

Embeddings of Metrics on Strings and Permutations

Graham Cormode

joint work with S. Muthukrishnan,
Cenk Sahinalp

“Miss Hepburn runs the gamut of emotions from A to B”

Dorothy Parker, 1933

Permutations and Strings

Strings

Web pages, email messages, PS/PDF files, books, letters, lecture notes... strings are ubiquitous

Sequences of n characters from an alphabet of size Σ

Permutations

Arrangement of n objects is modelled by a permutation
eg arrangement of chromosomes on a gene

Foundational combinatorial objects

A sequence of n integers $1 \dots n$, each appears once

Editing Distances

We consider a broad class of metrics on sequences
(Permutations and Strings):

Editing distances — define a set of permitted unit cost editing operations. Model this as a graph where vertices are sequences, edges link unit cost edits

Given two objects A and B , $d(A,B)$ = shortest path in the graph between nodes A and B

Clearly a metric. Usually, the graph will be connected.

Particular Metrics

We will consider each particular metric in turn

Many different metrics of interest on Strings and Permutations, most can be classed as editing distances.

Examples:

- Hamming distance on Strings (Communication Theory)
- Edit distance on Strings (Text Mining, Comp Bio)
- Inversion and Transposition Distance on Permutations (Comp Bio)

Problems on Editing Metrics

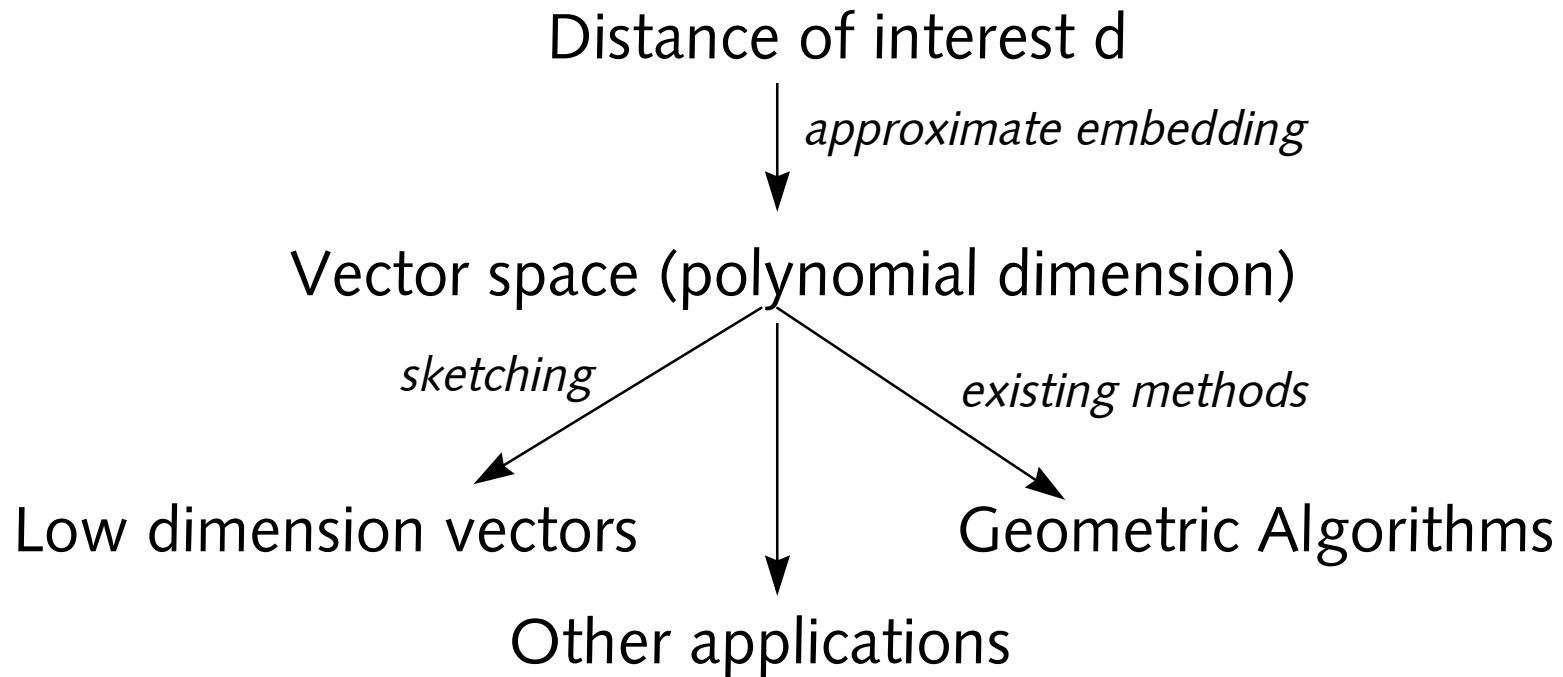
Many natural questions are parametrised by the metric in question.

- “Geometric” questions: approximate nearest neighbors, furthest neighbors, clustering, data mining
- Approximate Pattern Matching: find the subsequence of a long sequence that best matches a pattern sequence
- Compact representation: make a sketch of the sequence so that $d(A,B)$ can be approximated using $\text{sketch}(A)$, $\text{sketch}(B)$ — allows efficient communication etc.

We don't want to solve problems afresh for every metric!

Embedding Approach

Given a metric d , embed into a known space, solve the problems in the target space: gives an (approximate) solution to the problem in the original space.



Goals to strive for

- Embed into low dimensional space
- Embed into well-known metric (L_1 , L_2 or Hamming space)
- Low distortion embedding
- Embedding is easy to compute (time polynomial in n)
- Embedding can be computed in restricted model, especially streaming model

We will often be able to achieve several of these

These are the first results on these problems, drawing on techniques from geometry, parallel, string matching, information theory, graph theory, comp bio, databases.

Contrast to other methods

Bourgain-style embeddings: take n items in a metric space and embed into Euclidean space with $O(\log n)$ distortion

We have sequences of **length** n : Σ^n strings of length n .

Bourgain embedding would give distortion $O(n)$ - much too large!

Explicit representation of the metric requires $O(\Sigma^n)$ space.

We give embeddings that are computable for a sequence based only on that sequence by making observations about the combinatorial structure of the metric.

Permutations

Results from Cormode Muthukrishnan Sahinalp 2001

*“A, B, C
It’s as easy as 1, 2, 3
As simple as do re mi
A, B, C, 1, 2, 3,
Baby you and me”*

The Jackson Five, 1970

Toy Example

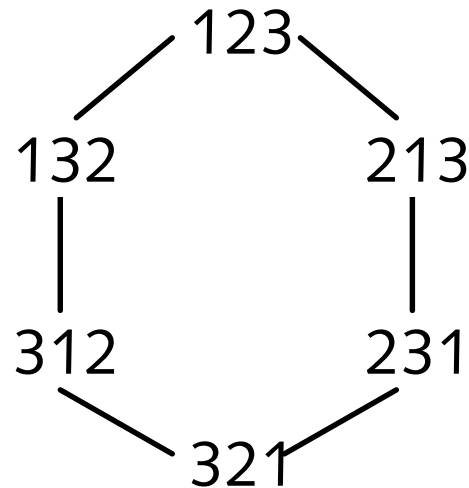
“Swap distance” between permutations of length n : edit operation is to swap two adjacent items.

Example

A = 123

B = 321

$d(A,B) = 3$



As the size of the permutation grows, the metric becomes less trivial. The distance corresponds to the number of exchanges in a bubblesort.

Combinatorial Structure

We observe that:

- Every swap in an optimal sequence 'fixes' a pair that occur one way round in A and the other way round in B
- No other swaps are necessary
- Therefore, swap distance is exactly the number of pairs which occur in different orientations

We can encode the relative ordering of each pair (i,j) occurring in A in a matrix $S(A)$ with $O(n^2)$ entries:

Put 1 in location (i,j) if i occurs before j in the permutation, and put 0 otherwise.

Embedding to Euclidean Space

Straightforward to see that $\|S(A) - S(B)\|_2 = d(A,B)$

Therefore, any algorithm to solve a problem in Euclidean space can be applied to swap distance by using this transform.

Pros: non-distortive embedding (rare for nontrivial egs)

Cons: bit array of size $O(n^2)$ instead of a permutation of n integers. Can reduce to $O(\log n)$ bits in Euclidean space using dimensionality reduction techniques.

Most other embeddings will be approximate...

Transposition Distance

Transposition Distance between permutations:

1 3 5 6 8 4 2 7 1 3 4 2 5 6 8 7

The minimum number of transpositions needed to turn A into B is their Transposition Distance, $t(A,B)$.

- 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 B but not in A

A: **0** 3 6 5 1 2 4 7 B: **0** 5 1 2 3 6 4 7

↑
↑
↑


Approximating Transposition Distance

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

- Any transposition can remove at most 3 transposition breakpoints (because only 3 adjacencies change)
- Can remove at least one breakpoint per transposition

B: 0 B_1 ... B_i B_{i+1} B_n $n+1$

A: 0 B_1 ... B_i A_j ... B_{i+1} ... A_n $n+1$



Therefore, the true transposition distance is at most the no. of breakpoints, and at least 1/3 the no. of breakpoints

Embedding to Euclidean Space

Embed into Euclidean space: Build a binary matrix $T(A)$
so that $T(A)[i,j] = 1$ if j immediately follows i in A
and $T(A)[i,j] = 0$ otherwise

Each breakpoint between A and B corresponds to a place where $T(A) = 1$ and $T(B) = 0$, and vice-versa.

The Euclidean distance of these matrices leads to a 3-approximation for the Transposition distance. \square

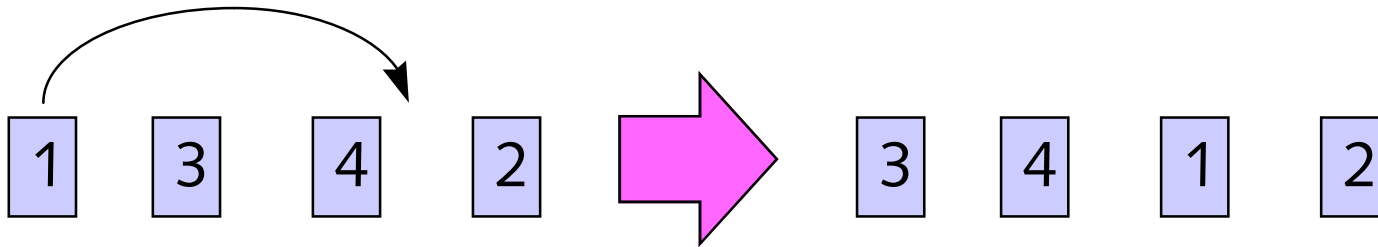
Improve to $9/4$ approx using Walter Dias Meidanis 00

Although $O(n^2)$ bits, only $O(n)$ are 1 so process in linear time by ignoring zero entries. Can compute on stream.

Permutation Edit Distance

Permutation Edit Distance, $e(P,Q)$ (the Ulam Metric)

Permitted operation is to move a single symbol at a time



$e(P,Q) = n - \text{LCS}(P,Q)$. Very important foundational problem.

Classical String Edit distance is strongly related to this: edit distance of two strings is $n - \text{Longest Common Subsequence}$

This problem is more restricted, gives insights into string edits

Embedding Ulam Metric

For $n = 3$:

$$E(123) = [0, 0, 0, 0]$$

$$E(132) = [0, 0, 1, 1]$$

$$E(213) = [1, 0, 0, 1]$$

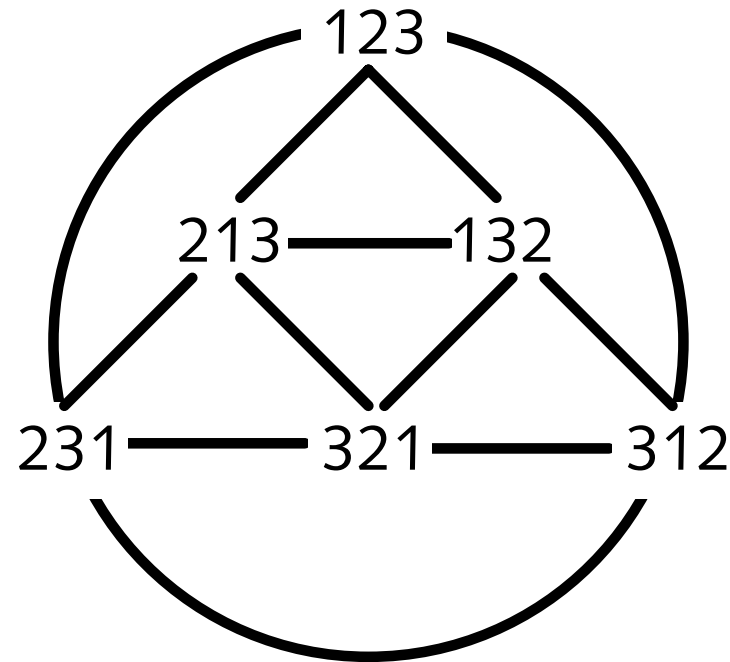
$$E(231) = [1, 1, 0, 0]$$

$$E(312) = [0, 1, 1, 0]$$

$$E(321) = [1, 1, 1, 1]$$

$$\|E(A) - E(B)\|_2 = 2e(A, B)$$

A non-distortive embedding! What about $n=4$?
Arbitrary n ?



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 and 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 Edit

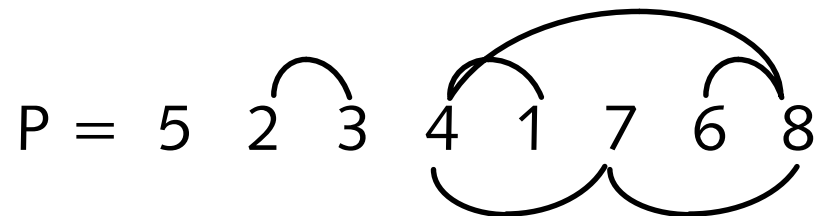
$$\begin{array}{r} P = 5 \quad \underline{2} \quad 3 \quad \underline{4} \quad 1 \quad 7 \quad 6 \quad \underline{8} \\ Q = 5 \quad 8 \quad 3 \quad 1 \quad 2 \quad 7 \quad 6 \quad 4 \end{array}$$

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 **other way round** in Q .

Each “intersecting” pair means 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 $\log n \times e(P, Q)$ must be at least $I(A(P), B(Q))$. \square

Lower bound

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

Notionally relabel Q so it is 1 ... n, and apply relabelling to P

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

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

$$\text{So } e(P,Q) = e(P',Q') = n - \text{LIS}(P')$$

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: **1 5 3 8 4 6 7 2**
Count the number of "breaks" as $b(P')$ — here, $b(P') = 3$

Split P' two interleaved parts:

$$\begin{array}{l} P'_{\text{odd}} = \mathbf{1 \quad 3 \quad 4 \quad 7} \\ P'_{\text{even}} = \mathbf{5 \quad 8 \quad 6 \quad 2} \end{array}$$

Try extending LIS of P'_{odd} to be an increasing sequence of P' .
Between 2 consecutive members of $\text{LIS}(P'_{\text{odd}})$, 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 $\text{LIS}(P')$, by definition.

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

Lower bound

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

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

Substitute back into *:

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

Hence $I(A(P), B(Q)) \geq n - \text{LIS}(P') = e(P', Q') = e(P, Q)$ □

Consequences

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

Intersection size is not a metric space, and it is harder to deal with than Euclidean space.

But 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

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

Can find eg Approx Furthest Neighbors under this measure after preprocessing, adapting results of Indyk-Motwani 98.

Strings

Initial ideas in Cormode Paterson Sahinalp Vishkin 00

Developed in Muthukrishnan Sahinalp 00

Extended in Cormode Muthukrishnan 02

“Bypasses are devices which allow some people to drive from point A to point B very fast while people dash from point B to point A very fast. People living at point C, being a point directly in between, are often given to wonder what’s so great about point A that so many people of point B are so keen to get there, and what’s so great about point B that so many people of point A are so keen to get there. They often wish that people would just once and for all work out where the hell they wanted to be.”

Douglas Adams, 1979

***q*-grams**

Embedding ideas have been used in strings for a while...
not always deliberately!

q-grams: A *q*-gram is just a substring of length *q*

The *q*-gram representation of a string *A* is the histogram of *q*-grams of that string. Call this $F_q(A)$.

We can then look at $\|F_q(A) - F_q(B)\|_1$ as a measure of string distance (Ukkonen 92).

But : $F_q(A) = F_q(B)$ does not mean $A = B$, so not a metric

Still a good heuristic, often used in database applications.

Other Failed Ideas

We want to take same approach to strings as permutations

Look for Combinatorial features that capture edit distances

Presence / Frequency of substrings: q -grams don't work.

Try a binary tree structure: but a single character insert changes the substring set completely

Try **all** substrings of length 2^i : edits still have too much effect on the set

So we will need something more sophisticated

Same idea, different substrings...

Now describe a method that uses the same underlying idea: represent a string by a histogram of substrings so that L_1 difference of histograms approximates an editing distance.

Difference is that we obtain a guaranteed distortion embedding, poly-log in max length of string.

The embedding is fairly efficient to compute, based on parsing derived from deterministic coin tossing

Same ideas used in string matching by Sahinalp Vishkin 96, Mehlhorn Sundar Uhrig 97, Alstrup Brodal Rauhe 00.

String Edit Distance with Moves

We will study the string edit distance *with moves*:

$d(A,B)$ = smallest no. of editing operations to turn A into B

- insert a character
- delete a character
- replace a character
- move a substring

Substring moves are relevant to many situations, eg
Computational Biology, Text Editing, Web Page updates etc.

We embed with a distortive factor of $O(\log n \log^* n)$

Overview of Structure

We will build a 2-3 tree on the string

Each node corresponds to a substring that we will store in a histogram

Iterative procedure: parse the string into pairs and triples to make the nodes at the next level, then repeat on shorter string

Parsing has several parts:

- isolate simple patterns
- alphabet reduction on remainder
- mark certain features
- use these to divide into pairs and triples

Parsing for the Embedding

Embedding is based on parsing strings in a deterministic way

We parse the strings in a way so that edit operations have only a limited effect on the parsing — this will allow us to make the approximation.

Find 'landmarks' in the string based only on their locality.

- Repetitions (aaa) are easily identifiable landmarks
- Local maxima are good landmarks in varying sequences, but may be far apart — so reduce the alphabet to ensure landmarks occur often enough.

So: Isolate repetitions, leave substrings with no repeats.

Alphabet Reduction

Write each character as a bitstring ie $a = 00000$, $b = 00001$

Reduce the alphabet. For each character, find a new label as:

Smallest bit location where it differs from its left neighbor + Bit value there

e.g.	Char	b	d	a
	Binary	00001	00011	00000
	Location	-	001	000
	Label	-	001 1	000 0

Alphabet Reduction

If starting alphabet is Σ , new alphabet has $2 \log |\Sigma|$ values

Repeat the procedure on the string iteratively until the alphabet is size 6, $\Sigma' = \{0,1,2,3,4,5\}$

Then reduce from 6 to 3, ensuring no adjacent pair are identical (first remove all 5s, then all 4s, then all 3s)

Properties of the final labels:

- Final alphabet is $\{0,1,2\}$
- No adjacent pair is identical
- Takes $\log^* |\Sigma|$ iterations
- Each label depends on $O(\log^* |\Sigma|)$ characters to left

Marking characters

Consider the final labels, and mark certain characters:

- Mark labels that are local maxima (greater than left & right)
- Also mark any local minima not adjacent to a marked char

Clearly, no two adjacent characters are marked.

Also, marked labels are separated by at most two labels

Text	c	a	b	a	g	e	f	a	c	e	d
Labels	-	010	001	000	011	010	001	000	011	010	011
Final	-	<u>2</u>	1	<u>0</u>	3 1	<u>2</u>	1	<u>0</u>	3 1	<u>2</u>	3 0

Group into pairs and triples

Now, whole string can be arranged into pairs and triples:

- For repeats, parse in a regular way

aaaaaaaa → (aaa)(aa)(aa)

- For varying substrings, use alphabet reduction, define pairs and triples based on the marked characters.

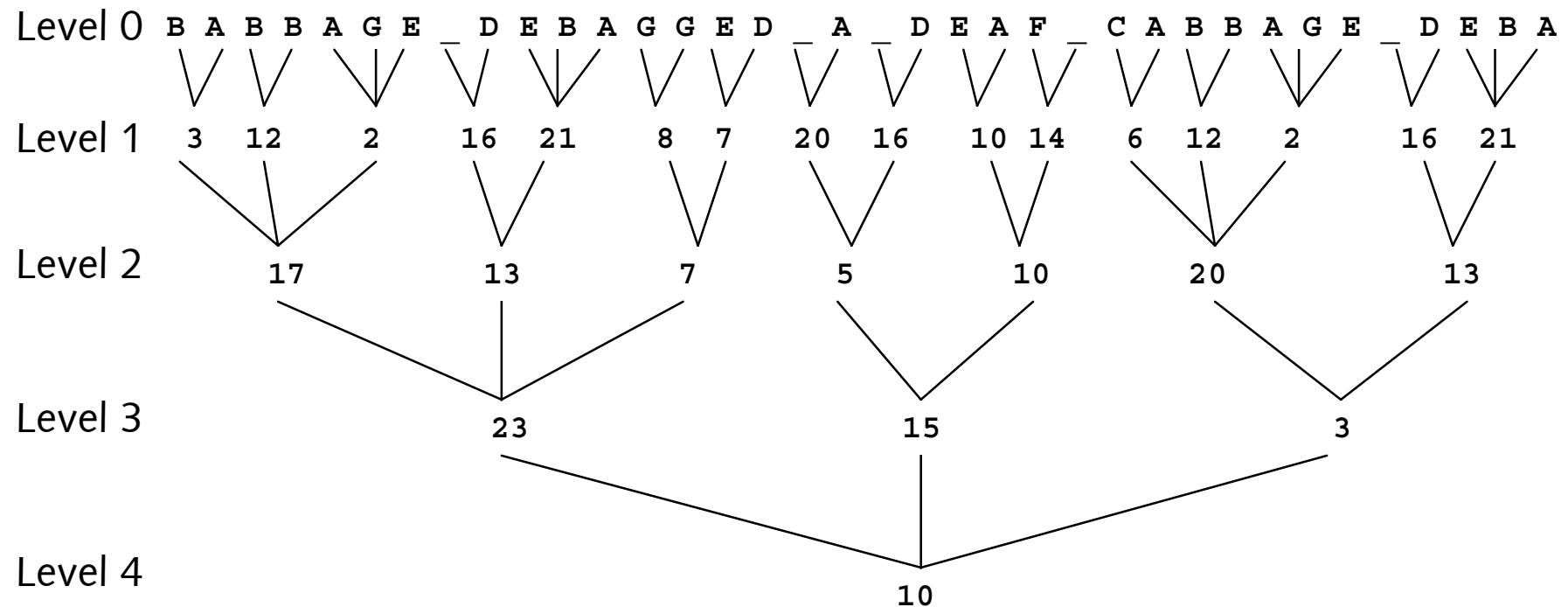
Text	c	a	b	a	g	e	f	a	c	e	d
Final	-	<u>2</u>	1	<u>0</u>	1	<u>2</u>	1	<u>0</u>	1	<u>2</u>	0

Parsing of each character depends on $\log^* n + c$ neighborhood

Relabel each pair or triple — do this deterministically, building a dictionary of labels using Karp-Miller-Rosenberg labelling.

Build Hierarchical Structure

Given new labels, repeat the process... this builds a 2-3 tree



Can be constructed in time $O(n \log^* n)$

Vector Representation

From the structure, derive vector representation V recording occurrence frequency of each (level, label) pair:

(0,a)	(0,b)	(0,c)	(0,d)	(0,e)	(0,f)	(0,g)	(0,_)
8	7	1	4	6	1	4	5

(1,2)	(1,3)	(1,6)	(1,7)	(1,8)	(1,10)	(1,12)	(1,14)	(1,16)	(1,20)	(1,21)
2	1	1	1	1	1	2	1	3	1	2

(2,5)	(2,7)	(2,10)	(2,13)	(2,17)	(2,20)	(3,3)	(3,15)	(3,23)	(4,10)
1	1	1	2	1	1	1	1	1	1

Theorem: $\frac{1}{2}d(A,B) \leq \| V(A) - V(B) \|_1 \leq O(\log n \log^* n) d(A,B)$

Upper bound

$$\|V(A) - V(B)\|_1 \leq O(\log n \log^* n) d(A,B)$$

Consider the effect of each permitted edit operation:

- Insert / change / delete a character:
Fairly straightforward, at most $\log^* n$ nodes can change per level
- Move a substring:
Within the substring, there are no changes.
At fringes, only $O(\log^* n)$ nodes change per level

As each operation changes V by $O(\log n \log^* n)$, so

$$\|V(A) - V(B)\|_1 / O(\log n \log^* n) \leq d(A,B)$$

Hence the bound holds.

Lower bound

A constructive proof: we give an algorithm to transform A into B using at most $2 \lceil |V(A) - V(B)| \rceil_1$ operations.

Be sure to keep hold of large pieces of the string that are common to both, so 'protect' enough pieces of A that are needed in B, and avoid changing these.

Then we will go through level by level to turn A into B:

- At the bottom, add or remove characters as needed.
- For each subsequent level, proceed inductively:
 - Assume we have enough nodes of the level below.
 - Then to make any node only need to move at most 2 nodes from the level below. □

Extensions to this method

- Can allow the editing distance to include copy substring operations by keeping the same parsing but embedding into Hamming distance instead of L_1 !
- Can add other operations with some extra technology (linear scaling, substring reversals etc.)
- Can compute the embedding in the streaming model (feeding into a streaming algorithm for L_1 eg Indyk 00)

Open question: what are other applications for this structure outside embedding — new kinds of wavelets?

Questions

Why do string metrics seem to require so much more effort than permutations? Are there “neater” embeddings?

Can the distortion factors be improved? To $O(\log n)$? To $O(1)$? To $1 + \epsilon$?

Can we extend to non-editing metrics eg with weighted operation costs instead of unit costs?

What about other combinatorial object distances: between trees, graphs, restricted classes of graphs?

String Edit Distance

There are very few results on embedding string distances — no other work on the subject, plenty of open problems.

My open question for several years now:

Is there a computable embedding of unit cost edit distance (insert / delete characters only) into another metric space?

Related results in Cormode Paterson Sahinalp Vishkin 00, some recent progress by Indyk and Sahinalp.

Permutation Edit Distance (Ulam Metric) is strongly related, but only limited results there.