Towards a Logic Programming tool for cancer data analysis

Alice Tarzariol Eugenia Zanazzo Agostino Dovier Alberto Policriti

Abstract

The main goal of this work is to propose a tool-chain capable of analyzing a data collection of temporally qualified (genetic) mutation profiles, i.e., a collection of DNA-sequences that present variations with respect to their "healthy" versions. We implemented a system consisting of a front-end, a reasoning core, and a post-processor: the first transforms the input data retrieved from medical databases into a set of logical facts, while the last displays the computation results as graphs. Concerning the reasoning core, we employed the Answer Set Programming paradigm, which is capable of deducing complex information from data. However, since the system is modular, this component can be replaced by any logic programming tool for different kinds of data analysis. Indeed, we tested the use of a probabilistic inductive logic programming core.

1 Introduction

A very simplified view of cancer is that of a game of *mutations accumulation* in the genome—both genes and regulatory regions—of an organism. Although biological data is produced by today's technology at an unprecedented rate, it is also extremely noisy, a fact which represents one of the most delicate aspects of this game. More precisely, even though *any* collection of mutations—i.e. variations in the DNA-sequence, with respect to what is considered the "standard" sequence—might be triggering significant biological processes involved in cancer progression, only some of the mutations are classified as "drivers" (of the progression) 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 more basic biological mechanisms already independently at work.

Tackling a classification problem for the collection of detected mutations in a given tissue, requires a multi-faceted approach. In order to work towards building reliable filters suitable for defining the *blueprint* of the disease under study, different kinds of knowledge must be integrated with brute force analysis. This paper aims at proposing the architecture of a pipeline capable of analyzing a data collection of *temporally qualified* mutation profiles and synthesizing a dependency graph (see Figure 1). At a high level, the general idea of our approach can be summarized as an attempt to infer a graph whose nodes and edges represent mutations in genes and their causality relation, respectively, describing their most plausible temporal sequence.

When dealing with biological data, many auxiliary problems—mostly related to I/O—must be faced; concerning this aspect, the data processing pipeline described here is interfaced with specialized languages and file formats for the field. Our data consist of variant allele frequencies in genes, whose outstanding values describe their mutations. Thresholds have been introduced to accept or discard a temporal dependency among mutated genes. The combination of information inferred by cross-comparison performed "locally" on input data, and knowledge coming from external sources, allows for deducing the output graph that should represent a set of putative dependencies.

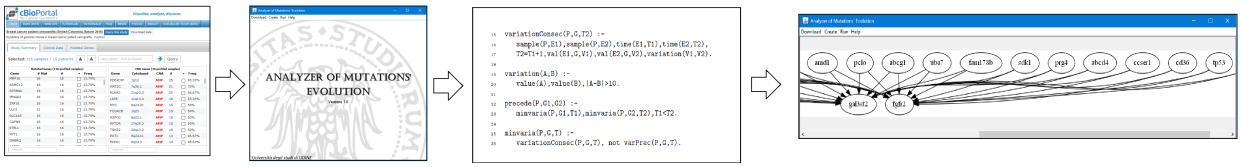

Figure 1: The proposed pipeline: data is retrieved from the web database and converted into ASP input facts by a Java interface. The ASP engine elaborates the data, whose output is successively converted into graph format.

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

Answer Set Programming (ASP) [14] allows for encoding compactly, elegantly, and with great flexibility, problems that belong to NP or even to Σ_2^P. In our application, we implement parametrically designed strategies that use thresholds to fix conditions on the addition and removal of edges in the inferred graph, represented by logical predicates. However, the ASP program reported should be seen only as a possible example, since the focus of this paper relies on the general method and its flexibility. The reasoning core can be replaced by any other one, possibly implemented using further logic programming dialects.

To prove the generality of the approach, we have experimented with the use of probabilistic inductive logic programming (PILP). Since PILP performs both structure and parameters learning, the input collection must contain a considerable number of patients sampled to obtain results with reasonable confidence. Concerning the type of analysis conducted with our ASP engine, large-sized data collections are not available yet; thus, we decided to test PILP to conduct a different analysis on a bigger data set, comparing its preliminary results with those produced by other existing tools rather than with our ASP tool.

The paper is organized as follows: some preliminaries and related works are discussed in Section 2, while Section 3 presents the overall ideas of our approach. The ASP implementation, some examples of programs, and a brief analysis of the results found are presented in Section 4. In Section 5, we present a PILP approach, while related results and comparison are reported in Section 6. Lastly, Section 7 contains some conclusions drawn from this work. The tool we have developed is available for free download and use from `http://clp.dimi.uniud.it/sw/`.

2 Related work

Cancer data processing is, obviously, a very active and lively field in many scientific environments. This section reports on some works using techniques stemming from the logic programming community. We assume the reader already possess basic knowledge of logic and logic programming.

2.1 Inductive Logic Programming

Inductive Logic Programming (ILP) allows for implementing 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., [23] or [9] for a more general approach to logical and relational learning).

Let us assume that O^+ and O^- are a set of positive and negative observations, respectively. Furthermore, consider them as two disjoint sets of ground atoms. Suppose also that a (possibly empty) background theory P is given, defined by a set of clauses. The general inductive problem consists in 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 in ILP systems to identify the sets H consisting of clauses as *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 than 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, unless 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 possibly affected by errors (very common in medical data), the working set H might be neither *complete*, namely H "explains" only a (proper) subset G (good) of O^+, nor *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 minimizing the size of W.

An extension of ILP is PILP (probabilistic inductive logic programming) [31] where facts (observable) and rules (background theory) can be annotated by a probability estimate. In this case, a semantics based on probabilistic inference rules is adopted (distribution semantics); the inherent forms of uncertainty are therefore handled in a natural way. There are several dialects of probabilistic logic programming in the literature (see, e.g., [30]). We use probabilistic rules of the form $A : \pi :\!- Body$, meaning that if the $Body$ is true, then the atom A is true with a probability π.

[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."

Inductive Logic Programming has been successfully applied to analyze big data and, in particular, cancer databases: for instance, in [34] the authors participated in a world-wide carcinogenicity prediction competition (organized by the National Toxicology Program in the USA) using the ILP system Progol [24]. In the first round of the competition Progol produced the highest predictive accuracy of any automatic system participating in the test.

Furthermore, Progol has been also applied in [27] to infer general properties on pancreatic cancer as well as to allow its early diagnosis. For instance, one of the inductively computed predicates (quantitatively) confirms that measurements of *CEA* and *Elastase I* are very useful to detect pancreatic cancer. In this study, data was collected from 438 cases and the classification was based on the observation of the lymph node metastatic status and tumor differentiation status. Available data was provided by lab test (e.g., CEA, CA19-9, Glucose, Elastase I, Serum Amylase, ...). After having identified the most promising features using standard feature selection criteria, data was divided into three groups (low, normal, and high), proceeding similarly to domain experts. Each patient record in the database represented an abnormality of a patient and the records were split into positive and negative. Data was translated into a set of facts, and Progol executed to rank rules by their capability of explaining the database.

In [2] the authors proposed an ILP approach to the problem of modeling evolution patterns for breast cancer. Data was obtained from a cohort of 124 patients at different progression stages. Data *and* background knowledge were expressed in logic programming. Then, a set of hypotheses built on this knowledge was computed using refinement, least general generalization, inverse resolution, and most specific corrections.

2.2 Answer Set Programming

Answer Set Programming (ASP) has been exploited for myriads Bioinformatics applications (for an overview, see [6, 7]). Among them, we would like to point out the approaches for the phylogenetic reconstruction (see, e.g., [11]). Indeed, the origin of this work comes from the desiderata, presented in the 2016 edition of the conference CILC [5], of extending the ASP tools for phylogenetic reconstruction to build a phylogenetic reconstruction of cancer progression. If a specific form of cancer is related to a mutation of a gene presents both in a human and in another species (e.g. rat), this may justify a deeper study of the development of cancer, carried on using the individuals from the other species as models. It can also explain why a specific drug working for the other species does not work in a human. However, in general, in order to perform such type of analyses large amounts of temporal data (among other things) is needed.

There are further approaches to biological data analysis based on ASP. For instance, in [19] the authors use ASP to compute an over-approximation of the set of Boolean networks which best fit with experimental data and provide the corresponding encodings. That approach can be used to explain the experimental data in a similar way as made by ILP systems, but using a purely logical approach (this is also what we do in the first part of this paper).

2.3 Cancer data analysis

Cancer research in recent years has witnessed the proliferation of *cancer progression models*. A comprehensive survey on the methods and tools developed for phylogenetic studies of tumour evolution can be found in [33]. It classifies the tumour phylogeny methods in three groups, according to the design of the study: cross-sectional, regional bulk, and single-cell. The first method aims to build trees describing the common progression pathways across a population, using sample tissues belonging to a number of different patients. Instead of using data from multiple patients, both regional bulk and single-cell methods use input derives from a single individual to build a model. The former uses subregions of a tumour or distinct tumour sites (i.e., bulk genomic samples) while the latter uses single cells in one or more tumour sites.

For the purpose of our study, we focused on two cross-sectional methods that are close to the current state of the art in the field of tumour phylogenetic: TRONCO and BML. Other tools are surveyed in [33], for example, the cross-sectional model TO-DAG [20] or the regional bulk model BitPhylogeny [36]. Concerning single-cell tools, some examples are OncoNEM [32] and SCITE [18] (for which analysis and comparison can be found in [8]), or more recently SiFit [37].

TRONCO

TRONCO (TRanslational ONCOlogy Package) [3] is an R package that, given a dataset of tumour samples, infers either a DAG (Directed Acyclic Graph) or a tree depending on the inference algorithm used. We briefly illustrate the inference algorithm CAPRI [28] that produces as output a DAG. CAPRI associates a node to each gene while the edges represent the causal and temporal relations occurring between two genes. After the input preprocessing phase, CAPRI selects the candidate edges to be inserted in the model based on Suppes Theory of *selective influence* [35]. The basic aspect of a selective influence theory states that given two events i and j, the event i has selective influence on j if the two following properties hold:

$$\begin{array}{ll} \textit{Temporal Priority} & p(i) > p(j) \\ \textit{Probability Rising} & p(j|i) > p(j|\neg i) \end{array}$$

CAPRI performs a non-parametric bootstrap with rejection resampling to obtain the estimates $\hat{p}(i), \hat{p}(j), \hat{p}(j|i), \hat{p}(j|\neg i)$. At the end of the testing phase, two p-values are obtained, if their value is below a chosen threshold (usually 0.05 or 0.01) an edge is entered in the model, otherwise, it is discarded. The presence of any loop is avoided by removing an edge with the *highest* p-value within the loop. Finally, in the last phase, CAPRI finds the optimal solution given by a subset of the edges.

BML

Bayesian Mutation Landscape [22] infers a tree that represents the set of possible evolutionary paths of the tumour. In order to do so, BML calculates the *evolutionary probability* $P(g)$, which is the probability of observing the genotype g starting from the non-mutated (normal) genotype g_0. In the first phase, BML builds a Bayesian network consisting of a DAG G and a set of parameters Θ that represent the conditional probabilities $P(C = c \,|\, \Pi_C = \pi)$. These values indicate the probability that a variable (a gene) would find itself in a state c, conditioned to the fact that its parents find themselves in state π. The value $n_{c,\pi}$, which is the number of samples in which $C = c$ and $\Pi_C = \pi$, is computed for every pair (C, Π_C) using input data. The score BIC (*Bayesian Information Criterion*) is used for selecting the optimal DAG.

To obtain the evolutionary probabilities P, a dataset containing both tumor genotypes and unobserved ancestral genotypes is necessary; from these sets, BML builds binary trees that have the genotype g_0 as root, the tumour genotypes as leaves, and ancestral genotypes as internal nodes. In this phase, BML finds the tree T_* and the Bayesian network B_* that maximize the BIC score. This maximization is performed by an ordering-based search. In this search, after initializing an ordering of the variables, every variable is constrained to choose its parents among preceding variables in the ordering only. The algorithm performs its search in the space of all possible variable ordering. Once a tree is found, the solution gets perturbed using *Nearest Neighbour Interchange* and the BIC score is re-computed. This loop—perturbations plus re-computation—continues until a local maximum is found. The last phase consists in the reconstruction of the most likely evolutionary paths. Starting from all genotypes with k mutations (default k is 3), the algorithm deduces recursively the most likely ancestral states (i.e., the ones with the largest evolutionary probability P), terminating when the root is reached.

3 Data-sets and general approach

In this section, we describe the format of the analyzed data, contained in the online repository cBioPortal. Given the peculiarity of the input used with ASP, we will describe the study analyzed explaining how we interpreted its data. Lastly, we outline the structure of our system, excluding the reasoning engine that is examined in more details in the following sections.

3.1 Cancer data repository

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 [4, 12] (see Panel 1). 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 conceived to bring cancer researchers near to the complexities of human genome data, allowing a quick, intuitive, and precise visualization of expression profiles as well as clinical attributes from large-scale cancer genomics projects. A brief tutorial

Panel 1 *Data download from cBioPortal.*

From the cBioPortal website it is possible to download the entire archive, or to query and/or download data from one or more studies satisfying given criteria (for instance, studies related to specific values of gene). 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 downloading directly 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.

Panel 2 *The MAF fields used by our tool.*

In this panel, we briefly report the Mutation Annotation Format (MAF) fields used in our program, redirecting the interested reader to the official page for a full-fledged description of the format: `https://wiki.nci.nih.gov/display/TCGA/Mutation+Annotation+Format+(MAF)+Specification`.
Mutations are detected by alignment and comparison between DNA sequences relative to biopsies and a human genome reference (without mutations) available at NCBI (National Center for Biotechnology Information) [25]. A MAF file identifies mutations, reports their type (SNPs—Single Nucleotide Polymorphisms, deletions, or insertions) and if they are somatic (originated in the specific tissue—and therefore more interesting) or belong to the germinal line. These information are 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, while the second reports the fields headers, the first 34 of which are mandatory.
Concerning the Variant Allele Frequency, it is given by the reads *depth* on cancer's sample, referring to an altered allele `t_alt_count`, with respect to total reads `t_depht`.

of the tools provided by cBioPortal is available on the web site; further documentation is available at `https://www.cbioportal.org/tutorials`.

3.2 Raw input processing

A pre-processing phase is required to use data correctly. A Mutation Annotation Format (MAF) file [21] (see Panel 2) is taken in input and the HUGO (HUman Genome Organization nomenclature) [17] gene symbols are extracted. In addition, the experiment code (identifier), sequencing depth—of the tumour sample in support of the variable allele—and total sequencing depth of tumour sample, are uploaded. The ratio of the last two values is then computed, obtaining the Variant Allele Frequencies (VAF) for each sample gene. Concretely, we will use this ratio in ASP to describe the mutation level, leaving the possibility of more elaborated/precise measurements open in our code. These numbers are rounded to integers in order to make them more easily manageable by the ASP engine.

3.2.1 Breast cancer xenograft

We tested our ASP program over the—real and publicly available—data set *Breast cancer patient xenografts* [10] downloadable from `http://www.cbioportal.org/`. This particular study was chosen as it provides different temporal analysis of the same patient's cancer. The temporal information relative to the order of findings is extracted from the experiment identification, along with data that allow for classifying cases—in our main

sample case, different organoid[2] tissues.

For each patient, there is an average of seven temporally-qualified samples. In addition, data provided from human biopsy xenotransplantation allows for observing and collecting further 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. Among its results, it appeared 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 the article, we can find pictures showing that, concerning the same patient, several analyses were carried out over different xenograft tissues, that present mutations with independent developments. Therefore, since different situations have been studied for the same patient, we adapt our ASP code in order to discriminate across the genotype analyzed on the xenograft.

3.2.2 Breast Invasive Carcinoma

We tested our PILP implementation, TRONCO, and BML on the dataset relative to the study Breast Invasive Carcinoma [26]. For clarity, we reduced the size of the original dataset (that tracks the mutations of 321 genes in 507 patients) to the ten most frequently mutated genes, inferring relations occurring among them only. Although the input for our PILP implementation, too, originates from a MAF file, we used the an already pre-processed plain text file. Said file is available, alongside other 4 sample files, in the `data` folder of BML.

The file consists in a matrix M whose rows and columns contain the patients and the observed genes, respectively; the element $M[a, b]$ will be equal to 1 if patient a presents a mutation in gene b, and 0 otherwise.

3.3 Interface

The cBioPortal repository—as well as many other similar sites—is currently adding new case studies every year. Therefore, in order to facilitate their usage, we also implemented a (simple) graphical user-interface in Java providing functionalities to automate data extraction, instances creation, as well as Clingo ASP execution [13]. Subsequently, the ASP solver output is processed to display results by using the open source program GraphViz (Graph Visualization Software) [16].

Figure 2 shows the first screenshot of the GUI that we named `Mutations Evolution Analyzer`. It contains a menu with four items: `Download`, `Create`, `Run`, and `Help`.

The `Download` command sends a query to the web interface of the Cancer Genomic Data Server (CGDS) and, if available, it 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 3.

Once the archive has been unzipped, we find a *data_mutations_extended* file. By clicking on `Create` and then on `Selected fields` the program allows for extracting the values of the selected fields belonging to a MAF file (see Figure 4). To speed up selection, there is a checkbox called *Select standard field*, that automatically selects the fields used for our analysis: $Hugo\_Symbol$, $tumour\_Sample\_Barcode$, t_alt_count, and t_depth. In particular, for *Breast cancer patient xenografts*, the identification field called $tumour\_Sample\_Barcode$ is obtained by the union of the patient ID, the type of the sample (tumour 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`), is the first sample of this type (`1`)—as samples have a sequential number that identifies their temporal order—, applies to the genotype `A`[3], for which it has been analyzed a particular subsequence of the genome (`targeted`). Selecting the checkbox named *Extract data from identifier*, each information contained in $tumour\_Sample\_Barcode$ becomes a new field, together with the selected ones; their headers are respectively: $patientID$, $Type$, $Time$, $Genotype$, and $Wide$.

[2] In-vitro produced, simplified version of an organ.

[3] For our usage it is sufficient to know that different letters correspond to different types. Refer to [10] for a full (and cleaner) explanation.

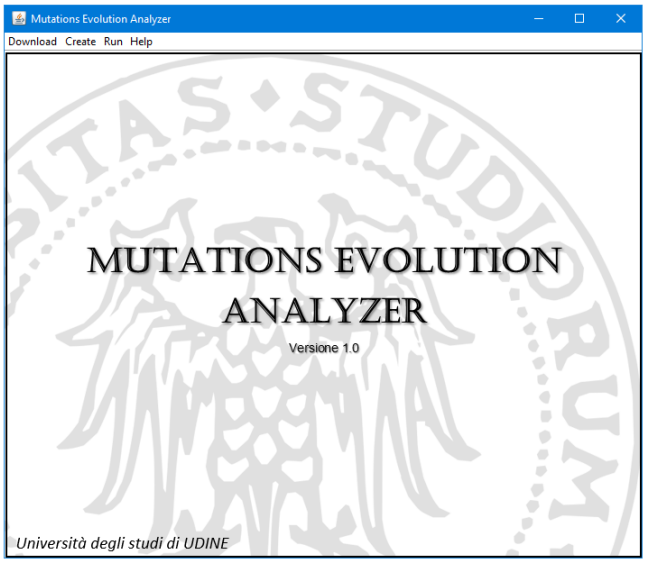

Figure 2: Analyzer of Mutations' Evolution.

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 is created, by clicking on `Create -> ASP file`, a graphical interface will appear asking the number k of genes that we want analyze. In Figure 5, the reader can see the default value for k. On click, data are sorted according to the most frequent genes and then the first k elements are selected. Among the genes, the term `Unknown` is discarded by our program, since it refers to a set of genes whose sequence is not known. Therefore, there not exist necessarily a correlation between different unknown genes.

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

Then our system creates the instances and adds them to the main ASP program under construction. Next, we perform further modifications to adapt the analysis to the specific considered data. For instance, we discriminate sample refereed to the 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. As example, we report below the predicates deduced from sample `SA429X1A-targeted`:

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

Once the ASP program is created, the command `Run -> Clingo and GraphViz` allows the user for selecting and solving it with Clingo (see Section 4 for the ASP implementation, while Section 5 contains the the alternative PILP engine). The result is then redirected to a temporary text file, from which the inferred ASP predicates that represent the edges are selected and converted in the dot format. Then, a post-processing of the output based on GraphViz [16] is implemented in order to depict the computation results as a direct graph.

Finally with the last item of the menu, `Help`, we can open links to the documentation of the resources used.

4 ASP engine

In this section, we describe the ASP engine in some detail and we show how it works over some example instances.

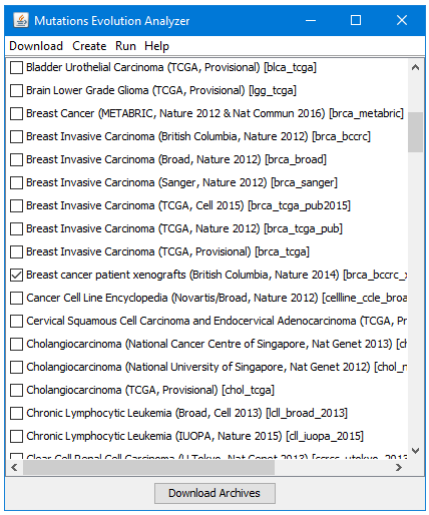

Figure 3: Download of one ore more studies.

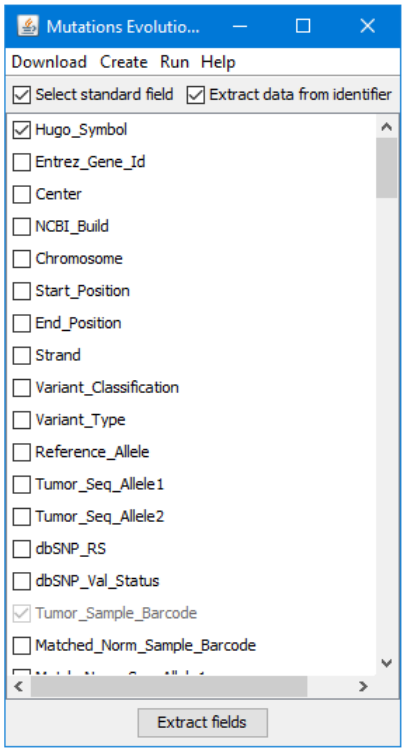

Figure 4: Create → Selected fields.

The predicate representing the direct edges in the deduced graph will be `maybePrecede`; in order to understand how it is computed, we start illustrating the auxiliary predicates.

Once filtered, input data is stored by two predicates defined extensionally by facts. Each patient can be involved in more than one experiment; the ternary predicate `sample` relates the patient to its 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:

```
1 sample(pat-A,exp-A1,1). sample(pat-A,exp-A2,2). sample(pat-A,exp-A3,3).
2 sample(pat-A,exp-A4,4). sample(pat-B,exp-B1,1). sample(pat-B,exp-B2,2).
3 sample(pat-B,exp-B3,3). sample(pat-B,exp-B4,4). sample(pat-C,exp-C1,1).
```

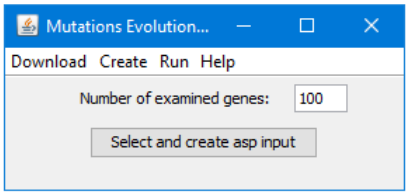

Figure 5: Create → ASP file.

```
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` ....

In order to limit grounding, the "domain" predicates are computed from the input data 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 hold for `patient(P)` and `value(V)`. Moreover, `time(T)` is instanciated with a list that goes from a maximum temporal value contained in the predicates `sample(P,E,T)` to 0.

The following predicates allow retrieving information from the data.

```
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.
firstVariation(P,G,T) :-
                variationConsec(P,G,T), not varPrec(P,G,T).
varPrec(P,G,T) :-
                variationConsec(P,G,T1), variationConsec(P,G,T),T1<T.
precedeKTimes(G1,G2,K) :-
                gene(G1),gene(G2),K=#count{patient(P):precedes(P,G1,G2)}.
reachable(G1,G2) :-
                maybePrecede(G1,G2).
reachable(G1,G2) :-
                maybePrecede(G1,G3), reachable(G3,G2).
maybePrecede(G1,G2) :-
                precedeKTimes(G1,G2,K1),precedeKTimes(G2,G1,K2),K1>K2,
                not reachable(G2,G1).
#show maybePrecede/2.
```

The predicate `variationConsec(P,G,T2)` holds whenever for patient `P` there exist 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, such that their difference in absolute value is greater than 10. This threshold is the variation sensibility and is related to the normalization of the variant allele frequency. For this simple example, we choose a low fixed value in order to take always the consecutive variation of the mutated gene. However, the system can be easily extended to consider it as a parameter, e.g., testing different values according to the considered form of cancer. 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 predicate `precedes(P,G1,G2)`, let us examine the rule `varPrec(P,G,T)` at row 10: it holds if a gene `G` of a certain patient `P` exhibits variations in the sample at time T and also in the 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 two patients that exhibit variation of the former before the latter, while just one exhibits the opposite variation. Therefore, we deduce `maybePrecede(a,b)`; on the other hand, we do not infer any *most likely* order with genes `a` and `d`.

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. Moreover, we impose the absence of the predicate `reachable`, which states that there exists already a path between the two genes in the inferred graph.

The visualisation of the identified edges is shown in Figure 6, where the pairs highlighted in the table represent the variations revealed. For simplicity, the variant allele frequency value has been omitted. The result automatically produced by GraphViz is shown in Figure 7 (right), which is obtained after the conversion of the Clingo output in the dot edges format (see Figure 7 (left)).

4.1 ASP Results

Since the data-set *Breast cancer patient xenografts* (described in Section 3) contains data derived from both humans and xenograph on different genotypes, the ASP program requires an adaptation. Running the Mutation

	Pat-A	Pat-B	Pat-C
1	a b	d e	b
2	a c	a d	a b
3	b c	a b	a z
4	a b c d	a b	d z
5	-	-	d

Figure 6: Summarising table.

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

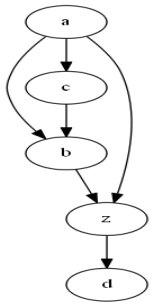

Figure 7: Graph from `maybePrecede`, in dot edges format (left), and as visualized by GraphViz (right).

Evolution Analyzer generates the previous predicates, with the exception of a subset that is substituted with the following elements:

```
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) :-
                firstVariation(P,GT,G1,T1),firstVariation(P,GT,G2,T2),T1<T2.
firstVariation(P,GT,G,T) :-
                variationConsec(P,GT,G,T), not varPrec(P,GT,G,T).
varPrec(P,GT,G,T) :-
                variationConsec(P,GT,G,T1), variationConsec(P,GT,G,T),T1<T.
precedeKTimes(G1,G2,K) :-
                gene(G1),gene(G2),
                K=#count{patient(P),genotype(GT):precedes(P,GT,G1,G2)}.
```

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

Once the `maybePrecede` predicates are deduced and subsequently processed into edges, GraphViz displays the result—see Figure 8. This picture shows the variations detected over 100 genes, with the predicate `variation` that identifies variations with a value above 1000.

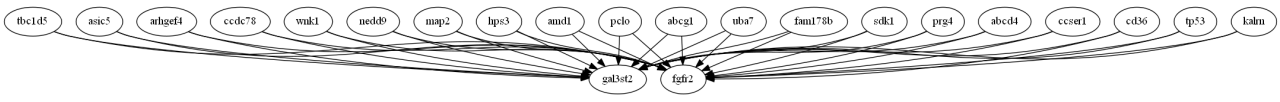

Figure 8: Graph for 100 genes.

Despite the presence of an average of seven samples for each of the fifteen individuals, in practice, at most (only) three temporal values are available for each of them: since data refer to different genotypes, they represent parallel evolution paths for the same patient. Therefore, the number of "samples to analyse" increases, resulting in a significant reduction in the quantity of relations found. For this reason, the height of the graph is just one.

The situation described above is typical in this field nowadays: for a certain patient, very few and short sequences of *temporal snapshots* are publicly available. E.g., after cancer diagnoses, if possible, a surgery follows immediately in order to remove the tumor. This phenomenon is going to change dramatically in the near future, due to the drop in costs and technological simplifications available for data production in xenograph analysis.

5 PILP engine

In this section, we will use the tools/libraries available in PILP in order to learn causal relations between gene mutations. To do so, we performed structure learning through the algorithm SLIPCOVER [1] that given a language bias and a set of positive and negative examples returns a probabilistic logic program (hereinafter referred to as *theory*) such that the probability of positive examples is maximized and the probability of negative examples is minimized [29]. The algorithm first performs a beam search in the space of clauses to find the set of candidate (promising) clauses, then, guided by the log likelihood, performs a greedy search in the space of theories.

The input file given to the algorithm is a Prolog file divided into three parts: a preamble, some language bias information and a set of example interpretations. The preamble includes the definition of the parameters given to the algorithm. In this section we set the value of the seed that will be used by the algorithm in its greedy search phase with the fact `set_sc(c_seed,1518110625)`.

In the language bias section we associate a predicate of arity 0 to each gene observed and we put some loose constraints on the form of the output theory. In practice this section is a sequence of facts:

```
output(<predicate>/<arity>).
modeh(<recall>/<predicate>).
modeb(<recall>/<predicate>).
determination(<predicate>/<arity>,<predicate>/<arity>).
```

The definition `output` indicates the predicates for which the score of the output theory is optimized. In our case we considered every predicate as an output predicate. The definition `modeh` specifies the atoms that can appear in the head of clauses while `modeb` specifies the predicates that can appear in the body of clauses. In our model every predicate can appear either in the head or the body of clauses. Finally, the definition `determination` states that the second predicate can appear in the body of clauses for the first predicate.

In the interpretation section each sample in our database is converted into a model as follows, if a gene is mutated we have a positive example otherwise we have a negative example. A model example is given below:

```
begin(model(2)).
neg('APC').
'TP53'.
neg('KRAS').
end(model(2)).
```

Finally it is necessary to indicate how the models are divided into folds with facts

```
fold(<fold\_name>,<list of model identifiers>).
```

We decided to divide the models equally into five sets and to use the first four of them as our *training set*, while the last one is used as a *test set*. The output produced after the execution of SLIPCOVER is the list of clauses that make up the theory:[4]

```
[  (APC:0.9998437800170417;:0.00015621998295833883:-TP53,KRAS),
   (APC:0.4892755480369245;:0.5107244519630755:-TP53),
   (TP53:0.763403270990247;:0.23659672900975304:-APC).]
```

[4]The output of SLIPCOVER has a syntax slightly different from the one used throughout the paper. The clause $A : \pi :\!-\ B$ is written as $A : \pi; : 1 - \pi :\!-\ B$ (actually this is an instance of a more complex form of clause with disjunctive head).

In addition to the theory SLIPCOVER might return At the end of the learning phase we refine the theory to produce a weighted graph according to the following criteria. Firstly we do not take into consideration the clauses with a probability lower than 0.5, such threshold might seem high but given that the theory is the result of structure learning we decided to keep in knowledge base only the clauses learned with a sufficiently high degree of certainty. This decision was taken after the initial testing phase showed that the graphs produced using either no threshold at all or a lower threshold (0.1,0.3) were almost complete graphs and thus not sufficiently informative. Secondly, as opposed to the standard deductive interpretation, we give an abductive interpretation to clauses in the theory. In such an interpretation, for each clause we insert in the graph as many edges as there are predicates in the body of the clause, such edges will go from the gene in the head of the clause to the genes in the body of the clause with weight equal to the (positive) probability of the clause—by convention if the (positive) probability is greater than 0.995 the weight of the edge to be inserted is rounded up to 1. Finally if in the output theory both clauses $A : \pi_1 :- B$ and $B : \pi_2 :- A$ are present, we insert in the graph only the edge (or edges) belonging to the clause with the highest (positive) probability. If, however, both clauses have probability 1, then both edges are inserted in the graph. The heuristics just introduced, unlike the other methods presented above, do not guarantee the absence of loops.

It should be noted that our current interpretation does not give an exact representation of the information learned. Let us consider, as an example, the clause $A : \pi_i :- B, C$. In our interpretation two edges are added to the output graph, one from A$\rightarrow$ B and the other one from A $\rightarrow$ C ,both with weight π_i. Our assumption is that if there is a relation between B and C then another clause—with one of the genes in its head and the other in its body—must be present in the output theory. However, in order to give a proper inductive interpretation, it would be more accurate to insert a single edge from A$\rightarrow$ B $\wedge$ C introducing the AND port to our representation. Moreover, it is possible to have two (or more) clauses with the same atom in their head in the output theory:

`A:`π_1`:- B,C`

`A:`π_2`:- C,D`

In this case, in order to make the two clauses completely independent, in addition to AND ports we would need to introduce and use a XOR port: a single edge A $\rightarrow$((B$\wedge$C) $\oplus$ (C$\wedge$D)) would then be inserted in the revised model.

5.1 Input/output specification

The matrix described in section 3.2 is the input of our tool. This type of representation of the dataset can be used as input for a number of other methods develop to study tumour evolution, including TRONCO and BML. The input matrix was initially reduced to take into consideration only the ten most frequently mutated genes removing the samples that did not present any of the selected mutations accordingly. This reduced matrix was then used to generate the interpretation section of the Prolog file to be given to SLIPCOVER. Additional code was used to suitably fill the preamble and language bias sections of the Prolog file. The complete Prolog file was then given as input to SLIPCOVER. The resulting theory produced at the end of SLIPCOVER's computation was refined accordingly with the criteria previously underlined in this section. At the end of this process the graph associated to the refined theory was represented in form of a .dot file. Finally the .dot file was given as input to GraphViz in order to produce as output a graphical representation of the refined theory.

6 Experimental Results and Comparison

We first run some initial tests on the web site server of CPLINT (`http://ml.unife.it/cplint/`), then we downloaded the tool, interfaced with the "7.7.9 development" version of SWI Prolog (`http://www.swi-prolog.org/`) and run on a Linux desktop machine.

Given the partially greedy nature of the learning algorithm we decided to run our method 50 times on the same dataset to assess stability of results. In order to do so, before every test the value of the seed given to the algorithm was changed. As mentioned before, the first four sets of folds were used to train the model while the fifth set was used to test the success of the model on previously unseen data. The results obtained were generally not stable and the results of the algorithm never yielded the same output graph(/model) twice. The main issues with our model were overfitting of data—a common issue in learning problems—and a serious computation time problem, with computation/running times of over an hour on our dataset. As a remark, a comparison with [15] would be unfair since this proposal requires trees while TRONCO and our tool generates DAGs that are not necessarily rooted trees.

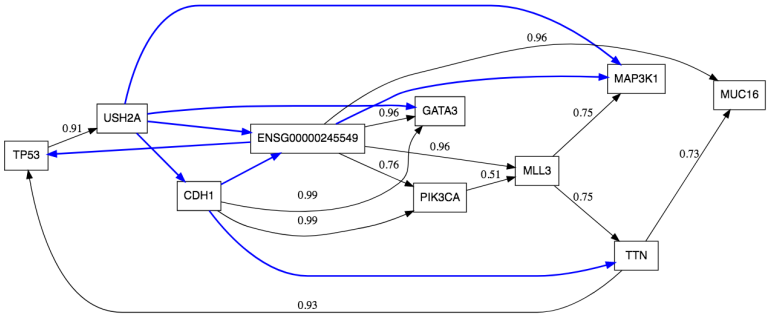

Figure 9: The result of the PILP engine analysis.

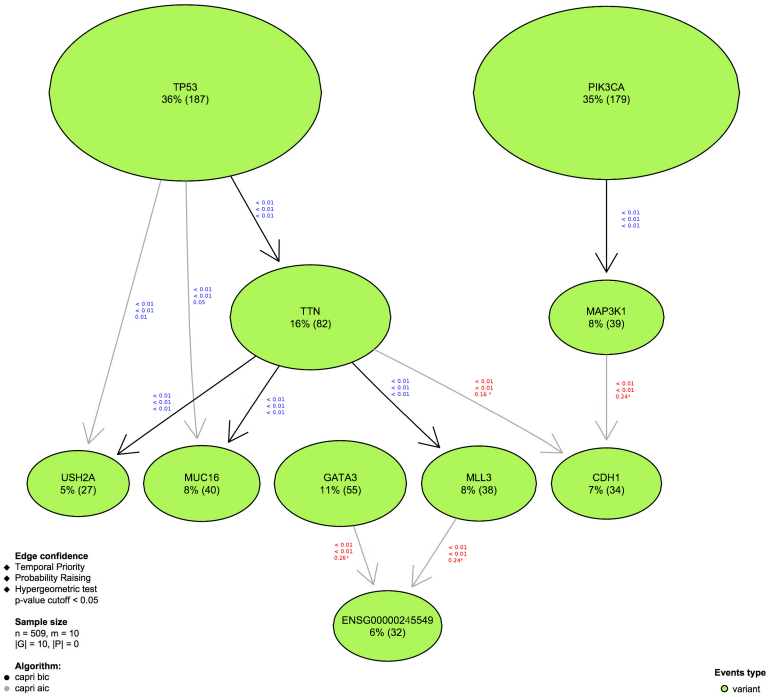

Figure 10: The result of TRONCO analysis.

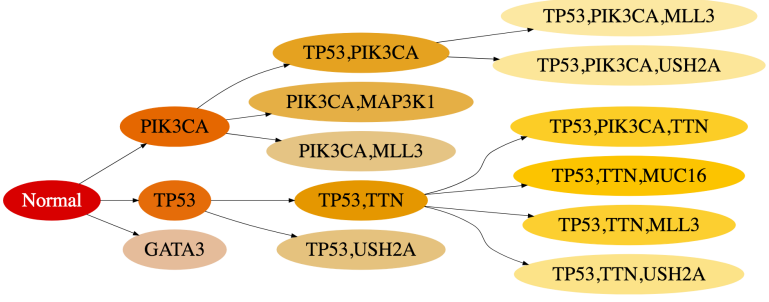

Figure 11: The result of BML analysis.

The graph presented in Figure 9 is the output of the theory that yielded the best testing results (i.e., the one that returned the highest score on the test set). Overfitting is clearly evident and results in a model that is overly complex to deduce any clear information from it. We notice that all genes observed have a node associated and two loops are present. Moreover, the graph does not have a root, while *MAP3K1*, *GATA3*, *MUC16* are its sinks. It should be noted that the subgraph constituted by the edges with weight one only (blue in the picture) is a DAG. The output returned after the execution of TRONCO is a DAG with three roots (Figure 10): *TP53*, *PIK3CA*, *GATA3*. While the mutations of genes/proteins *TP53* and *PIK3CA* have influence on a number of other mutations, the mutation of gene/protein *GATA3* has a much lower frequency and has influence exclusively on a single other mutation. As in the case of our model all the genes observed in the dataset are represented in TRONCO's output graph. Lastly (Figure 11) BML returns a tree that is somewhat similar to the model returned by TRONCO, with genes *TP53*,*PIK3CA* central to most of the possible mutation paths while the gene *GATA3* is much less relevant to the progression of the tumour. In this case two of the genes observed (*CDH1* and *ENSG00000245549*) are not present in any of the paths found.

We quantified the degree of similarity between the different graphs using two measures reflecting two different notions of similarity. Given two graphs, the first measure computes the percentage of edges that the graphs share. Computing this measure we see that BML and TRONCO have in common 0.571 percent of edges, while our model shares 0.200 percent of edges with BML and 0.129 percent of edges with TRONCO. The second measure, on the other hand, represents the percentage of pairs of genes *(x,y)* connected by a path in both graphs. As before the comparison between BML and TRONCO returns the highest score (0.47), while our model shows a slightly higher degree of similarity with TRONCO (0.354) compared to BML (0.32).

The computation of the two measures confirms that the results obtained with our method are significantly different from the ones obtained with TRONCO and BML. We can identify two main reasons to explain such differences. The first is that, contrary to TRONCO and BML where the model selection criteria is performed using the BIC score, SLIPCOVER uses the log-likelihood as a selection criteria thus explaining the complexity of the graph obtained. Secondly, one of the main assumption that both TRONCO and BML make is that the graph obtained cannot contain a loop. Such assumption however represents somewhat a reduction of the underlying cancer process, so while the clarity and readability of the graphs is increased, it is questionable how much they reflect the reality of the process observed. In contrast our tool allows the presence of loops and, while in principle the model is more realistic, it's more difficult to identify precisely the evolutionary paths.

Moreover our tool makes, by design, as few assumptions as possible on the process observed and does not rely on any field-specific knowledge thus testing the limits of the learning process when the background knowledge available is very limited.

7 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 for visualising 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 [5]). For doing that we need access to much more data—and, in particular, of several stages per patient. Data sources of this type are not yet easily available. We also repeated the analysis with a tool based on PILP, showing the modularity of the approach. The paper shows the potentiality of Logic Programming for clinical data analysis.

Acknowledgments. We thank Alessandro Dal Palù, Andrea Formisano, and Enrico Pontelli for the several discussions on this material. A. Dovier and A. Policriti are members of INdAM-GNCS and partially supported by INdAM-GNCS 2015–2019 projects and by PRID ENCASE Uniud project.

References

[1] Elena Bellodi and Fabrizio Riguzzi. Structure learning of probabilistic logic programs by searching the clause space. *Theory and Practice of Logic Programming*, 15(4):213,229, 2013.

[2] Vitoantonio Bevilacqua, Patrizia Chiarappa, Giuseppe Mastronardi, Filippo Menolascina, Angelo Paradiso, and Stefania Tommasi. Identification of tumor evolution patterns by means of inductive logic programming. *Genomics, Proteomics & Bioinformatics*, 6(2):91–97, 2008.

[3] Giulio Caravagna, Alex Graudenzi, Daniele Ramazzotti, Rebeca Sanz-Pamplona, Luca De Sano, Giancarlo Mauri, Victor Moreno, Marco Antoniotti, and Bud Mishra. Algorithmic methods to infer the evolutionary trajectories in cancer progression. *Proceedings of the National Academy of Sciences*, 113(28):E4025–E4034, 2016.

[4] Ethan Cerami, Jianjiong Gao, Ugur Dogrusoz, Benjamin E. Gross, Selcuk Onur Sumer, Bülent Arman Aksoy, Anders Jacobsen, Caitlin J. Byrne, Michael L. Heuer, Erik Larsson, Yevgeniy Antipin, Boris Reva, Arthur P. Goldberg, Chris Sander, and Nikolaus Schultz. The cBio Cancer Genomics Portal: An Open Platform for Exploring Multidimensional Cancer Genomics Data. *Cancer Discovery*, 2:1–6, 2012.

[5] Alessandro Dal Palù, Agostino Dovier, Andrea Formisano, Alberto Policriti, and Enrico Pontelli. Logic programming applied to genome evolution in cancer. In Camillo Fiorentini and Alberto 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.

[6] Alessandro Dal Palù, Agostino Dovier, Andrea Formisano, and Enrico Pontelli. ASP applications in bio-informatics: A short tour. *KI*, 32(2-3):157–164, 2018.

[7] Alessandro Dal Palù, Agostino Dovier, Andrea Formisano, and Enrico Pontelli. Exploring life: Answer set programming in bioinformatics. In M. Kifer and Y. A. Liu, editors, *Declarative Logic Programming, Theory, Systems, and Applications*, pages 359–426. ACM Press, 2018. Preliminary version available as TR-CS-NMSU-2014-10-24, NMSU.

[8] Alexander Davis and Nicholas E Navin. Computing tumor trees from single cells. *Genome biology*, 17(1):113, 2016.

[9] Luc De Raedt. *Logical and relational learning*. Cognitive Technologies. Springer, 2008.

[10] Peter Eirew and et al. Dynamics of genomic clones in breast cancer patient xenografts at single-cell resolution. *Nature*, 518(7539):422–426, 2014.

[11] Esra Erdem. Applications of answer set programming in phylogenetic systematics. In Marcello Balduccini and Tran Cao Son, editors, *Logic Programming, Knowledge Representation, and Nonmonotonic Reasoning*, volume 6565 of *Lecture Notes in Computer Science*, pages 415–431. Springer, 2011.

[12] Jianjiong Gao and et al. Integrative analysis of complex cancer genomics and clinical profiles using the cbioportal. *Science Signaling*, 2(6):269, 2013.

[13] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, and Torsten Schaub. *Answer Set Solving in Practice*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2012.

[14] Michael Gelfond and Yulia Kahl. *Knowledge representation, reasoning, and the design of intelligent agents: The answer-set programming approach*. Cambridge University Press, 2013.

[15] Kiya Govek, Camden Sikes, and Layla Oesper. A consensus approach to infer tumor evolutionary histories. In *Proceedings of the 2018 ACM International Conference on Bioinformatics, Computational Biology, and Health Informatics*, pages 63–72. ACM, 2018.

[16] GraphViz. http://www.graphviz.org/.

[17] HUGO. http://www.genenames.org/.

[18] Katharina Jahn, Jack Kuipers, and Niko Beerenwinkel. Tree inference for single-cell data. *Genome biology*, 17(1):86, 2016.

[19] Tiep Le, Hieu Nguyen, Enrico Pontelli, and Tran Cao Son. ASP at work: An ASP implementation of phylows. In Agostino Dovier and Vítor Santos 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.

[20] Paola Lecca, Nicola Casiraghi, and Francesca Demichelis. Defining order and timing of mutations during cancer progression: the to-dag probabilistic graphical model. *Frontiers in genetics*, 6:309, 2015.

[21] MAF. https://wiki.nci.nih.gov/display/tcga/mutation+annotation+format.

[22] Navodit Misra, Ewa Szczurek, and Martin Vingron. Inferring the paths of somatic evolution in cancer. *Bioinformatics*, 30(17):2456–2463, 2014.

[23] Stephen Muggleton. Inductive logic programming. *New Generation Comput.*, 8(4):295–318, 1991.

[24] Stephen Muggleton. Inverse entailment and progol. *New Generation Comput.*, 13(3&4):245–286, 1995.

[25] NCBI. https://www.ncbi.nlm.nih.gov/.

[26] The Cancer Genome Atlas Network. Comprehensive molecular portraits of human breast tumours. *Nature*, 2012.

[27] Yushan Qiu, Kazuaki Shimada, Nobuyoshi Hiraoka, Kensei Maeshiro, Wai-Ki Ching, Kiyoko F. Aoki-Kinoshita, and Koh Furuta. Knowledge discovery for pancreatic cancer using inductive logic programming. *IET Systems Biology*, 8:162–168, 2014.

[28] Daniele Ramazzotti, Giulio Caravagna, Loes Olde Loohuis, Alex Graudenzi, Ilya Korsunsky, Giancarlo Mauri, Marco Antoniotti, and Bud Mishra. Capri: efficient inference of cancer progression models from cross-sectional data. *Bioinformatics*, 31(18):3016–3026, 2015.

[29] Fabrizio Riguzzi. Probabilistic inductive logic programming. *ALP Newsletter*, (1), March 2014.

[30] Fabrizio Riguzzi. *Foundations of Probabilistic Logic Programming*. River Publishers, 2018.

[31] Fabrizio Riguzzi, Elena Bellodi, and Riccardo Zese. A history of probabilistic inductive logic programming. *Front. Robotics and AI*, 2014, 2014.

[32] Edith M Ross and Florian Markowetz. Onconem: inferring tumor evolution from single-cell sequencing data. *Genome biology*, 17(1):69, 2016.

[33] Russel Schwartz and Alejandro Schäffer. The evolution of tumour phylogenetics: principles and practice. *Nature Review Genetics*, 18(4):213–229, 2017.

[34] Ashwin Srinivasan, Ross D. King, Stephen Muggleton, and Michael J. E. Sternberg. Carcinogenesis predictions using ILP. In Nada Lavrac and Saso 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.

[35] Patrick Suppes. *A probabilistic theory of causality*. North-Holland Publishing Company Amsterdam, 1970.

[36] Ke Yuan, Thomas Sakoparnig, Florian Markowetz, and Niko Beerenwinkel. Bitphylogeny: a probabilistic framework for reconstructing intra-tumor phylogenies. *Genome biology*, 16(1):36, 2015.

[37] Hamim Zafar, Anthony Tzen, Nicholas Navin, Ken Chen, and Luay Nakhleh. Sifit: inferring tumor trees from single-cell sequencing data under finite-sites models. *Genome biology*, 18(1):178, 2017.