DNA AND NATURAL LANGUAGES
Text Mining

Gemma Bel-Enguix[1], Veronica Dahl[1,2] and M. Dolores Jiménez-López[1]
[1]*Rovira i Virgili University, Tarragona*
[2]*Simon Fraser University, Burnaby*

Keywords: Text mining, Constraint handling rules, Oligonucleotides, Natural language processing.

Abstract: We present, discuss and exemplify a fully implemented model of text mining that can be applied to spoken languages as well as to molecular biology languages. This is based in the model presented in (Zahariev et al., 2009) oriented to discovering DNA barcodes for sequences. The novelty of our methodology is the use of Constraint Based Reasoning to detect string repetitions through unification, by introducing a new general rule for matching. We claim that the same method can be succesfully applied to mining natural language texts.

1 INTRODUCTION

In this article we propose a model of text mining through constraint based reasoning that has application in two important types of natural languages: human languages per se, and the (also human albeit less overtly so) languages of molecular biology.

The model is based in the proposal by (Zahariev et al., 2009) who introduced an efficient new approach for the case of discovering DNA barcodes for sequences. DNA sequences consist in sentences formed from an alphabet of four ”words”, or oligonucleotides: A,C,T and G. This algorithm, unlike previous methods, neither necessitates a preliminary alignment, which would reduce its efficiency for intron-rich regions (i.e. regions which are not translated into protein), nor to resort to a brute-force approach, which would reduce efficiency as well, and even compromise feasibility. These methods, based on group oligonucleotide sorting, have been successfully used as part of a signature oligo microarray design process.

Therefore, the methodology is first conceived for mining molecular biology texts, but as we shall argue, in its high level incarnation here proposed, is also adequate for dealing with human languages.

We formulate our methodology in Sicstus Prologs Constraint Handling Rules, explaining it first by means of an example. We then show how it extends to ambiguous matching, and we test it as well for two other string mining applications which are frequent in DNA mining: finding a substrings frequency and finding gapped patterns. We end with a brief discussion of future work and extensions, in particular for mining human language texts.

Our focus at this point is expressiveness and elegance of formulation rather than efficiency, but our results are nevertheless surprisingly efficient considering the tasks at hand.

Section 2 briefly reviews the computational background needed to understand the implementation details of our model. Section 3 presents it through a toy example. Section 4 develops fully working solutions, in terms of our model, to three important classes of problems in text mining. Section 5 discusses the implications of Specialized Concept Formation for human language texts, and section 6 presents our conclusions. Complete running programs are given in the following web: www.geocities.com/CHRPrograms/SCF.html

2 COMPUTATIONAL PRELIMINARIES: CHR

As in (Dahl and Voll, 2004), we use *Constraint Handling Rules (CHR)* as our implementation methodology. CHR provide a simple bottom-up framework which has proved useful for algorithms dealing with constraints (Fruhwirth, 1993; Fruhwirth, 1998). Because logic terms are used, grammars can be described in human-like terms and are powerfully extended through (hidden) logical inference. The format of CHR rules is:

Bel-Enguix G., Dahl V. and Dolores Jimenez-lopez M. (2009).
DNA AND NATURAL LANGUAGES - Text Mining.
In *Proceedings of the International Conference on Knowledge Discovery and Information Retrieval*, pages 140-145
DOI: 10.5220/0002292201400145
Copyright © SciTePress

`Head ==>Guard|Body`

`Head` and `Body` are conjunctions of atoms and `Guard` is a test constructed from (Prolog) built-in or system-defined predicates. The variables in `Guard` and `Body` occur also in `Head`. If the `Guard` is the constant "true" (i.e., no tests need succeed in order for the rule to apply), then it is omitted together with the vertical bar. Its logical meaning is the formula $(Guard \rightarrow (Head \rightarrow Body))$ and the meaning of a program is given by conjunction. There are three types of CHR rules:

- *Propagation rules*, which add new constraints (body) to the constraint set.
- *Simplification rules*, which also add as new constraints those in the body, but remove as well the ones in the head of the rule.
- *Simpagation rules*, which combine propagation and simplification traits, and allow us to select which of the constraints mentioned in the head of the rule should remain and which should be removed from the constraint set.

The rewrite symbols for the first two rules are respectively: `==>`, `<=>` and for simgation rules, the notation is `Head1\Head2<=>body`. Anything in `Head1` remains in the constraint set and anything in `Head2` is removed from the constraint set.

3 OUR METHODOLOGY, EXPLAINED THROUGH AN EXAMPLE

3.1 Mining Human Languages

Let us consider a short sample problem for explanatory purposes: that of finding a string of words of any length which is common to three short sentences given as input. For instance, for the input corpus:

The drought of March has pierced to the root.

Alice has had enough of hares of March.

Waters of March was written by Jobim.

the output should include "of March" as one of the common sequences found, and we moreover want to know the position where the sequence starts within each sentence.

Our system's utilities first compile the sentences into Prolog definitions of each (named s1, s2, s3), done in terms of atoms of the form w(i,j,W), where i is the sentence number, j the word's position in that sentence, and W the word itself. The above given input, for instance, compiles into:

```
(1) s1:- w(1,1,the), w(1,2,drought), w(1,3,of),w(1,4,march),
         w(1,5,has), w(1,6,pierced), w(1,7,to), w(1,8,the),
         w(1,9,root).

(2) s2:- w(2,1,alice), w(2,2,has), w(2,3,had),w(2,4,enough),
         w(2,5,of), w(2,6,hares), w(2,7,of), w(2,8,march).

(3) s3:- w(3,1,waters), w(3,2,of), w(3,3,march), w(3,4,was),
         w(3,5,written), w(3,6,by), w(3,7,jobim).
```

If we now initialize the system by calling all three strings, i.e.:

```
(4)   ?- s1, s2, s3.
```

we are in a position to extract substrings from these sentences, through the following two propagation rules:

```
(5) w(Row,C,N), w(Row,C1,N1) ==> C1 is C+1 |
                                 sub([N,N1],Row,C).

(6) w(Row,C,N), sub(S,Row,C1) ==> C1 is C+1 |
                                  sub([N|S],Row,C).
```

Rule (5) detects two subsequent words in the same sentence, or row, and records them through a new constraint sub/3 in list form (in the first argument of sub/2), keeping as well, in its second argument, a record of the row (i.e., the sentence number) the substring was found in, and in its third argument, the column it starts at within that row. Rule (6) similarly identifies all other substrings in the input strings, by adding one more word at a time to an already found string.

Of course, for different problems we may specialize these rules further, so that they zoom onto some sufficient subset of the set of all substrings, e.g. on all those substrings of a given size.

We have now enough utilities for the first incarnation of our Power matching rule, which extracts a substring S that is common to all three strings, and records the position in each sentence where the substring appears:

```
(7) sub(S,1,C1), sub(S,2,C2), sub(S,3,C3) ==>
    common(S,[C1,C2,C3]).
```

This completes our formulation for this toy example. Among the results the system outputs, we have:

```
common([of,march],[3,5,2])
```

Notice that in their declarative reading, our system's rules form a specialized concept, such as that of a substring, or of a common string, and in their operational reading, they produce all instances of that concept with respect to given input. Thus our methods can be directly incorporated into the Cognitive Sciences theory of Concept Formation, which also uses CHR for its implementation (Dahl and Voll, 2004).

3.2 Mining Molecular Biology Text

The same methodology can be directly used for mining sequences of nucleotides given as input, without touching the system itself. All we need to do is change the input so that the compiler will treat strings of nucleotides rather than strings of words, e.g. from the three sequences of nucleotides:

```
c a t g g c a a
t g g c a c t g
a c g t g g c a
```

the compiler will obtain:

```
(1′) s1:-w(1,1,c), w(1,2,a), w(1,3,t), w(1,4,g), w(1,5,g),
         w(1,6,c), w(1,7,a), w(1,8,a).

(2′) s2:- w(2,1,t), w(2,2,g), w(2,3,g), w(2,4,c), w(2,5,a),
         w(2,6,c), w(2,7,t), w(2,8,g).

(3′) s3:- w(3,1,a), w(3,2,c), w(3,3,g), w(3,4,t), w(3,5,g),
         w(3,6,g), w(3,7,c), w(3,8,a).
```

The system is then run by calling all input strings, as before, through rule(1), which will result in the output:

```
common([t,g,g,c,a],[3,1,4])
```

being generated among others, indicating that t g g c a is a common substring, and that its start position in strings s1, s2 and s3 is respectively 3, 1 and 4. The complete output is shown in Appendix I at http://www.geocities.com/ CHRPrograms/SCF.html.

So far we've only considered identical subsequences, i.e. there are no ambiguous elements in the vocabulary. Our formulation however has been designed to accommodate ambiguous input with minimum extra apparatus and computational overhead, as we discuss in section 4.1.

3.3 Efficiency Considerations

Our core rule for finding common substrings in a sequence of strings is computationally intensive in the case of molecular biology applications because we must actually examine each sequence entirely, drawing subsequences of different lengths from each, before our core rule discovers through unification which substrings are common to all strings given. Even in these applications, however, there are subproblems where the search space can be reduced, for instance it is not uncommon to look for common substrings of a given length, or of a maximum given length. Thus our approach could be modified in these cases in order to take advantage of the smaller search space (by only looking for common substrings of length L where L is known).

With human language texts, however, the search space can be greatly reduced. For instance, imagine that instead of having to find arbitrary substrings of arbitrary lengths as we did above, we are given a known sequence of words and all we have to do is check whether they show up in every string. This would be useful for instance in *automatic authorship attribution and genre classification* (Stamatatos et al., 2000) where the use of certain subphrases, word frequencies, word length and sentence length can be calculated for specific authors or genres and used to prove or disprove authorship of texts. It could be useful also to *determine the age of a manuscript*, e.g. by chequing how frequently a series of words which might be in disuse in our times appears in a text presumed to be of a certain age.

4 THREE SPECIAL CASES OF STRING ANALYSIS

4.1 Ambiguous Matching

Whereas the basic nucleotide set consists of the nucleotides A,C,T,G, ambiguity (where a given string's position can take one value or another) is typically expressed by using extra names for the ambiguous nucleotides, so for instance a nucleotide denoted as R can materialize as either A or G.

Ambiguous matching usually introduces considerable extra work, both in terms of representing ambiguous strings, and of processing them. Representation wise, it is combinatorially explosive to explicitly construct all alternative strings, one with each possible value of the ambiguous nucleotides. The alternative of compacting the representations usually complicates their processing, by having to unfold them at runtime. Specific procedures might be needed as well in order to, for instance, explicitly block any proposed solutions in which the ambiguous nucleotides are not compatible with their counterparts in other input sequences among the comparison set.

In contrast, all our formulation needs in order to represent and process any ambiguous nucleotide is for the compiler to materialize all its incarnations locally when the ambiguous string is read in. For instance, a nucleotide of type R appearing in the third sequence, column 7, which following our notation will be input as as n(3,7,r), compiles into the two nucleotides n(3,7,a) and n(3,7,c). Non-ambiguous nucleotides in the same sequence remain represented as before, so that complexity-wise, the representation grows only linearly with respect to the number of ambiguous nu-

cleotides. In order to process ambiguous strings, once we have compiled them as just described, no further modifications are needed to our system: it runs as is. No specific blocking of potential solutions that are not compatible is needed: our Power Matching rule ensures that they will simply not be generated, thus ensuring both elegance and efficiency. The only modification needed to transform our previous example into one exhibiting an R ambiguity at position 7 of string 3 is the replacement of (3') by:

```
s3:- n(3,1,a), n(3,2,c), n(3,3,g), n(3,4,t), n(3,5,g),
     n(3,6,g),n(3,7,a), n(3,7,c), n(3,8,a).
```

This alghorithm is more efficient than the one presented in (Zahariev et al., 2009), because these methods are quite intricate programming-wise, and must be complemented with further work in the case of ambiguous matching, since sets of sequences where at least one sequence contains at least one occurrence of an ambiguous nucleotide cannot be sorted. Other methods in the literature resort to probabilistic analysis (Manning and Schutze, 1999; Mikheev, 2003).

In our methodology, the ambiguous case, which posed considerable problems in the previous work, is directly solved, as we have seen, as a side effect of the formulation chosen. In addition, we share with (Zahariev et al., 2009) the desirable feature of needing neither preliminary alignment nor probabilistic analysis. To the best of our knowledge, this is the first time such an approach has been proposed and explored.

4.2 Finding a Substring's Frequency

In cryptanalysis (Becket, 1988; Menezes et al., 1996), frequency analysis has been defined as the study of the frequency of letters or groups of letters in a ciphertext. Frequency analysis is based on the fact that, in any given stretch of written language, certain letters and combinations of letters occur with varying frequencies. It is clear that the methodology presented here can be used as a tool to identify the common combinations of letters and to assign them frequencies. In this section we exemplify with DNA strings, but as before, the same methodology is applicable to linguistic texts.

In molecular biology, finding a substrings' frequency is an interesting task that can help, among others, to find DNA words. Those sequences more frequently repeated have a high probability of being meaningful in the genetic code (Basu et al., 2003).

For approaching this problem, we now modify our input to consist of just one sequence (which results in binary atoms compiling from the input, since we no longer need to record the sequence, or row, number), we introduce a parameter N into the call, which now becomes go(N), with N being the length of the subsequences sought, and we calculate subsequences of that length. The Power Matching rule now becomes, for Max being the length of the substrings whose occurrences we want to count:

```
(8) n(C,N), sub(S,C1,L,Max) ==>
                  L < Max, L1 is L+1, C1 is C+1 |
                  sub([N|S],C,L1,Max).

(9) sub(S,C1,Max,Max), sub(S,C2,Max,Max) ==>
                  dif(C1,C2) |
                  repeated(S,[C1,C2]).

(10) sub(S,C1,Max,Max) \  repeated(S,Where) <=>
                  notin(C1,Where) |
                  repeated(S,[C1|Where]).
```

This rendition of the Power Matching schema illustrates matching an unknown number of string occurrences. Rule (8) creates substrings of increasing length up to the maximum, rule (9) detects two equal such substrings, starting respectively in positions C1 and C2, and after checking that these two positions are different, records the fact that the string S appears in both those positions. Rule (10) finds one more occurrence of the same string, and updates the information accordingly, adding the new position in the list of positions where the string repeats.

Appendix II at http://www.geocities.com/CHRPrograms/SCF.html shows the complete program, including the definition of auxiliary predicates called above, and also shows the results of searching for repeated occurrences of strings of length 2, 3 and 4 within the input sequence c a t g g c a a t g g c a c t g a c g t g g a c a .

Here again, adapting our system to human language applications only involves a change of input: the rest of the system remains as is.

4.3 Finding Gapped Patterns

Because of the existence of introns and junk, in some molecular biology contexts it is reasonable to search for patterns that repeat in different sequences, even though they may be interrupted by an arbitrary number of words (Parida, 2007). Several systems exist and are available online to find these gapped patterns in molecular biology, like MOTIF (http://motif.stanford.edu/) or TEIRESIAS (http://www.research.ibm.com/bioinformatics/home.html). Finding the maximal (gapped) patterns in a text (phrases with discontinuities), combined with the study of frecuencies, can easily help to text summarization and text classification. Our methodology can also be adapted to finding maximal gapped pat-

terns, by keeping not only the start point but also the end point of the subsequences found, and using equations on them in this version of the matching rule (see Appendix III at http://www.geocities.com/ CHRPrograms/SCF.html).

For instance, for the input:

```
s1:- w(1,1,the), w(1,2,big), w(1,3,wolf).
s2:- w(2,1,the), w(2,2,big), w(2,3,bad), w(2,4,wolf).
s3:- w(3,1,the), w(3,2,big), w(3,3,ugly), w(3,4,silly),
     w(3,5,wolf).
```

our system produces as output:

```
pattern([[the,big],_B,[wolf]])
```

5 FURTHER IMPLICATIONS OF OUR RESEARCH FOR MINING HUMAN LANGUAGE TEXTS

Topics such as information retrieval, text summarization, text categorization, sentence extraction and even frequency analysis in cryptography can benefit from the methodology introduced in this paper. All these issues rely in the identification of some significant segments in a given text, with no previous information about these strings. Our core rule for finding common substrings in a sequence of strings, as our examples have shown, is very versatile and thus eminently suitable e.g. for the rapid prototyping and experimentation typically needed to test and fine tune linguistic theory. Other uses in specific tasks include the following.

The major task in *information retrieval* (Manning et al., 2008) is to find relevant documents for a given query. In order to find such documents it is important to consider the presence of some relevant words related to the topic of the query. Taking into account this fact, our methodology could look for the key words in a given amount of texts, providing a set of documents that contain common substrings that match the user's query.

Text summarization (Mani and Maybury, 1999) addresses both the problem of selecting the most important portions of text and the problem of generating coherent summaries. The goal in a summarization system is to extract the most informative sentences that summarize the whole document, so features like the number of named entities in the sentence or the rank of the sentence in the document are very useful. Therefore, in text summarization we have to identify the most important portions of the text which will be topically most salient. The method presented here could help in this task by first identifying maximal common patterns (maybe with discontinuities) and then filtering the patterns in order to obtain the main term that will help to summarize the text. For example, if in our text we find sequences like "the big wolf", "the bad wolf", "the ugly wolf", after the filtering process we could get just the sequence "the wolf" that will give us the main topic of the text helping, therefore, in the task of summarizing it.

Text categorization or *text classification* (Forsyth, 1999) is the task of automatically sorting a set of documents into categories, classes or topics from a predefined set. By using our methodology we are able to identify the common strings in a given text and to assign them a frequency. This idea could help in the task of categorizing a text, because the most frequent words identified in the text could determine the main topic of the document and, therefore, give us a key for choosing the category or class of the text.

Information distillation (Hakkani-Tur and Tur, 2007) aims to extract the most useful pieces of information related to a given query from massive textual document sources. One critical component for distillation is detecting sentences to be extracted from each relevant document. The goal of sentence extraction is to tag each sentence as relevant or not given a set of documents relevant to a distillation query. Again here, our methodology could help in the task of extracting relevant sentences.

A certain type of natural language disambiguation allows us to identify patterns where a word or group of words are interchangeable (Banko and Brill, 2001; Ginter et al., 2004). This is very useful for second language acquisition drills, in which a student is given for instance the pattern "I will look ... up" and must fill in the slot. The slot is ambiguous regarding which word can fit in, except for the requirement that it be the appropriate type of pronoun (i.e., either "you", "him", "her",...). We can exploit this symmetry by adapting our program into such language acquisition drills.

Similarly, we can extend the same program to admit extensible slots, as in "I will look John up", "I will look the boy up", "I will look the man with the yellow hat up", and so on.

The topics listed above could be potential natural language applications of our methodology. The simplicity and high level of the method, the fact that it is not necessary to know in advance what we are searching for, and the way it works by being able to find patterns with different length but with basically the same main structure, makes of the model introduced here a good candidate to simplify many of the tasks related to the processing of natural language texts.

For pattern or word frequencies, simple programs can be designed with a high performance. The code

introduced in section 4.2 and shown in Appendix II at http://www.geocities.com/CHRPrograms/SCF.html, is able to find repeated sequences in a string- whether of DNA or of human language text- and their collocation, from which the number of occurrences can be deduced.

6 CONCLUDING REMARKS

This paper had a double goal: a) to improve some solutions for DNA mining, using a high level programming language, and b) to extend these solutions to text mining. Therefore, we have shown how several problems that were hard to solve with other algorithms become simple to tackle with our Parallel Matching methodology. Later, we have tested the suitability of these same techniques in natural language processing. The first results, that we are just introducing here, suggest this is a promising approach to text mining and processing, with some important features:

- the programs do not need to know anything about the data they must process or search;
- the use of statistics can be minimized in future applications based on this approach;
- no previous alignments are needed in future DNA applications based on this approach;
- and simple programs can achieve high results.

As we have highlighted in the first section, our focus at this point is expressiveness and elegance of formulation rather than efficiency. With this initial incursion into our methodology and its applications, we hope to motivate further research on its suitability for many other different kinds of problems in string analysis.

ACKNOWLEDGEMENTS

We are grateful to Agostino Dovier and Andre Levesque for useful comments on a first draft of this paper.

REFERENCES

Banko, M. and Brill, E. (2001). Scaling to very very large corpora for natural language disambiguation. In *ACL '01: Proceedings of the 39th Annual Meeting on Association for Computational Linguistics*, pages 26–33. Morristown, NJ, Association for Computational Linguistics.

Basu, S., Burma, D., and Chaudhuri, P. (2003). Words in dna sequences: Some case studies based on their frequency statistics. *Journal of Mathematical Biology*.

Becket, B. (1988). *Introduction to Cryptology*. Blackwell.

Dahl, V. and Voll, K. (2004). Concept formation rules: an executable cognitive model of knowledge construction. In *proceedings of* First International Workshop on Natural Language Understanding and Cognitive Sciences. INSTICC Press.

Forsyth, R. (1999). *New Directions in Text Categorization*, pages 151–185. Springer, Berlin.

Fruhwirth, T. (1993). User-defined constraint handling. In *ICLP 93*, Budapest. MIT Press.

Fruhwirth, T. (1998). Theory and practice of constraint handling rules. *Journal of Logic Programming. Special Issue on Constraint Logic Programming*, (37(1-3)):95–138.

Ginter, F., Boberg, J., Jarvinen, J., and Salakoski, T. (2004). New techniques for disambiguation in natural language and their application to biological text. *Journal of Machine Learning Research*, (5):605–621.

Hakkani-Tur, D. and Tur, G. (2007). Statistical sentence extraction for information distillation. In *Acoustics, Speech and Signal Processing. ICASSP 2007, IEEE International Conference*, volume vol. 4.

Mani, I. and Maybury, M. (1999). *Advances in Automatic Text Summarization*. MIT Press, Cambridge.

Manning, C., Raghavan, P., and Schutze, H. (2008). *Introduction to Information Retrieval*. Cambridge University Press.

Manning, C. and Schutze, H. (1999). *Foundations of Statistical Natural Language Processing*. MIT Press, Cambridge.

Menezes, A., Oorschot, P., and Vanstone, S. (1996). *Handbook of Applied Cryptography*. CRC Press.

Mikheev, A. (2003). *Text Segmentation*. Oxford, Oxford University Publications.

Parida, L. (2007). *Pattern Discovery in Bioinformatics: Theory and Algorithms*. Chapman & Hall/CRC.

Stamatatos, E., Fakotakis, N., and Kokkinakis, G. (2000). Automatic text categorization in terms of genre and author. *Computational Linguistics*, (26(4)):471–495.

Zahariev, M., Dahl, V., Chen, W., and Levesque, A. (2009). Efficient algorithms for the discovery of dna oligonucleotide barcodes for dna sequences and groups of sequences.