Edward C. Zimmermann <edz@nonmonotonic.net>

http://www.nonmonotonic.net/re-isearch

Project re-Isearch has been made possible by NONMONOTONIC.NET and a grant from NlnetNGI0. Some additional support for ISO-8601 date formats has been supplied by StandICT.

The 27 year old new kid on the search block

History

- Isearch was a legendary open-source text retrieval software first developed in 1994 as part of the Isite Z39.50 information framework with support from NSF.
- Development was divided between CNDIR/MCNC and Bsn (Germany).
- In 1998 BSn launched a proprietary fork using new algorithms.
- In was deployed in 100s of high profile sites….
- In 2011 Bsn/NONMONOTONIC's proprietary fork ceased development and it moved to the attic..

Despite lack of support some servers are still running!???

Don't break a working system…..

You mentioned sites??? Curious!

The U.S. Patent and Trademark Office (USPTO) patent search, the Federal Geographic Data Clearinghouse (FGDC), the NASA Global Change Master Directory, the NASA EOS Guide System, the NASA Catalog Interoperability Project, the astronomical pre-print service based at the Space Telescope Science Institute, The PCT Electronic Gazette at the World Intellectual Property Organization (WIPO), the SAGE Project of the Special Collections Department at Emory University, Eco Companion Australasia (an environmental geospatial resources catalog), the Open Directory Project, genomic search for the Australian National Genomic Information Service's (ANGIS) human genome project (and its eBiotechnology workbench split-off); the D-A-S-H search portal against racism, antisemitism and exclusion (funded within the framework of the action program "Youth for tolerance and democracy - against right-wing extremism, xenophobia and anti-Semitism", the YOUTH program of the European Community and with additional support from the German Federal Agency for Civic Education); the e-government search (Yeehaw) of the U.S. State of Utah to agronomic cooperation across the Mediterranean region (supported by the EU's DG). Integrated into a number of CMS platforms it powered search for a number of high volume web sites. It was also used as a database accelerator by a number of eCommerce shops.

Development had been terminated.. Loads of people asked if I could open source IB.. and we were constantly surprized at ApacheCon just how primitive the offerings were..

And thanks to the kind support of Nlnet/NGI0...

Reborn in 2020 in the middle of the global Covid19 pandemic as Project re-Isearch. Like the original, it is not just about textual words but pushes the envelope.

Motivation. What is different?

Mainstream search engines are about finding any information:

"*a list of all documents containing a specific word or phrase*".

So search engines paradoxically return both too much information (i.e. long lists of links) and too little information (i.e. links to content, not content itself).

The re-Isearch engine is, by contrast, about exploiting document structure, both implicit (XML and other markup) and explicit (visual groupings such as paragraph), to zero in on relevant sections of documents, not just links to documents.

We call it: **Smart queries and run-time (dynamic) unit of retrieval.**

What do you mean?

In "traditional" search engine models there is a standard unit of the record. Its the unit of index (the page, PDF, Word document) and retrieval. By contrast we have a user defined "search time" unit of retrieval: the structure of documents can be exploited to identify which document elements (such as the appropriate chapter or page) to retrieve. Retrieval granularity may be on the level of sub-structures of a given document or page such as line, paragraph but may also be as part of a larger collection. Since we know the location of hits (matches) within a record we can transverse its structure (which can be viewed as a graph) to find other relevant bits or retrieve relevant elements. These can even be virtual.

Pretty cool, eh!?

What do you mean by virtual?

Instead of just searching for results one can search for clusters of elements (we call them “semantic spheres”) that are relevant. Words derive their meaning in context. In a social network these are, for example, what people call “bubbles“.

On can dig deeper into these to find new insights and discover different views and aspects (and not just some monlithic “Big Brother” mediated message).

We call it multi-modal re-search!

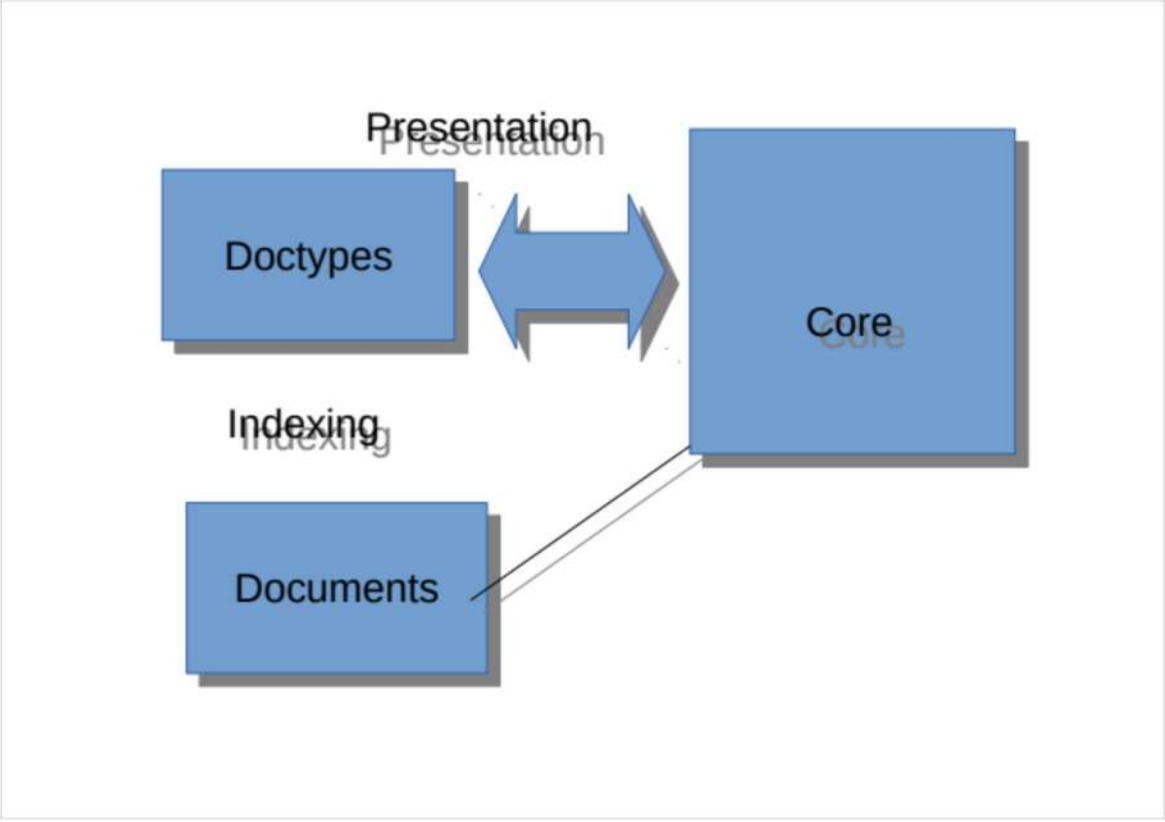
Exodus:Cross Search
ISEA 08 Singapore

Artist: Metahaven (NL)
Collaborators

Tsila Hassine
Maurits de Bruijn
Edward Zimmerman
Norbert Poëllmann
Florian Schneider - KEING.ORG
Zhou Xiangdong - Fudan University

https://isea-archives.siggraph.org/art-events/metahaven-exodus-cross-search/

Design



Index Algorithm

- Based roughly on pat-arrays going back to the original work of Gonnet, Baeza-Yates, Uri Mamber et al.
- In contrast to them it also stores a cache of the first X characters and the address range in the index. This can be mapped into memory to significantly save on I/O.
- Each word has a composite address made up from the address (key) of the file it is contained in, the file offset to the start of record (a file can contain multiple records) and the offset from the start of record to the start of the word.
- Each field (container) instance encountered get in a seperate file its start and end address stored.
- The vector of the pairs <word,address> is sorted by word. The address column is dumped to disk, the addresses whose words match the first and/or last X characters is dumped as <word fragement, start offset, end offset> where are offsets are into the index above.
- The address of every word is stored and the address range of every field instance is stored.
- An index file tracks the correlation of composite address to file and record.

What else do we do? Polymorphisms

- We also for specific fields or paths create **object specific indexes**.
- This lets us search the objects too.. and get addresses and...
- This lets us search not just text but also object (using algorithms appropriate, e.g. numerical for numerical indexes etc.).
- Example (Date fields, date indexes):

  6 Feb 2022.

  - Searching as date means same as 02062022 (or even Luty 6, 2022) , but searching for Feb also possible (which won't match Luty).

How do we address I/O bottlenecks?

- We exploit the virtual memory system of the underlying operatting system.
- The magic is a class called MMAP (built upon mmap(2) resp. CreateFileMapping/MapViewOfFile in Win32)

  mmap works by manipulating the process's page table. The CPU will translate "virtual" addresses to "physical" ones, and does so according to the page table set up by the kernel. When you access the mapped memory for the first time, your CPU generates a page fault. The OS kernel can then jump in, "fix up" the invalid memory access by allocating memory and doing file I/O in that newly allocated buffer, then continue your program's execution as if nothing happened. See the Translation Lookaside Buffer part of the MMU
- Since our accress patterns are sparse and we use random access (binary search) we get significant performace gain over file I/O-- why people develop in-memory Dbs.
- MMAP has the advantage that if the page is already in RAM the file does not need to be read. Since the use case is to typically have multiple processes accessing data (searching) in a read only fashion from the same file we win big.

We’ve indexed.. What do we now know?

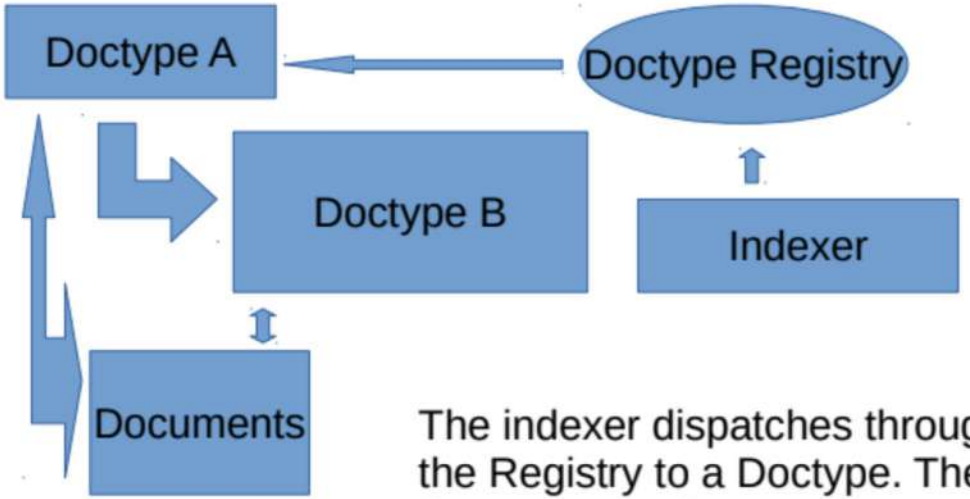
- For every word we have an address and can open the file and read it. The word could have been encoded. How it was encoded we know from index time and can have classes do the right magic (I discuss this shortly).
- Since we know the addresses in all our fields (paths) we can find the paths for any word.
- This lets us, for example, not just search for words in a specific field, a specific path but also for words in the same container without knowing its name or path.
- Since we can read the orginal file we can also search for unlimited length literals using any wildcard or regex scheme of our dreams.
- We can even go the other way and ask: what words are in a given field or path instance?
- Since we know all the words and the structure we can even reconstitute an alternative encoding without needed to parse the original (we call this “presentation syntax”, but more later).

Virtual Indexes (Collections)

- We can define a physical index to have different properties by just creating a file pointing to the index but with different parameters.
- We can also define a virtual index by just creating a file pointing to multiple indexes.
- We can also define a virtual index by just creating a file pointing to a virtual index which itself pointed to ….
- This allows for not just extreme flexibility but also the trivial creation of „on the fly“ indexes to search.

Design (Handling file types)

We talked about doctypes..



The indexer dispatches through the Registry to a Doctype. The Doctype may, in turn, pass to another doctype and so on.

What doctypes are available?

```
Available Built-in Document Base Classes (v28.8):
       AOLLIST           ATOM     AUTODETECT       BIBCOLON
        BIBTEX         BINARY            CAP       COLONDOC
      COLONGRP         CSVDOC       DIALOG-B            DIF
          DOCX        DVBLINE        ENDNOTE      EUROMEDIA
      FILMLINE         FILTER    FILTER2HTML    FILTER2MEMO
   FILTER2TEXT     FILTER2XML     FIRSTLINE            FTP
          GILS        GILSXML       HARVEST           HTML
         HTML--      HTMLCACHE      HTMLHEAD       HTMLMETA
    HTMLREMOTE       HTMLZERO       IAFADOC       IKNOWDOC
        IRLIST        ISOTEIA          JIRA          LATEX
    LISTDIGEST     MAILDIGEST    MAILFOLDER       MARKDOWN
       MEDLINE           MEMO       METADOC       MISMEDIA
    NEWSFOLDER         NEWSML           OCR            ODT
       ONELINE       OZSEARCH        PANDOC        PAPYRUS
          PARA            PDF     PLAINTEXT             PS
         PTEXT            RDF      REFERBIB            RIS
       ROADS++         RSS.9x          RSS1           RSS2
    RSSARCHIVE        RSSCORE           SGML       SGMLNORM
       SGMLTAG         SIMPLE           SOIF            TSV
        TSVDOC        XBINARY        XFILTER            XML
       XMLBASE         XMLREC     YAHOOLIST
External Base Classes ("Plugin Doctypes"):
  RTF:               // "Rich Text Format" (RTF) Plugin
  ODT:               // "OASIS Open Document Format Text" (ODT) Plugin
  ESTAT:             // EUROSTAT CSL Plugin
  EXIF:              // EXIF Plugin
  MSOFFICE:          // M$ Office OOXML Plugin
  USPAT:             // US Patents (Green Book)
  ADOBE_PDF:         // Adobe PDF Plugin
  MSOLE:             // M$ OLE type detector Plugin
  MSEXCEL:           // M$ Excel (XLS) Plugin
  MSRTF:             // M$ RTF (Rich Text Format) Plugin [XML]
  NULL:              // Empty plugin
  MSWORD:            // M$ Word Plugin
  PDFDOC:            // OLD Adobe PDF Plugin
  TEXT:              // Plain Text Plugin
  ISOTEIA:           // ISOTEIA project (GILS Metadata) XML format locator records
```

How do we use it? Tools

The engine is C++ and can be bound using SWIG to a number of scripting languages.. but it also contains some powerfull convienient command line tools:

- Iindex // Indexing tool, index, append..
- Iutil // Utility program to set/modifiy parameters , import etc.
- Idelete // Move indexed files around, delete, shred (really remove)
- Iwatch // Simple tool to use directories as a index inbox queue
- Isearch // „Demo“ search but super powerful and useful!

Iindex

The most minimal run is something like:

Iindex -d MYDB file1 file2 file3

Here the indexer will create a MYDB index (should an index already exist it will be erased) and add file1 file2 file3 to it. Since the default document handler is AUTODETECT it tends to be able to guess a suitable handler.

Iindex has loads of options...

- -d db // Use database db.
- -setuid X // Run under user-id/name X (when allowed).
- -setgid X // Run under group-id/name X (when allowed).
- -cd X // Change working directory to X before indexing.
- -thes source // Compile search thesaurus from file source.
- -T Title // Set Title as database title.
- -R Rights // Set Rights as Copyright statement.
- -C Comment // Set Comment as comment statement.
- -mem NN // Advise min. of NN RAM.
- -memory NN // Force min. of NN RAM.
- // Note: Specifying more memory than available process RAM can
- // have a detrimental effect on performance. .

- --relative_paths // Use relative paths (relative to index path).
- -base path // Specify a base path for relative paths.
- -rel // Use relative paths and assume relation between index location
- // and files remains constant.
- -absolute_paths // Make file paths absolute (default).
- -ds NN // Set the sis block to NN (max 64).
- -mdt NN // Advise NN records for MDT.
- -common NN // Set common words threshold at NN.
- -sep sep // Use C-style sep as record separator.
- -s sep // Same as -sep but don't escape (literal).
- -xsep sep // Use C-style sep as record separator but ignore sep.
- -start NN // Start from pos NN in file (0 is start).
- -end nn // End at pos nn (negative to specify bytes from end-of-file).
- -override // Override Keys: Mark older key duplicates as deleted (default).
- -no-override // Don't override keys.
- -centroid // Create centroid.

- -t [name:]class[:] // Use document type class to process documents.
- -charset X // Use charset X (e.g. Latin-1, iso-8859-1, iso-8859-2,..).
- -lang ISO // Set the language (ISO names) for the records.
- // Specify help for a list of registered languages.
- -locale X // Use locale X (e.g. de, de_CH, pl_PL, ...)
- // These set both -charset and -lang above.
- // Specify help for a list of registered locales.
- -stop // Use stoplist during index (default is none)
- -l name // Use stoplist file name; - for "builtin".

Options just to select files..

- -f list // File containing list of filenames; - for stdin.
- -recursive // Recursively descend subdirectories.
- -follow // Follow links.
- -include pattern // Include files matching pattern.
- -exclude pattern // Exclude files matching pattern.
- -incldir pattern // Include dirs matching pattern.
- -excldir pattern // Exclude dirs matching pattern.
- -name pattern // Like -recursive -include.
- // pattern is processed using Unix "glob" conventions:
- // * Matches any sequence of zero or more characters.
- // ? Matches any single character, [chars] matches any single
- // character in chars, a-b means characters between a and b.
- // {a,b...} matches any of the strings a, b etc.
- // -include, -incldir, -excldir and -name may be specified multiple
- // times (including is OR'd and excluding is AND'd).

Indexing Performance Benchmarking on mainstream hardware:

Reference: Intel® Core™ i7-6920HQ (2.90 GHz) notebook using a low cost Samsung T5 USB 3.2 drive (500 MB/sec read/write) and indexing with 512MB Memory.

- Average around 5600k words/min (roughly ½ million emails in under 20 minutes).
- Indexing full text (without deep parsing) we see speeds 17-20x as fast. On the i7 notebook that results in better than 70 million words/min.
- Despite the process being I/O bound and designed for minimal system impact, we see on generic desktops and servers performance as fast as 99000k words/min . Most of the indexing time is spent analyzing document structure and parsing.

To search?

- There is an Isearch tool but generally we use custom code in C++ or in one of the scripting languages (such as Python).
- Isearch is still usefull „as is“. It is quite powerful and even can be used as a search daemon etc.
- As you might have guessed Isearch has loads and loads of options.. But I’ll spare you and move to more general topics on search…. Queries..

Query Languages / Boolean

What would a powerfull engine be without the possibility to do fine grained search?

- Support of a number of query methods
- A rich query language featuring an extensive collection of boolean and unary operators as well as all kinds of operators, relations and ....

Query Languages…

Generic Languages

- CQL (Contextual/Common Query Language) maintained by the Library of Congress

Its own language (IB-Language)

- Smart Queries
- Relevant Feedback (searching for „find me like this“)
- Infix notation
  - Infix notation is the common arithmetic and logical formula notation, in which operators are written infix-style between the operands they act on (e.g. 2 + 2).
- RPN notation
  - Reverse Polish notation (RPN), also known as postfix notation, was invented by Australian philosopher and computer scientist Charles Hamblin in the mid-1950s. It is derived from the Polish notation, which was introduced in 1920 by the Polish mathematician Jan Łukasiewicz.

Smart Queries

One of the overriding motivations for “smart queries” is to, for the non-technical user, counter the typical problem with full-text information search and retrieval systems: they either return too little or too many results.

Looking at the intersection of all terms (AND’d) can miss some important and highly relevant records, while looking at the union (OR’d) can sometimes overwhelm, depending all too heavily on some score normalization algorithm to relay relevance. Smart tries to guess intention.

Smart Query Logic

Smart queries try to interpret the query if its RPN or Infix or maybe just some terms. The logic for handling just terms is

as-if it first searched for a literal expression,

if not then tying to find the terms in a common leaf node—either anonymous or path specified—in a record's object model (we call this when anonymous PEER)

if not then AND'd (Intersection of sets)

if not then OR'd (Union) but reduced to the number of words in the query.

Smart Query Examples:

Searching in the collected works of Shakespeare (XML):

- (a) rich water

It finds that there are no phases like “rich water” but in the `The Life of Timon of Athens' it finds that both the words “rich” and “water” are in the same line: “.. And rich: here is a water, look ye..”. The query is confirmed as "rich" "water" PEER (see binary operators below)

Smart Query Examples:

(b) hate jews

- It finds that there are no phrases like “hate jews” but in the `The Merchant of Venice' we have in a specific scene lines that both talk about “hate” and “jews”. The smart query gets confirmed as "hate" "jews" || REDUCE:2 (see unary operators below)

  with the result “.. I hate him for he is a Christian ..” found in a PLAY\ACT\SCENE\SPEECH\LINE spoken by Shylock

Smart Query Examples:

<SPEECH>
<SPEAKER>SHYLOCK</SPEAKER>
<LINE><STAGEDIR>Aside</STAGEDIR> How like a fawning publican he looks!</LINE>
<LINE>I hate him for he is a Christian,</LINE>
<LINE>But more for that in low simplicity</LINE>
<LINE>He lends out money gratis and brings down</LINE>
<LINE>The rate of usance here with us in Venice.</LINE>
<LINE>If I can catch him once upon the hip,</LINE>
<LINE>I will feed fat the ancient grudge I bear him.</LINE>
<LINE>He hates our sacred nation, and he rails,</LINE>
<LINE>Even there where merchants most do congregate,</LINE>
<LINE>On me, my bargains and my well-won thrift,</LINE>
<LINE>Which he calls interest. Cursed be my tribe,</LINE>
<LINE>If I forgive him!</LINE>
</SPEECH>

Smart Query Examples:

(c) out out

- It finds a number of lines (5 plays) where “out out” is said such as “Out, out, Lucetta! that would be ill-favour'd” in `The Two Gentlemen of Verona'.

  The Tragedy of Macbeth: “The way to dusty death. Out, out, brief candle!“

  The Merry Wives of Windsor: “polecat, you runyon! out, out! I'll conjure you,“

  The First Part of Henry the Sixth: “Out, out! My lords, an please you, 'tis not so;“

  The Tragedy of Hamlet, Prince of Denmark: “Out, out, thou strumpet, Fortune! All you gods,“

  The confirmed query is: “out out”.

Who is on the stage?

We can even walk up the adressess previous to our hit looking at the values for <STAGEDIR> keeping to the current act and processing Enter, Exeunt,, etc. to even answer the question at search-time „Who is on the stage“?

<STAGEDIR>Enter DEMETRIUS and PHILO</STAGEDIR>

<STAGEDIR>Flourish. Enter ANTONY, CLEOPATRA, her Ladies, the Train, with Eunuchs fanning her</STAGEDIR>

<STAGEDIR>Enter an Attendant</STAGEDIR>

<STAGEDIR>Embracing</STAGEDIR>

<STAGEDIR>Exeunt MARK ANTONY and CLEOPATRA with their train</STAGEDIR>

Exeunt is used as a stage direction in a play to indicate that a group of actors leave the stage.

General Queries

Terms are constructed as

[[path][relation]]searchterm[:n]

And Terms are put together with relations to make expressions.

Infix Expression	Prefix Expression	RPN (Postfix) Expression
A + B	+ A B	A B +
A + B * C	+ A * B C	A B C * +
A + B * C + D	+ + A * B C D	A B C * + D +
(A + B) * (C + D)	* + A B + C D	A B + C D + *
A * B + C * D	+ * A B * C D	A B * C D * +
A + B + C + D	+ + + A B C D	A B + C + D +

Terms and Paths support Wildcards

Wild card	Description	Example	Matches
*	matches any number of any characters	`Law*`	`Law`, `Laws`, or `Lawyer`
?	matches any single character	`?at`	`Cat`, `cat`, `Bat` or `bat`
[abc]	matches one character given in the bracket	`[CB]at`	`Cat` or `Bat`
[a-z]	matches one character from the (locale-dependent) range given in the bracket	`Letter[0-9]`	`Letter0`, `Letter1`, `Letter2` up to `Letter9`
{xx,yy,zz}	match any of "xx", "yy", or "zz"	{C,B}at	Cat or Bat

Postfix term ops beyond just wildcards..

Append ~ for phonetic (soundex) search.

Append = for exact (case dependent) search.

Append > for exact right truncated search (=*).

Append . for "exact-term" (e.g. "auto" won't match "auto-mobile")

Boolean Operators?

Operator	
>	Greater than
<	Less than
<=	Less than or equal to
>=	Greater then or equal to

The above operators are overloaded and their semantics are specific to the object type of a field.

Operator	Semantics
< <=	Objects like date, numbers etc. have an order and <, resp. <=, applies as one might expect. For general "string" fields its interpreted as the equivalent to "*" (right truncation)
=	For general "string" fields its interpreted just as "/", viz. a member of the field.
> >=	As </<= above. For general "string" types the sematics are left truncation (as in *term)

Long Operator Name	Sym	Description
OR	||	Union, the set of all elements in either of two sets
AND	&&	Intersection, the set of all elements in both sets
ANDNOT	&!	Elements in the one set but NOT in the other
NOTAND	!&	As above but operand order reversed
NAND	&!	Complement of AND, elements in neither
XOR	^^	Exclusive Union, elements in either but not both
XNOR	^!	Exclusive not OR, complement set of XOR

More Boolean...

PROX:num		PROX:0 := ADJ. PROX:n := NEAR:n
NEAR[:num]		matching terms in the sets are within num bytes in the source
BEFORE[:num]		As above but in order before
AFTER[:num]		As above but in order after
DIST[>,>=,<,<=]num		Distance between words in source measured in bytes. (and order)
num > 1: integer for bytes. As fraction of 1: % of doc length (in bytes).		
NEIGHBOR	.~.	
PEER	.=.	Elements in the same (unnamed) final tree leaf node
PEERa		like PEER but after
PEERb		like PEER but ordered after
XPEER		Not in the same container
AND:field		Elements in the same node instance of field
BEFORE:field		like AND:field but before
AFTER:field		like AND:field but after
ADJ	##	Matching terms are adjacent to one another
FOLLOWS	#>	Within some ordered elements of one another
PRECEDES	#<	Within some ordered elements of one another
PROX		Proximity
FAR		Elements a "good distance" away from each other

Unary?

Unary Operators

Operator	Sym	Description
NOT	!	Set compliment
WITHIN[:field]		Records with elements within the specified field. RPN queries "term WITHIN:field" and "field/term" are equivalent. (for performance the query "field/term" is preferred to "term WITHIN:field")
WITHIN[:daterange]		Only records with record dates within the range
WITHKEY:pattern		Only records whose key match pattern
SIBLING		Only hits in the same container (see PEER)
INSIDE[:field]		Hits are limited to those in the specified field
XWITHIN[:field]		Absolutely NOT in the specified field
FILE:pattern		Records whose local file path match pattern
REDUCE[:nnn]		Reduce set to those records with nnn matching terms This is a special kind of unary operator that trims the result to metric cutoff regarding the number of different terms. Reduce MUST be specified with a positive metric and 0 (Zero) is a special case designating the max. number of different terms found in the set.
HITCOUNT:nnn		Trim set to contain only records with min. nnn hits.
HITCOUNT[>,>=,<,<=]num		As above. Example: HITCOUNT>10 means to include only those records with MORE than 10 hits.

More Unary...

TRIM:nnn		Truncate set to max. nnn elements
BOOST:nnn		Boost score by nnn (as weight)
SORTBY:<ByWhat>		Sort the set "ByWhat" (reserved case-insensitive names: "Key", "Hits", "Date", "Index", "Score", "AuxCount", "Newsrank", "Function", "Category", "ReverseHits", "ReverseDate", etc.)

SortBy:<ByWhat>

The SORTBY unary operator is used to specify a specific desired sort of the result set upon which it applies. It is used to sort (ByWhat keywords are case insensitive) by

- ➔ “Score” → Score (the default). The score, in turn, depends upon the selected normalization algorithm.
- ➔ “Key” → Alphanumeric sort of the record key.
- ➔ “Hits” → Number of hits (descending)
- ➔ “ReverseHits” → Number of hits (ascending)
- ➔ “Date” → Date (descending)

More relational algebra: JOIN, JOINL, JOINR

We can even join and combine search across multiple physical indexes.

- Since each „record“ get a key (either explicitly set or autogenerated from the file inode and host on which it resides) to uniquely identify it (it is also used for revision control).

- We can have multiple physical indexes to views of information with shared keys and then combine search in virtual Dbs across them and join.

Some Query Examples

NOTE: Infix is internally parsed into RPN as a stack..

Example RPN	**Example Infix**
title/cat title/dog || title/mouse ||	title/cat || title/dog || title/mouse
	title/(“cat” || “dog” || “mouse”)
speaker/hamlet line/love AND:scene	(speaker/hamlet and:scene line/love)
out spot PEER	out PEER spot
from/edz 'subject/"EU NGI0"' &&	from/edz && 'subject/"EU NGI0"'

Search Performance

Search time is most strongly tied to the size of each intermediate set built to process the query. Each query must allocate storage and has performance O(ln n + ln m) where n is the number of unique words in the index and m is the number of instances of the field searched.

The query time is then the sum of all these. Once done we have the time it takes to rank (should that have been requested) the results.

Sorting alone by score is O(ln n) where n is the number of items in the set.

==> The smaller the result sets encountered the faster the search...

(and we have a number of features like „fuel“...)

Example Reuters-21578

Indexing the famous Reuters (toy) dataset.

- **Indexing** time: < 10 seconds. 23167k words/min
- **Search**:
  - indian sugar → „indian sugar“. 1 result. 3 ms.
  - Computer venture → „computer“ „venture“ PEER. 2 results. 12 ms.
  - “india” ! → 21457 results. 26 ms (25 ms search).
  - india → 121 records. 3 ms (1 ms search)
  - places/india → 82 records. 3 ms (2 ms search)

Example: Indexing Mailinglist

Several years of the BSD mailing list...

Total Number of Words: 88851349 (1390171 unique to 28 characters)

Total number of messages: 265070

- **Search**:

  nvidia → 1 result. 4 ms (2 ms search)

  ibm → 3001 results. 8 ms (6 ms search)

  Computer memory → „computer memory“. 2 results. 72 ms (70 ms)

  Bandwidth fiber → „bandwidth“ „fiber“ PEER. 3 results. 21 ms (19 ms)

  FreeBSD bugs → „freeBSD Bugs“. 20669 results. 392 ms (391 ms)

  FreeBSD → 208948 results. 391 ms (389 ms)

  FreeBSD oracle → „freeBSD oracle“ 12 results. 43 ms (42 ms)

Semantic Search

Like any good engine its got also personalized (global, group or individual at will) “Thesauri” .

- These get expanded at search time.. and are also weighted..
- Since they are really only meaningfull by domain they are generally either created using a bag-of-words vectorization model applied to a subject specific corpus or by hand.

Personalized Thesauri Example.

```
# Sample synonyms
war=krieg:2+combat+battle
peace=pax
```

The Infix query: `("War" and "Peace")`

is expanded as-if `(("war" or "krieg":2 or "combat" or "battle")` and `("peace" or "pax"))` was entered into the system.

The RPN query: `"War":2 "Peace" &&`

(really the same as the above Infix query save the use of the weight "2" on the "war" term) is expanded as-if "war" `"krieg":4 || "combat" || "battle" || "peace" "pax" || &&` was entered into the system. Notice that the weights are multiplicative.

Objects: Beyond Text

We support many object types. Example geospatial:

Geospatial Proximity search

- Fieldname{{Latitude, Longitude}[operator]value were operator: is <, <=, =, >, >= value=distance is km (default) with the modifiers N, K, M for, resp., nautical miles, km and statue miles.
- Geospatial Bounding box search

Search for a box is defined by the RECT operator: RECT{N,W,S,E}.

Objects: What else

Data-type name	Description
string	String (full text)
numerical	Numerical IEEE floating
computed	Computed Numerical
range	Range of Numbers
date	Date/Time in any of a large number of well defined formats including common ISO, W3 and IETF formats.
date-range	Range of Date as Start/End but also +N Seconds (to Years)
box	Geospatial bounding box coordinates (N,W,S,E): RECT{c0 c1 c2 c3}
gpoly	Geospatial n-ary coordinates as list of vertices.
time	Numeric computed value for seconds since 1970, used as date.
ttl	Numeric computed value for time-to-live in seconds.
expires	Numeric computed ttl value as date of expiration
boolean	Boolean type. 1, 0, True, False, Yes, No, Ja, Nein, Ein, Aus, On, Off, Oui, No, Tack ...
currency	Monetary currency
dotnumber	Dot number (Internet v4/v6 Addresses, UIDs etc)
phonetic	Computed phonetic hash applied to each word (for names)
phone2	Phonetic hash applied to the whole field
metaphone	Metaphone hash applied to each word (for names)
metaphone2	Metaphone hash (whole field)

Objects: What else

hash	Computed 64-bit hash of field contents
casehash	Computed case-independent hash of text field contents
lexi	Computed case-independent lexical hash (first 8 characters)
privhash	Undefined Private Hash (callback)
isbn	ISBN: International Standard Book Number
telnumber	ISO/CCITT/UIT Telephone Number
creditcard	Credit Card Number
iban	IBAN: International Bank Account Number
bic	BIC : International Identifier Code (SWIFT)
db_string	External DB String (callback)
callback	Local callback 0 (External)
Local1 – local7	Local callback 1 - 7 (External)
special	Special text (reserved)

xs:string	Alias of string
xs:normalizedString	Alias of string
xs:boolean //	Alias of boolean
xs:decimal	Alias of numerical
xs:integer	Alias of numerical
xs:long	Alias of numerical
xs:int	Alias of numerical
xs:short	Alias of numerical
xs:unsignedLong	Alias of numerical
xs:unsignedInt	Alias of numerical
xs:unsignedShort	Alias of numerical
xs:positiveInteger	Alias of numerical
xs:nonNegativeInteger	Alias of numerical
xs:negativeInteger	Alias of numerical
xs:positiveInteger	Alias of numerical
xs:dateTime	Alias of date
xs:time	Alias of time

Objects: Relations

Recall we have all those typical comparison relations: >, <, >=, <=, ..

I mentioned that their meaning depends upon the view chosen. Looking at something as a string its like *. Other types have their own comparison metrics such as Cartesian.

Objects....

So the design contains a large number of objects: numerical, range, geospatial etc.

These objects don't even have to be part of any document but may be available via interface glue into other systems via ODBC, CORBA or object embedding. This allows indexing content to be stored in and searched from other systems. This is useful in many dynamic applications in commerce and trading...

OK.. I'm impressed.. But unique?

What other uses for objects ?

We support something called “**Dynamic Presentation**”.

- When things get indexed (ingested) the structure gets stored with pointers to content. So we can not just search but also reconstruct formats on-the-fly at time of presentation without re-parsing. An e-mail, for example, can be viewed as not just e-mail but als XML, as HTML as .. but there is more..
- Content can come (pointers) from those other object services, remote databases etc. merging into a presentation. It can also (rights based) provide different views (hiding or redacting content).

Scoring and Ranking..

And, of course, we have a large selection of scoring and ranking algorithms.

- They can even be „pre-calculated“ (defined and specified in those virtual definitions I spoke of) to allow for ranking criteria that, for example, are computationally expensive.
- And at query time we can even do all kinds of magic to boost some things.. bring similar things together (I call it „magnetism“)... and and and..

Normalization

• Cosine normalization (TD-IDF): Despite being many decades old, still seems to be among the best. It is a bag-of-words retrieval function that ranks a set of documents based on the query terms appearing in each document, regardless of their proximity within the document. Its main drawback (beyond the need to create full sets) is that it tends to be biased towards shorter records, finding them more frequently than alone the linear distribution of lengths might suggest. This has led to the development of the Okapi system with a number of variants such as BM1, BM11, BM15, BM25 and others. At their heart (beyond using some empircal values for weighing parts of their metric) a comparison of document length with the average document length.

• Log Normalization: A cosine variant normalized (dampened) according to the log of document length.

• Max Normalization: Normalized to favor those with more different hits.

• Bytes Normalization: Various document length normalization models have been proposed to address the bias of Cosine Normalization towards shorter records. They, however, nearly always tend to overly penalize long records—including BM25. With Byte Normalization a middle ground approach is taken: the cosine model is slightly modified to also take document length distribution into consideration.

• Euclidean Normalization: Yet another variant of Cosine Normalization. The byte metric distance between "hits" (multiple term searches) is used to favor records where these are closer. Since hits are typically closer in shorter records given their lesser maximum distance, we limit ourselves to the minimum of all the distances rather than an average and adjust.

• One may extend the engine with custom normalizations.

Some virtual db Ranking tweaks...

[Ranking]

PriorityFactor=fff.ff # Priority factor

IndexBoostFactor=fff.ff # Boost score by index position

FreshnessBoostFactor=fff.ff # Boost score by freshness

FreshnessBaseDateLine=date # Date/time, Records newer than this date, get FreshnessBoostFactor added, older get substracted. The unit of resolution is defined by the precision of the specified date. Default is the date specified in DateLastModified [DbInfo] (Minutes resolution)

LongevityBoostFactor=fff.fff # Boost score by difference in days between the date created and date modified of the records

Any bullet points?

- Low-code ETL / "Any-to-Any" architecture (Dynamic Presentation)
- Handles a wide range of document formats including “live” data.
- Powerful Search (Structure, Objects, Spatial) / Relevancy Engine
- NoSQL Datastore
- Set based with an exhaustive collection of (binary and unary) set operations.
- Dynamic search-time virtual collections of indexes (Shards)..
- Useful for Analytics, Recommendation / Autosuggestion and ...
- Embeddable in products (comparatively low resource demands)
- Customization.
- Support Peer-to-Peer and Federated architectures (e.g. OASIS SRU/W, ISO 23950/Z.39.50).
- Freely available under a permissive software license (Apache 2.0).

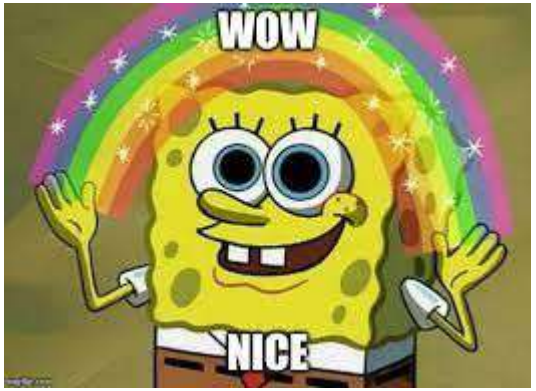
That is really a lot.. Any more features we may have of dreamt of?

Of course. We’ve barely touched on them (Can go on for hours)....

Make that weeks..

Visit: http://www.nonmonotonic.net/re-isearch
To learn more.

Software, documentation and handbooks are freely available on Github:



Thank You.