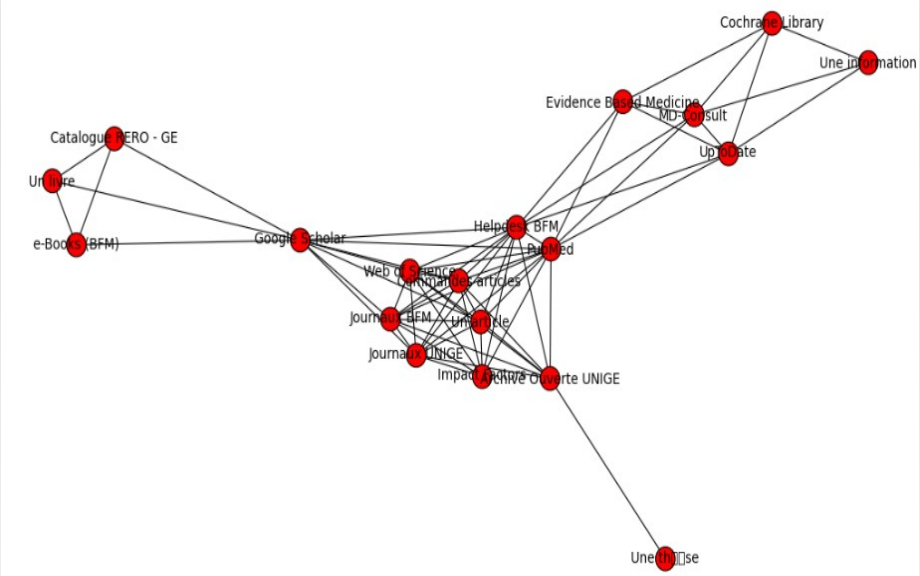
MarcXimiL



Analyse de similarité

- **détection de doublons**
- veille documentaire
- analyse structurelle de collections
- détection de plagiat
- suggestions à l'usager

http://marcximil.sourceforge.net

MarcXimiL : philosophie

- **Flexible à tous les niveaux de l'analyse**
- Algorithmes importés de la recherche d'informations + algorithmes spécifiques
- Open source : GPLv3
- Configuration centralisée (stratégie de similarité)
- Standard (MARCXML, OAI-PMH2, structure UNIX)
- Compatible (tout systèmes, python 2.4.0 → 3.1.1)
- Installation simple

MarcXimiL : flexibilité

- **Méthode: chaque niveau de l'analyse de similarité est pilotée par une fonction**
  - **Utile pour la configuration :** à chaque niveau l'usager peut choisir parmi une gamme de fonctions interchangeables.
  - **Utile pour le développement :** pour introduire une nouvelle fonctionnalité → ajouter une fonction.
- NB: Chaque niveau de l'analyse est caractérisé par une **interface de programmation standardisée**, utilisée par toutes les fonctions du niveau, et prévoyant tous les cas de figures.

MarcXimiL : niveaux de flexibilité

- Fonctions de chargement (MARCXML, Invenio, futur: DC, Onyx ....?)
- Fonctions de parsing des notices (3 type de champs MARC + multi-champs: concaténation, ensembles)
- Fonctions de pré-traitement des champs (RAW, WC, INITIALS, SHINGLES, ...) + normalisation (case, diacritiques, ponctuation...)
- Schéma des comparaisons : les paires de notices (triang., 2 colls...)
- Similarité entre notices : moyennes pondérées (classiques & rapides), et fonctions spécialisées (maxsim, ubiquist, boundaries, ...)
- Similarité entre les champs: auteurs, date/année, doi/uid, textuelles (vectorielle [Dice,Salton,Jaccard,Initials, Shingles], probabiliste, Levenshtein, ensembliste), ...
- Fonctions de sortie : tabulé (tableur) , XML, ... Devel. : Invenio

Stratégie de similarité (configuration)

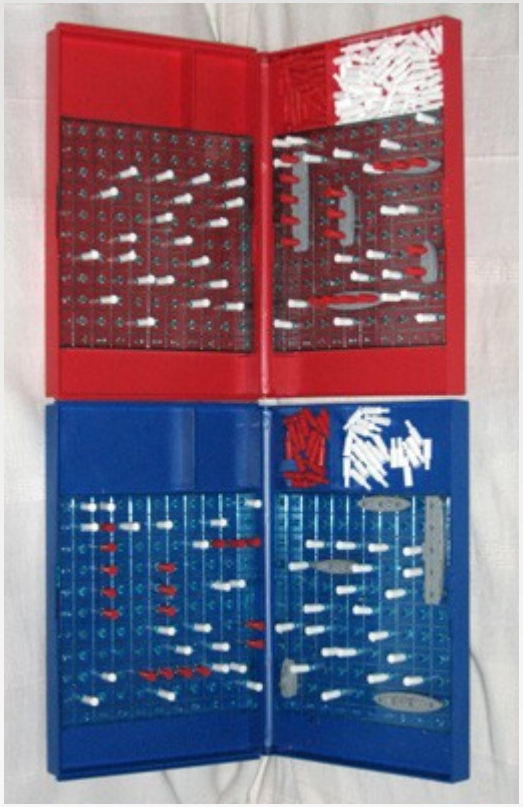
```
INPUT_FILES = ['collection.xml']
records_comp = records_comp_single
report = report_tab
record_rules = geometric_mean

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

MarcXimiL : performance

Étude en double aveugle (une règle) :
collection = 1990 notices + 10 quasi-doublons

AB
5 stratégies

4 collections
RERODOC et
ETH E-Collection
(article + thèses),
•Erreurs fréquentes
•Variantes catalogage



JK
5 stratégies

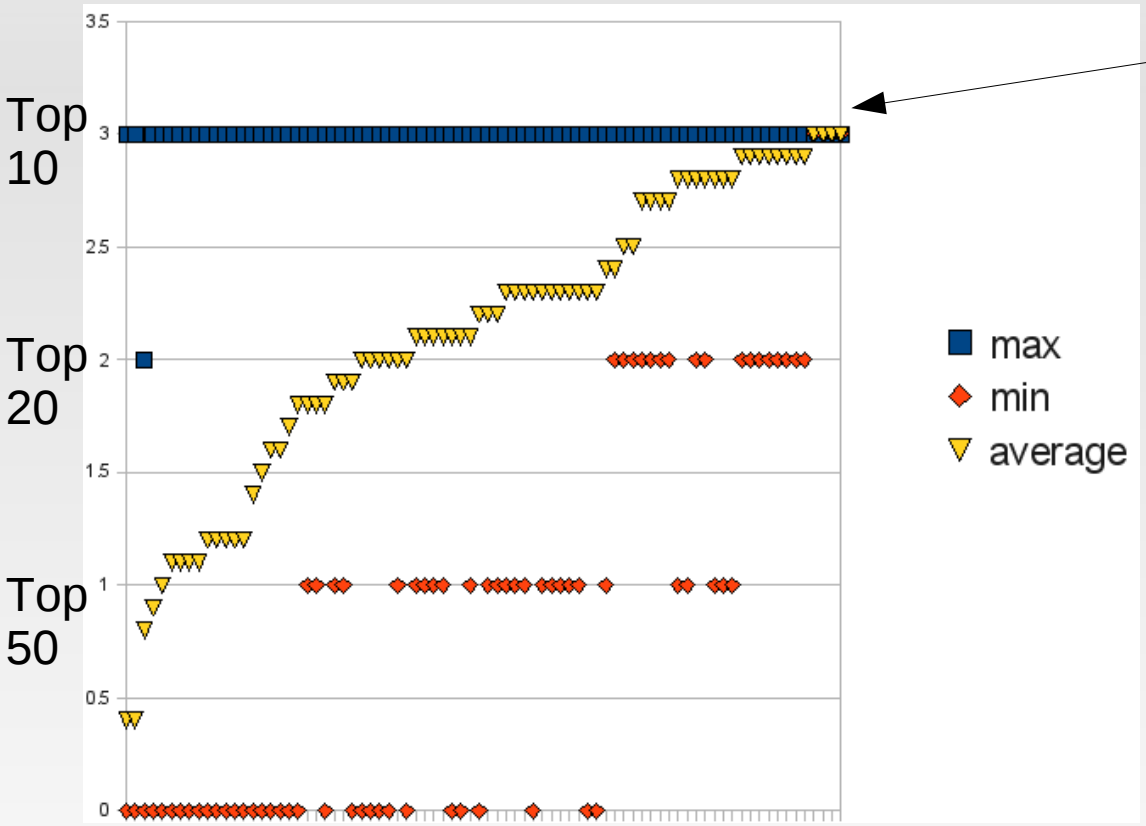
4 collections
CERN (articles),
10 notices avec
Modifications de plus
en plus sévères
(d > c > b > a)

MarcXimiL : performance

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

Résultats : complémentarité

Meilleure stratégie = combiner les meilleures stratégies



Engineered duplicates

Au moins une stratégie réussit à placer caque doublon dans les 10 premiers résultats.

Initials | ubiquist
=> 93% de précision
Initials & ubiquist
=> 46% de précision

Initials | ubiquist | okapigeom
=> 96% de précision
Initials & ubiquist & okapigeom
=> 29% de précision

Results: bruit et rappel

Les meilleures stratégies supportent jusqu'à 2 modifications t.q.:

1/3 des auteurs **OK**
1/3 du titre **OK**
plusieurs phrases de l'abstract **OK**
2 ans de différence **OK**
permutations de mots **OK**
Modification du format des auteurs **OK**

Le plus difficile:

Fautes d'authographe dans le titre (sauf avec Levenshtein, qui est lent)

Améliorations de la précision (v0.3):

- Ubiquist : concaténer les sous-titres aux titres
- Initials : utiliser des digrammes / soundex (pour éviter les collisions).
- Combinier les meilleures stratégies (supporte altérations de 3 ou 4 champs)

Resultats : vitesse

- La temps de traitement augmente rapidement avec le nombre de notices:

  Dell Latitude D620 (Intel 1.83GHz) / Moyenne géom. sur 5 champs.
  - 1'000 notices en moins de 1 minute
  - 2'000 notices en moins de 5 min
  - 5'000 notices en moins de ½ heure
  - 10'000 notices en moins de 3 heures
- Il faut faut faire des ajustements pour traiter les grandes collections en bloc, mais si on analyse que le flux entrant:
  - Si un catalogue reçoit ~100 notices par semaine, soit ~5000 notices par année, la détection de doublons sur les notices de la semaine par rapport aux deux dernières années prend:
    - 100 * (2*5000) = 10^6 comparaisons = env. 2 minutes
    - Pour ~1000 notices hebdomadaires -> 100 · 10^6 comparaisons -> 6h

Optimisation de la vitesse

Possiblement jusqu'à 1'000x ou 10'000x:

- **Module multi-processing** (Python>=2.6) — ~ 3x (quadcore)
- Python code optimisation (profiling) — > 2x
- Compiler les fonctions critiques (C++) — ~ 2-20x
- Optimiser les fonctions de similarité entre notices — ~ 1.2x
- **Pré-clustering**: subject headings, publication year, collections, ... — 100-1000x
- Caching plus efficace : utiliser un SGDBR — ~ 1-20x
- Optimiser les index ntf-nidf (tout pré-calculer) — ~ 1.5x

Améliorations v 0.3 : pré-clustering , multiprocessing -> accélération de min. 300x

MarcXimiL : autres fonctionnalités

Application	Description
Core – similarity.py	Analyse de similarité
Core – batch.py	Analyse de similarité en batch
Core – sort.py	Trie et tronque l'output
Core – colldescr.py	Description statistique d'une collection
Core – oai.py	Moissonnage de métadonnées par OAI-PMH2
Core – text2xmlmarc.py	Conversion: text MARC VTLS à MARCXML
enrich.py (prototype)	« More litke this » pour catalogues
monitor.py (prototype)	Veille documentaire basée sur Invenio
plagiarism.py (prototype)	Détection et gestion de la base de connaissance
visualize.py (prototype)	Représentation des résultats en Graph 2D
semantic.py (prototype)	Éditeur de relations sémantiques

MarcXimiL : dans le futur proche

- Ajouter une interface graphique simple (l'API est déjà faite)
- Archive Ouverte UNIGE (JK) ?
- EPFL Infoscience (AB)
  - ~ 75'000 notices
  - Rang webmometrics[1]: 15ème mondial
- CDS Invenio (AB+JK / API à convenir cette semaine)
  - ~ 1 million de notices
  - rang ROAR/activity : top 10 mondial (bien devant PMC)
  - rang webometircs[1] : 19ème mondial
  - autres instances locales: rerodoc, infosciences

1: http://repositories.webometrics.info

A propos d'Inspire www.projecthepinspire.net

- HEP : CERN, DESY, SLAC (SPIRES), Fermilab
  - TOUTES les données bibliographiques du domaine
- Fonctionnalités ajoutées récemment à Invenio:
  - Notices riches: fulltext, code, multimédia, citations, ...
  - Thésaurus: indexation automatique + folksonomie
  - Blogging sur les notices
  - Modules complets de circulation et de catalogage
  - Métriques sur les auteurs et gestion offres emplois
  - APIs et formats d'exportation
  - ... *Qualité des collections (doublons, fusion de notices, ...)*

MarcXimiL : conclusion

- Bon outil de détection de quasi-doublons.
- Offre d'autres opportunités: détection de plagiat, veille documentaire, etc.
- Atout principal : très grande flexibilité.
  - Configuration
  - Développement
  - Interopérabilité (MARCXML, OAI-PMH2)
- Faiblisse : la vitesse (version actuelle 0.2.1b).
  - Mais des solutions pour traiter les grandes collections sont en cours d'implémentation (v0.3).