Inside the IMS Corpus Workbench

http://cwb.sf.net/

Stefan Evert

Institute of Cognitive Science
University of Osnabrück

stefan.evert@uos.de | purl.org/stefan.evert

CWB The IMS Open Corpus Workbench

Talk overview

- History of the IMS Corpus Workbench (CWB)
- The CWB data model
- Recently added CQP features
- The CWB architecture (CL, CQP, CWB/Perl, CQi)
- Corpus indexing in the CWB
- Inside CQP
- Critical evaluation & Plans for future development

History

The need for a corpus workbench

Flashback to the early 1990s

- Rising interest in corpus-based and statistical approaches
  - part-of-speech tagging with HMM (Church 1988)
  - computational lexicography & collocation extraction (Sinclair et al. 1970/2004; Church & Hanks 1990)
  - statistical machine translation (Brown et al. 1990, 1993)
  - special issue on Using Large Corpora (J of Comp Ling, 1993)
- Main resource: **large text corpora** with shallow annotation
  - collect > 100 M words (e.g. British National Corpus)
  - usually with part-of-speech tagging & lemmatisation
  - textual structure: sentences, paragraphs, documents + metadata

The need for a corpus workbench

Flashback to the early 1990s

- Standard format: ASCII/Latin-1 text with inline annotation
  - one sentence per line, with inline POS tags
  - one word per line, with annotations in TAB-separated fields
  - well-suited for statistical exploitation, as training data, etc.
- Interactive use: linguists, lexicographers, terminologists, …
  - need for more interactive corpus search & processing
  - concordance & collocation analysis for specified word
  - frequency lists for keyword/terminology identification
  - search for complex linguistic patterns (based on POS tags)
- Requires special **database engine for text corpora**
  - efficient indexing of large text corpora with linguistic annotation
  - allow non-technical users to write complex search patterns

History of the IMS Corpus Workbench

1993–2008 — the official timeline

- 1993–1996: Project on Text Corpora & Exploration Tools
  - IMS Stuttgart, financed by the state of Baden-Württemberg
  - **TreeTagger** by Helmut Schmid (Schmid 1994, 1995)
  - **Corpus Workbench** by Oliver Christ (Christ 1994)
- 1994–1998: EAGLES & DECIDE projects
  - additional funding for TreeTagger and STTS tagset (EAGLES)
  - application in computational lexicography (DECIDE)
  - Xkwic & macro processor MP (Christ & Schulze 1996)
- 1996: First stable public release of CWB (v2.2)
  - non-commercial use only, binary packages for SUN Solaris
  - experimental (i.e. buggy) Linux version

History of the IMS Corpus Workbench

1993–2008 — the official timeline

- 1998–2004: In-house development continues
  - sporadic funding from various projects & other sources
  - beta versions of CWB 2.3/3.0 available since 2001
  - binary packages for SUN Solaris (SPARC) & Linux (i386)
- 2000: First "clones" of the CWB appear
  - Manatee (ca. 2000, open-source in late 2005)
  - Poliqarp (ca. 2007)
- 2005: CWB becomes open-source software
  - new "official" name: IMS Open Corpus Workbench
- 2008: Official release of OCWB version 3.0 (hopefully!)

The true history of the CWB

- 1993–1995: Development of indexing library & CQP
- 1996: Poor design choices (Xkwic & MP)
- 1997: Everybody leaves
- 1998: New maintainers (Stefan Evert, Arne Fitschen)
- 1998–2000: Bug fixes & support for in-house users
  - first "bug fix": discontinue Xkwic & MP development
  - improved Linux support (later main development platform)
- 2000–2003: Incremental addition of new features
  - driven by requirements of IMS users and some other groups
  - frequently updated beta releases (2.2.b17–2.2.b98)
- 2005–2008: Preparation of open-source release (OCWB 3.0)

Data Model

A typical text corpus

A fine example. Very fine examples!

<text id="42" lang="English">
<s>
A/DET/a *fine*/ADJ/fine *example*/NN/example *.*/PUN/.
</s>
<s>
Very/ADV/very *fine*/ADJ/fine *examples*/NN/example *!*/PUN/!
</s>
</text>

Representation in tabular format

as used by relational databases, tables of statistical observations, …

#	word	pos	lemma
0	A	DET	a
1	fine	ADJ	fine
2	example	NN	example
3	.	PUN	.
4	Very	ADV	very
5	fine	ADJ	fine
6	examples	NN	example
7	!	PUN	!

corpus position ("cpos")

Representation in tabular format

Special representation of XML tags

#	word	pos	lemma
(0)	<text id="42" lang="English">		
(0)	<s>		
0	A	DET	a
1	fine	ADJ	fine
2	example	NN	example
3	.	PUN	.
(3)	</s>		
(4)	<s>		
4	Very	ADV	very
5	fine	ADJ	fine
6	examples	NN	example
7	!	PUN	!
(7)	</s>		
(7)	</text>		

XML tags inserted as "invisible" tokens

Representation in tabular format

Special representation of XML tags

#	word	pos	lemma
(0)	<text id="42" lang="English">		
(0)	<s>		
0	A	DET	a
1	fine	ADJ	fine
2	example	NN	example
3	.	PUN	.
(3)	</s>		
(4)	<s>		
4	Very	ADV	very
5	fine	ADJ	fine
6	examples	NN	example
7	!	PUN	!
(7)	</s>		
(7)	</text>		

p-attributes

s-attributes

id="42" lang="English"

XML regions represented internally as ranges of tokens, i.e. start/end #

Representation in tabular format

Lexicon of annotation strings for each table column (p-attribute)

#	word		pos		lemma	
(0)	<text id="42" lang="English">					
(0)	<s>					
0	A	0	DET	0	a	0
1	fine	1	ADJ	1	fine	1
2	example	2	NN	2	example	2
3	.	3	PUN	3	.	3
(3)	</s>					
(4)	<s>					
4	Very	4	ADV	4	very	4
5	fine	1	ADJ	1	fine	1
6	examples	5	NN	2	example	2
7	!	6	PUN	3	!	5
(7)	</s>					
(7)	</text>					

lexicon IDs for annotation strings (per column)

New CQP Features

The matching strategy of CQP queries

Pattern: `DET? ADJ* NN (PREP DET? ADJ* NN)*`

the old book on the table in the room

The matching strategy of CQP queries

Pattern: `DET? ADJ* NN (PREP DET? ADJ* NN)*`

the old book on the table in the room

the old book
old book
book
the table
table
the room
room

This is useful for the extraction of cooccurrence data, e.g.

`[pos="ADJ"] []{0,5} [pos="NN"]`

"traditional" strategy

```
> set MatchingStrategy "traditional";
```


The matching strategy of CQP queries

Pattern: `DET? ADJ* NN (PREP DET? ADJ* NN)*`

the old book on the table in the room

the old book
the table
the room

new standard strategy

The matching strategy of CQP queries

Pattern: `DET? ADJ* NN (PREP DET? ADJ* NN)*`

the old book on the table in the room

book
table
room

shortest match strategy

```
> set MatchingStrategy "shortest";
```


The matching strategy of CQP queries

Pattern: `DET? ADJ* NN (PREP DET? ADJ* NN)*`

the old book on the table in the room

the old book on the table in the room

longest match strategy

```
> set MatchingStrategy "longest";
```

Label references & anchors

- CQP v2.2 supported labels, but only for simple queries
  - more complex expressions would lead to random errors
- Reimplementation of label handling
  - almost always works correctly now ;-)
  - speed penalty ca. 10% for typical queries
- The "classic" example of label references:
  - ```
    n1:[pos="NN"] "by" n2:[pos="NN"]
          :: n1.word = n2.word;
    ```
- But there are many other applications, e.g.
  - ```
    pron:[pos="PP"] []{0,5} verb:[pos = "VB.*"]
    :: verb.pos = "VBZ" -> pron.lemma = "he|she|it";
    ```

logical implication operator (new)

Label references & anchors

- Anchors are implicitly-defined labels:
  - match — first token of current match
  - matchend — last token of current match
  - target — define with @ marker or set target command
  - keyword — only with set keyword command
- Anchors are available within a CQP query …
  - ```
    [pos="DT"]? [pos="RB|JJ.*"]* [pos="NN"]
         :: distabs(match, matchend) >= 5
    ```
  - find "long" NPs consisting of 6 or more tokens
- … and they are stored with a named query result
  - ```
    group MyQuery target lemma;
    ```


Label references & anchors

- A little-known feature: matches can be modified by changing labels with the set keyword command
  - ```
    Time = [lemma = "time" & pos = "NN.*"];
    ```
  - ```
    set Time keyword nearest [pos = "JJ.*"]
                     within left 3 words;
    ```
  - ```
    set Time match keyword;
    ```
    - if keyword anchor is not defined, match remains unchanged
  - ```
    set Time keyword NULL;
    ```
    - delete keyword anchor when no longer needed
  - NB: we could also have used set match directly

Using XML annotation

- XML tags are stored with (optional) attribute-value pairs:
  - <s> … </s>
  - <text>[id="42" lang="English"] … </text>
  - can be matched by including tag in CQP query, e.g. <text>
- Newer versions of CWB improve support for XML tags
  - attribute-value pairs can be split automatically and stored in new s-attributes, e.g. text_id and text_lang
  - tags in CQP query allow regular expression constraint: <text_lang = "en.*"%c> … ;
  - matching start and end tag correspond to single region: <s> … </s>; → matches exactly one sentence
  - also automatic renaming of nested XML regions, but currently no access to tree structure in CQP queries

Using XML annotation

- Check whether token is contained in specific XML region
  - `<s> [word = "[a-z].+" & !(caption | item)]`
  - searches for lowercase word at start of a sentence, but not in figure captions (`<caption>`) or list items (`<item>`)
- Access attributes of XML region with label references
  - e.g. textual metadata → start tag cannot be included in query
  - `group MyQuery match text_domain;`
  - `… :: match.text_domain = "economy";`
- Special "this" label _ points to current token:
  - `[word = "decency" & _.text_domain = "economy"]`
  - `… [ … & distabs(_, match) < 3 ] … ;`
    → must be within first 4 tokens

Feature sets

- Sometimes it is useful to annotate multiple values
  - disambiguation problems: `NNS` or `VBZ` ?
  - annotation of WordNet synonyms: *appear*, *look*, *seem*
  - morphosyntactic features in German (→ syncretism):
    der = `Nom:M:Sg:Def` | `Gen:F:Sg:Def` | `Dat:F:Sg:Def` | `Gen:M:Pl:Def` | `Gen:F:Pl:Def` | `Gen:N:Pl:Def`
- CWB solution: **feature sets** encoded as special strings
  - `|NNS|VBZ|` → alternative values separated & enclosed by |
  - `|appear|look|seem|`
  - `|Dat:F:Sg:Def|Gen:F:Pl:Def|Gen:F:Sg:Def|Gen:M:Pl:Def|Gen:N:Pl:Def|Nom:M:Sg:Def|`
    → notice alphabetical ordering of items
  - `|` → empty feature set

Feature sets

- Feature sets can be queried with cleverly designed regexps
  - e.g. `[pos = ".*\|NNS\|.*"]` → may be a plural noun
  - made easier by special notation, but still very cumbersome
- CQP provides special operators for convenience
  - `[pos contains "NNS"]` → expands to regexp above
  - `[agr matches "Gen:.*"]` → unambiguous genitive
  - `[ambiguity(syn) = 0]` → no synonyms found in WordNet

Feature sets

- Combining feature sets & labels: agreement in German NPs
  - NP agreement between determiner, adjectives and noun
  - i.e., valid feature combinations must be compatible with other words in NP → unification
  - corresponds to intersection of feature sets
- Testing NP agreement in CQP queries:
  - `d:[pos="ART"]? a:[pos="ADJA"]? n:[pos="NN"] …`
  - `… :: amibguity(/unify[agr, d, a, n]) > 0;`
    → check agreement in potential NP
  - `… :: /unify[agr, d, a, n] matches ".*:Sg:.*";`
    → unambiguously identified as singular NP
  - NB: undefined labels are automatically ignored

The CQP macro language

- CQP macros are a simply, but flexible templating system
  - partial replacement for discontinued Macro Processor
  - built directly into CQP → also available in interactive mode
  - works by substitution of unparsed strings → can be used (almost) everywhere: within constraint, multiple commands, …
  - nested macro calls → non-recursive phrase structure grammar
- Define macros in separate file or with interactive commands
- Macro invocation syntax: `/np["coffee"]`
  - `/unify[`*`attribute`*`, `*`label`*`, …]`
    is a built-in macro with a variable number of arguments
  - also try `/codist["that", pos];`
    → mini-script with multiple commands

Macro definition example

If you fully understand this code, you can consider yourself a CQP expert!

```
MACRO adjp()
   [pos = "RB.*"]?
   [pos = "JJ.*"]
;

MACRO np($0=Noun $1=N_Adj)
   [pos = "DT"]
   ( /adjp[] ){$1}
   [(pos = "NNS?")
     & (lemma = "$0")]
;

MACRO pp($0=Prep $1=N_Adj)
   [(word = "$0")
     & (pos = "IN|TO")]
   /np["$1"]
;
```

```
MACRO np($0=N_Adj)
   [pos = "DT"]
   ( /adjp[] ){$0}
   [pos = "NNS?"]
;

MACRO np()
   [pos = "DT"]
   ( /adjp[] )*
   [pos = "NNS?"]
;

MACRO pp($0=N_adj)
   /pp[".*", "$0"]
;
MACRO pp()
   /pp["0,"]
;
```


Secret feature: zero-width assertions

- Look-ahead patterns `[:…:]` perform test on next token without including it in the query match
  - simulate longest match strategy in standard mode:
    `[pos = "NNS?"]{2,} [: pos != "NNS?" :];`
  - `[pos = "VB.*"] "that" [: pos != "JJ.*|N.*" :];`
    → demonstrative or clausal verb complement
- Look-ahead patterns are called **zero-width assertions** because they do not "consume" a token
  - convenient for complex constraints on XML tags:
    ```
    <text> [: _.text_domain = "law"
              & !(_.text_lang = "English") :] …
    ```
  - add label or target marker before/after group:
    `… a:[::] ( … | … | … ) b:[::] … ;`

Zero-width assertions & label scope

- Zero-width assertions have been used to implement macros for German NPs with agreement:
  - ```
    DEFINE MACRO np_agr(0)
      a:[pos="ART"]?
      b:[pos="ADJA"]*
      c:[pos="NN"]
      [: ambiguity(/unify[agr, a,b,c]) > 0 :]
      [: /undef[a,b,c] :]
    ;
    ```
  - built-in macro `/undef[]` deletes label references
    → scope of labels limited to macro body
  - allows query `/np_agr[] [pos="V.*"]+ /np_agr[]`
    to work correctly (without label interference)
    ```

Embedding CQP

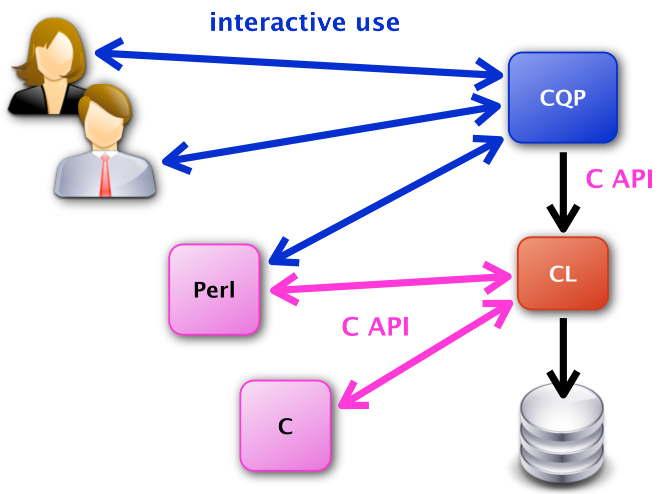
- Various new functions improve data exchange with external programs
  - dump → table of query matches (cpos in text format)
  - undump → load table of query matches into CQP (can be sorted in arbitrary order)
  - tabulate → list arbitrary information for each query match
    - e.g. corpus position, matching string, POS, metadata, …
    - suitable as input for statistical software (→ frequency analysis etc.)
- Embedding CQP as a background process ("slave")
  - set PrettyPrint off; produces machine-readable output
  - child mode (cqp -c) for more robust communication
  - these features are used heavily by the CWB/Perl interface

Further topics

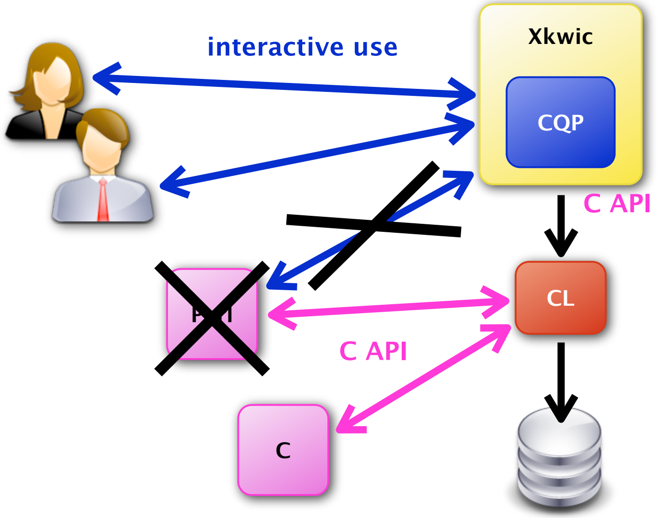
- Built-in sort command has been re-implemented
  - well-defined syntax & robust operation
  - additional features, e.g. reverse sorting
- Frequency lists with new count command
  - based on sort → frequency list for sort keys
  - alternative to group command
    - count strings of arbitrary length
    - case/diacritic-folding, count reverse strings
    - easy access to corpus examples for each item in frequency list
    - counts only continuous strings, sometimes slower than group
- Read the latest version of the **CQP tutorial**: http://cwb.sourceforge.net/documentation.php

Architecture

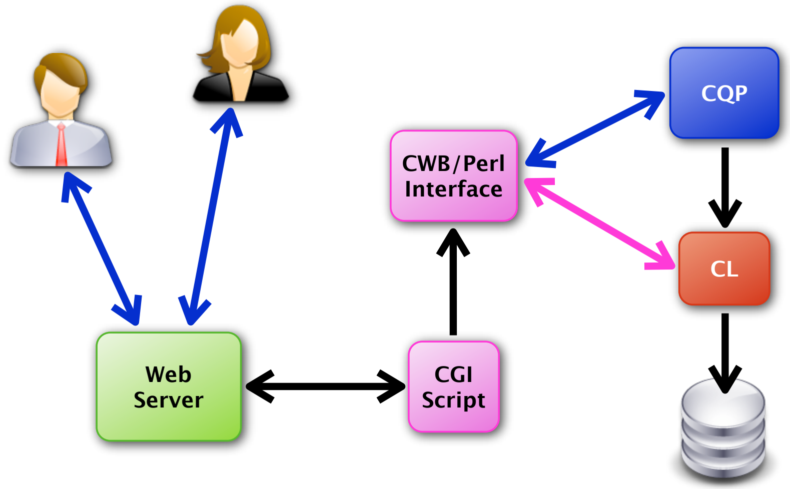
Architecture of the CWB



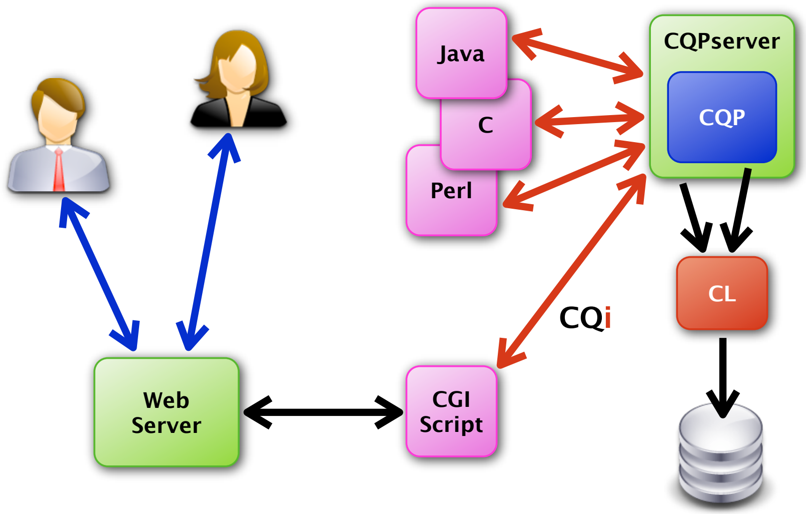
Xkwic: a monolithic dead end



Web-based interfaces to the CWB

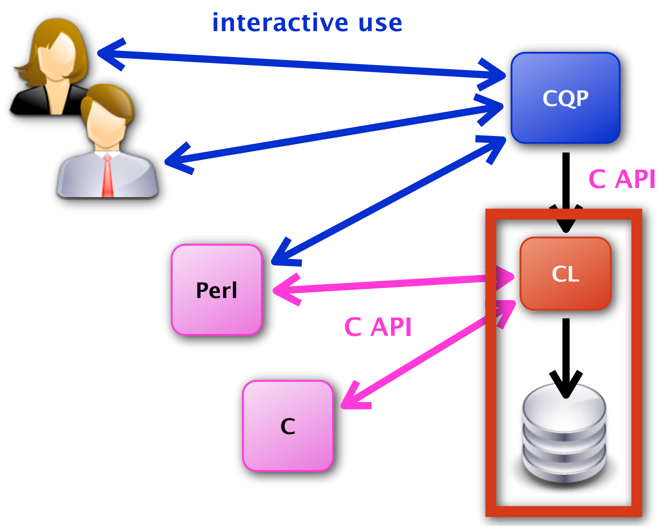


CQi — a network protocol for the CWB



Indexing

CWB architecture: corpus indexing (CL)



CWB architecture: corpus indexing (CL)

- Traditional wisdom on managing large data sets:
  - divide into fixed-size records (table rows) for compact storage
  - use indexing (based on sort operations) for fast access
  - CWB: data record = token + annotations
    - no support for character-level matching & alternative tokenisations
- Why not use an existing relational database engine?
  - table rows in relational database are independent & unordered
    - fast access to token sequence is essential for CQP queries
  - large text corpora are mostly static → optimisations possible
    - more compact representation of data & index
    - record can be identified by its corpus position (integer constant)
    - no overhead from table locking, transactions and journaling
  - lack of sufficiently powerful non-commercial RDBMS in 1993

Design choices for the CL library

- Static corpus → compact storage & optimal compression
  - not possible to add/delete documents (≠ search engines)
- Table-like data model (record = token + annotations)
  - column-major representation (≠ relational database)
    → all p-attributes and s-attributes are stored independently
- All annotations are ASCII/Latin-1 strings
  - Unicode immature in 1993, Latin-1 is compact & efficient
- No support for structured annotation / XML necessary
  - sentences, paragraphs, etc. = flat sequence of regions
- **The whole story:** Witten, Ian H.; Moffat, Alistair; Bell, Timothy C. (1999). Managing Gigabytes. Morgan Kaufmann Publishing, San Francisco, 2nd edition.

CWB index structures (p-attribute)

#	word		pos		lemma	
(0)	<text id="42" lang="English">					
(0)	<s>					
0	A	0	DET	0	a	0
1	fine	1	ADJ	1	fine	1
2	example	2	NN	2	example	2
3	.	3	PUN	3	.	3
(3)	</s>					
(4)	<s>					
4	Very	4	ADV	4	very	4
5	fine	1	ADJ	1	fine	1
6	examples	5	NN	2	example	2
7	!	6	PUN	3	!	5
(7)	</s>					
(7)	</text>					

CWB index structures (p-attribute)

cpos	tokens
0	0
1	1
2	2
3	3
4	4
5	1
6	2
7	3
8	…
…	…

id	lexicon
0	DET
1	ADJ
2	NN
3	PUN
4	ADV
…	…

id	freq
0	1
1	2
2	2
3	2
4	1
…	…

id	occurrences (cpos)
0	0, …
1	1, 5, …
2	2, 6, …
3	3, 7, …
4	4, …
…	…

CWB index structures (s-attribute)

<s>	start	end
0	0	3
1	4	7
2	…	…
…	…	…

<text>	start	end
0	0	7
1	…	…
2	…	…
…	…	…

annotation
id="42" lang="English"
…
…
…

Huffman coding for p-attributes

- Huffman code = optimal compression for independent coding of individual tokens with static codebook
  - similar to Morse code: use short bit patterns for frequent items
- Sample Huffman codes for part-of-speech tags

NN	110	JJ	0100
IN	111	JJR	000000100
DT	101	JJS	0000000001
PP	1001		
VB	00111		

- Encoding of POS tags for *go to the prettiest beach*:

0 0 1 1 1 1 1 1 1 0 1 0 0 0 0 0 0 0 0 0 1 1 1 0

VB IN DT JJS NN

Golomb coding for index files

- Encode distances between occurrences of lexicon entry
  - assumption: occurrences randomly distributed across corpus
- Golomb codes = mixed unary/binary representation
  - fixed size of binary part ≈ average distance value
  - optimal for random distribution, good worst-case bounds
- Golomb code example:
  - distance to next occurrence = 26 tokens
  - e.g. 3-bit binary representation: 26 = 3 * 8 + 2
  - Golomb code: 000 1 010

unary part: 3 x 8 = 24

stop bit

binary part: 24 + 2 = 26

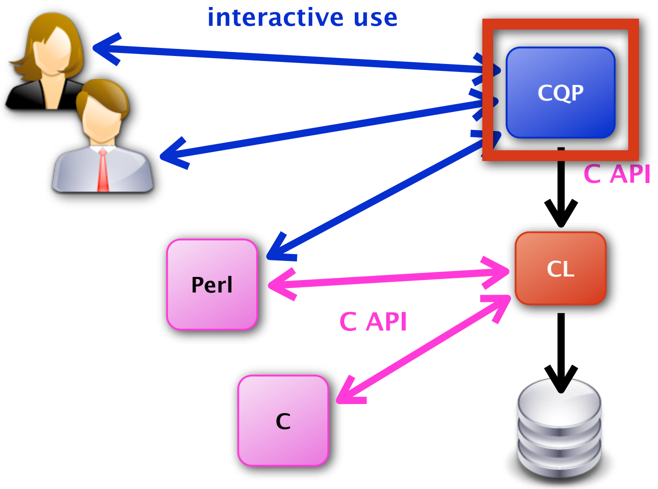
Data compression rates of the CWB

Typical data sizes of p-attributes for 100 M word corpus

Plain text:	ca. **400–600 MB**
Uncompressed attribute: (including all index & lexicon files)	ca. **800 MB**
Word form, lemma, etc.:	ca. **320–360 MB**
POS, morphological features:	ca. **100–150 MB**
Binary attribute:	ca. **50 MB**

CQP

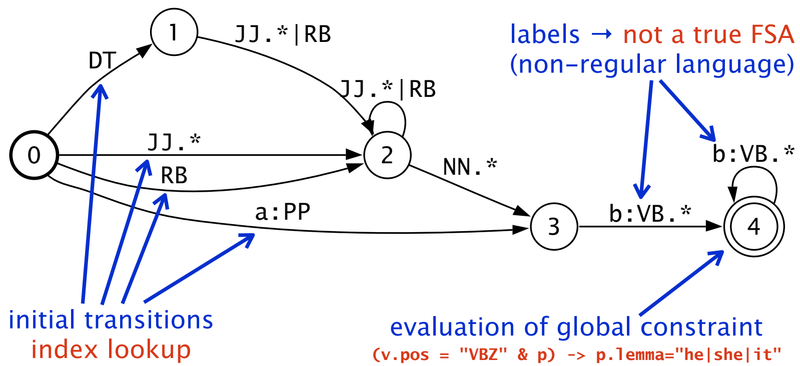
CWB architecture: corpus indexing



Evaluation of CQP query as FSA

This query is intended for illustration purposes only :-)

```
( [pos="DT"]? ([pos="JJ.*"] | [pos="RB"])+ [pos="NN.*"]
  | p:[pos="PP.*"] )
v:[pos="VB.*"]+
:: (v.pos = "VBZ" & p) -> p.lemma="he|she|it";
```



Consequences of FSA algorithm

- Multi-pass query evaluation
  - index lookup for each possible initial transition (= pass) → may need to store large vector of cpos in memory
  - then simulate FSA from these corpus positions → slow & computationally expensive
- Query execution speed depends on query-initial patterns
  - i.e. patterns that correspond to initial transitions of the FSA
  - fast queries for infrequent lexical items: `[lemma="sepal"]`
  - slow queries for general patterns: `[pos="NN"]`

Consequences of FSA algorithm

- Key to query optimisation: avoid FSA simulation
  - reduce number of start positions for FSA simulation as far as possible by (combined) index lookup
  - cannot rely solely on query-initial patterns: `[pos = "DET"]? [pos="ADJ"]* [lemma="song"];`
- Automatic query optimisation difficult in FSA representation
  - needs advanced graph manipulation algorithms in C
  - must also avoid expensive lexicon search with complex regexp
- Standard FSA techniques not applicable because of labels
  - in particular, it is difficult to change FSA evaluation order (so as to start from the least frequent pattern)

Future

Good things about the CWB

(in the author's opinion)

- Static corpus & token-based data model
  - straightforward implementation (KISS!)
  - allows compact storage & efficient access
- Annotations as strings
  - numbers rarely needed, structured data too complex
- Regular query language (in formal sense)
  - good balance between expressiveness and efficiency
  - but not suitable for querying hierarchical structure
  - recursion (CFG) needed for linguistic queries (even on POS tags)
  - FSA implementation hinders query optimisation

Urgently needed extensions

(reasonable decisions in 1993, but the times have changed)

- Full support for Unicode data (UTF-8)
  - essential for multilingual corpora, software libraries available
  - "legacy" encodings such as Latin-1 are no longer needed
- Handling of very large corpora (> 1 billion words)
  - 32-bit version limited to 200-500 million tokens
  - 64-bit version: up to 2 billion tokens, but queries are slow
  - design limit is 2.1 billion tokens (signed 32-bit integers), but the ukWaC corpus is already larger
- Support for hierarchical structures / XML trees
- APIs for high-level programming languages
  - CL API available for C and Perl, but undocumented
  - also need API for CQP queries, kwic output, etc.

Big mistakes of the CWB

(*&$*%#!!!!)

- Overzealous data compression
  - dogma of search engine optimisation, but decompression is inefficient (see e.g. Anh & Moffat 2005)
- Poor/non-existent software engineering
  - insufficient abstraction layers, memory management, etc.
- Almost everything about the CQP architecture
  - FSA implementation of regular query language
  - labels make query optimisation all but impossible (mea culpa!)
  - monolithic design, many internal functions too specialised
- Feeping Creaturism
  - incremental addition of work-arounds & clever tricks, rather than addressing basic design limitations

Strategies for future development

The best strategy depends on user requirements, available developers, …

- Re-implementation from scratch
  - low-level CL layer → corpus query library → CQP
  - alpha version after 1 year, stable beta after 2 years (optimistic)
- Keep adding features & fixing problems
  - until we're ready to release CWB Vista …
  - but this approach might have best payoff for majority of users
- Attempt refactoring of CWB source code
  - implement urgently needed features one by one
  - keep as much of existing codebase as possible, but make sure new code is well-designed (as basis for further refactoring)
  - new code immediately usable, but overall effort is larger
- Re-think corpus indexing & query processing

Thank you!