GRAS Stemmer

Marco Romanelli – Federico Ghirardelli

Index

- GRAS
- Implementation
- Performances
- Evaluation
- Conclusions

Bibliography

GRAS:

An Effective and Efficient Stemming Algorithm for Information Retrieval

JIAUL H. PAIK, MANDAR MITRA, and SWAPAN K. PARUI

Indian Statistical Institute

KALERVO JA¨RVELIN

University of Tampere

Purpose

1. Retrieval effectiveness
2. Language independent
3. Low computational cost

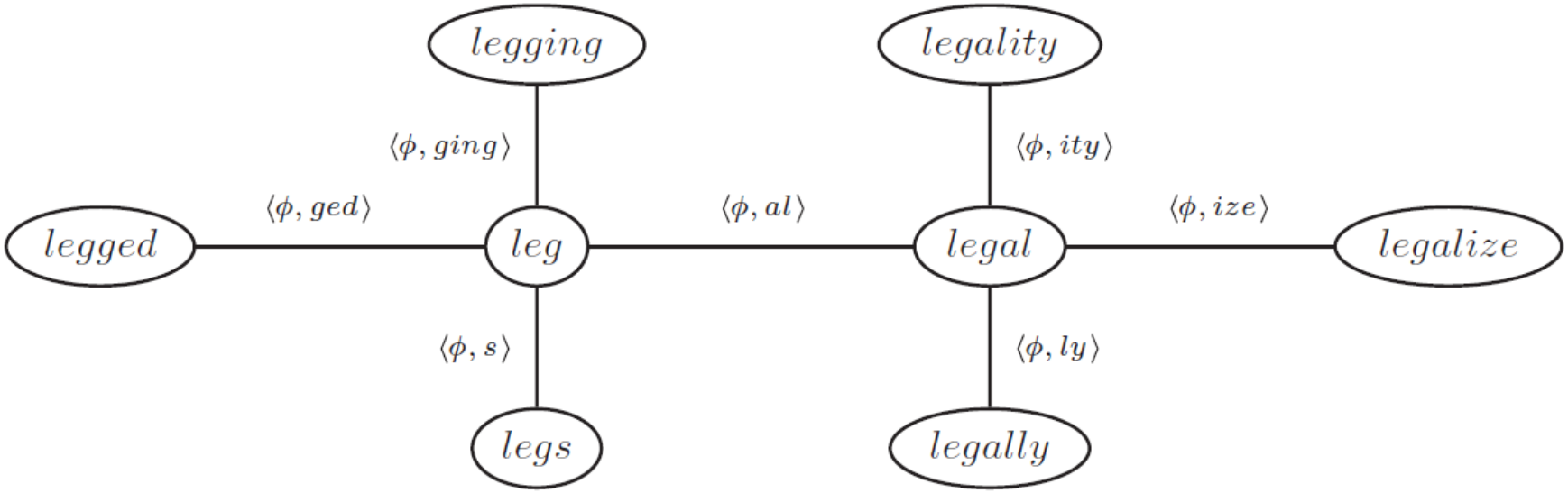
GRAS (2)

- Suffix Pair Identification
  - Partition words into groups such that any pair of words shares a LCP greater than a threshold.
  - For any pair of words w_j, w_k in each group, compute the suffix pair (s_1, s_2) such that:
    - $w_j = r\ s_1$, $w_k = r\ s_2$ and r is the Longest Common Prefix
  - Then compute the frequency of all suffix pairs

GRAS (3)

- Class Formation
  - Construct a weighted undirected graph where
    - node represent word
    - edge connect related words (using the suffix pairs)
  - Identify the *pivot* node
  - If a node share many common neighbors with the pivot, put it in the same class
  - Stemming is done by mapping all the words in a class to the pivot for that class.

- Class Formation



Implementation

- **Python**
  - PROS: Read it, use it with ease.
  - CONS: relative slow speed of execution.

Implementation

- **Python**
  - PROS: Read it, use it with ease.
  - CONS: relative slow speed of execution.
- **Problem:**
  - Computing the (alpha) suffix pairs takes too long (more then one hour)

Implementation

- **Python**
  - PROS: Read it, use it with ease.
  - CONS: relative slow speed of execution.
- **Problem:**
  - Computing the (alpha) suffix pairs takes too long (more then one hour)
- **Possibile solutions:**
  - try other libraries, modules, python compiler
  - use some supporting data structures
  - change programming language

Implementation (2)

- **Java**
  - PROS: sufficient hight computational speed
  - CONS: more laborious than a scripting language (like python)

Implementation (2)

- **Java**
  - PROS: sufficient hight computational speed
  - CONS: more laborious than a scripting language (like python)

- **Problem remains:**
  - but only with word generated from german corpus

Time and memory consumptions

	Italian	German (SDA1994)	Russian
Time	16 : 42.40	21 : 25.82	13 : 00.88
Memory	537′500	512′552	525′556
$\|V\|$	357′904	373′156	377′166
$\|E\|$	516′066	119′812	204′885

- Time: elapsed real (wall clock) time of the process [minutes:seconds]
- Memory: RSS size of the process [Kilobytes]

Collections

CLEF Ad-Hoc experimental collection for Italian, German and Russian.

	Italian	German	Russian
No. of documents	157′558	225′371	16′716
No. of unique words	357′904	1′026′017	377′166
No. of queries	51	56	28

Evaluation: Italian

Table 3.2: Retrieval results for Italian

Pipeline	RUN	MAP	P@10	P@30
stoplist, Snowball	1	0,0795	0,049	0,0418
stoplist2, Snowball	2	0,0792	0,0451	0,0399
stoplist, GRAS	3	0,2987	0,2314	0,1431
stoplist2, GRAS	4	**0,3079**	**0,2373**	**0,1516**
Snowball	5	0,0823	0,049	0,0444
GRAS	6	0,3075	0,2529	0,1503

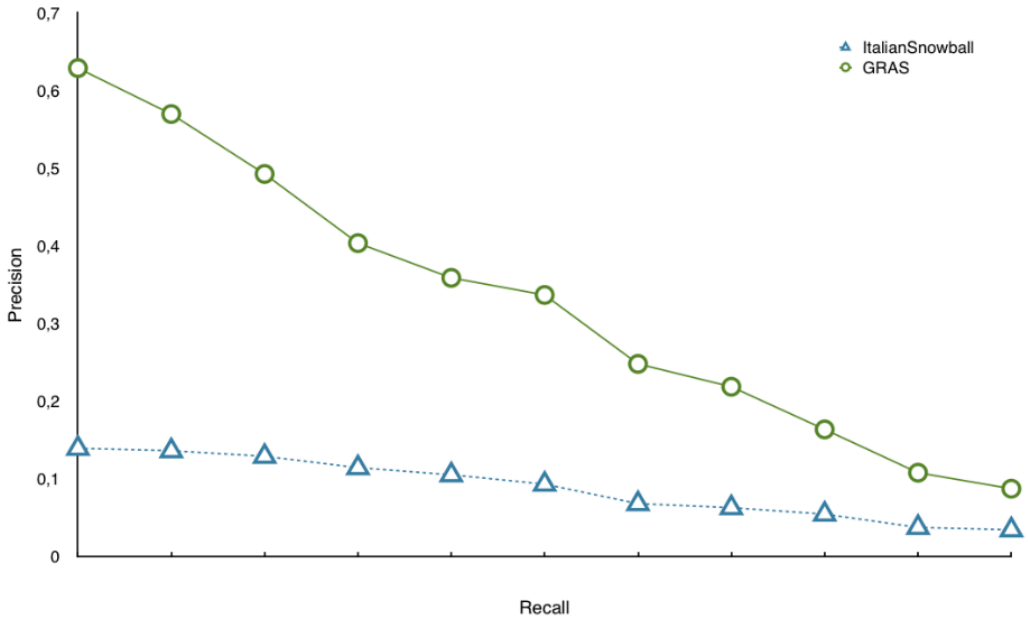


Figure 3.1: Precision recall curve for Italian

Evaluation: German

Table 3.4: Retrieval results for German

Pipeline	RUN	MAP	P@10	P@30
stoplist, Snowball	1	0, 1383	0, 2107	0, 1375
stoplist2, Snowball	2	0, 1358	0, 2089	0, 1381
stoplist, GRAS	3	0, 1902	0, 2857	0, 1762
stoplist2, GRAS	4	**0, 1923**	**0, 2929**	**0, 1863**
Snowball	5	0, 1349	0, 2054	0, 1357
GRAS	6	0, 1903	0, 2964	0, 1863

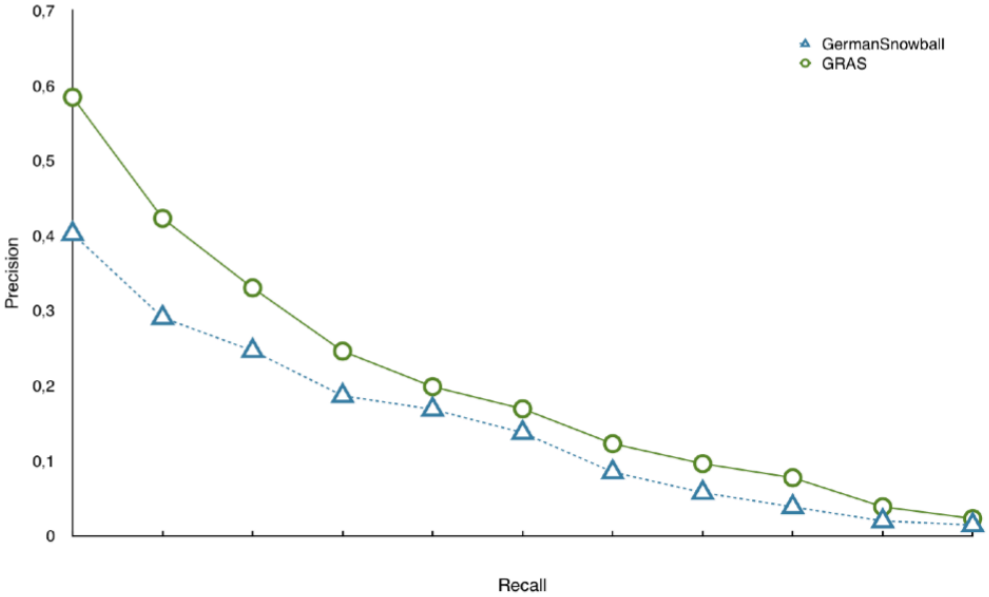


Figure 3.3: Precision recall curve for German

Evaluation: Russian

Table 3.3: Retrieval results for Russian

Pipeline	RUN	MAP	P@10	P@30
stoplist, Snowball	1	0, 1381	0, 0857	0, 0405
stoplist2, Snowball	2	0, 1362	0, 0893	0, 0405
stoplist, GRAS	3	**0, 3549**	**0, 2071**	0, 100
stoplist2, GRAS	4	0, 3515	0, 2071	**0, 1024**
Snowball	5	0, 1386	0, 0857	0, 0381
GRAS	6	0, 3424	0, 2036	0, 0976

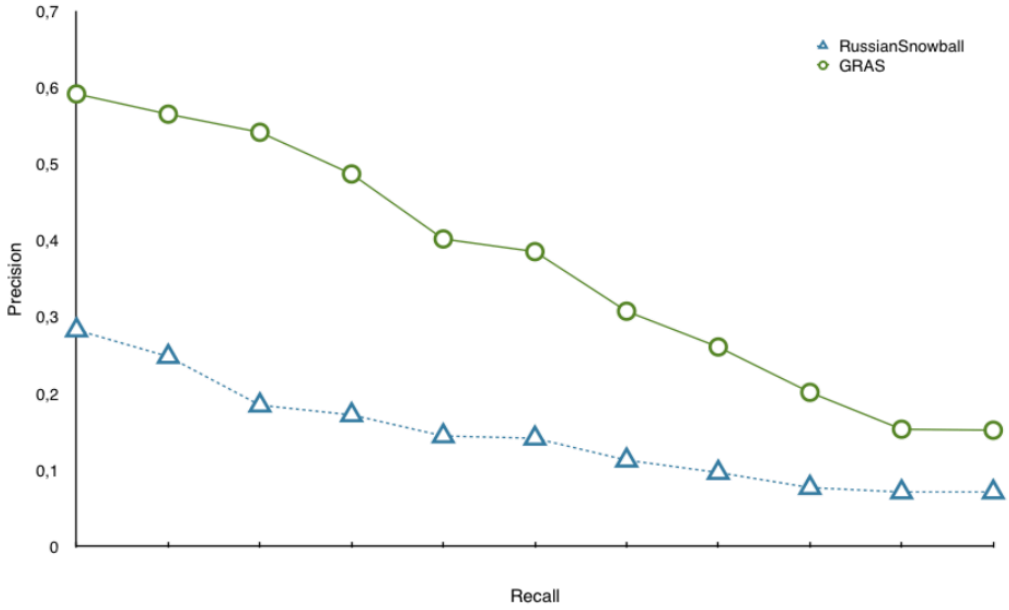


Figure 3.2: Precision recall curve for Russian

Further Improvements

- **Implement supporting data structures**
  - Suffix tree (Tries)
- **Optimize java code**
  - Parallelism / Concurrency

THE END

Parameter setting

- ***l***
  - threshold for the longest common prefix (average word length)
- ***alpha***
  - suffix frequency cut-off (4, 6, 8)
- ***delta***
  - cohesion cut-off (0.8, 0.9)