we don’t get so many “free” results

This approach lets us solve some problems quickly
eg Approximate Pattern Matching

> Given a long string T and a permutation P, find the approx. cost of matching P at each location in T.

## Further consequences

The weight of these matrices is fixed, $|A(P)| = n \log n$

and one is much smaller than the other $|B(Q)| = n^2/2$

So can approximate $|A(P) \cap B(Q)|$ with

$$n^2/2\ |A(P) \cap B(Q)| \,/\, |A(P) \cup B(Q)|$$

Can find eg Approx Furthest Neighbors under this measure after preprocessing with sublinear time per query: adapt the results of Indyk-Motwani `98, with appropriate Locality Sensitive Hash functions.

# Conclusions

Extensions that can be made

- Distances based on reversals of sections of permutations
- Allow insertions / deletions so P and Q are not exact perms.
- Allow 1 of P or Q to be a string (contain repeats)
- Combinations of operations eg Reversals and Transpositions
- Any mixture of the above!

Future Directions

- Better approximations
- Different Embeddings, eg Edit Distance to Hamming Dist?
- Other approaches to Nearest Neighbors under edit distances.
- Similar transforms for distances between strings?
- Other applications?