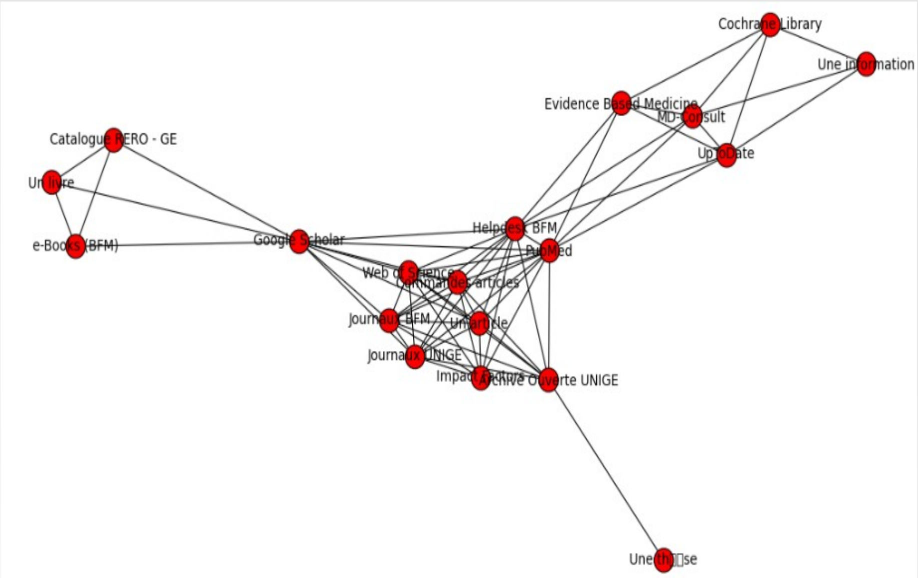
MarcXimiL



Similarity analysis

- **Near-duplicates detection**
- Information monitoring
- Collection structure analysis
- Plagiarism detection
- Related records suggestion

http://marcximil.sourceforge.net

MarcXimiL philosophy

- **Flexible at all levels**
- Information Retrieval algorithms (adapted) and bibliographic specific algorithms
- Open source : GPL 3
- Central configuration (similarity strategy)
- Standard (MARCXML, OAI-PMH2, UNIX structure)
- Compatible (all systems, python 2.4.0 → 3.1.1)
- Easy to install (unpack)

MarcXimiL flexibility philosophy

- **Method: each level of the similarity analysis is piloted by a function:**
  - **Useful in config:** For each level a range of exchangeable functions that the end user may choose from.
  - **Useful in devel:** new functionalities or variations of existing ones take the form of new functions (and the existent remains usable).
- In the config file one sets up a combination of functions on a set of fields → **similarity strategy**
- NB: At one level, functions are exchangeable because they share a common programming interface.

MarcXimiL flexibility at all levels

- Parsing functions (3 type of MARC fields + concatenation)
- Pre-treatment functions (RAW, WC, INITIALS, SHINGLES) + normalisation (case, diacritics, punctuation...)
- Comparisons pairs and order (triangular, multiple colls ...)
- Global record similarity : 6 weighted averages and several specialized functions (maxsim, ubiquist, boundaries, ...)
- Field similarity functions: authors, date/year, vectorial[Dice,Salton,Jaccard], probablistic, initials, shingles, doi/uid, Levenshtein based, % of items, ...
- Output functions : tabs, XML, devel. + global threshold

Configuration

```
INPUT_FILES = ['testdataCERNb.xml']
records_comp = records_comp_single
report = report_tab
globalvars.output_threshold = 0.5
record_rules = geometric_mean_breakout

record_structure = { \
        '01recid'   :{'marc'       : '001',
                      'weight'     : 0,
                      'parse-func': parse_controlfield,
                      'comp-func' : fields_concat__raw },
        '02year'    :{'marc'       : '260  c',
                      'weight'     : 1,
                      'parse-func': parse_nonrep,
                      'comp-func' : years_comp__raw },
        '03authors' :{'marc'       : ['100  a', '700 a'],
                      'weight'     : 2,
                      'parse-func': parse_multi,
                      'comp-func' : authors_comp__raw },
        '04title'   : {'marc'       : ['245  a', '245  b'],
                      'weight'     : 3,
                      'parse-func': parse_concat,
                      'comp-func' : okapibm25__wc },
        '05title'   :{'marc'       : '245  a',
                      'weight'     : 3,
                      'parse-func': parse_nonrep,
                      'comp-func' : levenshtein__raw } }
```

etc/similarity_config.py

=

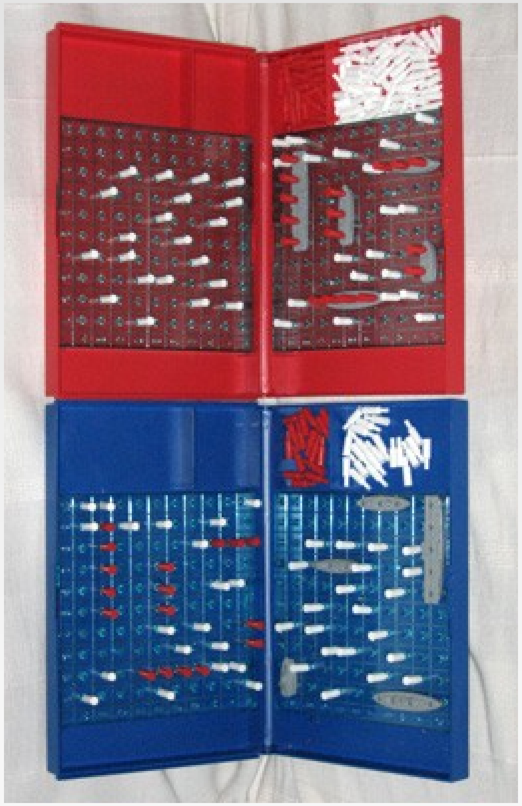
a strategy

MarcXimiL performance

Double blind study with only 1 rule :
a collection = 1990 records + 10 engineered near-duplicates

AB
5 strategies

4 collections
RERODOC et
ETH E-Collection
(article + theses),
•Frequent errors
•Cataloging variations



JK
5 strategies

4 collections
CERN (articles),
10 records with
Increasingly severe
modifications
(d > c > b > a)

Results: and the winners are ...

Strategy	Recall 10	Recall 20	Recall 50	Average	Timing [min]
2geom	5.6	6.8	8.5	7.0	8.3
2geombreak	5.0	6.4	7.5	6.3	7.1
abstract_fallback	5.4	6.9	9.1	7.1	5.3
boundaries_max	4.9	6.4	9.1	6.8	5.8
geometric_jk	4.1	7.3	8.6	6.7	6.4
initials	7.4	8.1	8.8	8.1	5.6
initialsbreak	7.3	7.3	8.9	7.8	5.3
maxsim	5.1	6.9	9.4	7.1	56.1
okapigeom	5.9	7.4	8.6	7.3	7.6
ubiquist	7.2	8.5	9.0	8.2	7.9

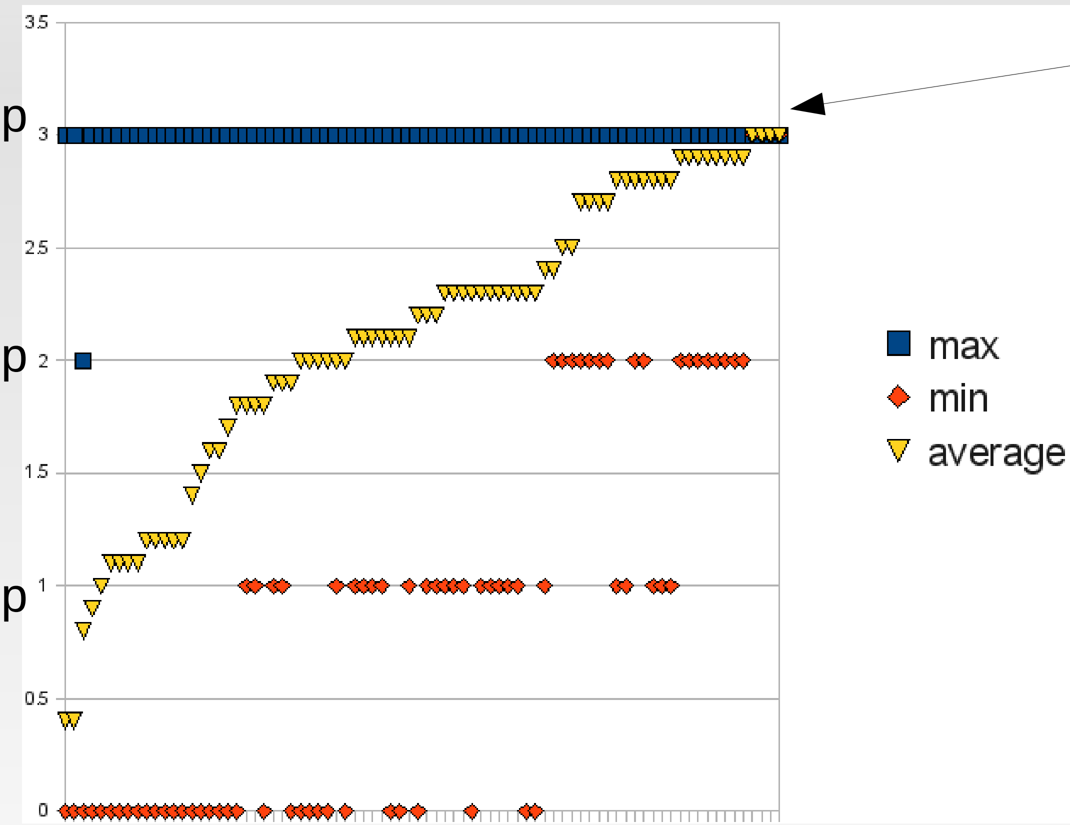
Results: complementarity

Best strategy = combine good and complementary strategies

Top 10

Top 20

Top 50



Engineered duplicates

At least one method was almost always able to place each engineered duplicate in the top 10.

Initials | ubiquist
=> 93% of precision
Initials & ubiquist
=> 46% of precision

Initials | ubiquist | okapigeom
=> 96% of precision
Initials & ubiquist & okapigeom
=> 29% of precision

Results: recall & noise analysis

The best strategies withstand well up to 2 altered fields:

1/3 of authors **OK**
1/3 of title **OK**
Several sentences from the abstract **OK**
2 years of difference **OK**
Word permutations **OK**
Author format modification **OK**

Most difficult:

Typos in title (except using the Levenshtein algorithm that is slow)

The best approach (withstands 3 or 4 altered fields)

Combine the results of the 3 best unrelated strategies (withstands 3 or 4 alterations)

Ameliorations:

Ubiquist : use subtitles as well (if cataloguing rules are compatible).
Initials : use digrams, soundex, or initials with shingling (to avoid collisions).

Results: speed

- Python 3.1.x is faster than Python 2.x (~25%)
- Duration increases rapidly with the number of records:

  Dell Latitude D620 (Intel 1.83GHz) / Geometric mean on 5 fields.
  - 1'000 records in less than a minute
  - 2'000 records in less than 5 min
  - 5'000 records in less than half an hour
  - 10'000 records in less than 3 hours

- Adjustments must be made for large collections.
  - NB: If a catalogue receives 100 records weekly, 5000 record yearly : check the new records against the last two years will take:

    100 * (2*5000) = 1'000'000 comparisons = much less than 5 minutes

Speed optimisations

In total possibly more than 1'000x or 10'000x:

- Multiprocessing module (Python>=2.6) : use all CPUs. Just replace the loop on record pairs by multiprocessing execution. ~ 3x (quadcore)
- Python code optimisation (profiling) : > 2x
- Compile comparison and indexation functions : C++, ... ~ 2-20x
- Optimise record global similarity functions : stop field comparisons as soon as possible ~ 1.2x
- Intelligent records_comp functions : e.g. pre-grouping based on subject indexation, then full comparisons on subsets only 100-1000x
- More efficient caching : use a relational database ~ 1-20x

MarcXimiL : other functionalities

Application	Description
Core – similarity.py	Similarity analysis
Core – batch.py	Batch similarity analysis
Core – sort.py	Sort and truncates output
Core – colldescr.py	Collection statistical description
Core – oai.py	OAI-PMH2 harvesting
Core – text2xmlmarc.py	Conversion: text MARC VTLS to MARCXML
enrich.py (prototype)	« More litke this » enhancement for catalogues
monitor.py (prototype)	Invenio based information monitoring
plagiarism.py (prototype)	Plagiarism detection and KB management
visualize.py (prototype)	2D Graph representation
semantic.py (prototype)	Semantic MARCXML collection editor

CDS Invenio : a way of integration

- MarcXimiL as python package (import in Invenio, e.g. bibsim, that would be bibsched compatible).
- Put Invenio search equations in config file instead of file names
- New level of flexibility: load_records_invenio function and associated field parsing functions
- Create a new output function that is able to feed back similarity data to Invenio. Or eventually use enrich.py to do so, then : `bibupload -c collection.xml`

Conclusion

- MarcXimL = good near-duplicate detection tool
- Offers many other opportunities: information monitoring, plagiarism detection, and so on.
- Main strength: flexibility in setup as well as in future developments.
- For large collections, speed is presently an issue. But this can be improved in many ways.
- Integration with Indico should be quite straight forward (depending on the requirements).