Intro: Indexing

Ben Langmead

JOHNS HOPKINS
WHITING SCHOOL *of* ENGINEERING

Department of Computer Science

Indexing

Imagine we have recorded the ages of many people; say, voters:

How many voters are aged 27?

To find out, we have no choice but to scan $n = 1$M records

Order to the rescue

Index	Age
1	78
2	19
3	20
4	50
⋮	
500,000	54
500,001	50
500,002	19
500,003	77
⋮	
999,997	40
999,998	27
999,999	71
1,000,000	44

Example modeled on: Prezza, Nicola. Compressed Computation for Text Indexing. Diss. PhD thesis, University of Udine, 2016.

Indexing

17	18
33	18
39	18
60	18

⋮

999,905	49
999,985	49
4	50
18	50

⋮

999,649	101
999,811	101
433,034	103
377,003	104

Suppose our list is age-ordered

How many voters are aged 27?

Binary search

More specifically?

2 searches, one for the first age-27 person, one for last

Indexing

17	18
33	18
39	18
60	18

⋮

999,905	49
999,985	49
4	50
18	50

⋮

999,649	101
999,811	101
433,034	103
377,003	104

Simply ***ordering*** the data allows us to query it more efficiently

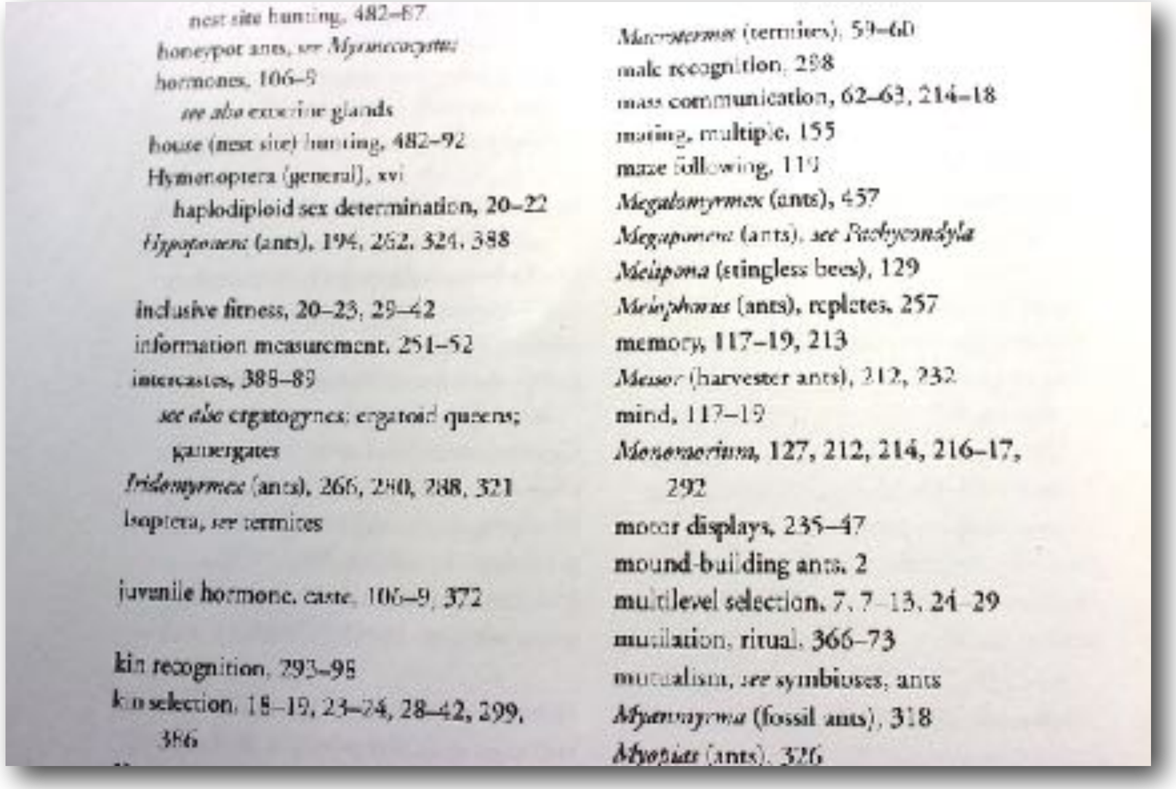
From n-item scan to two $\log_2 n$ binary searches

Did it also improve our ability to ***compress*** the age data?

Yes; we now have "runs" of same value, monotonicity, etc

Indexing

Grouping



Ordering

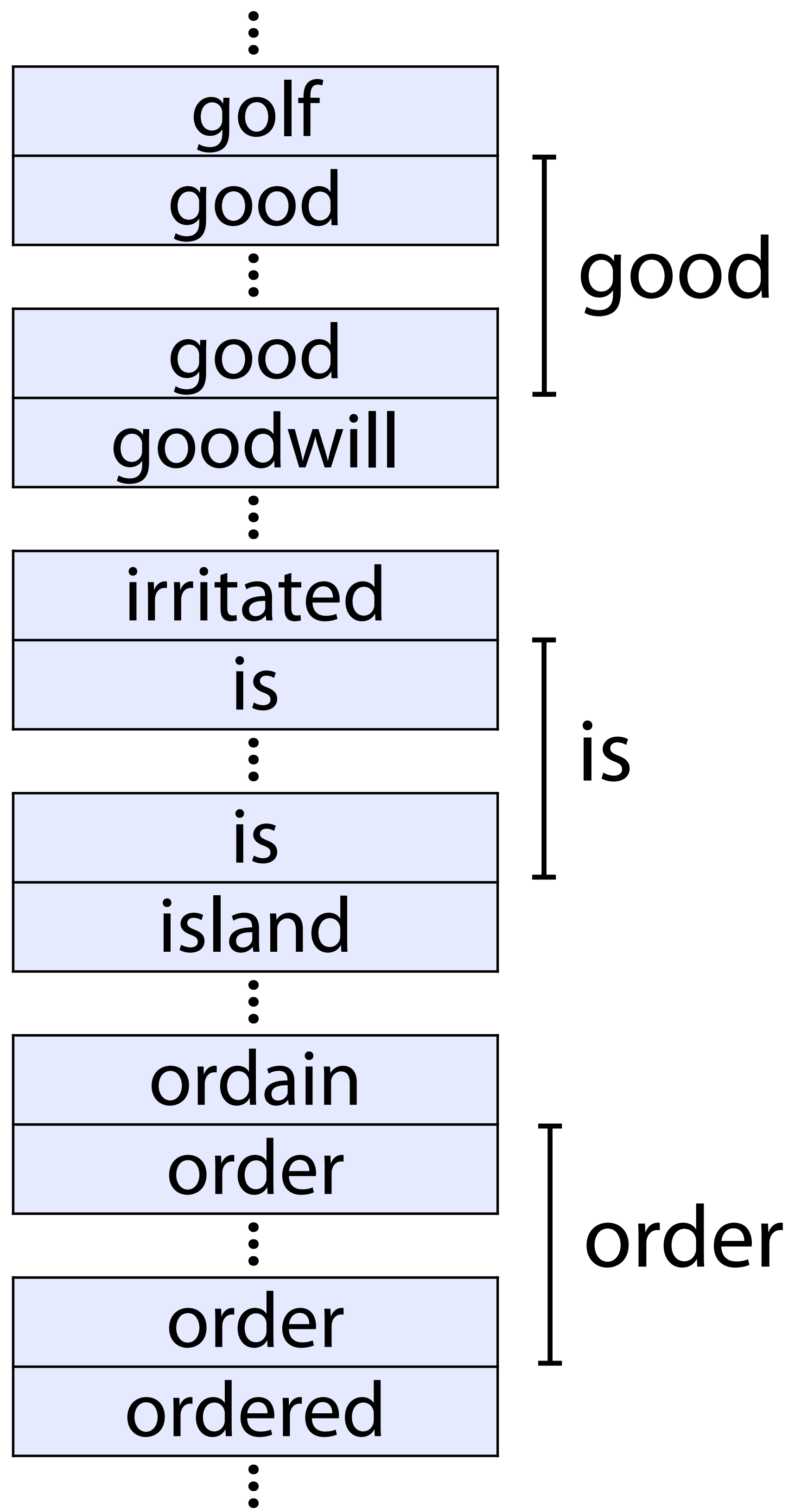
Indexing

We are working with a text. We want to know if some word occurs. The text is big but an excerpt is:

…		o	r	d	e	r		i	s		g	o	o	d		…

Ordering words alphabetically: good < `is` < `order`

Indexing



Can we still use binary search?

Yes, but what's the cost of comparing 2 words?

Several character comparisons needed to get relative order of `dinosaur` & `dinosaurs`

Again, we've improved queryability & compressibility

Indexing

Queries only on *words* is limiting

Texts might not consist of words e.g. DNA

Word matches might not be the right query

e.g. autocomplete

e.g. inexact matching

What if we'd like to be able to query ***any substring?***

Indexing

…	o	r	d	e	r	_	i	s	_	g	o	o	d	…

Use underscore (_) for space, assume it comes first alphabetically

Put all suffixes in order...

Indexing

_good...
_is_good...
d...
der_is_good...
er_is_good...
good...
is_good...
od...
ood...
order_is_good...
r_is_good...
rder_is_good...
s_good...

(This is just the relative order of the order_is_good suffixes)

Can we use binary search?

Yes; still might need several character comparisons to get relative order of suffixes

Motivating questions

How do we measure the amount of redundant ***information*** in a string?

How do we represent strings so that redundant information takes ***minimal space***?

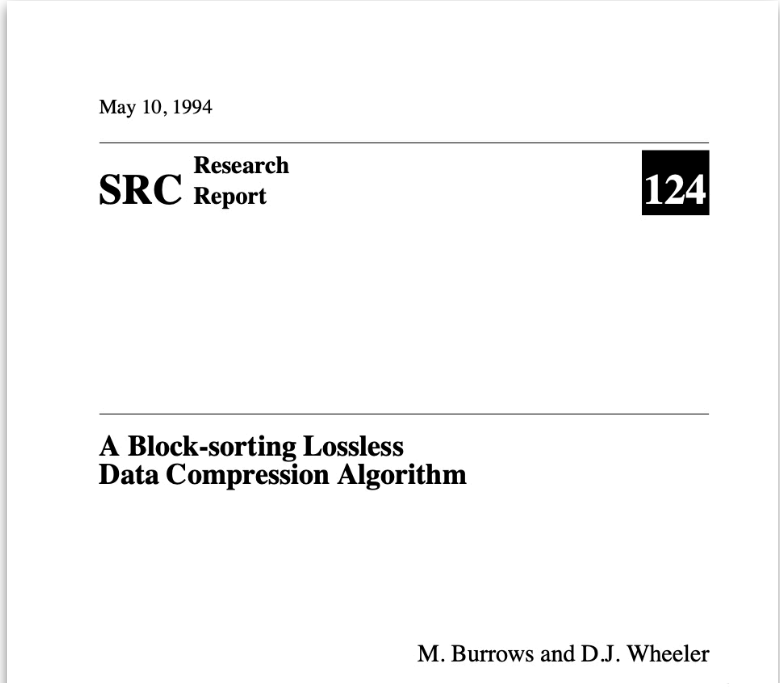
How can orderings "***reveal***" structure and make strings compressible?

How can ordering make strings fast to ***search***, faster than binary search?

Burrows-Wheeler Transform

David Wheeler

Michael Burrows


May 10, 1994

SRC Research Report 124

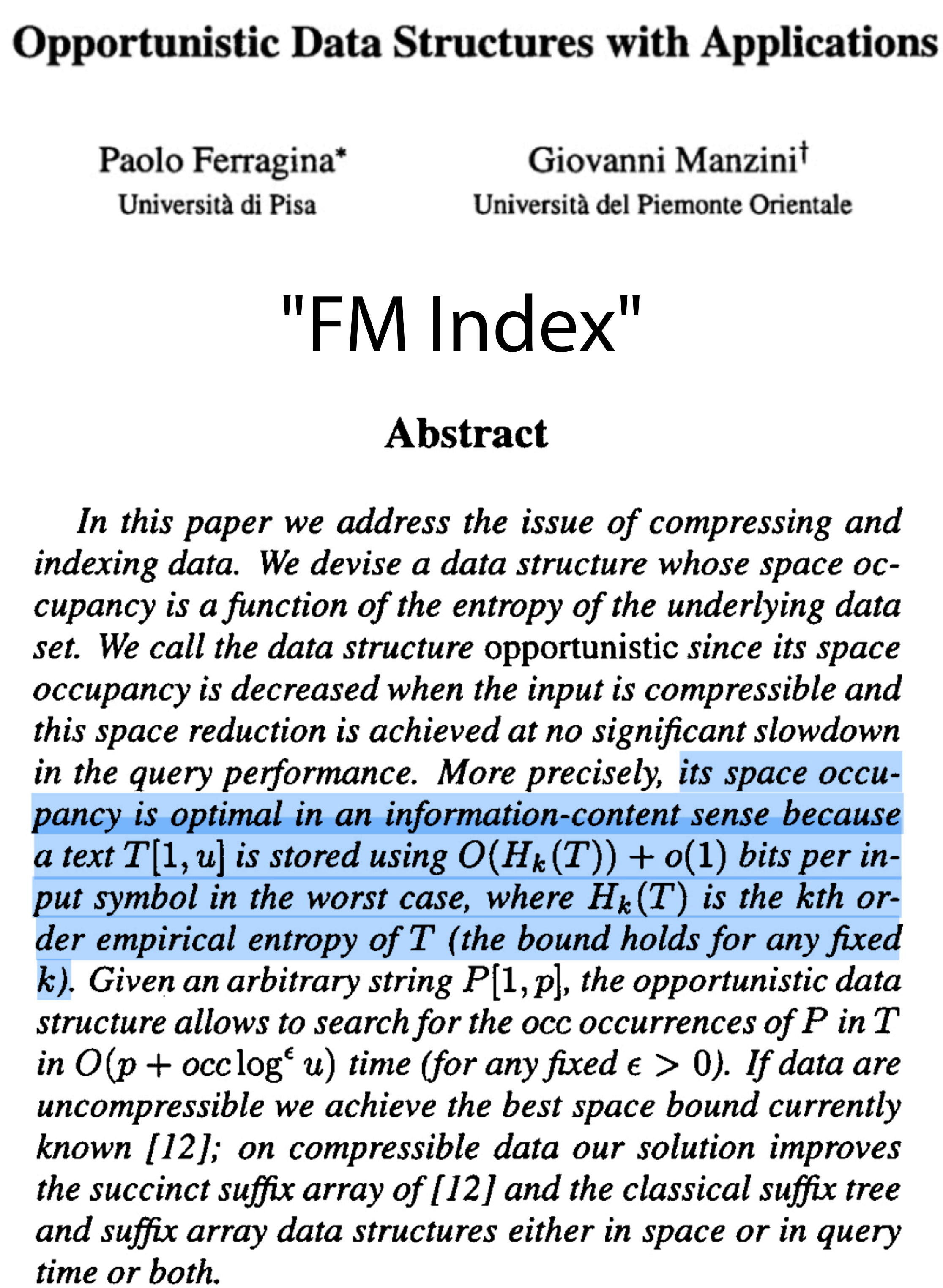
A Block-sorting Lossless Data Compression Algorithm

M. Burrows and D.J. Wheeler

Briefly, our algorithm transforms a string S of N characters by forming the N rotations (cyclic shifts) of S, sorting them lexicographically, and extracting the last character of each of the rotations. A string L is formed from these characters, where the ith character of L is the last character of the ith sorted rotation. In addition to L, the algorithm computes the index I of the original string S in the sorted list of rotations. Surprisingly, there is an efficient algorithm to compute the original string S given only L and I.

The sorting operation brings together rotations with the same initial characters. Since the initial characters of the rotations are adjacent to the final characters, consecutive characters in L are adjacent to similar strings in S. If the context of a character is a good predictor for the character, L will be easy to compress with a simple locally-adaptive compression algorithm.

Burrows-Wheeler Transform


Opportunistic Data Structures with Applications

Paolo Ferragina* — Università di Pisa

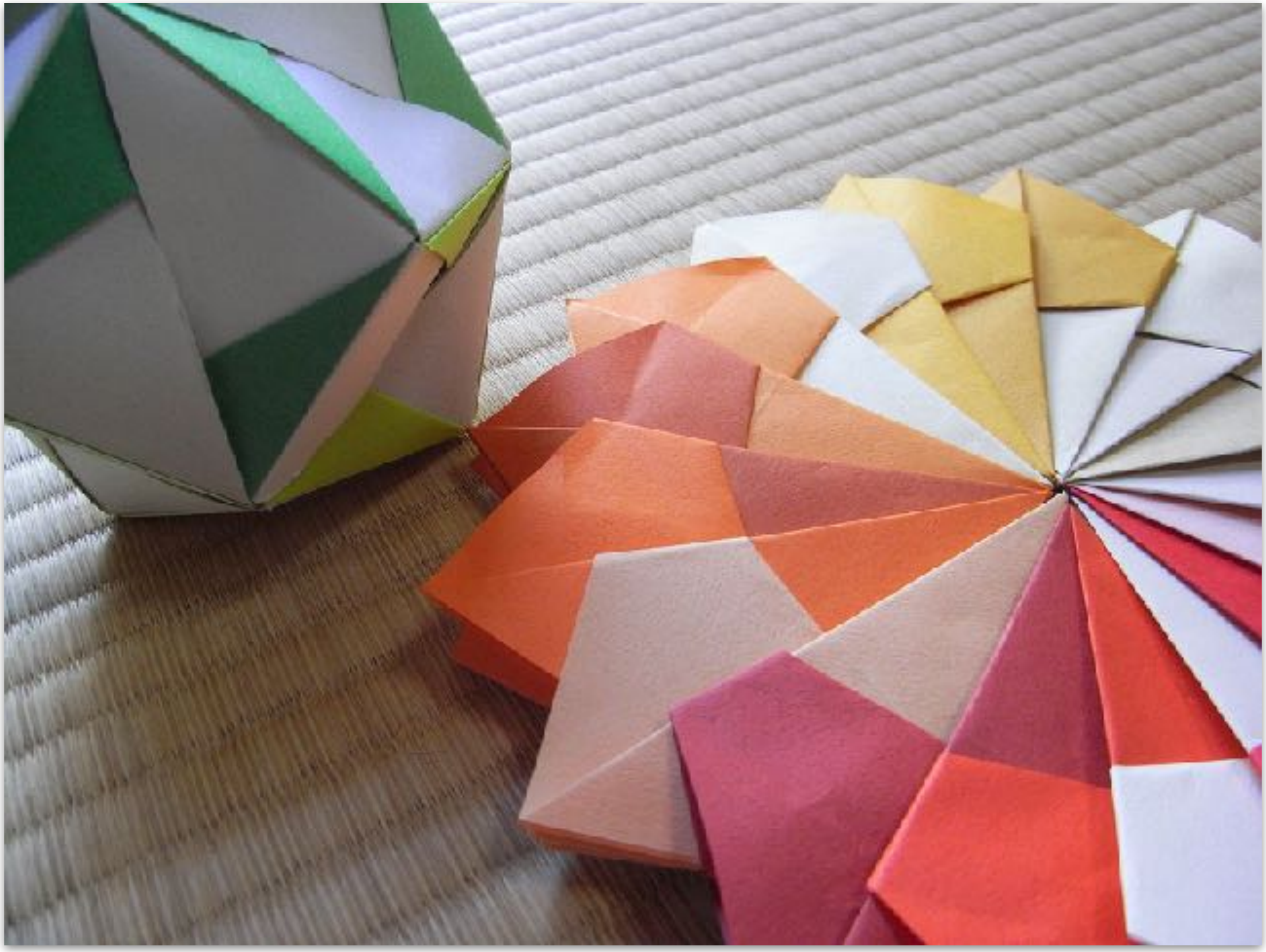
Giovanni Manzini† — Università del Piemonte Orientale

"FM Index"

Abstract

In this paper we address the issue of compressing and indexing data. We devise a data structure whose space occupancy is a function of the entropy of the underlying data set. We call the data structure opportunistic *since its space occupancy is decreased when the input is compressible and this space reduction is achieved at no significant slowdown in the query performance. More precisely, its space occupancy is optimal in an information-content sense because a text $T[1,u]$ is stored using $O(H_k(T)) + o(1)$ bits per input symbol in the worst case, where $H_k(T)$ is the kth order empirical entropy of T (the bound holds for any fixed k). Given an arbitrary string $P[1,p]$, the opportunistic data structure allows to search for the occ occurrences of P in T in $O(p + occ \log^\epsilon u)$ time (for any fixed $\epsilon > 0$). If data are uncompressible we achieve the best space bound currently known [12]; on compressible data our solution improves the succinct suffix array of [12] and the classical suffix tree and suffix array data structures either in space or in query time or both.*

Burrows-Wheeler Transform



Reveal structure by turning string "inside out"

https://commons.wikimedia.org/wiki/File:Image-2D_and_3D_modulor_Origami.jpg