On searching and extracting strings from compressed textual data

Rossano Venturini

University of Pisa

Supervisor
Paolo Ferragina

Referees
Ricardo Baeza-Yates
Jeffrey Vitter

Data Structures

Data Compression

Data Structures

+ Fast queries
- Add extra data

Data Compression

Data Structures

+ Fast queries
- Add extra data

Data Compression

+ Space efficiency
- Slow access to data

Data Structures

+ Fast queries
- Add extra data

&

Data Compression

+ Space efficiency
- Slow access to data

Data Structures

\+ Fast queries

\- Add extra data

&

Data Compression

\+ Space efficiency

\- Slow access to data

Fit as much as data as possible in
the higher levels of memory hierarchy

Google

diego armando maradona

- diego armando maradona
- diego armando
- diego armando maradona biography
- diego armus

Search

Web
Images
Maps
Videos
News
Shopping
More

Show search tools

Diego Maradona - Wikipedia, the free encyclopedia
en.wikipedia.org/wiki/**Diego_Maradona**
Diego Armando Maradona Franco (Spanish pronunciation: [ˈdjeɣo maraˈðona]; born 30 October 1960) is a retired Argentine football player, and current ...
↳ Al Wasl FC - Argentina 2–1 England (1986 ... - Sergio Agüero - Diego Sinagra

Images for **diego armando maradona** - Report images

Diego Armando Maradona | Facebook
www.facebook.com/pages/**Diego-Armando-Maradona**/14650999732
Diego Armando Maradona is on Facebook. To connect with **Diego Armando Maradona**, sign up for Facebook today. Sign UpLog In · Like · Profile Picture ...

MARADONA'S TRICKS - YouTube

www.youtube.com/watch?v=i_PP7QJwNpM
30 Oct 2006 - 4 min - Uploaded by maverenzo
Diego Maradona - The Best Of El Pibe de Oro 8:45. Watch Later Error **Diego Maradona** - The Best Of El ...

Diego Armando Maradona - YouTube

www.youtube.com/watch?v=cx0a2ykZdSc
28 Feb 2012 - 20 min - Uploaded by wnapoli2012
IL CALCIO : **DIEGO ARMANDO MARADONA** . È considerato il miglior calciatore di tutti i tempi , ha ...

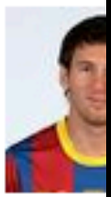
More videos for **diego armando maradona** »

Diego Maradona

sitesmexico.com

People also search f

Pelé



Lionel Messi

Google

diego armando maradona

diego armando maradona
diego armando
diego armand
diego armus

A prefix search on a dictionary of strings containing past queries

Search

Web
Images
Maps
Videos
News
Shopping
More

Show search tools

Diego Marado
en.wikipedia.org/wi
Diego Armando Maradona Franco (Spanish pronunciation: [ˈdjeɣo maraˈðona]; born 30 October 1960) is a retired Argentine football player, and current ...
↳ Al Wasl FC - Argentina 2–1 England (1986 ... - Sergio Agüero - Diego Sinagra

Images for **diego armando maradona** - Report images

Diego Armando Maradona | Facebook
www.facebook.com/pages/**Diego-Armando-Maradona**/14650999732
Diego Armando Maradona is on Facebook. To connect with **Diego Armando Maradona**, sign up for Facebook today. Sign UpLog In · Like · Profile Picture ...

MARADONA'S TRICKS - YouTube


www.youtube.com/watch?v=i_PP7QJwNpM
30 Oct 2006 - 4 min - Uploaded by maverenzo
Diego Maradona - The Best Of El Pibe de Oro 8:45. Watch Later Error **Diego Maradona** - The Best Of El ...

Diego Armando Maradona - YouTube


www.youtube.com/watch?v=cx0a2ykZdSc
28 Feb 2012 - 20 min - Uploaded by wnapoli2012
IL CALCIO : **DIEGO ARMANDO MARADONA** . È considerato il miglior calciatore di tutti i tempi , ha ...

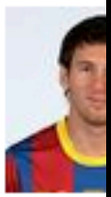
More videos for **diego armando maradona** »

go Maradona

sitesmexico.com

People also search f

Pelé



Lionel Messi

Google

diego armando maradona

diego armando maradona
diego armando
diego armand
diego armus

Search

A prefix search on a dictionary of strings containing past queries

Trie is the classical efficient solution.
Each leaf corresponds to a query.

Web
Images
Maps
Videos
News
Shopping
More

Show search tools

Diego Marado
en.wikipedia.org/wi
Diego Armando Maradona Franc (Spanish pronunciation: ['djeyo mara'ðona]; born 30 October 1960) is a retired Argen
↳ Al Wasl FC - Argentina 2–1 E

go Maradona

Images for **diego armand**

Diego Armando Maradona | Facebook
www.facebook.com/pages/**Diego-Armando-Maradona**/14650999732
Diego Armando Maradona is on Facebook. To connect with **Diego Armando Maradona**, sign up for Facebook today. Sign UpLog In · Like · Profile Picture ...

MARADONA'S TRICKS - YouTube

www.youtube.com/watch?v=i_PP7QJwNpM
30 Oct 2006 - 4 min - Uploaded by maverenzo
Diego Maradona - The Best Of El Pibe de Oro 8:45. Watch Later Error **Diego Maradona** - The Best Of El ...

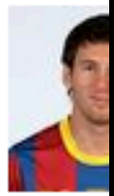
Diego Armando Maradona - YouTube

www.youtube.com/watch?v=cx0a2ykZdSc
28 Feb 2012 - 20 min - Uploaded by wnapoli2012
IL CALCIO : **DIEGO ARMANDO MARADONA** . È considerato il miglior calciatore di tutti i tempi , ha ...

More videos for **diego armando maradona** »

People also search f


Pelé


Lionel Messi

Google

diego armando maradona

diego armando maradona
diego armando
diego armand
diego armus

Search

Web
Images
Maps
Videos
News
Shopping
More

Show search tools

A prefix search on a dictionary of strings containing past queries

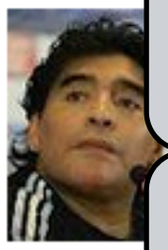
Trie is the classical efficient solution.
Each leaf corresponds to a query.

A search for each symbol we digit
=>
Trie must fit in RAM

Diego Marado
en.wikipedia.org/wi
Diego Armando Maradona Franc
October 1960) is a retired Argen
↳ Al Wasl FC - Argentina 2–1 E

Images for **diego armand**





Diego Armando Marador
www.facebook.com/pages/Dieg
Diego Armando Maradona is
Maradona, sign up for Facebook

MARADONA'S TRICKS - YouTube

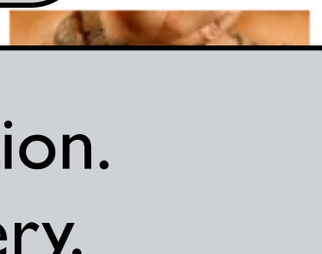
www.youtube.com/watch?v=i_PP7QJwNpM
30 Oct 2006 - 4 min - Uploaded by maverenzo
Diego Maradona - The Best Of El Pibe de Oro 8:45. Watch Later
Error **Diego Maradona** - The Best Of El ...

Diego Armando Maradona - YouTube

www.youtube.com/watch?v=cx0a2ykZdSc
28 Feb 2012 - 20 min - Uploaded by wnapoli2012
IL CALCIO : **DIEGO ARMANDO MARADONA** . È considerato il
miglior calciatore di tutti i tempi , ha ...

More videos for **diego armando maradona** »

go Maradona




Pelé


Lionel Messi

Google

diego armando maradona

diego armando maradona
diego armando
diego armand
diego armus

Search

Web
Images
Maps

A prefix search on a dictionary of strings containing past queries

153,645,450
distinct queries
in 6 weeks Altavista
query log

Trie is the classical efficient solution.
Each leaf corresponds to a query.

A search for each symbol we digit
=>
Trie must fit in RAM

Diego Marado
en.wikipedia.org/wi
Diego Armando Maradona Franc
October 1960) is a retired Argen
Al Wasl FC - Argentina 2–1 E

r diego armand

rmando Marador
www.facebook.com/pages/Dieg
Diego Armando Maradona is c
Maradona, sign up for Facebook

MARADONA'S TRICKS - YouTube
www.youtube.com/watch?v=i_PP7QJwNpM
30 Oct 2006 - 4 min - Uploaded by maverenzo
Diego Maradona - The Best Of El Pibe de Oro 8:45. Watch Later
Error **Diego Maradona** - The Best Of El ...

Diego Armando Maradona - YouTube
www.youtube.com/watch?v=cx0a2ykZdSc
28 Feb 2012 - 20 min - Uploaded by wnapoli2012
IL CALCIO : **DIEGO ARMANDO MARADONA** . È considerato il
miglior calciatore di tutti i tempi , ha ...

More videos for **diego armando maradona** »

go Maradona

Pelé

Lionel
Messi

Google

diego armando maradona

diego armando maradona
diego armando
diego armand
diego armus

Search

Web
Images
Maps

A prefix search on a dictionary of strings containing past queries

153,645,450
distinct queries
in 6 weeks Altavista
query log

Trie is the classical efficient solution.
Each leaf corresponds to a query.

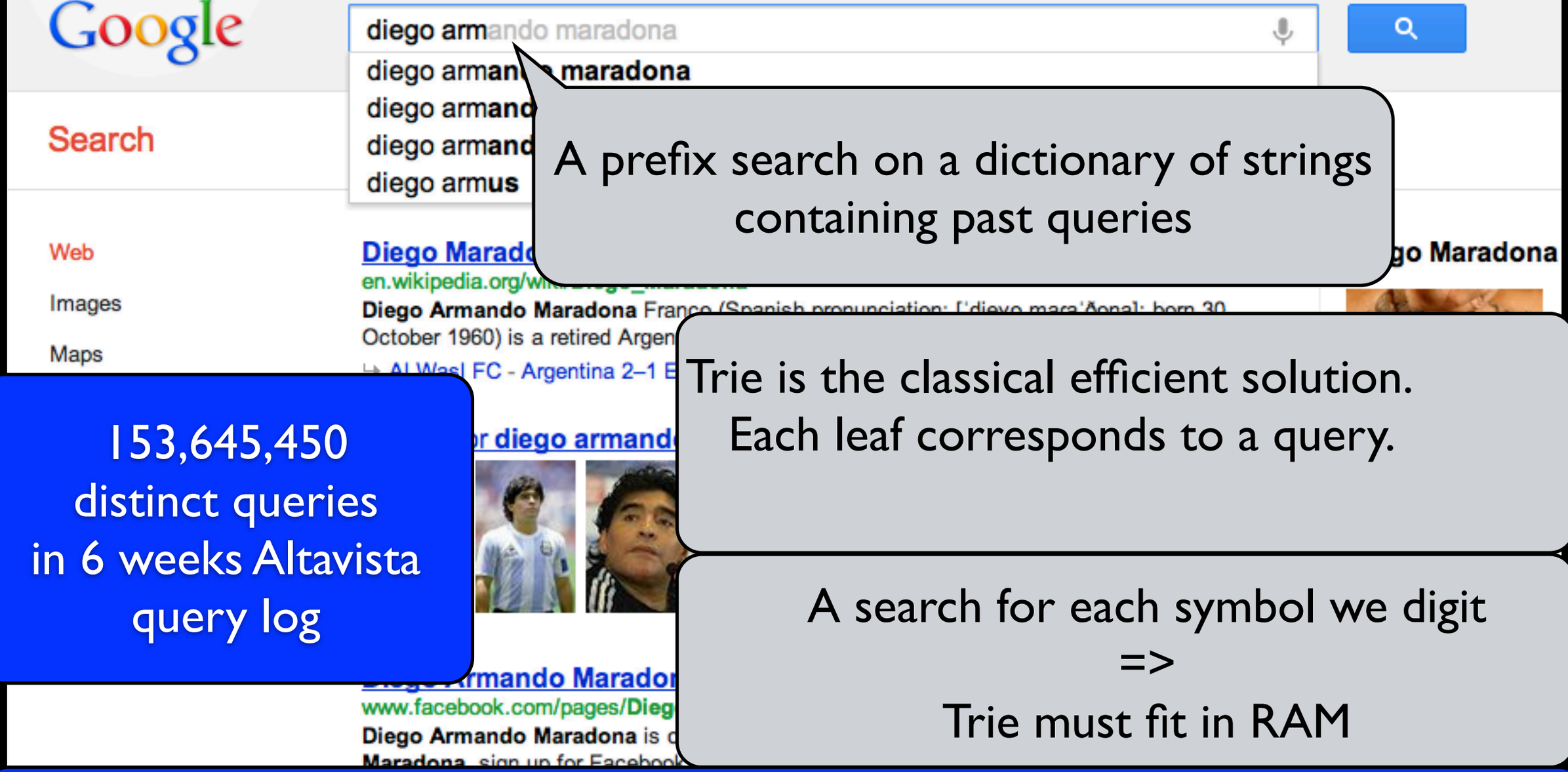
A search for each symbol we digit
=>
Trie must fit in RAM

Tree + Edges labels + Text ≈ 12 Gbytes
(assuming queries of 5 symbols each)

Diego Armando Maradona - YouTube

www.youtube.com/watch?v=cx0a2ykZdSc
28 Feb 2012 - 20 min - Uploaded by wnapoli2012
IL CALCIO : **DIEGO ARMANDO MARADONA** . È considerato il miglior calciatore di tutti i tempi , ha ...

More videos for **diego armando maradona** »



A prefix search on a dictionary of strings containing past queries

153,645,450 distinct queries in 6 weeks Altavista query log

Trie is the classical efficient solution.
Each leaf corresponds to a query.

A search for each symbol we digit
=>
Trie must fit in RAM

Tree + Edges labels + Text ≈ 12 Gbytes
(assuming queries of 5 symbols each)

Compressed Tree + Compressed text with fast decompression < 800 Mbytes

Active field of research

Integers

FOCS, 1989
FOCS, 1997
SODA, 2002
FOCS, 2008
SODA, 2010

Active field of research

Integers	Text
FOCS, 1989	STOC, 2000
FOCS, 1997	FOCS, 2000
SODA, 2002	SODA, 2001
FOCS, 2008	SODA, 2007
SODA, 2010	PODS, 2011

Active field of research

Integers	Text	Trees
FOCS, 1989	STOC, 2000	FOCS, 1989
FOCS, 1997	FOCS, 2000	FOCS, 1997
SODA, 2002	SODA, 2001	SODA, 2002
FOCS, 2008	SODA, 2007	SODA, 2007
SODA, 2010	PODS, 2011	SODA, 2010

Active field of research

Integers	Text	Trees	Graphs
FOCS, 1989	STOC, 2000	FOCS, 1989	FOCS, 1997
FOCS, 1997	FOCS, 2000	FOCS, 1997	DCC, 2001
SODA, 2002	SODA, 2001	SODA, 2002	WWW, 2004
FOCS, 2008	SODA, 2007	SODA, 2007	ESA, 2008
SODA, 2010	PODS, 2011	SODA, 2010	FOCS, 2009

Active field of research

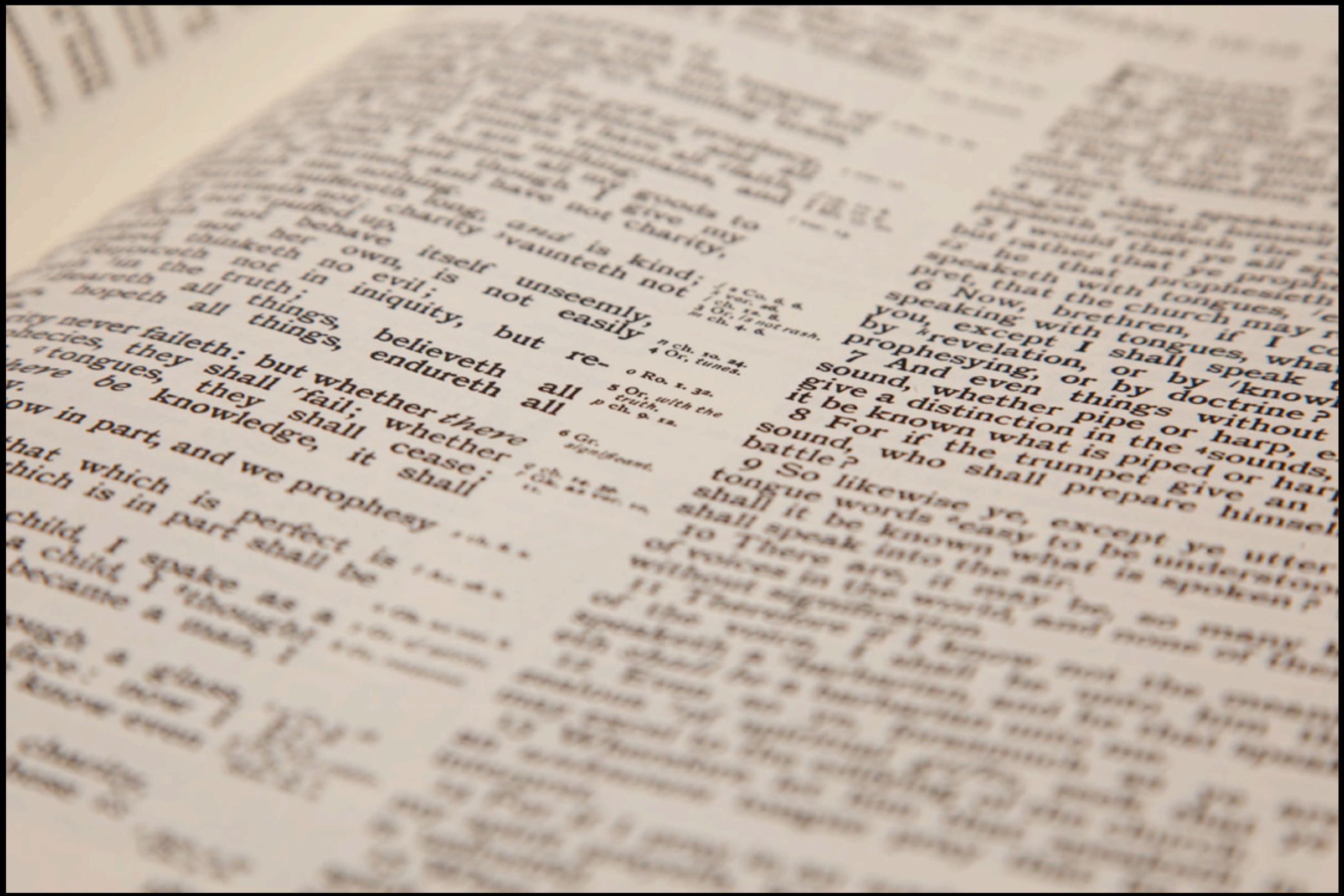
Integers	Text	Trees	Graphs
FOCS, 1989	STOC, 2000	FOCS, 1989	FOCS, 1997
FOCS, 1997	FOCS, 2000	FOCS, 1997	DCC, 2001
SODA, 2002	SODA, 2001	SODA, 2002	WWW, 2004
FOCS, 2008	SODA, 2007	SODA, 2007	ESA, 2008
SODA, 2010	PODS, 2011	SODA, 2010	FOCS, 2009

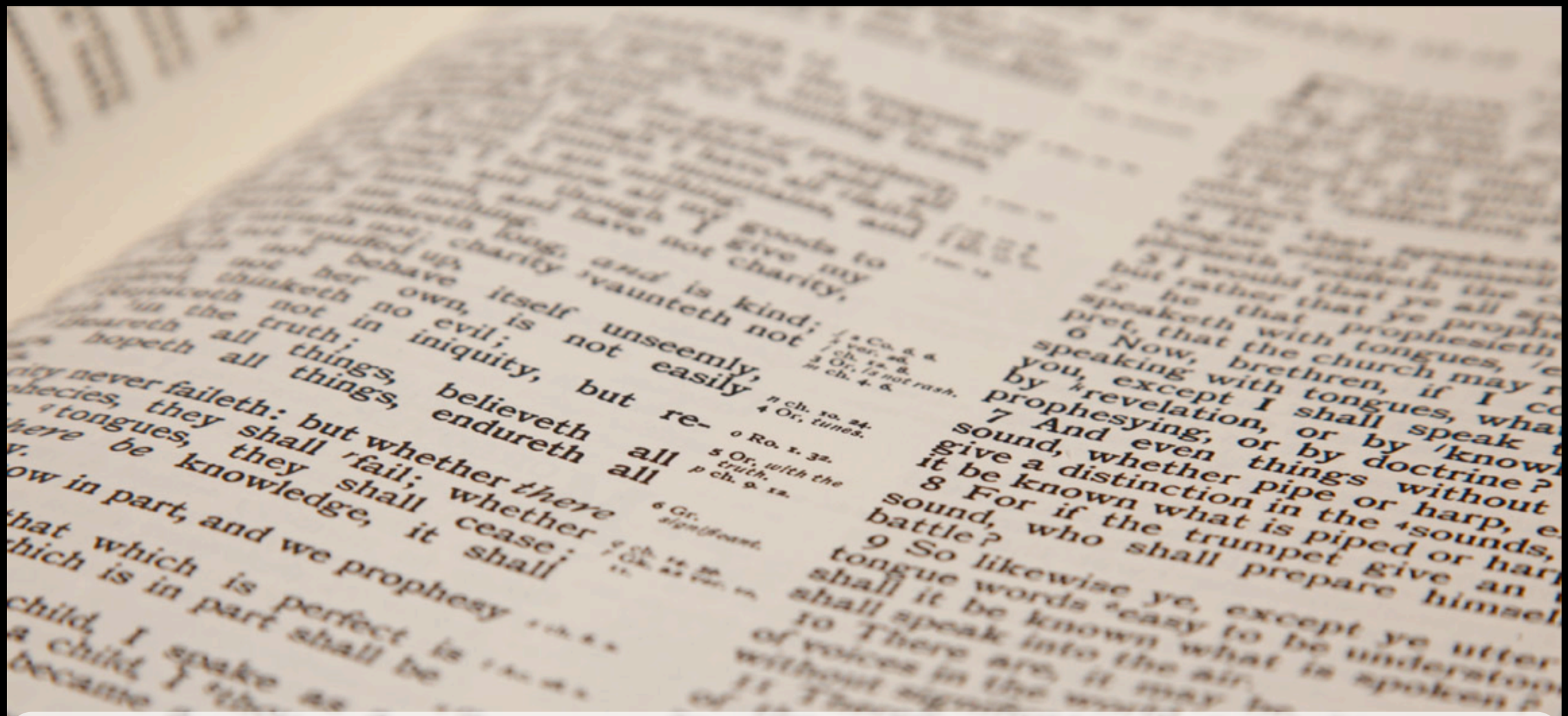
Labeled Trees e.g. XML	Functions	Point Sets	Hashing
FOCS, 2005	ICALP, 2003	SODA, 2003	SODA, 2004
WWW, 2006	ICALP, 2004	TALG, 2007	SODA, 2009
SODA, 2007	SODA, 2004	WADS, 2009	ESA, 2009
	ICALP, 2008	SODA, 2011	ICALP, 2009
	ESA, 2009	SOCG, 2011	SODA, 2013

Active field of research

Integers	Text	Trees	Graphs
FOCS, 1989 FOCS, 1997 SODA, 2002 FOCS, 2008 SODA, 2010	STOC, 2000 FOCS, 2000 SODA, 2001 SODA, 2007 PODS, 2011	FOCS, 1989 FOCS, 1997 SODA, 2002 SODA, 2007 SODA, 2010	FOCS, 1997 DCC, 2001 WWW, 2004 ESA, 2008 FOCS, 2009

Labeled Trees e.g. XML	Functions	Point Sets	Hashing
FOCS, 2005 WWW, 2006 SODA, 2007	ICALP, 2003 ICALP, 2004 SODA, 2004 ICALP, 2008 ESA, 2009	SODA, 2003 TALG, 2007 WADS, 2009 SODA, 2011 SOCG, 2011	SODA, 2004 SODA, 2009 ESA, 2009 ICALP, 2009 SODA, 2013





Maximize Compression and Efficiently Access and Search

Compressor's performance optimization

with P. Ferragina, I. Nitto, ESA 2009 / Algorithmica 2011

- Input: a text T[1,n] and ANY compressor C

- Goal: an optimal partition of T in blocks such that compress size achieved by compressing them individually with C is better than that of the whole T

Why?

Why?

$$c \cdot n \cdot H_0(T) + f(n, \Sigma) \text{ bits}$$

e.g., Arithmetic $\quad n \cdot H_0(T) + O(\mid \Sigma \mid \log n) \text{ bits}$

Why?

$$c \cdot n \cdot H_0(T) + f(n, \Sigma) \text{ bits}$$

e.g., Arithmetic $\quad n \cdot H_0(T) + O(\mid \Sigma \mid \log n) \text{ bits}$

T

A	A	A	A	A	A	A	A	B	B	B	B	B	B	B	B

Why?

$c \cdot n \cdot H_0(T) + f(n, \Sigma)$ bits

e.g., Arithmetic $n \cdot H_0(T) + O(\mid \Sigma \mid \log n)$ bits

T

Why?

$c \cdot n \cdot H_0(T) + f(n, \Sigma)$ bits

e.g., Arithmetic $n \cdot H_0(T) + O(\mid \Sigma \mid \log n)$ bits

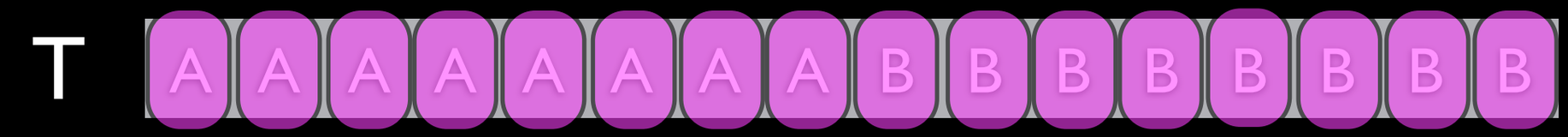
T A A A A A A A A B B B B B B B B

n + log n bits

Why?

$c \cdot n \cdot H_0(T) + f(n, \Sigma)$ bits

e.g., Arithmetic $n \cdot H_0(T) + O(\mid \Sigma \mid \log n)$ bits

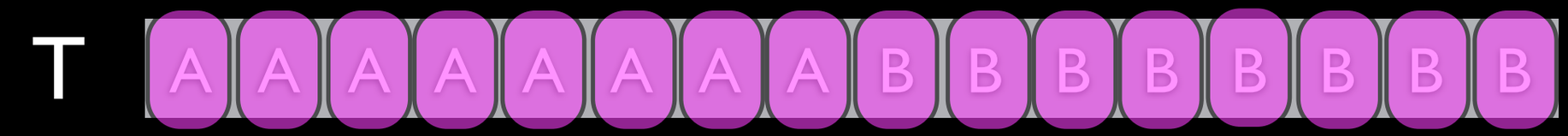


n + log n bits

Why?

$c \cdot n \cdot H_0(T) + f(n, \Sigma)$ bits

e.g., Arithmetic $n \cdot H_0(T) + O(\mid \Sigma \mid \log n)$ bits



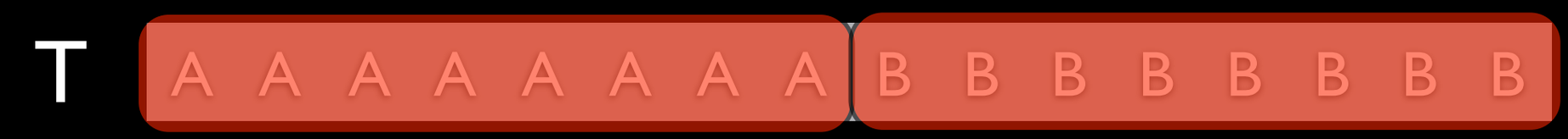
n + log n bits

n log n bits

Why?

$c \cdot n \cdot H_0(T) + f(n, \Sigma)$ bits

e.g., Arithmetic $n \cdot H_0(T) + O(\mid \Sigma \mid \log n)$ bits



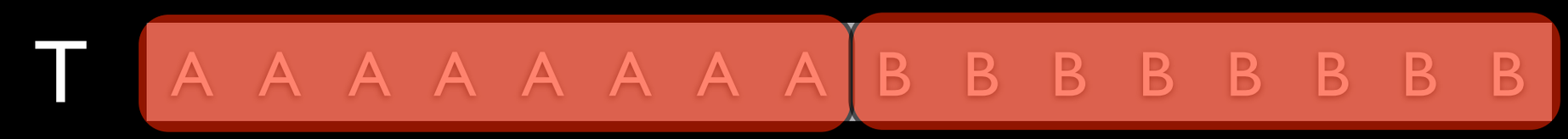
n + log n bits

n log n bits

Why?

$c \cdot n \cdot H_0(T) + f(n, \Sigma)$ bits

e.g., Arithmetic $n \cdot H_0(T) + O(\mid \Sigma \mid \log n)$ bits



n + log n bits

n log n bits

0 + 2 log n bits

Result

with P. Ferragina, I. Nitto, ESA 2009 / Algorithmica 2011

Result

with P. Ferragina, I. Nitto, ESA 2009 / Algorithmica 2011

- Optimal partition can be found with Dynamic Programming in $O(n^2)$ time
  - Not usable in practice for texts longer than few Mbs

Result

with P. Ferragina, I. Nitto, ESA 2009 / Algorithmica 2011

- Optimal partition can be found with Dynamic Programming in $O(n^2)$ time
  - Not usable in practice for texts longer than few Mbs
- For any fixed parameter $\epsilon>0$, our algorithm computes an $(1+\epsilon)$-approximation of the optimal partition in $O(n \log_{1+\epsilon} n)$ time and linear space

Result

with P. Ferragina, I. Nitto, ESA 2009 / Algorithmica 2011

- Optimal partition can be found with Dynamic Programming in $O(n^2)$ time
  - Not usable in practice for texts longer than few Mbs
- For any fixed parameter $\epsilon>0$, our algorithm computes an $(1+\epsilon)$-approximation of the optimal partition in $O(n \log_{1+\epsilon} n)$ time and linear space
- The idea is exploiting a particular property of the cost function to speed up Dynamic programming solution
  - The property (monotonicity) is quite common. Thus, the idea can be used in other contexts.

Lempel-Ziv 77

with P. Ferragina, I. Nitto, SODA 2009

- Input: a text T[1,n]
- Goal: find the optimal LZ77 parsing of T (i.e., the parsing that minimizes the compress size)

Lempel-Ziv 77

with P. Ferragina, I. Nitto, SODA 2009

Lempel-Ziv 77

with P. Ferragina, I. Nitto, SODA 2009

Many implementations: gzip, arj, .gif, jar, pkzip, compress, lzma, ...

Lempel-Ziv 77

with P. Ferragina, I. Nitto, SODA 2009

Many implementations: gzip, arj, .gif, jar, pkzip, compress, lzma, ...

Parse input text from left to right splitting it into phrases.
A phrase is either a single symbol or a repetition in the already parsed part.

Lempel-Ziv 77

with P. Ferragina, I. Nitto, SODA 2009

Many implementations: gzip, arj, .gif, jar, pkzip, compress, lzma, ...

Parse input text from left to right splitting it into phrases.
A phrase is either a single symbol or a repetition in the already parsed part.

Greedy strategy always selects the longest repetition

Lempel-Ziv 77

with P. Ferragina, I. Nitto, SODA 2009

Many implementations: gzip, arj, .gif, jar, pkzip, compress, lzma, ...

Parse input text from left to right splitting it into phrases.
A phrase is either a single symbol or a repetition in the already parsed part.

Greedy strategy always selects the longest repetition

Backward-References: (distance, length) or (0,c) if single symbol

Lempel-Ziv 77

with P. Ferragina, I. Nitto, SODA 2009

Many implementations: gzip, arj, .gif, jar, pkzip, compress, lzma, ...

Parse input text from left to right splitting it into phrases.
A phrase is either a single symbol or a repetition in the already parsed part.

Greedy strategy always selects the longest repetition

Backward-References: (distance, length) or (0,c) if single symbol

Lempel-Ziv 77

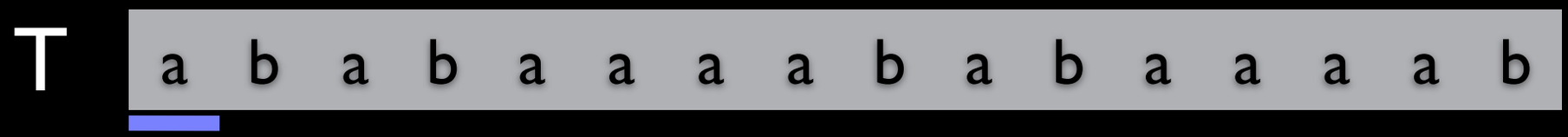
with P. Ferragina, I. Nitto, SODA 2009

Many implementations: gzip, arj, .gif, jar, pkzip, compress, lzma, ...

Parse input text from left to right splitting it into phrases.
A phrase is either a single symbol or a repetition in the already parsed part.

Greedy strategy always selects the longest repetition

Backward-References: (distance, length) or (0,c) if single symbol



(0,a)

Lempel-Ziv 77

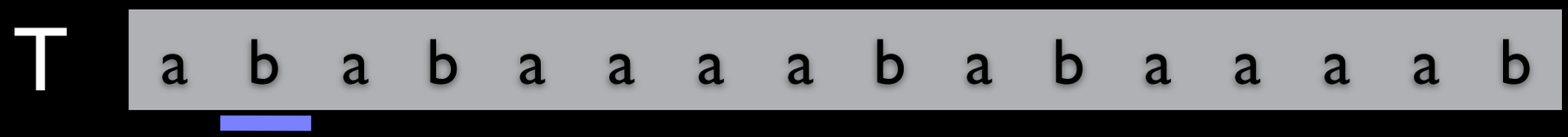
with P. Ferragina, I. Nitto, SODA 2009

Many implementations: gzip, arj, .gif, jar, pkzip, compress, lzma, ...

Parse input text from left to right splitting it into phrases.
A phrase is either a single symbol or a repetition in the already parsed part.

Greedy strategy always selects the longest repetition

Backward-References: (distance, length) or (0,c) if single symbol



(0,a) (0,b)

Lempel-Ziv 77

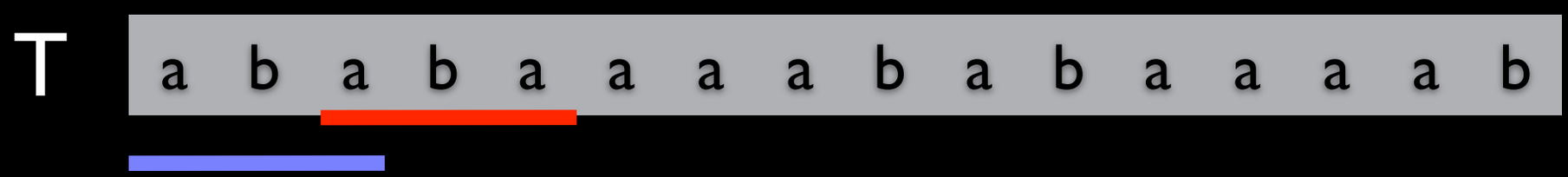
with P. Ferragina, I. Nitto, SODA 2009

Many implementations: gzip, arj, .gif, jar, pkzip, compress, lzma, ...

Parse input text from left to right splitting it into phrases.
A phrase is either a single symbol or a repetition in the already parsed part.

Greedy strategy always selects the longest repetition

Backward-References: (distance, length) or (0,c) if single symbol



(0,a) (0,b) (2,3)

Lempel-Ziv 77

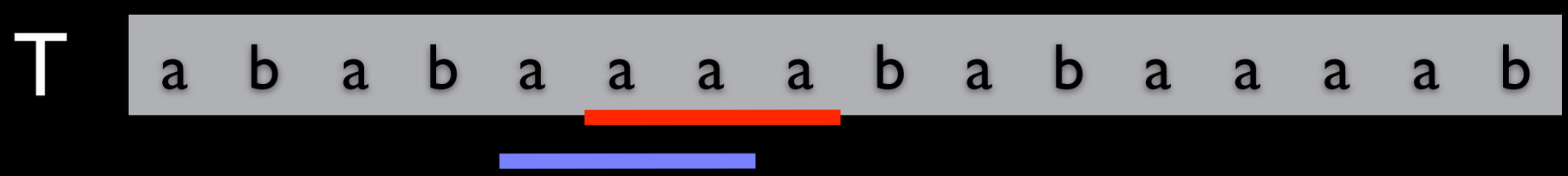
with P. Ferragina, I. Nitto, SODA 2009

Many implementations: gzip, arj, .gif, jar, pkzip, compress, lzma, ...

Parse input text from left to right splitting it into phrases.
A phrase is either a single symbol or a repetition in the already parsed part.

Greedy strategy always selects the longest repetition

Backward-References: (distance, length) or (0,c) if single symbol



(0,a) (0,b) (2,3) (1,3)

Lempel-Ziv 77

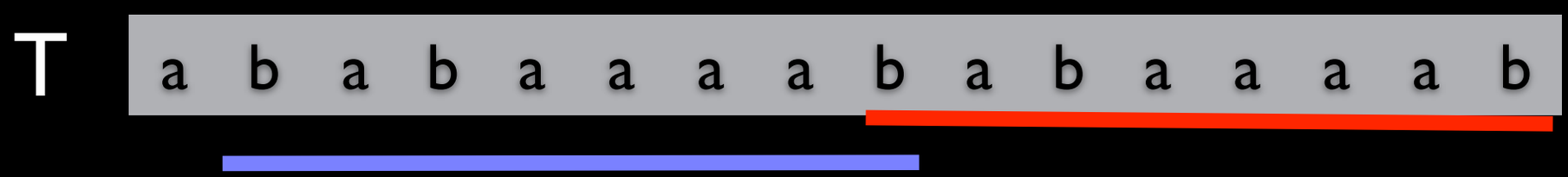
with P. Ferragina, I. Nitto, SODA 2009

Many implementations: gzip, arj, .gif, jar, pkzip, compress, lzma, ...

Parse input text from left to right splitting it into phrases.
A phrase is either a single symbol or a repetition in the already parsed part.

Greedy strategy always selects the longest repetition

Backward-References: (distance, length) or (0,c) if single symbol



(0,a) (0,b) (2,3) (1,3) (7,8)

Lempel-Ziv 77

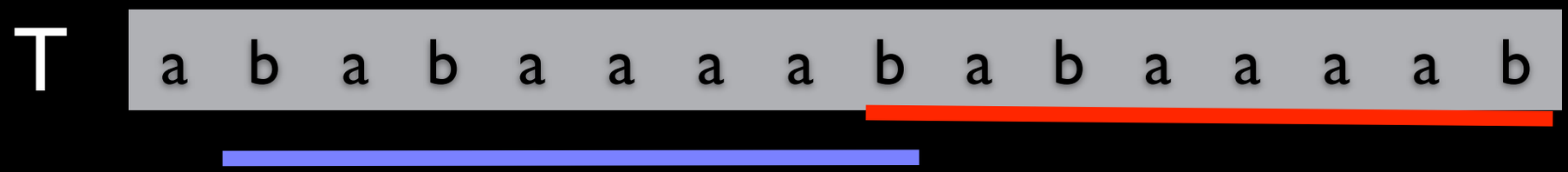
with P. Ferragina, I. Nitto, SODA 2009

Many implementations: gzip, arj, .gif, jar, pkzip, compress, lzma, ...

Parse input text from left to right splitting it into phrases.
A phrase is either a single symbol or a repetition in the already parsed part.

Greedy strategy always selects the longest repetition

Backward-References: (distance, length) or (0,c) if single symbol



(0,a) (0,b) (2,3) (1,3) (7,8)

Encode distances and lengths with two uniquely decodable encoding functions

$f,g:[n] \rightarrow \{0,1\}^*$

Lempel-Ziv 77

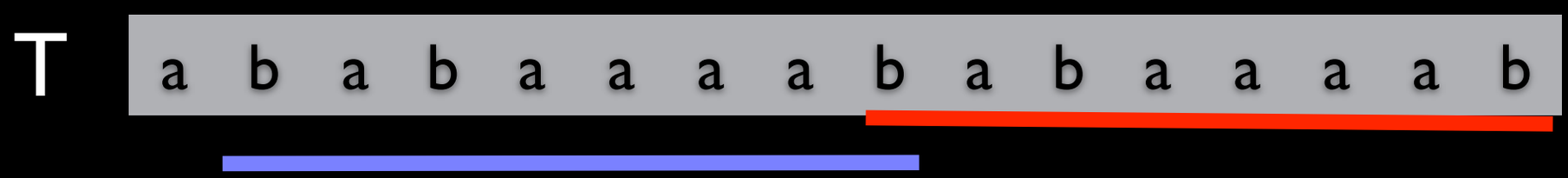
with P. Ferragina, I. Nitto, SODA 2009

Many implementations: gzip, arj, .gif, jar, pkzip, compress, lzma, ...

Parse input text from left to right splitting it into phrases.
A phrase is either a single symbol or a repetition in the already parsed part.

Greedy strategy always selects the longest repetition

Backward-References: (distance, length) or (0,c) if single symbol



(0,a) (0,b) (2,3) (1,3) (7,8)

Encode distances and lengths with two uniquely decodable encoding functions

$f,g:[n] \rightarrow \{0,1\}^*$

Decompression is simple and efficient in practice

Greedy strategy wastes space

T … abacdde ……..... abae .... cdde ..... abacddf ...

Greedy strategy wastes space

Assume any value j is encoded by f() with log j bits

T … abacdde …..... abae .... cdde ..... abacddf ...

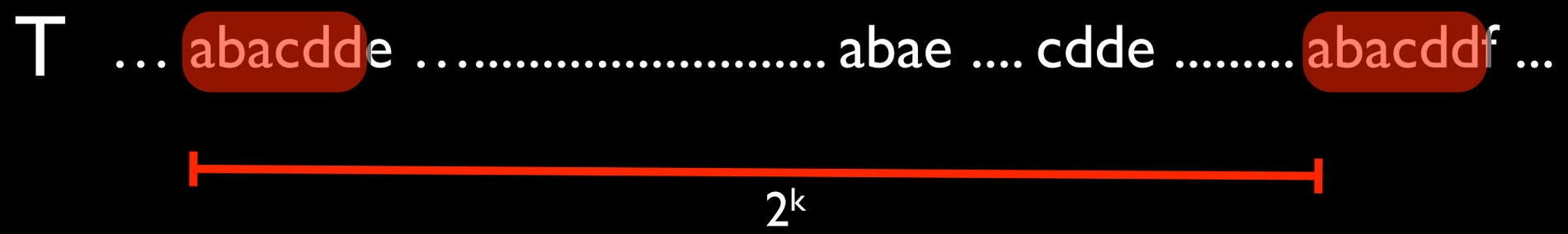
Greedy strategy wastes space

Assume any value j is encoded by f() with log j bits

T … abacdde …..... abae .... cdde ..... abacddf ...

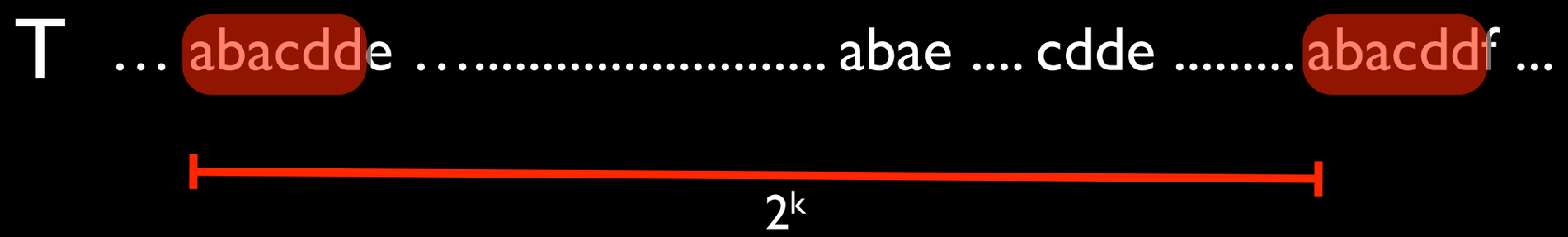
Greedy strategy wastes space

Assume any value j is encoded by f() with log j bits



Greedy strategy wastes space

Assume any value j is encoded by f() with log j bits



Cost = $|f(2^k)|$ = log 2^k = k bits

Greedy strategy wastes space

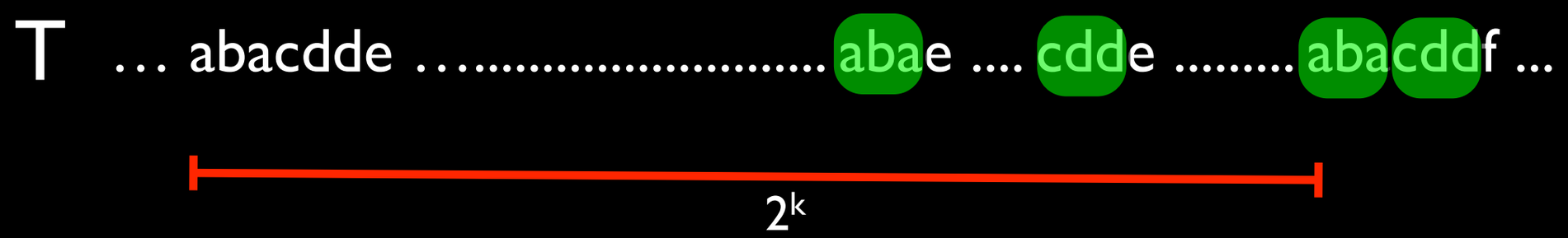
Assume any value j is encoded by f() with log j bits

T … abacdde ……..... abae .... cdde ..... abacddf ...

Cost = $|f(2^k)|$ = log 2^k = k bits

Greedy strategy wastes space

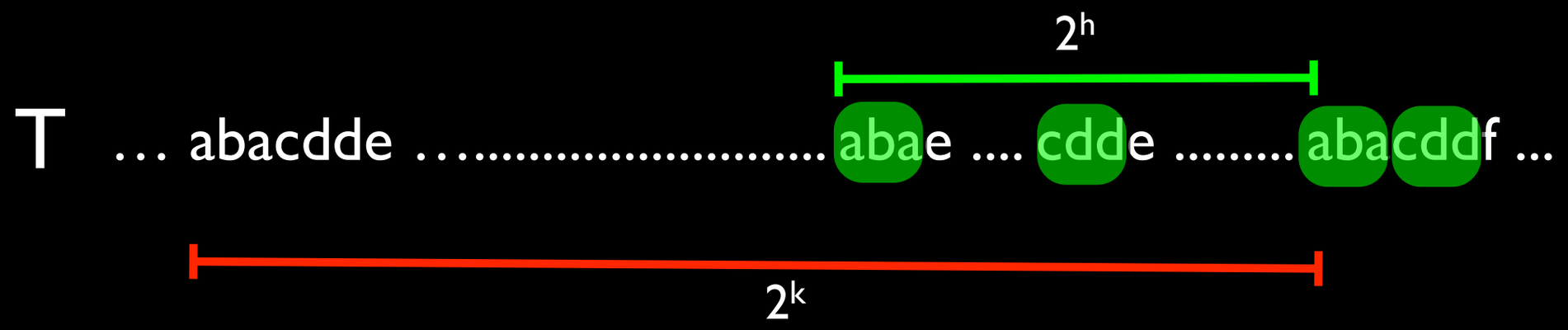
Assume any value j is encoded by f() with log j bits



Cost = $|f(2^k)|$ = log 2^k = k bits

Greedy strategy wastes space

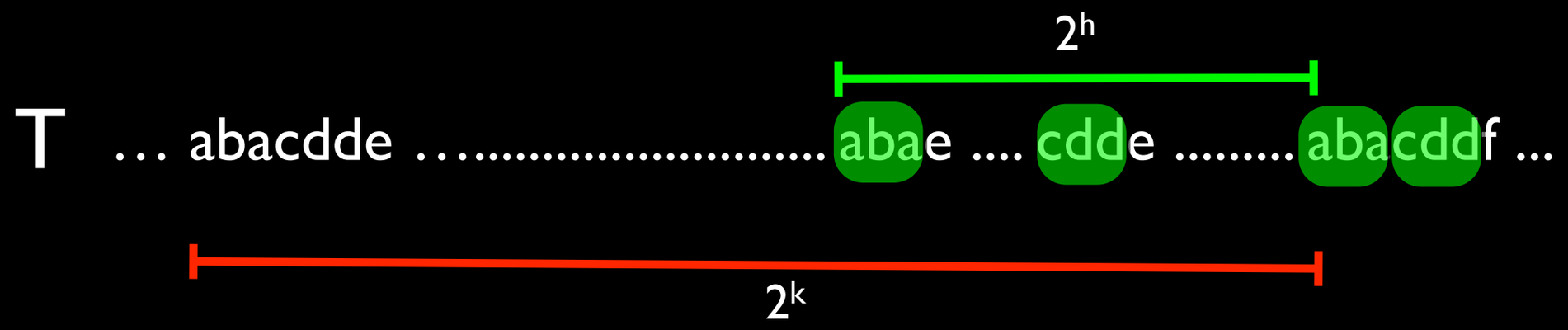
Assume any value j is encoded by f() with log j bits



Cost = $|f(2^k)|$ = log 2^k = k bits

Greedy strategy wastes space

Assume any value j is encoded by f() with log j bits

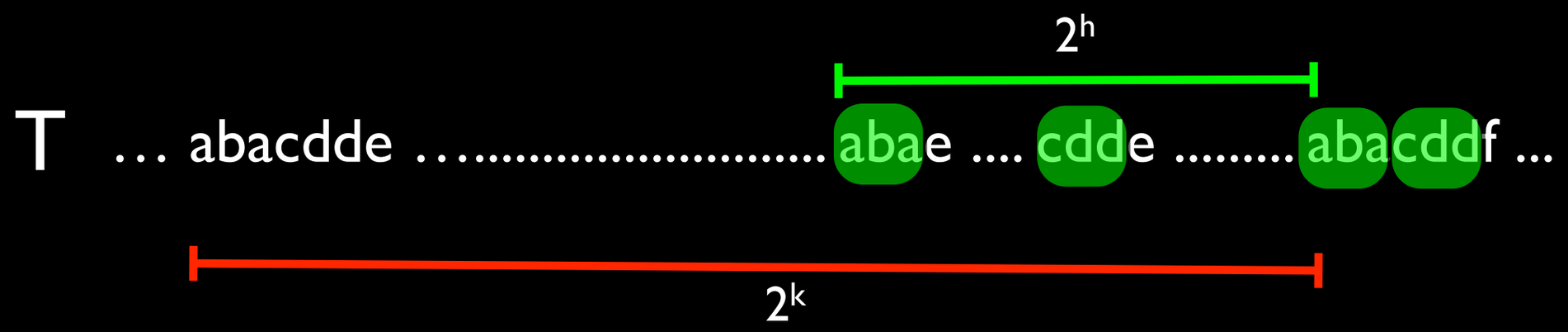


Cost = $|f(2^k)|$ = log 2^k = k bits

Cost ≤ $2|f(2^h)|$ = 2log 2^h = 2h bits

Greedy strategy wastes space

Assume any value j is encoded by f() with log j bits



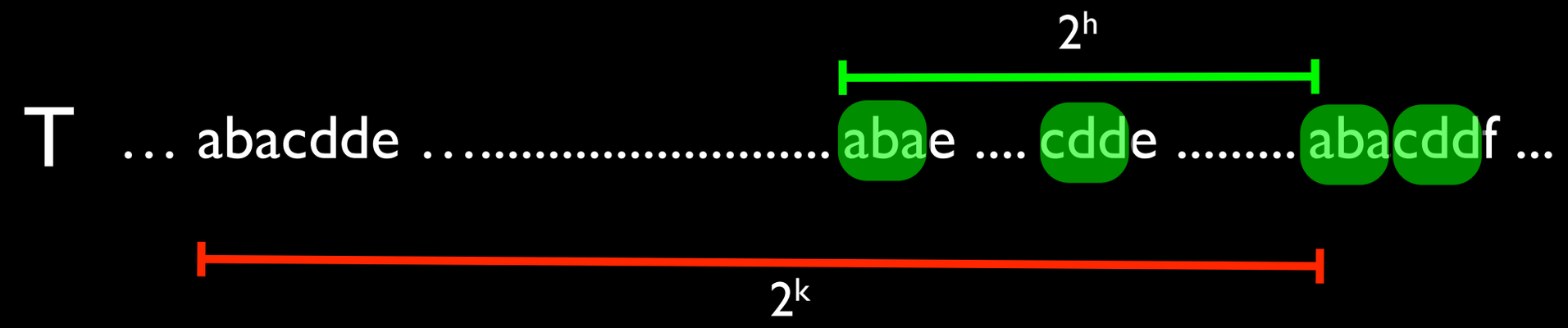
Cost = $|f(2^k)|$ = log 2^k = k bits

Cost ≤ $2|f(2^h)|$ = 2log 2^h = 2h bits

Non greedy strategy is better if h < k/2

Greedy strategy wastes space

Assume any value j is encoded by f() with log j bits



Co

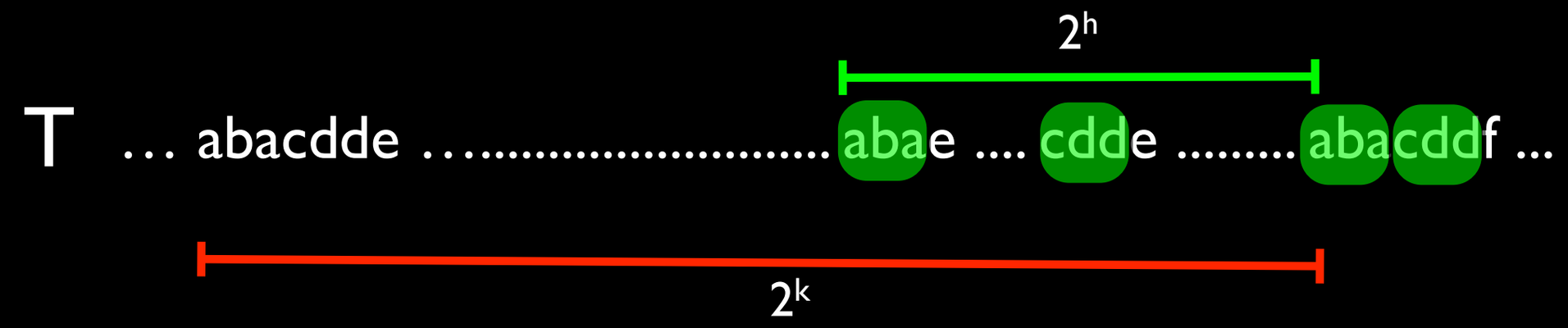
$$\frac{|\,LZ(T)\,|}{|\,OPT(T)} = \Omega(\frac{\log|\,T\,|}{\log\log|\,T\,|})$$

Co bits

Non greedy strategy is better if h < k/2

Greedy strategy wastes space

Assume any value j is encoded by f() with log j bits



Co

Co

$$\frac{|\,LZ(T)\,|}{|\,OPT(T)} = \Omega(\frac{\log|\,T\,|}{\log\log|\,T\,|})$$

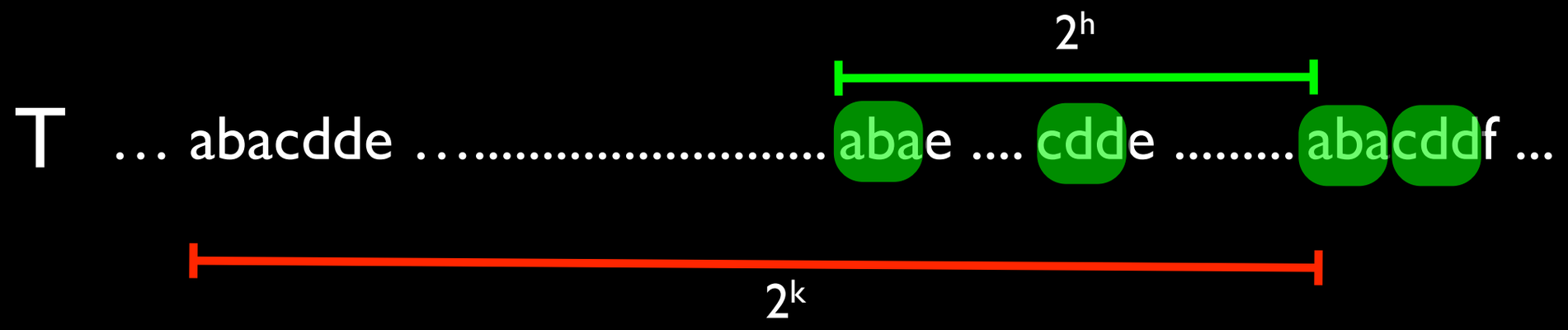
bits

Non greedy strategy is better if h < k/2

good in practice?

Greedy strategy wastes space

Assume any value j is encoded by f() with log j bits



Co

$$\frac{|\,LZ(T)\,|}{|\,OPT(T)} = \Omega(\frac{\log|\,T\,|}{\log\log|\,T\,|})$$

Co bits

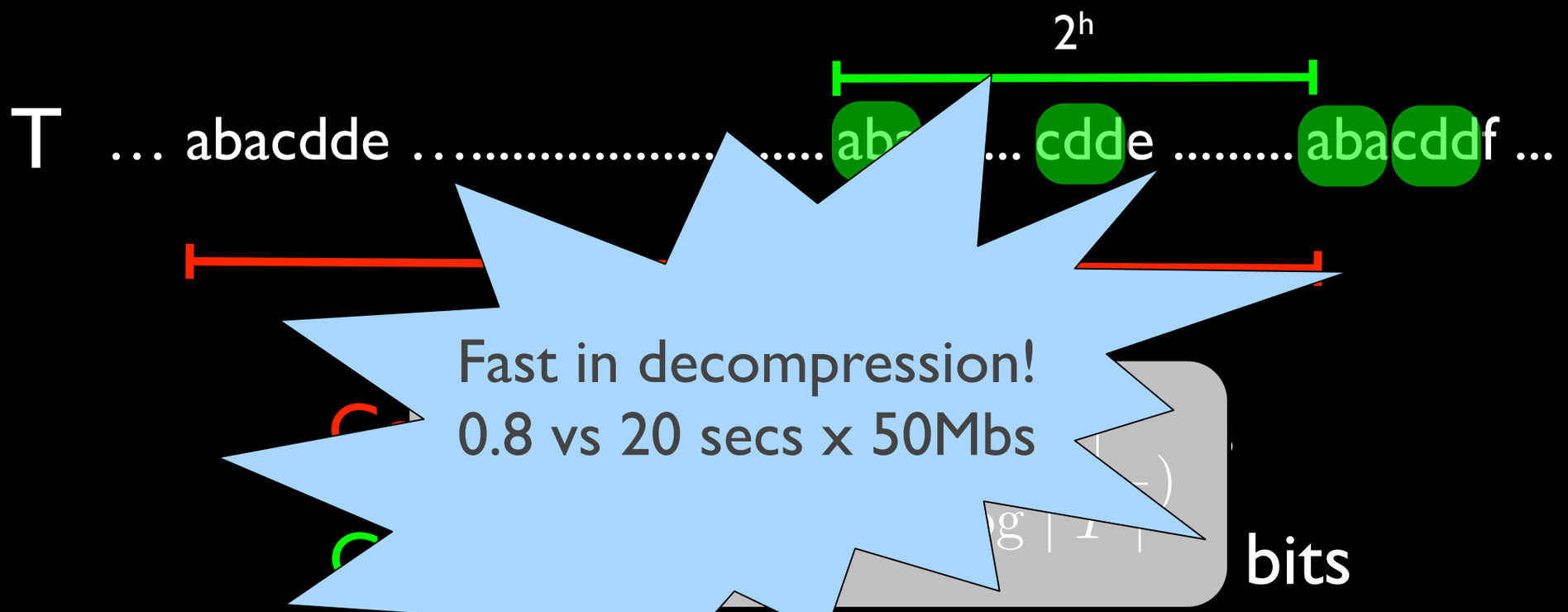
Non greedy strategy is better if h < k/2

good in practice?

File	English	HTML	Sources
Bwt	20.6%	17.3%	3.8%
LZ-fix	26.1%	24.6%	4.9%

Greedy strategy wastes space

Assume any value j is encoded by f() with log j bits



Non greedy strategy is better if h < k/2

good in practice?

File	English	HTML	Sources
Bwt	20.6%	17.3%	3.8%
LZ-fix	26.1%	24.6%	4.9%

Lempel-Ziv 77

with P. Ferragina, I. Nitto, SODA 2009

Lempel-Ziv 77

with P. Ferragina, I. Nitto, SODA 2009

- Optimal parsing can be found with Dynamic Programming in $O(n^2)$ time

Lempel-Ziv 77

with P. Ferragina, I. Nitto, SODA 2009

- Optimal parsing can be found with Dynamic Programming in $O(n^2)$ time
- Our algorithm computes the optimal LZ77-parsing in $O(n \log n)$ time and linear space

Lempel-Ziv 77

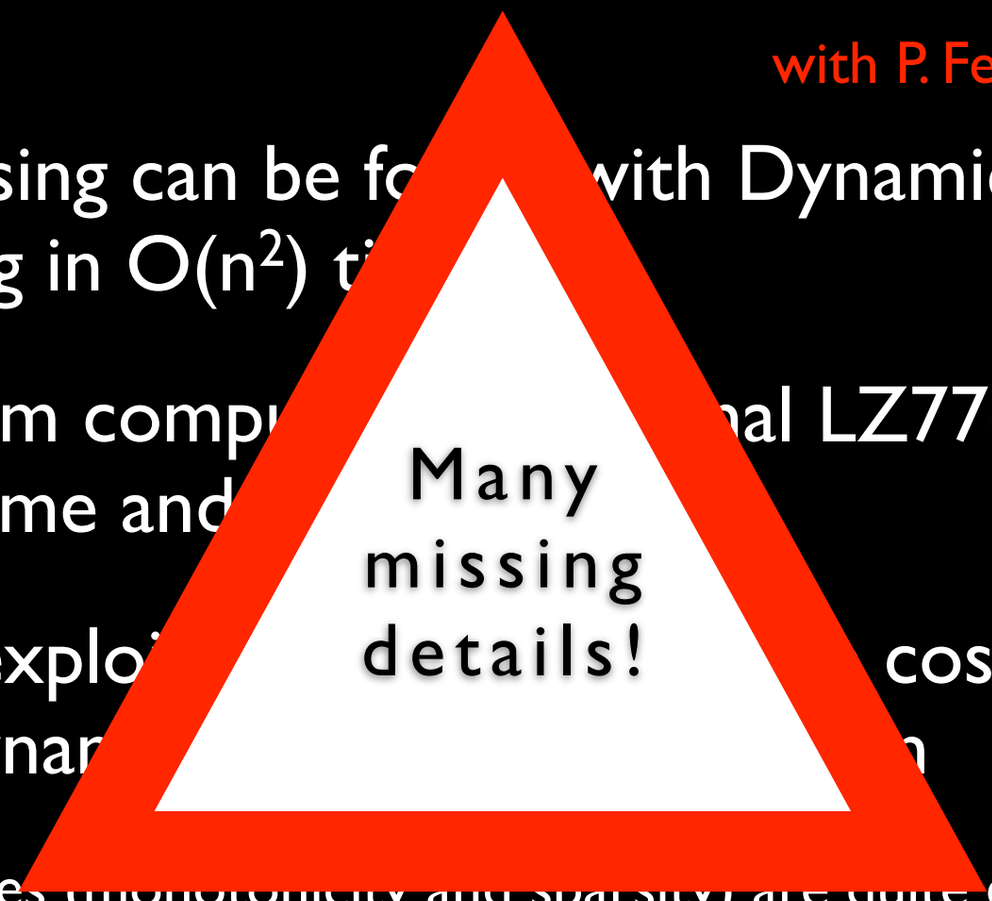
with P. Ferragina, I. Nitto, SODA 2009

- Optimal parsing can be found with Dynamic Programming in $O(n^2)$ time
- Our algorithm computes the optimal LZ77-parsing in O(n log n) time and linear space
- The idea is exploiting properties of the cost function to speed up Dynamic programming solution
  - The properties (monotonicity and sparsity) are quite common. Thus, the idea can be used in other contexts.

Lempel-Ziv 77

with P. Ferragina, I. Nitto, SODA 2009

- Optimal parsing can be fo with Dynamic Programming in $O(n^2)$ t
- Our algorithm compu al LZ77-parsing in $O(n \log n)$ time and
- The idea is explo cost function to speed up Dynar n
  - The properties (monotonicity and sparsity) are quite common. Thus, the idea can be used in other contexts.



Lempel-Ziv 77

with P. Ferragina, I. Nitto, SODA 2009

- Optimal parsing can be found with Dynamic Programming in $O(n^2)$ time
- Our algorithm computes the optimal LZ77-parsing in O(n log n) time and linear space
- The idea is exploiting properties of the cost function to speed up Dynamic programming solution
  - The properties (monotonicity and sparsity) are quite common. Thus, the idea can be used in other contexts.

in practice?

Lempel-Ziv 77

with P. Ferragina, I. Nitto, SODA 2009

- Optimal parsing can be found with Dynamic Programming in $O(n^2)$ time
- Our algorithm computes the optimal LZ77-parsing in O(n log n) time and linear space
- The idea is exploiting properties of the cost function to speed up Dynamic programming solution
  - The properties (monotonicity and sparsity) are quite common. Thus, the idea can be used in other contexts.

in practice?

File	English	HTML	Sources	Dec. time
Bwt	20.6%	17.3%	3.8%	20.2 s
LZ-fix	26.1%	24.6%	4.9%	0.8 s
LZ-OPT	21.6%	17.6%	3.8%	0.9 s

Lempel-Ziv 77

with P. Ferragina, I. Nitto, SODA 2009

- Optimal parsing can be found with Dynamic Programming in $O(n^2)$ time
- Our algorithm computes the optimal LZ77-parsing in O(n log n) time and linear space
- The idea is to exploit the monotonicity of the cost function to speed up Dynamic Programming
- The properties we exploit in the parsing are quite common. Thus, the idea can be used in other contexts

BWT compression
&
LZ77 speed!

in practice?

File	English	HTML	Sources	Dec. time
Bwt	20.6%	17.3%	3.8%	20.2 s
LZ-fix	26.1%	24.6%	4.9%	0.8 s
LZ-OPT	21.6%	17.6%	3.8%	0.9 s

Compressed Scheme with optimal access

with P. Ferragina, SODA 2007 / TCS 2007

- Input: a text T[1,n]
- Goal: design a compressed scheme able to access any portion of T in optimal time (i.e., w bits of information in O(1) time)
- Space is bounded in terms of k-th order entropy of T ($H_k(T)$ + o(n log |Σ|) bits)

Simple scheme with interesting analysis

with P. Ferragina, SODA 2007 / TCS 2007

- $b = \frac{1}{2} \log_\sigma n$
- # blocks = $n/b = O(n / \log_\sigma n)$
- #distinct blocks = $O(\sigma^b) = O(n^{1/2})$

code	block
ε	A
0	B
1	X
00	Y
01	...
10	...
11	...
000	...

frequency

Simple scheme with interesting analysis

with P. Ferragina, SODA 2007 / TCS 2007

- $b = ½ \log_\sigma n$
- # blocks = $n/b = O(n / \log_\sigma n)$
- #distinct blo… = … $O(n^{½})$

Building block for many subsequent compressed DSs

code	block
ε	A
0	B
1	X
00	Y
01	…
10	…
11	…
000	…

frequency

T

…	B	A	Y	X	A	…

b

C

--	0	--	00	1	0	…

P

1	1	2	2	4	5	6	…

Compressed Full-text Indexing

with R. Gonzalez, P. Ferragina, G. Navarro, JEA 2008

Compressed Full-text Indexing

with R. Gonzalez, P. Ferragina, G. Navarro, JEA 2008

Input: a text T[1,n]

Queries:

Count(P): # times string P[1,p] occurs in T
Locate(P): positions of the occurrences of P[1,p] in T
Extract(i,j): return T[i,j]

Compressed Full-text Indexing

with R. Gonzalez, P. Ferragina, G. Navarro, JEA 2008

Input: a text T[1,n]

Queries:

Count(P): # times string P[1,p] occurs in T
Locate(P): positions of the occurrences of P[1,p] in T
Extract(i,j): return T[i,j]

Time-efficient solutions, but not compressed

Suffix Tree, Suffix Array ...
Time: O(|P|+occ)
Space: Θ(n log n) bits --- in practice 5x-20x the text size

Compressed Full-text Indexing

with R. Gonzalez, P. Ferragina, G. Navarro, JEA 2008

Input: a text T[1,n]

Queries:

Count(P): # times string P[1,p] occurs in T
Locate(P): positions of the occurrences of P[1,p] in T
Extract(i,j): return T[i,j]

Time-efficient solutions, but not compressed

Suffix Tree, Suffix Array ...
Time: O(|P|+occ)
Space: Θ(n log n) bits --- in practice 5x-20x the text size

Space-efficient solutions, but slow

Zgrep: uncompress and scan-based algorithm
Time: O(n)
Space: LZ77 compression

Compressed Full-text Indexing

with R. Gonzalez, P. Ferragina, G. Navarro, JEA 2008

Input: a text T[1,n]

Queries:

Count(P): # times string P[1,p] occurs in T
Locate(P): positions of the occurrences of P[1,p] in T
Extract(i,j): return T[i,j]

Time-efficient solutions, but not compressed

Suffix Tree, Suffix Array ...
Time: O(|P|+occ)
Space: Θ(n log n) b… …e text size

Space-efficient …

Zgrep: uncompr… …lgorithm
Time: O(n)
Space: LZ77 compr…ession

Compressed Full-text Indexes

Compressed Full-text Indexing

with R. Gonzalez, P. Ferragina, G. Navarro, JEA 2008

Suffix Array's operations but in compressed space

Three Families:

FM-index [Ferragina-Manzini, FOCS'00, JACM'05]
CSA [Grossi-Vitter, STOC'00, Sadakane SODA'02]
LZ-index [Navarro SPIRE'02]

Compressed Full-text Indexing

with R. Gonzalez, P. Ferragina, G. Navarro, JEA 2008

Suffix Array's operations but in compressed space

Three Families:

FM-index [Ferragina-Manzini, FOCS'00, JACM'05]
CSA [Grossi-Vitter, STOC'00, Sadakane SODA'02]
LZ-index [Navarro SPIRE'02]

Studies at a theoretical stage. No experimental comparison.

Compressed Full-text Indexing

with R. Gonzalez, P. Ferragina, G. Navarro, JEA 2008

Suffix Array's ope

Three Families:

FM-index [F 1'05]

CSA [Gross DA'02]

LZ-index [N

Studies at a theoretical stage. No experimental comparison.

Contribution: Algorithmic engineering and experimetal effort

Pizza&Chili Corpus
Compressed Indexes and their Testbeds

The Italian mirror The Chilean mirror

The Prologue

The new millennium has seen the born of a new class of *full-text indexes* which are structurally similar to Suffix Trees and Suffix Arrays, in that they support the powerful *substring search* operation, but are *succinct* in space, in that it is close to the empirical entropy of the indexed data. They are therefore called *compressed* Suffix Trees and *compressed* Suffix Arrays, or in general *compressed indexes*.

In the literature we counted more than 20 papers authored by more than 20 different researchers. This interest is motivated by the large availability of textual data in electronic form, by the ever increasing gap in performance among the memory levels of current PCs, and by the "non negligible" space occupancy of classic data structures like Suffix Trees and Suffix Arrays which are pervading the BioInformatics and the Text Mining fields.

Don Knuth already observed, in its famous 3rd Volume on the *Art of Computer Programming*, that "space optimization is closely related to time optimization in a disk memory". So we believe that compressed indexes may become a crucial tool for the design of sophisticated and efficient software solutions given the ubiquity of indexing data structures in them. We nevertheless note that the algorithmic technology underlying these compressed indexes stays not at an undergraduate level. Consequently the implementation of any known compressed index requires much engineering effort, a strong algorithmic background, and still the final program may possibly not achieve its best performance!

This site has two mirrors: one in Italy and one in Chile. Hence you can argue the why of its name ;-) Its ultimate goal is to push towards the *technology transfer* of this fascinating algorithmic technology lying at the crossing point of *data compression* and *data structure design*. In order to achieve this goal the Pizza&Chili site offers publicly available implementations of compressed indexes. Each implementation follows a suitable API of functions which should, in our intention, allow any programmer to *plug* the provided compressed indexes within their own software and *play* with their functionalities and efficiency. The site also offers a collection of texts for experimenting and validating compressed indexes. In detail it offers three kinds of material:

- A set of *compressed indexes* which are able to support the same search functionalities of Suffix Trees and Suffix Arrays (e.g., substring searches), but requiring succinct space occupancy and offering, in addition, some text access operations that make them useful within text retrieval and mining software systems.
- A set of *text collections* of various types and sizes to test experimentally the available compressed indexes, or the new compressed indexes that researchers like to submit to this site. The text collections have been selected to form a representative sample of different applications where indexed text searching might be useful. The sizes of these texts are large enough to stress the impact of data compression over memory usage and CPU performance. The goal of experimenting with this testbed is to conclude whether, or not, compressed indexing is beneficial over uncompressed indexing approaches, like Suffix Trees and Suffix Arrays. And, in case it is

Pizza&Chili Corpus
Compressed Indexes and their Testbeds

The Italian mirror The Chilean mirror

The Index Collection

Up to now, the following compressed indexes have been implemented and made available in this site:

- Suffix Array
- Succinct Suffix Array
- Alphabet-Friendly FM-Index
- Compressed Compact Suffix Array
- Run Length FM index
- FM-Index
- LZ-index
- Compressed Suffix Array
- Repair Suffix Array

Api

We are particularly interested in *self-indexes*, namely compressed indexes that encapsulate sufficient information to reproduce any substring of the indexed text, and thus possibly the text itself. If a compressed index is not a self-index, then one must keep the text together with the index and report the text size plus the index size.

To use a compressed index over a text, we first have to *build* it, and then we can either *query* it to *count* or *locate* the occurrences of the queried pattern, or we can *access* some snippets of the indexed text for *displaying* the context of a pattern occurrence, or for *retrieving* some text substrings (possibly the whole text).

Indexes are used through the following API interface, written in the C/C++ language. We actually use *uchar* for denoting *unsigned char* and *ulong* for denoting *unsigned long*. The interface assumes that each text symbol is represented in one byte. The integer *e* returned by any procedure indicates an *error code*, if different of zero. The error message can be accessed by calling the procedure *char* ***error_index**(*e*). We further recall that text and pattern indexes start at zero. Below you find a schematic summary of the API interface offered by all the compressed indexes available for downloading. Please read carefully the `COPYRIGHT` information that comes with each of them.

Building the index

Function	Parameters	Comment
int **build_index**	*uchar* *text, *ulong* length, *char* *build_options, *void* **index	Creates *index* from *text*[0..*length*-1]. Note that the index is an opaque data type. Any build option must be passed in string *build_options*, whose syntax depends on the index. The index must always work with some default parameters if *build_options* is NULL. The returned *index* is ready to be queried.
int **save_index**	*void* *index, *char* *filename	Saves *index* on disk by using single or multiple files, having proper extensions.

The Text Collection

The choice of the types of texts to be indexed and experimented followed some basic considerations. First, we wished to cover a representative set of application areas where the problem of full-text indexing might be relevant, and for each of them selected texts *freely available* over the web. Second, we aimed at maintaining the number of these texts reasonably small in order to avoid long experiments and unreadable tables of results. In particular, we have only one text of each type. Finally, the size of the texts has been chosen large enough to make indexing relevant and compression apparent. Note however that experimenting may be performed at different scales, depending on users' RAM, by using the tool **cut** which allows one to limit the indexed text to any possible length (see below).

Follow the links of each type of text to reach a directory containing one gzipped file, <filename>.gz. Download and gunzip this file to get the original text file, <filename>. The directory also contains other files, named <filename>.<X>MB.gz. These are prefixes of <filename> of <X> megabytes. Of course, some of these files may not exist if <filename> is not long enough.

These are the current collections provided in the Pizza&Chili repository:

- SOURCES (source program code). This file is formed by C/Java source code obtained by concatenating all the .c, .h, .C and .java files of the linux-2.6.11.6 and gcc-4.0.0 distributions. Downloaded on June 9, 2005.
- PITCHES (MIDI pitch values). This file is a sequence of pitch values (bytes in 0-127, plus a few extra special values) obtained from a myriad of MIDI files freely available on Internet. The MIDI files were processed using semex 1.29 tool by Kjell Lemstrom, so as to convert them to IRP format. This is a human-readable tuple format, where the 5th column is the pitch value. Then the pitch values were coded in one byte each and concatenated. Downloaded during April 2005.
- PROTEINS (protein sequences). This file is a sequence of newline-separated protein sequences (without descriptions, just the bare proteins) obtained from the Swissprot database. Each of the 20 amino acids is coded as one uppercase letter. Updated on December 15, 2006.
- DNA (gene DNA sequences). This file is a sequence of newline-separated gene DNA sequences (without descriptions, just the bare DNA code) obtained from files 01hgp10 to 21hgp10, plus 0xhgp10 and 0yhgp10, from Gutenberg Project. Each of the 4 bases is coded as an uppercase letter A,G,C,T, and there are a few occurrences of other special characters. Downloaded on June 9, 2005.
- ENGLISH (english texts). This file is the concatenation of English text files selected from etext02 to etext05 collections of Gutenberg

File Edit View History Bookmarks Tools Help

http://pizzachili.di.unipi.it/texts.html

Google

Pizza&Chili Corpus
Compressed Indexes and their Testbeds

The Italian mirror The Chilean mirror

Home | Index Collection | Text Collection | API | Experimental Setup | The Initiative | Additional Material

The Text Collection

The choice of the types of texts to be indexed and experimented followed some basic considerations. First, we wished to cover a representative set of application areas the web. Second, we aime les of results. In particular, w levant and compression appa e tool **cut** which allows one t

Follow the links of eac he original text file, <filena megabytes. Of course

These are the current

- SOURCES (sou files of the linux-2.6.11
- PITCHES (MIDI a myriad of MIDI f convert them to IRP for ded in one byte each a
- PROTEINS (protein sequences). This file is a sequence of newline-separated protein sequences (without descriptions, just the bare proteins) obtained from the Swissprot database. Each of the 20 amino acids is coded as one uppercase letter. Updated on December 15, 2006.
- DNA (gene DNA sequences). This file is a sequence of newline-separated gene DNA sequences (without descriptions, just the bare DNA code) obtained from files 01hgp10 to 21hgp10, plus 0xhgp10 and 0yhgp10, from Gutenberg Project. Each of the 4 bases is coded as an uppercase letter A,G,C,T, and there are a few occurrences of other special characters. Downloaded on June 9, 2005.
- ENGLISH (english texts). This file is the concatenation of English text files selected from etext02 to etext05 collections of Gutenberg

Are they of practical impact?

Done

File Edit View History Bookmarks Tools Help

http://pizzachili.di.unipi.it/texts.html

Pizza&Chili Corpus
Compressed Indexes and their Testbeds

The Italian mirror The Chilean mirror

Home | Index Collection | Text Collection | API | Experimental Setup | The Initiative | Additional Material

The Text Collection

The choice of the types of texts to be indexed and experimented followed some basic considerations. First, we wished to cover a representative set of application areas … the web. Second, we aime… les of results. In particular, w… levant and compression appa… e tool **cut** which allows one t…

Follow the links of eac… he original text file, <filena… megabytes. Of course…

These are the current …

- SOURCES (sou… files of the linux-2.6.11…
- PITCHES (MIDI… a myriad of MIDI f… convert them to IRP for… ded in one byte each a…
- PROTEINS (protein sequences). This file is a sequence of newline-separated protein sequences (without descriptions, just the bare proteins) obtained from the Swissprot database. Each of the 20 amino acids is coded as one uppercase letter. Updated on December 15, 2006.
- DNA (gene DNA sequences). This file is a sequence of newline-separated gene DNA sequences (without descriptions, just the bare DNA code) obtained from files 01hgp10 to 21hgp10, plus 0xhgp10 and 0yhgp10, from Gutenberg Project. Each of the 4 bases is coded as an uppercase letter A,G,C,T, and there are a few occurrences of other special characters. Downloaded on June 9, 2005.
- ENGLISH (english texts). This file is the concatenation of English text files selected from etext02 to etext05 collections of Gutenberg

Are they of practical impact?

- Count(P) takes 2 microsecs/char -- 50% space

Done

File Edit View History Bookmarks Tools Help

http://pizzachili.di.unipi.it/texts.html

Google

Pizza&Chili Corpus
Compressed Indexes and their Testbeds

The Italian mirror The Chilean mirror

Home | Index Collection | Text Collection | API | Experimental Setup | The Initiative | Additional Material

The Text Collection

The choice of the types of texts to be indexed and experimented followed some basic considerations. First, we wished to cover a representative set of application areas … the web. Second, we aime… les of results. In particular, w… levant and compression appa… e tool **cut** which allows one t…

Follow the links of eac… he original text file, <filena… megabytes. Of course…

These are the current …

- SOURCES (sou… files of the linux-2.6.11…
- PITCHES (MIDI… a myriad of MIDI f… convert them to IRP for… ded in one byte each a…
- PROTEINS (protein sequences). This file is a sequence of newline-separated protein sequences (without descriptions, just the bare proteins) obtained from the Swissprot database. Each of the 20 amino acids is coded as one uppercase letter. Updated on December 15, 2006.
- DNA (gene DNA sequences). This file is a sequence of newline-separated gene DNA sequences (without descriptions, just the bare DNA code) obtained from files 01hgp10 to 21hgp10, plus 0xhgp10 and 0yhgp10, from Gutenberg Project. Each of the 4 bases is coded as an uppercase letter A,G,C,T, and there are a few occurrences of other special characters. Downloaded on June 9, 2005.
- ENGLISH (english texts). This file is the concatenation of English text files selected from etext02 to etext05 collections of Gutenberg

Are they of practical impact?

- Count(P) takes 2 microsecs/char -- 50% space

5x slower!

Done

File Edit View History Bookmarks Tools Help

http://pizzachili.di.unipi.it/texts.html

Google

Pizza&Chili Corpus
Compressed Indexes and their Testbeds

The Italian mirror The Chilean mirror

Home | Index Collection | Text Collection | API | Experimental Setup | The Initiative | Additional Material

The Text Collection

The choice of the types of texts to be indexed and experimented followed some basic considerations. First, we wished to cover a representative set of application areas … the web. Second, we aime… les of results. In particular, w… levant and compression appa… e tool **cut** which allows one t…

Follow the links of eac… he original text file, <filena… megabytes. Of course…

These are the current …

- SOURCES (sou… files of the linux-2.6.11…
- PITCHES (MIDI… a myriad of MIDI f… convert them to IRP for… ded in one byte each a…
- PROTEINS (protein sequences). This file is a sequence of newline-separated protein sequences (without descriptions, just the bare proteins) obtained from the Swissprot database. Each of the 20 amino acids is coded as one uppercase letter. Updated on December 15, 2006.
- DNA (gene DNA sequences). This file is a sequence of newline-separated gene DNA sequences (without descriptions, just the bare DNA code) obtained from files 01hgp10 to 21hgp10, plus 0xhgp10 and 0yhgp10, from Gutenberg Project. Each of the 4 bases is coded as an uppercase letter A,G,C,T, and there are a few occurrences of other special characters. Downloaded on June 9, 2005.
- ENGLISH (english texts). This file is the concatenation of English text files selected from etext02 to etext05 collections of Gutenberg

Are they of practical impact?

- Count(P) takes 2 microsecs/char -- 50% space

5x slower!

- Extract 1 Mb/sec

Done

File Edit View History Bookmarks Tools Help

http://pizzachili.di.unipi.it/texts.html

Google

Pizza&Chili Corpus
Compressed Indexes and their Testbeds

The Italian mirror The Chilean mirror

Home | Index Collection | Text Collection | API | Experimental Setup | The Initiative | Additional Material

The Text Collection

The choice of the types of texts to be indexed and experimented followed some basic considerations. First, we wished to cover a representative set of application areas … the web. Second, we aime… les of results. In particular, w… levant and compression appa… e tool **cut** which allows one t…

Follow the links of eac… he original text file, <filena… megabytes. Of course…

These are the current …

- SOURCES (sou… files of the linux-2.6.11…
- PITCHES (MIDI… a myriad of MIDI f… convert them to IRP for… ded in one byte each a…
- PROTEINS (protein sequences). This file is a sequence of newline-separated protein sequences (without descriptions, just the bare proteins) obtained from the Swissprot database. Each of the 20 amino acids is coded as one uppercase letter. Updated on December 15, 2006.
- DNA (gene DNA sequences). This file is a sequence of newline-separated gene DNA sequences (without descriptions, just the bare DNA code) obtained from files 01hgp10 to 21hgp10, plus 0xhgp10 and 0yhgp10, from Gutenberg Project. Each of the 4 bases is coded as an uppercase letter A,G,C,T, and there are a few occurrences of other special characters. Downloaded on June 9, 2005.
- ENGLISH (english texts). This file is the concatenation of English text files selected from etext02 to etext05 collections of Gutenberg

Are they of practical impact?

- Count(P) takes 2 microsecs/char -- 50% space

5x slower!

- Extract 1 Mb/sec

- Locate(P) takes 15 microsecs/occ -- 70% space

Done

File Edit View History Bookmarks Tools Help

http://pizzachili.di.unipi.it/texts.html

Google

Pizza&Chili Corpus
Compressed Indexes and their Testbeds

The Italian mirror The Chilean mirror

Home | Index Collection | Text Collection | API | Experimental Setup | The Initiative | Additional Material

The Text Collection

The choice of the types of texts to be indexed and experimented followed some basic considerations. First, we wished to cover a representative set of application areas … the web. Second, we aime… les of results. In particular, w… levant and compression appa… e tool **cut** which allows one t…

Follow the links of eac… he original text file, <filena… megabytes. Of course…

These are the current …

- SOURCES (sou… files of the linux-2.6.11…
- PITCHES (MIDI… myriad of MIDI f… them to IRP for… one byte each a…
- PROTEINS (protein sequences). This file is a sequence of newline-separated protein sequences (without descriptions, just the bare proteins) obtained from the Swissprot database. Each of the 20 amino acids is coded as one uppercase letter. Updated on December 15, 2006.
- DNA (gene DNA sequences). This file is a sequence of newline-separated gene DNA sequences (without descriptions, just the bare DNA code) obtained from files 01hgp10 to 21hgp10, plus 0xhgp10 and 0yhgp10, from Gutenberg Project. Each of the 4 bases is coded as an uppercase letter A,G,C,T, and there are a few occurrences of other special characters. Downloaded on June 9, 2005.
- ENGLISH (english texts). This file is the concatenation of English text files selected from etext02 to etext05 collections of Gutenberg

Are they of practical impact?

- Count(P) takes 2 microsecs/char -- 50% space — 5x slower!
- Extract 1 Mb/sec
- Locate(P) takes 15 microsecs/occ -- 70% space — 500x slower!

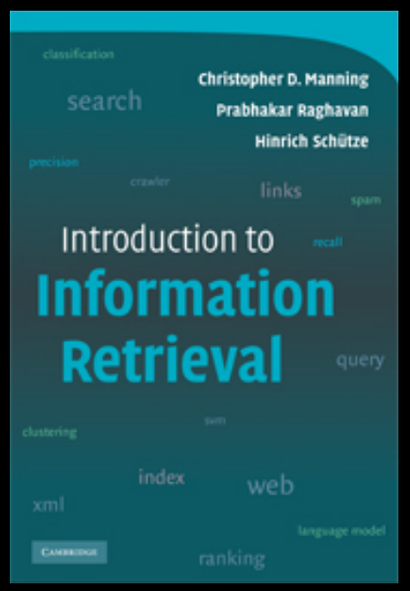
Done

Pattern matching on Dictionary of Strings

with P. Ferragina, SIGIR 2007 / TALG 2010

Pattern matching on Dictionary of Strings

with P. Ferragina, SIGIR 2007 / TALG 2010



Input: dictionary D of strings

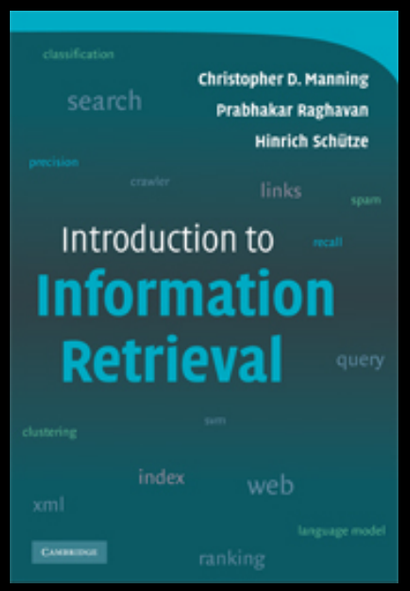
Queries:

Id ⇔ String

Prefix(P): strings prefixed by P[1,p]
Suffix(P): strings suffixed by P[1,p]
PrefixSuffix(P,Q): strings prefixed by P[1,p] and suffixed by Q[1,q]

Pattern matching on Dictionary of Strings

with P. Ferragina, SIGIR 2007 / TALG 2010



Input: dictionary D of strings

Queries:

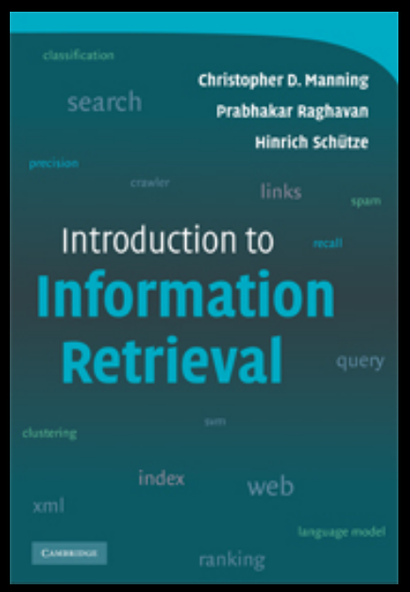
Id ⇔ String

Prefix(P): strings prefixed by P[1,p]
Suffix(P): strings suffixed by P[1,p]
PrefixSuffix(P,Q): strings prefixed by P[1,p] and suffixed by Q[1,q]

Hashing

Pattern matching on Dictionary of Strings

with P. Ferragina, SIGIR 2007 / TALG 2010



Input: dictionary D of strings

Queries:

Id ⇔ String

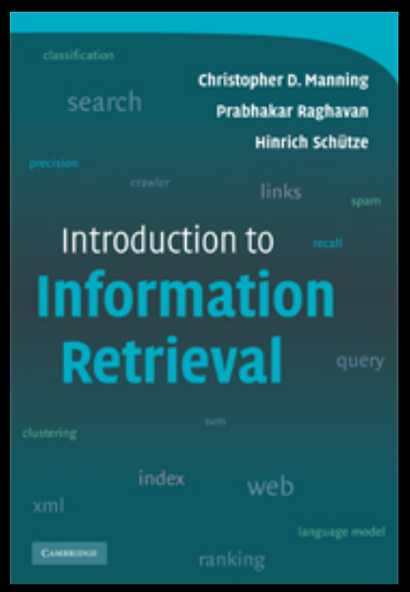
Prefix(P): strings prefixed by P[1,p]
Suffix(P): strings suffixed by P[1,p]
PrefixSuffix(P,Q): strings prefixed by P[1,p] and suffixed by Q[1,q]

Hashing

Need D to avoid false-positive

Pattern matching on Dictionary of Strings

with P. Ferragina, SIGIR 2007 / TALG 2010



Input: dictionary D of strings

Queries:

Id ⇔ String

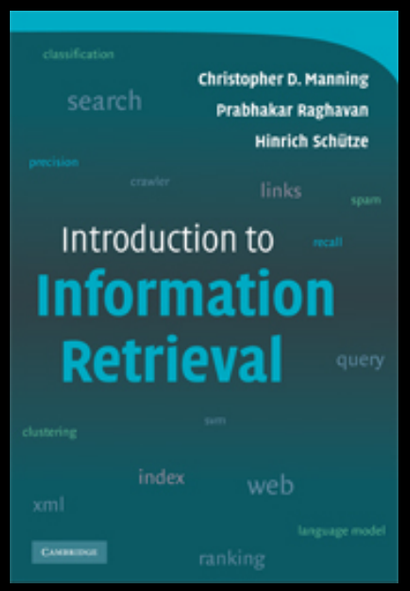
Prefix(P): strings prefixed by P[1,p]
Suffix(P): strings suffixed by P[1,p]
PrefixSuffix(P,Q): strings prefixed by P[1,p] and suffixed by Q[1,q]

Hashing

Need D to avoid false-positive
Solve only Id ⇔ String

Pattern matching on Dictionary of Strings

with P. Ferragina, SIGIR 2007 / TALG 2010



Input: dictionary D of strings

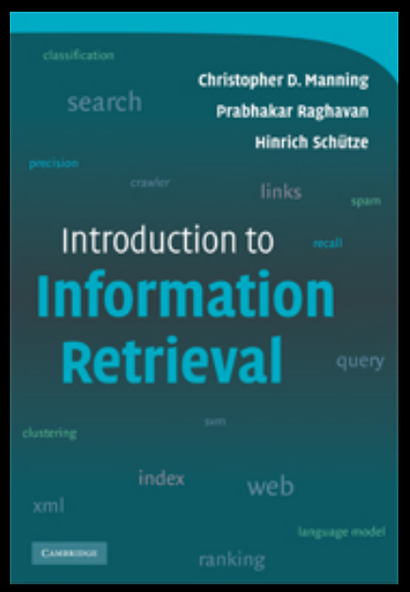
Queries:

Id ⇔ String

Prefix(P): strings prefixed by P[1,p]
Suffix(P): strings suffixed by P[1,p]
PrefixSuffix(P,Q): strings prefixed by P[1,p] and suffixed by Q[1,q]

Pattern matching on Dictionary of Strings

with P. Ferragina, SIGIR 2007 / TALG 2010



Input: dictionary D of strings

Queries:

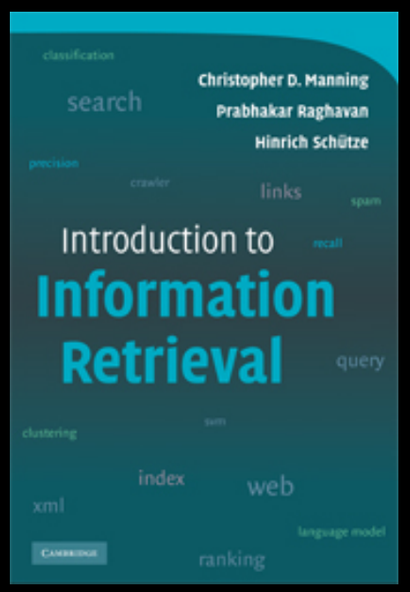
Id ⇔ String

Prefix(P): strings prefixed by P[1,p]
Suffix(P): strings suffixed by P[1,p]
PrefixSuffix(P,Q): strings prefixed by P[1,p] and suffixed by Q[1,q]

Compact tries

Pattern matching on Dictionary of Strings

with P. Ferragina, SIGIR 2007 / TALG 2010



Input: dictionary D of strings

Queries:

Id ⇔ String

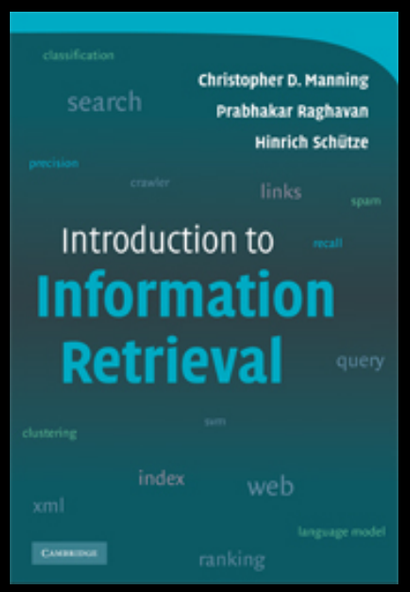
Prefix(P): strings prefixed by P[1,p]
Suffix(P): strings suffixed by P[1,p]
PrefixSuffix(P,Q): strings prefixed by P[1,p] and suffixed by Q[1,q]

Compact tries

Need node/edges pointer

Pattern matching on Dictionary of Strings

with P. Ferragina, SIGIR 2007 / TALG 2010



Input: dictionary D of strings

Queries:

Id ⇔ String

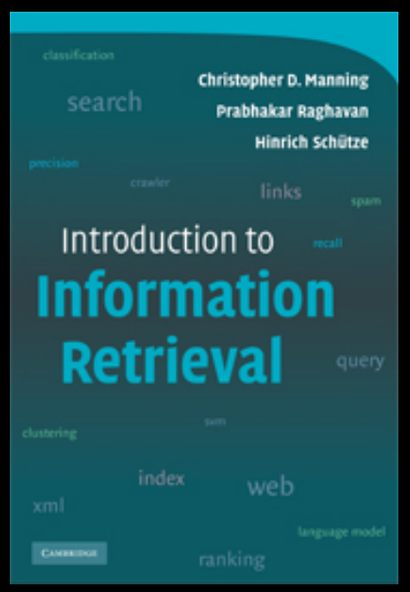
Prefix(P): strings prefixed by P[1,p]
Suffix(P): strings suffixed by P[1,p]
PrefixSuffix(P,Q): strings prefixed by P[1,p] and suffixed by Q[1,q]

Compact tries

Need node/edges pointer
Need D to retrive edges' labels

Pattern matching on Dictionary of Strings

with P. Ferragina, SIGIR 2007 / TALG 2010



Input: dictionary D of strings

Queries:

Id ⇔ String

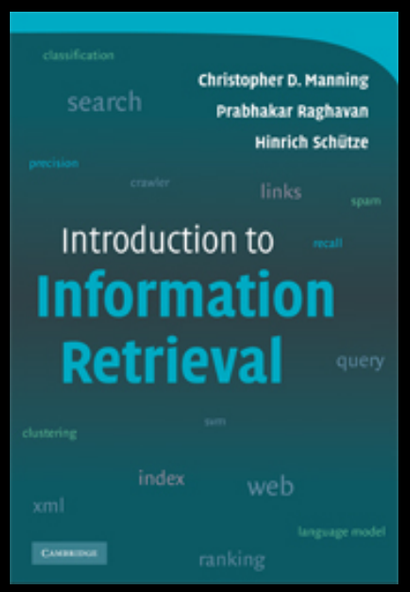
Prefix(P): strings prefixed by P[1,p]
Suffix(P): strings suffixed by P[1,p]
PrefixSuffix(P,Q): strings prefixed by P[1,p] and suffixed by Q[1,q]

Compact tries

Need node/edges pointer
Need D to retrive edges' labels
Need Ids

Pattern matching on Dictionary of Strings

with P. Ferragina, SIGIR 2007 / TALG 2010



Input: dictionary D of strings

Queries:

Id ⇔ String

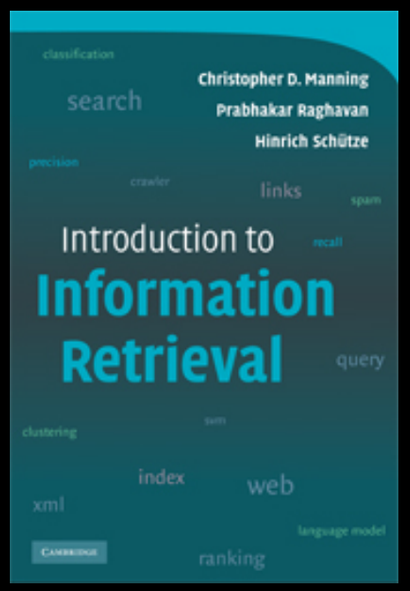
Prefix(P): strings prefixed by P[1,p]
Suffix(P): strings suffixed by P[1,p]
PrefixSuffix(P,Q): strings prefixed by P[1,p] and suffixed by Q[1,q]

Compact tries

Need node/edges pointer
Need D to retrive edges' labels
Need Ids
Suffix query needs Trie on D^R

Pattern matching on Dictionary of Strings

with P. Ferragina, SIGIR 2007 / TALG 2010



Input: dictionary D of strings

Queries:

Id ⇔ String

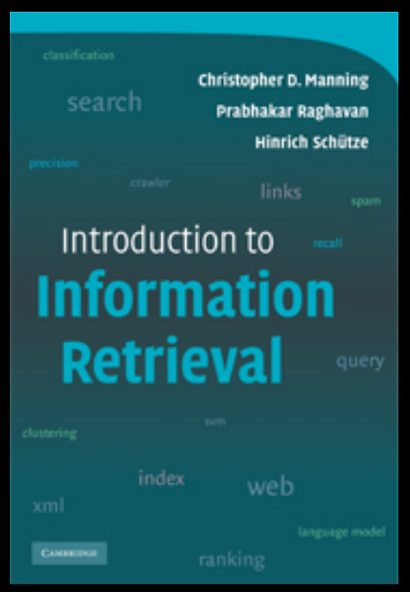
Prefix(P): strings prefixed by P[1,p]
Suffix(P): strings suffixed by P[1,p]
PrefixSuffix(P,Q): strings prefixed by P[1,p] and suffixed by Q[1,q]

Compact tries

Need node/edges pointer
Need D to retrive edges' labels
Need Ids
Suffix query needs Trie on D^R
PrefixSuffix query have to intersect the results $\Theta(occP+occQ)$

Pattern matching on Dictionary of Strings

with P. Ferragina, SIGIR 2007 / TALG 2010



Input: dictionary D of strings

Queries:

Id ⇔ String

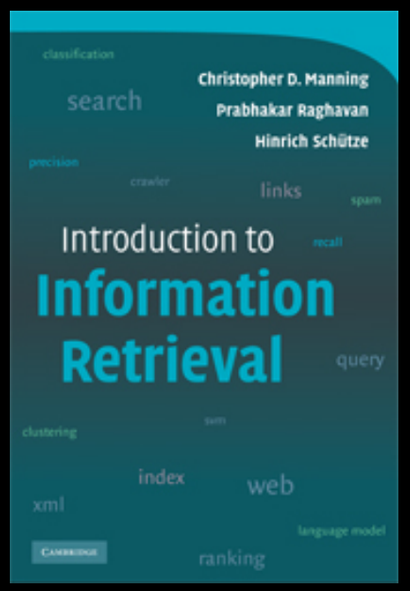
Prefix(P): strings prefixed by P[1,p]
Suffix(P): strings suffixed by P[1,p]
PrefixSuffix(P,Q): strings prefixed by P[1,p] and suffixed by Q[1,q]

Compact tries

~~Need node/edges pointer~~
Need D to retrive edges' labels
Need Ids
Suffix query needs Trie on D^R
PrefixSuffix query have to intersect the results $\Theta(occP+occQ)$

Pattern matching on Dictionary of Strings

with P. Ferragina, SIGIR 2007 / TALG 2010



Input: dictionary D of strings

Queries:

Id ⇔ String

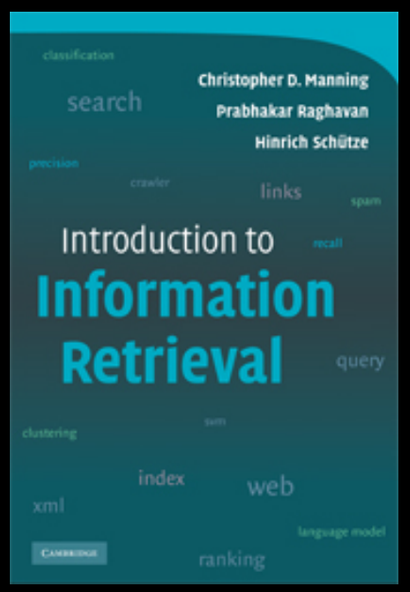
Prefix(P): strings prefixed by P[1,p]
Suffix(P): strings suffixed by P[1,p]
PrefixSuffix(P,Q): strings prefixed by P[1,p] and suffixed by Q[1,q]

Compact tries

~~Need node/edges pointer~~
~~Need D to retrive edges' labels~~
Need Ids
Suffix query needs Trie on D^R
PrefixSuffix query have to intersect the results $\Theta(occP+occQ)$

Pattern matching on Dictionary of Strings

with P. Ferragina, SIGIR 2007 / TALG 2010



Input: dictionary D of strings

Queries:

Id ⇔ String

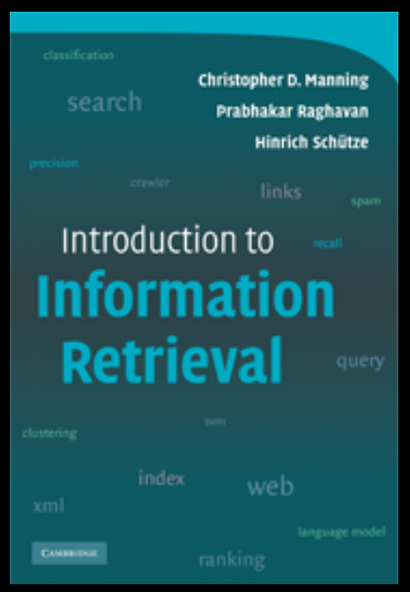
Prefix(P): strings prefixed by P[1,p]
Suffix(P): strings suffixed by P[1,p]
PrefixSuffix(P,Q): strings prefixed by P[1,p] and suffixed by Q[1,q]

Compact tries

~~Need node/edges pointer~~
~~Need D to retrive edges' labels~~
~~Need Ids~~
Suffix query needs Trie on D^R
PrefixSuffix query have to intersect the results $\Theta(occP+occQ)$

Pattern matching on Dictionary of Strings

with P. Ferragina, SIGIR 2007 / TALG 2010



Input: dictionary D of strings

Queries:

Id ⇔ String

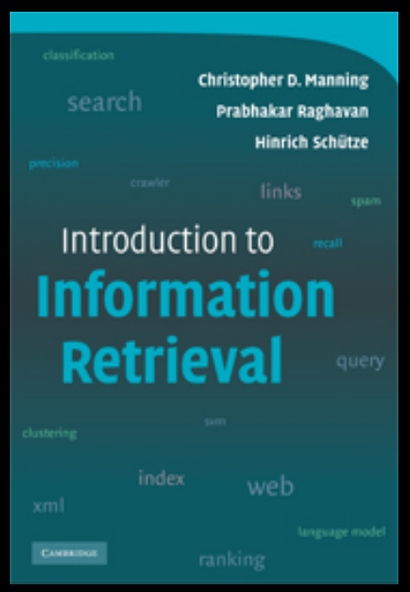
Prefix(P): strings prefixed by P[1,p]
Suffix(P): strings suffixed by P[1,p]
PrefixSuffix(P,Q): strings prefixed by P[1,p] and suffixed by Q[1,q]

Compact tries

~~Need node/edges pointer~~
~~Need D to retrive edges' labels~~
~~Need Ids~~
~~Suffix query needs Trie on D^R~~
PrefixSuffix query have to intersect the results Θ(occP+occQ)

Pattern matching on Dictionary of Strings

with P. Ferragina, SIGIR 2007 / TALG 2010



Input: dictionary D of strings

Queries:

Id ⇔ String

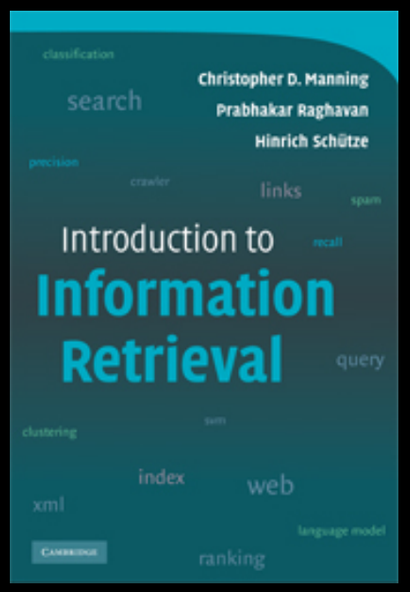
Prefix(P): strings prefixed by P[1,p]
Suffix(P): strings suffixed by P[1,p]
PrefixSuffix(P,Q): strings prefixed by P[1,p] and suffixed by Q[1,q]

Compact tries

- ~~Need node/edges pointer~~
- ~~Need D to retrive edges' labels~~
- ~~Need Ids~~
- ~~Suffix query needs Trie on D^R~~
- ~~PrefixSuffix query have to intersect the results $\Theta(occP+occQ)$~~

Pattern matching on Dictionary of Strings

with P. Ferragina, SIGIR 2007 / TALG 2010



Input: dictionary D of strings

Queries:

Id ⇔ String

Prefix(P): strings prefixed by P[l,r]

Suffix(P): strings suffixed by P[

PrefixSuffix(P,Q): strings prefix

Compact tries

~~Need node/edges pointer~~

~~Need D to retrive edges' labe~~

~~Need Ids~~

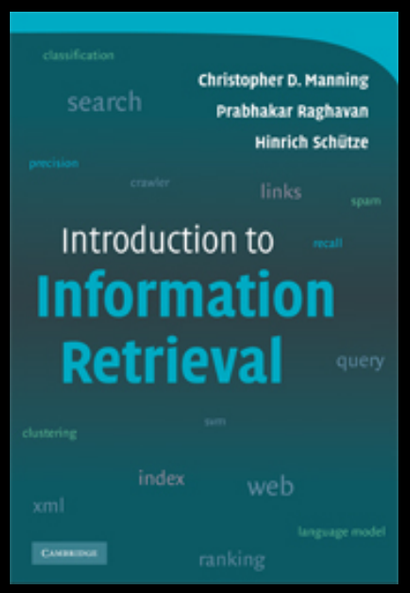
~~Suffix query needs Trie on D~~

~~PrefixSuffix query have to int~~

Method	DictUrl
Trie	1374.29%
FC-32	109.95%
FC-128	107.41%
FC-1024	106.67%
CPI-AFI	49.72%
CPI-CSA-64	37.82%
CPI-CSA-128	31.57%
CPI-CSA-256	28.45%
CPI-FMI-256	24.27%
CPI-FMI-512	18.94%
CPI-FMI-1024	16.12%

Pattern matching on Dictionary of Strings

with P. Ferragina, SIGIR 2007 / TALG 2010



Input: dictionary D of strings

Queries:

Id ⇔ String

Prefix(P): strings prefixed by P[l,r]

Suffix(P): strings suffixed by P[

PrefixSuffix(P,Q): strings prefix

Compact tries

~~Need node/edges pointer~~

~~Need D to retrive edges' labe~~

~~Need Ids~~

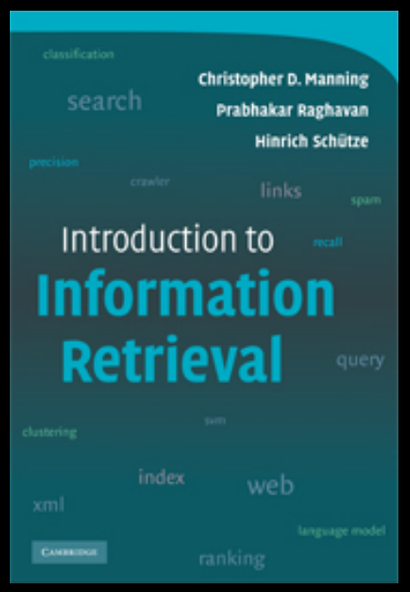
~~Suffix query needs Trie on D~~

~~PrefixSuffix query have to int~~

Method	DictUrl
Trie	1374.29%
FC-32	109.95%
FC-128	107.41%
FC-1024	106.67%
CPI-AFI	49.72%
CPI-CSA-64	37.82%
CPI-CSA-128	31.57%
CPI-CSA-256	28.45%
CPI-FMI-256	24.27%
CPI-FMI-512	18.94%
CPI-FMI-1024	16.12%

Pattern matching on Dictionary of Strings

with P. Ferragina, SIGIR 2007 / TALG 2010



Input: dictionary D of strings

Queries:

Id ⇔ String

Prefix(P): strings prefixed by P[l,r]

Suffix(P): strings suffixed by P[

PrefixSuffix(P,Q): strings prefix

Compact tries

~~Need node/edges pointer~~

~~Need D to retrive edges' labe~~

~~Need Ids~~

~~Suffix query needs Trie on D~~

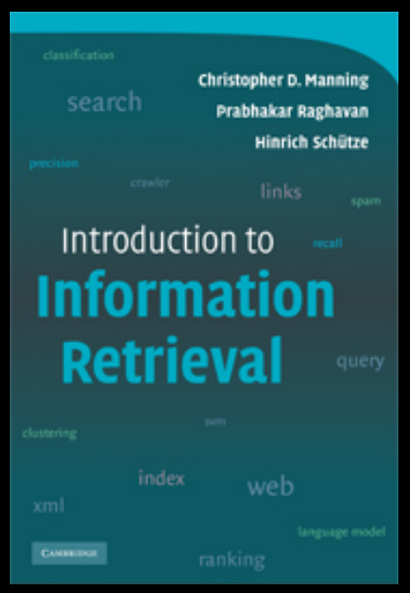
~~PrefixSuffix query have to int~~

Method	DictUrl
Trie	1374.29%
FC-32	109.95%
FC-128	107.41%
FC-1024	106.67%
CPI-AFI	49.72%
CPI-CSA-64	37.82%
CPI-CSA-128	31.57%
CPI-CSA-256	28.45%
CPI-FMI-256	24.27%
CPI-FMI-512	18.94%
CPI-FMI-1024	16.12%

trade-off

Pattern matching on Dictionary of Strings

with P. Ferragina, SIGIR 2007 / TALG 2010



Input: dictionary D of strings

Queries:

Id ⇔ String

Prefix(P): strings prefixed by P[1,p]

Suffix(P): strings suffixed by P[

PrefixSuffix(P,Q): strings prefix

Compact tries

~~Need node/edges pointer~~

~~Need D to retrive edges' labe~~

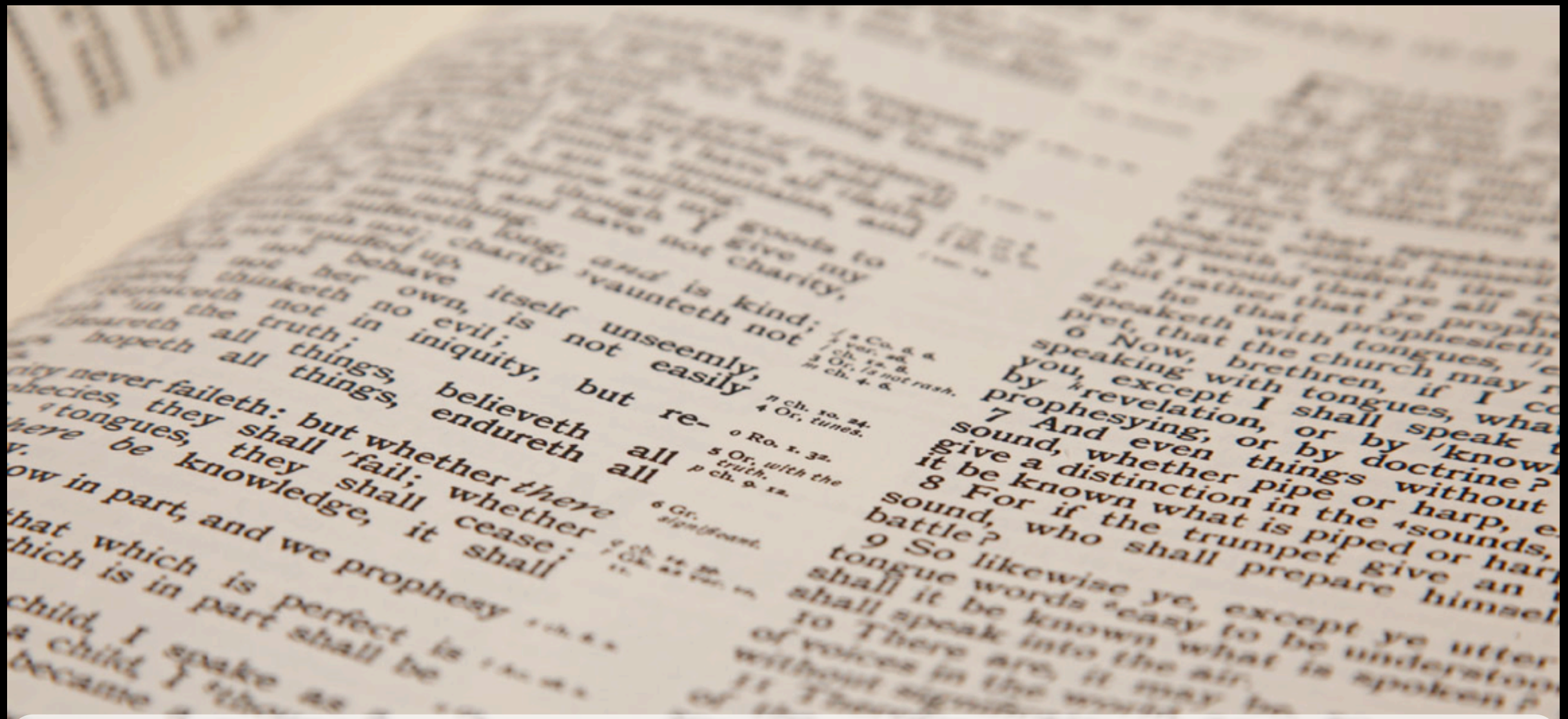
~~Need Ids~~

~~Suffix query needs Trie on D~~

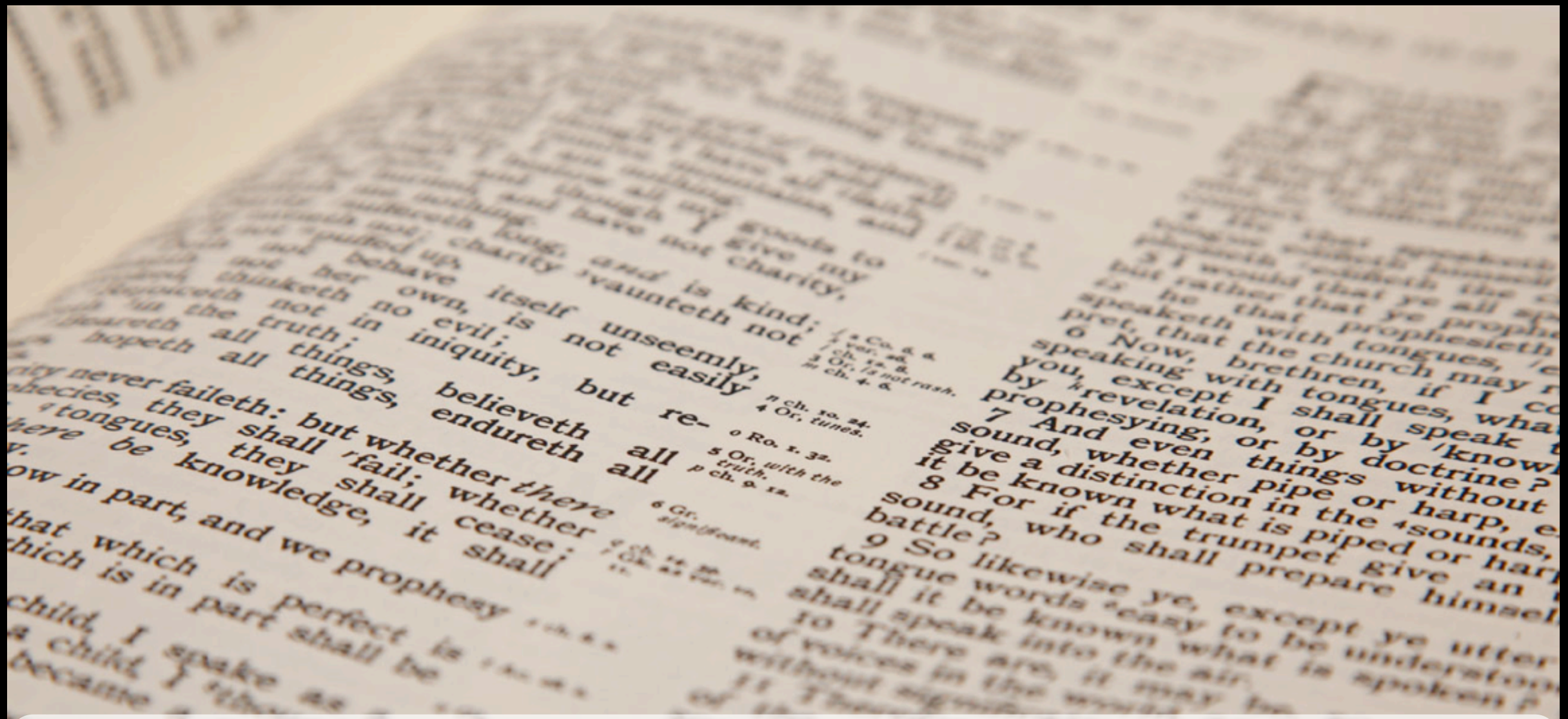
~~PrefixSuffix query have to int~~

Method	DictUrl
Trie	1374.29%
FC-32	109.95%
FC-128	107.41%
FC-1024	106.67%
CPI-AFI	49.72%
CPI-CSA-64	37.82%
CPI-CSA-128	31.57%
CPI-CSA-256	28.45%
CPI-FMI-256	24.27%
CPI-FMI-512	18.94%
CPI-FMI-1024	16.12%

>85x



Maximize Compression and Efficiently Access and Search



Maximize Compression and
Efficiently Access and Search

What's next?

External Memory

Maximize Compression and Efficiently Access and Search

What's next?

External Memory

Multi-core processors
GPUs

Solid disk

Maximize Compression and
Efficiently Access and Search

What's next?

External Memory

Multi-core processors
GPUs

Solid disk

Distributed Computing
Cloud computing

Maximize Compression and
Efficiently Access and Search

What's next?

External Memory

Multi-core processors
GPUs

Solid disk

Distributed Computing
Cloud computing

Energy/Money

Maximize Compression and
Efficiently Access and Search

What's next?

External Memory

Multi-core processors
GPUs

Solid disk

Distributed Computing
Cloud computing

Energy/Money

Export ideas in
other fields

Maximize Compression and
Efficiently Access and Search

What's next?

Circumvent Lower Bounds

Time lower bounds

Space lower bounds

Circumvent Lower Bounds

Time lower bounds

What is the minimum time complexity of your query (worst case)?

Space lower bounds

Circumvent Lower Bounds

Time lower bounds

What is the minimum time complexity of your query (worst case)?

There exists superconstant time lower bounds for many problems.

Space lower bounds

Circumvent Lower Bounds

Time lower bounds

What is the minimum time complexity of your query (worst case)?

There exists superconstant time lower bounds for many problems.

Obtained via sophisticate information theoretic and probabilistic arguments.

Space lower bounds

Circumvent Lower Bounds

Time lower bounds

What is the minimum time complexity of your query (worst case)?

There exists superconstant time lower bounds for many problems.

Obtained via sophisticate information theoretic and probabilistic arguments.

- Choose a set of queries Q uniformly at random among a set of possible queries

Space lower bounds

Circumvent Lower Bounds

Time lower bounds

What is the minimum time complexity of your query (worst case)?

There exists superconstant time lower bounds for many problems.

Obtained via sophisticate information theoretic and probabilistic arguments.

- Choose a set of queries Q uniformly at random among a set of possible queries
- Prove all Q must require T(Q) time

Space lower bounds

Circumvent Lower Bounds

Time lower bounds

What is the minimum time complexity of your query (worst case)?

There exists superconstant time lower bounds for many problems.

Obtained via sophisticate information theoretic and probabilistic arguments.

- Choose a set of queries Q uniformly at random among a set of possible queries
- Prove all Q must require T(Q) time
- Worst case time per op at least T(Q)/|Q|

Space lower bounds

Circumvent Lower Bounds

Time lower bounds

What is the minimum time complexity of your query (worst case)?

There exists superconstant time lower bounds for many problems.

Obtained via sophisticate information theoretic and probabilistic arguments.

- Choose a set of queries Q ~~uniformly at random~~ among a set of possible queries
- Prove all Q must require T(Q) time
- ~~Worst case time per op at least T(Q)/|Q|~~

Space lower bounds

Circumvent Lower Bounds

Time lower bounds

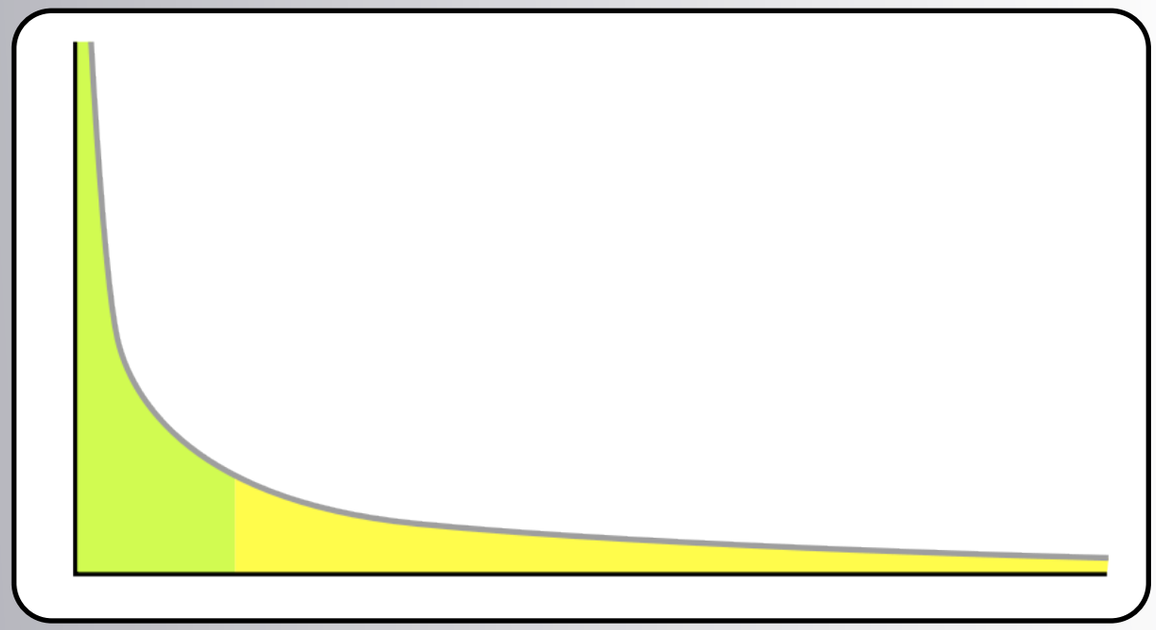
What is the minimum time complexity of your query (worst case)?

There exists superconstant time lower bounds for many problems.

Obtained via sophisticate information theoretic and probabilistic arguments.

- Choose a set of queries Q ~~uniformly at random~~ among a set of possible queries
- Prove all Q must require T(Q) time
- ~~Worst case time per op at least T(Q)/|Q|~~

Space lower bounds



Circumvent Lower Bounds

Time lower bounds

What is the minimum time complexity of your query (worst case)?

There exists superconstant time lower bounds for many problems.

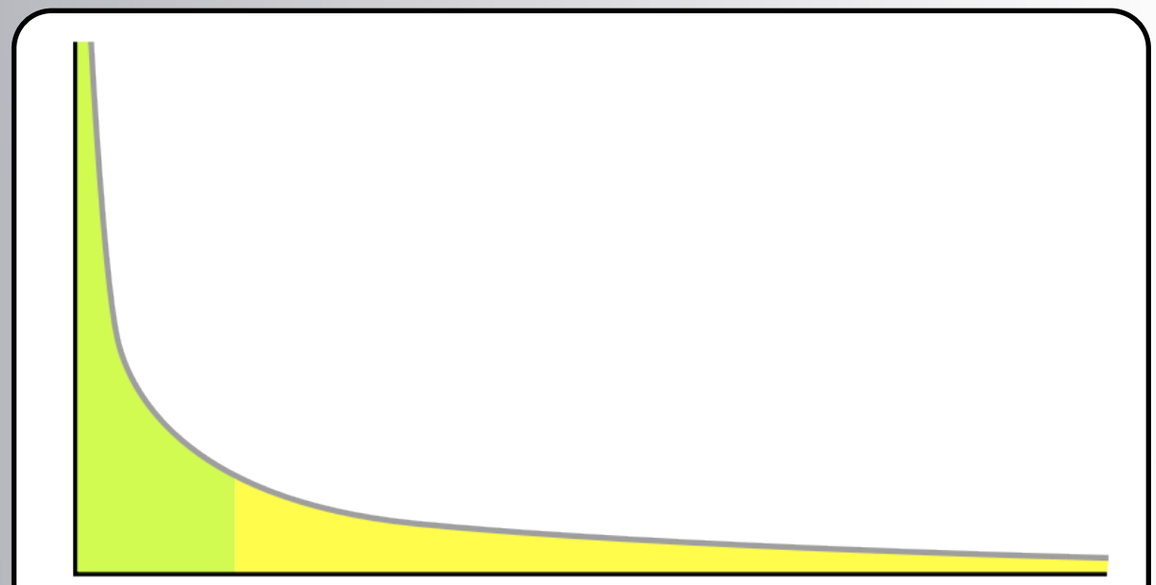
Obtained via sophisticate information theoretic and probabilistic arguments.

- Choose a set of queries Q ~~uniformly at random~~ among a set of possible queries
- Prove all Q must require T(Q) time
- ~~Worst-case time per op at least T(Q)/|Q|~~

- Design Distribution-aware (compressed) data structures

  DS knows the distribution of subsequent queries, or self-adapts to unknown distribution to reduce time complexity

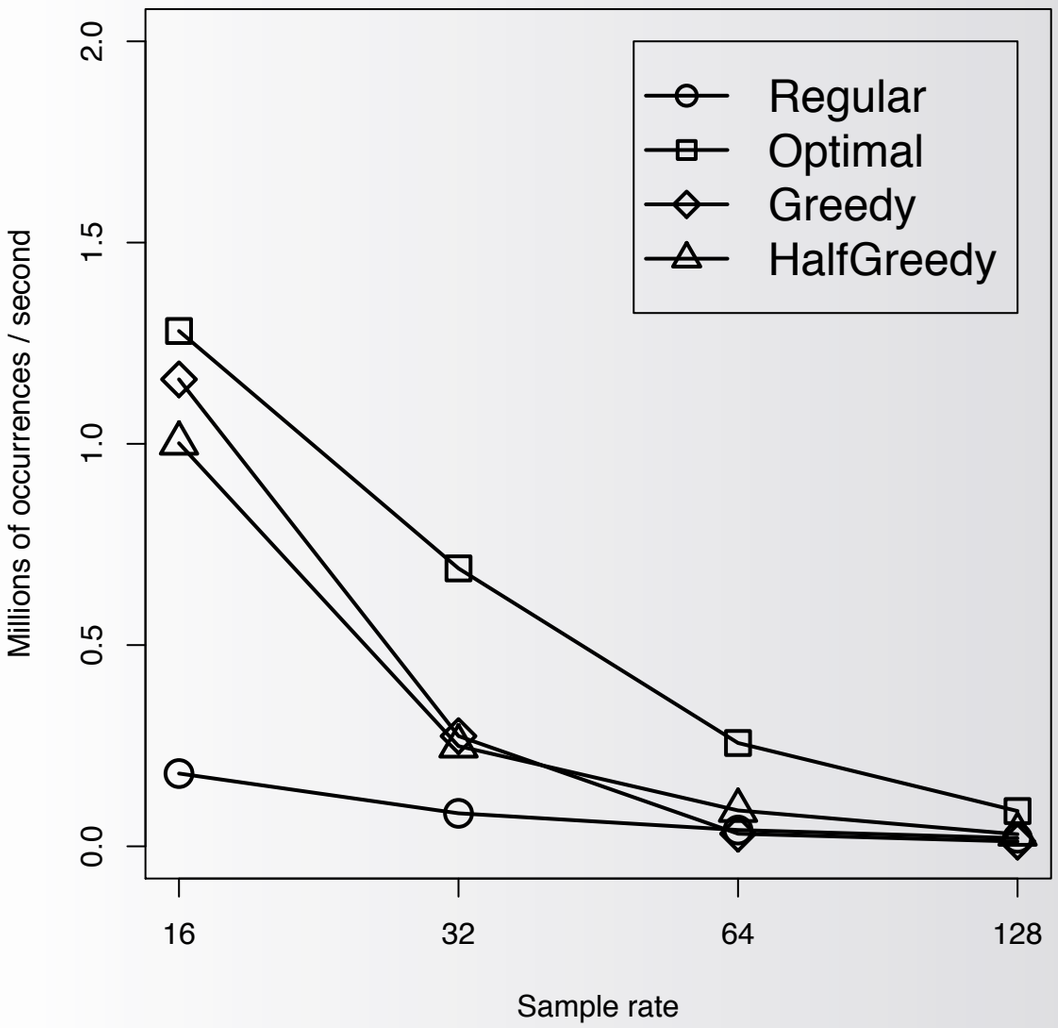
Space lower bounds



Distribution-Aware Compressed Full-text Indexing

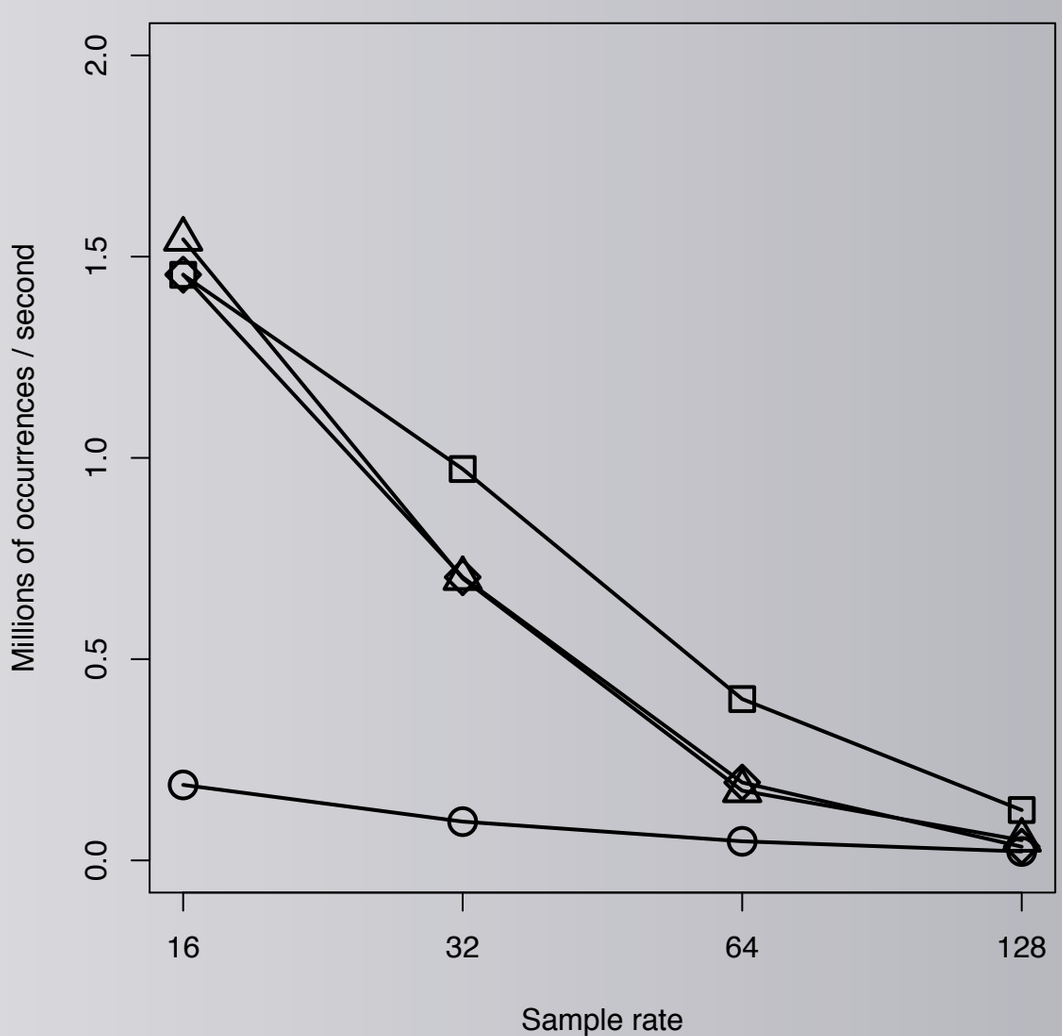
with P. Ferragina, J. Siren, ESA 2011

HTML Pages



10,000 patterns
187 million of positions

DBLP

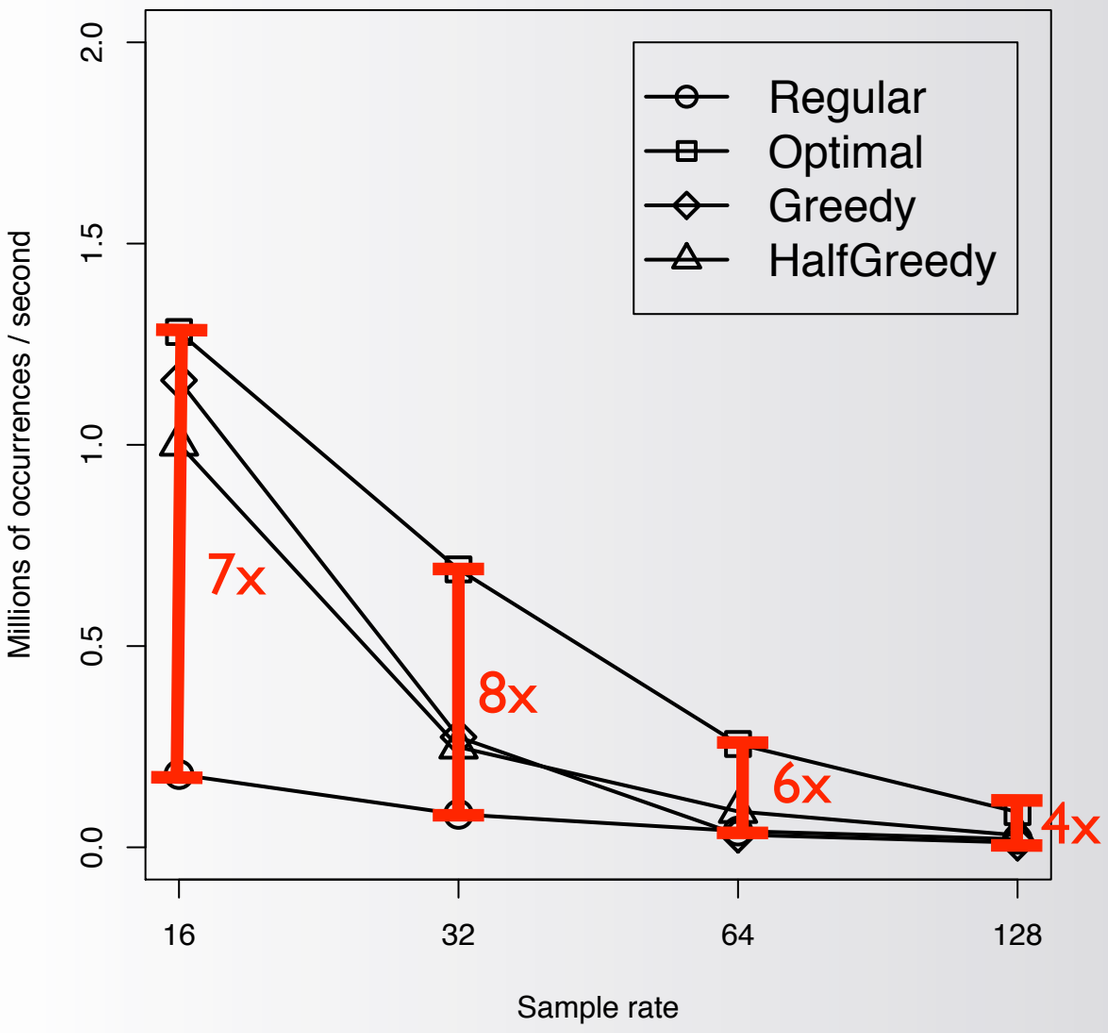


10,000 patterns
276 million of positions

Distribution-Aware Compressed Full-text Indexing

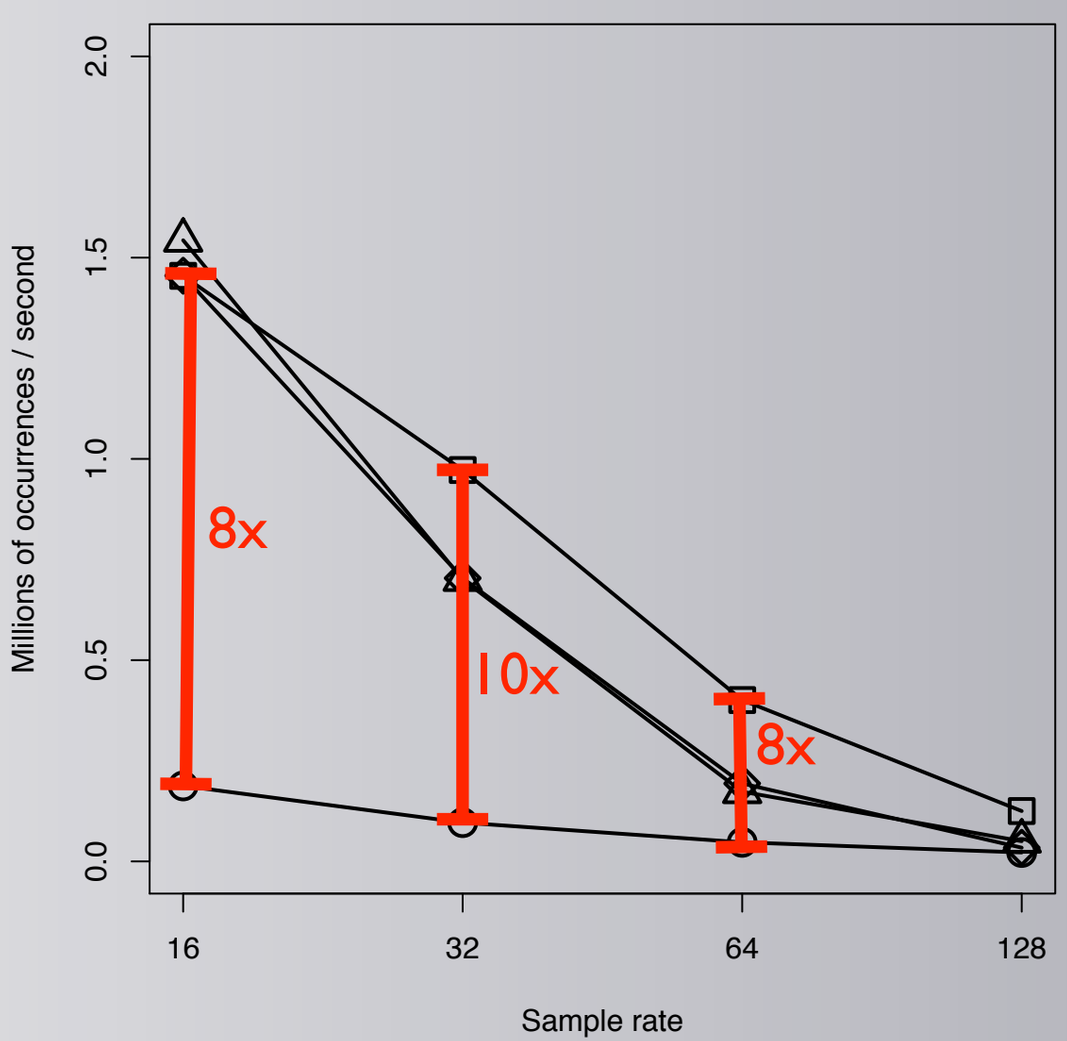
with P. Ferragina, J. Siren, ESA 2011

HTML Pages



10,000 patterns
187 million of positions

DBLP



10,000 patterns
276 million of positions

Circumvent Lower Bounds

Time lower bounds

What is the minimum time complexity of your query (worst case)?

There exists superconstant time lower bounds for many problems.

Obtained via sophisticate information theoretic and probabilistic arguments.

- Choose a set of queries Q ~~uniformally at random~~
- Prove all Q must require T(Q) time
- ~~Worst case time per op at least T(Q)/|Q|~~
- Design Distribution-aware (compressed) data structures
    DS knows the distribution of subsequent queries, or self-adapts to unknown distribution

Space lower bounds

Circumvent Lower Bounds

Time lower bounds

What is the minimum time complexity of your query (worst case)?

There exists superconstant time lower bounds for many problems.

Obtained via sophisticate information theoretic and probabilistic arguments.

- Choose a set of queries Q ~~uniformally at random~~
- Prove all Q must require T(Q) time
- ~~Worst case time per op at least T(Q)/|Q|~~

- Design Distribution-aware (compressed) data structures

  DS knows the distribution of subsequent queries, or self-adapts to unknown distribution

Space lower bounds

What is the minimum numbers of bits you need?

Circumvent Lower Bounds

Time lower bounds

What is the minimum time complexity of your query (worst case)?

There exists superconstant time lower bounds for many problems.

Obtained via sophisticate information theoretic and probabilistic arguments.

- Choose a set of queries Q ~~uniformally at random~~
- Prove all Q must require T(Q) time
- ~~Worst case time per op at least T(Q)/|Q|~~

- Design Distribution-aware (compressed) data structures

  DS knows the distribution of subsequent queries, or self-adapts to unknown distribution

Space lower bounds

What is the minimum numbers of bits you need?

Obtained via combinatorial arguments.

Circumvent Lower Bounds

Time lower bounds

What is the minimum time complexity of your query (worst case)?

There exists superconstant time lower bounds for many problems.

Obtained via sophisticate information theoretic and probabilistic arguments.

- Choose a set of queries Q ~~uniformally at random~~
- Prove all Q must require T(Q) time
- ~~Worst case time per op at least T(Q)/|Q|~~

- Design Distribution-aware (compressed) data structures

  DS knows the distribution of subsequent queries, or self-adapts to unknown distribution

Space lower bounds

What is the minimum numbers of bits you need?

Obtained via combinatorial arguments.

Space is log of number of possible objects of your type.

E.g., 2^n binary texts of len n $\Rightarrow$ log 2^n = n bits.

Circumvent Lower Bounds

Time lower bounds

What is the minimum time complexity of your query (worst case)?

There exists superconstant time lower bounds for many problems.

Obtained via sophisticate information theoretic and probabilistic arguments.

- Choose a set of queries Q ~~uniformally at random~~
- Prove all Q must require T(Q) time
- ~~Worst case time per op at least T(Q)/|Q|~~

- Design Distribution-aware (compressed) data structures
  - DS knows the distribution of subsequent queries, or self-adapts to unknown distribution

Space lower bounds

What is the minimum numbers of bits you need?

Obtained via combinatorial arguments.

Space is log of number of possible objects of your type.

E.g., 2^n binary texts of len n $\Rightarrow$ log 2^n = n bits.

No two objects can have the same representation!

Circumvent Lower Bounds

Time lower bounds

What is the minimum time complexity of your query (worst case)?

There exists superconstant time lower bounds for many problems.

Obtained via sophisticate information theoretic and probabilistic arguments.

- Choose a set of queries Q ~~uniformally at random~~
- Prove all Q must require T(Q) time
- ~~Worst case time per op at least T(Q)/|Q|~~

- Design Distribution-aware (compressed) data structures
  - DS knows the distribution of subsequent queries, or self-adapts to unknown distribution

Space lower bounds

What is the minimum numbers of bits you need?

Obtained via combinatorial arguments.

Space is log of number of possible objects of your type.

E.g., 2^n binary texts of len n $\Rightarrow$ log 2^n = n bits.

No two objects can have the same representation!

What if your (available) memory does not suffice?

Circumvent Lower Bounds

Time lower bounds

What is the minimum time complexity of your query (worst case)?

There exists superconstant time lower bounds for many problems.

Obtained via sophisticate information theoretic and probabilistic arguments.

- Choose a set of queries Q ~~uniformally at random~~
- Prove all Q must require T(Q) time
- ~~Worst case time per op at least T(Q)/|Q|~~

- Design Distribution-aware (compressed) data structures
  DS knows the distribution of subsequent queries, or self-adapts to unknown distribution

Space lower bounds

What is the minimum numbers of bits you need?

Obtained via combinatorial arguments.

Space is log of number of possible objects of your type.

E.g., 2^n binary texts of len n $\Rightarrow$ log 2^n = n bits.

No two objects can have the same representation!

What if your (available) memory does not suffice?

- Buy more memory!

Circumvent Lower Bounds

Time lower bounds

What is the minimum time complexity of your query (worst case)?

There exists superconstant time lower bounds for many problems.

Obtained via sophisticate information theoretic and probabilistic arguments.

- Choose a set of queries Q ~~uniformally at random~~
- Prove all Q must require T(Q) time
- ~~Worst case time per op at least T(Q)/|Q|~~

- Design Distribution-aware (compressed) data structures
  DS knows the distribution of subsequent queries, or self-adapts to unknown distribution

Space lower bounds

What is the minimum numbers of bits you need?

Obtained via combinatorial arguments.

Space is log of number of possible objects of your type.

E.g., 2^n binary texts of len n $\Rightarrow$ log 2^n = n bits.

No two objects can have the same representation!

What if your (available) memory does not suffice?

- Buy more memory!
- Design data structures without data.
  DS has to err! but error rate can be guaranteed

Compressed Functions

with D. Belazzougui, SODA 2013

Compressed Functions

with D. Belazzougui, SODA 2013

TOY problem

Compressed Functions

with D. Belazzougui, SODA 2013

problem

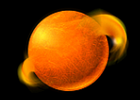
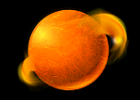
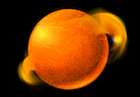
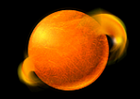
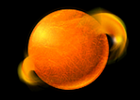
City

New York
Rome
Madrid
Helsinki
Oslo
London
Athens
Berlin

Compressed Functions

with D. Belazzougui, SODA 2013

problem

City	Weather
New York	
Rome	
Madrid	
Helsinki	
Oslo	
London	
Athens	
Berlin	

Compressed Functions

with D. Belazzougui, SODA 2013

problem

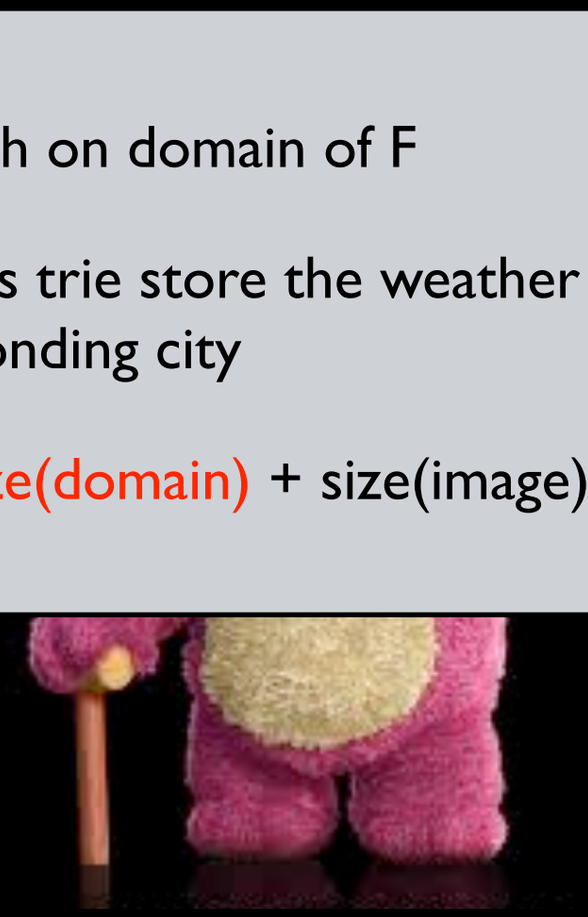
City	Weather
New York	
Rome	
Madrid	
Helsinki	
Oslo	
London	
Athens	
Berlin	

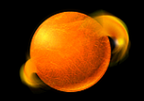
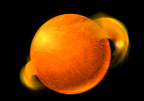
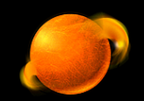
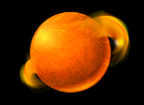
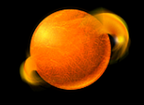
$$F : S \subseteq U \to \Sigma$$

Compressed Functions

with D. Belazzougui, SODA 2013

- Trie to search on domain of F
- Leaves of this trie store the weather in the corresponding city
- Space: size(domain) + size(image)



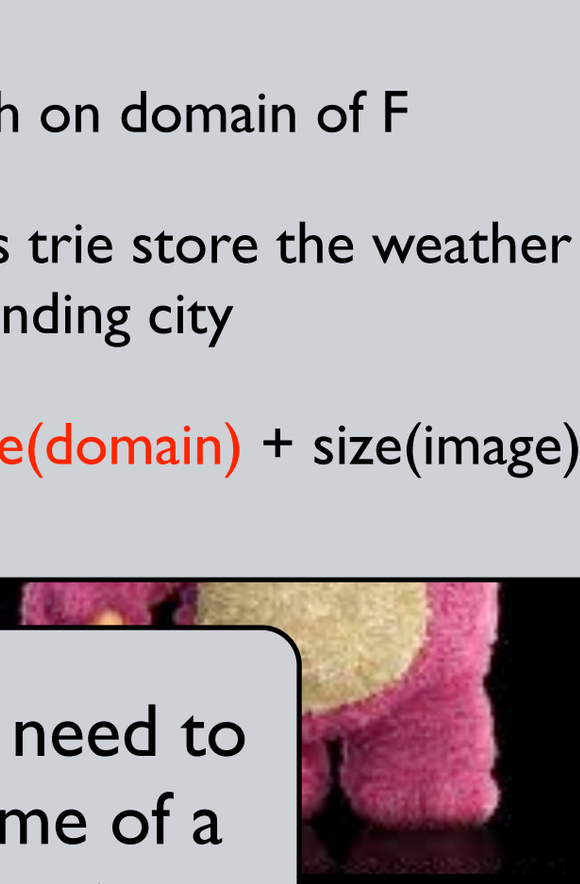
City	Weather
New York	
Rome	
Madrid	
Helsinki	
Oslo	
London	
Athens	
Berlin	

$$F : S \subseteq U \to \Sigma$$

Compressed Functions

with D. Belazzougui, SODA 2013

- Trie to search on domain of F
- Leaves of this trie store the weather in the corresponding city
- Space: size(domain) + size(image)



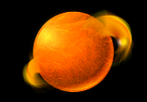
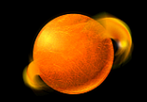
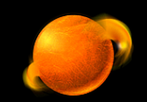
Do we really need to store the name of a city to know its weather?

City	Weather
New York	
Rome	
Madrid	
Helsinki	
Oslo	
London	
Athens	
Berlin	

$$F : S \subseteq U \rightarrow \Sigma$$

Compressed Functions

with D. Belazzougui, SODA 2013

City	Weather
New York	
Rome	
Madrid	
Helsinki	

- Trie to search on domain of F
- Leaves of this trie store the weather in the corresponding city
  - Space: size(domain) + si

Do we really need to store the name of a city to know its weather?

F

- Compressed representation of F so that, given a key X in O(1)
  - we return F(X), if X belongs to S
  - we return an arbitrary value, otherwise
- No need to store/access/remember the domain of F
- Codomain is stored in compressed space (Entropy of values)

Occurrence Estimation

with A. Orlandi, PODS 2011

Occurrence Estimation

with A. Orlandi, PODS 2011

- A (large) text T is preprocessed and a (small) index I is built

Occurrence Estimation

with A. Orlandi, PODS 2011

- A (large) text T is preprocessed and a (small) index I is built
  - I is able to estimate the number of occurrences of any given substring P in T without the need of T

Occurrence Estimation

with A. Orlandi, PODS 2011

- A (large) text T is preprocessed and a (small) index I is built
  - I is able to estimate the number of occurrences of any given substring P in T without the need of T
  - T is not needed and I uses sublinear space
  - Results are incorrect for (at most) a term +E

Occurrence Estimation

with A. Orlandi, PODS 2011

- A (large) text T is preprocessed and a (small) index I is built
  - I is able to estimate the number of occurrences of any given substring P in T without the need of T
  - T is not needed and I uses sublinear space
  - Results are incorrect for (at most) a term +E
- Our PODS provides an O(|P|) time space/error optimal solution

Occurrence Estimation

with A. Orlandi, PODS 2011

- A (large) text T is preprocessed and a (small) index I is built
  - I is able to estimate the number of occurrences of any given substring P in T without the need of T
  - T is not needed and I uses sublinear space
  - Results are incorrect for (at most) a term +E
- Our PODS provides an O(|P|) time space/error optimal solution
  - i.e., O(|T|/E) bits

Occurrence Estimation

with A. Orlandi, PODS 2011

- A (large) text T is preprocessed and a (small) index I is built
  - I is able to estimate the number of occurrences of any given substring P in T without the need of T
  - T is not needed and I uses sublinear space
  - Results are incorrect for (at most) a term +E
- Our PODS provides an O(|P|) time space/error optimal solution
  - i.e., O(|T|/E) bits
- Application to Selectivity Estimation in DB for execution planing optimizations

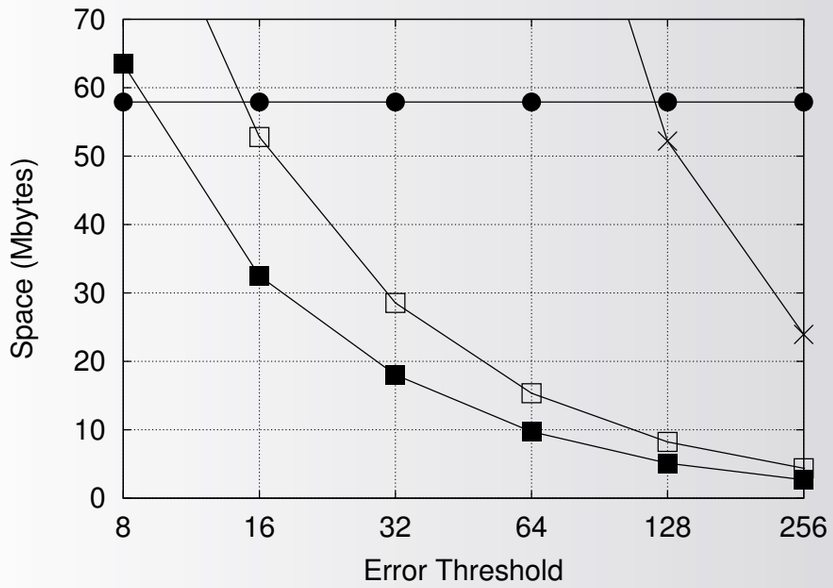
Occurrence Estimation

with A. Orlandi, PODS 2011

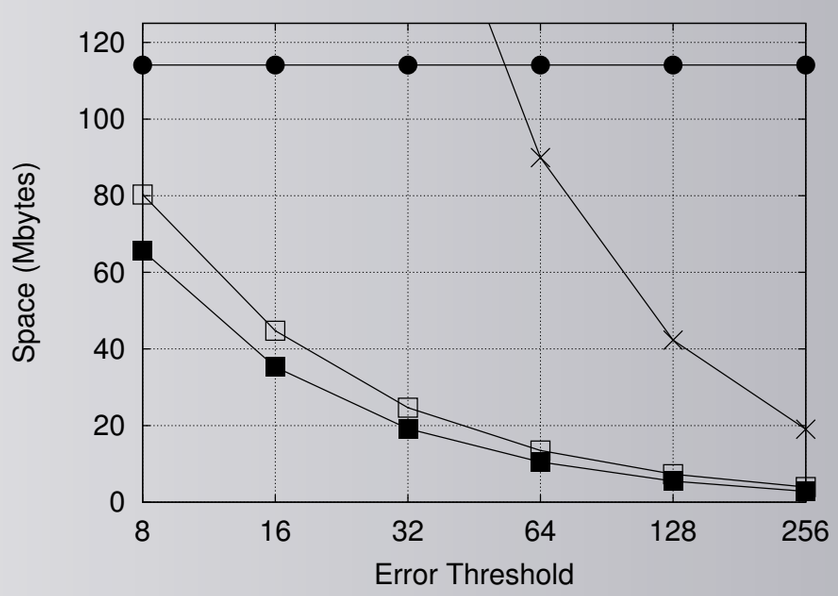
- A (large) text T is preprocessed and a (small) index I is built
  - I is able to estimate the number of occurrences of any given substring P in T without the need of T
  - T is not needed and I uses sublinear space
  - Results are incorrect for (at most) a term +E
- Our PODS provides an O(|P|) time space/error optimal solution
  - i.e., O(|T|/E) bits
- Application to Selectivity Estimation in DB for execution planing optimizations

Occurrence Estimation

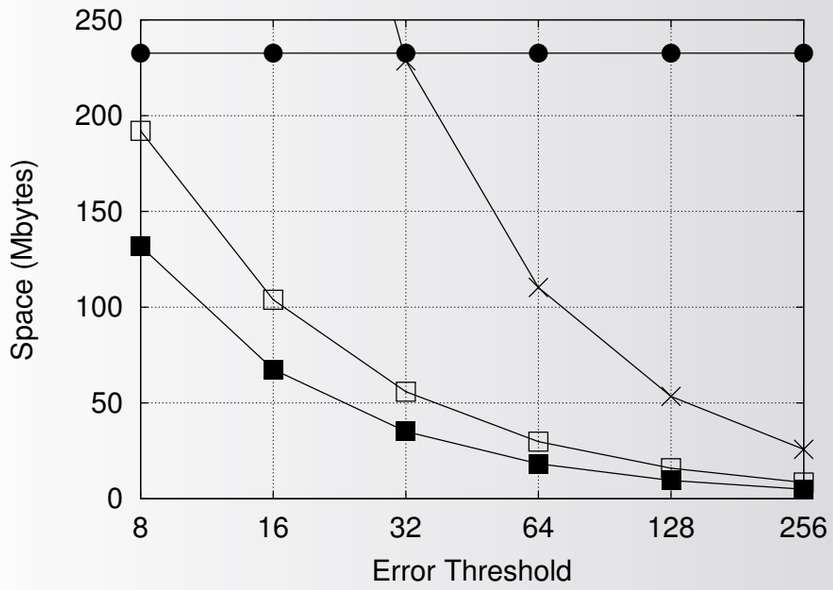
with A. Orlandi, PODS 2011



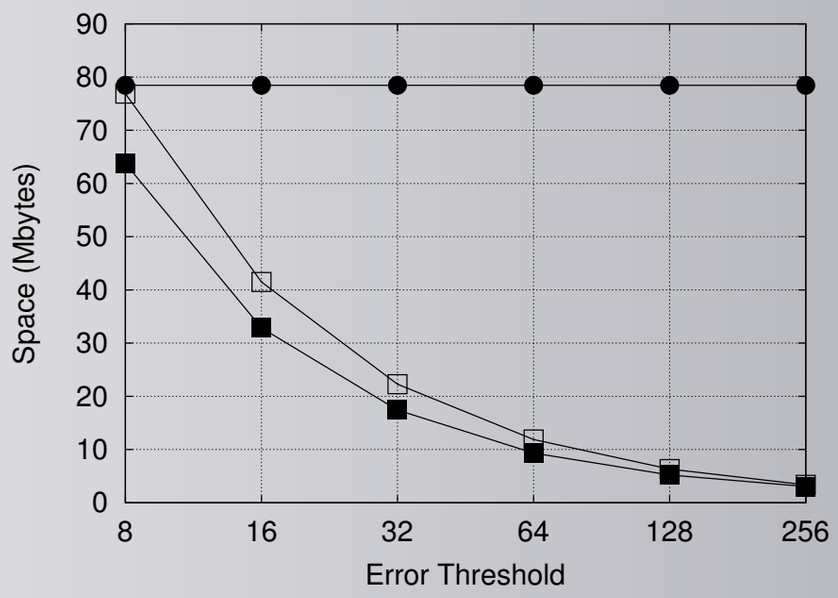
(a) `dblp`



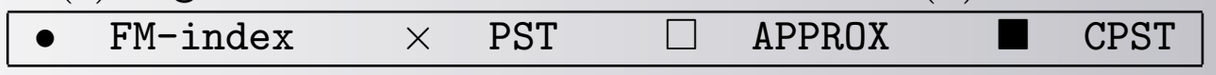
(b) `dna`



(c) `english`

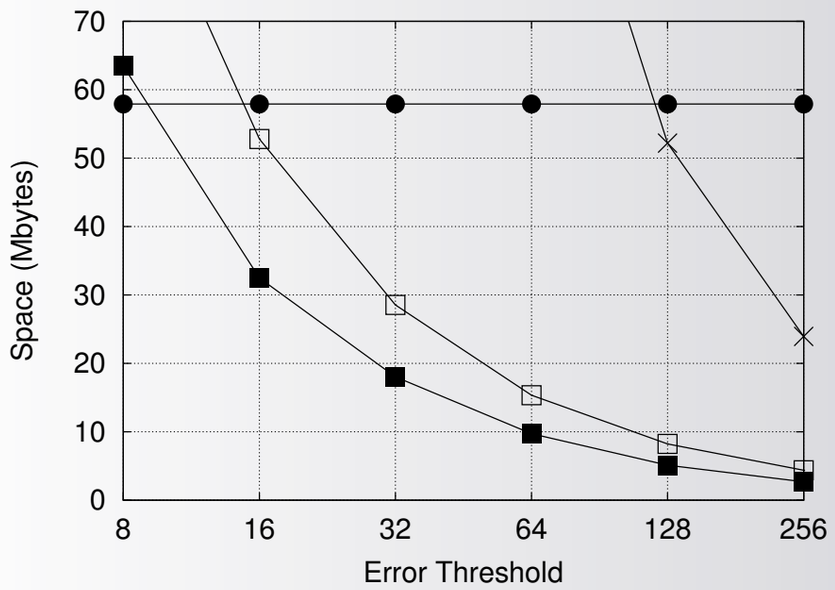


(d) `sources`

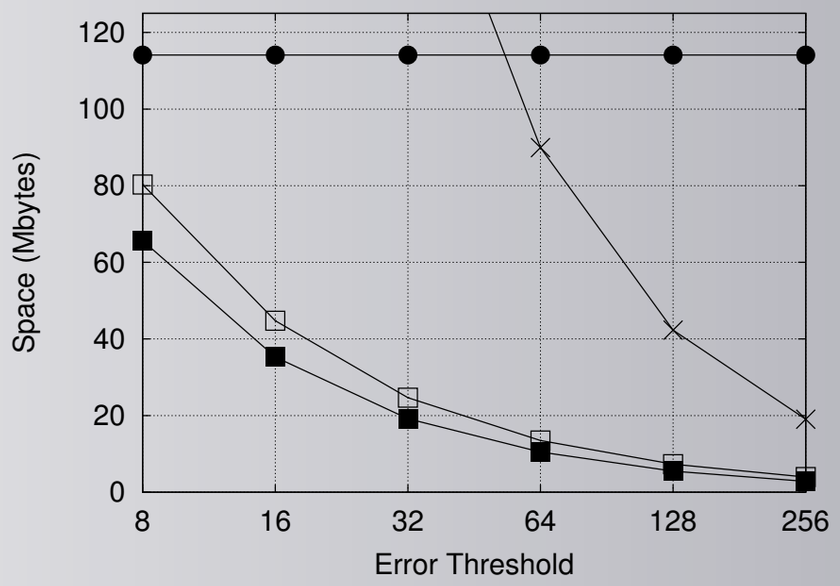


Occurrence Estimation

with A. Orlandi, PODS 2011

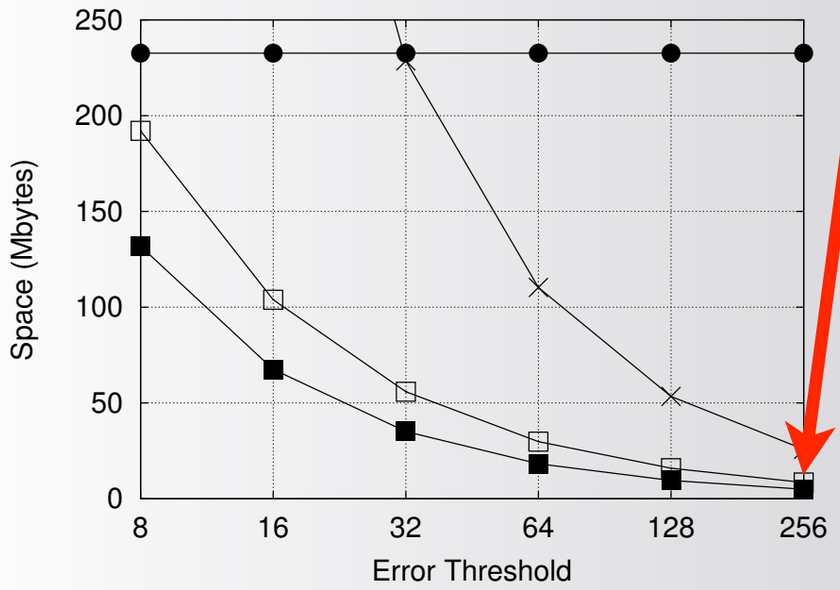


(a) dblp

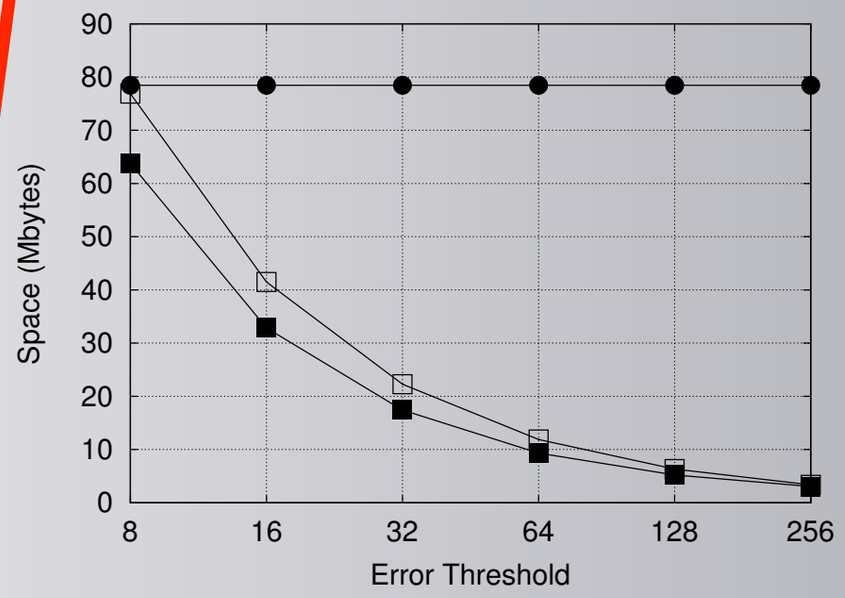


(b) dna

5 Mb, 100x smaller than text



(c) english

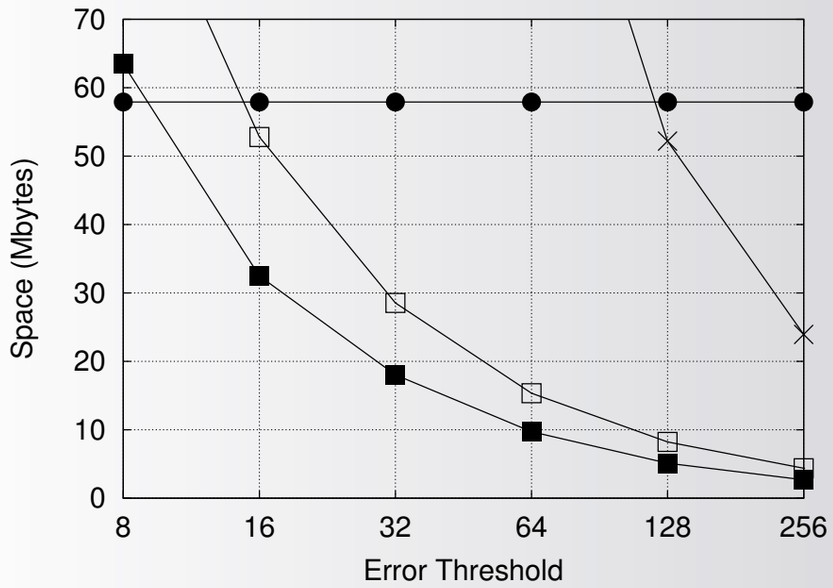


(d) sources

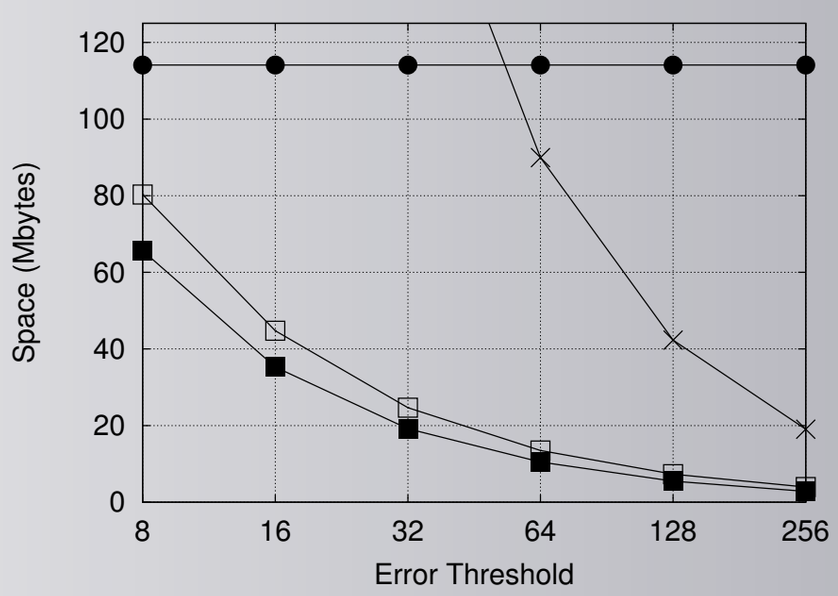
FM-index × PST □ APPROX ■ CPST

Occurrence Estimation

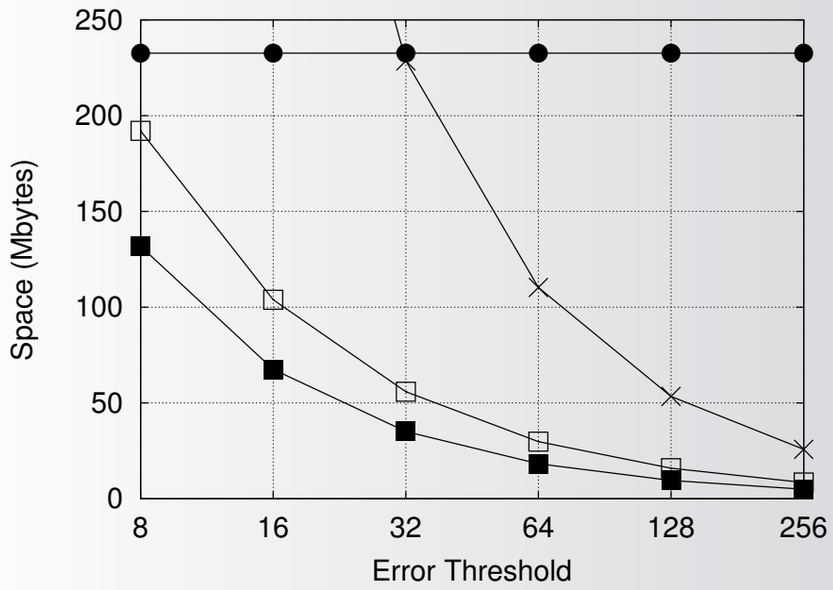
with A. Orlandi, PODS 2011



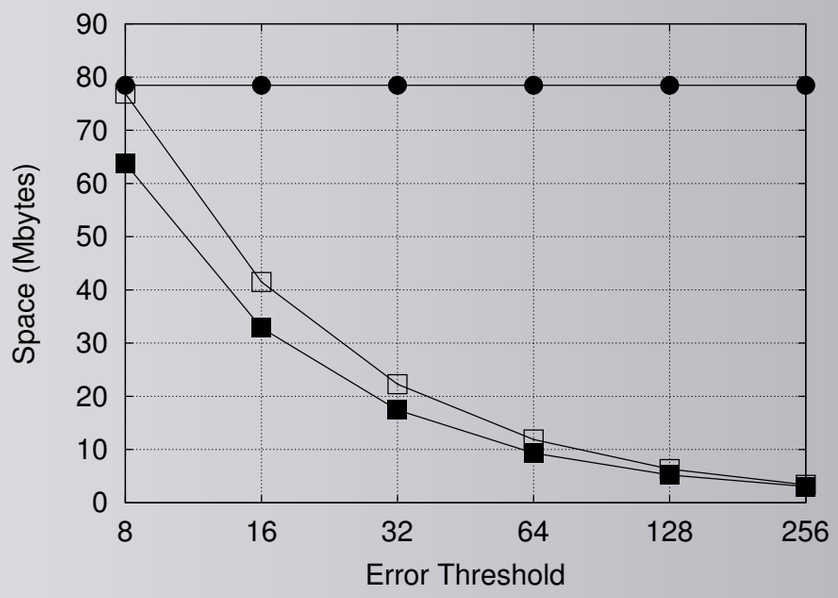
(a) `dblp`



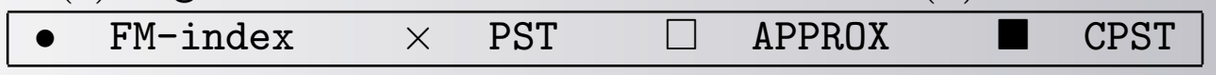
(b) `dna`



(c) `english`

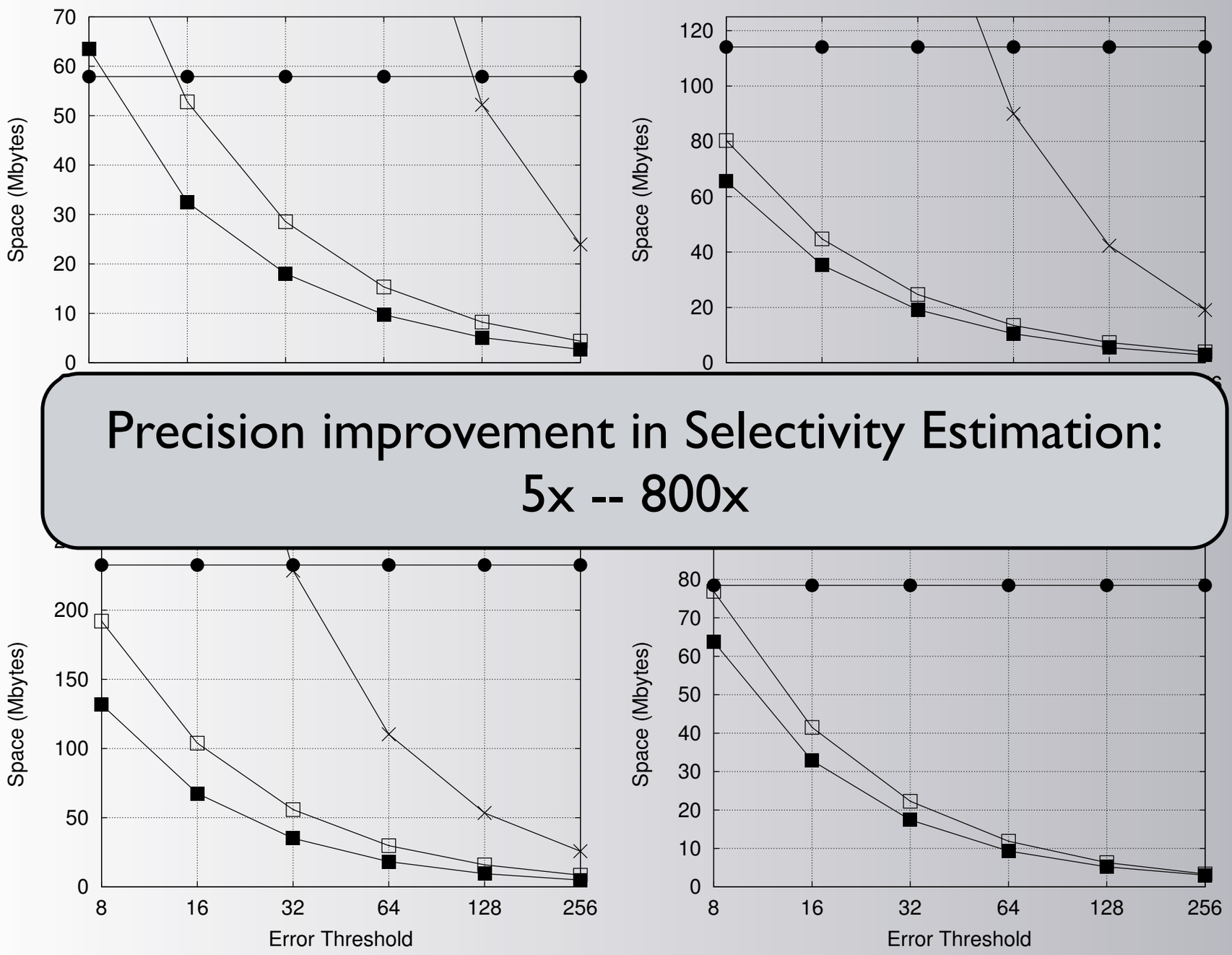


(d) `sources`



Occurrence Estimation

with A. Orlandi, PODS 2011



Precision improvement in Selectivity Estimation:
5x -- 800x

Thank You!