Working with biological databases

Nicos Angelopoulos[*] and Georgios Giamas

Department of Surgery and Cancer, Division of Cancer, Imperial College London, Hammersmith Hospital Campus, Du Cane Road, London W12 ONN, UK.

July 25, 2015

Abstract

It has been argued before that Prolog is a powerful platform for research and code development in bioinformatics and computational biology. This position has been based on both the intrinsic strengths of Prolog and recent advances in its technologies. Here we strengthen the case for the deployment and penetration of Prolog into bioinformatics, by introducing *bio_db*, a comprehensive and extensible system for working with biological data. We focus on databases that translate between biological products and product-to-product interactions, the latter of which can be visualised as graphs. This library allows easy access to high quality data in two formats: as Prolog fact files and as SQLite databases. On-demand downloading of prepacked data files in these two formats is supported in all operating system architectures as well as reconstruction from latest data files from the curated databases. The methods used to deliver the data are transparent to the user while they present the data uniformly in the familiar format of Prolog facts.

1 Introduction

Prolog's traditional playground is that of knowledge representation and AI applications on crisp, logical inference and search. In addition to being a research tool in these areas, Prolog implementations have been developing to full fledged general purpose programming environments. These developments have started shaping a role for logic programming in a variety of new areas.

Bioinformatics has been the meeting point of a number of influences since its emergence as a field of study. Being on the intersection of biology, statistics and computing, it has meant that a multitude of languages, systems and

[*]Email: `nicos.agnelopoulos@imperial.ac.uk`
please note, an extended version of this paper is presented in ICLP as technical communication

Database	Abbv.	Description
HGNC	hgnc	HUGO Gene Nomenclature Committee
NCBI/entrez	entz	Nat. Center for Biot. Inf.
Uniprot	unip	Universal Protein Resource
GO	gont	Gene Ontology
Interactions database		
String	string	protein-protein interactions

Table 1: Supported biological databases and data sources.

paradigms has been developed and utilised for bioinformatics research. One of the strongest contestants in this field comes from the statistics community in the shape of the *R* (R Core Team, 2015) language and its Bioconductor (Gentleman et al., 2004) bioinformatics suite. The strength of these statistical tools is on providing a versatile platform that can incorporate a menagerie of paradigms and programming styles.

Using an interface to *R* (Angelopoulos et al., 2013) would be one way to access biological databases via packages such as *org.Hs.eg.db* (Carlson, 2014). However, this approach would increase reliance on *R* and create a further layer of complications. Here, we take a logical approach to incorporating biological knowledge. With the advances in modern Prolog systems in database integration (Canisius et al., 2013; Wielemaker, 2014) and indexing technologies (Santos Costa and Vaz, 2013; Morales and Hermenegildo, 2014) working with big data within Prolog is set to become an important application area for Prolog.

In this paper we describe the capabilities and design structure of an extensible library for working with and managing biological databases. Distinctive features of the package include: on-demand downloading of prepacked databases, and single entry interface for accessing databases in 2 underlying serving mechanisms. Our library focuses on Homo sapiens data and uses high-quality databases.

2 A logical approach to big biological datasets

Data from biological experiments and data codifying biological knowledge have seen a sharp increase in the last decades. Here we will concentrate on two main categories of databases. First, we consider maps of biological products and nomenclatures. Examples include mapping gene synonyms to a standard name and mapping proteins to genes, both of which are many-to-one relations, whereas many-to-many maps can be used to define membership to multiple sets. Maps are conveniently and efficiently implemented as Prolog facts of arity 2.

A summary of the databases supported are shown in Table 1. HGNC (Gray et al., 2015), is our primary gene identifying data source. Each gene is assigned a unique incremental integer identifier and each current identifier is mapped to a unique symbol which is the short name for that gene. Example of symbols are: *LMTK3*, *EGFR* and *BRC1*. We will use `hgnc` to refer to both the database

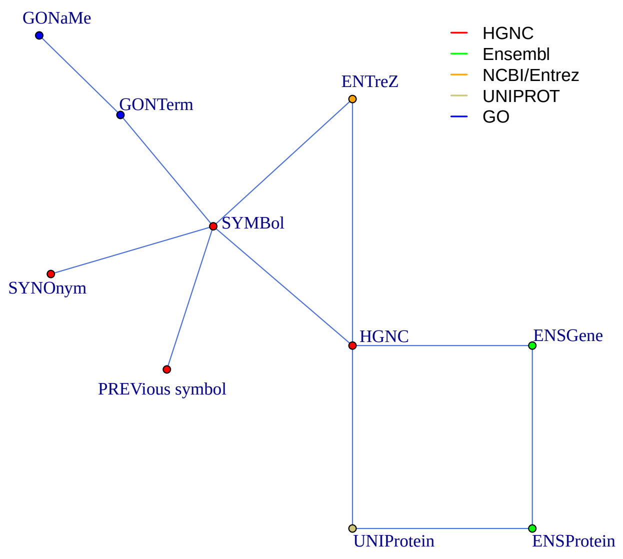

Figure 1: Mapping predicates connect vertices of the displayed graph. The legend shows the database from which the field for each argument in the predicates is drawn from.

and the unique integer identifier field of the database. Symbols are shortened to `symb`. As can be seen in Fig. 1, the HGNC database plays a central role in *bio_db*. Its primary identifier (e.g 19295) connects to protein and gene resources, and its Symbol (short name, e.g. *LTMK3*) connects to gene ontology terms and other naming conventions.

The National Center for Biotechnology Information (NCBI) makes available a large number of datasets (NCBI Resource Coordinators, 2013). Here we only incorporate their unique gene identifier, which is often referred to as `gene id` and was for many year the main way to uniquely refer to genes (here referred to as `entz`.). Uniprot (The UniProt Consortium, 2015) is a curated and well established database of proteins and related information. The relation between proteins and genes is a many to one correspondence. Gene ontology (GO) (The Gene Ontology Consortium, 2000) provides a controlled vocabulary to describe biological knowledge. The basic representation unit in GO are its GO terms. They are connected in a web of referential relations. Each term, in addition to its relative position to other terms, contains a number of genes which are involved in the process characterised by the term. Here we concentrate on

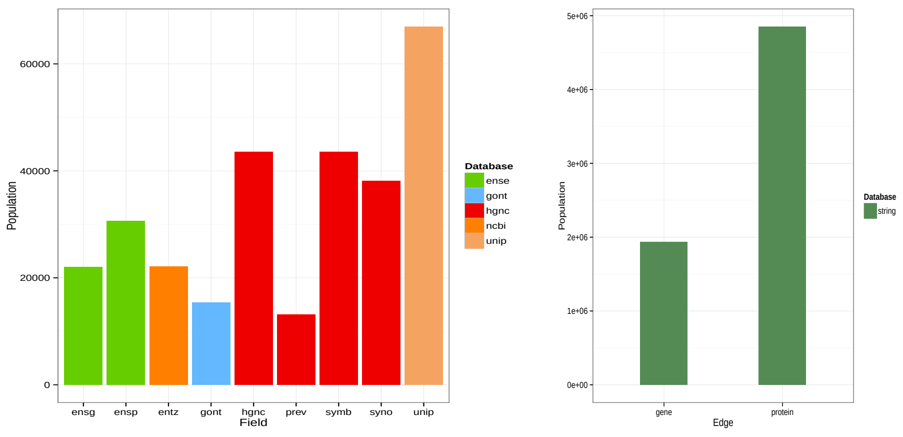

Figure 2: Populations of the main fields in the supported databases. Each bar corresponds to a field in one of the databases. Colours correspond to databases from which the field was drawn from. In the LHS are the fields associated with maps and in the RHS are the String DB edges.

this membership, which defines a many to many relation. Each term contains a number of genes and each gene potentially belongs to a number of terms. String (Szklarczyk et al., 2015) is a comprehensive protein-protein interaction database that incorporates a large number of interactions present in one of a large number of species. Here we are only concerned with the 4850628 interactions of human proteins (Fig. 2 1936162 interactions amongst genes). The database provides an overall integer score in $(0, 1000)$. The closer to 1000 this score is, the stronger the likelihood that the link represents a real physical interaction.

3 Data management

The library presents those facts to the programmer as a unifying level of abstraction. Beneath this level there are two mechanisms via which the data are delivered to the predicates: (a) Prolog fact files and (b) SQLite databases.

3.1 Predicate naming

An example of a map predicate is

```
map_hgnc_hgnc_symb( Hgnc, Symb ).
```

The predicate translates between HGNC identifiers and HGNC symbols. The predicate name consists of 4 components, the first of which determines the type of data, a map in this case, the second, `hgnc`, corresponds to the database and the third, also `hgnc`, identifies the first argument of the map to be the unique identifier field for that database (here a positive integer starting at 1 and with no gaps. The last part of the predicate name corresponds to the second argument, which here is the unique gene Symbol. In the current version of *bio_db*, all tokens in map predicate names are 4 characters long. The abbreviations for the database component are shown in the second column of Table 1 whereas the abbreviations for the database fields are the capitalised parts of vertice's names of Fig. 1. The following query maps an HGNC identifier to a Symbol

```
?- map_hgnc_hgnc_symb( 19295, Symb ).
Symb = 'LMTK3'.
```

3.2 Data serving methods

There are two mechanisms via which the library's data predicates can be stored and served. One is as plain Prolog fact files, and the other is via SQLite databases as implement in the proSQLite Prolog library (Canisius et al., 2013). The former requires in-memory loading, thus requiring more memory and loading time. The main benefit of Prolog facts, is that there are extremely fast particularly when requests for data instantiate the first argument of their call. Memory itself is in our experience not a particular limitation as computer memory is readily available in bioinformatics settings and SWI-Prolog along with most modern Prolog systems are well tuned to dealing with such data. The time taken when loading everything to memory is a more severe limitation particularly in development settings. It might thus be desirable to use SQLite during development and testing and Prolog for when big time consuming searches are required. Switching between the mechanisms for serving the files is done via a simple call to a predicate,

```
bio_db_interface( ?Interface ).
```

All data predicates loaded after such a call will be following the interface method dictated by `Interface`. The following example shows a simple interaction with debugging statements on.

```
?- debug( bio_db ).

?- bio_db_interface( Iface ).
Iface = prolog.

?- map_hgnc_prev_symb( Prev, Symb ).
% Loading prolog db: .../map_hgnc_prev_symb.pl
Prev = 'A1BG-AS',
Symb = 'A1BG-AS1';
Prev = 'A1BGAS',
Symb = 'A1BG-AS1'...
```

3.3 Downloading datasets

The library comes with placeholder code for each supported database table. On first call the relevant datafile is downloaded from the web-server and consulted on-the-fly after the place-holding code is removed. In each new interactive invocation, hot-swapping and then consulting of the relevant and data file will make the data available as facts. The facts are served transparently to the user by the two different technologies detailed above. The downloading of non-installed datasets occurs automatically and transparently to the user. This is triggered by a call to the corresponding data predicate and the actual call is served within the same interaction as demonstrated below

```
?- map_hgnc_symb_hgnc( 'LMTK3', Hgnc ).
% prolog DB:table hgnc:map_hgnc_symb_hgnc/2 is not installed,
                               do you want to download it (Y/n) ?
% Trying to get: url_file(... )
% Loading prolog db: .../hgnc/map_hgnc_symb_hgnc.pl
Hgnc = 19295.
```

The data files are stored in a directory organised into maps and graphs reflecting the two main type of information supported. Within these two subdirectories data are organised as per database of origin. The root of this filestore organisation defaults to the data directory of the library or can be set via an environment variable or by using the `set_prolog_flag/2` predicate. The default location for storing data files is at the level of an SWI-Prolog pack located at `pack(bio_db_repo)`. Each dataset contains a set of house keeping information that shows, among other things, the download and build dates.

```
map_hgnc_hgnc_symb_info(date, date(2015, 4, 28)).
map_hgnc_hgnc_symb_info(map_type, map_type(1, 1)).
map_hgnc_hgnc_symb_info(unique_lengths, c(43592, 43592, 43592)).
map_hgnc_hgnc_symb_info(header, row('HGNC ID', 'Approved Symbol')
```

The Prolog scripts used to download and convert the data are given in the library source code. The overall work-flow normally is as follows: (a) download a remote file to a local date-stamped file, (b) read the downloaded file, (c) produce *bio_db* outputs, and (d) move or link files from downloads directory to loadables directory.

4 Examples

Here we will look into the GO terms of the LMTK3 tyrosine kinase (Giamas et al., 2011). The following code shows how to produce the GO terms, their names and their populations, which are shown in Table 2.

```
lmtk3_go :-
   map_gont_symb_gont( 'LMTK3', Gont ),
   findall( Symb, map_gont_gont_symb(Gont,Symb), Symbs ),
   map_gont_gont_gonm( Gont, Gonm ),
   sort( Symbs, Oymbs ), length( Oymbs, Len ),
```

GO term	GO name	population
GO:0003674	molecular_function	764
GO:0004674	protein serine/threonine kinase activity	340
GO:0004713	protein tyrosine kinase activity	89
GO:0005524	ATP binding	1488
GO:0005575	cellular_component	497
GO:0006468	protein phosphorylation	557
GO:0010923	negative regulation of phosphatase activity	53
GO:0016021	integral component of membrane	200
GO:0018108	peptidyl-tyrosine phosphorylation	131

Table 2: Gene ontology terms and associated GO term names for LMTK3. Third column shows the total number of genes in the GO term

```
   write( Gont-Gonm-Len ), nl, fail.
lmtk3_go.
```

As a second example we combine GO terms with String interactions. For a given GO term we can construct a weighted graph reflecting the interactions from the String database. This is build by first mapping an input GO term to the list of symbols it contains and then collecting all edges amongst these symbols that have a weight that exceeds that of a provided limit. The graph in Fig. 3 shows such a graph for term `GO:0010332` for a minimum weight of 500.

```
go_term_graph(GoTerm,Min,Graph):-
   findall( Symb, map_gont_gont_symb(Gont,Symb), Symbs ),
   findall( Symb1-Symb2:W,  (  member(Symb1,Symbs),
                               member(Symb2,Symbs),
                      edge_string_hs_symb(Symb1,Symb2,W),
                               Lim < W   ),
                                 Graph ).

?- go_term_graph( 'GO:0010332', 500, W ).
```

The software described in this paper is available as an easy to install library for the SWI-Prolog system. Installation can be done within the system with a single call`?- pack_install( bio_db ).`

This will only install the library source code but not the datasets. These will be downloaded transparently and on demand, the first call to a predicate. Alternatively, the core databases can be installed with pack `bio_db_repo`.

5 Conclusions

We have argued that Prolog is a powerful language for building bioinformatics pipelines and that its role can be of crucial importance as biological data is

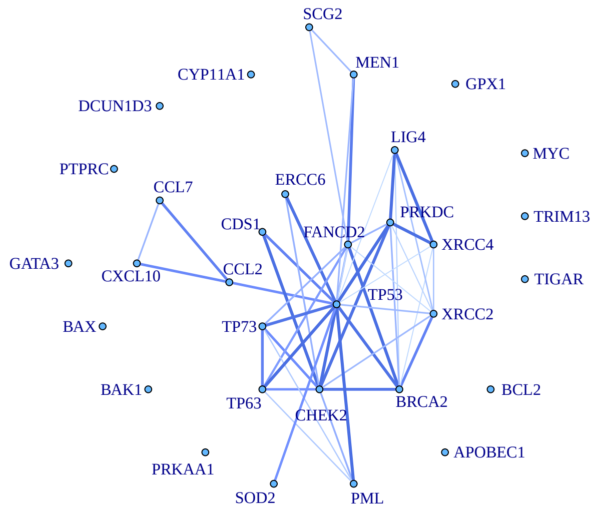

Figure 3: Gene ontology term, GO:0010332: response to gamma radiation. Edges are provided by the String database. Width and darkness of edge colour signify higher belief in the interaction being a real protein-protein interaction

increasingly needed to be viewed as knowledge both in the contexts of analysis and that of statistical inference or machine learning. We presented a library that is easily installed from within SWI-Prolog (Wielemaker et al., 2008). This library presents a convenient and intuitive way for working with biological data. All available data have been sourced from high quality and wherever possible curated databases. The emphasis of our approach is to provide easy of use, via automatically downloading datasets and using code hot-swapping, as well as flexibility by de-coupling data from code and allowing transparent ways of only downloading the necessary datasets. Current work on the library includes extending to other databases and additional database interfaces such as ODBC. Prolog is well suited for research and code development in the areas of bioinformatics and computational biology. The code presented here, can play a strong role in promoting Prolog in these areas.

References

N. Angelopoulos, V. S. Costa, J. Azevedo, J. Wielemaker, R. Camacho, and L. Wessels. Integrative functional statistics in logic programming. In *Proc.*

of PADL, volume 7752 of *LNCS*, pages 190–205, Jan. 2013.

Sander Canisius, Nicos Angelopoulos, and Lodewyk Wessels. ProSQLite: Prolog file based databases via an SQLite interface. In *Proc. of Practical Aspects of Declarative Languages*, volume 7752 of *LNCS*, pages 222–227, Jan. 2013.

Marc Carlson. *org.Hs.eg.db: Genome wide annotation for Human*, 2014. R package version 2.14.0.

Robert C. Gentleman, Vincent J. Carey, Douglas M. Bates, and others. Bioconductor: Open software development for computational biology and bioinformatics. *Genome Biology*, 5:R80, 2004.

G. Giamas, A. Filipovic, J. Jacob, W. Messier, H. Zhang, D. Yang, W. Zhang, B. A. Shifa, A. Photiou, C. Tralau-Stewart, L. Castellano, A. R. Green, R. C. Coombes, I. O. Ellis, S. Ali, H-J. Lenz, and J. Stebbing. Kinome screening for regulators of the estrogen receptor identifies LMTK3 as a new therapeutic target in breast cancer. *Nat Med*, 17:715–719, 6 2011.

K.A. Gray, B. Yates, R.L. Seal, M.W. Wright, and E.A. Bruford. Genenames.org: the HGNC resources in 2015. *Nucleic Acids Res*, 2015.

J.F. Morales and M. Hermenegildo. Towards pre-indexed terms. In *Proceedings of CICLPOPS*, pages 79–92, 2014.

NCBI Resource Coordinators. Database resources of the national center for biotechnology information. *Nucleic Acids Research*, 41:D8–D20, 2013.

R Core Team. *R: A Language and Environment for Statistical Computing.* R Foundation for Statistical Computing, Vienna, Austria, 2015.

Vítor Santos Costa and David Vaz. BigYAP: Exo-compilation meets UDI. *Theory and Practice of Logic Programming*, 13(4-5):799–813, 2013.

D. Szklarczyk, A. Franceschini, S. Wyder, K. Forslund, D. Heller, J. Huerta-Cepas, M. Simonovic, A. Roth, A. Santos, K. P. Tsafou, M. Kuhn, P. Bork, L. J. Jensen, and C. von Mering. STRING v10: protein-protein interaction networks, integrated over the tree of life. *Nucleic Acids Research*, 43(D1): D447–D452, 2015.

The Gene Ontology Consortium. Gene ontology: tool for the unification of biology. *Nat. Genet.*, 25(1):25–9, 2000. URL `http://geneontology.org`.

The UniProt Consortium. Uniprot: a hub for protein information. *Nucleic Acids Res.*, pages D204–D212, 2015.

Jan Wielemaker. SWI-Prolog ODBC interface, 2014. URL `http://www.swi-prolog.org/pldoc/package/odbc.html`.

Jan Wielemaker, Zhisheng Huang, and Lourens van der Meij. SWI-Prolog and the web. *TPLP*, 8(3):363–392, 2008.