Prime Time for Minimal–Interval Semantics

Paolo Boldi **Sebastiano Vigna**
DSI, Università degli Studi di Milano, Italy

A Crash Course in Minimal–Interval Semantics

A Crash Course in Minimal–Interval Semantics

- Given a query using AND/OR, its minimal-interval semantics for a given document is a set of regions of text of the document

A Crash Course in Minimal–Interval Semantics

- Given a query using AND/OR, its minimal-interval semantics for a given document is a set of regions of text of the document

- We number the words within a document, so regions are represented by *intervals* $[\ell \,..\, r]$

A Crash Course in Minimal–Interval Semantics

- Given a query using AND/OR, its minimal-interval semantics for a given document is a set of regions of text of the document
- We number the words within a document, so regions are represented by *intervals* $[\ell \mathinner{.\,.} r]$
- The intervals are *minimal*: no interval is contained in another one

A Crash Course in Minimal–Interval Semantics

- Given a query using AND/OR, its minimal-interval semantics for a given document is a set of regions of text of the document
- We number the words within a document, so regions are represented by *intervals* $[\ell \mathinner{.\,.} r]$
- The intervals are *minimal*: no interval is contained in another one

- In other words, intervals form an *antichain* with respect to inclusion

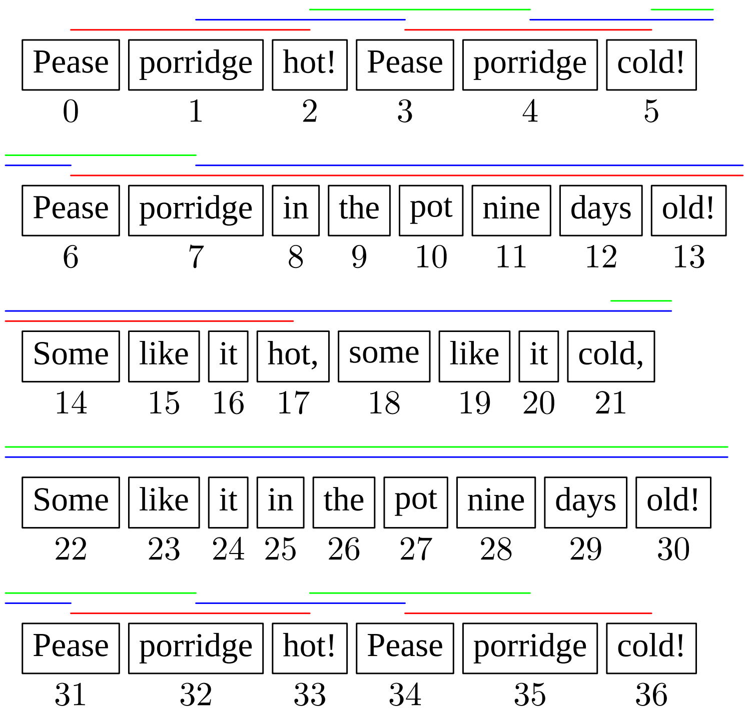
A Crash Course in Minimal–Interval Semantics

- Given a query using AND/OR, its minimal-interval semantics for a given document is a set of regions of text of the document
- We number the words within a document, so regions are represented by *intervals* $[\ell \,..\, r]$
- The intervals are *minimal*: no interval is contained in another one
- In other words, intervals form an *antichain* with respect to inclusion
- Very natural: AND of terms has as semantics the smallest regions of text containing the terms

A Crash Course in Minimal–Interval Semantics

- Given a query using AND/OR, its minimal-interval semantics for a given document is a set of regions of text of the document
- We number the words within a document, so regions are represented by *intervals* $[\ell \,..\, r]$
- The intervals are *minimal*: no interval is contained in another one
- In other words, intervals form an *antichain* with respect to inclusion
- Very natural: AND of terms has as semantics the smallest regions of text containing the terms
- Introduced by Clarke, Cormack and Burkowski in ’95.

Why Minimal–Interval Semantics?



Pease porridge hot! Pease porridge cold!
0 1 2 3 4 5

Pease porridge in the pot nine days old!
6 7 8 9 10 11 12 13

Some like it hot, some like it cold,
14 15 16 17 18 19 20 21

Some like it in the pot nine days old!
22 23 24 25 26 27 28 29 30

Pease porridge hot! Pease porridge cold!
31 32 33 34 35 36

(hot OR cold) AND porridge AND pease

Why Minimal–Interval Semantics?

Why Minimal–Interval Semantics?

- Antichains form a lattice, so you can evaluate any Boolean formula

Why Minimal–Interval Semantics?

- Antichains form a lattice, so you can evaluate any Boolean formula
- Minimal-interval semantics makes it trivial to compute. . .

Why Minimal–Interval Semantics?

- Antichains form a lattice, so you can evaluate any Boolean formula
- Minimal-interval semantics makes it trivial to compute. . .
    - Phrasal queries: there's a simple linear algorithm

Why Minimal–Interval Semantics?

- Antichains form a lattice, so you can evaluate any Boolean formula
- Minimal-interval semantics makes it trivial to compute. . .
    - Phrasal queries: there's a simple linear algorithm

    - Proximity restrictions: measure the shortest interval in the antichain and check that it satisfies the restriction

Why Minimal–Interval Semantics?

- Antichains form a lattice, so you can evaluate any Boolean formula
- Minimal-interval semantics makes it trivial to compute. . .
  - Phrasal queries: there's a simple linear algorithm
  - Proximity restrictions: measure the shortest interval in the antichain and check that it satisfies the restriction
  - Ordered conjunction (I want this words *in this order*, possibly with something else inbetween)

Why Minimal–Interval Semantics?

- Antichains form a lattice, so you can evaluate any Boolean formula
- Minimal-interval semantics makes it trivial to compute. . .
  - Phrasal queries: there's a simple linear algorithm
  - Proximity restrictions: measure the shortest interval in the antichain and check that it satisfies the restriction
  - Ordered conjunction (I want this words *in this order*, possibly with something else inbetween)

  - . . . and all of these can be freely combined!

Why Minimal–Interval Semantics?

- Antichains form a lattice, so you can evaluate any Boolean formula
- Minimal-interval semantics makes it trivial to compute. . .
  - Phrasal queries: there's a simple linear algorithm
  - Proximity restrictions: measure the shortest interval in the antichain and check that it satisfies the restriction
  - Ordered conjunction (I want this words *in this order*, possibly with something else inbetween)
  - . . . and all of these can be freely combined!
  - High-quality snippets

Why Minimal–Interval Semantics?

- Antichains form a lattice, so you can evaluate any Boolean formula
- Minimal-interval semantics makes it trivial to compute. . .
  - Phrasal queries: there's a simple linear algorithm
  - Proximity restrictions: measure the shortest interval in the antichain and check that it satisfies the restriction
  - Ordered conjunction (I want this words *in this order*, possibly with something else inbetween)
  - . . . and all of these can be freely combined!
  - High-quality snippets

- . . . and so on.

How to compute it?

How to compute it?

- The semantics of a term is the set of one-point intervals in which the term appears

How to compute it?

- The semantics of a term is the set of one-point intervals in which the term appears
- The OR of antichains is computed by putting together the antichains, comparing each pair of intervals and eliminating the nonminimal ones

How to compute it?

- The semantics of a term is the set of one-point intervals in which the term appears
- The OR of antichains is computed by putting together the antichains, comparing each pair of intervals and eliminating the nonminimal ones

- The AND of antichains is computed as follows: take one interval from the first antichain, one interval from the second one and so on; compute the resulting spanned interval; do this for each tuple of intervals; eliminate non-minimal intervals

How to compute it?

- The semantics of a term is the set of one-point intervals in which the term appears
- The OR of antichains is computed by putting together the antichains, comparing each pair of intervals and eliminating the nonminimal ones
- The AND of antichains is computed as follows: take one interval from the first antichain, one interval from the second one and so on; compute the resulting spanned interval; do this for each tuple of intervals; eliminate non-minimal intervals
- Looks complicated? It's actually very natural!

How to compute it?

Pease	porridge	hot!	Pease	porridge	cold!	Pease	porridge	in	the	pot	nine	days	old!	Some	like	it	hot,	some	like	it	cold,
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21

(hot OR cold) AND porridge AND pease

How to compute it?

Pease	porridge	hot!	Pease	porridge	cold!	Pease	porridge	in	the	pot	nine	days	old!	Some	like	it	hot,	some	like	it	cold,
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21

(hot OR cold) AND porridge AND pease

How to compute it?

Pease	porridge	hot!	Pease	porridge	cold!	Pease	porridge	in	the	pot	nine	days	old!	Some	like	it	hot,	some	like	it	cold,
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21

(hot OR cold) AND porridge AND pease

How to compute it?

Pease	porridge	hot!	Pease	porridge	cold!	Pease	porridge	in	the	pot	nine	days	old!	Some	like	it	hot,	some	like	it	cold,
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21

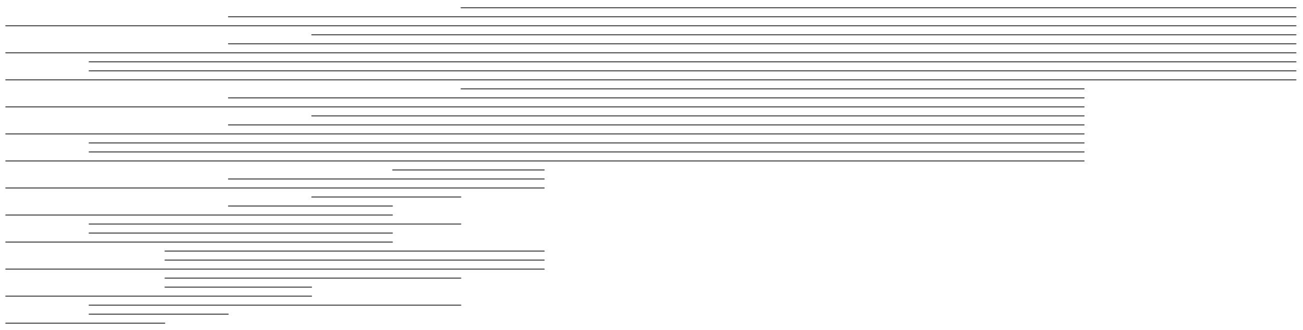
(hot OR cold) AND porridge AND pease

How to compute it?

Pease	porridge	hot!	Pease	porridge	cold!	Pease	porridge	in	the	pot	nine	days	old!	Some	like	it	hot,	some	like	it	cold,
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21

(hot OR cold) AND porridge AND pease

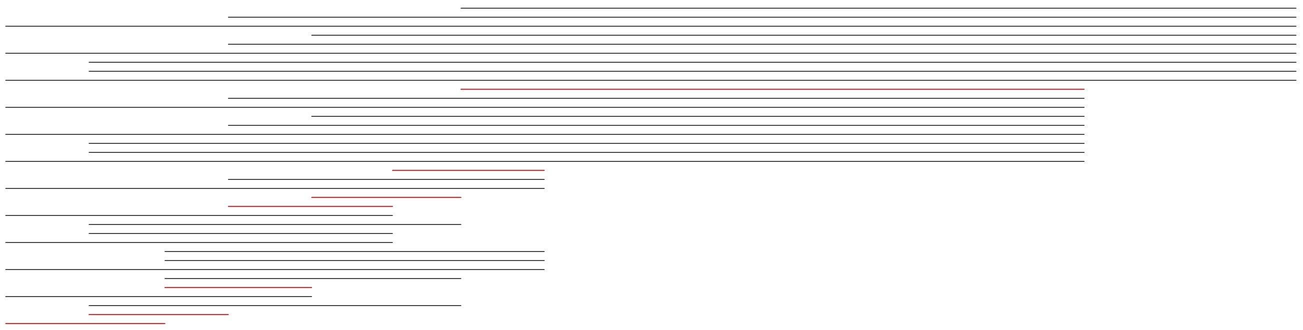
How to compute it?



Pease	porridge	hot!	Pease	porridge	cold!	Pease	porridge	in	the	pot	nine	days	old!	Some	like	it	hot,	some	like	it	cold,
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21

(hot OR cold) AND porridge AND pease

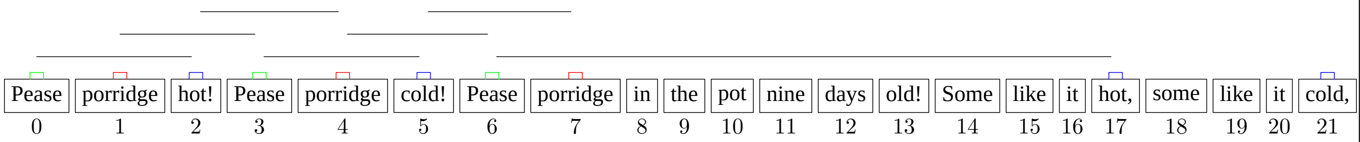
How to compute it?



Pease	porridge	hot!	Pease	porridge	cold!	Pease	porridge	in	the	pot	nine	days	old!	Some	like	it	hot,	some	like	it	cold,
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21

(hot OR cold) AND porridge AND pease

How to compute it?



Pease	porridge	hot!	Pease	porridge	cold!	Pease	porridge	in	the	pot	nine	days	old!	Some	like	it	hot,	some	like	it	cold,
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21

(hot OR cold) AND porridge AND pease

You Can Do It Quickly!

You Can Do It Quickly!

- The original algorithms provided in ’95 were eager, and computing operators required random access to the component antichains ($ns \log m$ for n operators, s results, and m overall input intervals given in sorted arrays)

You Can Do It Quickly!

- The original algorithms provided in ’95 were eager, and computing operators required random access to the component antichains ($ns \log m$ for n operators, s results, and m overall input intervals given in sorted arrays)

- We discovered (almost) linear lazy algorithms ($m \log n$ with input intervals arriving lazily from lists) which require no more computation than standard proximity computation, yet provide much more valuable data

You Can Do It Quickly!

- The original algorithms provided in '95 were eager, and computing operators required random access to the component antichains ($ns \log m$ for n operators, s results, and m overall input intervals given in sorted arrays)
- We discovered (almost) linear lazy algorithms ($m \log n$ with input intervals arriving lazily from lists) which require no more computation than standard proximity computation, yet provide much more valuable data
- Antichains are by definition at most as large as the number of words in the document

You Can Do It Quickly!

- The original algorithms provided in '95 were eager, and computing operators required random access to the component antichains ($ns \log m$ for n operators, s results, and m overall input intervals given in sorted arrays)
- We discovered (almost) linear lazy algorithms ($m \log n$ with input intervals arriving lazily from lists) which require no more computation than standard proximity computation, yet provide much more valuable data
- Antichains are by definition at most as large as the number of words in the document

- We believe it scales linearly to the web

Can We Get Out Of The Boolean Tunnel?

Can We Get Out Of The Boolean Tunnel?

- Search engines have traditionally provided Boolean and phrasal operators

Can We Get Out Of The Boolean Tunnel?

- Search engines have traditionally provided Boolean and phrasal operators
- The additional operators provided by minimal-interval semantics are very intuitive!

Can We Get Out Of The Boolean Tunnel?

- Search engines have traditionally provided Boolean and phrasal operators
- The additional operators provided by minimal-interval semantics are very intuitive!

- Proximity restriction is what every user of a search engine at some point craved for (after getting answers from long mailing lists or blogs)

Can We Get Out Of The Boolean Tunnel?

- Search engines have traditionally provided Boolean and phrasal operators
- The additional operators provided by minimal-interval semantics are very intuitive!
- Proximity restriction is what every user of a search engine at some point craved for (after getting answers from long mailing lists or blogs)
- Ordered conjunction can be used when you don't know all the words of a verse of a song

Can We Get Out Of The Boolean Tunnel?

- Search engines have traditionally provided Boolean and phrasal operators
- The additional operators provided by minimal-interval semantics are very intuitive!
- Proximity restriction is what every user of a search engine at some point craved for (after getting answers from long mailing lists or blogs)
- Ordered conjunction can be used when you don't know all the words of a verse of a song

- Ordered conjunction can be also used to implement phrasal searches with wildcards (also very natural)

Wanna Try?

Wanna Try?

- Have a look at MG4J (`http://mg4j.dsi.unimi.it/`)

Wanna Try?

- Have a look at MG4J (`http://mg4j.dsi.unimi.it/`)
- Full, free Java implementation of our new algorithms for minimal-interval semantics

Wanna Try?

- Have a look at MG4J (`http://mg4j.dsi.unimi.it/`)
- Full, free Java implementation of our new algorithms for minimal-interval semantics

- Easy testbed for new ideas

Wanna Try?

- Have a look at MG4J (`http://mg4j.dsi.unimi.it/`)
- Full, free Java implementation of our new algorithms for minimal-interval semantics
- Easy testbed for new ideas
- An application: Twease (`http://twease.org/`) indexes 80 million sentences from medical literature

Wanna Try?

- Have a look at MG4J (`http://mg4j.dsi.unimi.it/`)
- Full, free Java implementation of our new algorithms for minimal-interval semantics
- Easy testbed for new ideas
- An application: Twease (`http://twease.org/`) indexes 80 million sentences from medical literature

- Albeit unremarkable in precision/recall, MG4J was by far the largest contributor of unique relevant documents to TREC 2005: searching with powerful operators pays