The University of Padova at CLEF 2003: Experiments to evaluate probabilistic models for automatic stemmer generation and query word translation

G.M. Di Nunzio N. Ferro M. Melucci N. Orio

University of Padova
Department of Information Engineering

{dinunzio,nf76,melo,orio}@dei.unipd.it

1 Introduction

In 2002, we, the Information Management Systems (IMS) research group of the Department of Information Engineering of the University of Padova, were a newcomer to CLEF and participated to the mono-lingual track and specifically carried out experiments on stemming Italian text. The participation was quite successful because the proposed graph-based stemmer generator allowed to reach retrieval effectiveness being comparable to the Porter's stemmer.

This year we have again participated to the monolingual track but have enlarged the spectrum of languages up to five and have re-defined the theoretical framework underlying the stemmer generator - from a graph-based framework we moved to a probabilistic framework, yet keeping the notion of mutual reinforcement between stems and derivation, which characterizes our proposal. In addition to this, we have experimented this year a new approach to stemmer generation based on Hidden Markov Models, which has reached good results too. We have also participated to the bi-lingual (language-to-English) track to test whether the same mutual reinforcement relationship can apply to machine readable dictionary-based keyword translation. The increased number of languages and then of sub-collections to be indexed required to re-engineer the information retrieval system used in our experiments.

2 Mono-lingual Track Experiments

The approach that has been carried out for mono-lingual experiments focused on the development of stemming algorithms for each of the five tested languages (Dutch, French, German, Italian, Spanish). Our goal was to develop algorithms that do not exploit linguistic knowledge about the morphology of a given language and the rules to form words derivations. To this end, we make the assumption that statistical models can be inferred from the set of words of a given language and that they can be applied for stemming words. We tested two different approaches. In Section 2.1 we present a model based on a probabilistic framework for the notion mutual reinforcement between stems and derivation. In Section 2.2 we present an approach based on Hidden Markov Models (HMMs).

2.1 A Probabilistic Framework for Stemmer Generation

We have designed and experimented a probabilistic version of the *Stemming Program for Language Independent Task* (SPLIT), which is a language independent stemming algorithm originally

proposed in [2, 1] and being capable of automatically generate stemmers for Indo-European languages. SPLIT rests on a suffix stripping paradigm and assumes that the stem of a word can be found splitting the word, and holding the first part as stem. So the problem of stemming is to find the right split for each word. In order to develop SPLIT:

- we define a *probabilistic framework*, which sets up a suitable environment for the development of such algorithm, as explained in Section 2.1.1;
- we employ, within the probabilistic framework, a model based on the concept of *mutual reinforcement* between stems and derivations, as presented in Section 2.1.2;
- we design an *algorithm* which carries out the mutual reinforcement model and generates stemmers for Indo-European languages according to the probabilistic framework, as described in Section 2.1.3.

2.1.1 Probabilistic Framework

Given a finite collection W of words, a word $w \in W$ of length n can be split into $n-1$ possible positions, which do not generate empty sub–strings. So each split is associated with a pair of sub–strings, named *prefix* and *suffix*, which form the word w once they are concatenated. Among all the possible pairs (prefix, suffix) of a word, only one is the pair corresponding to the *stem* and *derivation* of the word. The probabilistic framework assumes that the pairs (prefix, suffix) are not equiprobable, but that the concatenation of a stem with a derivation is an event more probable than the concatenation of two generic prefixes and suffixes.

Thus we can employ a maximum likelihood (ML) criterion for identifying the most probable pair of sub–strings. i.e. the stem and derivation, as shown in figure 1. Let U be the set of sub–

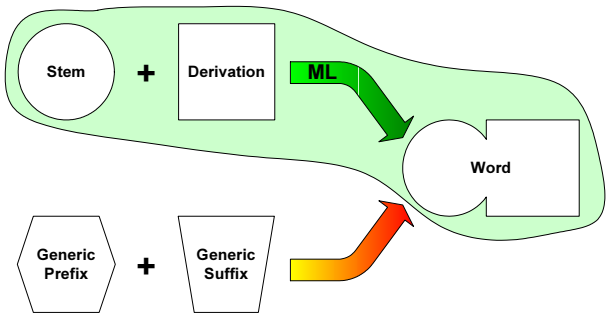

Figure 1: Choice of the (stem, derivation) pair of a word within the probabilistic framework.

strings generated after splitting each word $w \in W$ into all the possible positions. If $x \in U$ and $y \in U$ are the prefix and the suffix of the word w respectively, then we obtain that $w = xy$. Let $\Omega = \{\omega = (x, y) \in U \times U : \exists w \in W, w = xy\}$ be the set of all possible pairs (prefix, suffix), and $\Omega(w) = \{\omega = (x, y) \in \Omega : w = xy\}$ the set of all the pairs (prefix, suffix) leading to the word w. Thus, given a word w, the pair $\omega^* = (\text{stem}, \text{derivation})$ can be found as follows:

$$\omega^* = \arg\max_{\omega \in \Omega} \Pr(\omega \mid w) \stackrel{a}{=} \arg\max_{\omega \in \Omega} \frac{\Pr(w \mid \omega)\Pr(\omega)}{\Pr(w)} \stackrel{b}{=} \arg\max_{\omega \in \Omega(w)} \Pr(\omega) \tag{1}$$

where (a) is obtained applying the Bayes Rule and (b) is obtained observing that $\Pr(w \mid \omega) = 1$, since $\omega \in \Omega(w)$ yields to w only, and $\Pr(w)$ is the same for all ω and so it does not influence the maximization.

2.1.2 Mutual Reinforcement Model

A mutual reinforcement relationship exists among sub-strings of a language and can be stated as follows: Stems are prefixes that are more likely to be completed by derivations, and derivations

are suffixes that are more likely to complete stems. Figure 2 presents an intuitive view of the mutual reinforcement relationship: on the left of the figure a community of stems and derivations is shown; note that we can guess that `comput` is a stem, since it refers to sub–strings, such as `ation` and `er`, which in turn are used in order to create also other words, such as `compilation` or `reader`. On the right of the figure, a community of generic prefixes and suffixes is shown; note that we discard the prefix `c` as stem, since it forms words with suffixes, such as `omputer` or `ompilation`, which are not used in turn for creating other words. Note that if you wrote the words resulting from the graph of Figure 2 and then did split them into all the possible positions, the new resulting graph would have a much larger number of nodes and edges than that of Figure 2. For sake of clarity we have left, on the left, the nodes and edges corresponding to the best splits, and on the right a subset of the other possible splits. This sort of *coupled frequent usage* allows

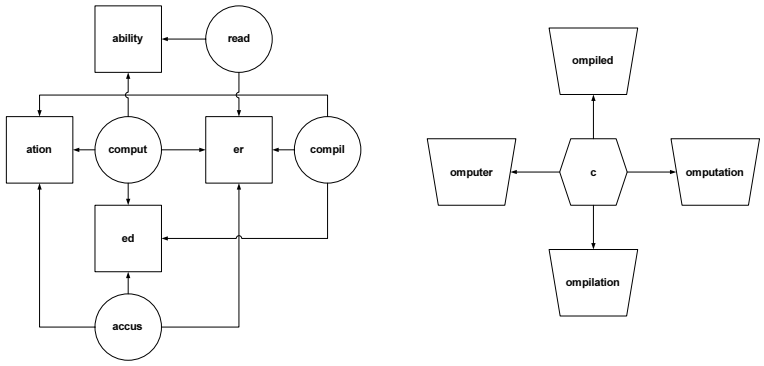

Figure 2: Example of mutual reinforcement relationship.

us to estimate the probability distribution of prefixes and suffixes, which is necessary in order to compute the probability distribution of the pairs for applying the probabilistic framework.

The mutual reinforcement relationship can be expressed using the total probability theorem:

$$\begin{aligned} \Pr(x) &= \sum_{y \in Y} \Pr(x, y) = \sum_{y \in Y} \Pr(x \mid y) \Pr(y) \\ \Pr(y) &= \sum_{x \in X} \Pr(x, y) = \sum_{x \in X} \Pr(y \mid x) \Pr(x) \end{aligned} \tag{2}$$

where:

- $\Pr(x)$ is the probability that x is a good prefix, that is a prefix candidate to be a stem. Similarly $\Pr(y)$ is the probability that y is a good suffix, that is a suffix candidate to be a derivation;
- $\Pr(y \mid x)$ is the conditional probability of transition from the prefix x to the suffix y. Viceversa for $\Pr(x \mid y)$;
- $X \subseteq U$ is the set of all the prefixes and $Y \subseteq U$ is the set of all the suffixes.

2.1.3 The SPLIT Algorithm

Figure 3 shows the architecture of the SPLIT algorithm, which is described in the following:

- *Prefix/Suffix Estimation* is a global step, which tries to infer some knowledge about the language, estimating the distribution of prefixes and suffixes, according to equation (2). It is a global step, since it concerns the whole set U and not a word in particular. This step uses the following estimations for transition probabilities:
  - $\Pr(x \mid y) = \frac{1}{P_y}$, where $P(y) = \{x \in U : \exists w \in W, w = xy\}$ and $|P(y)| = P_y$ is the number of words which end with suffix y;

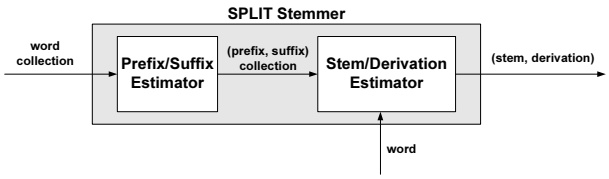

Figure 3: Architecture of the SPLIT algorithm.

– $\Pr(y \mid x) = \frac{1}{S_x}$, where $S(x) = \{y \in U : \exists w \in W, w = xy\}$ and $|S(x)| = S_x$ is the number of words which begin with prefix x;

The algorithm iteratively computes:

$$\Pr^{(t)}(x) = \sum_{y \in Y} \Pr(x \mid y) \Pr^{(t-1)}(y)$$
$$\Pr^{(t)}(y) = \sum_{x \in X} \Pr(y \mid x) \Pr^{(t)}(x)$$

for $t = 0, 1, 2, \ldots$, where $\Pr^{(0)}(y)$ is a vector of uniform probabilities.

- *Stem/Derivation Estimation* is a local step, which tries to distinguish among all the pairs leading to the same word, according to equation (1). It is a local step, since it concerns a particular word. Equation (1) is solved in two distinct ways:

$$\arg \max_{\omega \in \Omega(w)} \Pr(\omega) = \arg \max_{\omega \in \Omega(w)} \Pr(x, y) \propto \begin{cases} \arg\max_{\omega \in \Omega(w)} \Pr(x) \Pr(y) & \text{Case 1} \\ \arg\max_{\omega \in \Omega(w)} \Pr(x) \Pr(y \mid x) & \text{Case 2} \end{cases}$$

Case 1 assumes that, during the estimation of prefix and suffix probabilities, given by equation (2), the probabilities $\Pr(x)$ and $\Pr(y)$, each one on its own, have absorbed some knowledge about the morphology of the language. Therefore x and y can be considered as independent events and the probability of generation of the word w is the product of the two marginal probabilities. Case 2 assumes that the algorithm does not capture the whole mutual reinforcement relationship between x and y and that it is also necessary to consider a stochastic dependence.

In addition we inject a little of linguistic knowledge, inserting an heuristic rule which forces the length of the prefix to be at least α characters, and the length of the suffix to be at most β characters.

We used the CLEF 2002 collection in order to tune the algorithm and to set appropriate values for all the parameters. After the training, we chose the following parameters, since they gave the better performances in each language:

– Dutch: case 1 with $\alpha = 4$, $\beta = 4$;
– French: case 2 with $\alpha = 1$, $\beta = 3$;
– German: case 1 with $\alpha = 4$, $\beta = 4$;
– Italian: case 2 with $\alpha = 1$, $\beta = 3$;
– Spanish: case 1 with $\alpha = 3$, $\beta = 3$.

2.2 An Approach based on Hidden Markov Models for Stemmer Generation

We experimented another statistical approach to stemming, which is based on Hidden Markov Models (HMM) [3]. HMMs are finite-state automata where transitions between states are ruled

by probability functions. At each transition, the new state emits a symbol with a given probability. HMMs are called *hidden* because states cannot be directly observed, what is observed are only the symbols they emit. The parameters that completely define an HMM are, for each state: the probabilities of being the initial and the final state, the transition probabilities to any other state, and the probability of emitting a given symbol.

2.2.1 HMMs as Word Generators

HMMs are particularly useful in modeling processes that are, in general, unknown but that can be observed through a sequence of symbols. For instance, the sequence of letters that forms a word in a given language can be considered as a sequence of symbols emitted by a HMM. The HMM starts in an initial state and performs a sequence of transitions between states emitting a new letter at each transition, until it stops in a final state. In general, more state sequences, or *paths*, can correspond to a single word. It is possible to compute the probability of each path, and hence to compute the most probable path corresponding to a word. This problem is normally addressed as *decoding*, for which an efficient algorithm exists: the Viterbi decoding.

In order to apply HMMs to the stemming problem, like in the approach carried out for the SPLIT algorithm, a sequence of letters that forms a word can be considered the result of a concatenation of two subsequences of letters: a prefix and a suffix. A way to model this process is through a HMM where states are divided in two disjoint sets: states in the *stem-set* generate the first part of the word and states in the *suffix-set* possibly generate the last part, if the word has a suffix. For many Indo-european languages, there are some assumptions that can be made on the model:

- initial states belong only to the stem-set, i.e. a word always starts with a stem;
- transitions from states of the suffix-set to states of the stem-set have always a null probability, i.e. a word can be only a concatenation of a stem and a suffix;
- final states belong to both sets, i.e. a stem can have a number of different derivations, but it may also not have any suffix.

A general HMM topology that fulfills these conditions is depicted in Figure 4.

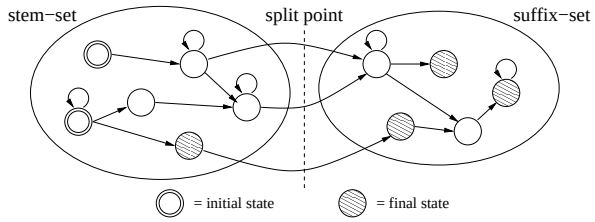

Figure 4: HMM topology with highlighted the stem-set and the suffix-set.

Once a complete HMM is available for a given language, stemming can be straightforwardly carried out considering a word as a sequence of symbols emitted by the HMM. As a first step, the most probable path that corresponds to the observed word is computed using decoding. Then the analysis of this path highlights the transition from a state of the stem-set to a state of the suffix-set, we call this transition the *split-point*. If there is no split-point then the word has no suffix, otherwise the sequence of letters observed before the split-point is taken as the stem and the one observed after is taken as the suffix.

2.2.2 Training the HMM

The proposed topology defines the number of states, their labels indicating the sets to which they belong, the initial and final states, and the allowable transitions. Yet all the probability

functions that constitute the HMM parameters need to be computed. The computation of these parameters is normally achieved through *training*, which is based on the Baum-Welch expectation-maximization (EM) algorithm. As in the case of SPLIT, our goal is to develop fully automatic stemmers that do not require previous manual work. This means that we consider that neither a formalization of morphological rules nor a training set of manually stemmed words are available.

We propose to perform an unsupervised training of the HMM using only a sample of the words of the considered language. The training set can be taken at random by documents that are available at indexing time. It can be noted that an unsupervised training does not guarantee that the split-point of the most probable path has a direct relationship with the stem and the suffix of a given word. With the aim of creating such a relationship, we propose to inject some more knowledge about the general rules for the creations of word inflections. Thus, we make the reasonable assumption that, for each language, the number of different suffixes is limited compared to the number of different stems. Suffixes are a set of letter sequences that can be modeled by chains of states of the HMM. This assumption suggests a particular topology for the states in the suffix-set, which can be made by a number of state chains with different lengths, where: transitions from the stem-set are allowed only to the first state of each chain; the transition from one state to the next has probability one; each chain terminates with a final state. The maximum length of state chains gives the maximum length of a possible suffix. Analogously, also the stem-set topology can be modeled by a number of state chains, with the difference that a state can have non-zero self-transition probability. The minimum length of a chain gives the minimum length of a stem. Some examples of topologies for the suffix-set are depicted in Figure 5, where the maximum length of a suffix is set to four letters, and the minimum length of a stem is set to three letters.

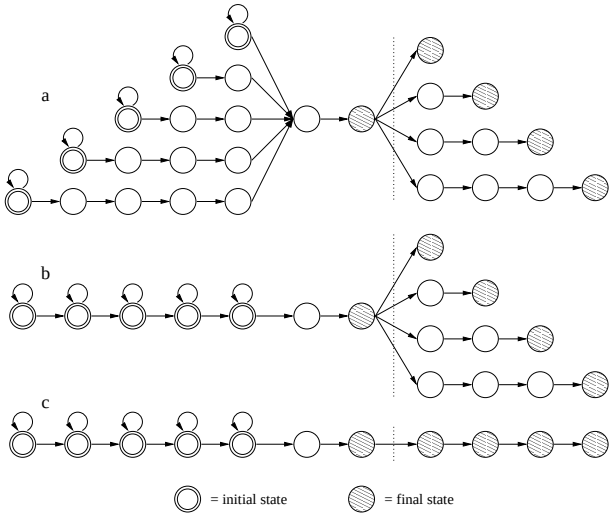

Figure 5: Three topologies of the HMM that have been tested.

After the redefinition of the suffix-set topology, the HMM can be trained using the EM algorithm on a set of words. Given the previous assumption, it is likely that a sequence of letters that corresponds to a suffix will be frequently present in the training set. For this reason, the EM algorithm will give to the states in the suffix-set a higher probability of emitting the letters of frequent suffixes. For example, considering the suffix-set chains, the state in the one-state chain will emit with the highest probability the last letter of each word, the states in the two-states chain will respectively emit the most frequent couple of ending letters of each word, and so one. Once the model has been trained, for each word the corresponding path that terminates with the most frequent sequence of letters it is expected to have an high probability of being selected as the most probable path, giving a correct stemming of the word.

2.2.3 The STON Algorithm

We develop an algorithm, named STON, to test the methodology and the changes on retrieval effectiveness depending on some of its parameters. STON needs an off-line training, while stemming can be performed on line for any new word. After the training, STON receives as input a sequence of letters corresponding to a word an gives as output the position of the split-point. Hence, STON performs in two steps:

- *Training/off-line*: given a set of words $w \in W_L$, taken from a collection of documents written in a language L, and a HMM with parameters λ which define the number of states and the set of allowable transitions, STON computes through the EM algorithm:

$$\lambda_L^* = \arg\max_{\lambda} \prod_{w \in W_L} Pr(w \mid \lambda)$$

  This step needs to be performed only once for each language, possibly using a sample of the words of a document collection.

- *Stemming/on-line*: given a word w_L, written in language L, and a trained model λ_L, STON computes the most probable path q across the states corresponding to w_L by Viterbi decoding:

$$q^* = \arg\max_{q} Pr(q \mid w_L, \lambda_L)$$

  Decoding can be carried out also for words that were not part of the training set.

  Once the most probable path q^* is computed, the position of the split-point, and hence the length of the stem, can be computed by simple inspection of the path, that is considering when the path enters the suffix-set.

3 Bi-lingual Track Experiments

Our approach to bi-lingual retrieval aimed at testing whether the notion of mutual reinforcement relationship can effectively work within the context of keyword translation based on machine readable dictionaries, like it has been shown to be effective within stemming as illustrated in Section 2.1. The approach can be summarized as follows.

Free dictionaries being available on the Web have been employed. We used the dictionaries being available at `http://www.travlang.com/Ergane` and `http://www.freedict.com/`. Origin words have been stemmed and dictionaries have been merged. Stemming increased the number of translations but reduced the number of entries. The following table summarizes the size and the coverage after stemming and merging the two dictionaries.

origin	entries	av. translations
German	67889	3.69
French	29819	2.97
Spanish	15697	3.52
Italian	8958	3.93

For each mono-language collection, the five most frequent collocates of each keyword were computed to create word contexts – the most frequent collocate is a keyword that frequently occurs just before or just after the keyword. The aim of this step is to overcome the problem of missed translations that might frequently occur whenever small or simple dictionaries are employed. If collocates are available, and an origin word cannot be translated, other translations of the collocates of the origin word can be found. In this way, an origin word might not be directly translated, but it is connected to the contexts of the target language that are related to the context of the origin word.

Translation has been performed between (origin) word contexts and (target) word contexts rather than between single words. As context translations are uncertain events, they have modeled using a probability space. Let X be the set of target words, A be the set of origin words, and $D \subseteq A \times X$ be the dictionary and also be the universe of elementary event. Using first letters of the alphabet to indicate origin words and the last ones to indicate the target words, we define $D_b = \{(a, x) \in D \mid a = b\}$ and $D_y = \{(a, x) \in D \mid x = y\}$ respectively as the subset of translations of a to any target word and the subset of translations of a for which y is a target word. We define contexts as subsets $A_b \subseteq A$ or $X_y \subseteq X$, respectively depending on whether they are of an origin word or of a target word. We say that a (target) context X_y is a translation of the (origin) context A_b if there is $(a, x) \in D$ such that $a \in A_b$ and $x \in X_y$. Of course, there may be $0, 1, \ldots$ pairs $(a, x) \in D$ such that $a \in A_b$ and $x \in X_y$ and then X_y is a translation of the (origin) context A_b to different degrees. Intuitively, the degree to which X_y is a translation of A_b is directly proportional to the size of the set $D_{by} = \{(a, x) \in D, a \in A_b, x \in X_y\}$.

The aim of the probabilistic model for translation is to find the best target context, i.e. the most probable target context that is a translation of the origin context. The mechanism can be formalized by $\Pr(X_y \text{ translates } A_b)$ and approximated by the conditional probability, i.e.

$$\Pr(X_y \text{ translates } A_b) \approx \Pr(X_y \mid A_b).$$

Given that the chance that X_y translates A_b is directly related to $|D_{by}|$, then we define

$$\Pr(X_y \mid A_b) = \frac{|D_{by}|}{|D_b|} \qquad \Pr(A_b \mid X_y) = \frac{|D_{by}|}{|D_y|}$$

At searching time, for each origin query word b, all the target contexts X_y corresponding to each possible translations y of b are considered. There may be several candidate translation contexts X_y and then a criterion to choose what translation context should be used is necessary. The idea is that the best translations of an origin context is one that is a translation of other origin contexts and that can be conversely translated back to one of the origin contexts. The mutual reinforcement relationship between contexts states that the best target translations of an origin context are translated by the best origin contexts and viceversa. Thus the following mutual definition is considered:

$$\Pr(X_y) = \sum_{A_b} \Pr(X_y \mid A_b) \Pr(A_b)$$

$$\Pr(A_b) = \sum_{X_y} \Pr(A_b \mid X_y) \Pr(X_y)$$

where $\Pr(X_y)$ is the probability that X_y is a translation of an origin context, $\Pr(A_b)$ is the probability that A_b is a translation of a target context.

The most probable X_y has been taken as translations of each query word b such that a translation of a word in A_b occurs in X_y. The experimental results have been rather dissatisfying. The Porter's stemming algorithm used for an origin language were rather aggressive and then were detriment of retrieval performance because many translations were erroneously mixed. Furthermore, the procedure to generate context was not very effective because many contexts contained unrelated words. Average precision was between 15% and 20%. Yet we think the approach is a seed for further research.

4 Experiments

The aim of our experiments of the mono-lingual track is to compare the retrieval effectiveness of the language independent stemmers, illustrated in the previous Sections, with that of an algorithm based on a-priori linguistic knowledge – we have chosen the Porter's stemmers because it is widely used. The hypothesis is that the proposed probabilistic approaches generate stemmers that perform as effectively as the Porter's ones. In order to evaluate stemming algorithms, we decided to compare the performance of an IR system, changing only the stemming algorithms for different runs, all other things being equal.

4.1 Experimental Prototype System

In order to carry out the evaluation campaign, we developed an experimental information retrieval system, called IRON (Information Retrieval ON), which has been used for the first time at CLEF 2002 and has been completely re-engineered for CLEF 2003. The aim is to design and develop a software tool, which provides a modular environment suitable for testing the performances of different IR components, e.g. stemmers, allowing us to easily plug-in the components under examination. It is java multi-threaded program, which provides IR functionalities and enables concurrent indexing and searching of document collections. It is built on top of the Lucene[1] 1.3 RC1 library, which is a high-performance text search engine library written entirely in Java.

IRON implements the vector space model, and a $(tf \cdot idf)$–based weighting scheme, as provided by the Lucene library, and offers an open software infrastructure which allows us to dynamically plug-in IR tools, such as lexical analyzers (lexers) or stemmers. For example, it is possible to add completely new stemmers at runtime, without affecting the existing code; this mechanisms is used for embedding Porter stemmers for different languages, which are developed and maintained by M. Porter at the Snowball[2] Web site. Moreover, implements an efficient lexer using JFlex 1.3.5[3], a lexer generator for Java, written in Java. This lexer is able to process any collection in a transparent way with respect to the user. Some details of its implementation will be given in section 4.1.1.

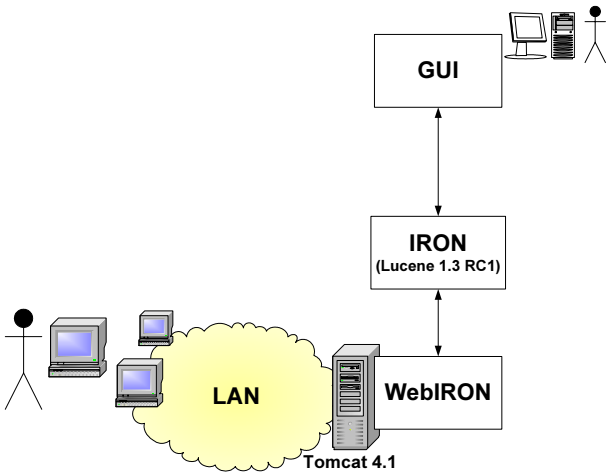

Figure 6: Architecture of IRON.

As shown in figure 6, IRON functionalities can be accessed in two different ways: last year a GUI (Graphical User Interface) was developed so that IRON could be used as a stand-alone application. This year a completely new Java Servlet based Web interface was developed, which is called WebIRON. WebIRON is based on the Tomcat[4] 4.1 Web server and makes IRON a Web application and it provides a set of wizards which help the user to set all the parameters and choose the IR components, which are needed in order to conduct a run or, more generally, an IR experiment.

IRON and WebIRON are written in Java, and hence they can be run over multi-platforms; we have chosen to run them on a personal computer equipped with Red Hat Linux operating system and on another personal computer equipped Windows XP operating system, in order to distribute the work load necessary for conducting all the runs.

[1] `http://jakarta.apache.org/lucene/docs/index.html`
[2] `http://www.snowball.tartarus.org`
[3] `http://www.jflex.de`
[4] `http://jakarta.apache.org/tomcat/index.html`

4.1.1 Re-engineering the Lexer

In this section we briefly present the CLEF2003 Lexer that was used for this evaluation campaign.

Just a few words about what a lexical analyzer is. A lexer is a program that analyzes a stream of characters and groups it into word-like units usually called tokens (or lexemes). The tokens are defined by regular expressions that are matched by the program according to some rule. Usually, the output of the 'lexer' is processed by some other programs (such as a parser). In our case, the output is properly formatted in order to be fed as input to Lucene.

These complete re-engineering of the lexer has been due mainly for the following reasons:

- there are a lot of collections formatted in different ways;
- the languages are different, this would possibly means that we need to manage some particular characters, or want the lexer to behave differently with respect to the language;
- the volume of the collections require an efficient lexer

Moreover, we would like the CLEF2003 lexer to be transparent with respect to the collection we are processing, that is, we just give the lexer a collection without being necessary any additional information.

Our final aims were:

1. Write a lexical analyzer able to process documents of the CLEF collections, written in different European languages;
2. Make this property transparent, that is the lexer should be able to recognize by itself what collection and language it is analyzing;
3. Make the lexer fast and efficient, at least faster than the one used in the CLEF02 campaign.

A possible solution of how to recognize the collection stays within the files of the collections. Here, it follows an example of a file:

```
<DOC>
<DOCNO>GH950103-000000</DOCNO>
<DOCID>GH950103-000000</DOCID> <DATE>950103</DATE>
<HEADLINE>Chance of being a victim of crime ... </HEADLINE>
<EDITION>3</EDITION>
<PAGE>3</PAGE>
<RECORDNO>980549733</RECORDNO>
<TEXT> PEOPLE greatly overestimate the likelihood of being a
victim of crime, according to a survey out today. On average
...
</TEXT>
</DOC>
<DOC>
<DOCNO>GH950103-000001</DOCNO>
<DOCID>GH950103-000001</DOCID>
...
```

Have a look at the first line after the tag <DOC>

```
<DOCNO>GH950103-000000</DOCNO>
```

Each document has a unique identifier <DOCNO>, and the prefix of this identifier (the letters behind the digits, in the example above the string 'GH') is unique for each collection. In particular, the unique codes for the collection we were asked to use in CLEF03 were:

- Dutch: AD, NH

- English: GH, LA
- French: ATS, LEMONDE
- German: FR, SPIEGEL, SDA
- Italian: AGZ, LASTAMPA
- Spanish: EFE

At this point we are able to answer to our first and second aim: to design a transparent lexer able to recognize different collections (and languages as well). We use these unique codes as regular expression that 'address' the lexer to the specific sub-part able to process a particular collection (see Figure 7).

It is not our intention here to give all the details about how to build the complete lexer, nevertheless, we want to remind that once a collection-state has been reached by the lexer we need to extract the information that we want to index. The Guidelines of CLEF define clearly what are, for each collection, the part of the documents (enclosed by some distinctive tags like <TEXT>, </TEXT>) that are allowed to be used for the indexing, so as for the retrieval.

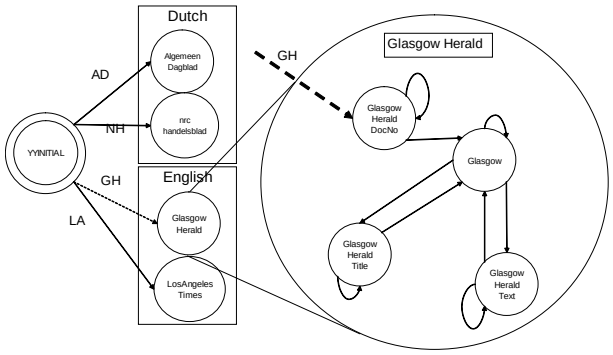

Figure 7: (Right) An example of part of the DFA CLEF2003. (Left) A close-up on the lexer for the Glasgow Herald Collection.

One last remark about the efficiency of the lexer. In fact, only using JFlex, the lexer did not perform as well as we expected. We found that most of the inefficiency stayed in the String management of Java when using the String Class. A better solution solution was found using the StringBuffer Class. A StringBuffer implements a mutable sequence of characters, it is like a String, but can be modified. At any point in time it contains some particular sequence of characters, but the length and content of the sequence can be changed through certain method calls (StringBuffer it behaves somewhat similarly to an ArrayList).

A more detailed description of this implementation can be found at [4].

4.2 Runs

We participated at the following monolingual tracks: Dutch, French, German, Italian and Spanish. For each tracks we tested four different stemming algorithms:

- **No Stem**: no stemming algorithm was applied;
- **Porter**: we used the stemming algorithms for different languages, which are freely available in the Snowball Web site, edited by M. Porter;
- **SPLIT**: we implemented our stemming algorithm based on the notion of mutual reinforcement;

- **STON**: we implemented our stemming algorithm based on Hidden Markov models.

As regards the stop-words used in the experiments, i.e. the words which have little semantic meaning, we chose stop-lists which are available at `http://www.unine.ch/info/clef/`. This stop-list are recommended by CLEF consortium for the participants of the CLEF campaigns.

4.3 A Global Evaluation

For each language, we carried out a macro evaluation by summarizing the result over all the queries of the test collection.

Table 1 compares the number of relevant retrieved documents and the recall for the different algorithms under examination.

considered stemming algorithms.

Algorithm	**Relevant Retrieved (Recall %)**				
	Dutch	**French**	**German**	**Italian**	**Spanish**
No Stem	1,419 (89.98)	869 (91.86)	1,330 (72.88)	488 (60.32)	2,084 (88.01)
SPLIT	1,420 (90.04)	886 (93.66)	1,376 (75.40)	497 (61.43)	2,122 (89.61)
STON	1,386 (87.33)	891 (94.19)	1,384 (75.84)	503 (62.18)	2,148 (90.71)
Porter	1,416 (89.79)	911 (96.30)	1,434 (78.58)	492 (60.82)	2,202 (92.99)
Total Relevant Docs	1,577	946	1,825	809	2,368

Table 1: Relevant retrieved document number (recall) for 2003 Topics.

Note that, in general, stemming improves the recall, since the number of relevant retrieved documents is larger than the number of relevant retrieved documents observed in the case of retrieval without any stemmer. Dutch represents an exception to this note, since both the STON and the Porter stemmers retrieve less relevant documents than without any stemmer.

Table 2 reports, for each language, the average precision attained by the system with the considered stemming algorithms.

Algorithm	**Average Precision (%)**				
	Dutch	**French**	**German**	**Italian**	**Spanish**
No Stem	42.11	42.86	34.92	34.76	39.27
SPLIT	42.84	45.60	37.11	38.17	38.25
STON	42.57	45.67	36.68	34.66	40.56
Porter	43.49	45.87	37.88	35.53	43.42

Table 2: Average Precision for 2003 Topics.

Table 3 reports, for each language, the exact R-precision attained by the system, which is the precision after R documents have been retrieved, where R is the number of relevant documents for the topic.

Algorithm	**Exact R-Precision (%)**				
	Dutch	**French**	**German**	**Italian**	**Spanish**
No Stem	40.51	39.45	36.59	36.32	40.26
SPLIT	41.54	43.22	37.80	38.39	39.85
STON	39.66	42.20	37.53	33.26	39.90
Porter	40.55	41.68	38.73	34.79	42.70

Table 3: Exact R-Precision for 2003 Topics

Note that for French, German, Italian and Spanish stemming positively affects the precision, thus improving the overall performance of system; Dutch stemming does not hurt the overall

performances of the system. Furthermore, when the stemming algorithms positively affect the performances, SPLIT and STON perform as effectively as the Porter's stemmer. This experimental evidence confirms the hypothesis that it is possible to generate stemmers using probabilistic models without or little knowledge about the language. We are going to carry out a statistical analysis to assess the significance of the differences between the levels of precision.

5 Conclusions and future work

This year the IMS research group has carried out many experiments and developed methodological results on automatic stemmer generation and query translation. The former has been tested within the mono-lingual track, whereas the latter has been tested within the bi-lingual track. The idea underlying both automatic stemmer generation and query translation has been the use of diverse probabilistic models. Whereas the probabilistic models for stemmer generation has confirmed the positive results observed last year, those employed for query translation need further experiments and refinement.

References

[1] M. Agosti, M. Bacchin, N. Ferro, and M. Melucci. Improving the Automatic Retrieval of Text Documents. In C. Peters, M. Braschler, J. Gonzalo, and M. Kluck, editors, *Evaluation of Cross–Language Information Retrieval Systems, Third Workshop of the Cross–Language Evaluation Forum, CLEF 2002, Revised Papers*. Springer, Berlin/Heidelberg, (in print), 2003.

[2] M. Bacchin, N. Ferro, and M. Melucci. The Effectiveness of a Graph-based Algorithm for Stemming. In E. P. Lim, S. Foo, C. S. G. Khoo, H. Chen, E. A. Fox, S. R. Urs, and C. Thanos, editors, *Digital Libraries: People, Knowledge, and Technology, Proc. 5th International Conference on Asian Digital Libraries, ICADL 2002*, pages 117–128. Lecture Notes in Computer Science (LNCS) 2555, Springer, Berlin/Heidelberg, 2002.

[3] L. Rabiner and B.H. Juang. *Fundamentals of speech recognition* Prentice Hall, Englewood Cliffs, NJ, 321–389, 1993.

[4] G.M. Di Nunzio. *The CLEF2003 Lexer.* Available at `http://www.dei.unipid.it/~ims`.