Prime Time for Minimal–Interval Semantics

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

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* $[\ell, r]$. 
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.
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 $[\ell \ldots r]$.
- 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?

(hot OR cold) AND porridge AND pease
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
  - 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).

...and all of these can be freely combined!
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 in between)
  - ... and all of these can be freely combined!
  - High-quality snippets

... and so on.
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.
- 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?

(\text{hot OR cold}) \text{ AND} \ \text{porridge} \ \text{AND} \ \text{pease}
### How to compute it?

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

(hot OR cold) AND porridge AND pease
How to compute it?

(hot OR cold) AND porridge AND pease
<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
<th>18</th>
<th>19</th>
<th>20</th>
<th>21</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pease</td>
<td>porridge</td>
<td>hot!</td>
<td>Pease</td>
<td>porridge</td>
<td>cold!</td>
<td>Pease</td>
<td>porridge</td>
<td>in</td>
<td>the</td>
<td>pot</td>
<td>nine</td>
<td>days</td>
<td>old!</td>
<td>Some</td>
<td>like</td>
<td>it</td>
<td>hot,</td>
<td>some</td>
<td>like</td>
<td>it</td>
<td>cold,</td>
</tr>
</tbody>
</table>

(hot OR cold) AND porridge AND pease
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.
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.
Search engines have traditionally provided Boolean and phrasal operators
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).
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?

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

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