xkcd-repeats

Matthias Gallé & Matías Tealdi

March 2014

Motivation

Given a text, find "good" *constituents* (semantically meaningful pieces of text)

Motivation

Given a text, find "good" *constituents* (semantically meaningful pieces of text)

- NLP: bag-of-words is not enough
- Grammatical Inference: ADIOS, ABL; Smallest Grammar Problem find good set of constituents, and then relate them hierarchically

Constituents: Context matters

Constituents: Context matters

You shall know a word by the company it keeps

J.R. Firth

Constituents: Context matters

You shall know a word by the company it keeps

J.R. Firth

Automatic Word Sense Discrimination

Hinrich Schütze*
Xerox Palo Alto Research Center

This paper presents context-group discrimination, a disambiguation algorithm based on clustering. Senses are interpreted as groups (or clusters) of similar contexts of the ambiguous word. Words, contexts, and senses are represented in Word Space, a high-dimensional, real-valued space in which closeness corresponds to semantic similarity. Similarity in Word Space is based on second-order co-occurrence: two tokens (or contexts) of the ambiguous word are assigned to

Constituents: Context matters

You shall know a word by the company it keeps

J.R. Firth

Automatic Word Sense Discrimination

Hinrich Schütze*
Xerox Palo Alto Research Center

This paper presents context-group discrimination, a disambiguation algorithm based on clustering. Senses are interpreted as groups (or clusters) of similar contexts of the ambiguous word. Words, contexts, and senses are represented in Word Space, a high-dimensional, real-valued space in which closeness corresponds to semantic similarity. Similarity in Word Space is based

No direct links to stringology

Existing Classes of Constituents

Dear valued customer X . . .

Dear valued customer X . . .

Dear valued customer Y . . .

Dear valued customer Y . . .

Dear valued customer Y with respect to . . .

Dear valued customer Y with respect to . . .

Existing Classes of Constituents

Repeats

Any substring that occurs > 1

Dear valued customer X ...

Dear valued customer X ...

Dear valued customer Y ...

Dear valued customer Y ...

Dear valued customer Y with respect to ...

Dear valued customer Y with respect to ...

Existing Classes of Constituents

Repeats

Any substring that occurs > 1

Ex: There are 32: Dear, Dear valued, Dear valued customer, etc

Dear valued customer X . . .

Dear valued customer X . . .

Dear valued customer Y . . .

Dear valued customer Y . . .

Dear valued customer Y with respect to . . .

Dear valued customer Y with respect to . . .

Existing Classes of Constituents

Maximal Repeats

Cannot be extended without losing support

Ex: Dear valued customer, Dear valued customer X, Dear valued customer Y, Dear valued customer Y with respect to

Dear valued customer X . . .

Dear valued customer X . . .

Dear valued customer Y . . .

Dear valued customer Y . . .

Dear valued customer Y with respect to . . .

Dear valued customer Y with respect to . . .

Existing Classes of Constituents

Super-Maximal Repeats

Does not appear inside another one

Ex: Dear valued customer X, Dear valued customer Y with respect to

Dear valued customer X . . .

Dear valued customer X . . .

Dear valued customer Y . . .

Dear valued customer Y . . .

Dear valued customer Y with respect to . . .

Dear valued customer Y with respect to . . .

xkcd-repeats

left-context: $lc_s(\omega) = |\{s[i-1] : i \in occ_s(\omega)\}|$

right-context: $rc_s(\omega) = |\{s[i+|\omega|] : i \in occ_s(\omega)\}|$

Definition

A word ω is $\langle x, k \rangle$–context-diverse iff $lc_s(\omega) \geq x$ and $rc_s(\omega) \geq k$

Properties

- A family of repeat classes
  - ω is a maximal repeat iff it is $\langle 2, 2 \rangle$-cd
  - ω is a super-maximal repeat iff it is $\langle |occ(\omega)|, |occ(\omega)| \rangle$-cd

Properties

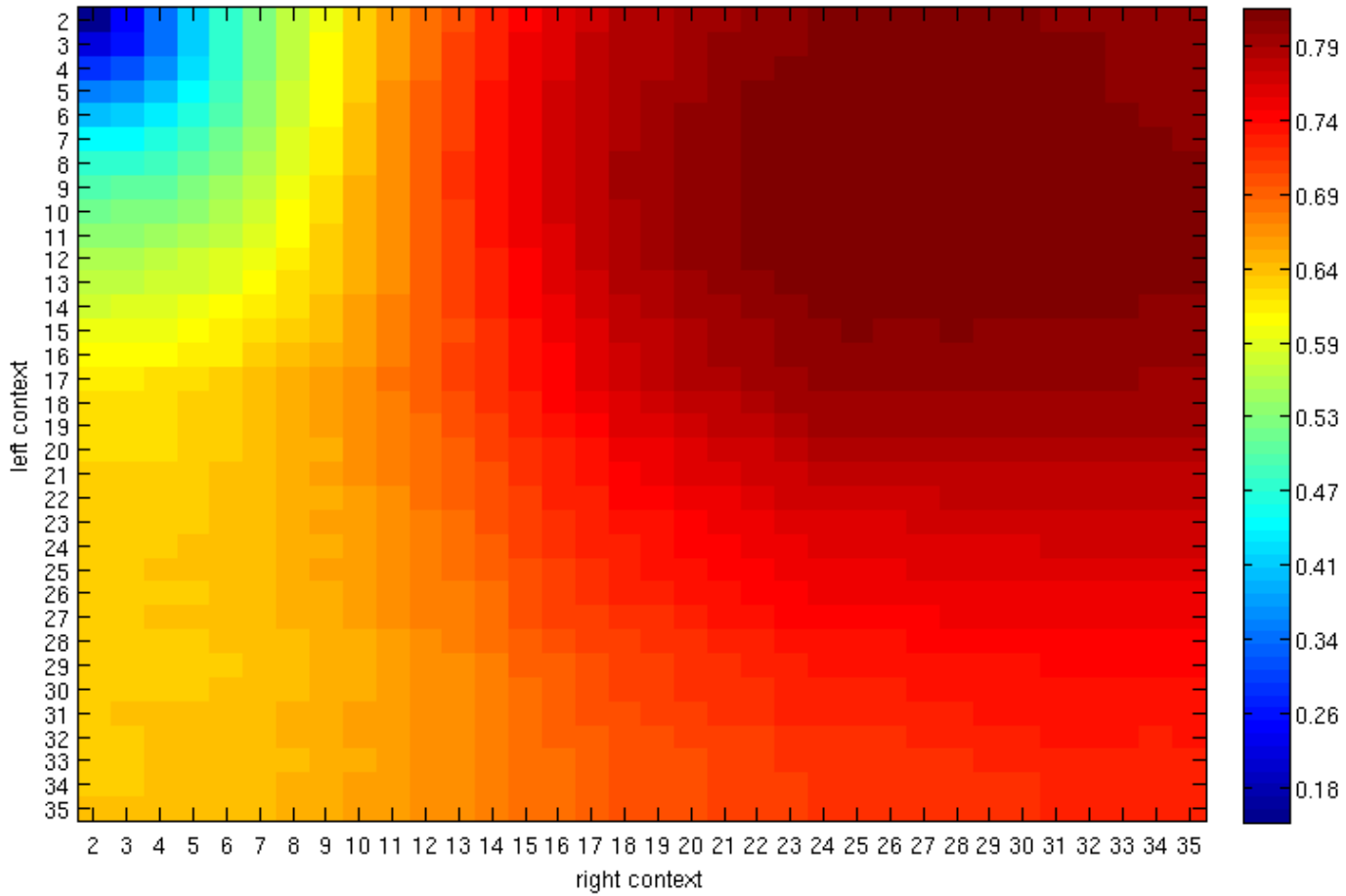
- A family of repeat classes
  - ω is a maximal repeat iff it is $\langle 2, 2 \rangle$-cd
  - ω is a super-maximal repeat iff it is $\langle |occ(\omega)|, |occ(\omega)| \rangle$-cd
- Bounds: the number of *xkcd*-repeats is $\mathcal{O}(n)$ iff $\max(x, k) \geq 2$

Motivation Confirmation

F_1 measure of constituents on the Penn Treebank (POS tags)

Motivation Confirmation

F_1 measure of constituents on the Penn Treebank (POS tags)



Algorithms

1. The optimal one: $\mathcal{O}(n)$
2. The easy one: $\mathcal{O}(|\Sigma|n)$
3. The fast one: $\mathcal{O}(n \log n)$

The optimal one: $\mathcal{O}(n)$

Computation of right-maximal is easy (number of children on the suffix tree)

1. Compute x–context-diverse
2. Compute k–context-diverse on reversed string
3. Merge them in *linear time*

Computing right context with suffix array

lcp	**suffix**
...	...
0	$\alpha\$$
$\lvert\alpha\rvert$	$\alpha ab \ldots$
$\lvert\alpha\rvert + 1$	$\alpha ac \ldots$
$\lvert\alpha\rvert$	$\alpha b \ldots$
$\lvert\alpha\rvert$	$\alpha c \ldots$
0	$\beta \ldots$
...	...

Computing right context with suffix array

lcp	suffix
...	...
0	$\alpha\$$
$\lvert\alpha\rvert$	$\alpha ab \ldots$
$\lvert\alpha\rvert + 1$	$\alpha ac \ldots$
$\lvert\alpha\rvert$	$\alpha b \ldots$
$\lvert\alpha\rvert$	$\alpha c \ldots$
0	$\beta \ldots$
...	...

stack	
α	2

Computing right context with suffix array

lcp	suffix
$\ldots$	$\ldots$
0	$\alpha\$$
$\lvert\alpha\rvert$	$\alpha ab\ldots$
$\lvert\alpha\rvert + 1$	$\alpha ac\ldots$
$\lvert\alpha\rvert$	$\alpha b\ldots$
$\lvert\alpha\rvert$	$\alpha c\ldots$
0	$\beta\ldots$
$\ldots$	$\ldots$

stack	
αa	2
α	2

Computing right context with suffix array

lcp	**suffix**
$\ldots$	$\ldots$
0	$\alpha\$$
$\lvert\alpha\rvert$	$\alpha ab \ldots$
$\lvert\alpha\rvert + 1$	$\alpha ac \ldots$
$\lvert\alpha\rvert$	$\alpha b \ldots$
$\lvert\alpha\rvert$	$\alpha c \ldots$
0	$\beta \ldots$
$\ldots$	$\ldots$

stack	*.pop()*
αa	2
α	3

Computing right context with suffix array

lcp	suffix
$\ldots$	$\ldots$
0	$\alpha\$$
$\lvert\alpha\rvert$	$\alpha ab\ldots$
$\lvert\alpha\rvert + 1$	$\alpha ac\ldots$
$\lvert\alpha\rvert$	$\alpha b\ldots$
$\lvert\alpha\rvert$	$\alpha c\ldots$
0	$\beta\ldots$
$\ldots$	$\ldots$

stack	
α	4

Computing right context with suffix array

lcp	**suffix**
...	...
0	$\alpha\$$
$\lvert\alpha\rvert$	$\alpha ab \ldots$
$\lvert\alpha\rvert + 1$	$\alpha ac \ldots$
$\lvert\alpha\rvert$	$\alpha b \ldots$
$\lvert\alpha\rvert$	$\alpha c \ldots$
0	$\beta \ldots$
...	...

stack	*.pop*()
α	4

The easy one: $\mathcal{O}(|\Sigma|n)$

lc	lcp	suffix
...	...	...
a	0	$\alpha\$$
b	$\|\alpha\|$	$\alpha ab \ldots$
c	$\|\alpha\| + 1$	$\alpha ac \ldots$
c	$\|\alpha\|$	$\alpha b \ldots$
d	$\|\alpha\|$	$\alpha c \ldots$
a	0	$\beta \ldots$
...	...	...

The easy one: $\mathcal{O}(|\Sigma|n)$

lc	lcp	suffix
...	...	...
a	0	$\alpha\$$
b	$\|\alpha\|$	$\alpha ab \ldots$
c	$\|\alpha\| + 1$	$\alpha ac \ldots$
c	$\|\alpha\|$	$\alpha b \ldots$
d	$\|\alpha\|$	$\alpha c \ldots$
a	0	$\beta \ldots$
...	...	...

stack		
α	2	$\{a, b\}$

The easy one: $\mathcal{O}(|\Sigma|n)$

lc	lcp	suffix
...	...	...
a	0	$\alpha\$$
b	$\|\alpha\|$	$\alpha ab \ldots$
c	$\|\alpha\| + 1$	$\alpha ac \ldots$
c	$\|\alpha\|$	$\alpha b \ldots$
d	$\|\alpha\|$	$\alpha c \ldots$
a	0	$\beta \ldots$
...	...	...

stack		
αa	2	$\{b, c\}$
α	2	$\{a, b\}$

The easy one: $\mathcal{O}(|\Sigma|n)$

lc	lcp	suffix
...	...	...
a	0	$\alpha\$$
b	$\|\alpha\|$	$\alpha ab \ldots$
c	$\|\alpha\| + 1$	$\alpha ac \ldots$
c	$\|\alpha\|$	$\alpha b \ldots$
d	$\|\alpha\|$	$\alpha c \ldots$
a	0	$\beta \ldots$
...	...	...

stack		.pop()
αa	2	$\{b, c\}$
α	3	$\{a, b, c\}$

The easy one: $\mathcal{O}(|\Sigma|n)$

lc	lcp	suffix
$\ldots$	$\ldots$	$\ldots$
a	0	$\alpha\$$
b	$\lvert\alpha\rvert$	$\alpha ab\ldots$
c	$\lvert\alpha\rvert + 1$	$\alpha ac\ldots$
c	$\lvert\alpha\rvert$	$\alpha b\ldots$
d	$\lvert\alpha\rvert$	$\alpha c\ldots$
a	0	$\beta\ldots$
$\ldots$	$\ldots$	$\ldots$

stack		
α	4	$\{a, b, c, d\}$

The easy one: $\mathcal{O}(|\Sigma|n)$

lc	lcp	suffix
...	...	...
a	0	$\alpha\$$
b	$\|\alpha\|$	$\alpha ab \ldots$
c	$\|\alpha\| + 1$	$\alpha ac \ldots$
c	$\|\alpha\|$	$\alpha b \ldots$
d	$\|\alpha\|$	$\alpha c \ldots$
a	0	$\beta \ldots$
...	...	...

stack	$.pop()$	
α	4	$\{a, b, c, d\}$

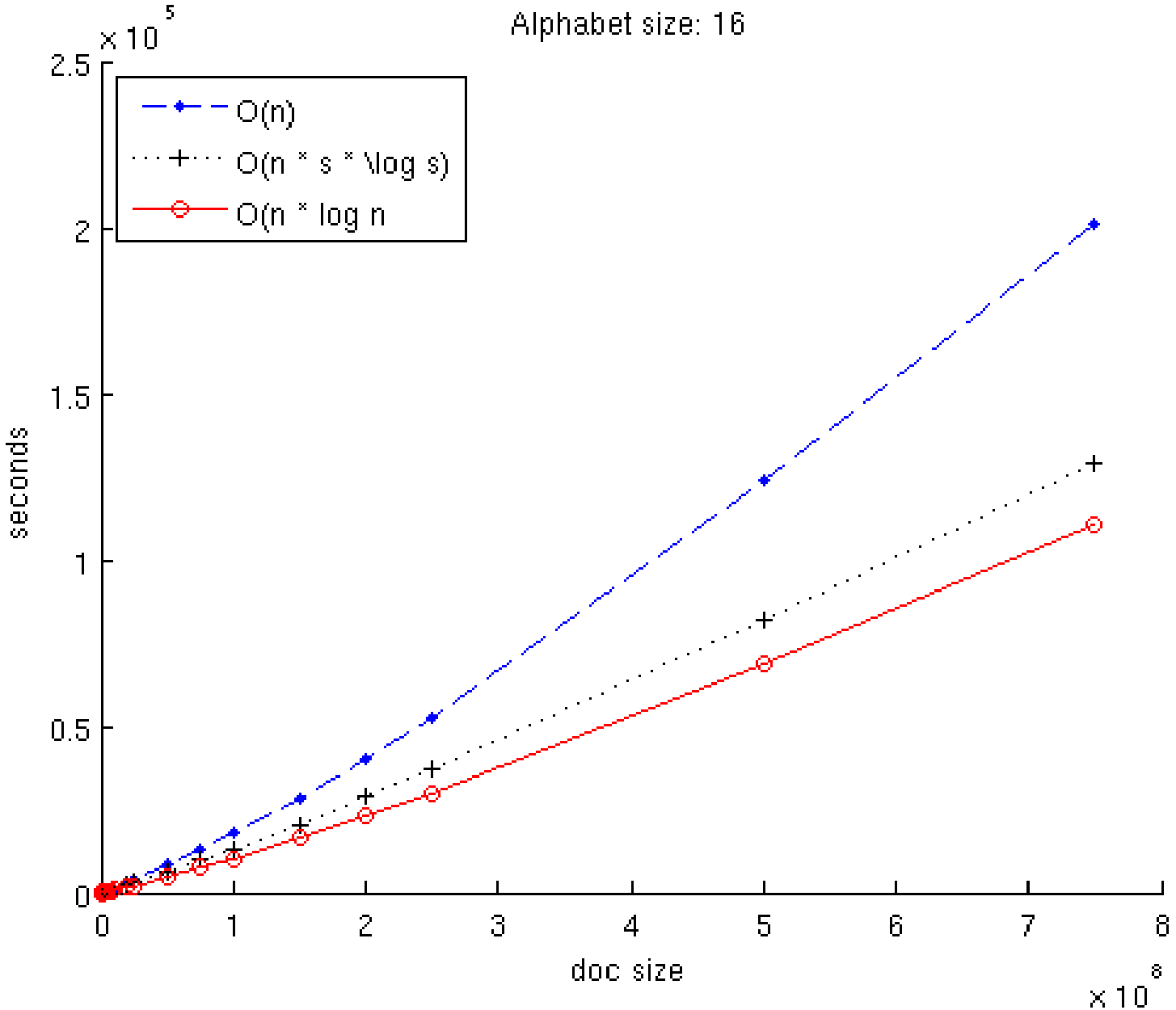
The fast one: $\mathcal{O}(n \log n)$

- Count different symbols between i and j over lc array
- Colour Counting: state-of-the-art $\mathcal{O}(\log n / \log \log n)$
- We used a simpler solution based on Fenwick trees (BIT's):
  - given an array of numbers, permit to
    - compute prefix sum's
    - modify

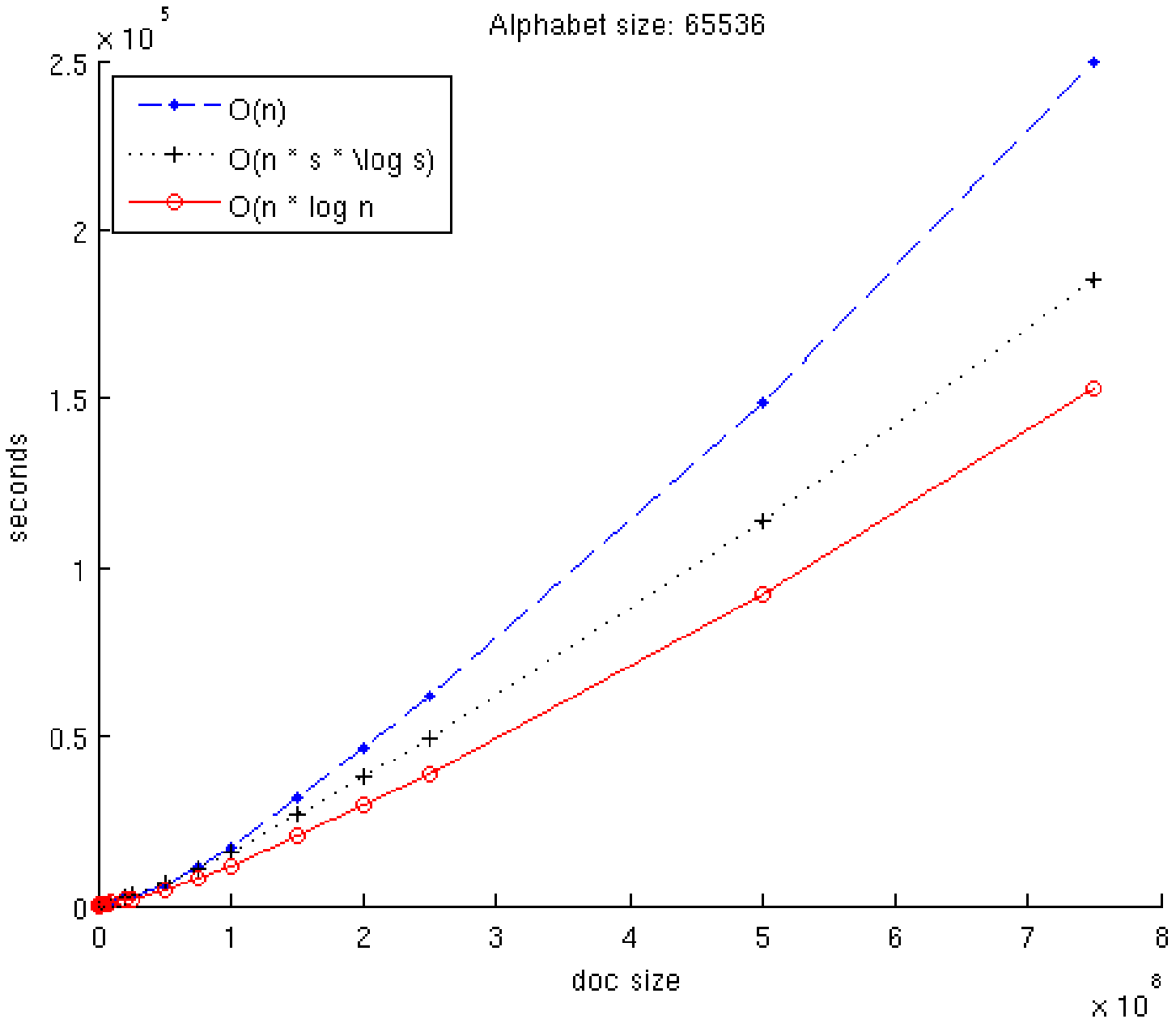
    in $\log n$ time

Comparison

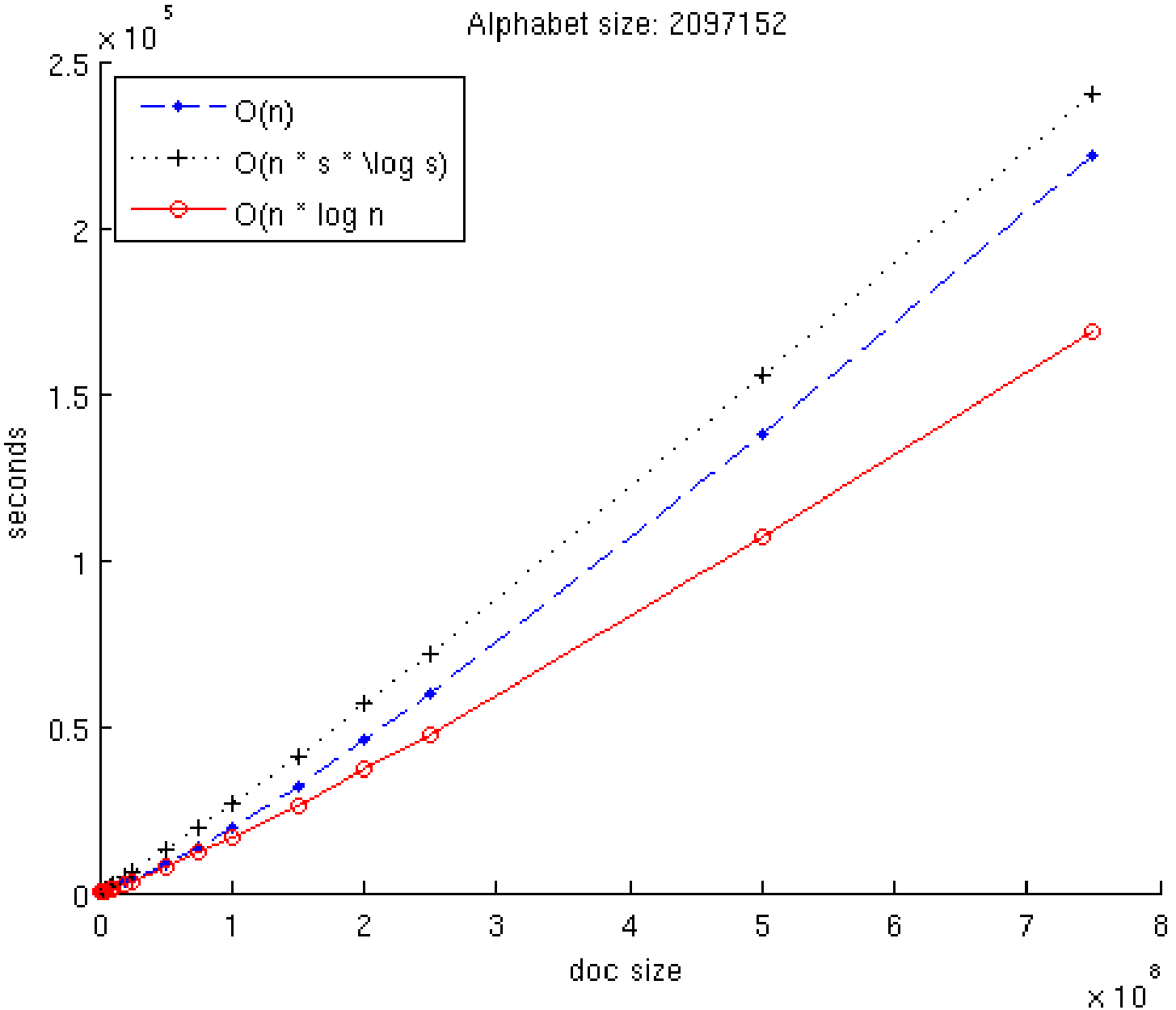
Comparison


Alphabet size: 16
× 10^5
2.5
2
1.5
1
0.5
0
seconds
O(n)
O(n * s * \log s)
O(n * log n
0
1
2
3
4
5
6
7
8
doc size
× 10^6

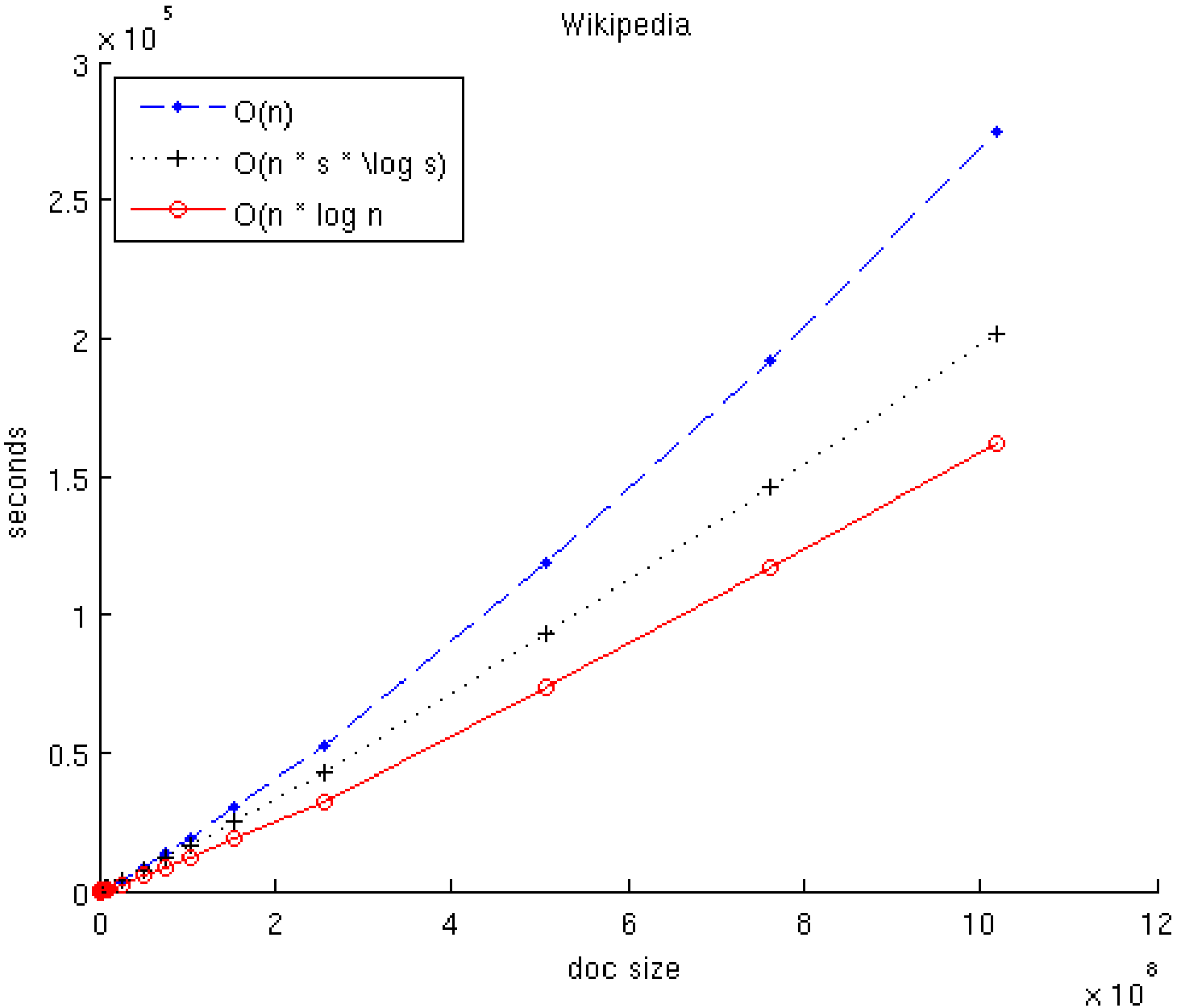
Comparison


Alphabet size: 65536
× 10^5
2.5
2
1.5
1
0.5
0
seconds
O(n)
O(n * s * \log s)
O(n * log n
0
1
2
3
4
5
6
7
8
doc size
× 10^6

Comparison


Alphabet size: 2097152
x 10^5
2.5
2
1.5
1
0.5
0
seconds
O(n)
O(n * s * \log s)
O(n * log n
0
1
2
3
4
5
6
7
8
doc size
x 10^6

Comparison


Wikipedia
x 10^5
3
2.5
2
1.5
1
0.5
0
seconds
O(n)
O(n * s * \log s)
O(n * log n
0
2
4
6
8
10
12
doc size
x 10^6

Conclusions

- Importance of including context in definition of constituents
- *xkcd*-repeats: family of classes of repeats
- Optimal and fast ($\neq$) algorithms to compute them

Remarks and other stuff

- Iterative computation of repeats
- The linear algorithm needs the construction of two suffix array's: bulk of the time


xerox