Interactive GP for Data Retrieval in Medical Databases

Yann Landrin-Schweitzer, Pierre Collet, and Evelyne Lutton

INRIA - Rocquencourt, B.P. 105, 78153 LE CHESNAY Cedex, France,
Yann.Landrin-Schweitzer.inria.fr, Pierre.Collet@inria.fr,
Evelyne.Lutton@inria.fr,
http://www-rocq.inria.fr/fractales/

Abstract. We present in this paper the design of ELISE, an interactive GP system for document retrieval tasks in very large medical databases. The components of ELISE have been tailored in order to produce a system that is capable of suggesting documents related to the query that may be of interest to the user, thanks to evolved profiling information. Tests on the "Cystic Fibrosis Database" benchmark [2] show that, while suggesting original documents by adaptation of its internal rules to the context of the user, ELISE is able to improve its recall rate.

1 Introduction

Medical databases of large pharmaceutical companies are becoming really huge, not only in a linear or polynomial way, but by discrete steps each time a new company is absorbed that already had its own database. The result is a patchwork of smaller databases that each have their own structure and format, focussed on different fields and in possibly different languages.

However, modern search engines have very impressive precision and recall rates meaning that they manage to retrieve nearly all the relevant documents for a specific query.

This leads to the paradoxical problem that for each query, the user gets back hundreds if not thousands of documents so precisely focussed on his query that all first documents contain more or less the same data in a way that may recall convergence in evolutionary algorithms. After a certain rank, precision decreases rapidly and documents become only very loosely related to the user query.

The new difficult challenge for search engines operating on such databases is to introduce diversity in the retrieved documents, while keeping at the same time a high relevance to the original query.

One way of achieving this feat is to take into account some kind of user profiling, based on the history of the previous requests made by the user, with the aim of retrieving data both of interest to a specific user and at the same time matching his query.

This remains however quite deterministic, and other methods are seeked to introduce more diversity and randomness in the results, so as to mimic lateral thinking.

These specifications triggered the idea to use evolutionary algorithms, and more precisely genetic programming to evolve a user profile and use it to rewrite

[0] This research is partly funded by Novartis-Pharma (IK@N/KE)

user queries that may retrieve documents of interest of the user, even if they do not absolutely exactly correspond to his query.

The ELISE (Evolutionary Learning Interactive Search Engine) uses the specificities of a Parisian Approach to cut down the number of necessary evaluations to see an evolution in the model.

Section 2 presents an overview of the ELISE system. The genome and structure of the GP on which ELISE is based are detailed in section 3, with special focus on the structure of the rewriting language used to encode the profile and on the Parisian approach scheme. Section 4 presents the characteristics of the evolutionary engine used in ELISE and experiments are analysed in section 5. Conclusion and future works are described in section 6.

2 ELISE : an Evolutionary Learning Interactive Search Engine.

ELISE relies on an interactive EA that evolves a "user profile" with a *Parisian Approach*[4] —i.e. represented by the whole population of the EA. This profile determines the process that translates user queries into an alternate form taking into account user specificities. A new generation is produced with each query and fitness is computed by the analysis of the user's behaviour.

Experience and earlier tests on ELISE have shown that the choice of this particular structure is more convenient than other potential strategies of queries rewriting, like the exploitation of user-specific thesauri (as in a classical semantic expansion) or the deterministic use of one or several complete rewriting rules.

A set of small parts of rewriting rules, hereafter referred to as *modules*, is evolved by the system. The evolution relies on a Parisian EA: some modules are extracted out of a population of modules by a ranking selection and fit into templates to produce a fixed number of rewriting rules. Each new query obtained by applying a rule to the original query is then passed on to the boolean search engine and results lists are merged and reordered to be presented to the user. Among the list returned by the engine, documents that are actually viewed by the user translate into an external fitness bonus to any rule that triggered the apparition of this document and by extension to any module involved in the production of this rule. A proportion of the individuals, with lower fitnesses, is then replaced by new individuals produced through mutation and crossover from the best part of the population.

ELISE can be used as an improving mechanism on top of any kind of boolean search engine. The one used for experiments (see section 5) is SWISH++, [19], a basic public-domain boolean search engine, without any semantic expansion or stemming tool. Independantly it needs a semantic network to perform semantically sound term replacement. As the test where performed on a medical database, the EMBase thesaurus was used [6].

3 Genome and structure

3.1 Rewriting language

The way the modules are coded and their expressive capabilities (i.e. complexity) have a deep impact on the depth and generality of the rules that can

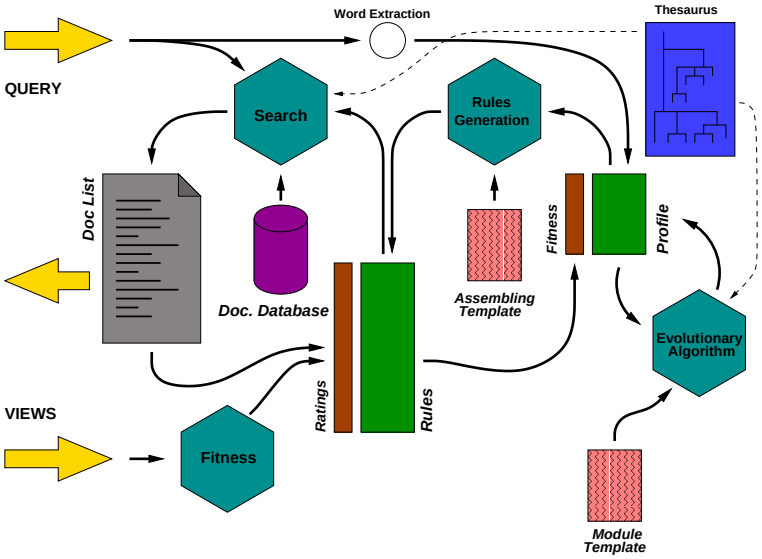

Fig. 1: ELISE evolves a set of rewriting modules, the "user profile." Queries are rewritten by rules derived from this profile and returned documents are displayed to the user. Fitness functions of modules are then updated and activate a new generation of the EA.

be evolved. These must meet two main requirements in order to be used within ELISE:

- The first requirement is a structure constraint. Since modules are basic *components* of more elaborate structures, they must be assembled together in a flexible, yet reliable way: the resulting rule must be valid, whatever modules it is made of. In order to support genetic operators, both local modifications (mutations) and global relocations (implied by recombination) must not violate syntax and must be compatible with the main part of the data implicitely encoded into the module. Moreover, relocations must be made at a small computational cost.
- The second requirement is a power requirement. These structures need to operate efficiently, although with enough generality, on boolean queries. A boolean query is essentially a tree, where leaves are query terms and nodes are boolean operators. There is no theoretical limit on the complexity of such trees (i.e. depth) and no meaningful empirical limit either —as common web search-engine practice reveals. This implies that the rules must be able to process arbitrarily large sets of data.

The most common descriptions used by rewriting systems (e.g. in regular-expression-based rewriting) are based on models equivalent to finite automata, operating on finite words (i.e. finite length data sets). This is the case of tree-based coding used in genetic programming, meaning that it is not powerful enough to handle the full generality of boolean queries.

Extended rewriting systems rely on *Rabin automata* [15] that are able to recognize infinite words described by ω-rational expressions which therefore fit the power requirement. Unfortunately, a system directly adapted from this paradigm needs a precise definition of its *alphabet*, i.e. the set of terminal *letters* that trigger state transitions. Since ELISE operates on natural language, these terminals would be terms or senses. If it were possible to define properly such an alphabet, the number of terminals would be huge or even undefined, if new terms are to be recognized dynamically.

The classical tree coding fits the first requirement very well, as it allows for subtree replacement or pruning, harmless terminals and tree node changes, or subtree swappings. When trees are coded as lists —thanks to a postfix representation, where terminals appear first and tree nodes appear last—, genetic operations become list cutting, splicing and replacement. (Such coding is a particular case of concatenative language.)

The descriptive power of these list-coded programs can be enhanced to equal that of Rabin automata by providing them with looping capabilities to fulfill the second requirement. The result is close enough to the classical tree representation to make it possible to reuse an important part of the theoretical and practical tools developed throughout the history of genetic programming.

3.2 Language structure

Generality and description capacity is a common problem in GP program coding. A generic set of tools was therefore developed to handle this kind of representation, independently of the text-retrieval context[14]. A library was written that provides a generic parser, and memory management systems —necessary in a stack-based language and execution tools. This allowed to reuse such coding to evolve rules in other contexts. In this model, programs are lists, containing either atoms or sublists. Two types of atoms are available: instructions, taking arguments and putting back results on a stack and character strings, that are simply added onto the stack when the program is run. No distinction is made between data (lists of strings, for instance) and program parts.
A program written with this language may therefore look like:

```
[ [ "hickory" "dickory" "dock" ] <DUP> <CAT> ]
```

If strings are delimited by quotes, instructions by angle brackets and lists by square brackets, if the `DUP` instruction duplicates the top item of the stack and the `CAT` instruction concatenates two lists, executing this program would put the list `[ "hickory" "dickory" "dock" ]` on the stack, duplicate it and concatenate the two identical lists.
After execution, the top stack item would contain the list:

```
[ "hickory" "dickory" "dock" "hickory" "dickory" "dock" ]
```

The lax syntax of concatenative languages allows for the construction of syntactically valid programs from simple operations such as cut and paste or replacement, thanks to the fact that on the stack, data and instruction code are similar and interchangeable. However, this has a drawback: some programs produced this way can not be guaranteed to run. But evolutionary optimization has a wide range of tools to deal with invalid individuals, and allowing for such transgressions in many cases actually benefits to the efficiency of optimisation algorithms.

The instruction set is specifed on a per-application basis, within a general framework. Tools and information necessary to design efficient genetic operators and "repair" systems are available using instruction signatures (arity, input and output type requirements).

3.3 Instruction set

If the representation is general enough to be usable in most genetic programming cases, the necessary adaptation to the specificities of the problem lies mainly in the creation of an appropriate set of instructions. While wide enough to allow for all meaningful rewriting method to be evolved, it must be specific enough so as to limit the size of the representation space.

In most applications, basic stack operations are needed (e.g, duplication or deletion of an element on the stack, list cutting and pasting operations, etc...). Other instructions need to be tailored to operate on the particular data the genetic algorithm deals with.

In ELISE, this mainly consists in terms and senses or query parts. So semantic instructions (sense or concept lookups in a semantic network, returning sets of terms) and tree operations (since queries have a tree structure) have been developed to handle this data.

Semantic instructions actually reflect the different flavours of term expansion that can be built over a semantic network. As these depend heavily on the actual capabilities of the semantic network used, they will not be listed here, but among them are generally available: several flavours of synonymy, hypernyms and hyponyms, and sometimes meronyms and holonyms when those are defined.

Tree operations consist of:

- conditionals, based on a particular value of the root node,
- splitting instructions, capable to break a tree into subtrees,
- grouping instructions, that join two subtrees into a single one with a given root node value, one of the boolean operators used by the search engine.

However, the main purpose of this language, compared to classical tree structures, is to obtain more descriptive power, which is usually provided by looping or recursive instructions. However a completely general loop instruction (i.e, *do* `instructions` *while* `condition`) considerably enlarges the description space of the coding, which is also the search space of the ELISE learning machine.

A *mapping* instruction is used instead. This restricted version of loops takes advantage of the fact that there is no difference between data (terminals) and instruction code : the *mapping* instruction applies a program to each and every node of the list.

For instance, `[ [ "hickory" "dickory" "dock" ] [ "mouse" <#AND> ] <MAP> ]` executes the subprogram `[ "mouse" <#AND> ]` on each element of the first list, to produce the new list:

```
[[ "hickory" "#and" "mouse" ][ "dickory" "#and" "mouse" ]
[ "dock" "#and" "mouse" ]]
```

where `"#and"` refers to the operator used by the boolean search engine.

3.4 Derived rewriting rules

Modules such as the one described above may perform adequate transcriptions for the task at hand, but there is not much chance that a single module represents the solution that is sought. Therefore, following the Parisian approach[4], many modules are merged into complete rewriting rules through a set of templates that perform various combinations of two or three modules via a set of boolean operators available in the underlying search engine (for example "AND," "OR," "NO" and possibly "NEAR" or "LIKE" if they are available). Plausibly meaningful combinations of evolved modules can be randomly built this way, without *a priori* preference for any particular combination. From a higher viewpoint, modules contain evolved user-specific knowledge, while templates represent possible combinations of elementary rewriting procedures that use the boolean system.

Templates need some *a priori* designer input into the learning system and need to be carefully constructed. But they enable the human supervisor to finely tune the system behaviour and its reaction to query structures, as well as to adapt to particular quirks of the underlying search engine. Obviously, the quality and the number of templates available influences the performance of the system.

4 Evolving a user profile: GP genetic operators

The paradigm of genetic algorithms was originally developped to deal with bit-string individuals, on which the definition of genetic operators —mutation and crossover— is straightforward. However, when working on more complex genome structures, an important part of the EA performance is linked to the accurate tailoring of these operators to the problem that needs to be solved.

As explained earlier, a module (the ELISE genome) is a small program part. Even with the lax syntax that is adopted, some of the modules fail to run. While this is desirable, to move more easily between basins of attraction in fitness landscape, the occurrence of such individuals should remain low. Consequently, genetic operators must be designed in order to produce valid structures most of the time. This means essentially taking into account instruction arity and input types when doing mutations or crossover and "repairing" discrepancies when needed by introducing extra arguments or ignoring some, as required to keep the overall input and output signature of genetically modified (mutated or recombined) instructions.

Therefore, three types of mutation operators are used, with distinct probabilities:

- Local, intra-class mutation
- Local, cross-class mutation
- Global (structural) mutation

The intra-class mutation operates on atoms, albeit in differents ways depending on the type of the atoms. When it operates on strings, it replaces a term by another connected to it in the semantic network used. When instructions are mutated, it replaces one instruction by another with the same prototype: same number and types of required arguments and same number and types of returned values.

The cross-class mutation changes an instruction into another, with a different prototype or turns an atom into another atom of a different sort altogether: in other words it changes an instruction into a terminal or the opposite. These operations are followed by a specific repair phase, aiming at keeping input and output signatures coherence. For instance, replacing an instruction taking two arguments with one taking three, the first of which being a string, will call for the insertion of an additional string, to preserve input consistency. This string comes from a bank of terms used in the queries.

The global mutation changes the structure itself of the genome, by replacing an atom by a list or the reverse. When mutating a string, as no prior arity information is available, no reparation is attempted (resulting in numerous infeasible individuals). When mutating an instruction, conserving the number of arguments and results is attempted. This calls upon the notion of "arity" of a list, defined —in much the same way as for instructions— as the number and types of objects needed on the stack for a correct execution. However, since complex switches and conditions can be implemented in a subprogram, "arity" is not always unique and generally tricky to determine, resulting in possibly incomplete repair.

Following the same idea, two types of crossover are used. The local version will not descend into sublists (from a functional point of view, these are subprograms), leaving them unchanged, while the global version will also apply a crossover in sublists when possible. These crossovers rely on the same mechanisms as those used for bit strings (remember that genomes are lists, too): cut points are positioned randomly along the two parents' lists and list sections are exchanged in alternance. One of the modified lists is then taken as result of the crossover.
Here is an example of these operators in action for the two following modules:

```
[ <VOCGEN> "metabolic disorder" <VOCSYN> <#AND> ]
[ [ <SPLIT> "acetic acid" <VOCGEN> <#AND> <#OR> ] <IFOR> ]
```

The first module applies `<VOCGEN>` (generalisation operator) to the query terms (that are by default on top of the stack) and puts the result back onto the stack. Then, `"metabolic disorder"` is added on the stack and `<VOCSYN>` (synonym expansion) is applied to it. Finally an `#<AND>` operator is applied to the two results. All in all, only documents containing terms common to the generalisation of the original query and metabolic disorder synonyms will be retrieved.

The second module restricts the set of documents retrieved by the second part of an `<#OR>` query to those having to do with generalisations of "acetic acid."

A crossover between both modules could read:

```
[ [ <SPLIT> "acetic acid" <VOCGEN> <#AND> <#OR> ] <IFOR> <VOCSYN> <#AND> ]
```

A mutation on the resulting module could be:

```
[ [ <SPLIT> "paramyxoviridae infections" <VOCGEN> <#AND> <#OR> ] <IFOR>
<VOCSYN> <#AND> ]
```

4.1 Fitness function and user interaction

One of the main problems encountered in interactive EAs is to build a reliable and appropriate fitness function out of user input, i.e. data that by essence is relatively noisy and not always translatable into numeric values. In the context of ELISE, a simple measurement of the "user satisfaction" quality was available as the time spent browsing documents among those proposed in a result list. In a Parisian perspective, this is a global efficiency measurement, from which a local fitness value must be derived for each module.

However, in test settings, this "spent time" notion was not available and was replaced by a formula based on the *recall rate* and the *precision* of rules[1], as these are quantities commonly used to evaluate the performance of search engines and were easily computable with the used datasets. In effect, the fitness of a module is thus the ratio of $Recall\_rate + \alpha * Precision$ over the total number of documents returned for rules using this module.

4.2 Initialisation and parameters

To avoid a "slow start" syndrome, where the 30 or so first queries return very frustrating answers, as the user's profile is still in the first stages of evolution, a minimal (at least 30%) proportion of the initial modules is created from an initialisation template, containing modules performing "reasonable" rewritings, such as thesaurus-based term expansion, boolean form weakening, sense disjunction, etc... The rest of the population is made of randomly generated modules, with a preset ratio of instructions over strings, and subprograms depth.

To cope with noisy fitness evaluations and to stay within the Parisian paradigm, a slow evolution process is applied: only a small part of the population (the 15% individuals of lower fitness) is updated with each generation, with a 80% probability of Xover and using a very high mutation rate (at much as 30% local intra-class mutations per gene and only around 5% local cross-class and global mutation) to improve "creativity."

Those figures come from the fact that most functional modules contain around 5 genes (below, they do not get high enough rewards, and above, they rapidly become infeasible).

The typical population size used in experiments is 50 individuals, from which an equal number of rules is derived.

5 Experiments and analysis

Tests presented below have been performed automatically on the Cystic Fibrosis Database (CFD)[2], using the SWISH++ [19] basic boolean search engine. The semantic operators are based on a medical oriented thesaurus: MeSH [13].

[1] The recall rate is the percentage of returned documents that match the target with respect to the total size of the target. The precision is the percentage of relevant documents in the document set, returned by the system.

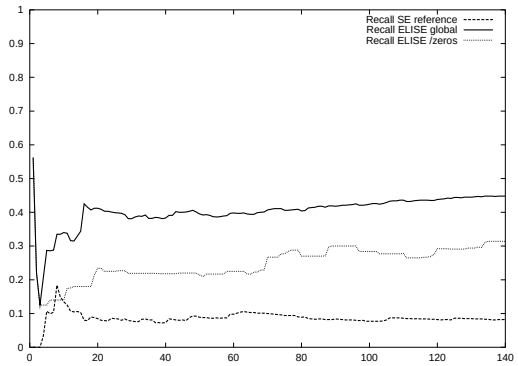

Fig. 2:
Cumulated mean recall rates of the underlying boolean search engine (dashes, bottom curve), of ELISE (solid line, top curve) and of ELISE for queries for which the underlying search engine returns no answer (dots, middle curve).

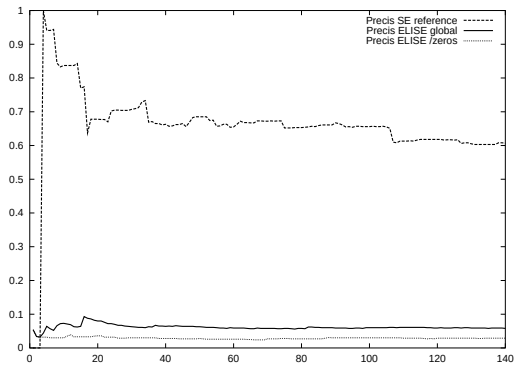

Fig. 3:
Cumulated mean precisions of the underlying boolean search engine (dashes, top curve), of ELISE (solid line, middle curve) and of ELISE for queries for which the underlying search engine returns no answer (dots, bottom curve).

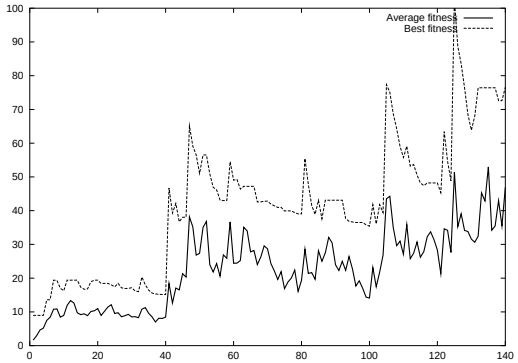

Fig. 4:
Evolution of the fitness of the modules : best individual (dashes, top curve) and mean fitness for the whole population(solid line, bottom curve).

Of course, it is very difficult to automatically test such an interactive system (especially with respect to the "satisfaction of the user" quantity). Results remains however interesting in the sense that they allow to gather statistics rather easily. The CFD test consists of 1239 documents published from 1974 to 1979 on a specific topic (Cystic Fibrosis Aspects) and a set of 100 queries with the respective relevant documents as answers. For each query, the global recall rate and precision of the system can be precisely measured by comparing the returned

document set to the relevant registered answers that have been determined by experts of the field.

Figure 5 presents three curves: the cumulated mean[2] recall rate of the basic boolean search engine (less than 10%), the cumulated mean recall rate of ELISE using the same boolean engine (around 45% after 18 queries only and constantly growing until the end of the test of 140 queries, i.e. 140 generations of the EA). A learning behaviour can be seen on this curve, resulting on a definite increase of the recall rate.

This behaviour is even more obvious on the middle curve, showing ELISE results on queries where the underlying boolean search engine returns no relevant document: even in this extremely unfavourable case, ELISE is able to obtain interesting recall rates.

Figure 5 presents the same set of cumulated mean curves but for the precision measurement. ELISE comes up with a much lower precision than the basic seach engine, which is an expected result, showing that diversity is increased: all modules of the user-profile need to be evaluated at least once, even if they are the result of a recent mutation. The fact that they may very well return random answers lowers the global precision of ELISE, but knowing that the module contains bad genes is a crucial information to evolve the whole system.

Figure 5 shows the evolution of best and mean fitness of the modules: the fitness function is very noisy, even if it is itself a mean evaluation of the efficiency of a module over several evaluations.

Some of the best modules of the final population (after 140 generations) of the run that produced the curves are explained below.

[`"artery, ciliary" <#OR>`] Returns documents obtained from the original query, plus documents about "ciliary arteries." This is a generalisation (extension of the scope) of the original query.

[`<#NOT> "ananase" <SPLIT>`] Performs a search for documents about "ananase," while ignoring the original query (!)

[`"arteries, ciliary" <#AND>`] Among documents obtained from the original query, this module returns only those about "ciliary arteries." This is a restriction of the original query.

[`"artery, ciliary" <VOCGEN>`] Performs a search for documents having to do with "ciliary arteries" or generalisations of this concept, in the set of documents returned by the original query.

[`"dayto anase" <#OR>`] Performs a search for documents about "dayto anase," and appends them to those returned by the original query.

[`<SPLIT> "bromelain" <#VOID>`] Uses the implicit operator of boolean search engines (that is, generally a loose "and") to restrict the scope of the right part of the original query (i.e. at the right of the root boolean operator).

The evolved modules, along with the submitted queries, have been presented to medical practitioners. Their analysis shows that after 140 generations, the modules contain a whole set of enzymes that are used as anti-inflammatory agents, related to CF only at a secondary degree. The recurrent apparition of the term "ciliary arteries" is stranger, although a hypothesis can be formulated:

[2] This is the mean value computed on all queries tested so far, implying that the first values of the curves are not significant, the size of the sample being too small.

ciliary arteries are localised in the eyes, and ocular arterial disease is frequent in diabetes, itself frequent in CF.

This far-fetched explanation, plus a couple of other such findings seem to show that ELISE would exhibit some "lateral-thinking" abilities that were sought in this research. This same conclusion also seems to arise from a global analysis of the terms that appear in the modules, as shown in the tests below.

In order to evaluate more precisely the character strings that are evolved in the user profile after a given set of questions the questions of the CFD have been classified in several sub-classes by the same medical practitioners (see table 1).

Set n^o	Classification 1: CF afflictions and consequences by organ or function
10	respiratory system
11	epithelia or secretions in the respiratory tract
12	infectious pathologies and immunologic mecanisms in the lung
13	treatment of respiratory system damage
14	other respiratory afflictions
20	gastrointestinal and hepatic consequences of CF
21	pancreatic pathologies
22	other gastrointestinal and hepatic pathologies
30	physiologic and biochemical consequences of CF in other organs
40	epidemiology and genetics of CF
50	diagnosis of CF

Set n^o	Classification 2: specialities
10	diagnosis of CF or of CF-related pathologies
20	epidemiology and genetics of CF
30	clinical features of CF
40	biochemical and physiologic manifestations of CF
41	in the respiratory system
42	other
50	CF treatments
51	treatment of pulmonary manifestations
52	treatment of digestive consequences
53	other treatments
60	other pathologies and manifestations

Table 1: Query sets classification key: subsets of the CFD and sub-subsets

Two types of tests have been performed, see table 5:

1. with a set of questions presented to ELISE, restricted to a specific sub-class,
2. by first evolving the profile during 100 generations on the whole set of questions of the CFD, except a given sub-class of questions (learning set) and then specialise the profile on this sub-class during 50 generations (test set).

Evolved terms have been sorted in 4 relevance categories, see table 5. This second analysis seems to confirm that the ELISE system hosts unexpected keywords in its modules, that might potentially provide lateral thinking capabilities to conventional search engines.

6 Conclusion and future works

In this paper, research was carried by adding ELISE on top of an extremely basic boolean search engine, meaning that results discussed above can only be

Test run number	Run 1	Run 2	Run 3	Run 4	Run 5
Learning set[1]	1-13	1-12	1-!12	2-50	2-!50
Number of terms in learning set	14	14	85	21	81
Test set[1]	1-13	1-12	1-12	2-50	2-50
Number of terms in test set	14	13	13	21	21
Recall rate	38.1	38.4	42.3	33.3	35.4
Precision	9.8	10.4	12.2	12.0	9.1
% of terms directly related to the test set	19.2	28.6	26.6	20.0	28.6
% of terms undirectly related to the test set	19.2	57.1	40.0	10.0	42.8
% of terms related to the CF but not the test set	11.6	0	20.0	30.0	5.7
% of unrelated terms	50.0	14.3	13.3	40.0	22.8

Note: The "N-S" notation means "Class N, subset S." "N-!S" means "Class N, all the CFD *except* subset S," see table 1.

Table 2: Runs performance and relevance classification of the vocabulary extracted from final modules

improved, not only in terms of recall rate but also in precision if a state of the art engine is used by ELISE.

Right now, however, tests are quite promising in that the performance of the boolean engine is quite enhanced while at the same time rewriting queries with related terms that did not appear in any previous request. Evolution takes place at a reasonable pace which is compatible with a human user (nice results begin to show up after around 20 queries).

ELISE is currently being ported to ULIX[22], the state of the art search engine of Novartis-Pharma. Future work will consist in analysing and tuning ELISE in real-world conditions and with real users (among others, diversity of the relevant returned documents will be precisely analysed, as well as systems response time and load issues). The conditions being different, it is very likely that this feedback will bring many changes to the present prototype. Future papers on ELISE will discuss this issue.

Acknowledgements

The authors are grateful to Thierry PROST (M.D. PhD), for his educated analysis of the innards of ELISE and to Gilles ROGER (M.D. PhD), for his help as Cystic Fibrosis specialist.

References

1. E. Cantu-Paz and C. Kamath, "On the use of evolutionary algorithms in data mining," In Abbass, H., Sarker, R. and Newton, C. (Eds.) *Data Mining: a Heuristic Approach*, pp. 48-71. Hershey, PA: IDEA Group Publishing, 2002.
2. Cystic Fibrosis Collection, `http://www.sims.berkeley.edu/~hearst/irbook/cfc.html`
3. Y. Landrin-Schweitzer, E. Lutton, P. Collet "Interactive Parisian GP for an Evolutionary Search Engine", submitted.
4. P. Collet, E. Lutton, F. Raynal, M. Schoenauer, "Polar IFS + Parisian Genetic Programming = Efficient IFS Inverse Problem Solving," In *Genetic Programming and Evolvable Machines Journal*, Volume 1, Issue 4, pp. 339-361, October, 2000.

5. A. Dix, "Interactive Querying, locating and dicovering information," *Second Workshop on Information Retrieval and Human Computer Interaction*, Glasgow, 11th September 1998, `http://www.hiraeth.com/alan/topics/QbB/`
6. EMBASE, the Excerpta Medica database Elsevier Science, Secondary Publishing Division, New York
7. D. J. Foskett. "Thesaurus," *Readings in Information Retrieval*, K. S. Jones, P. Willet, M. Kaufmann Publishers, San Fransisco, 1997
8. A.A. Freitas, "Data Mining with Evolutionary Algorithms: Research Directions," AAAI Worshop, Tech. Report WS-99-06, The AAAI Press, 1999.
9. A.A. Freitas, "A survey of evolutionary algorithms for data mining and knowledge discovery," in: A. Ghosh and S. Tsutsui. (Eds.) *Advances in Evolutionary Computation*, Springer-Verlag, 2002.
10. M. D. Gordon, "Probabilistic and Genetic Algrithms for Document Retrieval," *Communications of the ACM* 31, pp 1208-1218, 1988.
11. J.-T. Horng and C.-C. Yeh, "Applying Genetic Algorithms to Query Optimisation in Document Retrieval," *Information Processing and Management* 36, 2000
12. Y.-H. Kim, S. Kim, J.-H. Eom and B.-T. Zhang, "SCAI Experiments on TREC-9," Proceedings of the *Ninth Text REtrieval Conference* (TREC-9), pp. 392-399, 2000.
13. "MeSH (Medical Subject Headings), a controlled vocabulary thesaurus" `http://www.nlm.nih.gov/pubs/factsheets/mesh.html` National Library of Medicine (NLM), National Institutes of Health, Bethesda, Maryland
14. Y. Landrin-Schweitzer, "OKit, a Virtual Machine and Compiler for Concatenative Languages." `http://varkhan.free.fr/Software/OKit`
15. R. McNaughton, "Testing and generating infinite sequences by a finite automaton," *Information and Control*, vol 9, 1966, pp 521–530.
16. R. Poli and S. Cagnoni, "Genetic Programming with User-Driven Selection : Experiments on the Evolution of Algorithms for Image Enhancement," in *2nd Annual Conf. on Genetic Programming*, pp 269-277, 1997.
17. D.-H. Shin , Y.-H. Kim, S. Kim, J.-H. Eom, H.-J. Shin and B.-T. Zhang, "SCAI TREC-8 Experiments," Proceedings of TREC 8, pp. 511-518, 1999.
18. W. M. Spears "Adapting crossover in a Genetic Algorithm," *ICGA '91, International Conference on Genetic Algorithms*, 1991.
19. SWISH++, Simple Web Indexing System for Humans: C++ version, `http://homepage.mac.com/pauljlucas/software/swish/`
20. H. Takagi, " Interactive Evolutionary Computation : System Optimisation Based on Human Subjective Evaluation," *IEEE Int. Conf. on Intelligent Engineering Systems* (INES'98), Vienna, Austria, pp 1-6, Sept 17-19, 1998.
21. The Text REtrieval Conference (TREC) homepage `http://trec.nist.gov/`
22. T. Vachon, N. Grandjean, P. Parisot, "Interactive Exploration of Patent Data for Competitive Intelligence: Applications in Ulix (Novartis Knowledge Miner)," *International Chemical Information Conference and Exhibition*, Nîmes, France, 21-24 October 2001. `http://www.infonortics.com/chemical/ch01/01chempro.html`
23. D. Vrajitoru, "Genetic Algorithms in Information Retrieval," *AIDRI97*, Learning; From Natural Principles to Artificial Methods , Genve, June 1997.
24. D. Vrajitoru, "Large Population or Many Generations for Genetic Algorithms ? Implications in Information Retrieval", In F. Crestani, G. Pasi (eds.): *Soft Computing in Information Retrieval. Techniques and Applications*, Physica-Verlag, Heidelberg, pp 199-222, 2000.
25. G. A. Miller, C. Fellbaum, R. Tengi, P. Wakefield, "WordNet, an Electronic Lexical Database for the English language: `http://www.cogsci.princeton.edu/ wn/`" Cognitive Science Laboratory, Princeton University.
26. J. Yang, R. R. Korfhage and E. Rasmussen, "Query Improvement in Information Retrieval using Genetic Algorithms: A Report on the Experiments of the TREC project", The first Text Retrieval Conference (TREC-1), 1993.