Pacific Symposium on Biocomputing 4:290-301 (1999)

IMPROVING THE EFFICIENCY OF A USER-DRIVEN LEARNING SYSTEM WITH RECONFIGURABLE HARDWARE. APPLICATION TO DNA SPLICING.

E. LEMOINE, D. MERCERON, J. SALLANTIN
LIRMM, 161 rue Ada, 34392 Montpellier cedex 5, France
{*lemoine,merceron,sallantin*}*@lirmm.fr*

E. MEPHU NGUIFO
CRIL - IUT de Lens, Rue de l'université SP 16, 62307 Lens, France
mephu@cril.univ-artois.fr

This paper describes a new approach to problem solving by splitting up problem component parts between software and hardware. Our main idea arises from the combination of two previously published works. The first one proposed a conceptual environment of concept modelling in which the machine and the human expert interact. The second one reported an algorithm based on reconfigurable hardware system which outperforms any kind of previously published genetic data base scanning hardware or algorithms. Here we show how efficient the interaction between the machine and the expert is when the concept modelling is based on reconfigurable hardware system. Their cooperation is thus achieved with an real time interaction speed. The designed system has been partially applied to the recognition of primate splice junctions sites in genetic sequences.

1 Introduction

Molecular biology requires both flexible and powerful tools to adapt itself to a rapidly expanding field. In such a domain, an active research area in the hierarchical approach of sequence analysis is the prediction of primate splice junction sites from the DeoxyriboNucleic Acid (DNA) sequence. Splice junction sites are points on a DNA sequence at which 'superflous' DNA is removed during the process of protein creation in higher organisms. Established approaches to this problem have involved handcrafted rules by experts, and statistical methods. More recently, a variety of machine learning methods [17,24] have been applied: both neural networks, and symbolic induction. The results of these methods are often more effective and accurate than their human-designed counterparts. We have therefore successfully applied our system LEGAL [22] to the prediction of splice junction sites from genetic sequences.

LEGAL methodology [21] focuses on one concept with imperfect domain theory, builds a concept formulation and the user is able to examine how the concept formulation is efficient on data, and how it is possible to force an evolution of the concept formation. Its methodology is described in a data

driven way and is based on the different levels expressed by Russell [27] when searching for an abstraction using the domain of number theory. We have extended Russell's process by relying it to Lakatos' work [12,16] on proofs and refutations in order to establish a cooperative control between the user and the system. As we deal with an imperfect domain theory, initial data may be incomplete or may contain many biases. Our goal is to find a constructive concept formation that gives to the user the tools to formulate a concept and to revise this formulation with a human admissible flow of interaction. As reported by Clark and Rawlings [5], the output of LEGAL forms part of an active dialog with the user which may be critiqued so that the final hypothesis is a synthesis of human knowledge and the empirical data.

However, the progresses made all around the world concerning the mapping and sequencing of the genome of *Homo sapiens* and other species have increased the size of databases exponentially. Therefore even the best workstation would not be able to reach the scanning speed required. Consequently, this will limit the cooperation between the user and the system as the required speed for an active dialog would considerably decrease with a huge amount of data. So to avoid this shortcoming, we have designed a new method based on the partition of problem component parts between software and hardware [11,15,29]. As it had been already demonstrated, many advantages arises from the use of an accelerator dedicated to molecular biology such as [18]:

- the interaction between the user and the system is possible with an admissible time complexity;
- many operations can be done;
- the user has enough time to validate the results produced through its cooperation with the system;

Our approach derives from a new pattern matching algorithm [18] based on reconfigurable hardware, A^2R^2, that outperforms any kind of previously published genetic database. This algorithm runs on a new kind of massively parallel and low cost computer now available: the reconfigurable hardware systems. It is based on a bit level operation model that enables the implementor to fit the hardware to the problem rather than distorting the problem to fit the computer. The main idea with A^2R^2 was to significantly increase the speed of an algorithm for molecular genetics in order to increase the quality of the results produced.

Our approach is based on **hardware/software dynamic reconfigurability through data behavior and result interpretation**. The problem solving is partitioned between several units (see fig 1). The system adjusts

the hardware/software design according to its use and to the user behavior during the application execution. Thus, we expect to have the best algorithm for the best architecture. As a matter of fact, our approach can be seen as an optimization of problem solving. Figure 1 illustrates the stream controls between the three design units:

- **User interface** : takes charge of the hardware/software partition. This unit assumes the interface between the hardware/software mapped functions and the user actions.
- **Software** : takes charge of functions to be mapped on the software.
- **Hardware** : takes charge of functions to be mapped on the hardware.

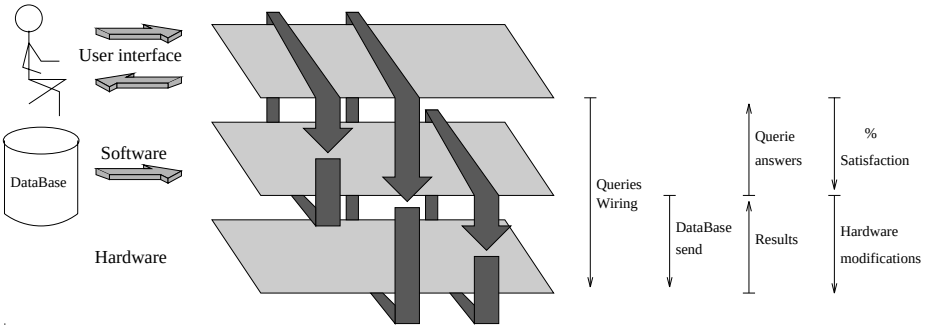


Figure 1: **Relation between our approach units.** Our approach admits 3 layout units (user interface, software, hardware) in which several data streams allow a good communication. In this case, the user requests to the user interface, which makes a first hardware/software partition and hardwires the queries on the hardware unit. After sending the database, the user (knowing the results) may modify his queries or request new ones.

The paper is organized as follows. Section 2 overviews the system LEGAL which is an implementation of concept modelling. Section 3 describes our dynamic approach. Section 4 illustrates comparative results obtained with or without our reconfigurable hardware system using DNA splicing problem [7,10,19]. As many works have been devoted to software/hardware methodologies, the last section discusses the strengths of our approach.

2 Concept Modelling: An Overview of LEGAL

We review here the foundations [21] of LEGAL. Automated knowledge acquisition and machine learning systems generally consist in defining a language

description, finding an abstraction, and validating it on data. Our process of concept abstraction is progressive and requires different levels.

The **first** level is related to the choice by the user of the examples characterizing the concept. At the **second** level, the language used to describe objects is defined. Level **3** defines regularity, which is a feature that is retrieved among objects descriptions. A regularity is a conjunction of attributes which often holds for an example, like the presence of GGTG at the middle of sequence segment. Level **4** is related to hypothesis, which is a combination of a regularity and a subgroup of objects, such that those objects verify the regularity. Concept formulation in level **5** is related to an organization of hypothesis which is a lattice of hypothesis.

Knowledge acquisition is achieved by first validating concept formulation on new data sets, and then by interacting with the user. These are the aims of levels **6** to **9** which respectively express decision and argumentation using concept formulation. The decision in LEGAL is expressed by two different reasoning mecanisms. The first one, similar to that defined by Peirce [25], is called **empirical reasoning** which is a deduction based on a principle of majority vote onto the set of built regularities. The second one, **analogical reasoning**, is based on a similarity measure to a reference set of objects, which can be for example the training set of examples and counter-examples.

These decisions initiate an interaction between the user and the system, so that the system becomes able to provide a plausible explanation to the user, who in turn can validate or refute it. The control is partially based on the notion of objection. An objection is what is sufficient on an object to refute it as an example. There are two kinds of objections. The first one, termed **general objection** is a sufficient reason to reject an object. The user may analyse built general objections and reject thoses which are not relevant. For example in biology it is well-known that there is a consensus for good splice junction sites in about 90% of cases. If the user rejected it as an objection, he can then introduce in the training set, some objects that have no consensus. He can also change the object description. In both cases this allows the system to refine its knowledge. **Contextual objection** is related to a reference example given by the user. Searching contextual objections comes to build new existential propositions that can be experimentally refuted. Thoses qualities expressed modifications required to an object to be an example.

The cooperation between the user and the system allows to return to previous levels if the user critics the system justification. The concept formulation is refined by removing or modifying this knowledge. Recent years have witnessed a growing interest in developping multistrategy systems that integrate two or more inferences types and/or computational paradigms [20]. Our goal

in LEGAL has been to build different strategies (induction, deduction, analogy) that are from now handly managed by the user. Such goal allows the system to take advantage of the complementarity of different inference types of representational mechanisms.

In a domain such as molecular biology, there is often a great exchange of information between the user and any dedicated system when analyzing new sequences. This interaction may really decrease the utility of LEGAL if the dialog with the user isn't done with an admissible required speed. In order to avoid this shortcoming, we have designed a new methodology which is based on dynamic reconfigurable hardware.

3 Reconfigurable Hardware and CoDesign

The reconfigurable hardware concept, as well as its implementation with Field Programmable Gate Arrays (FPGAs) was introduced by different teams at the end of the Eighties. Vuillemin, Lopresti and Kean described various systems based on this concept[2,9,8]. In fact, this concept had been latent in the literature since the Seventies. One of the first to have expressed it was Schaffner[28] in 1972. However, it is only with the arrival of the SRAM based FPGA in 1985 by Xilinx that it became possible to implement this concept. The characteristics of reconfigurable hardware are, a great development facility, together with a flexibility that is only encountered in programmable systems. The level of performances reached by a reconfigurable hardware are those of a dedicated system.

3.1 The Experimental Platform

FPGA

Basically a FPGA is a grid of small memories 16x1 bits interconnected by a network across the whole grid. These small memories are lookup tables that can implement any logic functions of four variables. In the FPGA that we used, the lookup tables are grouped by two with two additional 1 bit registers and form a CLB (Configurable Logic Block) the basic element of the Xilinx FPGA. The network is fully configurable, each interconnection is made with on pass transitors controlled by the state of an one bit static memory. The bitmap (bitstream file in the FPGA terminology) form by all this one bit memory plus the CLBs look-up table was download in the FPGA before the execution. Therefore change the design of an FPGA is done by download a new bitstream file that configurate the interconnections between binary functions (CLB look-up table).

DecPerle-1

The experimental platform is constitued by a DEC workstation and a co-processing card attached to it. This card contains 23 Xilinx FPGAs and 4 MBytes of SRAM. 16 FPGAs are connected in a 2D array with local interconnection and memory buses. Therefore 5120 (16×320: 16 FPGA, 320 CLBs per FPGA), binary fonctions, equivalent to 1 bit arithmetic and logic unit, can be evaluated, at each cycle, in the array. The maximum bandwidth between this array and the memory is 320 MBytes/s with an access time from SRAM to FPGAs of 50ns. The hardware is connected to a DecStation 5000/240 by a bus TurboChannel at 100 MBytes/s.

On this workstation the design is developed in C++ and translated through Xilinx tools in a bitstream configuration ready to be loaded on the FPGAs. The bitstream configuration can be considered as a unique nanoinstruction of 1.4 Megabits width that loads itself in less than 50 ms on the card [30].

3.2 Design Framework

Our approach is based on hardware/software dynamic reconfigurability through data behavior and result interpretation. The problem solving is partitioned in between several units such as User interface, Software or Hardware (see fig 1) that take charge of a set of functions. The system adjusts the hardware/software design according to its use and to the user behavior during the application execution. After running, we expect to have the best algorithm for the best architecture. The streams (represented by loops in the figure) are :

- **User interface/Software stream** : The user interface configures the software unit for the problem solving (after the hardware/software partition). It can receive informations back from the software compilation (errors, possible optimizations) or results from the software execution.

- **User interface/Hardware stream** : The user interface configures the hardware unit for the problem solving (after the hardware/software partition). It can receive informations back from the hardware compilation (wiring & placement problems) or results from the hardware execution.

- **Hardware/Software stream** : This stream constitutes in fact the hardware/software refinement in order to reach the optimal problem solving. It also allows a hardware/ software data exchange.

MajAnd : A threshold Parallel Counter

In order to implement the majority vote needed by the empirical proof we used a parallel counter (PCs) with a thresholded output. Dadda in[6] has described an optimal combination of PCs as building blocks. For the application purpose we only need PCs smaller than (31,5) therefore with a 5 inputs CLB (two 4 inputs look-up table grouped in one) we can implement any threshold functions thus realize an n inputs majoritaire And with just an extra cost of one CLB over the corresponding (n,m) PCs. We change the threshold by writing straight into the bitstream configuration of the FPGAs into the corresponding CLB look-up tables.

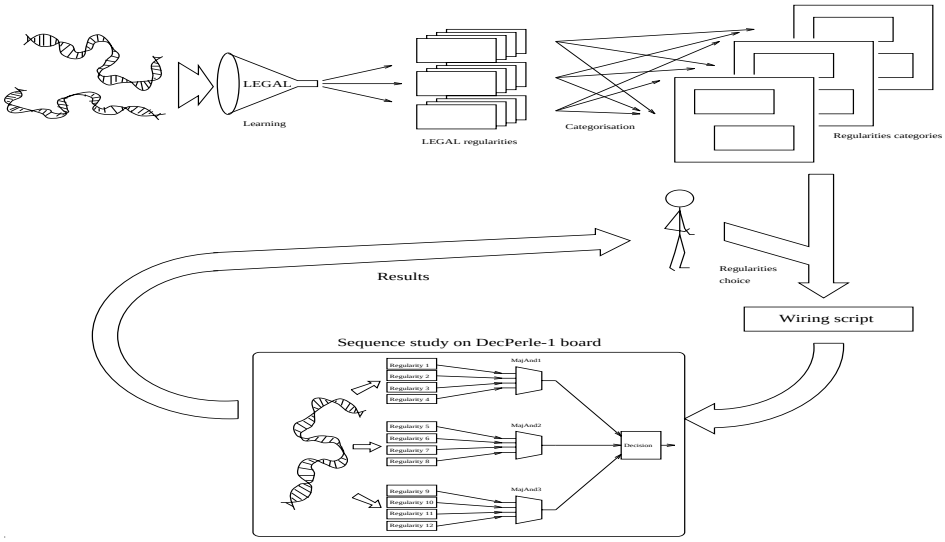
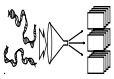


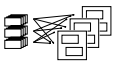
Figure 2: **Genetic sequence study through our approach.**

3.3 Hardware/Software LEGAL Implementation

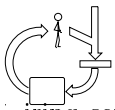
Basically the LEGAL system defines three levels (see fig 2) :



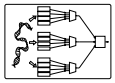
The first step is dedicated to the learning of a set of example and counter example. For each set in input the learning algorithm produces a set of regularities that characterizes the set and concept associated with it. This step is performed totaly in software owing to the complexity of the operations involved and also because it is not interactive and therefore can be executed in batch mode.



The categorisation level is performed directly by the biologist through an interface that requires little computing time.



The third stage is a loop between the biologist's queries set and data bases answers. A query based on a sub set of regularities generated by the previous stages of LEGAL is translated in a wiring script and then integrated in the bitstream produced by the wiring script. Eventually the bitstream is downloaded in the DecPeRLe-1 board and the data base scanning can start. The answers produced by the scanning are dispatched to the biologist who interacts and requests a new set of queries closing the loop.



The most time consuming and time critical part of the LEGAL system is the data base scanning with a subset of regularities. The regularities are formed by a set of Binary Substitution Vectors (BSVs). A BSV is simply the list of authorized letters at the position of the vector. A regularity of length N on an alphabet of L letters is made up by a list of N BSV of L bits. For example [A,G]...G..[C,T,G] is a regularity (. is a short cuts for [A,T,G,C]) that can match with sequences like ATTTGAAC or GATAGCCT.

Let us now examine how the detection of a given regularity is translated into the hardware. The key point is to fit a BSV with a lookup table. Then a 4x1 (20x1) bit lookup table implements a nucleotide (amino acid) BSV. This is due to the number of bits necessary to code the nucleotide or amino acid alphabet. As we have already pointed out one CLB contains two 16x1 bit lookup tables. One CLB can implement two nucleotide BSV or one amino acid BSV by bringing together each of their lookup tables of 16 bits. A regularity is made up of a certain number of BSV therefore a motif detector is made up of several CLBs whose outputs are combined by an AND gate. See figure 3.

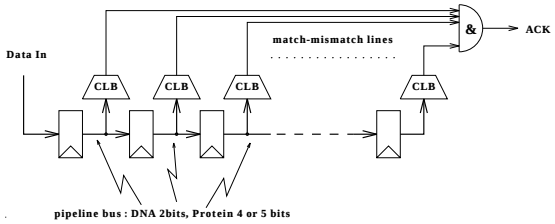


Figure 3: **Scheme of a single regularity detector.**

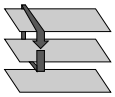
The data base is injected nucleotide by nucleotide sequentially and is

pipelined in the FPGA in such a way that a new word can be presented at each cycle to all the regularity detectors. The pipeline can be seen as a window that moves along the sequence. The output of each detectors are connected to the inputs of a Majority And according to their categories. The output of MajAnd are directly sent to the host station or through a hard wired decision function (implemented with a look-up table).

4 Benchmarks

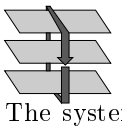
In order to estimate our approach performances, let's have a look at the main benchmarks for the three streams previously introduced.

4.1 User interface/Software stream benchmark



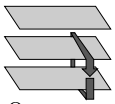
This stream is the less used in our system. Its consists in compiling the software unit, which takes arround 2 seconds, and in sending the results to the user interface. This step is almost immediate, but still depends on the way to present the results (for instance, a Yes/No answer will spend less time than a statistic answer).

4.2 User interface/Hardware stream benchmark



Once the user has queried the system, the queries must be wiried on the hardware unit. This stream (which is actually the slowest) is totaly depent on the wiring and routing tools performances. The system can easily wire 64 regularities per FPGA XC3090 (used on the DecPerle-1) with a 60 letter width pattern size. To be accomplished, this step requires around 100 seconds per FPGA for wiring, placing and downloading the queries on the hardware unit.

4.3 Hardware/Software stream benchmark



In order to have a better idea of the time scales, we will, in this section, compare the application on the DecPerle-1 board with the same application running only on a R3000 processor (40 Mhz). Once programmed in C, the R3000 software has been strongly optimized thanks to the **-O3** compilation option and the **pixie** profiler tools. We also assume an optimistic 1 cycle per instruction and an infinite size cache. The performances depend on the type of study we want to make. The first one concerns n sequences with a PatternSize size, where PatternSize is the width of the study pattern (see fig 4).

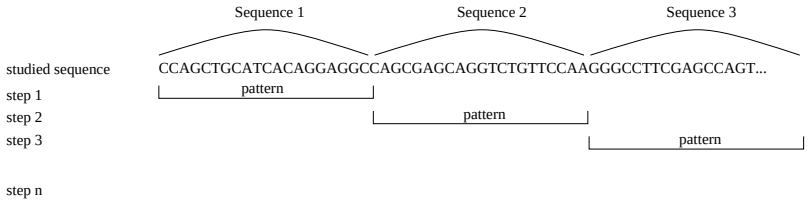


Figure 4: **N genetic pattern studied with the pattern size equal to the sequence size.** We study here n sequences where both pattern and sequence sizes are equal. We therefore need to serialy send the genetic sequences to the board which will make a logical shift before starting the study.

In this case the **speedup obtained by wiring queries is 440**. This result is independant of the sequence size and pattern size. This is due to a logical shift on the hardware unit that slows down the sequence analysis

The second kind of study concerns only one sequence of size S on which we shift the pattern. We would use this kind of study for the whole genomic sequence (see fig 5).

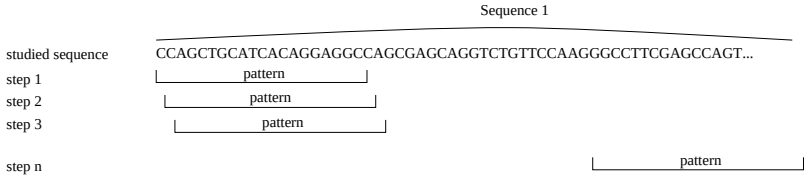


Figure 5: **Genetic sequence studied by shifting pattern.** We study here a sequence by shifting nucleotide by nucleotide the window onto the sequence.

For the different types of studies, we can obtain the following formulas to calculate the time spent, where n is the number of sequences studied, S is the sequence size, and PSize is the pattern (or window) size.

Type of study	R3000 (40 Mhz)	DecPerle-1 (40 Mhz)	SpeedUp
n seq. with S=PSize	11.10^{-6}*n*S (s)	25.10^{-9}*n*S (s)	440
one sequence size S	11.10^{-6}*S*PSize (s)	25.10^{-9}*S (s)	440*PSize

5 Summary and Conclusions

We have shown that a Software/Hardware architecture is a solution for constructive concept modelling in a field such as molecular biology. This derived

platform has been tested for the LEGAL empirical reasoning level. Our approach opens a new field to genetic sequence investigation because we are able to speed up the scanning of genetic databank and to control the time required by the interaction between the user and the databank when he tries to build a new concept. This acceleration allows a best control of the knowledge acquisition process as increasing the flow of results is sufficient to achieve statistical interpretation and interactive construction of a concept.

In our methodology the more the user is satisfy by the given result, the more the time to retrieve new results is accelerated (see fig 6). This is a consequence of the fact that the user has gained confidence with the hardwiring queries. However two main limitations of this work arise from the knowledge management [26] and the sequence description language which is due to the difficulty to take in account knowledge provided by the sequence secondary structure.

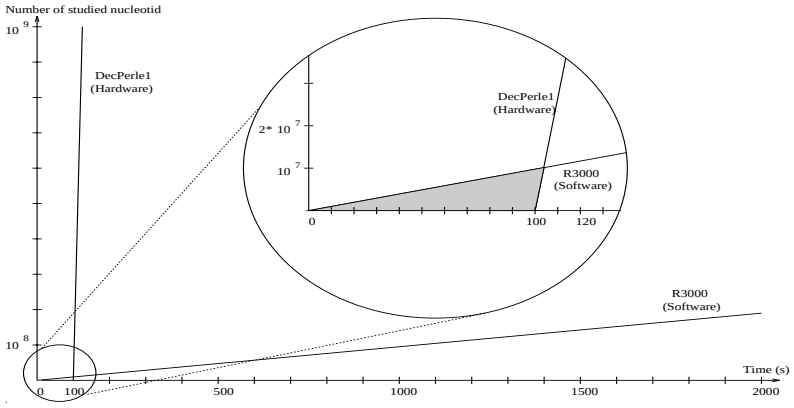


Figure 6: **Hardware and software genetic sequence study comparison.** This figure shows the number of nucleotide studied during a given time. The hardware study needs around 100 s for hardwiring the new queries (we use one FPGA here while we could use 16 times more). The intersection between hardware and software study is for 10^7 nucleotides.

Acknowledgments

Financial support for this work has been made available by French programmes: GDR 1029 (*Informatique et Genomes*) and GIP GREG. Most of this research was completed while all the authors were at LIRMM. Thanks to our latex guru, Olivier Roussel. We would like to thank our anonymous reviewers.

References

1. *Proc of ACM Intl. Symp. on Field Programmable Gate Arrays*, ACM Press, (1997).
2. P. Bertin *et al*, *Systolic Array Processors*, Prentice Hall, 300-309, (1989).
3. P. Bertin *et al*, *Proc of the Symp. on Integrated Systems*, (1993).
4. S.D. Brown *et al*, *Field Programmable Gate Arrays*, (1992), Kluwer.
5. D.A. Clark and C.J. Rawlings, *CABIOS*, 10(2), 199-205, (1994).
6. L. Dadda, *Alta Frequenza*, 19, 349-356, (1965).
7. J. Ficket, *Nucleic Acids Research*, 10, 5303-5318, (1982).
8. J.P. Gray and T.A. Kean, *Proc of Caltech Conf.*, 279-295, (1989).
9. M. Gokhale *et al*, *IEEE Computer*, Vol. 24(1), 81-89, (1991).
10. M.R. Green, *Annual Review Genetics*, 20, 671-708, (1986).
11. R. K. Gupta and G. De Micheli, *IEEE TR No. CSL-TR-92-518*, (1992).
12. F. Hayes-Roth, *Machine Learning: An AI Approach*, 221-240, (1986).
13. *Proc IEEE Symp. on FPGAs for Custom Computing Machines*, (1996).
14. J.H. Jenkins, *Designing with FPGAs and CPLDs*, Prentice Hall, (1994).
15. A. Kalavade and E.A. Lee , *IEEE Design & Test of comp.*, 16-28, (1993).
16. I. Lakatos, *Preuves et Refutations*, Hermann Ed, (1984).
17. A. Lapedes *et al*, *Proc. of the Interface between Computation Science and Nucleic Acid Sequencing Workshop*, Addison-Wesley, 157-182, (1990).
18. E. Lemoine *et al*, *Proc. of ISMB*, (1994), AAAI Press, Standford, CA.
19. M. Li, *Proc. of IEEE symp. on Found. of Comp. Sci.*, 125-134, (1990).
20. *Machine Learning Journal*, 11, (1993).
21. E. Mephu Nguifo and J. Sallantin, *Proc. of ISMB*, 292-300, (1993).
22. E. Mephu Nguifo, *Proc. of TAI*, IEEE Press, (1994).
23. W.R. Moore and W. Luk, *FLP'95 - LNAI*, Springer-Verlag, 975, (1995).
24. M. O. Noordewier *et al*, *Advances in NIPS*, M. Kaufmann Ed, 3, (1991).
25. C.S. Pierce, *Reasoning and the Logic of Things*, Havard U. Press, (1898).
26. H. Ripoche *et al*, *Proc of GIW*, (1994), Japan.
27. B. Russel, *The Principles of Mathematics*, A. Unwin Ed., (1956).
28. M. Schaffner, *IEEE Trans. on Computers*, C27(11), 1015-1028, (1978).
29. D.E.Thomas *et al*, *IEEE Design & Test of computers*, 6-15, (1993).
30. H. Touati, Technical Report TN4, Paris Research Laboratory, (1992).