Towards a Logic Programming Tool for Cancer Analysis

Towards a Logic Programming tool for cancer analysis

Alice Tarzariol, Agostino Dovier, and Alberto Policriti

Dipartimento di Scienze Matematiche, Informatiche e Fisiche
Università degli Studi di Udine, Italy

Abstract. The main goal of this work is to propose a pipeline capable to analyze a data collection of temporally qualified mutation profiles. A front-end and a post-processor are implemented. The front-end basically transforms the input data retrieved from medical databases into a set of facts. The post-processor allows the user to visualize the computation results as graphs. The reasoning core is based on Answer Set Programming that, allowing to deal with NP or Σ_2^P properties, is capable of deducing complex info from data. The system is modular: the reasoning tool can be replaced by any logic programming tool (e.g., Prolog, ILP) for different kinds of data analysis.

Keywords: Bioinformatics, Big Data Analysis, ASP.

1 Introduction

Cancer is a game of mutation accumulation in the genome of an organism, that is the set of its genes. One of the most delicate aspects of this game is the fact that, while data is produced by today's technology at an unprecedented rate it is, however, extremely noisy data. More precisely, even though *any* mutation might be triggering significant biological processes involved in cancer development, only some of the mutations are classified as "drivers" by experts in the field. The remaining—large—majority of mutations must be classified as "passengers", as they are probably accumulating only as a consequence of the (devastating) side-effects of a more basic biological mechanisms already at work.

Tackling a classification problem for the collection of detected mutations in a tissue requires a multi-faceted approach. In order to build reliable filters suitable to define the *blueprint* of the disease under study it is essential to integrate different kinds of knowledge with brute force analysis.

The main goal of this work is to propose a pipeline capable to analyze a data collection of *temporally qualified* mutation profiles and synthesize a dependency graph. The information inferred by cross comparisons performed "locally" on input data, combined with information coming from external sources, allows to deduce the output graph that should represent a set of putative dependencies.

The idea is to prove the potential of the employed programming technologies in mining data that, when analyzed over realistically long sequences of time steps, rapidly generates (computationally) heavy collections of putative evolution paths.

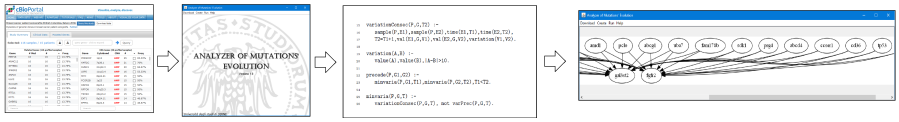

Fig. 1. The proposed pipeline: from the WEB Database data is retrieved and converted into ASP input facts by a Java interface. The ASP engine elaborates the data. Finally, ASP output is converted in Graph format

ASP allows to encode compactly and elegantly problems that belongs to NP or even to Σ_2^P. Encodings allows great flexibility. In particular, the already implemented strategies are parametrically designed and use thresholds fixing conditions on the addition and removal of edges. Moreover, as it is always the case when dealing with biological data, many auxiliary problems—mostly related with I/O—must be faced. The data processing pipeline described here is interfaced with specialized languages and file formats for the field.

The paper is organized as follows: some related work are discussed in Sec. 2, the main ideas of our approach are presented in Sec. 3. A discussion on the data set used an the applications we have developed for extracting/converting data are presented in Sec. 4, and a brief analysis of the results found is shown in Sec. 5. Finally, some conclusions are drawn in Sec. 6.

2 Related work

Cancer data processing is, obviously, a very active and lively field in many scientific environments. We just report here on some work using techniques stemming from the logic programming community. We assume the reader has basic knowledge of logic and logic programming.

2.1 Inductive Logic Programming

Inductive Logic Programming (ILP) has been succesfully used to analyze big data and, in particular, to analyze cancer databases. ILP allows to implement a "structured" form of machine learning, by inferring a first-order theory that "explains" a set of input ground facts (defining extensionally, possibly partially, some predicates). Let us briefly recall the main ideas behind ILP (see, e.g., [14] or [5] for a more general approach to logical and relational learning).

Assume to have a set of positive observations O^+ and a set of negative observations O^-. We can assume O^+ and O^- as two disjoint sets of ground atoms. Assume also to have a (possibly empty) background theory (a set of clauses) P, the general inductive problem is the one of finding a set of hypotheses H (clauses) such that $P \wedge H \models O^+$ and such that for all $\ell \in O^-$ it holds that $P \wedge H \not\models \ell$.

Several strategies have been investigated and implemented by ILP systems to identify sets H made by clauses as much *general* as possible. Generality in this

context can be defined in terms of logical entailment: if clauses r_1 and r_2 are such that $P \wedge r_1 \models r_2$, then r_1 is more general of r_2 w.r.t. P.[1] Of course, this implies that if $P \wedge r_2 \models B$ (for some $B \subseteq O^+$), then $P \wedge r_1 \models B$. Therefore, r_1 is preferable to r_2 in the set of hypotheses H (unless it introduces errors, namely, there is $\ell \in O^-$ s.t. $P \wedge r_1 \models \ell$ and $P \wedge r_2 \not\models \ell$).

In presence of large amounts of data that might be affected from errors since the beginning (e.g., medical data), one might accept a set H that might be not complete, namely it "explains" only a subset G (good) of O^+: $P \wedge H \models G$, and that might be not correct, namely such that $P \wedge H \models \ell$ for ℓ in a subset W (wrong) of O^-. In this case, of course, the ILP algorithm aims at maximizing the size of G and at minimizing the size of W.

An extension of ILP is PILP (probablistic inductive logic programming) [18] where facts (observables) and rules (background theory) can be annotated by a probability. In this case a semantics based on probabilistic inference rules is adopted; the handling of uncertainty is therefore handled in a natural way.

Inductive Logic Programming has been used to analyze cancer data. For instance in [19] the authors participated to a world-wide carcinogenicity prediction competition (organized by the National Toxicology Program in the USA) using the ILP system Progol [15]. In the first round of the competition Progol produced the highest predictive accuracy of any automatic system participating the test.

Progol have been also used in [17] to infer general properties on pancreatic cancer and to allow early detection of this kind of cancer. For instance, one of the inductively computed predicates indicates that the measurements of *CEA* and *Elastase I* are very useful for the detection of pancreatic cancer.

They use data from 438 cases. The classification is based on the observation of the lymph node metastatic status and tumour differentiation status. They use a set of available lab data (e.g., CEA, CA19-9, Glucose, Elastase I, Serum Amylase, ...). After having identified the most promizing features using standard feature selection criteria data is divided in three groups (low, normal, and high), as usually made by domain experts. Each patient record in the database represents an abnormality of a patient, split into positive and negative. Data is translated into a set of facts, and Progol is executed in order to rank the rules by their capability of explaining the database.

In [1] the authors propose an inductive logic programming approach to the problem of modeling evolution patterns for breast cancer. Data is obtained from a cohort of 124 patients at different stages. Data and *background knowledge* are expressed in logic programming. A set of hypotheses built on this knowledge using refinement, least general generalization, inverse resolution, and most specific corrections is computed.

[1] For definite clause programs $\models$ is the standard logical consequence operator. For programs with negation its meaning becomes instead "if in every stable model of $P \wedge r_1$, the clause r_2 holds."

2.2 Answer Set Programming

Answer Set Programming has been used in many Bioinformatics applications. For an overview, see [4]. Among the many of them we would like to point out the approaches to the phylogenetic reconstruction (see, e.g., [7]). The origin of this work comes in fact from the desiderata, presented in the 2016 edition of the conference CILC [3], of extending the ASP tools for phylogenetic reconstruction to build a phylogenetic reconstruction of cancer evolution. If the cancer is related to a mutation of a gene present both in a human and in another specie (e.g., a rat), this may justify a deeper study of the evolution of that cancer in the other specie's body. Or, it can explain why a drug working for the other specie does not work in a human. Of course for doing that we need much more temporal data.

There are other approaches to biological data analysis based on ASP. For instance, in [12] the authors use ASP to compute an over-approximation of the set of Boolean networks which fit best with experimental data and provide the corresponding encodings. Of course, the approach can be used to explain the experimental data in a similar way as made by ILP systems, but using a purely logical approach, as we do in this paper.

3 The general approach

The general idea of our approach is simple and can be summarized as an attempt to infer a graph whose nodes are mutations (in genes) and whose arcs represent the causality relation between mutations, that is their most plausible (temporal) sequence. Data consists of variant allele frequencies and thresholds are used to accept/discard a temporal dependency.

The initial steps consist in a deterministic analysis followed by a blind search entirely left to the inferential engine. A pre-processing phase is necessary in order to correctly use data. A file MAF (Mutation Annotation Format) [13] is taken in input and the HUGO (HUman Genome Organization nomenclature) [11] gene symbols are extracted. In addition, experiment code, depth of readings—of the tumor sample in support of the variable allele—, and total depth of tumor sample readings, are uploaded. The ratio of the last two values is then calculated, thus obtaining variant allele frequencies for each sample gene. These numbers are then rounded to integers in order to make them more easily manageable by the ASP engine. The temporal information relative to the order of findings is extracted from the experiment identification, along with data that allow to classify cases—in the case of study, different organoid tissues. At this point the ASP system (grounder+solver) Clingo [9] is executed on the data to compute the predicates explained in the code (according to the stable model semantics). A post-processing of Clingo's output is also implemented in order to depict the computation results as a graph.

3.1 ASP code over simple data

In this section we will see our ASP code and how it works over some example instances. Once filtered, input data is stored by two predicates, defined exten-

sionally by facts. Each patient can be involved in more than one experiment; the ternary predicate `sample` relates the patient ID with its experiment IDs and their times (given the experiments ordering), while the ternary predicate `val` reports each instance of mutated gene in any experiment and its value:

```
sample(pat-A,exp-A1,1).  sample(pat-A,exp-A2,2).  sample(pat-A,exp-A3,3).
sample(pat-A,exp-A4,4).  sample(pat-B,exp-B1,1).  sample(pat-B,exp-B2,2).
sample(pat-B,exp-B3,3).  sample(pat-B,exp-B4,4).  sample(pat-C,exp-C1,1).
sample(pat-C,exp-C2,2).  sample(pat-C,exp-C3,3).  sample(pat-C,exp-C4,4).
sample(pat-C,exp-C5,5).
val(exp-A1,a,3027).  val(exp-A1,b,3027).  val(exp-A2,a,1245).
val(exp-A2,c,3245).  val(exp-A3,b,1245).  val(exp-A3,c,1234).
val(exp-A4,a,2555).  val(exp-A4,b,1324).  val(exp-A4,c,1254).
val(exp-A4,d,1092).  val(exp-B1,d,3027).  val(exp-B1,e,3027).
val(exp-B2,d,1245).  val(exp-B2,a,3245).  val(exp-B3,a,1245).
val(exp-B3,b,1234).  val(exp-B4,a,1334).  val(exp-B4,b,1334).
val(exp-C1,b,3027).  val(exp-C2,a,1245).  val(exp-C2,b,3245).
val(exp-C3,a,1415).  val(exp-C3,z,1415).  val(exp-C4,z,9145).
val(exp-C4,d,9145).  val(exp-C5,d,145).
```

In the first five rows we define the values of the samples of three patients `pat-A`, `pat-B` and `pat-C`, with the (temporal) order of samples. Subsequently, starting at row six, we indicate mutated genes values. For the sake of simplicity we named values `a`,`b`,`c` ....

A number of "domain" predicates are computed from the input data (needed to limit the grounding), as follows:

```
gene(G)    :- val(_,G,_).
patient(P) :- sample(P,_,_).
value(V)   :- val(_,_,V).
time(0..T) :- sample(_,_,T), not sample(_,_,T+1).
```

The predicate `gene(G)` forces value `G` to appear as second parameter of `val` and, therefore, will hold for each gene uploaded in our samples. Analogous considerations for `patient(P)` and `value(V)`. `time(T)`, instead, is instanced with a list of values that goes from 0 to a maximum temporal value contained in the predicates `sample(P,E,T)`.

The following predicates allow to retrieve information from the data. Here we report some examples. Being ASP capable to deal with the whole NP class, and even the Σ_2^P class if disjunctive heads are used, one can capture/infer a wide family of properties. We will be back on this issue in Sec. 6.

```
variationConsec(P,G,T2) :-
                sample(P,E1,T1),sample(P,E2,T2),T2=T1+1,
                val(E1,G,V1),val(E2,G,V2),variation(V1,V2).
variation(A,B) :-
                value(A),value(B),|A-B|>10.
precedes(P,G1,G2) :-
                firstVariation(P,G1,T1),firstVariation(P,G2,T2),T1<T2.
```

```
8  firstVariation(P,G,T) :-
9                  variationConsec(P,G,T), not varPrec(P,G,T).
10 varPrec(P,G,T) :-
11                 variationConsec(P,G,T1), variationConsec(P,G,T),T1<T.
12 precedeKTimes(G1,G2,K) :-
13                 gene(G1),gene(G2),K=#count{patient(P):precedes(P,G1,G2)}.
14 reachable(G1,G2) :-
15                 maybePrecede(G1,G2).
16 reachable(G1,G2) :-
17                 maybePrecede(G1,G3), reachable(G3,G2).
18 maybePrecede(G1,G2) :-
19                 precedeKTimes(G1,G2,K1),precedeKTimes(G2,G1,K2),K1>K2,
20                 not reachable(G2,G1).
21 #show maybePrecede/2.
```

`variationConsec(P,G,T2)` holds whenever for patient `P` there are two consecutive samples (the second one with time `T2`) with the same gene `G` with two values V_1 e V_2 that satisfy `variation`, namely their difference in absolute value is greater than 10. This value is the variation sensibility, in this simple example we choose a low parameter in order to take always the consecutive variation of the mutated gene. In our example we can infer that:

```
variationConsec(pat-A,a,2).  variationConsec(pat-A,c,3).
variationConsec(pat-A,b,4).  variationConsec(pat-A,c,4).
variationConsec(pat-B,d,2).  variationConsec(pat-B,a,3).
variationConsec(pat-B,a,4).  variationConsec(pat-B,b,4).
variationConsec(pat-C,b,2).  variationConsec(pat-C,a,3).
variationConsec(pat-C,z,4).  variationConsec(pat-C,d,5).
```

Before defining the main predicate of this example, namely, the predicate `precedes(P,G1,G2)`, let us see the rule `varPrec(P,G,T)` at row 10: it holds if a gene `G` of a certain patient `P` has some variations in the sample at time T and also in some previous samples—and therefore it is *not* the first variation detected. This predicate is used in the body of `firstVariation(P,G,T)`: the predicate inferring the minimum time `T` for which we have a variation of gene `G` in `P`.

At this point we can illustrate `precedes(P,G1,G2)` in row 6: it is true if, for a given patient, the first variation of `G1` happens before the first variation of `G2`. Notice that it is not necessary to force that the two genes are different as it is implicit in the fact that the time of their first variation is different. So from the previous example we can infer:

```
precedes(pat-A,a,c).    precedes(pat-A,a,b).    precedes(pat-A,c,b).
precedes(pat-B,d,a).    precedes(pat-B,d,b).    precedes(pat-B,a,b).
precedes(pat-C,b,a).    precedes(pat-C,b,z).    precedes(pat-C,a,z).
precedes(pat-C,b,d).    precedes(pat-C,a,d).    precedes(pat-C,z,d).
```

Predicate `precedeKTimes(G1,G2,K)` stores in `K` the number of patients for which the gene `G1` precedes the gene `G2`, while `maybePrecede(G1,G2)` is inferred

if G1 precedes G2 more frequently than G2 precedes G1. We report below the inferred instances of precedeKTimes in which the guard K is greater than zero (namely there is at least one case in which G1 precedes G2.

```
precedeKTimes(b,a,1).  precedeKTimes(d,a,1).  precedeKTimes(a,b,2).
precedeKTimes(c,b,1).  precedeKTimes(d,b,1).  precedeKTimes(a,c,1).
precedeKTimes(a,d,1).  precedeKTimes(b,d,1).  precedeKTimes(z,d,1).
precedeKTimes(a,z,1).  precedeKTimes(b,z,1).
```

For a and b, there are 2 patients that exhibit variation of a before b, while just one exhibits the opposite variation. Hence we deduce maybePrecede(a,b), while in case of a and d we don't infer any *most likely* order.

The final result in our example is:

```
maybePrecede(a,b).   maybePrecede(a,c).   maybePrecede(a,z).
maybePrecede(b,z).   maybePrecede(c,b).   maybePrecede(z,d).
```

In order to guarantee that the direct graph resulting from maybePrecede is acyclic, we check that the number of times the variation of the first gene precedes the second is strictly greater than the opposite order and we also impose the absence of the predicate reachable that states that already exists a path between two genes in the graph inferred.

The visualisation of the graph obtained is shown in Figure 2. The table containing the data (the variant allele frequency value has been omitted for simplicity) where the pairs highlighted are the variations revealed. In Figure 3 (right) it is shown the result automatically produced by Graphviz, after the Clingo output has been converted in the dot edges format Figure 3 (left).

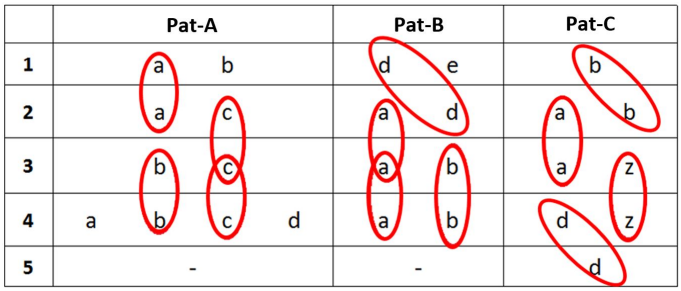

Fig. 2. Summarising table

4 Data sets

We tested our ASP program over the—real and publicly available—data set *Breast cancer patient xenografts* [6] available from http://www.cbioportal.org/.

```
digraph G {
            a -> b;
            a -> c;
            a -> z;
            b -> z;
            c -> b;
            z -> d;
}
```

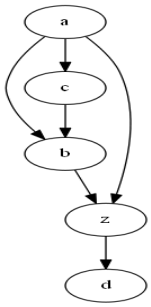

Fig. 3. Graph from `maybePrecede`, in dot edges format (left), and as visualized by Graphviz (right)

This particular study was chosen as it provides different temporal analysis of the same patient's cancer. The cBioPortal repository—as well as many other similar sites—is currently adding new case studies every year. In order to facilitate usage we, therefore, also implemented a (simple) graphical user-interface in Java providing functionalities to automate the data extraction, instances creation, as well as Clingo execution. Subsequently the ASP solver output is processed to display results by using the open source program GraphViz (Graph Visualization Software) [10].

The open source platform cBioPortal provides data from many cancer genomics data-sets (by now there are 150 of them) allowing their download, analysis, and visualisation. It was originally developed at Memorial Sloan Kettering Cancer Center (MSK) and now its software is available under an open source license via GitHub at https://github.com/cBioPortal. The portal is designed to bring cancer researchers near to the complexities of human genome data, allowing a rapid, intuitive, and precise visualization of expression profiles as well as clinical attributes from large-scale cancer genomics projects. A brief tutorial of the tools provided by cBioPortal is available on the web site; further documentation is available at https://cbioportal.readthedocs.io/en/latest/README.html.

4.1 Data download

From the website it is possible to query and/or download data from one or more studies satisfying given criteria (for instance, studies related to specific gene's values) or it is possible to download the entire archive. From the home page it is also possible to query databases specified within studies, select the genetic profile, the single patient or a group of them and then specify collections of genes of interest. Users can submit even more specific queries by using the Onco Query Language (OQL), for which a brief guide is available on the website. The same information can be obtained through web API and library in R and Matlab. However, the faster way to visualize all genes from a study, consists in directly downloading the whole study database from https://github.com/cBioPortal/datahub/tree/master/public,

containing all complete archives available from the portal. Once unzipped the archive, a series of file is at hand, among which *data_mutations_extended* written in MAF that will be our input.

Mutation Annotation Format (MAF) We briefly report here the MAF fields used by our program, redirecting the interested reader to the official page https://wiki.nci.nih.gov/display/TCGA/Mutation+Annotation+Format+(MAF)+Specification for a full-fledged description of the format.

Mutations are detected by alignment and comparison between DNA sequences relative to biopsies and the human genome reference (without mutations) available at NCBI (National Center for Biotechnology Information) [16]. A MAF file identifies mutations, reports type (SNPs—Single Nucleotide Polymorphisms, deletions, or insertions) and identifies if they are somatic (originated in the specific tissue—and therefore more interesting) or belonging to the germinal line. This is recorded with further annotations provided by the format.

The format of a MAF file is tab-delimited columns. The first row contains the version used, the second one the fields headers, the first 34 of which are mandatory.

Variant Allele Frequency (VAF) is given by reads *depth* on cancer's sample, referring to an altered allele `t_alt_count`, with respect to total reads `t_depht`. We will use precisely this ratio to describe mutation level, leaving the possibility of more elaborated/precise measurements, open in our code.

4.2 Breast cancer xenograft

Data from the above mentioned study *Breast cancer patient xenografts* [2, 8] contains, for each patient, an average of seven temporally-qualified samples. Not all provided data is from human biopsy as, thanks to xenotransplantation, it has been possible to observe and collect information from cancer cells grown in different tissues, called xenograft.

The study shows the clonal dynamics of initial engraftment and subsequent serial propagation of primary and metastatic human breast cancers in immunodeficient mice. One of the results showed by the study is that similar clonal expansion patterns can emerge in independent grafts of the same starting tumour population, indicating that genomic aberrations can be reproducible determinants of evolutionary trajectories. So we can see that measurement of genomically defined clonal population dynamics will be highly informative for functional studies using patient-derived breast cancer xenoengraftment.

The study, employing single-cell sequencing, was able to analyze and provide data on three different tissues: subcutaneous, subrenal capsule, and mammary fat pad. In [6] we can find pictures showing that different analyses were carried out, referred to the same patient. However, this was done over different xenograft tissues, that present mutations with independent developments. Therefore, for the same patient, different situations based on the genotype analysed on the xenograft, have been studied. This required an adaptation of our ASP code to discriminate across this particular data.

5 Mutations evolution analyzer

In order to get the complete archives stored in cBioPortal, automatically analyse data, and visualize the graph of variations found, we built a Java GUI called `Mutations Evolution Analyzer` (Figure 4) that contains a menu with four items: `Download`, `Create`, `Run`, and `Help`.

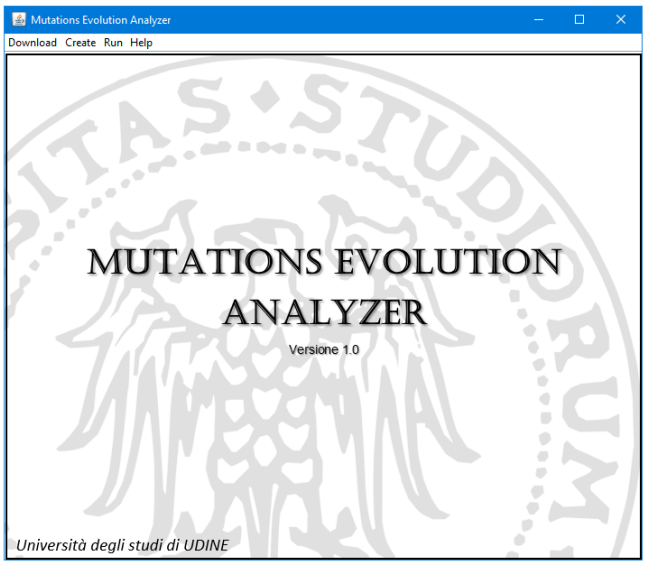

Fig. 4. Analyzer of Mutations' Evolution

The item `Download` sends a query to the web interface of the Cancer Genomic Data Server (CGDS) and, if available, returns an updated list of the studies, allowing the download over the site
https://github.com/cBioPortal/datahub/tree/master/public
of one or more of them in a user specified directory—see Figure 5.

Once the archive has been unzipped, we find *data_mutations_extended* file. By clicking on `Create` and then `Selected fields` the program allows to get, from a MAF file, the values of the selected fields (see Figure 6). To speed up selection, there is a checkbox called *Select standard field*, that automatically selects the fields used for our analysis: $Hugo\_Symbol$, $Tumor\_Sample\_Barcode$, t_alt_count, and t_depth. In particular, for *Breast cancer patient xenografts*, the identification field is obtained by the union of the patient ID, the type of the sample (tumor or xenograft), the position in temporal order of the sample, the genotype studied, and whether it is relative to the whole genome or if it is targeted. For instance, sample `SA429X1A-targeted` is relative to patient `SA429`, it has been obtained over a xenograft (`X` means xenograft), is the first sample of this type (`1`)—as samples have a sequential number that identifies their temporal order—, applies

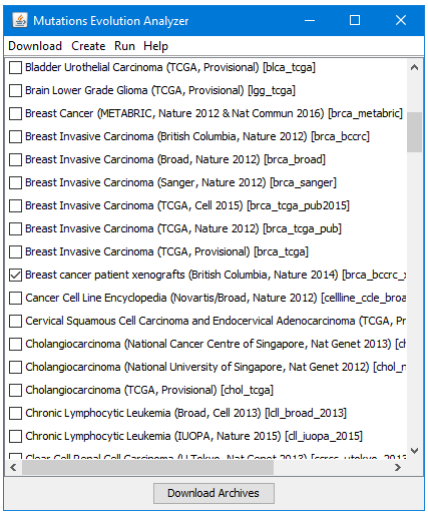

Fig. 5. Download of one ore more studies

to the genotype `A`[2], for which it has been analyzed a particular subsequence of the genome (`targeted`). In order to get this single information directly available (instead of searching them every time from the $Tumor\_Sample\_Barcode$) it is sufficient to select the checkbox named *Extract data from identifier* that adds those new fields to the selected ones, with headers named $patientID$, $Type$, $Time$, $Genotype$, and $Wide$.

Subsequently our program creates a new file containing the selected fields, as well as a new one called $ValGene$, obtained by normalizing the variant allele frequency—namely the ratio between t_alt_count e t_depth—, and then multiplies it by a reasonable value (we choose 10^7 in order to consider the first six digits after decimal point) rounding the result. We perform these additional operations to cast variant allele frequencies to integers, as it is easier to work with them in ASP with respect to the numbers in floating point.

Once the file containing the necessary fields has been obtained, by clicking on `Create -> ASP file`, a graphical interface asking the number of genes that we want analyze is shown. In Figure 7 the reader can see the default value. On click data are ordered, identifying the most frequent genes, taking the first k elements, with k as specified. An `Unknown` gene is a gene whose sequence is not known, so our program ignores it, as there is not necessarily a correlation between different unknown genes.

For the ASP syntax we use lowercase letters for all literal values, and replace characters "-" and "." by "_".

[2] For our usage it is sufficient to know that different letters correspond to different types. Refer to [6] for a full (and cleaner) explanation.

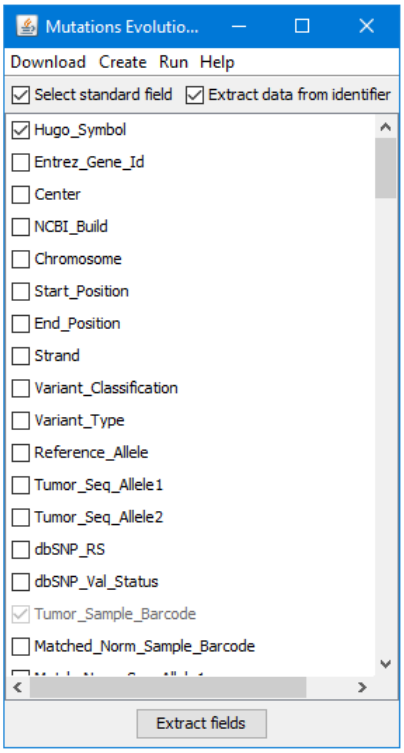

Fig. 6. Create → Selected fields

Then some Java code creates instances and adds them to the main ASP program under construction. Inessential modifications to adapt the analysis to the data under the study are performed at this stage in order, for example, to discriminate sample from same patient but with different genotypes: for each sample we add a fact for the type, wide, and genotype, in particular we consider the last one. Here there are the predicates deduced from sample `SA429X1A-targeted`:

```
sample(sa429,sa429x1a_targeted,1).
type(sa429x1a_targeted,x).
wide(sa429x1a_targeted,targeted).
genotype(sa429x1a_targeted,a).
```

As for the other rules we get:

```
variationConsec(P,GT,G,T2) :-
        sample(P,E1,T1),sample(P,E2,T2),T2=T1+1,
        val(E1,G,V1),val(E2,G,V2),variation(V1,V2),
        genotype(E1,GT),genotype(E2,GT).
precedes(P,GT,G1,G2) :-
```

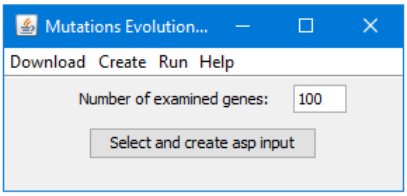

Fig. 7. Create → ASP file

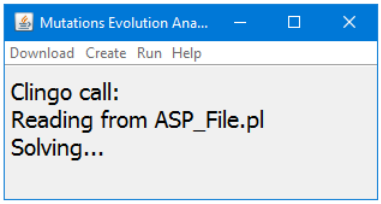

Fig. 8. Clingo execution

```
6       firstVariation(P,GT,G1,T1),firstVariation(P,GT,G2,T2),T1<T2.
7   firstVariation(P,GT,G,T) :-
8       variationConsec(P,GT,G,T), not varPrec(P,GT,G,T).
9   varPrec(P,GT,G,T) :-
10      variationConsec(P,GT,G,T1), variationConsec(P,GT,G,T),T1<T.
11  precedeKTimes(G1,G2,K) :-
12      gene(G1),gene(G2),
13      K=#count{patient(P),genotype(GT):precedes(P,GT,G1,G2)}.
```

The arity of those first four predicates is increased by one in order to discriminate different genotype variations. Moreover, we infer `variationConsec` if genotypes in the same sample turn out to be equal. Predicate `precedeKTimes` has the same arity but the function *count* works with different pairs of patient-genotype rather than just with patients.

Finally `Run -> Clingo and Graphviz` allows the user to select the ASP file created and solve it with Clingo (see Fig. 8). The result is then redirected to a temporary text file, from which the row containing inferred predicates `maybePrecede` is selected. Those are processed into edges of a direct graph in `dot` format, on which we run Graphviz to display the result—see Fig. 9. This picture shows the variations detected over 100 genes, with the predicate `variation` that identifies variations with a value above 1000.

Notice that the height of the graph is barely 1. This is because, despite there are on average seven samples for the fifteen patients, they are referred to different genotypes, so this increases the number of "sample to analyse" but at most (only) three temporal values are available for each of them, significantly reducing the number of relations found.

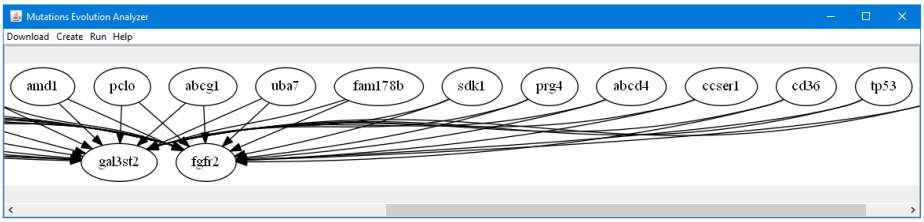

Fig. 9. Graph for 100 genes

Finally with the last item of the menu, `Help`, we can open links to the documentation of the resources used: the home page of cBioPortal, the official description of MAF, the web site of Potassco and the description of dot format.

6 Conclusions

In this paper we report on a new system capable of dealing with data retrieved from carcinome analysis. The system comes equipped with a front-end that pre-processes/filters standard-format data and transform them into a logic program, as well as with a post processor that allows to visualize the results in a graph. The reasoning core is based on ASP solving — the Clingo system is used. The predicates implemented for the data analysis reported in the paper are just simple examples and the expressivity of ASP can be exploited by more complex queries/analyses. In particular, one of our future work will be trying to reconstruct phylogenetics of cancer (using the ideas presented in [3]). For doing that we need access to much more data—and, in particular, of several stages per patient. We are looking for data sources.

A further direction of study consists in an integration with ILP, as a means to replace or collaborate with ASP, as far as the reasoning part is concerned.

Acknowledgments. We thank Alessandro Dal Palù, Andrea Formisano, and Enrico Pontelli for the several discussions on this material. Agostino Dovier and Alberto Policriti are partially supported by INdAM-GNCS 2015–2017 and by PRID ENCASE projects.

References

1. V. Bevilacqua, P. Chiarappa, G. Mastronardi, F. Menolascina, A. Paradiso, and S. Tommasi. Identification of tumor evolution patterns by means of inductive logic programming. *Genomics, Proteomics & Bioinformatics*, 6(2):91–97, 2008.
2. E. Cerami, J. Gao, U. Dogrusoz, B. E. Gross, S. O. Sumer, B. A. Aksoy, A. Jacobsen, C. J. Byrne, M. L. Heuer, E. Larsson, Y. Antipin, B. Reva, A. P. Goldberg, C. Sander, and N. Schultz. The cbio cancer genomics portal: An open platform for exploring multidimensional cancer genomics data. *Cancer Discovery*, 2:1–6, 2012.

3. A. Dal Palù, A. Dovier, A. Formisano, A. Policriti, and E. Pontelli. Logic programming applied to genome evolution in cancer. In C. Fiorentini and A. Momigliano, editors, *Proceedings of the 31st Italian Conference on Computational Logic, Milano, Italy, June 20-22, 2016.*, volume 1645 of *CEUR Workshop Proceedings*, pages 148–157. CEUR-WS.org, 2016.
4. A. Dal Palù, A. Dovier, A. Formisano, and E. Pontelli. Exploring life through logic programming: Answer set programming in bioinformatics. TR-CS-NMSU-2014-10-24, New Mexico State University, 2014. To appear as ACM Book Chapter.
5. L. De Raedt. *Logical and relational learning.* Cognitive Technologies. Springer, 2008.
6. P. Eirew, A. Steif, J. Khattra, G. Ha, D. Yap, H. Farahani, K. Gelmon, S. Chia, C. Mar, A. Wan, E. Laks, J. Biele, K. Shumansky, J. Rosner, A. McPherson, C. Nielsen, A. Roth, C. Lefebvre, A. Bashashati, C. de Souza, C. Siu, R. Aniba, J. Brimhall, A. Oloumi, T. Osako, A. Bruna, J. Sandoval, T. Algara, W. Greenwood, K. Leung, H. Cheng, H. Xue, Y. Wang, D. Lin, A. Mungall, R. Moore, Y. Zhao, J. Lorette, L. Nguyen, D. Huntsman, C. Eaves, C. Hansen, M. Marra, C. Caldas, S. Shah, and S. Aparicio. Dynamics of genomic clones in breast cancer patient xenografts at single-cell resolution. *Nature*, pages 422–426, 2014.
7. E. Erdem. Applications of answer set programming in phylogenetic systematics. In M. Balduccini and T. C. Son, editors, *Logic Programming, Knowledge Representation, and Nonmonotonic Reasoning*, volume 6565 of *Lecture Notes in Computer Science*, pages 415–431. Springer, 2011.
8. J. Gao, B. A. Aksoy, U. Dogrusoz, G. Dresdner, B. Gross, S. O. Sumer, Y. Sun, A. Jacobsen, R. Sinha, E. Larsson, E. Cerami, C. Sander, and N. Schultz. Integrative analysis of complex cancer genomics and clinical profiles using the cbioportal. *Science Signaling*, 2(6):269, 2013.
9. M. Gebser, R. Kaminski, B. Kaufmann, and T. Schaub. Clingo = ASP + control: Preliminary report. *CoRR*, abs/1405.3694, 2014.
10. GraphViz. http://www.graphviz.org/.
11. HUGO. http://www.genenames.org/.
12. T. Le, H. Nguyen, E. Pontelli, and T. C. Son. ASP at work: An ASP implementation of phylows. In A. Dovier and V. S. Costa, editors, *Technical Communications of the 28th International Conference on Logic Programming, ICLP 2012, September 4-8, 2012, Budapest, Hungary*, volume 17 of *LIPIcs*, pages 359–369. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2012.
13. MAF. https://wiki.nci.nih.gov/display/tcga/mutation+annotation+format.
14. S. Muggleton. Inductive logic programming. *New Generation Comput.*, 8(4):295–318, 1991.
15. S. Muggleton. Inverse entailment and progol. *New Generation Comput.*, 13(3&4):245–286, 1995.
16. NCBI. https://www.ncbi.nlm.nih.gov/.
17. Y. Qiu, K. Shimada, N. Hiraoka, K. Maeshiro, W.-K. Ching, K. F. Aoki-Kinoshita, and K. Furuta. Knowledge discovery for pancreatic cancer using inductive logic programming. *IET Systems Biology*, 8:162–168, 2014.
18. F. Riguzzi, E. Bellodi, and R. Zese. A history of probabilistic inductive logic programming. *Front. Robotics and AI*, 2014, 2014.
19. A. Srinivasan, R. D. King, S. Muggleton, and M. J. E. Sternberg. Carcinogenesis predictions using ILP. In N. Lavrac and S. Dzeroski, editors, *Inductive Logic Programming, 7th International Workshop, ILP-97, Prague, Czech Republic, September 17-20, 1997, Proceedings*, volume 1297 of *Lecture Notes in Computer Science*, pages 273–287. Springer, 1997.