On Compression and Indexing: two sides of the same coin

Paolo Ferragina
Dipartimento di Informatica, Università di Pisa

What do we mean by "Indexing" ?

Types of data

Linguistic or tokenizable text
Raw sequence of characters or bytes

DNA sequences
Audio-video files
Executables

Types of query

Word-based query
Character-based query

Arbitrary substring
Complex match

Two indexing approaches :

- Word-based indexes, here a notion of "word" must be devised !
  - » Inverted files, Signature files, Bitmaps.
- Full-text indexes, no constraint on text and queries !
  - » Suffix Array, Suffix tree, String B-tree [Ferragina-Grossi, JACM 99].

What do we mean by "Compression" ?

☺ Compression has two positive effects:

- ✓ Space saving (or, double memory at the same cost)
- ✓ Performance improvement
  - Better use of memory levels close to processor
  - Increased disk and memory bandwidth
  - Reduced (mechanical) seek time

» **CPU speed nowadays makes (de)compression "costless" !!**

Moral:
More economical to store data in compressed form than uncompressed

☑ From March 2001 the Memory eXpansion Technology (MXT) is available on IBM eServers x330MXT

- ✓ Same performance of a PC with double memory but at half cost

Compression and Indexing: Two sides of the same coin !

💣 Do we witness a paradoxical situation ?

- ✓ An index injects redundant data, in order to speed up the pattern searches
- ✓ Compression removes redundancy, in order to squeeze the space occupancy

☺ NO, new results proved a mutual reinforcement behaviour !

- ✓ Better indexes can be designed by exploiting compression techniques
- ✓ Better compressors c… …hniques

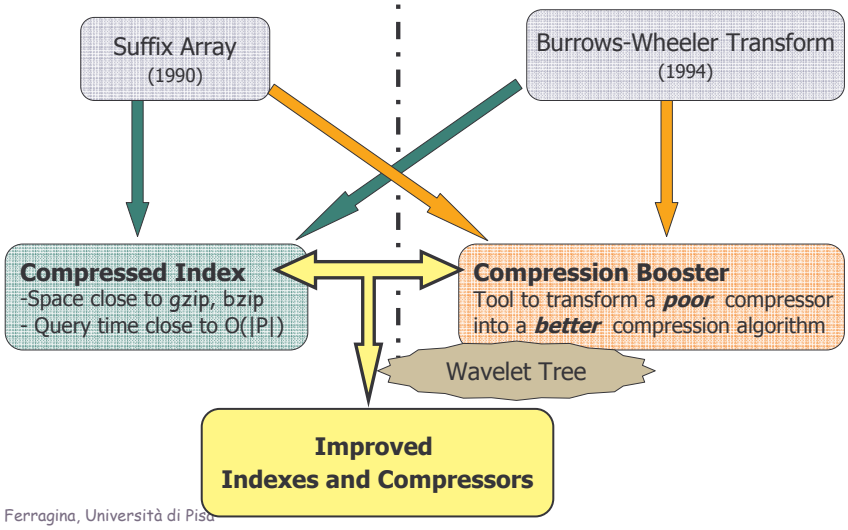
In terms of space occupancy

Also in terms of compression ratio

Moral:
CPM researchers must have a multidisciplinary background, ranging from Data structure design to Data compression, from Architectural Knowledge to Database principles, till Algoritmic Engineering and more…

Our journey, today...

Index design (Weiner '73) | Compressor design (Shannon '48)



The Suffix Array [BaezaYates-Gonnet, 87 and Manber-Myers, 90]

Prop 1. All suffixes in SUF(T) having prefix P are contiguous

Prop 2. These suffixes follow P's lexicographic position

$\Theta(N^2)$ space

T = mississippi# (5)

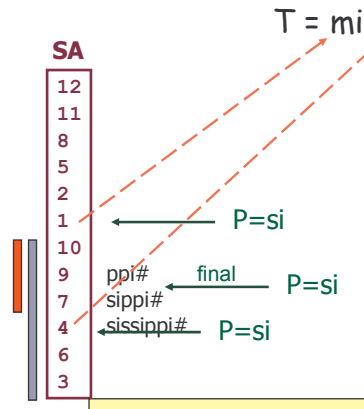
suffix pointer

SA	SUF(T)
12	#
11	i#
8	ippi#
5	issippi#
2	ississippi#
1	mississippi#
10	pi#
9	ppi#
7	sippi#
4	sissippi#
6	ssippi#
3	ssissippi#

P=si

SA + T occupy $\Theta(N \log_2 N)$ bits

Searching in Suffix Array [Manber-Myers, 90]



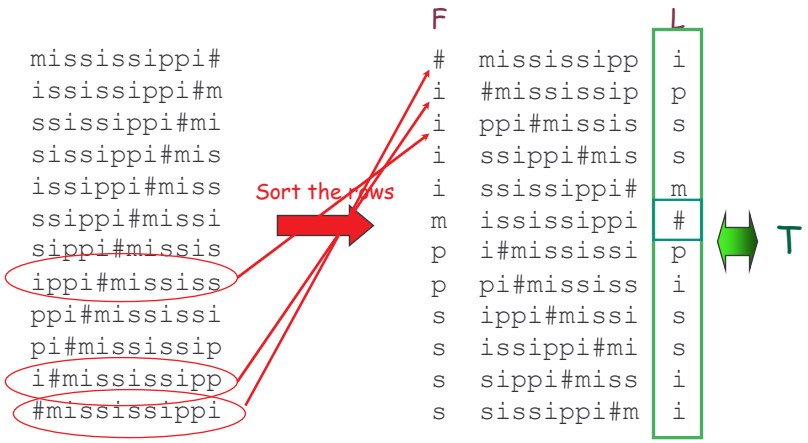
Suffix Array search
- $O(\log_2 N)$ binary-search steps
- Each step takes $O(|P|)$ char cmp
  - ⇨ overall, $O(|P| \log_2 N)$ time

⇨ $O(|P| + \log_2 N)$ time
⇨ $O(|P|/B + \log_B N)$ I/Os [JACM 99]
⇨ Self-adjusting version on disk [FOCS 02]

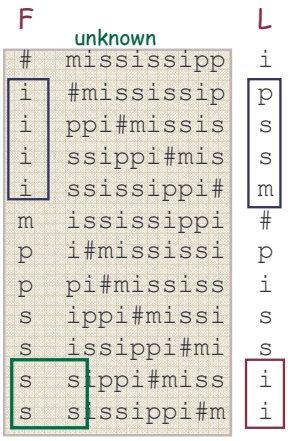
Suffix permutation cannot be any of {1,...,N}
binary texts = 2^N « N! = # permutations on {1, 2, ..., N}
⇨ N $\log_2$ N is not a lower bound to the bit space occupancy ⇦

The Burrows-Wheeler Transform (1994)

Let us given a text T = mississippi#



Why L is so interesting for compression ?



A key observation:

- L is locally homogeneous

⇒ L is highly compressible

Algorithm Bzip :

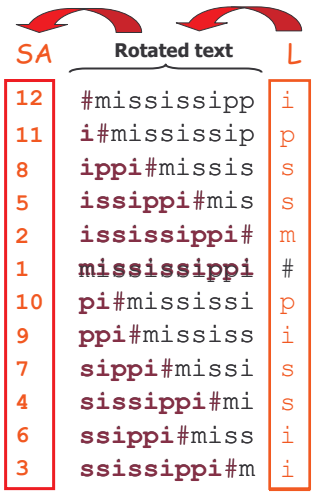
1. Move-to-Front coding of L
2. Run-Length coding
3. Statistical coder: Arithmetic, Huffman

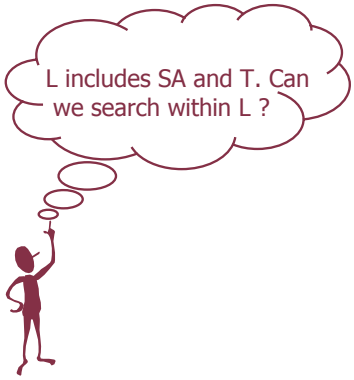
T = m i s s i s s i p p i #

☑ Bzip vs. Gzip: 20% vs. 33%, but it is s

Building the BWT ⇔ SA construction
Inverting the BWT ⇔ array visit
...overall O(N) time...

Suffix Array vs. BW-transform





A compressed index [Ferragina-Manzini, IEEE Focs 2000]

Bridging data-structure design and compression techniques:

- ✓ Suffix array data structure
- ✓ Burrows-Wheeler Transform

The theoretical result:

- ✓ Query complexity: $O(p + occ \log^{\varepsilon} N)$ time
- ✓ Space occupancy: $O(N\, \mathcal{H}_k(T)) + o(N)$ bits → o(N) if T compressible

Index does not depend on k
Bound holds for all k, simultaneously

k-th order empirical entropy

The corollary is that:

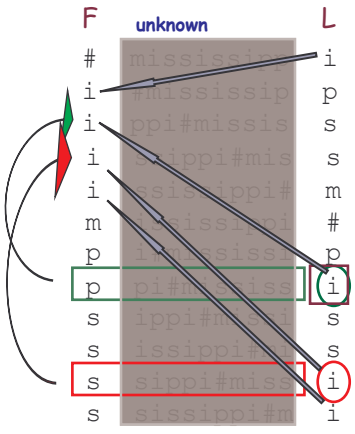
- ✓ The Suffix Array is compressible
- ✓ It is a self-index

O(n) space: A plethora of papers
H_k : Grossi-Gupta-Vitter (03), Sadakane (02),...
Now, more than 20 papers with more than 20 authors on related subjects

In practice, the index is much appealing:

- ✓ Space close to the best known compressors, ie. bzip
- ✓ Query time of few millisecs on hundreds of MBs

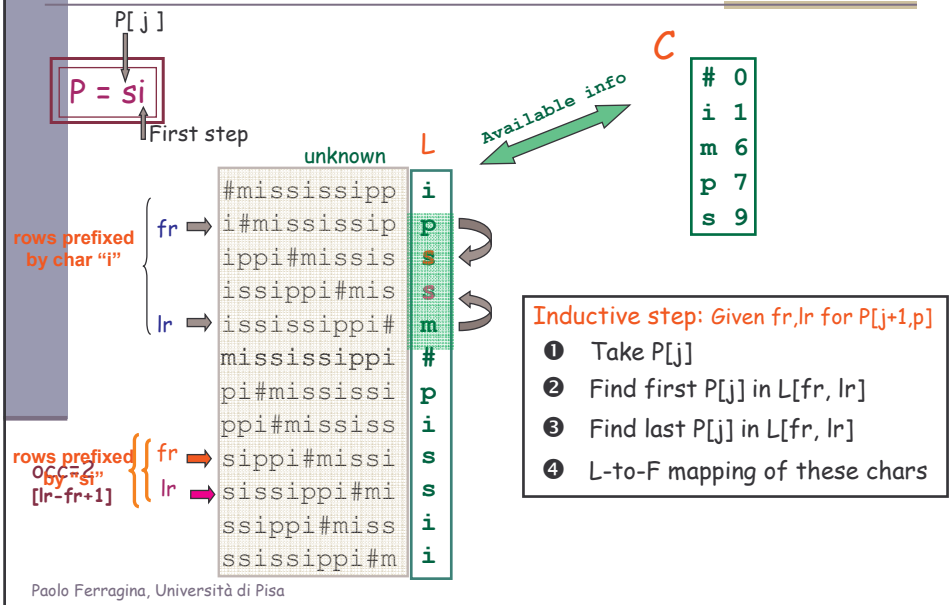
A useful tool: L → F mapping



How do we map L's onto F's chars ?

... Need to distinguish equal chars...

Substring search in T (Count occurrences)



Inductive step: Given fr,lr for P[j+1,p]

1. Take P[j]
2. Find first P[j] in L[fr, lr]
3. Find last P[j] in L[fr, lr]
4. L-to-F mapping of these chars

Many details are missing...

☺ The column L is actually kept compressed:

- Still guarantee O(p) time to count the P's occurrences

☺ The Locate operation takes $O(\log^{\varepsilon} N)$ time

- Some additional data structure, in o(n) bits

☺ Efficient and succinct index construction [Hon et al., Focs 03]

☺ Bio-application: fit Human-genome index in a PC [Sadakane et al., 02]

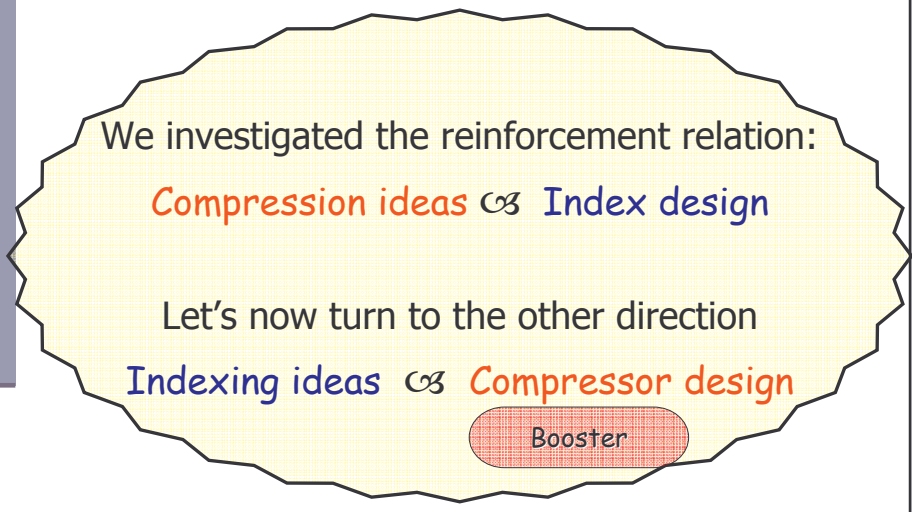
Interesting issues:

☑ What about arbitrary alphabets ? [Grossi et al., 03; Ferragina et al., 04]

☑ What about disk-aware, or cache-oblivious, or self-adjusting versions ?

☑ What about challenging applications: bio,db,data mining,handheld PC, ...

Where we are ...



What do we mean by "boosting" ?

It is a technique that takes a *poor* compressor *A* and turns it into a compressor with *better performance guarantee*

A memoryless compressor is *poor* in that it assigns codewords to symbols according only to their frequencies (e.g. Huffman)

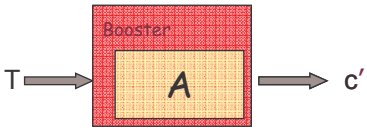
It incurs in some obvious limitations:

$T = a^n b^n$

T'= random string of length 2n and same number of 'a,b'

What do we mean by "boosting" ?

It is a technique that takes a *poor* compressor *A* and turns it into a compressor with *better performance guarantee*



Qualitatively, we would like to achieve

- **c′** is shorter than **c**, if **T** is compr
- Time(A_{boost}) = O(Time(*A*)), i.e. no slowdown
- *A* is used as a black-box

The more compressible is **T**, the shorter is **c′**

Two Key Components: Burrows-Wheeler Transform and Suffix Tree

The emprirical entropy H_0

$$H_0(T) = -\sum_i (n_i/n) \log_2 (n_i/n)$$

Frequency in T of the i-th symbol

- $|T|\, H_0(T)$ is the best you can hope for a memoryless compressor
- E.g. Huffman or Arithmetic coders come close to this bound

We get a better compression using a codeword that depends on the k symbols preceding the one to be compressed **(context)**

The empirical entropy H_k

For any k-long context

✓ Compress T up to $H_k(T)$
≅ compress all $T[\omega]$ up to their H_0

Use Huffman or Arithmetic

$$H_k(T) = (1/|T|) \sum_{|\omega|=k} |T[\omega]| \, H_0(T[\omega])$$

❖ $T[\omega]$ = string of symbols that precede ω in T

Example: Given T = "mississippi", we have T[i]= mssp, T[is] = ms

Problems with this approach:
- How to go from all $T[\omega]$ back to the string **T** ?
- How do we choose efficiently the best **k** ?

BWT

Suffix Tree

Use BWT to approximate H_k

unknown

	bwt(T)
#mississipp	i
i#mississip	p
ippi#missis	s
issippi#mis	s
ississippi#	m
mississippi	#
pi#mississi	p
ppi#mississ	i
sippi#missi	s
sissippi#mi	s
ssippi#miss	i
ssissippi#m	i

Remember that...

$$H_k(T) = (1/|T|) \sum_{|\omega|=k} |T[\omega]| \, H_0(T[\omega])$$

✓ Compress T up to $H_k(T)$
≅ compress all $T[\omega]$ up to their H_0
≅ compress pieces of bwt(T) up to H_0

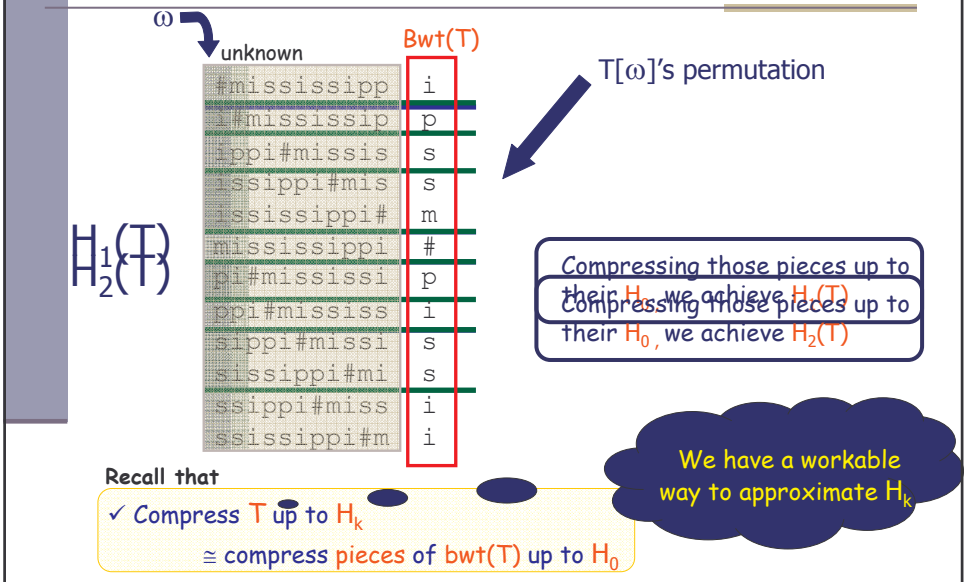
T[is] = ms

ω ω

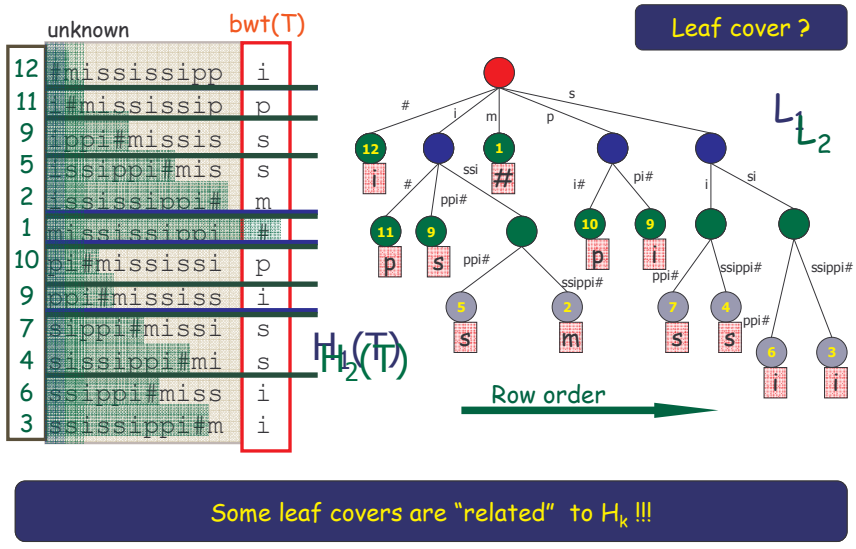
T = mississippi#

$T[\omega]$ is a permutation of a *piece* of bwt(T)

What are the pieces of BWT to compress ?



Finding the "best pieces" to compress...



Some leaf covers are "related" to H_k !!!

A compression booster [Ferragina-Manzini, SODA 04]

- Let A be the compressor we wish to boost
- Let $bwt(T)=t_1, \ldots, t_r$ be the partition induced by the leaf cover L, and let us define $cost(L,A)=\sum_j |A(t_j)|$

Goal: Find the leaf cover L^* of minimum cost

- ✓ It suffices a post-order visit of the suffix tree, hence linear time
- ✓ We have: $Cost(L^*, A) \leq Cost(L_k, A) \approx H_k(T), \ \forall k$

Technically, we show that

$$|c'| \leq \lambda |s| H_0(s) + f(|s|) + \log_2 |s| + \gamma_k \quad \forall k$$

Researchers may now concentrate on the "apparently" simpler task of designing 0-th order compressors

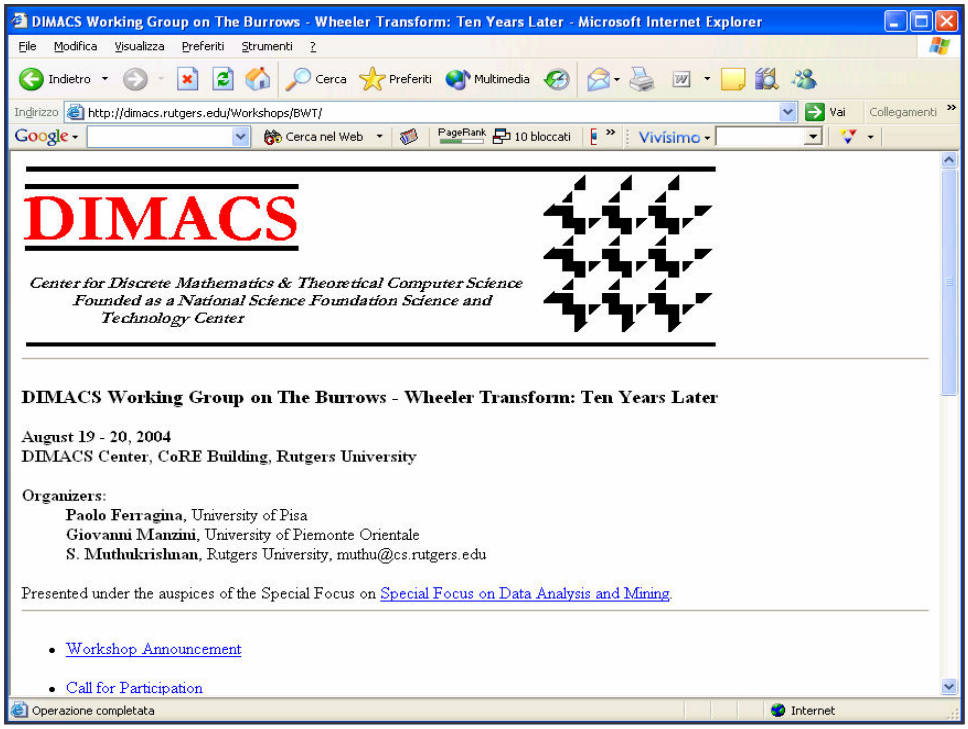
[further results joint with Giancarlo-Manzini-Sciortino]

May we close "the mutual reinforcement cycle" ?

- The Wavelet Tree [Grossi-Gupta-Vitter, Soda 03]
- Using the WT within each piece of the optimal BWT-partition, we get:
  - A compressed index that scales well with the alphabet size
    [joint with Manzini, Makinen, Navarro]
  - Reduce the compression problem to achieve H_0 on binary strings
    [joint with Giancarlo, Manzini,Sciortino]

Interesting issues:

- ☑ What about space construction of BWT ?
- ☑ What about these tools for "XML" or "Images" ?
- ☑ Other application contexts: bio,db,data mining, network, ...
- ☑ From Theory to Technology ! Libraries, Libraries,.... [e.g. LEDA]


DIMACS Working Group on The Burrows - Wheeler Transform: Ten Years Later - Microsoft Internet Explorer
File Modifica Visualizza Preferiti Strumenti ?
Indietro Cerca Preferiti Multimedia
Indirizzo http://dimacs.rutgers.edu/Workshops/BWT/
Vai Collegamenti
Google Cerca nel Web PageRank 10 bloccati Vivisimo
DIMACS
Center for Discrete Mathematics & Theoretical Computer Science
Founded as a National Science Foundation Science and
Technology Center
DIMACS Working Group on The Burrows - Wheeler Transform: Ten Years Later
August 19 - 20, 2004
DIMACS Center, CoRE Building, Rutgers University
Organizers:
Paolo Ferragina, University of Pisa
Giovanni Manzini, University of Piemonte Orientale
S. Muthukrishnan, Rutgers University, muthu@cs.rutgers.edu
Presented under the auspices of the Special Focus on Special Focus on Data Analysis and Mining.
Workshop Announcement
Call for Participation
Operazione completata
Internet