32nd Annual International Conference of the IEEE EMBS
Buenos Aires, Argentina, August 31 - September 4, 2010

A Knowledge Engineering Approach to Recognizing and Extracting Sequences of Nucleic Acids from Scientific Literature

Miguel García-Remesal, Víctor Maojo and José Crespo

***Abstract*—In this paper we present a knowledge engineering approach to automatically recognize and extract genetic sequences from scientific articles. To carry out this task, we use a preliminary recognizer based on a finite state machine to extract all candidate DNA/RNA sequences. The latter are then fed into a knowledge-based system that automatically discards false positives and refines noisy and incorrectly merged sequences. We created the knowledge base by manually analyzing different manuscripts containing genetic sequences. Our approach was evaluated using a test set of 211 full-text articles in PDF format containing 3134 genetic sequences. For such set, we achieved 87.76% precision and 97.70% recall respectively. This method can facilitate different research tasks. These include text mining, information extraction, and information retrieval research dealing with large collections of documents containing genetic sequences.**

I. Introduction

In this article, we present a knowledge engineering approach to automatically extract genetic sequences from scientific papers for biomedical text mining, information extraction, and information retrieval applications. These include, for instance, automatically annotating, indexing and retrieving papers based on the occurrence of certain Polymerase Chain Reaction (PCR) primers or probes, or discovering relationships between the different entities appearing in the texts—e.g. organisms, genes, etc.—and the target sequences.

Sequences of nucleic acids occurring in papers are strings composed of symbols from an alphabet $\mathbf{\Sigma}$ of 16 standard uppercase symbols and their lowercase counterparts provided by the International Union for Pure and Applied Chemistry (IUPAC). These include the standard nucleotide symbols A(denine), C(ytosine), G(uanine), T(hymine) and U(racil), plus the wildcards B, D, H, K, M, N, R, S, V, W and Y. The latter represent a single valid choice among two or more nucleotide symbols [1]. From now onwards we will denote with $\mathbf{\Sigma}^+$ the set of all different strings composed of 1 or more symbols belonging to $\mathbf{\Sigma}$. Apart from these symbols, sequences that appear as free text can also include blanks and dashes and may span multiple lines.

Automatically recognizing the sequences appearing in the manuscripts requires identifying many different English words—or sequences of words—belonging to $\mathbf{\Sigma}^+$ that can be incorrectly recognized as valid sequences by an automated detector. For instance, all characters in the string "standard assay" are actually symbols from $\mathbf{\Sigma}$. Thus, it is difficult for regular expression-based detectors to determine whether the former string is a valid sequence. Other problems include the successful recognition of two or more different sequences separated in the text by words belonging to $\mathbf{\Sigma}^+$, such as for instance "and", or properly detecting the beginning or the ending of a sequence, such as e.g. "**as** Gttca accGt Gtta" or "ACG TTG ACCGT-**TAMRA-T**" where the bold characters do not belong to the actual sequences.

To address these issues, we followed a knowledge engineering approach to design and create a rule-based system to assist a finite state machine in properly detecting and extracting genetic sequences avoiding the abovementioned problems.

This paper is organized as follows. Section II describes the methods we used to create the scanner and the knowledge-based system for sequence refinement. Section III presents the results of the evaluation of our approach using a test set of 211 papers containing 3134 genetic sequences. Section IV discusses the results of the evaluation and compares our method to other existing approaches. Finally, section V draws the conclusions.

II. Methods

A. Preliminary Recognition of Sequences of Nucleic Acids

We created a lexical scanner based on a finite state machine [2] to recognize all *potential* sequences of nucleic acids appearing in the text. We consider as *potential* sequences those strings that are composed solely of substrings belonging to $\mathbf{\Sigma}^+$, blanks, dashes and newline characters. For each recognized sequence, the scanner returns a list containing all *relevant* tokens belonging to the recognized sequence. *Relevant* tokens include strings

Manuscript received March 19, 2010. This work has been partially funded by the European Commission through the ACTION-Grid support action (FP7-ICT-2007-2-224176) and the ACGT integrated project (FP6-2005-IST-026996), the Spanish Ministry of Science and Innovation through the OntoMineBase project (ref. TSI2006-13021-C02-01), the ImGraSec project (ref. TIN2007-61768), FIS/AES PS09/00069 and COMBIOMED-RETICS, and the Comunidad de Madrid, Spain.

M. García-Remesal is with the Biomedical Informatics Group and the Dept. Inteligencia Artificial, Universidad Politécnica de Madrid, Campus de Montegancedo S/N, 28660 Boadilla del Monte, Madrid, Spain. (phone: (+34) 91 336 7441; fax: (+34) 91 352 4819; e-mail: mgarcia@infomed.dia.fi.upm.es)

V. Maojo is with the Biomedical Informatics Group and the Dept. Inteligencia Artificial, Universidad Politécnica de Madrid, Campus de Montegancedo S/N, 28660 Boadilla del Monte, Madrid, Spain (e-mail: vmaojo@infomed.dia.fi.upm.es).

J. Crespo is with the Biomedical Informatics Group and the Dept. Lenguajes, Sistemas Informáticos e Ingeniería del Software, Universidad Politécnica de Madrid, Campus de Montegancedo S/N, 28660 Boadilla del Monte, Madrid, Spain (email: jcrespo@infomed.dia.fi.upm.es)

belonging to $\mathbf{\Sigma}^+$ alone. Blanks, dashes and newline characters are discarded since they are no longer needed. For instance, for the string "ACG-CCG-A\nCC-GAC-GTA-CCG-TAC", the recognizer would produce the following list of tokens: {"ACG", "CCG", "A", "CC", "GAC", "GTA", "CCG", "TAC"}. The latter represents the genetic sequence "ACGCCGACCGACGTACCGTAC", which will be generated during the refinement and post-processing of the extracted list of tokens. However, due to the simplicity of the scanner, some of these lists may contain false positives and noisy or incorrectly merged sequences. The refinement and post-processing activity described next aims to automatically deal with these issues.

B. Automated Refinement of Extracted Sequences

Once all *potential* sequences occurring in a single manuscript have been recognized, we then proceed to the automated refinement task. The latter aims to: (1) discarding sequences whose length is shorter than a predetermined threshold, (2) discarding false positives—i.e. *potential* sequences that are not actual sequences—such as e.g. {"standard", "assay"}, (3) refining noisy sequences—i.e. those that were incorrectly appended by the recognizer one or more words from $\mathbf{\Sigma}^+$ not belonging to the actual sequence—e.g. the sequence occurring within the string "GCTACCGT-TAMRA-T" should be recognized as {"GCTACCGT"} rather than {"GCTACCGT", "TAMRA", "T"} despite both "TAMRA" and "T" being elements from $\mathbf{\Sigma}^+$ and (4) separate incorrectly merged sequences such as those contained in the string "ACC GCT and GGCTA-GCTA-ACGT"—i.e. the latter sentence should generate the sequences {"ACC", "GCT"}, {"GGCTA", "GCTA", "ACGT"} instead of the single sequence {ACC", "GCT", "and", "GGCTA", "GCTA", "ACGT"}.

The refinement task is performed using a rule-based expert system manually created by the authors. The complete knowledge base (KB) is summarized in Table I. As shown in the table, each *potential* sequence *s* is represented as a sorted list tokens $\{s_1,\ldots, s_n\}$. On the other hand, the actions *add*(*s*) and *discard*(*s*) assert and retract a sequence *s* from the facts base respectively. As shown in Table I, the KB is composed of 8 rules. A detailed description of each rule follows.

R1 is aimed to discard those sequences whose length is smaller than L_{min} symbols. We set this parameter to 7 symbols. R1 resorts to the function *length*(s_i), that returns the size—i.e. number of symbols—of the token s_i.

R2 and R3 are targeted to refine noisy sequences that include problem words from $\mathbf{\Sigma}^+$ at the tail (R2) or the head (R3) of *s*. The function *list_item_in_sequence_tail*(*s*) (and *list_item_in_sequence_head*(*s*)) attempts to match all elements from a list of problem affixes at the tail (head) of *s*. If there are one or more matches then the function returns the position of the first (last) token of the longest matched element. For instance, for the sequence of tokens {"ACG", "CCG", "A", "CC", "GCA", "GTA", "CCG", "TACC", "TAMRA", "T"}, the function would match the subsequence {"TAMRA", "T"} at the tail of the list, thus exiting with return value 9, corresponding to the position of the token "TAMRA". The list of problem affixes is available under request to the corresponding author.

R4 is aimed to discard false positives such as e.g. {"standard", "assay"}. This is achieved by resorting to the function *in_dictionary*(*s*) that returns the value *true* iff all tokens belonging to *s* are included in a dictionary of English words belonging to $\mathbf{\Sigma}^+$ created by the authors. The dictionary is also available under request to the corresponding author.

R6 and R7 are similar to R2 and R3. The only difference is that R6 and R7 use the dictionary of English words—composed of single words rather than sequences of words—instead of the list of problem affixes to refine noisy sequences.

TABLE I
THE KNOWLEDGE BASE

R1	$\sum_{\{s_i \in s\}} \text{length}(s_i) < L_{min} \rightarrow \text{discard}(s)$
R2	$\exists i \mid i = \text{list\_item\_in\_sequence\_tail}(s) \rightarrow \text{discard}(s) \wedge \text{add}(s')$ $s' = \{s_1, \ldots, s_{i-1}\}$
R3	$\exists i \mid i = \text{list\_item\_in\_sequence\_head}(s) \rightarrow \text{discard}(s) \wedge \text{add}(s')$ $s' = \{s_{i+1}, \ldots, s_n\}$
R4	$\bigwedge_{\{s_i \in s\}} \text{in\_dictionary}(s_i) \rightarrow \text{discard}(s)$
R5	$\exists (i,j) \mid (i,j) = \text{list\_item\_within\_sequence}(s)$ $\rightarrow \text{discard}(s) \wedge \text{add}(s') \wedge \text{add}(s'')$ $s' = \{s_1, \ldots, s_{i-1}\},$ $s'' = \{s_{i+j}, \ldots, s_n\}$
R6	$\text{in\_dictionary}(s_1) \wedge \text{length}(s_1) \geq 3 \rightarrow \text{discard}(s) \wedge \text{add}(s')$ $s' = \{s_2, \ldots, s_n\}$
R7	$\text{in\_dictionary}(s_n) \wedge \text{length}(s_n) \geq 3 \rightarrow \text{discard}(s) \wedge \text{add}(s')$ $s' = \{s_1, \ldots, s_{n-1}\}$
R8	$\text{size}(s) \geq 2 \rightarrow \text{merge}(s)$

TABLE II
SOME EXAMPLES OF AUTOMATED SEQUENCE REFINEMENT

List of tokens	Execution Trace	Refined sequences
{"RNA"}	R1	-
{"ACGATG", "ACG", "and", "TGAGGACG", "TAMRA", "T", "and", "CGACGG", "CGAC"}	R5, R3, R5, R8, R8	{"ACGATGACG"}, {"TGAGGACG"}, {"CGACGGCGAC"}
{"standard", "DNA", "strand"}	R6, R7, R1	-
{"tga", "agc", "ttt", "TAMRA", "T"}	R2, R8	{"tgaagcttt"}
{"cats", "and", "rats"}	R4	-
{"tga", "cgg", "acc", "gta", "tta", "gcc"}	R8	{"tgacggaccgtattagcc"}

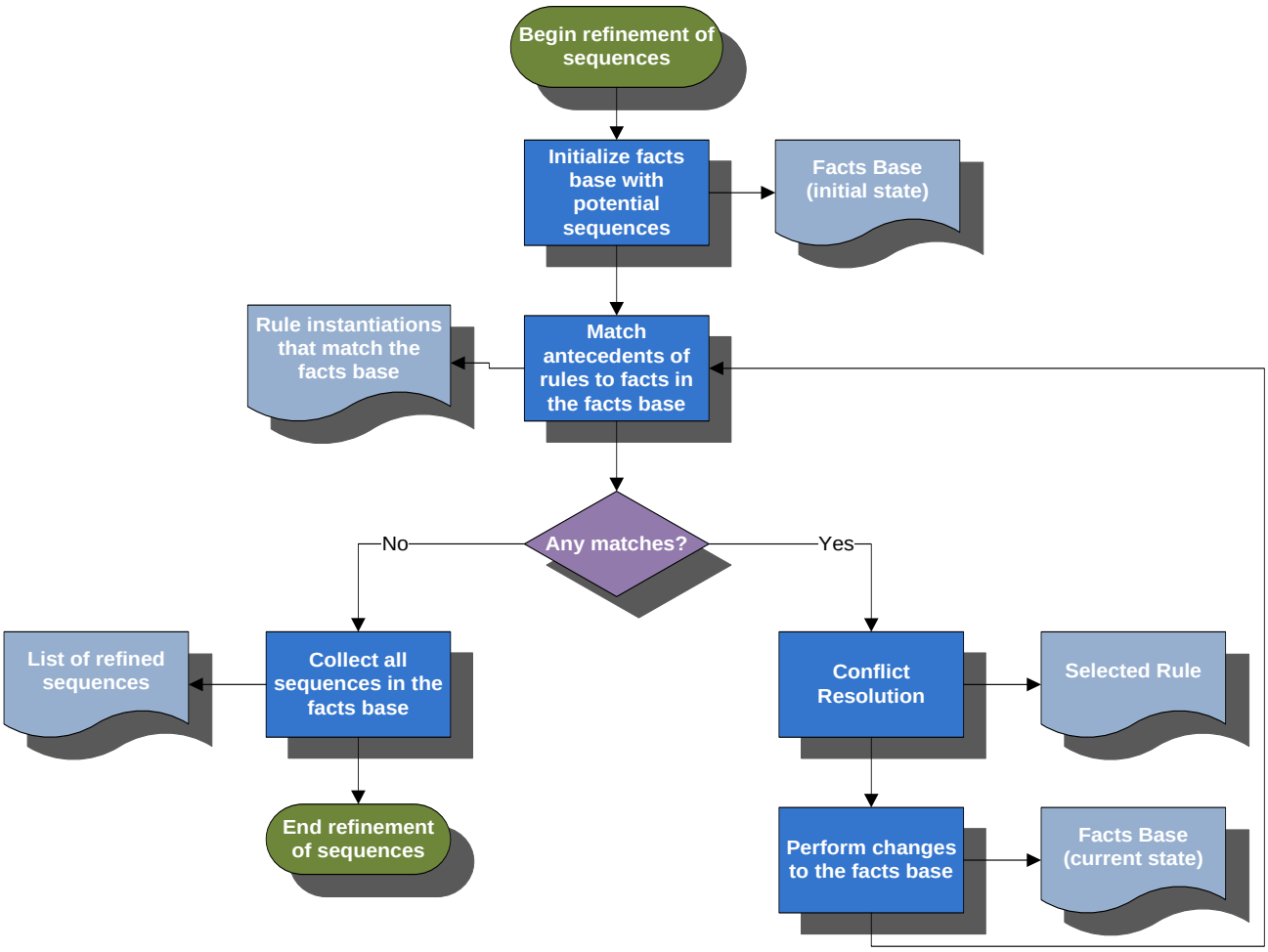

Fig. 1. Flowchart describing the sequence refinement process

On the other hand, R5 is targeted to separate two incorrectly merged sequences delimited by elements belonging to the list of problem affixes. This is achieved by using the function *list_item_within_sequence*(*s*) that attempts to match any element from the list of problem affixes to a subsequence of *s*—excluding the head and the tail of the list of tokens. If there are one or more matches then the function returns a tuple <*p*, *l*>, where *p* denotes the position of the first token of the longest matched element (in terms of the number of tokens), while *l* is the length (i.e. number of tokens) of the match. For instance, for *s* = {“CGT”, “TAMRA”, “T”, “and” “ACGT”}, the function would match the subsequences {“TAMRA”, “T”} and {“and”}, both belonging to the list of problem affixes. As the former is longer (2 tokens) than the latter (1 token), then the match {“TAMRA”, “T”} is selected. Therefore, the function returns the tuple <2, 2>, corresponding to the subsequence {“TAMRA”, “T”}.This rule can be recursively fired to successfully separate three or more incorrectly merged sequences.

Finally, the last rule (R8) concatenates all tokens belonging to the sequence *s* using the function *merge*(*s*)—e.g. for *s* = {“CGA”, “ACG”, “TTG”}, the function would return the sequence {“CGAACGTTG”}. The function *size*(*s*) returns the size—i.e. number of tokens—of the list *s*.

The refinement activity is performed as described in Figure 1. First, all potential sequences provided by the preliminary detector are added to the facts base. Then, the inference engine attempts to match the antecedent of all the rules to the elements in the facts base. Rules are fired by priority, being R1 (R8) the rule with highest (lowest) priority. As shown in the figure, firing a rule entails a change in the facts base. This procedure is iterated until no further rules can be fired. The final state of the facts base provides the set of refined sequences occurring in the manuscript. Table II shows a few examples of sequence refinement using the proposed approach. For further details on the structure and functioning of a rule-based expert system, see [3].

III. RESULTS

We implemented the scanner and the rule-based inference engine using the Java Programming Language.

Our approach was evaluated using a test set composed of 211 full-text manuscripts in Adobe’s Portable Document Format (PDF) containing 3134 sequences of nucleic acids. Each PDF article was first pre-processed using a PDF-to-text

conversion tool created by the authors based on the Apache PDFBox Library [4]. After the pre-processing, we fed the preliminary sequence detector with the text extracted from each paper. All the sequences recognized by the scanner were then forwarded to the rule-based system for refinement and post-processing. Sequences remaining in the facts base after the finalization of the automated refinement process were then compared to the actual sequences occurring in the document. We repeated this process for each document in the test set.

Table III shows the results of the evaluation in terms of precision/recall rates. Regarding performance issues, the software implementing our method—coded using the Java Programming Language and its associated technologies—required in average 75 ms to detect and refine all sequences occurring in a single paper. This timing result was obtained on a workstation using a 2.4 GHz Intel® Core ™ 2 Quad Q6600 processor with 2 GB physical memory.

TABLE III
SUMMARY OF RESULTS OF THE EVALUATION

No. of seqs. in the test set	Recognized (True Positives)	Not Recognized (False Negatives)	False Positives
3134	3062	72	427
Precision		87.76%	
Recall		97.70%	
F-Measure		0.9246	

IV. DISCUSSION

The results of the evaluation suggest that our method is suitable for automatically detecting, extracting and refining sequences of nucleic acids appearing in biomedical papers. Besides, the simplicity of the preliminary recognizer and the small size of the KB—8 rules—enable the proposed method to achieve high throughput rates. The approach is also flexible, since for refining sequences not currently being properly refined—often due to the occurrence of words from $\mathbf{\Sigma}^+$ currently missing from the list of problem affixes or from the dictionary—it will not normally be required to perform any modifications to the KB. Instead, the knowledge-based system can be adjusted by appending the required items to the list of problematic affixes or to the dictionary.

Regarding similar work, most major approaches for sequence alignment on biomedical unstructured text and databases concentrate on aligning sequences built upon the symbols A, C, G, U and T alone [5][6][7]—i.e. they do not consider any wildcards. Conversely, the Kangaroo system [8] is a web genomic pattern-matcher aimed to report back to users all GenBank [9][10] records that match a user query. The latter are regular expressions that may include not only the A, C, G, T and U symbols, but also the different wildcards. This approach, however, is different to ours since it is aimed to match short genetic sequences within a larger sequence retrieved from a structured database. Conversely, our method is aimed to recognize genetic sequences occurring in non-structured text. This is a more complex problem, since it requires discarding false positives, refining noisy sequences, etc.

V. CONCLUSION

In this paper we present a method for automatically detecting and extracting sequences of nucleic acids from scientific literature. Our approach can be adapted and extended to recognize other types of genetic sequences performing minor modifications to the scanner and the KB. The software implementing our method can be downloaded from http://gib.fi.upm.es/sites/default/files/PrimerXtractor-1.1-bin.rar

We believe that the proposed approach can facilitate different biomedical informatics research tasks, such as text mining or information indexing and retrieval activities involving large sets of documents containing biological sequences.

ACKNOWLEDGMENT

Authors would like to thank Alejandro Cuevas Candela for developing the software package that implements the method for DNA sequence recognition presented in this paper.

REFERENCES

[1] http://www.mun.ca/biochem/courses/3107/symbols.html. Last accessed March 10, 2010.
[2] M. Minsky, “Computation: finite and infinite machines”. Upper Saddle River, NJ: Prentice-Hall inc., 1967.
[3] P. Harmon, D. King, “Expert systems”. New York, NY: John Wiley & Sons, 1985.
[4] http://pdfbox.apache.org/. Last accessed March 10, 2010.
[5] H. Hyyrö, M. Juhola, M. Vihinen, “On exact string matching of unique oligonucleotides”. *Comput Biol Med*, vol. 35, no. 2, pp. 173-81, 2005.
[6] J. Tarhio, H. Peltola, “String matching in the DNA alphabet”. *Software Pract Exper*, vol. 27, no. 7, pp. 851-61, 1997.
[7] L.L. Cheng, D.W. Cheung, S.M. Yiu, “Approximate string matching in DNA sequences”, In *Proc of the 8th International Conference on Database Systems for Advanced Applications*, Kyoto (Japan), pp. 303-10, 2003.
[8] D. Betel, C.W.V. Hogue, “Kangaroo – a pattern matching program for biological sequences”. *BMC Bioinformatics*, vol. 3, pp. 20, 2002.
[9] D.A. Benson, I. Karsch-Mizrachi, D.J. Lipman, J. Ostell, E.W. Sayers, “GenBank”, *Nucleic Acids Res*, vol. 38, pp. D46-51, 2010.
[10] http://www.ncbi.nlm.nih.gov/Genbank/. Last accessed March 10, 2010.