Exploratory Analysis System for Semi-Structured Engineering Logs

Michael Flaster, Bruce Hillyer, Tin Kam Ho

Bell Laboratories, Lucent Technologies
700 Mountain Ave., Murray Hill, NJ 07974, USA
mflaster,hillyer,tkh@lucent.com

Abstract. Engineering diagnosis often involves analyzing complex records of system states printed to large, textual log files. Typically the logs are designed to accommodate the widest debugging needs without rigorous plans on formatting. As a result, critical quantities and flags are mixed with less important messages in a loose structure. Once the system is sealed, the log format is not changeable, causing great difficulties to the technicians who need to understand the event correlations. We describe a modular system for analyzing such logs where document analysis, report generation, and data exploration tools are factored into generic, reusable components and domain-dependent, isolated plug-ins. The system supports incremental, focused analysis of complicated symptoms with minimal programming effort and software installation. We discuss important concerns in the analysis of logs that sets it apart from understanding natural language text or rigorously structured computer programs. We highlight the research challenges that would guide the development of a deep analysis system for many kinds of semi-structured documents.

1 Introduction

Performance analysis for complex engineering systems often involves sifting through many large textual log files for a few rare target symptoms. Designed to serve as many diagnostic needs as possible, such logs typically include mixed printings of time stamps, numerical quantities, flags, and informal messages, in an order determined by the evolution of the system state. The logs can also be records of interrogations by an external observer, with variable sequences of queries representing particular monitoring needs.

For fear of regret on losing information, the design of such logs tends to be inclusive, which creates a cluttered background when one's interest is to detect a specific event. Moreover, the circumstances under which the logs are designed and studied are often less than ideal – there could be incomplete functionalities, chatter among different components, and non-deterministic occurrences of problems in a yet-to-be perfect product. All these contribute to the log documents having a loose structure despite that there is some intended, and still discernible, regularity.

Engineering logs are rarely customer-facing, and hence are unlikely to be rigorously designed and cleaned even after the system delivery. In many cases, the system is delivered as a sealed box and it is not possible to change the commands that generate and format such logs. Once the system is deployed, logs of specific events are not always

easy to regenerate because it may not be possible to reconstruct a particular system state.

Having to deal with such documents is an annoying aspect of many technicians' everyday life. While in principle one can always write specialized programs to search for the interesting phenomena within the logs, the engineers are often domain experts with little interest nor time for sophisticated search programming. Few can afford the luxury of having a dedicated debugging programmer in service. When the important symptoms involve joint occurrences of multiple flags and quantities that are scattered through the logs, or with specific sequential orders, the conditions could be very difficult to express even with complicated search programs. Worse, in highly complex systems with hundreds of critical parameters, events of interests sometimes cannot be articulated for tailored searching before they are recognized as a pattern. Discovery of correlations and causal relationships occurs more likely at the end instead of at the beginning of the debugging process. Thus an exploration tool that can operate with loosely structured documents is an essential aid to a diagnostic process.

Responding to this need, we propose a modular "log analysis" system that is designed to support complex event discovery within semi-structured documents. To address the problem in wide generality, we argue that the system should consist of separated lexical analysis, syntax-directed parsing, semantic translation and report generation, and exploratory analysis components. Wherever possible, generic tools should be used in each component, and the domain-dependent features should be minimized and contained within a set of plug-ins. In this paper we share our experiences in designing one such system that was motivated by real diagnostic scenarios in the telecommunication industry. Our goal is to call for attention to this ubiquitous challenge, and outline open questions for further research to improve such systems.

2 Rigidity of Textual Structures and Levels in Document Understanding

Semi-structured textual logs are predictable to an extent between those of natural language text (such as news stories) and artificial language text (such as computer programs). The technology of formal languages and program compilation is a well researched and mature field of study. Natural language understanding, while still difficult, has received much attention because of intense interests in internet searching. For the in-between continuum of structured or semi-structured text, however, research has been spotty.

Document understanding systems arising as a generalization of Optical Character Recognition systems [17] have had successes on documents with well-defined structure, such as mail addresses, forms, and tables [5][15][16].

These systems enjoy the advantage of having clear expectations of the document format, so that parsers can look for specific blocks, lines, or fields, though the expectations are not necessarily represented as a formal grammar. For these systems, emphasis is often put on the transcription and subsequent clean-up of the imaged text. Occasionally, statistical parsing, by way of computing the joint likelihood of each possible interpretation, is used to select among several syntactic alternatives [5]. For documents

without well defined syntax, attention tends to focus on the early stages of image processing, 2D layout analysis, and symbol (character/field/sentence) recognition. They often stop at obtaining logically ordered, tagged and transcribed text blocks.

Our concern is on methods to understand documents with contents that are very structured but there is no full grammar. Typically there are some expectations on the amount, type, and contents of the messages. But the underlying grammar is not available, or the text may not be produced by a single grammar. An example is a debugging trace generated by a program with several concurrent and interacting modules. Another example is a record of interrogation to several entities that produced indefinitely delayed replies. Few tools are known to handle such documents. Typically, their analysis means eyeballing, with aids of text editors (like `emacs`) or simple pattern search utilities such as the unix "grep". If more complex relationships are to be validated, special script programs (like those in `awk`, `perl`, or `python`) are constructed to search for the desired message.

The focus of this paper is to discuss technologies that can assist understanding of such semi-structured text. But just what do we mean by understanding what is in a document? One way of understanding is to focus on the morphology, and to classify which type of document the input is, i.e., a news story, a letter, a table, a form, a dialog, a computer program listing, a message log, etc. Each of these types of documents has a specific texture in terms of regularity in symbol density, line and field spacing, alignment, and repetition [13].

Going deeper, one may want to obtain the messages contained in the document. When the input is a page image, this involves symbol/token segmentation, symbol recognition, or OCR (Optical Character Recognition). This may also include removal of decorative graphics and markup tags, or determining the orientation of the text lines and the logical ordering of the text blocks [3][24]. Additionally, language identification may be needed for mixed language input [22], and context and language models may be called for in symbol recognition. At times, one may defer document classification to or integrate it with this stage if the document type must be disambiguated with symbol recognition. At this stage one gets ready for, but has not begun, extraction of information that connects to the application context.

In this work we analyze the information extraction steps after such early processing. Thus we assume the input to be ASCII text or transcriptions of text images with well delimited text lines in a logical order, and that the input has been accepted as an instance of a document of a known type. We do not consider graphics or decorative text printed in nonlinear layouts. Nor do we assume that the text strictly adheres to a pre-designed format or that we have a way to influence the behavior of the document generator. We explore the space beyond reach by existing specialized tools and commercial products that are designed for the analysis of data or event logs following a particular protocol, such as operating systems logs (e.g. `syslog`), web logs [8], or files generated by data logging software libraries coupled to the observed system. For such specialized tools we refer readers to [14].

We argue that, like steps in program compilation and natural language parsing, there can be cleanly separable stages in a system for extracting information from semi-structured documents. Generic and reusable components can comprise substantial parts

of the system, and domain-dependent drivers can be contained within a middle layer. We describe one such system that has been developed and used to parse and explore complicated engineering logs arising in network diagnosis (Figures 1, 2). Compared to single-shot, *ad hoc* pattern searchers, an important advantage of a modular architecture is that it supports incremental understanding and exploration. In the following sections, we discuss the details of the system, and highlight the problematic areas where practical difficulties have precluded a clean automation.

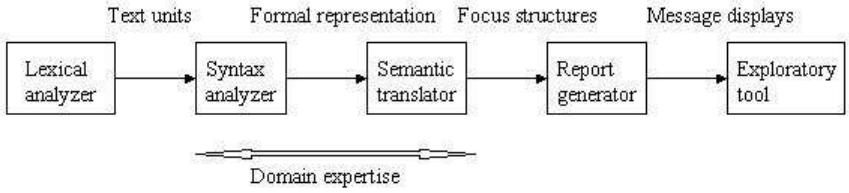


Fig. 1. A modular analysis pipeline for semi-structured text. Like computer program compilers and natural language understanding systems, it has distinct stages with well defined output. Up to the interfaces, the implementation in each stage is independent of those in others. We emphasize that domain-specific designs can be contained within the middle layers, leaving the front-end and back-end utilities generic and reusable.

```
mod1152 2005-05-24 21:46:06
... masterReadX() completed with 300 values of X read
... master initialized
line 557 reached
mod1205 2005-05-24 21:48:32
... !!!alarm thresholdCrossing T23
default value restored
mod567 2005-05-24 22:01:05
... slaveMakeY() paused at counter=46 y=577.03
...
```

Fig. 2. An example log file that records events in several software components. The more important flags and alarms in an intended regular format are mixed with transient, informal debugging messages.

3 Lexical Analysis, Syntax Guided Morphology Analysis and Shallow Parsing

In this stage, to understand is to parse the document according to the expected syntax. The input to this stage is a sequential chain of text blocks that have all tokens delimited. This could also be a transcription of imaged or audio documents. The output is a formal representation of the document as an internal data structure, which can be optionally serialized in XML or other forms of tagging [7].

The parser can be further decomposed into generic and domain-dependent modules. A generic module can be used for lexical analysis, where the purpose is to break down the incoming text into units suitable for comparison to the expected format. In this stage the system tokenizes the input stream, determines field types and looks for identifiers that are markers or delimiters of the expected messages. The identifiers can be symbols, symbol patterns, keywords, or stop words. Examples include the symbol ":" if a dialog is expected, the symbol "=" for key-value pairs, or fields in the form of "YY:MM:DD" for dates (Figure 3). Relevant tools include regular expression matchers, or search utilities taking a simple target symbol/string as an argument. Up to a simple parameterization, such search utilities can be generic, and many tools for this have been produced in early research in computer science. Simple data cleaning, such as repairing an inappropriate line breaks or converting time stamps to standard formats, may belong to this stage.

```
<logfile>
<logline>
<idstr> mod1152 </idstr>
<datestr> 2005-05-24 </datestr>
<timestr> 21:46:06 </timestr>
</logline>
<logline> ... masterReadX() completed with 300 values of X read </logline>
<logline> ... master initialized </logline>
<unknown> line 557 reached </unknown>
...
</logfile>
```

Fig. 3. Output of simple lexical analysis, where fields with recognizable format are marked.

Based on the expected message format, a subsequent syntax analyzer takes in text segments extracted from this stage and organizes them into the largest legitimate syntactic units (Figure 4). The syntax analyzer has to involve tailored code. If implemented in objected-oriented languages, the code would populate attributes of the objects that represent the target structure. If the structure of the input document follows the known object hierarchy, this can use classical techniques like recursive descent parsing. In less clean scenarios the parser must be tailored to provide more flexibility. For example, when there is indeterministic interleaving of multiple, spontaneous message streams, the parser has to be able to cope with concurrent, dangling structural references.

In practice, there is no guarantee that the object hierarchy is strictly observed in the input. While in program compilation it is perfectly reasonable to reject the input whenever the grammar is violated, pragmatic systems for log analysis must be error tolerant. The emphasis is on the search for useful fragments, and there is less concern in the validity of the format. Structure of the expected input provides guidance to the parser,

```
<sysXlog>
<modmsg>
<msgheader>
<id> mod1152 </id>
<date> <year> 2005 </year> <month> 5 </month> <day> 24 </day> </date>
<time> <hour> 21 </hour> <min> 46 </min> <sec> 06 </sec> </time>
</msgheader>
<msgbody>
<msg> ... masterReadX() completed with 300 values of X read </msg>
<msg> ... master initialized </msg>
</msgbody>
</modmsg>
...
</sysXlog>
```

Fig. 4. Output of a syntax analyzer that groups the input lines into largest recognizable units.

but it must not be overly controlling or limiting. Moreover, in many log analysis scenarios, the grammar is not available at the outstart. Instead, the expectation is developed incrementally by inspection and validation. Typically, a sequence of trial-and-error is applied to get rid of exceptions.

An interesting research question is to what extent pattern recognition techniques can be useful in this stage. Syntactic pattern recognition [4] immediately comes into the picture, and techniques such as automatic learning of regular expressions and pattern languages [1][2][19] and, more generally, grammatical inference [12][20][23] may provide help. Grammatical inference is known to be hard, though in recent years there have been moderate successes with re-stimulated interest. In time-pressured analysis scenarios, the difficulty of application of such techniques often denies them of their deserved opportunity. The offer of an easily applicable library of such elements should be welcomed [18].

We believe that, more likely in near-term, assistance would come from a collection of statistical tools for analyzing the text morphology, such as automated search of repeated units, computation of autocorrelation functions on simple quantities like line lengths, and other techniques commonly used in texture analysis, text compression, and crypto-analysis. These hints can aid the construction of the structural expectation.

A key point we emphasize here is the notion of incremental transformation. Syntax driven tools at this stage do not have to assume all the load of extracting the desired message from the document. Moderate goals, such as translating the document to a machine understandable structure, are sufficient and are often most easily accomplishable with tools designed for this level of processing. For example, while an `awk` script can easily extract a consecutive block of lines and translate them to a semantically related group of statements, it is not the ideal tool to implement the decision logic needed to look for low level components scattered within deeply nested structures. Semantic understanding can be deferred to the next stage, where target driven tools can be brought in to obtain the message of concern.

4 Deep Parsing, Focus Structure Selection and Target Generation

Documents converted to a formal representation are ready for event discovery and semantic translation. This is the stage where domain-specific knowledge is most intensively applied. Custom code is needed to express the goal of the analysis. Translation tools that are based on common representations of formal documents provide much

convenience. For example, for documents parsed into an XML representation, multiple XSLT scripts can be designed to apply tests to and extract contents from different components of the object hierarchy. Whenever necessarily, new data structures can be constructed to facilitate event discovery [21].

Again, rarely is it true that a full understanding can be achieved in a single shot. Conclusions need not be drawn at this stage. Flexibility can be sought from incremental constructions of the translation code, with the goal moderately set at producing new structures to represent grouping of potentially related sets of variables and messages, or at particular focus scales (Figure 5). For example, a translator for a time-stamped log may produce separate tables containing messages produced at the day, hour, or minute scales respectively Analysis of the messages and impacts can be deferred to the next stage.

Events

mod id	date	time	msg
mod1152	2005-05-24	21:46:06	masterReadX() completed with 300 values of X read
mod1152	2005-05-24	21:46:06	master initialized
mod1205	2005-05-24	21:48:32	alarm thresholdCrossing T23
mod567	2005-05-24	22:01:05	slaveMakeY() paused at counter=46 y=577.03
			...

Alarms

mod id	date	time	alarm type	port id
mod1205	2005-05-24	21:48:32	thresholdCrossing	T23
			...	

Fig. 5. Tables with different emphases created from the same log file.

When repeated analysis is needed on massive dumps, the target of this translation can be tables added to a full-strength relational database. For transient analysis, such tables can simply be stored as CSV (comma separated values) formatted files ready to be loaded into a display and reporting tool, such as an `Excel` spreadsheet. In either case, the notion of a record, i.e., a row in a table, represents an entity of analysis. Standard considerations in database design, such as attributes to contain in each record and keys to be chosen, are essential in designing such tables.

A research challenge for this stage is to find the most suitable data structures to support discovery of each type of events. Here events can be simple propositions and their boolean combinations, like “if flag A appears,” “if flag A appears but flag B is not seen,” or predicates like “if quantity x is less than 0.01.” An event may also involve more complex expressions like “flag A occurs 3 times and flag B never occurs more than 2 times before quantity x reaches 100.0.” While many questions about the data can be formulated as queries to a relational database, there are limits in the expressive power of such queries. Even if the event definition is well articulated, and stays within coverage of the query language, the decomposition of an engineering question into such queries is not always obvious. It could be a difficult maneuver to ask of the technician who, with intimate knowledge about the observed system, intuitively knows what should happen, but may not be able to tell before the event is first seen. Proper structures of the data and messages to support recognition of such events are much desired. This points back to a fundamental concern in pattern recognition, namely, the quest for the most suitable features and feature representations to support identification of a certain regularity.

5 Exploratory Analysis In Collections of Focus Structures

Once the document is converted to a collection of standard data structures such as tables or object trees, domain independent display and exploratory tools can again be introduced. It is well known that exploratory data analysis can benefit much from visual tools. Therefore we describe, as an example module in this stage, an interactive visual tool we developed for correlation of text strings organized in tables. The tool exploits the power of hypertext links and browsers to support relational querying without the need for a back-end database server.

Given a set of CSV formatted tables, the Java program `htmlgen` would infer simple connections between the tables. A connection can be as simple as that two tables share a common column name. In that case, the program constructs a set of table-specific web pages, another set of column-specific web pages, plus an index page containing links to both sets of pages. When a user clicks a link to a table-specific page, the next web page he sees contains the corresponding table. If instead he clicks into a column-specific page, he sees all the tables containing that column.

In either case, once presented with one or more tables, he can further click on a column name to sort the table by that column. More importantly, each cell in each table is a hyperlink. On a click at a particular cell, the script program embedded in the web page filters the currently seen tables by the value of this entry at this column. The next display is constructed which shows the filtered table or tables. The user may continue the filtering recursively, or back track and attempt a different filter.

The purpose of this design is to allow the user to see the text surrounded by different contexts. For example, if a column contains an alarm message, consecutive alarms may be seen in one table, and long range dependence among the alarms may be summarized in another table. Tables containing different sets of attributes and sharing several keys would allow projection of complicated message relationships onto different subspaces.

The design of the web pages and the embedded scripts follows the REST (Representational State Transfer) style [6]. The hyperlinks at the cells are constructed on-the-fly as a root site address concatenated with a sequence of operators and operands. The intent is to provide support for completely hassle-free querying by simple clicking. This removes unnecessary obstacles to the thought of the analyst who can now concentrate on the diagnostic tasks rather than struggle with forming proper symbolic queries in correct syntax. Thus the tool is another practice of the design philosophy in Mirage, another graphical tool of ours for numerical exploratory analysis [9][10][11].

A further advantage of the REST style address is that it records the sequence of steps the user has traced to arrive at a particular state of the tables. For example, the address for a page derived from filtering all tables created from the log of May 24, 2005 that contain the key field "modid" with values "modid=mod1152" and "hour=21" would be

```
http://ihost.xyz.com/Xlog/20050524/unique_modid.html?
modid=%20mod1152&hour=%2021
```

which can be bookmarked and revisited at a later time, after the analyst has a chance to pause and digest the information for what happened in the observed system that led to this state. Sometimes insights may come only after he sees the same thing happening

under different but similar scenarios. In this way, the tool can support discovery of certain events that can be articulated only after it is seen over and over again.

Our utility for automatically inferring such hyperlinks uses no other knowledge of the table contents. This guarantees that the pages can be rebuilt easily when the table changes. The need arises when the user decides to change the semantic translators to generate tables with different selected components. It also means that tables can be accumulated, recombined, and cross linked in an arbitrary way. This is especially suited for historical archives and for comparisons of system states from different environments and perspectives. The query processor is extremely light-weight: no database installation is involved, and after one application of `htmlgen` to build the basic sets of web pages, nothing more is needed from the server. The collection of web pages can be taken off-line for querying and browsing. This provides a critical functionality of relational databases – querying in the form of "select ... from ... where ..."– without the need for the expensive, and often intimidating process of installing and populating a full-scale database. Moreover, there is no need for continuous connection to a database server, which provides additional convenience for in-field analysis.

Despite its simplicity, the tool demonstrates powerful consequences when proper considerations are given on extracting and isolating common functionalities in a document analysis process. For further research, one may consider generalization towards more sophisticated schema matching rather than relying on column names, approximate or partial matches rather than exact matches, and facilities for multimedia content retrieval when the table cells feature combinations of text, graphics, images, and others. Meanings of the clicks can be changed with options in the page header. With every cell clickable, use of the display estate can be maximized. Development of sophisticated tools to do these would benefit from many current research activities in pattern recognition.

6 Conclusions

We have witnessed the power of modern internet search engines that rely on primitive keyword matching. Though efficient implementations of such simple searches are highly nontrivial, we have reasons to believe that the technology has only scratched the surface of complex pattern recognition and knowledge discovery from analysis of documents.

In this paper we analyzed the information extraction stages of a document understanding process. We argued the advantages of a modular system architecture, and outlined the relevant pattern recognition challenges at each stage. Starting with a particular demand for engineering diagnosis, we investigated the issues, built a system for practical use, and generalized it to suit many other scenarios where there is a need for progressive annotation, multi-scale and dynamic context display, and exploratory analysis with text. We summarized our observations from this experience as well as from our knowledge of similar systems.

Beyond the engineering diagnosis setting, there are many documents that have some minimal structural regularity that can be subjected to a similar "syntax, semantics, and event discovery" analysis pipeline. Examples are journals, blogs, interviews, plays,

court transcripts, news stories, and all kinds of transaction and event records in science and humanities. The analyses of these documents share a common characteristic that the process is explicitly driven by a search for events and patterns. There is a large, unfilled space between trivial keyword matching and the extremely difficult domain of natural language understanding. The popular "bag of words" analysis of documents using techniques like latent semantic indexing does not facilitate complex event searching and thus still leaves many unserved needs.

For text exploration, we developed a tool for automatic hypertext generation that embeds a light-weight query processor into a table display, such that relational projections can be applied without requiring the user to install special software or database servers. We believe that its generality and extreme ease of use can encourage incremental attempts on semantic translation, which allow for continuous accumulation and evolution of discoveries from a document.

Acknowledgements

We thank Phil Bohannon for organizing a text analysis seminar where many ideas presented here were discussed, Narasimhan Raghavan and Suresh Goyal for bringing to us the practical challenge, and Cliff Martin, Steve Fortune, and David James for useful exchanges.

References

1. D. Angluin, Finding Patterns Common to A Set of Strings, *Proc. of the 11th Annual ACM Symposium on Theory of Computing*, Atlanta, 1979, 130 - 141.
2. D. Angluin, Learning Regular Sets From Queries and Counterexamples, *Information and Computation*, Vol. 75, 1987, 87-106.
3. H.S. Baird. Anatomy of a versatile page reader. *Proceedings of the IEEE*, Vol. 80, No. 7, Jul 1992, 1059–1065.
4. H Bunke, A Sanfeliu, *Syntactic And Structural Pattern Recognition: Theory And Applications*, World Scientific, 1990.
5. P. Cullen, T.K. Ho, J.J. Hull, M. Prussak, S.N. Srihari, Contextual Analysis of Machine Printed Addresses, *Proc. of the 4th USPS Advanced Technology Conference*, Washington, D.C., Nov 1990, 779-793.
6. R.T. Fielding, *Architectural Styles and the Design of Network-based Software Architectures*, PhD Dissertation, Information and Computer Science, University of California, Irvine, 2000.
7. K. Franke, I. Guyon, L. Schomaker, L. Vuurpijl, The WANDAML Markup Language for Digital Document Annotation, *Proc. of the 9th International Workshop on Frontiers in Handwriting Recognition*, 563-568.
8. N.S. Glance, M. Hurst, T. Tomokiyo, BlogPulse: Automated Trend Discovery for Weblogs, *Proc. of WWW 2004*, New York, May 17-22, `http://www.blogpulse.com/research.html`.
9. T.K. Ho, Exploratory Analysis of Point Proximity in Subspaces, *Proc. of the 16th ICPR*, Quebec City, Canada, Aug 11-15, 2002.
10. T.K. Ho, Interactive Tools for Pattern Discovery, *Proc. of the 17th ICPR*, Cambridge, U.K. Aug 22-26, 2004, Vol. 2, 509-512.

11. T.K. Ho, Mirage project site: http://www.cs.bell-labs.com/who/tkh/mirage.
12. V. Honavar, G. Slutzki (eds.), *Proc. of the 4th International Colloquium on Grammatical Inference*, ICGI-98, Iowa, July 12-14, 1998. Lecture Notes in Artificial Intelligence, Vol. 1433, Springer.
13. J. Hu, R. Kashi, G. Wilfong, Document Image Layout Comparison and Classification, *Proc. of the 5th ICDAR*, Bangalore, 1999, 285.
14. Loganalysis.org, http://www.loganalysis.org.
15. D. Lopresti, G. Nagy, Automated Table Processing: An (Opinionated) Survey, *Proc. IAPR Workshop on Graphics Recognition (GREC99)*, Jaipur, Sep 1999, 109-134.
16. S. Madhvanath, V. Govindaraju, V. Ramanaprasad, D.S. Lee, S.N. Srihari, Reading Handwritten US Census Forms, *Proc. of the 3rd ICDAR*, Vol. 1, 1995, 82.
17. G. Nagy, Twenty Years of Document Image Analysis in PAMI, IEEE Trans. PAMI-22, 1, Jan 2000, 38-62.
18. V. Raman, J.M. Hellerstein, Potter's Wheel: An Interactive Data Cleaning System, *Proc. of the 27th VLDB Conference*, Roma, Italy, 2001.
19. P. Rossmanith, T. Zeugmann, Stochastic Finite Learning of the Pattern Languages, *Machine Learning*, Vol. 44, 2001, 67–91.
20. Y. Sakakibara, Grammatical Inference in Bioinformatics, *IEEE Trans. on Pattern Analysis and Machine Intelligence*, Vol. 27, 2005, 1051–1062.
21. J. Shanmugasundaram, K. Tufte, G. He, C. Zhang, D. DeWitt, J. Naughton, Relational Databases for Querying XML Documents: Limitations and Opportunities, *Proc. of the 25th VLDB Conference*, Edinburgh, Scotland, 1999.
22. A.L. Spitz, Determination of the Script And Language Content of Document Images, *IEEE Trans. Pattern Analysis and Machine Intelligence*, Vol. 19, No. 3, 1997, 235–245.
23. M. van Zaanen, The Grammatical Induction Website, http://eurise.univ-st-etienne.fr/gi.
24. T. Watanabe, T. Sobue, Layout Analysis of Complex Documents *Proc. of the 15th ICPR*, Barcelona, Vol. 4, 4447.