BLID: an Application of Logical Information Systems to Bioinformatics *

Sébastien Ferré and Ross D. King

Department of Computer Science, University of Wales, Aberystwyth,
Penglais, Aberystwyth SY23 3DB, UK
Tel: +44 1970 621922, Fax: +44 1970 622455, {sbf,rdk}@aber.ac.uk

Abstract BLID (Bio-Logical Intelligent Database) is a bioinformatic system designed to help biologists extract new knowledge from raw genome data by providing high-level facilities for both data browsing and analysis. We describe BLID's novel data browsing system which is based on the idea of Logical Information Systems. This enables combined querying and navigation of data in BLID (extracted from public bioinformatic repositories). The browsing language is a logic especially designed for bioinformatics. It currently includes sequence motifs, taxonomies, and macromolecule structures, and it is designed to be easily extensible, as it is composed of reusable components. Navigation is tightly combined with this logic, and assists users in browsing a genome through a form of human-computer dialog.

1 Motivation

Over the last decade many organisms have had their genomes fully sequenced. For example, the 17 chromosomes of the Baker's Yeast (*Saccharomyces Cerevisiae*) have been sequenced, and they code for about 6000 proteins [Gof97]. Yeast is one of the best studied of all organisms, yet about 30% of all its proteins still have not yet any known function. For other organism the percentage is higher. Therefore, one of most important current problems in biology is to discover the function of these proteins that are currently unknown, and to better understand the function of those that are putatively known. To help do this biologists require new and powerful tools to browse and compare bioinformatic databases, and so extract the wealth of information hidden in them.

Many bioinformatic databases are publicly available: e.g., the whole genome of the Yeast is accessible from MIPS[1]. Also, many tools are available: e.g., PSI-BLAST for comparing sequences, ExPASy for computing physical properties of proteins. However, these data sources and analysis tools are disconnected from each other, making it very difficult to perform genome-wide analysis. Moreover, they usually offer limited forms of querying and navigation.

Our aim is to provide biologists with a high-level and integrated interface for browsing and analyzing a whole genome. To do this we first must build a

* This project is funded by the BBSRC grant 21BEP17028.

[1] Munich Information center for Protein Sequences, http://mips.gsf.de/.

secondary database gathering data from different sources, and represent them in a uniform way. We then must define a querying language that fits the needs of bioinformatics, and allows browsing capabilities. In this paper, we focus on the second task. This language can deal with taxonomies of protein functions, with complex sequence patterns (as in Prosite), and with structures (e.g., the transcription of proteins from several RNA parts called *exons*). The need for complex representations and reasoning mechanisms leads us to the use of logics specialized to bioinformatics. Hence the name of our system, BLID, which stands for Bio-Logical Intelligent Database. The term "intelligent" refers to the automated analysis, such as machine learning or data-mining, that will be made available on top of the querying system in the future.

Section 2 discusses the use of Formal Concept Analysis (FCA) for *bio-logical browsing*, and presents Logical Information Systems (LIS) as a theoretical framework for BLID. Section 3 presents a logic for the representation and reasoning of descriptions and queries. Section 4 explains and illustrates how an automatic and non-hierarchical navigation can be combined with logical querying. Finally, Section 5 discusses related works, and Section 6 draws some future directions, especially w.r.t. analyses.

2 Concept Analysis and Logical Information Systems

Formal Concept Analysis (FCA) is a mathematical theory based on ordered sets and complete lattices [Wil82]. A *context* is a triple $(\mathcal{O}, 2^{\mathcal{A}}, d)$, where $\mathcal{O}$ is a set of *objects*, $\mathcal{A}$ a set of *attributes*, and d is a mapping from objects to their description, i.e. a set of attributes. Then, a Galois connection (ext, int) is defined between sets of objects and sets of attributes. For every set of attributes A, its *extent* $ext(A) = \{o \in \mathcal{O} \mid d(o) \supseteq A\}$ is defined as the set of objects whose description contains A (i.e., the answers of A, when A is seen as a query); and for every set of objects O, its *intent* $int(O) = \bigcap_{o \in O} d(o)$ is defined as the set of attributes shared by all objects in O. Pairs of related extent and intent, such as $(ext(A), int(ext(A)))$ or $(ext(int(O)), int(O))$, are called *concepts*, and form together a complete *lattice of concepts* when ordered by set inclusion on their extent (or equivalently on their intent). Numerous works have shown the usefulness of this concept lattice for information retrieval combining querying and navigation [GMA93,FR03], learning and data-mining [GK00,FR02b].

This applicability of FCA to information retrieval and learning is the basis for our choice of its use as a theoretical foundation. In BLID, objects are the ORFs[2] of some organism (the Yeast in the rest of this paper). However, simple sets of attributes are not an expressive enough language for object descriptions and queries. For example, a protein sequence can not be made an attribute as it is different for each gene. A set of predefined patterns could be used instead (e.g., from Prosite), but information about the gene would be lost, and querying with

[2] ORFs (Open Reading Frames) are segments of DNA in chromosomes supposed to be transcribed and translated into proteins. An ORF coincides with the coding region of a gene when this protein has directly been observed.

new patterns would no longer be possible. We wish to preserve all information and to dispose of an open query language.

Logical Concept Analysis (LCA, [FR03]) is an extension of FCA that allows for the replacement of sets of attributes $A \in 2^{\mathcal{A}}$ by formulas $f \in L$ of a logic. The formulas need only be ordered by a *subsumption* relation $\sqsubseteq$, and this must form a lattice. Logical Information Systems (LIS, [FR03]) are founded on LCA and are characterized by: (a) an object-centered representation; (b) a tight combination of querying and navigation; (c) a logical representation of object descriptions, queries, and navigation links; (d) genericity in the logic for customization.

Section 3 describes the building of a logic that is designed specifically for BLID. This comprises the definition of the language, as well as a few necessary operations: (1) the subsumption $\sqsubseteq$ for ordering formulas according to their specificity/generality[3] ($f \sqsubseteq g$ means f is more specific than g), (2) the conjunction $\sqcap$, and (3) the function *feat* that maps each object description to a set of more general formulas, their *features*. These features play an important role in navigation, which is presented in Section 4. They are generated mainly automatically by the operation *feat*, but can also be introduced at any time manually by users according to their needs.

3 Customized Logic and Querying for Bioinformatics

A logical context is build by creating an object for every ORF of the Yeast's genome. Each object/ORF is described by collecting information from various data sources (see Section 1). For instance, a partial description of the ORF YAL003w, incorporating various data types, is (sequences are shortened):

```
[ name is "YAL003w", nb_atoms = 3138, mol_weight = 22.627e3,
seq is MAS[..]QKL, struc is c(8)a(12)c(3)[..]b(10)c(5)a(25)c,
some exon is [1,80], some exon is [447,987],
'mfc05/04/02: elongation' ].
```

This description combines different concrete domains: text (name), integer (number of atoms), float (molecular weight), two kinds of sequences (over amino-acids and 3D structures), segments (exons). The first sequence (attribute `seq`) is made of amino-acids, and defines the protein expressed by the ORF. In solution the protein folds to form a specific 3D-shape. This shape, the tertiary structure, is still generally unknown, but it is possible to reliably predict an intermediate structure, the secondary structure [OK00]. This latter structure (attribute `struc`) is represented as a sequence composed of 3 kinds of structure element (helices `a`, sheets `b`, and connecting elements called coils `c`), which can have different lengths (given between brackets after each structure element). The exons are the gene segments, from which an ORF is composed. The last term in the

[3] Notice that the left argument in the subsumption relation is not restricted to be an object description, but can be any query as well as the right argument. This makes it much harder to define such logics, but is necessary for navigation.

description is an element of the taxonomy of functional classes, found in MIPS. (This taxonomy has been used as target classes in machine learning [KKCD00].)

Let the following expression be a query:

```
('mfc05: PROTEIN SYNTHESIS' or 'mfc06: PROTEIN FATE')
and some exon start >= 2 and nb_atoms in 3000..4000 and
seq match N-{P}-[ST]-{P} and not name ends with "w",
```

where the pattern `N-{P}-[ST]-{P}` is the Prosite motif PS00001, which is described as "N-glycosylation site". While none of the terms of this query appears as such in the description of YAL003w, the latter is still an answer of the query. This means that propositional logic is not expressive enough as a query language w.r.t. above description. One could wonder if propositional logic could be made suitable by adapting the object descriptions. The answer is no, because some data types have an infinite number of patterns (e.g., numerical intervals, sequence motifs); and even for data types where this adaptation is possible (e.g., finite taxonomy of functional classes), this would imply redundancy and potential exponential growth of the descriptions.

Our logical language for descriptions and queries can be understood as a propositional logic, whose atoms are replaced by *logical features* belonging to fragments of predicate logic. In fact, this is equivalent to saying that our logic is a controlled fragment of predicate logic, plus some theory about the considered data types. For instance, the logical feature `seq match N-{P}-[ST]-{P}` can be translated into predicate logic by:

$$\begin{gathered}\forall Orf : \exists Start, P1, P2, A2, P3, A3, P4, A4 : seq(Orf, Start) \wedge \\ somesucc(Start, P1) \wedge aa(P1, \text{'N'}) \wedge succ(P1, P2) \wedge aa(P2, A2) \wedge \\ A2 \neq \text{'P'} \wedge succ(P2, P3) \wedge aa(P3, A3) \wedge (A3 = \text{'S'} \vee A3 = \text{'T'}) \wedge \\ succ(P3, P4) \wedge aa(P4, A4) \wedge A4 \neq \text{'P'},\end{gathered}$$

given some theory to define the predicate *somesucc* as the transitive closure of predicate *succ*. It should be clear from this example that a customized logic is preferable to predicate logic as a query language. This is more than mere syntactic sugar because the use of specialized logics enables us to make the computation of subsumption decidable, simpler, and more efficient (remembering that subsumption needs to be applied between queries as well).

Building such a logic from scratch would be a tedious task because of the number of different concrete domains. Moreover, this would make it difficult to extend, or to reuse, parts of existing customized logics. In order to favor modularity and re-usability we apply the principles of *logic functors* [FR02a], which enable us to build complex logics by simple composition of smaller logic components, the *logic functors*. Essentially, a logic functor is a function from logics to logics, where logics are modeled as abstract types encapsulating both representation and reasoning. Most logic functors are reused from previous applications, and a few others, specific to bioinformatics, are created (e.g., for protein sequences and Prosite motifs). Due to limited space, we do not give in this paper formal definitions for the language, the semantics, and the subsumption of logic functors. For those interested, they are available for a few functors in [FR02a].

Figure 1 shows the way the logic functors are composed as a tree. Each node is a logic functor that is applied to the logics composed from its sub-nodes. At the root of the tree we recognize the propositional parts of the logic. The remainder of the tree describes the logic of features that replace the usual atoms. Features are used to represent both object descriptions and query terms. They are essentially conjunctions of terms taken in concrete domains. The functor `Sum` allows us to easily combine any number of concrete domains, and facilitate extensibility of the logic. Finally, the functor `AIK` (named after the epistemic logic *All I Know*) enables to apply the Closed World Assumption on object descriptions [FR02a].

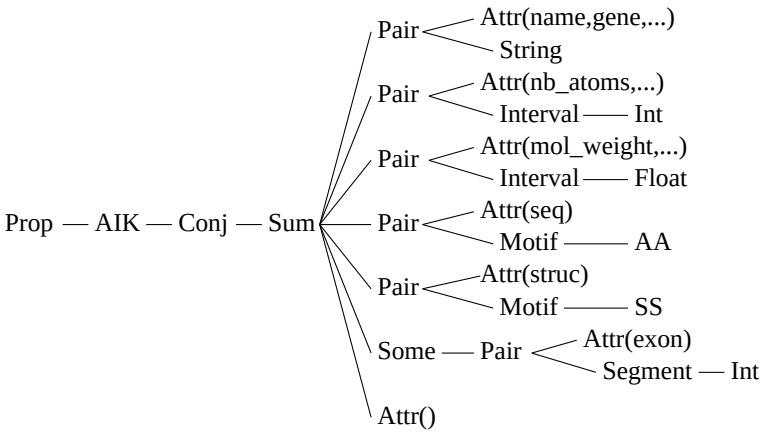

Figure1. The BLID's logic represented as a tree of logic functors.

The logical context of chromosome A contains 108 objects (ORFs), from which 4867 features are extracted. This makes an average of 161 features per object, and 3.5 objects per feature (feature sharing). As this context is extended to the whole genome (6141 ORFs), the number of features per object remains constant, and the sharing increases, which results in a total number of features of around 60,000. In such a large context, it becomes intractable to compute the concept lattice. However, it is important to provide users with navigation as they cannot remember by heart the function names or the Prosite motifs, and also because, given some previous query, it is difficult to guess relevant features to refine it. Section 4 develops an interactive and incremental way of building such queries: *logical navigation*.

4 Logical Navigation

The idea of navigation is to help users build their queries, and to enable them to form overviews on the data. In the domain of concept analysis, navigation is usually realized by a direct browsing of the concept lattice, or some part of it [GMA93]. However, this lattice becomes rapidly very large, and we prefer to

realize navigation by a form of human-computer dialog [FR03], as this gives more freedom for controlling the amount of answers.

To navigate from one query/concept to another, users specify query *increments* with *exclamatory commands*, such as `! name ends with "w"`, and they get suggestions for increments with the *interrogative* command "`?`". These suggestions are found among the features that have been automatically extracted from object descriptions by the logical operation *feat*. For example, in the context made of all ORFs of chromosome A, this command gives the following result:

```
[1] ?                                          What is there ?
  100  ! struc                          100 ORFs with known 2nd structure !
  101  ! 'MIPS function'                      101 ORFs with function !
  108  ? name                                    What kind of name ?
  108  ? some exon                               What kind of exon ?
  108  ? seq                                     What kind of sequence ?
  108  ? mol_weight in ..                        What kind of mol. weight ?
  108  ? nb_atoms in ..                          What kind of nb. of atoms ?
108 object(s)                                  There are 108 selected ORFs.
```

The system returns not only exclamatory suggestions (query increments), but also interrogative suggestions. These can be understood as "questions as answers to questions", and their purpose is to provide more concise answers, as without them many exclamatory suggestions would possibly replace each interrogative suggestion. These are called *view increments*, because they allow to focus on one kind of features. For instance, the user can select the command "`? nb_atoms in ..`" in order to focus on the number of atoms:

```
[2] ? nb_atoms in ..                           What kind of nb. of atoms ?
    5  ! nb_atoms = 2****          5 ORFs with nb. of atoms in [20000,30000[ !
   22  ! nb_atoms = 1****         22 ORFs with nb. of atoms in [10000,20000[ !
   81  ! nb_atoms = 0****            81 ORFs with nb. of atoms in [0,10000[ !
108 object(s)                                  There are 108 selected ORFs.
```

The formula `nb_atoms = 2****`, which means the number of atoms is comprised between 20,000 and 29,999, is a feature automatically generated by the functor `Int` to make the navigation more progressive than a flat set of values. This makes the answers look like a histogram, as values at the left of increments are the number of objects they would select (support). With query languages such as SQL or Prolog, one would either get a flat list of all ORFs along with their exact number of atoms, or have to ask an aggregative query for all relevant intervals; which are difficult to know without prior knowledge of the range and the scale of the attribute (which can change according to the working query).

Coming back to command `[1]`, we see that the feature `'MIPS function'` appears as a query increment, because it is not supported by all objects. However, we would expect it to be as well a view increment, focusing on the functions of ORFs. In fact, exclamatory suggestions can often be combined with an interrogative command.

```
[2] !? -l -i 'MIPS function'          Select ORFs with function ! What kind of
functions ?
```

```
   22  ! 'mfc01: METABOLISM'
    3  ! 'mfc02: ENERGY'   -> 'mfc01: METABOLISM'
[..]
   39  ! 'mfc99: UNCLASSIFIED PROTEINS'
101 object(s)                                    There are 101 selected ORFs.
```

The 101 objects are selected and functional classes are listed in lexicographical order (option `-l`). This shows that about 40% of ORFs are unclassified. Option `-i` displays contextual implications between suggested increments. This enables the user to discover that every ORF in chromosome A that has an energetic function, has also a metabolic function.

5 Related Work

Our logics are similar to Description Logics (DL, [Bra79]), in the sense that formulas are variable-free, and their semantics is based on object sets rather than on truth values. Our attributes are equivalent to functional roles, and our operator `some` corresponds to the existential quantification. The two key differences are the modularity of logic functors, and our focus on concrete domains. Furthermore, it would be possible to define a logic functor implementing a description logic in which atoms could be replaced by formulas of concrete domains; as it has been done with propositional logic. Both DL and our logics could be translated into predicate logic, which is more expressive; however they are more readable, and allow for logical navigation thanks to their compatibility with Logical Concept Analysis. We are also developing in parallel a querying interface in predicate logic (using Prolog) to offer more expressive power to expert users, but at the cost that no navigation is provided.

A project related to ours is GIMS [Cor01], which aims at providing querying and analysis facilities over a genome database. In this project, simple queries can be built incrementally by selecting attributes and predefined value patterns in menus. Canned queries are made available for more complex queries and analysis. We differ in that we have made the choice to give users an open language, knowing that navigation will be available to guide users; even if they have no prior knowledge. It is difficult, if not impossible, to forecast all types of queries that may be of interest in the future.

6 Future Work

Our future work concentrates on providing analysis facilities in addition to querying and navigation. The kind of analysis we are mostly interested in is to discover by machine learning techniques rules that predict the biological functions of ORFs from genomic data (i.e., *functional genomics* [KKCD00]). A first step will be to integrate existing machine learning techniques in BLID. Propositional learners (e.g., C4.5, concept analysis [GK00]) expect kind of attribute contexts, which can easily be extracted from the BLID's logical context by making each feature (e.g., sequence motifs) a Boolean attribute. Inductive Logic Programing

(ILP, [MR94]) expects a representation of examples in predicate logic, which can always be obtained by translating them from our specialized logics.

Ultimately, BLID could be made an Inductive Database [dR02] by unifying various machine learning and data-mining techniques under a unified *inductive query language*. For instance, such a language could allow to ask for "all most general rules predicting whether a protein is involved in metabolism according to its sequence". Such a high-level language would be very helpful to biologists and bioinformaticians, who strive to relate genomic data to biological functions.

A LIS executable for Unix/Linux can be freely downloaded at `http://users.aber.ac.uk/sbf/camelis`.

References

[Bra79] R. J. Brachman. On the epistemological status of semantic nets. In N. V. Findler, editor, *Associative Networks: Representation of Knowledge and Use of Knowledge by Examples*. Academic Press, New York, 1979.

[Cor01] M. Cornell et al. GIMS – a data warehouse for storage and analysis of genome sequence and functional data. In *IEEE Int. Symp. on Bioinformatics and Bioengineering*, pages 15–22. IEEE Press, 2001.

[dR02] L. de Raedt. A perspective on inductive databases. *SIGKDD Explorations*, 4(2):69–77, December 2002.

[FR02a] S. Ferré and O. Ridoux. A framework for developing embeddable customized logics. In A. Pettorossi, editor, *Int. Work. Logic-based Program Synthesis and Transformation*, LNCS 2372, pages 191–215. Springer, 2002.

[FR02b] S. Ferré and O. Ridoux. The use of associative concepts in the incremental building of a logical context. In G. Angelova U. Priss, D. Corbett, editor, *Int. Conf. Conceptual Structures*, LNCS 2393, pages 299–313. Springer, 2002.

[FR03] S. Ferré and O. Ridoux. An introduction to logical information systems. *Information Processing & Management*, 2003. To appear.

[GK00] B. Ganter and S. Kuznetsov. Formalizing hypotheses with concepts. In G. Mineau and B. Ganter, editors, *Int. Conf. Conceptual Structures*, LNCS 1867, pages 342–356. Springer, 2000.

[GMA93] R. Godin, R. Missaoui, and A. April. Experimental comparison of navigation in a Galois lattice with conventional information retrieval methods. *International Journal of Man-Machine Studies*, 38(5):747–767, 1993.

[Gof97] A. Goffeau et al. The Yeast genome directory. *Nature*, 387:1–105, 1997.

[KKCD00] R. D. King, A. Karwath, A. Clare, and L. Dehaspe. Genome scale prediction of protein functional class from sequence using data mining. In R. Ramakrishnan et al, editor, *ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining*, pages 384–389. ACM, 2000.

[MR94] S. Muggleton and L. De Raedt. Inductive logic programming: Theory and methods. *Journal of Logic Programming*, 19,20:629–679, 1994.

[OK00] M. Ouali and R. D. King. Cascaded multiple classifiers for secondary structure prediction. *Prot. Sci*, 9:1162–1176, 2000.

[Wil82] R. Wille. *Ordered Sets*, chapter Restructuring lattice theory: an approach based on hierarchies of concepts, pages 445–470. Reidel, 1982.