 ISBN: 978-989-95618-4-7

Data acquisition and modeling for learning and reasoning in probabilistic logic environment

Dimitar Sht. Shterionov, Gerda Janssens

Dep. of Computer Science, Katholieke Universiteit Leuven, Belgiu
Dimitar.Shterionov@cs.kuleuven.be
Gerda.Janssens@cs.kuleuven.be

Abstract. Deriving knowledge from real-world systems is a complex task, targeted by many scientific fields. Such systems can be viewed as collections of highly detailed data elements and interactions between them. The more details the data include the more accurate the system representation is but the higher the computational requirements become. Using abstractions to summarize details is a well-known technique. Abstraction often leads to an accurate model of a system but, in other cases, introduces inaccuracies that we want to quantify. In this paper we propose an approach based on different levels of abstraction to define different representational models for a system. We use probabilities to quantify the inaccuracies that are introduced during the abstraction process. Such models then are used for reasoning and learning in a probabilistic environment. We use three example datasets (a small and simple social network, a probabilistic dictionary of approximately 300 words and a real biological neural network) to support our abstraction approach in probabilistic context. The environment we use is ProbLog - a small but powerful probabilistic extension of Prolog.

Keywords: probabilistic logic programming, datasets, probabilistic graphs, detail abstraction, applications, social network, probabilistic dictionary, c. elegans neural network.

1 Introduction

Modeling systems is a very complex task. In this paper we use a probabilistic extension of logic programming, ProbLog, to represent and reason about (uncertain) models of real-world systems. Logic programs use predicates to express relations between objects and logical inference is used to derive implicit information from explicit data. Representing all aspects in full detail is often impossible – either data are not available or they are too complex to comprehend and to model. Although a more detailed model does allow to derive more precise knowledge, the computational cost might become too high. To balance computational complexity and representational completeness, we need to come up with a right abstraction level [1]. Using abstraction in modeling concrete information often introduces inaccuracies. In this paper we show how probabilities can be used to

quantify these inaccuracies. The results obtained by reasoning with this probabilistic information, can be interpreted as abstractions of some concrete aspects. Thus, probabilistic inference can be used to reason at a higher level of abstraction. Also, the learning methods available in the ProbLog system benefit from a higher level of abstraction.

In this paper we propose to consider explicitly different levels of abstraction when modeling a system. We also show how probabilities can be used to compensate for a loss of information. In particular, the probabilities can be computed in a bottom-up fashion (i.e., from low levels of abstraction to high levels of abstraction) such that they only rely on information from the previous level of abstraction (containing more detailed information). We illustrate this approach by means of three examples.

Our paper is organized as follows: In Section 2 we introduce modeling in the context of probabilistic logic programming. Section 3, then, introduces our approach for data abstraction. Data acquisition and experiments are described in Section 4, where for every examined dataset all levels of abstraction are thoroughly explained. We conclude in Section 5 where we state related work and our future tasks.

2 Context and Motivation

2.1 Modeling and Logic Programming

Logic programming provides an easy, natural way to represent knowledge by the use of predicates which express relations between objects. In a logic program, a predicate is defined by its set of clauses. Facts (clauses with an empty body) are used to represent explicit data. Rules define when a relation (the head of the clause) holds by expressing conditions in the body of the clause. For example, a particular graph can be defined by an `edge/2` predicate, which models the connection between nodes of the graph. A `path/2` predicate which propagates the edge relation between distinct nodes can be defined by 2 rules. Logical inference derives implicit data, e.g. the pairs of nodes for which the path relation holds.

In general, facts can model n-ary relations. But for binary relations, such as `edge/2` or `parent/2`, the data represented by the facts can be thought of as a graph, with as nodes the arguments of the fact and edges expressing that the relation holds for the nodes connected by the edge. For nodes that are not connected by any edge, we know that the relation does not hold.

When we model a concrete problem we aim to represent it as complete as possible, in order to derive precise results. But a highly detailed model can impose a very high computational cost. That is why we have to decide about the level of abstraction, while keeping in mind the trade-off between level of details and computational cost. Abstracting away some details can give rise to relations that are no longer definitely true or false. For example, a fact can relate two authors for each paper they have written together. We can unify all these single relations in only one, which simply defines that two authors are co-authors. By

this abstraction, information for a specific paper is lost. Also, it does not specify the number of papers the two authors have in common. With the addition of probabilities, though, such information can be implied. This can be expressed by a probabilistic logic program which involves probabilistic relations.

2.2 ProbLog

In this paper we use ProbLog as our probabilistic framework. ProbLog [2] is a simple but powerful probabilistic logic language.

As in Prolog, a ProbLog program consists of predicate facts and rules. Facts, though, are annotated with a probability p_i and have the form $p_i :: f_i$. For facts known to be true, we can omit the probability annotation and write them as Prolog facts. A probability annotation represents the validity of a fact, or the degree of certainty in its truth value. ProbLog facts are interpreted as mutually independent random variables. The set of ProbLog facts can be viewed as a graph but, as its edges are no longer deterministic, they are annotated with a probability p_i. Such graphs are call **probabilistic graphs**.

We denote a ProbLog program as T[1], the set of its facts without their probabilistic labels as L_T, and the rules (or background knowledge) as BK. Then T defines a probability distribution over all subprograms L of L_T as follows:

$$P(L|T) = \prod_{f_i \in L} p_i \prod_{f_i \in L_T \setminus L} (1 - p_i). \quad (1)$$

where p_i is the probabilistic annotation of the i^{th} fact. Consider the following set of probabilistic facts:

$0.7 :: from\_to(s, a).$	$0.4 :: from\_to(a, b).$	$0.5 :: from\_to(b, c).$
$0.8 :: from\_to(b, g).$	$0.6 :: from\_to(a, c).$	$0.9 :: from\_to(c, g).$
$0.7 :: from\_to(b, g).$		

and the background knowledge:

$$path(X, Y) :- from\_to(X, Y).$$
$$path(X, Y) :- from\_to(X, X1), path(X1, Y).$$

The set of facts includes the nodes $\{s, a, b, c, g\}$, which could be interpreted as some states, objects, or situations. The predicate `from_to/2` defines a unidirectional relation between some of the nodes. Thus, our program represents a **probabilistic graph** of five nodes and seven edges with probabilistic annotations. The predicate `path/2` computes the transitive closure of the `from_to/2` relation.

ProbLog supports several forms of probabilistic inference, such as computing the success probability of a query (the overall probability of a query being true) and finding the most likely explanation of a query (the proof of the query with the highest probability).

[1] Denotations are similar to ones used in [3].

Computing the success probability can be computationally expensive, however, even for small problems. ProbLog relies on the inclusion-exclusion principle and uses BDDs (Reduced Ordered Binary Decision Diagrams) [4] to deal with overlaps between different explanations (or proofs), which is a $\#P - complete$ problem, referred to in the literature as the "disjoint-sum problem" [5]. The implementation of the ProbLog system includes exact and approximate algorithms for computing the success probability.

2.3 Implications for the data model

We express our knowledge in terms of objects and relations. When the data are characterized by probabilities, we can model them as a ProbLog program where facts express explicit knowledge. By inference on this program we can derive implicit information.

Facts that relate data objects can have arbitrary arity. As we use the probabilistic graph as our data model we focus on binary relations (directly mapped into probabilistic graphs). Our research, though, is valid for n-ary relations since such can always be transformed into sets of binary relations without any loss of generality.

3 Data Abstraction and Probabilities

We often need a highly complex data representation to model a system in a precise way. The more details are taken into consideration, the harder it becomes to process the data. Using abstraction, we can lower the level of details, gaining computational power but also loosing precision. Quantifying these imprecisions (inaccuracies) with probabilities can provide knowledge about the abstracted details. The results obtained by probabilistic reasoning can be interpreted as the abstraction of implicit information. Hence, probabilities enable reasoning at a higher level of abstraction about information details from a lower level of abstraction.

Detail abstraction (or detail restriction) is used in fields like computer graphics and object oriented programming, relying on diverse techniques, e.g., classification and regression. Here, we present a general-to-specific model of data abstraction which aims to distinguish levels of details and to quantify the loss of precision due to the transition between levels by a probability measurement. Figure 1 depicts the five levels of our model, enumerated from the top level $L1$ (most general representation) to the bottom level $L5$ (most specific, most detailed representation).

The model aims to introduce a general classification framework for data abstraction. This way, transforming deterministic data into probabilistic and encoding them as a probabilistic logic program can be reduced to a systematical process.

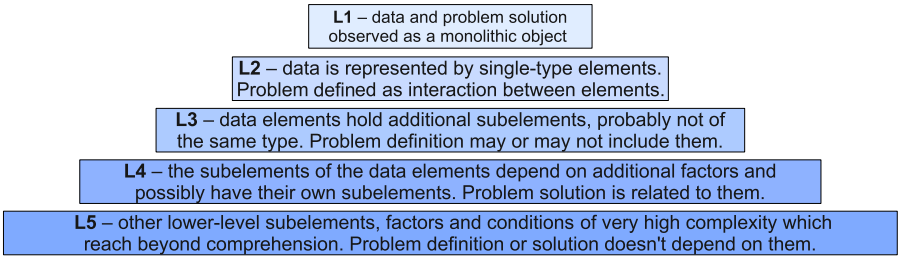

Fig. 1. A five-level pyramid, representing the levels abstraction.

Next, we define each level and the related processes for data modeling. We use a dataset representing the biological neural network of a worm as an example. A neural network is a grid of biological cells, called neurons. Each neuron receives signals from and sends signals to other neurons. The way signals are conducted relies on neurotransmitters and neuroreceptors. Neurotransmitters emit chemicals towards the receiving neuronal cell. Neuroreceptors catch specific chemicals and carry them towards the body of the neuron. The combination of different signals in the body result in diverse neuronal activity.

We start from $L5$ and show how abstraction is involved in data modeling.

1. **Level L5** corresponds to no abstraction - the data are left in full complexity. Elements of different types interact in complicated ways under dynamic conditions. It is impossible to construct a model including all these details. Because the data completely represents the system under investigation, there is no uncertainty [2]. Consider a dataset representing a living organism, e.g. a worm. It includes information about cells, genes, chemodynamics, molecular structures, behavior, environment, etc. This dataset is too complex, hence it is impossible to reason about.
2. **Level L4** is the first abstraction level. Focusing on solving a specific problem, $L4$ abstraction describes data cleared from incomprehensible or unrelated details. An $L4$ representation includes elements of different importance, conditions that influence the activity of the elements and factors that may influence either the whole system or a part of it. Because $L4$ abstracts away incomprehensible and unrelated details, they cannot be quantified by means of probabilities. Thus, $L4$ data are a set of complex and deterministic elements. For the worm, we want to model its neural network. A corresponding dataset includes neurons, connections, electrical and chemical characteristics (neuroreceptors, neurotransmitters) for each single neuron, temporal and spacial conditions, etc.
3. **Level L3** abstraction produces datasets of main and secondary elements. All the main elements have the same type. A main element is related to 0

[2] The "no uncertainty" statement is valid under the assumption that there do not exist any other data or data details that can be added to create more complete representation of the system under investigation.

or more secondary ones, referred to as sub-elements. Relations that characterize the data, appear between a main element and its sub elements, but also between two (or more) elements of the same hierarchy level. Abstracting from $L4$ to $L3$ excludes plenty of details introducing inaccuracies that can be quantified with probabilities. A probabilistic $L3$ model usually covers the realistic, although simplified dimensions of the system under investigation. We can reason on the $L3$ model by querying its main or secondary elements. Also, we use the $L3$ model while making further abstractions for level $L2$. From an $L4$ representation of the neural network, we can summarize a set of neuroreceptors and neurotransmitter [3] which characterizes a neuron. Also $L4$ details describe the type of signal corresponding to a tuple neurotransmitter-neuroreceptor. Therefore, we can construct an $L3$ model that relates neurons, neurotransmitters and neuroreceptors, and also contains a binary probabilistic relation expressing connectivity between neurons. In order to construct an $L3$ representation, it is only necessary to use the data from $L4$.

4. **Level L2** datasets contain only elements of one type involved in a relationship. In a graphical model these entities represent nodes and their relations - the edges of the graph. Edges can be weighed (labeled) to express some specifics of the relationship. Interaction between elements can be either deterministic or probabilistic, the latter leading to a probabilistic graph. On data of this level of abstraction we can post queries, define problems and experiments, which mainly investigate the relational dependencies between entities of the data. Data obtained by abstracting from $L3$, can be annotated with probabilities derived from the interaction of the main and the sub elements. The neural network of the worm becomes a graph of neurons, where edges are connections between two neurons. Since an $L3$ model of the neural network provides us with the knowledge about neurotransmitters and neuroreceptors we can evaluate the probability of positive or negative signaling between two neurons.
5. **Level L1** is the highest data abstraction level. An $L1$ representation of a system includes one single element. This element is static, not related to other objects neither influenced by external factors. For example, consider the dataset $D = \{worm\}$, representing the monolithic notion of the biological organism.

It is obvious that L2, L3 and L4 are the abstraction levels of interest. In the modeling process you can typically go from one level to another. Whether you are adding or removing details, depends on the problem we aim to solve.

[3] Neurotransmitters are chemicals that are emitted by the signaling nervous cell. Neuroreceptors are parts of the neuron (at the receiving site) which accept specific neurotransmitters and transform the chemical signal into electrical.

4 Modeling Examples

This section presents three datasets from different domains - linguistics, social networking and biology. Their investigation aims to illustrate how the abstraction model can be used. An overview is shown in Table 1 and Table 2.

Table 1. An overview of examined datasets in their initial form.

	"Les Miserables"	Probabilistic dictionary	Neural Network
Main element:	• character	• word	• neuron
Relation specifics:	• co-appearance in chapters	• synonymy	• connectivity
Relation from to:	• character-character	• word-word	• neuron-neuron
Relation type:	• non-directional	• non-directional	• unidirectional
Abstraction level:	$L2$	$L3$	$L2$

Table 2. An overview of examined datasets per abstraction level after processing.

Levels		"Les Miserables"	Probabilistic dictionary	Neural Network
$L2$	Main element:	• character	• word	• neuron
	Relation specifics:	• co-appearance in chapters	• synonymy/similarity	• signal transmission
	Relation from to:	• character-character	• word-word	• neuron-neuron
	Relation type:	• non-directional	• bidirectional	• unidirectional
$L3$	Subelements:	• number of chapters for single character • number of chapters of all novel	• meanings/definitions • usage frequencies • number of synonyms per single word	• neurotransmitters (chemicals) • neuroreceptors (genes)
$L4$	Conditions:	• story	• context	• previous states of neuronal membrane
	Factors:			• additional chemicals

4.1 Les Miserables

First, we investigate a dataset consisting of the characters from Victor Hugo's novel "Les Miserables" [6] and their co-appearance in chapters. The data, as derived from the source[4] is a deterministic weighed graph which we want to enhance with probabilities for more thorough analysis. The simplicity of this dataset allows us to outline some specifics in computing probabilities without complexity being an obstacle.

The starting point is a deterministic dataset (we denote it as LM) which represents a graph of characters (identified by an ID or a Name) as nodes and

[4] "Les Miserables" dataset source: `http://www-personal.umich.edu/~mejn/netdata/lesmis.zip`

co-appearances as edges. Weights on the edges are already computed and give the number of chapters in which two characters appear together. Because the dataset contains one type of elements and one type of links between them, it is a model of $L2$ abstraction level.

Transforming the original dataset into a probabilistic model can be realized in several ways. On the one hand, we can rely only on the available data and stick to the current level of abstraction. Consider two connected characters Ch_i and Ch_j with $N_{i,j}$ denoting the weight of their connection. Also, the number of all chapters in which Ch_i appears, denoted with N_i, is the sum of the weights of all outgoing connections from Ch_i. Thus, we can compute a probability that characterizes the connection between Ch_i and Ch_j $P'_{i,j} = \frac{N_{i,j}}{N_i}$. This probability measure, though, doesn't improve the quality of our data[5].

On the other hand, probabilities calculated with respect to additional information can improve the quality of the model and make it possible to derive more information. We create the dataset $LM' = N_{total} + LM$, with $N_{total} = 356$ the total number of chapters of the novel. Then we can compute the probability that in a randomly selected chapter Ch_i and Ch_j are present as $P''_{i,j} = \frac{N_{i,j}}{N_{total}}$.

LM', containing the set of characters, the number of common chapters per tuple of characters and the total amount of chapters for the novel is an $L3$ data abstraction. We used LM' to compute the probabilities that characterize the relations and create an $L2$ model, containing characters and relations between them, labeled by independent probabilities. Such a model provides broader knowledge about the overall novel, while keeping the complexity of the representation at the same level as the initial dataset.

People, though, rarely pick up random chapters to read, but start from the beginning towards the end. We may be interested in the distribution of characters in the course of the novel. This knowledge cannot be derived from the $L2$ probabilistic graph discussed previously. That is due to the generality of the model, which omits information about chapter numbers[6]. By collecting chapter numbers per character, we could create another $L3$ model. Such information, though, is unavailable in our dataset, that is why further analysis of this model is impossible.

Probabilistic transformation of LM leads to additional knowledge about co-appearance of characters and the novel as whole. Queries like "What is the probability of reading about Character 1 and Character 2 together within X chapters randomly selected from the novel?" or "What is the minimum number of chapters to be read in order to connect Character 1 with Character 2?" can then easily be answered with ProbLog. For example, the probability that *Cosette* and *Jean Valjean* appear together in 1 chapter is 0.08707865. ProbLog can also compute the minimum number of chapters to be read in order to "connect" *Napoleon* and *Mlle Vaubois* - it is 4.

[5] We evaluate the quality of the data as the accuracy with which they represent the whole system.

[6] "Les Miserables" is divided into five volumes including 48 books or 356 chapters.

Figure 2 gives a visual interpretation of the five levels that logically can represent the novel "Les Miserables".

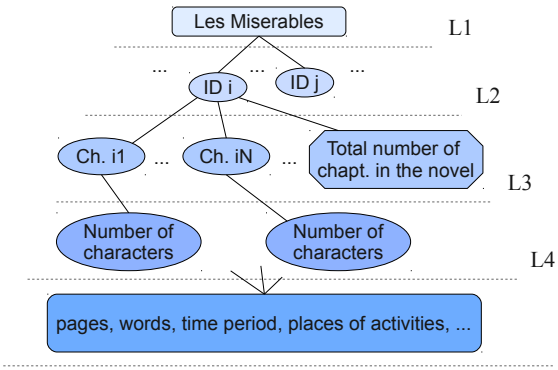

Fig. 2. "Les Miserables" dataset, with detail representation for each of the five levels of abstraction.

4.2 The Dictionary

The dictionary dataset (denoted as D) includes around 250 words (adjectives, verbs and nouns) from the English language and meanings for about 30 of them. Each word is related to a subset of other words, usually interpreted as synonyms – two words are synonyms if they have identical meanings. We can write a logic program that uses these data to determine that two words are synonyms, although the words are not directly connected by the synonymy relation.

Reasoning, based on such deterministic facts is, though, problematic and may lead to incorrect conclusions. That is why by attaching probabilities to synonymy relations we aim to quantify the actual resemblance between words. Encoding the probabilistic data into a ProbLog program allows querying about possible synonyms or meanings.

Our dictionary dataset D can be viewed as an $L2$ model where its elements are not monolithic but tuples of two objects - a word and a meaning. Unfortunately this simple view turns out to require complicated background knowledge to reason about. On the other hand, the dataset itself suggests an $L3$ representation where words are main elements and their meanings – sub-elements. We build a (probabilistic) logic program to model and reason about D as an $L3$ representation. Thus, we view the set of words as a (probabilistic) graph, with edges corresponding to their similarity relations. Meanings are not directly included in the graph. They are derived after a query on the graph has be conducted. Querying this model combines the simplicity of an $L2$ word-to-word similarity relation and the inclusion of a single fact to relate words with meanings. This model implies $L3$ knowledge.

Aiming to enhance our data with realistic similarities between words is a challenging task. The most accurate measure would involve comparing complete meanings of words and contexts where they are used. It is, though, impossible.

We observe, that the definition of synonymy implies comparison between word meanings and contexts. This suggests that a word is completely characterized by its synonyms. Our dataset (an $L2$ representation) provides this type of information, hence we can use it in computing probabilities. Algorithm 1 computes probabilities from the set of synonyms for a specific word.

Algorithm 1 Calculation of word similarity as a probability by sets of synonyms

```
INPUT:   word A, list of synonyms SAs = {SA1, SA2 ... SAn}
OUTPUT:  a list with tuples (SAi, A, P) defining the similarity
         between SAi and A, for i = 1 .. n
1.   ScoreList = null
2.   TempList = null
3.   P = 0
4.   M = 0
5.   for each SAi
5.1.     findall synonyms of SAi and store them in TempList
5.2.     for all mutual words between SAs U A and TempList U SAi
         increase M by one
5.3.     set P = M/N
5.4.     set TempList = null
5.5.     append (SAi, A, P) to ScoreList
6.   return ScoreList
```

Although intuitive, this approach returns faulty results due to the incomplete data - all possible words and synonym relations is impossible to be included.

The second observation that contributes to calculating more realistic similarities is: the more frequent a word is, the broader its definition and the higher its ambiguity becomes. Thus, we enhance the similarity calculation with a factor $f\omega$ inversely proportional to the usage frequency of a word: $f\omega = \frac{1}{lg(\omega_i)}$, where ω_i is the usage frequency of word i and lg is the logarithm with base 10. Hence, we modify the similarity function as shown in Algorithm 2.

This approach gives rather high importance to the usage frequency, which in some cases may be incorrect. Results were compared with "Measures of Semantic Relatedness (MSR)"[7] and showed to be very close. Therefore we model our probabilistic graph with the values derived by Algorithm 2.

For the above mentioned dataset, ProbLog infers that the word "cute" has the meaning "having qualities that give great pleasure or satisfaction to see, hear, think about" with probability 0.9735087. Also, the meaning that best suits the word "sweet" is computed to be "having qualities that give great pleasure or satisfaction to see, hear, think about" with probability 0.7817569.

Probabilistic reasoning about meanings and semantics applies in many fields, including robotics. In robotics a perceived object is categorized probabilistically into a class with which it has highest resemblance.

[7] "MSR" source: `http://cwl-projects.cogsci.rpi.edu/msr/`

Algorithm 2 Calculation of word similarity as probability by sets of synonyms and usage frequencies

```
INPUT:   word A, list of synonyms SAs = {SA1, SA2 ... SAn}
OUTPUT:  a list with tuples (SAi, A, P) defining the similarity
         between SAi and A, for i = 1 .. n
1.   ScoreList = null
2.   TempList = null
3.   P = 0
4.   M = 0
5.   FqAcc = 0
6.   TempFq = 0
7.   for each SAi
7.1.     findall synonyms of SAi and store them in TempList
7.2.     for each mutual word Bj between SAs U A and TempList U SAi set
         TempFq = the usage frequency ratio between A and Bj
7.3.     FqAcc = FqAcc + TempFq
7.4.     increase M by one
7.5.     P = FqAcc/M
         set TempList = null
7.6.     append (SAi, A, P) to ScoreList
8.   return ScoreList
```

4.3 The Worm

The last experimental data represent a real biological system - the caenorhabditis elegans (c. elegans) nematode. C. elegans is a worm, reaching 1 mm of size with transparent body. The size and the transparency allow exhaustive researches on many aspects, e.g. behavior - reproductivity, nervous and muscular systems, responses to irritations, etc. Our research targets the neural network - a grid of 302 neurons sending and receiving signals from other neurons.

A bit of biology[7]: A neuron is a biological cell that can be viewed as a single core processing unit with multiple inputs and outputs. The neuronal cell consists of a body, dendrites, an axon, and axon terminals. The dendrites are many small extensions of the body that conduct signal inside the body. The body represents the processing core - it is where incoming signals are accumulated and a decision on whether an outgoing signal to be fired or not is taken. An outgoing signal travels along the axon - a very long extension of the body that conducts the output signal towards other cells. The end of the axon is a tree-like structure consisting of the so called axon terminals - end points that convert the electrical signals into chemical ones and distribute them to other neurons. The connection between the axon terminals and the other neurons is called a synapse. A synapse, though, does not represent a physical connection between the two. Rather, the signaling is conducted via chemicals, called neurotransmitters, into a medial space - the synaptic cleft. The environment of the synaptic cleft allows chemicals

to travel from the axon terminals to the dendrites of the other neuron. The dendrites are equipped with elements that are able to receive specific chemicals and to translate them as electrical signals into the body. This ends the cycle of neuro-signal transmission. Figure 3 depicts a biological neuron.

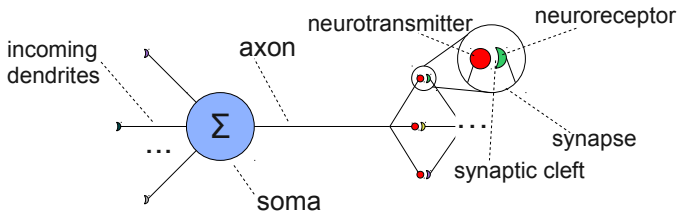

Fig. 3. A simplified representation of a biological neuron.

We want to reason about how the body determines whether a signal should be triggered. The load of the incoming electrical signals is accumulated and if it reaches a threshold then a spike is produced. This spike is called an action potential, and the threshold at which it appears is usually around +30 mV. Triggering an action potential results in depolarization and sets the neuron back to a resting state at -70 mV.

Signals can carry two types of electrical load - polarizing and depolarizing, thus characterizing the signal as excitatory or inhibitory. Mathematically, the former is a positive value while the latter is a negative one. The type of incoming signals determine the response of the cell. By propagating this knowledge we can investigate which neurons are activated and what is the character of their response. The accumulation L based on the positive or negative signals can be expressed mathematically as $L = \sum_{i=0}^{N} S_i$, where i denotes the i^{th} input to the neuron and S_i is the value of the signal.

Aims, approach and the data: The data we model are a set of neurons and mutual connections. Each connection is also labeled with a number, which defines how many actual connections appear between two neurons. Our research targets retrieving knowledge about single neuron activities, about signal activity between tuples of distinct neurons, and about larger groups of neurons. In order to achieve our goals, we need to determine the type of signal sent from one neuron to another as precisely as possible. This information is not directly available but is encoded in the relation between neurotransmitters and neuroreceptors.

Compiling the data was, by itself, a hard problem. The neural network of c. elegans has two almost separated parts - a pharyngeal and a somatic network which are documented mostly separated. Thus, data from different sources [8], [9], [10] had to be compared, cleared and transformed in order to become usable for our needs. After initial preprocessing we derive a deterministic $L2$ representation which we encoded in a logic program as:

Template	Example
`connection(N1, N2, T, C).`	`connection('ADAL', 'AIBR', 'Send', 2).`

where $N1$ is the origin neuron, $N2$ - the target neuron, $T = \{\text{Send, GapJunction}\}$ - the type of connection[8] where 'Send' means a synaptic connection, and C defines the count of connections between the two neurons.

Then additional information to allow defining the signaling is collected, namely the neurotransmitter and neuroreceptors that characterize every single neuron[11]. In the $L2$ representation neurons are the main elements. Adding neurotransmitters and neuroreceptors to our data results in an $L3$ representation, where to each neuron correspond a neurotransmitter and a set of neuroreceptors. In our logic program we encode this information as the predicate fact:

Template	Example
`neuron(ID, N, Tr, R).`	`neuron(1, 'ADAL', 'Glutamate', 'DOP#1').`

with ID denoting the id number of a neuron, N - its (unique) name, Tr - the neurotransmitter and R - the neuroreceptor.

To some elements, the corresponding sub-elements are unknown. Thus, we look for a method to derive such information and we rely on probabilities to quantify the possible inaccuracies. To reason about the relation between elements on a concrete level requires the use of more detailed data. Adding the information about neurotransmitter-neuroreceptor connections sets our detail level to $L4$. We focus on how data are processed to determine unknown values for neurotransmitters and neuroreceptors.

We wrote a Prolog program to automate this process. It considers neuron to neuron connections and receptor to transmitter relations. Initially, not all receptors and transmitters of the neurons are known. Our program derives the K best neuroreceptors for N based on the knowledge about its incoming signals, expressed by specific neurotransmitters, together with the neurotransmitter - neuroreceptor relations. We also derive the most probable neurotransmitter for N by combining the known neuroreceptors of the neurons to which N sends signals, with the neurotransmitters with which the receptors interact. The iterative process ends when no new information is found.

Relying on such information we build a program to determine unknown neurotransmitters and neuroreceptors based on connections between a neuron with known elements and a neuron with unknown ones. For example: for the neuron $'ALA'$ it was determined with probability 0.8 that it transmits *Glutamate*; for $'RIVR'$ the neuroreceptors $DOP-1$, $GGR-3$, $GAR-2$, $AVR-14$ were determined with probabilities 0.3, 0.3, 0.2, 0.1 correspondingly. The Prolog program serves to retrieve probabilities that are to be used in an $L3$ model.

Creating and learning the probabilistic graph is an ongoing process which we will not describe in detail but only outline its main idea. Our data collection includes possible signals for a tuple [neurotransmitter-neuroreceptor]. We build

[8] Connections can be synapses or gap junctions. Gap junctions are direct membrane-to-membrane connections which allow direct signal flow from one cell to another.

a probabilistic graph of neurons and define edges as probabilistic signals. Two approaches are considered - either using excitatory and inhibitory signals separately or setting one probability value to determine the character of the signal: the higher the value, the more excitatory the signal is.

Data so complex easily spread over all of the three main levels with plenty of sub-elements and factors. Our stepwise approach and the strict discrimination between different levels allow comprehensive models to be determined and to be used as a basis for solving our main goals.

5 Conclusions and Future Work

This paper investigates data abstraction for modeling complex systems in the context of probabilistic logic programming. Abstraction is a process of removing details in order to lower complexity. Whether this removal is good or bad depends on the context. Typically a model is based on a particular application. Sometimes we can find a good abstract representation for a set of concrete details: the reasoning we do on the abstract level gives us the results we need for the intended application. For real-world systems this is no longer the case: when we abstract away details, we loose information and we no longer obtain the desired results.

In this paper we show how probabilities can become part of the abstraction process. Probabilities allow us to express uncertainty about the relation between objects. Moreover, probabilistic inference can be used for reasoning.

[1] investigates an approach for leveled data abstraction with respect to a specific problem - causes of cancer. Also, [12] and [13] use detail abstraction to reduce the complexity of their data and quantify the quality of their results. We take a more general approach by presenting a framework for a broad range of problems and we use probabilities to quantify data imprecisions.

In the paper we propose the explicit use of different levels of abstraction. We also show that probabilities are typically computed based on details available in a level that has the relevant information. Once the probabilities are computed, we do no longer have to deal with the details. Implicit probabilistic information can be computed at the higher level of abstraction, e.g., once we have computed probabilistic relations between the main elements of an $L2$ representation from an $L3$ representation, we reason only on $L2$, ignoring all the complexity of $L3$.

We illustrated our approach with 3 examples. As is clear from the "Les Miserables" example, sometimes a dataset is available which contains very limited information. For the "Les Miserables" dataset, it was clear that more information was needed to compute correct probabilities. Here we started at $L2$ and we looked for the relevant $L3$ information. The worm example is another extreme. In order to solve our problem, we had to consider one big collection of all kinds of information. Thinking in terms of our levels helped to come up with an adequate model for the signal transmission between neurons. We believe that our approach is beneficial for this kind of modeling.

We showed that making a distinction between main and sub-elements can be helpful. If we do not make the split, we might end up with complex notions

that require complex logic. Consider the dictionary of Section 4.2. It clearly illustrates that is better to reason at level L3 with words being the main elements and meanings the sub-elements, then to use one single element for each word-meaning pair.

In the future, we aim to extend and automate the proposed approach, to complete the probabilistic model of the c. elegans and to use ProbLog as a platform to learn from it.

References

1. Kramer, S., Pfahringer, B., Helms, C.: Mining for causes of cancer: Machine learning experiments at various levels of detail. In: In Proceedings of the Third International Conference on Knowledge Discovery and Data Mining (KDD-97), Menlo Park, CA, AAAI Press (1997) 223–226
2. Kimmig, A., Santos Costa, V., Rocha, R., Demoen, B., De Raedt, L.: On the efficient execution of ProbLog programs. In de la Banda, M.G., Pontelli, E., eds.: ICLP. Volume 5366 of Lecture Notes in Computer Science., Springer (2008) 175–189
3. Shterionov, D., Kimmig, A., Mantadelis, T., Janssens, G.: Dnf sampling for problog inference. In: International Colloquium on Implementation of Constraint and LOgic Programming Systems (CICLOPS), Edinburgh, Scotland, The UK, July 2010. (July 2010)
4. Bryant, R.E.: Graph-based algorithms for boolean function manipulation. IEEE Trans. Computers **35**(8) (1986) 677–691
5. Valiant, L.G.: The complexity of enumeration and reliability problems. SIAM Journal on Computing **8**(3) (1979) 410–421
6. Knuth, D.: The Stanford GraphBase: A Platform for Combinatorial Computing. Addison-Wesley, MA (1993)
7. Reichert, H.: Introduction to Neurobiology. Oxford Univ. Press (1992)
8. Albertson, D.G., Thomson, J.N.: The pharynx of caenorhabditis elegans. Phil. Trans. R. Soc. London **B 275** (1976) 299–325
9. White, J.G., Southgate, E., Thomson, J.N., Brenner, S.: The structure of the nervous system of the nematode caenorhabditis elegans. Phil. Trans. R. Soc. London **B 314** (1986) 1–340
10. Oshio, K., Morita, S., Osana, Y., Oka, K.: "c. elegans synaptic connectivity data". Technical report, Keio University (1998)
11. www.wormatlas.org
12. Sadler, C.M., Martonosi, M.: Dali: a communication-centric data abstraction layer for energy-constrained devices in mobile sensor networks. In: Proceedings of the 5th international conference on Mobile systems, applications and services. MobiSys '07, New York, NY, USA, ACM (2007) 99–112
13. Cui, Q.: Measuring data abstraction quality in multiresolution visualization. IEEE InfoVis **12** (2006) 183–190