UNIVERSIDAD AUTÓNOMA DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

Bachelor's Degree in Computer Science

TRABAJO DE FIN DE GRADO

Symbolic XAI: automatic programming II

Author: Carlos Ignacio Isasa Martin

Tutor: Alireza Tamaddoni Nezhad

may 2021

Contents

Abstract

Explainable artificial intelligence (XAI) is a field blooming right now. With the popularity of opaque systems, the need of explanation methods that shed light on how this systems works has risen as well. In this work, we propose the usage of symbolic machine learning systems as explanation methods, a line that is yet to be fully explored. We will do this by reviewing this symbolic systems, analyzing the existing taxonomies of explanation methods and fitting the systems within the taxonomies. Finally, we will also do some testing on solving numerical problems with symbolic systems.

Keywords

Artificial Intelligence, Explainability, Symbolic Machine Learning Systems, Logic.

Resumen

La inteligencia artificial explicable (XAI) es un campo en pleno desarollo en la actualidad. Con la popularidad de los sistemas opacos, la necesidad de métodos explicativos que iluminen su funcionamiento ha aumentado también. En este trabajo propondremos el uso de sistemas simbólicos de aprendizaje automático como método explicativo, una línea de investigación que no se ha tenido en cuenta. Lo haremos empezando con una revisión de estos sistemas y continuando con un análisis de las taxonomías de métodos explicativos, para luego encuadrar los sistemas en dichas taxonomías. Finalmente llevaremos a cabo una serie de tests sobre sistemas simbólicos para ver su comportamiento frente a problemas numéricos.

Palabras Clave

Inteligencia Artificial, Explicabilidad, Sistemas Simbólicos de Aprendizaje Automático, Lógica.

CHAPTER 1

Introduction

Artificial Intelligence (AI) has risen in popularity through the last decade, as both its academic and industrial applications have grown. This recent developments have had opaque decision systems such as Deep Neural Networks (DNN) at the helm, while symbolic AI systems have been relegated to the background. However, as this black-box methods are increasingly being used to make important predictions in critical contexts [1], interpretability and explainability have become pressing issues: if things go wrong, how can we explain why? how do we deal with scenarios in which ethics matters?, how can we ensure fairness in processes such as Human Resources' hiring and firing or in forensics reports for the courts?

All of this has given birth to recent studies in explainable artificial inteligence (XAI), a field that studies how and why should we make systems that can be explained. However, these problems have pertained researchers since the early days of AI, as Michie[2] defined, as early as 1988, a classification for machine learning systems. Michie argued that *strong* machine learning systems are those that produce a declarative version of the process under consideration, while *ultra-strong* systems are those that produce knowledge about the process under consideration that could be further used to generate new knowledge about the process. This classification is still relevant to this day, with new systems being developed with this definitions in mind.

Symbolic machine learning systems (symbolic systems from now on) are a subset of AI systems, characterized by taking declarative inputs, using declarative methods to process them and outputting declarative statements, making them by design ultra strong systems. These systems were one of the earliest forms of AI, but, as time went on, they were phase out in favour of more adequate systems for the problems relevant at the time, such as the aforementioned DNNs. Nevertheless, symbolic systems have become again a relevant topic, due to the need of potent interpretable systems. This newfound relevancy has led to breakthroughs[3][4][5][6][7] in the field, making them, again, an interesting option.

The recent breakthroughs have created symbolic systems that not only keep mathematical guarantees of finding the correct solutions, but are efficient in doing so, making them a valid alternative to opaque systems that are currently en vogue. However, the state of the art of XAI has consistently ignored symbolical systems, which shows a disconnection between numerical systems experts and symbolic systems experts.

This is what motivated us to do this project, a survey of the state of the art of both symbolic systems able to generate explanations and XAI, that tries to fit the systems into the already existent classifications of XAI implementations.

Our objectives are: first to discuss the state of the art of symbolic systems able to generate explanations and XAI, paying special attention to XAI taxonomies that classify the current accepted explanation systems. Second, to fit those symbolic systems in those taxonomies, showing their spot in the already existing categories or creating new ones. And finally, testing some of the symbolical systems, trying to prove that they are able to handle numerical problems.

CHAPTER 2
State of the Art

2.1. Symbolic Systems Able to Generate Explanations

2.1.1. Inductive Logic Programming

Inductive Logic Programming (ILP) is a symbolic artificial intelligent system anchored in the concept of inductive logic (i.e. formally generalizing concepts from concrete examples), using logic programming to represent background knowledge (BK), positive examples, negative examples and learned programs. Its goal is to learn a hypothesis that, with the BK, explains all the positive examples and no negative one[8].

There is a wide variety of implementations of this paradigm, as ILP is one of the oldest learning systems, but the most relevant of them are anchored in the concept of generalization and specialization. We say that a clause A is more general than a clause B iff. $A \vdash B$ and $B \nvdash A$, analogously B is more specific than A. With this we could give the hypothesis space the structure of a lattice which we can travel by generalizing or specializing. Early ILP systems used the concept of inverse entailment which refers to the idea that if the BK and the hypothesis entail the positive examples ($BK, H \vdash E^+$), then $BK, \neg E^+ \vdash \neg H$. This systems used inverse entailment with a single example to build the bottom clause, which is the most specific clause that covers that example with the BK and then used a wide array of techniques to generalize bottom clauses into a hypothesis that covers all examples.[9]

On the other hand, new ILP systems like meta-interpretative-learners (MILs) use constraints on the hypothesis space such as metarules for MILs. This constraints reduce the search task, while allowing for new features, such as predicate invention through applying those constraints to new predicates.[3]

2.1.2. ILASP

ILASP is an ILP system based on answer-set programming (ASP)[10]. ASP is based around the notion of answer sets (or stable models) which are used to solve logical problems with negated rules in the body. Suppose a set of rules R where each rule may have negated grounded atoms in the body. We pick a set of atoms M that appear in the rules as derivable atoms and we assume that no atom outside from M is derivable so we can drop every rule with atoms not a where $a \in M$. Once this is done, we are left with a reduct of the program R'. If the set of atoms we can derive from R' coincides with M we can say that M is an answer set of R. ASP allows users to define the specification of a problem, rather than an algorithm to solve it, and then an ASP solver searches for the possible solutions, which are the answer sets. This differs from Prolog programs, as those can only have one model (the Herbrand model) while ASP programs may have many models. The nature of ASP makes ILASP radically different than other ILP systems as ILASP mainly focuses on finding the logical specification of the problem instead of the program itself[11].

ILASP's learning tasks have another component apart of the BK and the examples, called the mode bias or language bias. This mode bias is used to define the hypothesis space easing its navigation. ILASP's examples are also different from ILP's ones, as these examples entail partial interpretations that should (positive examples) or should not (negative examples) be answer sets of the learned program. Finally, ILASP allows for ordering examples, which in turn allows for preference learning (i.e. learning the user's preference for a type of solution).

The original ILASP1 mathematical basis lays in a generate-and-test approach. For this, ILASP1 generates the set of hypothesis that cover all positive examples and brave ordering examples (i.e. there should be at least one pair of answer sets that extend the referred examples which are ordered according to the ordering example). We call all the hypothesis belonging to the set *positive hypothesis*. Among these positive hypothesis we may find some that do not cover negative examples or cautious ordering, these are *violating hypothesis*. It is mathematically provable that the inductive solution of the learning problem is the set of positive hypothesis that are not violating hypothesis, so the algorithm of ILASP1 computes all positive hypothesis from the hypothesis space (which is easy given that we define the hypothesis space through the language bias) of a given length, starting at 1. Then it adds all the violating hypothesis as constraints and checks if there are any remaining positive hypothesis, increasing the length if not. Once there is a set of positive hypothesis that are not violating hypothesis it is returned as the solution.

ILASP2 brought improvements by replacing the idea of violating hypothesis with *violating reasons*. In general there are two reasons why a hypothesis may be violating, there could be an interpretation that accepts a negative example or there could be two interpretations that contradict a cautious ordering example. The set of all violating interpretations and interpretations pairs of a violating hypothesis is called violating reason. The algorithm now, which starts with an empty set of violating reasons, picks a positive hypothesis, checks if it has any of the violating reasons already detected, if not checks for new ones and finally if it finds no violating reasons whatsoever returns it as the learned program.

A whole change of paradigm was brought by ILASP2i, which changed ILASP from a batch learner (a learner that learns from all examples at the same time) to a middle ground of a batch learner and a loop learner (a learner that incrementally considers each example). It does this by using a set of examples called the *relevant examples*, using ILASP2 to find a hypothesis that covers all relevant examples and finally checking if any of the examples left out is not covered. If this happens, the example is added to the relevant examples and if not the hypothesis is returned as the induced program.

Both the changes induced in ILASP2 and ILASP2i were motivated by scalability issues. However ILASP3 was intended to learn from noisy examples, a hard task for a inductive learner. For this, ILASP3 uses an approximation function for the hypothesis, that contains at least all examples covered by it (initialized to be the complete set of examples). Then, it checks if any of the examples in the approximation coverage of the hypothesis is actually not covered by it. If this happens, ILASP3 updates the coverage function by changing the *coverage constraint*, a constraint that keeps account of which examples are not covered by which class of hypothesis (using *hypothesis schemas*, a set of structural conditions defining some class of hypothesis). ILASP3 keeps generating hypothesis that are optimal to the approximate coverage and updating it until it is not able to find an example in the approximate coverage that is not covered, returning the hypothesis found. This method is overall slower than ILASP2i, but allowing learning from noisy datasets is worth depending on the problem.[4]

Lastly, ILASP4 brings improvements on the paradigm that ILASP3 created. ILASP3 coverage constraints were necessary and sufficient to explain while certain hypothesis rejected certain examples and this made computations of those constraints pretty expensive. ILASP4 relaxes those constraints and makes them only neccesary, decreasing computation times by a huge margin. Overall, ILASP4 is able to achieve a similar performance than ILASP2i, while keeping the ability to learn from noisy datasets from ILASP3.[11]

2.1.3. Inductive Functional Programming

Inductive Functional Progamming (IFP) works on the same basis as ILP but instead of learning logical programs it learns functional programs. Its similarities to ILP lead to having closely the same characteristics, such as being an ultra-strong machine learning system and being transferable, but without the capacity of using logic entailment IFP requires different algorithms to move in the search-space. This algorithms used to be classified into two groups, analytical or generate-and-test, however recent innovations in the field made possible to combine both techniques in what is called the analytically-generate-and-test approach.

Analytical IFP systems, like *Igor II*[12], generally work solely with I/O examples, using recurrences in the given examples which then are generalized to recursive functions so hypothesis are, computed instead of searched[13]. This approach minimizes the search-space but needs a big set of I/O examples that covers all complex inputs to correctly limit the search-space, which in turn ends up causing a slow down on the synthesis of the function, creating a trade-off between efficiency and user specification[14].

On the other side, generate-and-test systems, like *MagicHaskeller* was originally, exhaustively search for all the possible programs that comply to a given set of constraints (which may be incomplete). To do this optimally it uses De Bruijn lambda calculus, so all α-equivalent expressions are also syntactically equivalent, and a triebased memoization system that stores in its leaves the different results of the memo function, as doing this with the function that travels the search space provides a time save[15]. All of this makes generate-and-test very time consuming for some assignments, but it allows it to have no limitations on the given specifications. Moreover, this systems are able to learn functions from only one example, provided it's general enough, allowing the user to not be as exhaustive as with an analytical system [14].

Finally, analytically-generate-and-test systems, like *MagicHaskeller* is now[14], divide the examples into two subsets, one that guides the search analytically and another one that serves as a test for the stream of function that the analytical search finds. Using the first subset to narrow the search space and the second one to reflect the user intentions allows analytically-generate-and-test systems to evade the trade off that analytical systems must endure[14].

2.1.4. LFIT-PRIDE

Learning From Interpretation Transition (LFIT) [16] has been proposed to automatically construct a model of the dynamics of a system from the observation of its state transitions. Given some raw data a discretization of those data in the form of state transitions is assumed. Several model extend from LFIT but we will focus ourselves in PRIDE[7], a multi valued logic based model.

Multi valued logic atoms are of the form a^{val} where v is a variable and $val \in \mathrm{dom}(a)$. The domain of each atom is a subset of natural numbers and may be different for each atom. Rules are of the form $v_0^{val_0} \leftarrow v_1^{val_1} \wedge \dots v_n^{val_n}$, where the head $(h(R))$ is on the left side of the arrow and the body $(b(R))$ is on the right side. We now proceed to separate the set of variables into two disjoint sets, $\mathcal{T}$ (targets) and $\mathcal{F}$ (features), such as $\forall R, \mathrm{var}(h(R)) \in \mathcal{T}$ and $\forall v \in b(r), v \in \mathcal{F}$. Then we define a set of functions, called discrete states, from variables in either $\mathcal{T}$ or $\mathcal{F}$ to the natural numbers. This discrete states can be equivalently represented by a set of variables $\mathcal{S}^{\mathcal{F}} = \{v^{s(v)} | \forall v \in \mathcal{F}\}$ (the same works for $\mathcal{T}$). A state transition is a pair $(s, s') \in \mathcal{S}^{\mathcal{F}} \times \mathcal{S}^{\mathcal{T}}$ Finally, we can say that a rule R matches a discrete state $s \in \mathcal{S}^{\mathcal{F}}$ $(R \sqcap s)$ if $b(R) \subseteq s$.

Given a set of observations T (which is a set of state transitions), PRIDE will learn a set of rules P that explains T: $\forall (s, s') \in T, \forall v^{\mathrm{var}} \in s', \exists R \in P, R \sqcap s, h(R) = v^{\mathrm{var}}$ and is correct w.r.t. T: $\forall R \in P, \forall (s1, s2) \in T, R \sqcap s1 \implies \exists (s1, s3) \in T, h(R) \in s3$. Finally, PRIDE also guarantees that the rules are minimal: $\forall R \in P, \nexists R', h(R) = h(R'), b(R) \subsetneq b(R')$.[17]

2.1.5. Evolutionary Algorithms

Evolutionary Algorithms (EAs) are a family of algorithms that take inspiration from the concept of "survival of the fittest", prevalent in biology, applying this idea over fitness functions to be maximized.

The first step in this application is called representation, where the real world problem is mapped to the problem solving space. Candidate solutions are called phenotypes or individuals, while their encoding (the points in the search space) is called genotype. Crucially, this relationship is one to one. Genotypes will have placeholders for objects to be placed in them, we call these placeholders genes and the objects alleles. Population is initialized as a random sample of the genotype space and its size will be constant through the learning progress. This induces competition through limitation of resources.

Next we need an evaluation function. This function takes a genotype, converts it into a phenotype and evaluates it. The evaluation function must be constructed for each problem in a way that the target phenotype is the one that maximizes it. The population will be evaluated at each step of the algorithm with individuals with high scores being more desirable.

After the population is evaluated the algorithm will pick parents for the next generation. The higher the score, the higher the probability of that individual being picked, but all of them have a minimum chance of being picked to prevent EAs to get stucked at a local maximum.

Once the parents have been picked, we generate children through two methods, mutation and recombination. Mutation takes one parent and induces random changes into the allele allocation, bringing "new blood" to the population. Recombination takes n parents and combines them into one offspring, randomly selecting which parts are to be kept from each parent. The logic behind recombination is that merging individuals with distinct but desirable properties we can produce one that combines them.

Finally we need to replace individuals in our population to keep the size constant. Replacement is, most of the time, done heuristically, with an emphasis in the evaluation of each individual. We are left then with a population that consists of a mix of old individuals and offsprings.

This algorithm is done until an termination condition is met. Usual termination conditions are CPU time limits, the reaching of an upper bound in the evaluation of an individual or that the evaluation improvement is under a threshold for enough iterations.

The EA-based field of machine learning is usually called automatic evolutionary programming. The first implementation of this kind of algorithms was Genetic Programming, which learns Lisp programs through the use of EA. It uses a tree structure as the genotypes where the genes are the leaves of the tree and the alleles are either functions or variables. Genetic Programming takes advantage of Lisp syntax as a functional programming language, which adapts very well to the tree structure. [18]

Grammatical Evolution [19] is one of the most famous and successful linear genotyped approaches to automatic evolutionary programming. The key idea is to design a separate genotype-to-phenotype mapping module that incorporates the context free grammar of the programming language under consideration. Genotypes are vectors of integers and the mapping procedure sets a simple transcription mechanisms that allow produce a syntactically correct program from the genotype. In this way, the genetic level remains always the same.

Further extensions to Grammatical Evolution such as [20, 21] replaces context free grammar by respectively attribute grammars and Christiansen grammars that produce programs that are both syntactically and semantically correct. These approaches offers the researchers the complete control of the kind of generated program. These contributions show that it is not needed to add to much semantics to dramatically reduce the search space. Computational costs of such a sophisticated translation process suggest to use these approaches only in the case of really huge search spaces.

There is another really innovative and interesting approach to automatic evolutionary programming that overcomes some of the drawbacks of these other approaches. They use the so called *straight line programs or SLPs* [22]. This approach can be considered an hybrid between linear and tree-likes genotyped methods because of the mixed nature of SLPs. Among its main features we can mention:

- SLPs can be seen in same way as quadruples in the sense in which compilers use them. They are thus, in fact, a way of expressing algorithms close to low level code (assembler) and are, because of that, very efficiently handled.
- The genetic operators proposed by the authors ensure to keep semantic blocks considering every sub expression as a semantic block avoiding the destructive effects of typical and general mutation and crossover operators.

From the viewpoint of approaches to generate explanations all these methods can be considered strong machine learning algorithms because the code itself for a task should be considered a declarative version of the task itself. Grammatical Evolution and its extensions provide the greatest flexibility while SLPs could result rather obscure as explanations due to the low level nature of quadruples.

2.1.6. DeepProbLog

DeepProLog is a learning system that integrates logic reasoning with a neural network, using a probabilistic logic language called ProbLog. The structure of a ProbLog program consists in a set of probablistic facts $\mathcal{F}$ of the form $p :: f$, where p is a probability and f a ground atom, and a set of rules $\mathcal{R}$. As a consequence, the result of a query in ProbLog is the probability of that query being inferred.

DeepProbLog addition is neural annotated disjunctions (nAD), expressions of the form $nn(m_q, \mathbf{t}, \mathbf{u}) :: q(\mathbf{t}, u_1); \ldots ; q(\mathbf{t}, u_n) : -b_1, \ldots, b_m$, where b_i are atoms, $\mathbf{t}$ is a vector containing all neural inputs for predicate q, $\mathbf{u}$ is a vector containing all neural outputs and m_q represents the neural network model we are using. This sentence makes DeepProbLog learn from the examples through a neural network a probability for each $q(\mathbf{t}, u_i)$ to be true with all the others being false (so the sum of all probabilities is 1) given that the atoms $b_1, \ldots, b_m$ are true.

Once the neural network side of DeepProbLog has learnt the probabilites of the nAD, the system learns a logical program by doing gradient descent over the ProbLog program. For this to happen we first have to define the *gradient semiring*, a semiring whose elements are tuples of the form $(p, \frac{\delta p}{\delta x}$ where p is a probability as in ProbLog and $\frac{\delta p}{\delta \mathbf{x}}$ is the gradient with respect to a parameter $\mathbf{x}$ a vector of x_i, each defined as p_i, the probability of probabilistic fact with a learnable probality $(t(p_i) :: f_i)$.

We can rewrite the labels of all probabilistic facts to include these gradients:

$$L(f) = (p, \mathbf{0}) \quad \text{for facts} \quad p :: f \quad \text{with fixed probability} \quad p \tag{2.1}$$

$$L(f_i) = (p_i, \mathbf{e}_i) \quad \text{for facts} \quad t(p_i) :: f_i \quad \text{with learnable probability} \quad p_i \tag{2.2}$$

with $\mathbf{e}_i$ beign the i-th canonical vector. As this this is a semiring it has two operations $\oplus$ and $\otimes$ defined as follows:

$$(a_1, \mathbf{a}_2) \oplus (b_1, \mathbf{b}_2) = (a_1 + b_1, \mathbf{a}_2 + \mathbf{b}_2) \tag{2.3}$$

$$(a_1, \mathbf{a}_2) \otimes (b_1, \mathbf{b}_2) = (a_1 + b_1, b_1\mathbf{a}_2 + a_1\mathbf{b}_2) \tag{2.4}$$

which correspond to the OR logic operator and the AND logic operator respectively. This way we can do gradient descent over logical operators.

The method we just discussed is built as a black box, only communicating with the neural network through the nAD, which serve as an interface. As an example of why this kind of learning is useful we can see that the the DeepProbLog system was able to give programs for operations over the MNIST dataset, where the neural network mapped the handwritten digits to integers while the logical part learnt the program that solved the operations from a couple of examples.[6]

2.1.7. δILP

δILP is a reimplementation of ILP in an end-to-end differentiable structure, trying to combine the advantages of ILP with those of numerical learning systems such as noise tolerance and the ability to work with unseen data. Although the concept is similar to DeepProbLog, the mathematical basis behind it is pretty different, as δILP was designed to work on its own, not supporting a neural network.

Before studying the nature of the algorithm behind δILP we should define some early concepts. A language frame $\mathcal{L}$ is a tuple $(target, P_e, arity_e, C)$ where $target$ is the predicate to learn, P_e is a set of given (extensional) predicates that will be used in the BK, $arity_e$ is a map from both the target predicate and those predicates in P_e to the natural numbers giving their arity and C is the set of constants present on the ILP problem. A program template Π, which specifies the range of possible programs (like metarules in MILs o mode bias in ILASP), is a tuple $(P_a, arity_a, rules, T)$ where P_a is a set of auxiliary invented predicates, $arity_a$ their arities, $rules$ is a map from each invented predicate to a pair of rule templates (τ_p^1, τ_p^2). Finally a rule template τ describes the clauses that can be generated and is a pair (v, int), where v is a natural number that specifies the number of existentially quantifiable variables, while int can be 1 or 0 whether it allows the clause to uses invented predicates or only extensional predicates, respectively.

We start by labeling each example,creating a set of tuples of the form $\{\gamma, 0\}$ or $\{\gamma, 1\}$, with the label being 0 if the example γ is negative or 1 if it is positive. We now construct a differentiable model that implements conditional probability for the label λ of the example γ:

$$p(\lambda|\gamma, W, \mathcal{L}, \Pi, \mathcal{B}) \tag{2.5}$$

where $\mathcal{L}$ and Π are the aforementioned language frame and program template, while W is a set of clause weights and $\mathcal{B}$ is the background knowledge. We want our predicted label to match the real one so we want to minimize the logarithmic likelihood when sampling pairs from the label set, so we want to minimize:

$$loss = -\mathbb{E}[\lambda \dot{\log}(p(\lambda|\gamma, W, \mathcal{L}, \Pi, \mathcal{B})) + (1-\lambda)\dot{\log}(1 - p(\lambda|\gamma, W, \mathcal{L}, \Pi, \mathcal{B}))] \tag{2.6}$$

Now we need to compute the value of $p(\lambda|\gamma, W, \mathcal{L}, \Pi, \mathcal{B})$ and for this we use four auxiliary functions:

$$p(\lambda|\gamma, W, \mathcal{L}, \Pi, \mathcal{B}) = f_{extract}(f_{infer}(f_{convert}(\mathcal{B}), f_{generate}(\mathcal{L}, \Pi), W, T), \gamma). \tag{2.7}$$

$f_{generate}$ generates a set of clause from the language frame and the program template using the rules defined in the program template. $f_{convert}$ takes the BK and returns a sequence of n zeroes or ones where n is the number of possible ground atoms and the i-th entry is 1 or 0 if the ground atom is in the BK or not respectively. f_{infer} is the most important function of the bunch, returning a valuation (a function that assigns every grounded atom a real number between 0 and 1) that represents the candidate hypothesis. It uses the clause weights to represent the likehood of each clause being in the final hypothesis. Finally $f_{extract}$ evaluates the example γ with the valuation returned by f_{infer}.

In conclusion what δILP does is turning an ILP problem into a classification problem, where the system must correctly classify examples into correct or incorrect. To do this, it finds the weights that return the hypothesis that correctly classifies all examples.[5]

2.2. Explainability

2.2.1. Overview

There has been a recent surge in publications about explainability and explainable artificial intelligence, corresponding with the recent developments in AI. This publications range from trying to standardize a set of good practices for developing XAI to discussing methods to explain AI systems. We shall first give an overview of the field before moving to discussing our main taxonomic references.

Defining what constitutes an explanation is an open debate in epistemology and philosophy of science[23], so defining what is *explainable* AI and *interpretable* AI is not an easy task which is why many of the publications belonging to this field start with a revision of those terms[24][1][25][26]. For this work, we define *interpretable system* as systems that generate their own explanations and *explanatory methods* as the methods that try to explain systems.

Metrics are another relevant topic in XAI, as evaluating which explanations are better is key to finding this explanations. However, it is still very much an open problem, with several approaches and with most of the current metrics having a lot of room for improvement. One of the approaches is evaluating explanations relating them to human interaction[27], factoring how humans understand and prefer explanations. This is, undoubtedly, a useful way to look at explanations, however human interaction may decrease in relevance in more general cases. Another approach is evaluating explanations on how good are them at explaining the system's particularities while remaining consistent[28], highlighting the features that are important to the system while ignoring the irrelevant ones and being stable to small changes in the input.

Lastly, we should highlight that symbolic systems tend to be ignored in the status quo of explainability which may signal a disconnect between numerical systems experts and symbolic systems experts. Some reviews[26] reference systems such as ILP, but they do so with outdated references that do not show the opportunities that this systems may present. Among the little amount of topics with relations to symbolic systems that appear in XAI publications is rule sets, which can be seen equivalent to logic systems. Rule set extraction is one of the possible methods to explain AI systems[29][30], but there is no discussion of its relationship with first order logic. Another interesting, but much more recent, point is hybrid systems that use their symbolic part solely to create explanations[31][32]

Now we are going to shift our focus to the study of four[1][28][33][25] taxonomic reviews of explainable systems, chosen because they give an exhaustive review of those systems, and while [28] focuses on Graph Neural Networks (GNNs), the discrete character of GNNs provides some similarities to symbolic systems. This study of taxonomic reviews is done in order to place the symbolic systems we have already discussed among these taxonomies.

2.2.2. Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI

This paper[1] aims to be an exhaustive review of the explainability field, starting by defining the key concepts, strongly relating them to human interaction, and reasoning why XAI should be taken into account in the future of AI. Another interesting point is the section about challenges and future research needs, where the authors outline the need of a set of standardized metrics among other points, like neglecting the idea of the existence of a tradeoff between accuracy and explainability in a model or addressing the security issues of XAI, regarding things like adversarial AI. However, our focus is on the taxonomy that the authors give, which we proceed to describe:

- **Transparent Models**: models that are explainable by themselves, such as linear regression, decision trees, KNN, etc.
- **Post-hoc explainability techniques**: as the name denotes, this denotes all techniques that are used to explain already developed models. Among this techniques the authors differentiate two big approaches: model agnostic techniques and model specific techniques.

- **Model agnostic techniques**: these techniques are meant to work on every system. The writers separate this techniques into three groups:
  - **Model simplification**: in this group we find techniques that try to extract a transparent model from the system, usually rule sets. In this group we can find systems like *LIME*[34] or *G-REX*[35] among others.
  - **Feature relevance**: techniques belonging to this group describe the functioning of a system by ranking the importance each feature of the input has on the output. Approaches such as *SHAP*[36] or *ASTRID*[37] are in this group.
  - **Visualization techniques**: these procedures return a visual explanation of the system. A portfolio of this techniques can be found here[36], but there are also other techniques not in the aforementioned paper such as sensitivity maps[38] or individual conditional expectation plots[39].
- **Model specific techniques**: approaches grouped in this category are tailor made for a single model, so we will use the models as a first categorization
  - **Tree ensembles and random forests**:
    - **Model simplification**: this techniques reduce tree ensembles or random forest to single decision trees[40]. However not many techniques can be found in this category, as model agnostic techniques cover tree ensembles well enough.
    - **Feature relevance**: classic methods used were based on measuring the mean decreased accuracy and mean increase error after permuting random variables of the input[41], while more recent ones pose recommendations that when taken, switch examples from one class to another.
  - **Support Vector Machine**:
    - **Model simplification**: this type of explanation systems is the prevalent one regarding SVMs. This simplification is done mostly through rule extraction[42].
    - **Feature relevance**: feature relevance is done mostly through rule extraction[43], which is prevalent in SVMs explanations.
    - **Visualization techniques**: there are several visualizarion techniques over SVMs, from visualizing which of the input variables are actually related with the associated output data[44], to using heat maps combined with the output[45] and statistics about the margins between the classes[46].
  - **Multi-layer neural networks**: as the number of layers grows, it gets harder and harder to provide a good simplification for the system, so the main method is feature relevance. However there are still examples of model simplification approeaches
    - **Model simplification**: DeepRED[30] is the most relevant approach, allowing rule extraction and simplification into decision trees in the system.

    - **Feature relevance**: since simplification in this systems is difficult, feature relevance is the most widely used approach. DeepLIFT[47] does this by evaluating the difference between a reference activation level and the actual activation level for each neuron.
  - **Convolutional neural networks**: as CNNs are mainly used in image processing, their explanations are mostly visual. We can separate those in two main categories that mix feature relevance and visualization:
    - **Mapping the output to the input**: The most relevant work of this group[48] uses another neural network, feeding it the output feature maps, to compute the maximum activation level from each layer, which can be used to reconstruct the parts of the image that produce that activation, using a saliency map to show them.
    - **Mapping the intermediate layers to the input**: This type of explanations try to show what does each layer of the CNN see. To do this, they use a neural network that generates the most representative image of a given neuron[49].
  - **Recurrent Neural Networks**: given the long term dependencies of RNNs, simplification approaches are not recommended for this systems. However other approaches are able to capture this dependencies.
    - **Feature relevance**: through back-propagation, with rules that account for long short term memory and gated recurrent units[50].
    - **Visualization techniques**: using finite horizon n-grams that discriminate cells in LSTM and GRU systems[51].
    - **Architecture modification**: modifying the underlying architecture of the system to include a subsytem that output an explanation. In [52] the writers implement an attention network, which highlights the information flow.

2.2.3. Explainability in Graph Neural Networks: A Taxonomic Survey

From the viewpoint of the survey[28] itself, it actually explains how the traditional methods for explaining Deep Learning environments have been tested in GNNs and how and why most of them failed. The interest of this survey for us is how it deals with the characteristics of the GNN's domain as it is a clear case of discrete input data environment.

- **Instance Level Methods**: this methods provide input-dependant explanations for each input graph. We can split them into 4 categories, depending at the type of algorithm they use to evaluate how relevant is each feature in the input.
  - **Gradients/Feature-Based Methods**: gradient-based methods compute the gradient of a prediction with respect to the input by back-propagation, while feature-based methods map the features of the nodes in the final cape to the input space, trying to find the most important features. Both

  approaches have its problems, as the first one can only show the sensitivity between input and output and struggles with inputs in the saturation zone between two classes. The second assumes that the final node embeddings can reflect the input importance, which is dependant of the implementation and can only explain graph classification problems. Models belonging to this class are SA[53], Guided-BP[53], CAM[54] and Grad-CAM[54].

  - **Perturbation-Based Methods**: the basis of this technique lies in making small changes to the input and studying the output, as ideally, if the important information remains unchanged, the output should remain equal. All implementations of this approach use masks as an interface between the input and the GNN, altering the input. This masks are recomputed after every iteration trying to find the important features. It is worth noting that most of this systems suffer from the *introduced evidence problem* where alterations of the input may introduce new semantic meaning. Methods belonging in this group are GNNExplainer[55], PGExplainer[56], GraphMask[57] and SubgraphX[58].
  - **Surrogate Methods**: surrogate methods use a simple and explainable model to approximate the input graph and a small locality around it. A suitable compirason could be Taylor series in mathematical analysis. Usual problems relating to these methods are its inability to provide a completely trustworthy explanation and that some of its implementations are not able to deal with all GNN problems. The main implementations are GraphLime[59], RelEx[60] and PGM-Explainer[61]
  - **Decomposition Methods**: this methods break down the output score of the system into terms, assigning terms to different input features. However, as the number of possible features in input graphs is pretty high (taking into account nodes, edges and edge labels) so thing such as graph walks or graph structure are not able to be taken into account by some models. Models in this group are LRP[53], Excitation BP[54] and GNN-LRP[62].

- **Model Level Methods**: the kind of explanations that should belong to this group are those that are able to explain the whole system. However, there is only one implementation of this kind of explanation techniques, due to the complexity of dealing with discrete inputs. XGNN[63] is a system that explains GNNs at a model level by using reinforcement learning to build a graph generator which generates graph patterns. This patterns match each possible output and are considered an explanation. However, XGNN can only explain graph classification problems, while it currently is unable to explain node classification.

2.2.4. Explaining Explanations: An Overview of Interpretability of Machine Learning

This review[33] is a mix of the two that we have already seen. It is focused on one particular type of AI systems, deep network systems, but it also aims to give a solid background in XAI. The taxonomy this publication gives is based on what the methods are trying to explain:

- **Explanations of deep network processing**: deep networks derive their decision using a large number of elementary operations, so in order to explain this process we should reduce the number of operations. There are several methods of doing this, but most of them fall in the *model simplification* category discussed previously.
  - **Linear proxy methods**: models like LIME[34] fall in this category, as it uses perturbations on inputs to probe the behaviour of the system. This behaviour is then built into a linear model, which is transparent.
  - **Decision trees**: simplifying a deep network using a decision tree is also an interesting option. Among the systems that do this we can find the aforementioned DeepRed[30].
  - **Automatic rule extraction**: rule extraction is widely used for deep networks. DeepRed[30] can also simplify a deep network into rule sets.
  - **Saliency maps**: creating maps that show which parts of the input have more influence over the output is a relevant visualization technique. Here we find algorithms such as CAM[64].
- **Explanation of deep network representations**: another interesting thing is trying to explain the role of singular components of the network in the whole system.
  - **Role of layers**: understanding the role of each layer in a deep network is a hard but useful task. In order to do this we can test a new problem on the layer once the deep network is already trained. This process is called *transfer learning* and we can find a framework for it here[65].
  - **Role of individual units**: layers can be divided further into neurons. This neurons can be explained qualitatively; sampling images that maximize its output or training a network to generate them; or quantitatively, testing their ability to solve a problem. A comprehensive survey of this methods can be found here[66].
  - **Role of representation vectors**: we can also try to explain linear combinations of neurons. Concept Activation Vectors[67] provide a framework to explain this subsystems.
- **Explanation-producing systems**: the final approach discussed by the paper is modifying the whole system to output an explanation.
  - **Attention networks**: attention networks capture the flow of the information, and although this can be seen as an explanation (even if it is not human readable), the most interesting approach is using a correct explanation to build an attention network (like modelling human attention in a natural language problem) and then using it to steer the information in the deep network[68].
  - **Disentangled representations**: this representations shows individual dimensions that represent the importance of features in the input. Deep

networks can learn this kind of representations[69] which are a valid explanation

  - **Generated Explanations**: deep networks can also be trained to output a human readable explanation along the solution[70]. The datasets for this kind of systems have large sets of human readable explanations used to train them.

2.2.5. A Survey Of Methods For Explaining Black Box Models

This paper[25] is definitely one of the most interesting papers for our approach. It takes a generic black box model and studies the different methods to explain it, grouped by their interaction with the black box.

- **Solving the model explanation problem**: the model explanation problem refers to the problem of finding an approximation (either local or global) of the black box model. It is akin to simplification methods or surrogate methods. Explanations in this category are divided by the type of approximation they return.
  - **Explanation by decision trees**: like the already mentioned [40] or [30]
  - **Explanation by rule sets**: like the already mentioned[30] or [42]
  - **Agnostic explanations**: this are explanation systems that change the output model depending on the black box that they are trying to explain. Systems such as GAM[71] or PALM[72] belong in this category.
- **Solving the outcome explanation problem**: the outcome explanation problem refers to finding an explanation for a specific input and its vecinity. It is akin to instance level methods.
  - **Explanation by saliency maps**: saliency maps are used to highlight the important part of the input. Models such as this are CAM[64] or attention network models[73].
  - **Agnostic explanations**: as it was explained before, this models explain all black boxes. A possible example is LIME[34].
- **Solving the inspection problem**: the inspection problem refers to the problem of trying to understand the inner workings of a black box. It is similar to explaining deep networks representations
  - **Sensitivity Analysis**: sensitivity analysis studies the correlation between the uncertainty in the output and the one in the input. Examples of methods belonging to this class are QII[74] or Integrated Gradients[75].
  - **Partial Dependence**: partial dependence study the relation between response of parts of the system and the predictor in a limited feature space. Examples of methods belonging to this list are individual conditional expectation plots[39] or Prospector[76].

- **Solving the transparent box design problem**: the transparent box desing problem refers to the problem of building a system that is equivalent to the original black box but is explainable locally or globally.
  - **Rule Extraction**: this method builds an equivalent predictor with a rule set. A notable example is CPAR[77], built over an old ILP system called FOIL[78].
  - **Prototype Selection**: a prototype is an object that is representative of a set of similar instances. In this case, prototypes are made for subsets of the dataset where each of those subsets has the same output. Notable examples are PS[79] and BCM[80].

CHAPTER 3

Symbolic Systems in Explanation Taxonomies

3.1. Overview

In order to overcome the reasons that originate the lack of symbolic methods in current taxonomies we propose in this contribution to add to the criteria used for classifying XAI methods the following questions:

1. Declarative (discrete) nature of input data
2. Declarative (discrete) nature of the target kind of explanation
3. Declarative (discrete) nature of the method to generate explanations

Once we answer those questions for the symbolic systems and for the methods categorized by the surveys, we will be able to fit the studied systems into the surveys.

3.2. Classification of Symbolic Systems

Symbolic Systems							
Criterion	ILP	ILASP	IFP	LLFIT	EA	DPL	δ ILP
1.	Yes	Yes	Yes	Yes	Yes	Mixed	Yes
2.	Yes	Yes	Yes	Yes	Yes	Yes	Yes
3.	Yes	Yes	Yes	Yes	Yes	No	No

After describing all of these systems all the answers feel pretty natural. The hybrid models, as expected, use numerical methods to generate the programs (which are the explanations themselves) and DPL has input data both declarative (logical facts) and numerical (the input of the neural network and the probabilities for each probabilistic logical fact).

3.3. Classification of the Categories in the Surveys

We proceed to apply our criterions to the different taxonomies that we have already discussed. Doing this will show that symbolic models are not considered in any of the categories of those taxonomies.

3.3.1. Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI

Classification of Survey 1			
Method	Criterion 1	Criterion 2	Criterion 3
Transparent Systems			
LR	No	No	No
kNN	No	No	No
Decision Trees	No	Yes	Yes
Rule Sets	No	Yes	Yes
Model Agnostic Techniques			
Model Simplification	No	Depends	No
Feature Relevance	No	No	No
Visualization techniques	No	No	No
Tree Ensembles & Random Forests			
Model Simplification	No	Yes	No
Feature Relevance	No	No	No
SVMs			
Model Simplification	No	Depends	No
Feature Relevance	No	Depends	No
Visualization techniques	No	No	No
Multi Layer NN			
Model Simplification	No	Depends	No
Feature Relevance	No	No	No
CNNs			
output$\rightarrow$ input	No	No	No
intermediate layers$\rightarrow$ input	No	No	No
RNNs			
Feature Relevance	No	No	No
Visualization techniques	No	No	No
Architecture modifications	No	No	No

The taxonomy that [1] proposes fits in a very good way with our criterions, as almost all methods answer either positively or negatively to our questions. However, model simplification methods depend on which transparent model the system is reduced to, as the answer to the second criterion changes if the model gives the explanation in a symbolical way or not.

3.3.2. Explainability in Graph Neural Networks: A Taxonomic Survey

Classification of Survey 2			
Method	Criterion 1	Criterion 2	Criterion 3
Instance Level Methods			
Gradient/Feature-Based Methods	Yes	No	No
Perturbation Based Methods	Yes	No	No
Surrogate Methods	Yes	Depends	No
Decomposition Methods	Yes	No	No
Model level methods			
XGNN	Yes	Yes	No

We consider graphs to be a symbolic way to convey information, so all the inputs are symbolical. Again, for surrogate methods, it depends on which transparent system is used to simplify the model. Finally, XGNN gives explanations in the form of graph patterns, which are symbolical due to them being generalized graph.

3.3.3. Explaining Explanations: An Overview of Interpretability of Machine Learning

Classification of Survey 3			
Method	Criterion 1	Criterion 2	Criterion 3
Explanations of deep network processing			
Linear Based Methods	No	No	No
Decision Trees	No	Yes	No
Rule extraction methods	No	Yes	No
Saliency maps	No	No	No
Explanation of deep network representations			
Role of layers	No	No	No
Role of individual units	No	No	No
Role of representation vectors	No	No	No
Explanation-producing systems			
Attention Networks	No	No	No
Disentangled representations	No	No	No
Generated explanations	No	Yes	No

This survey is not a good fit with our criterions, as the different categories do no correspond to different types of explanations, but rather to the different things that can be explained in a deep network. A noteworthy point is that the *generated explanations* framework returns text based explanations, which we consider symbolic.

3.3.4. A Survey Of Methods For Explaining Black Box Models

Classification of Survey 4			
Method	Criterion 1	Criterion 2	Criterion 3
Model explanation problem			
Decision Trees	Depends	Yes	No
Rule sets	Depends	Yes	No
Agnostic methods	Depends	Depends	No
Output explanation problem			
Saliency maps	Depends	No	No
Agnostic methods	Depends	Depends	No
Inspection problem			
Sensitivity analysis	Depends	No	No
Partial dependence	Depends	No	No
Transparent box problem			
Rule extraction	Depends	Yes	No
Prototypes Selection	Depends	No	No

As this survey uses a generic black box, the input data is neither described nor accounted for, so for every category the answer to our first criterion is that it depends on the system. Another thing that depends on the system is the answer for the second criterion for agnostic methods, as the output depends on the system.

3.4. Placing the Symbolic Systems

We are going to begin with the first review[1]. One can say that, in a vacuum, symbolic systems fall into **transparent systems**, but the interesting thing about the systems we have discussed through this work is that they have a mathematical guarantee to find the correct explanation if able to solve the problem. This makes them a perfect candidate for the category of **model simplifications**, not only as simplifications, but as replacements that find the correct solutions with no approximation needed.

Moving onto the second review[28], we can see that there is no category that fits with our models, as they are model level methods but not on the same way as XGNN. Surrogate methods comes close, as, as previously said, symbolic systems work as replacement without approximation. But since this survey only contemplates surrogate methods as instance level methods, we propose a new category, **model level surrogate methods**.

For the third review[24], symbolic systems could be positioned in either one of two categories. One can see them as **explanations of deep networks processing**, or as an **explanation-producing system** when the whole deep network is substituted by the symbolic system.

Finally, the last survey[25] is the one where the place of our system is most clearly defined. Symbolic systems are **solutions to the transparent box problem**, as they are equivalent systems that are equivalent to the black box model in question while being globally interpretable.

CHAPTER 4

Tests

4.1. Testing Objectives

For testing we chose to showcase how purely symbolic systems fare with numerical problems. There are many examples of symbolic models solving symbolic problems[81] and even outperforming numerical machine learning systems in some cases[82]. However there is not much literature about these kind of systems trying to solve more "numerical" problems.

These kind of problems are key examples to understand why symbolic AI is such an interesting line of research. Suppose some kind of bias is found in the example set, be it sexual, racial, etc, which could easily not be found by the users given a large enough dataset. A classic black-box, numerical-based, machine learning system will detect it as a pattern and have it factored in the output of the system without the user knowing it. However, symbolic systems by design will output an explanation of the decision process, allowing the user to see the bias from the get go.

4.2. System Election

We chose to try the purely symbolic systems as we wanted to test how a system with no numerical parts would deal with this data. This limited our options to ILP systems, IFP systems, ILASP and LFIT. We discarded IFP systems early in testing as the most recent breakthrough in the field was in 2012[14] and we wanted to focus on systems with open lines of investigation. Furthermore, a quick overview of system showcases its focus on recursive functions, which is not a natural solution to our problem. A couple of small tests showed that the system is still in an experimental phase and as its support was abandoned we decided to leave it out of the relevant tests. We also discarded LFIT as a very similar work was done on it recently[17].

This leaves us with ILP systems and ILASP as our test subjects. As there is only one implementation of ILASP (with different iterations, yet ILASP4 has showed to be arguably the best one to date[11]) there is no possible argument. However for ILP there has been several implementations through the years, so we had to choose one. The criterion we used was a mixture of recency in the development and relevancy in the field. With this criterion in mind Metagol[83] was an obvious choice, as its last update was recent enough and its backed by the most important researchers in the field. Popper[84] was also a candidate, being the most recent system developed by Andrew Cropper, the co-creator of Metagol. However, Popper is still at early development stages so right now Metagol is a safer option. Nonetheless it could be an interesting option for the future.

4.3. Problem Specification

The problem we chose to test is a resume classifier system. The input for this problem is a set of resumes from a set of possible workers and the output would be a ranking of those resumes. For this, the system will assign each curriculum a score depending on the values of different fields (such as experience, recommendation letters, etc), effectively rendering the problem as finding the weights of a weighted average from examples.

It is key to understand what this example truly is, which is a proof of concept. This is why the problem is simplified and stripped of all fluff, just to see if there is a viable implementation in the systems we are discussing. The intention is not building an application that classifies resumes, we are just trying to find the limits of this systems as problem solvers.

4.4. Implementation and Results

4.4.1. Metagol

To try to implement the problem we reduced it to its simplest form. We started with two different attributes, *experience* and *recommendation*, which were valued either at 1 or 2, having or not having each thing respectively. Experience was weighted as 75% of the score, while having a title was weighted 25%. The expected solution would be `prio(A,T):-exp(A,E),rec(A,R), T is 3*E/4 + R/4.`, unfortunately Metagol is only able to learn logical clauses so the arithmetic operations wont work. We tried to bypass that in our first try by giving the system a couple of "weight" functions. The finale program for our first attempt is

```
:- use_module('../metagol').

% tell Metagol to use the BK
body_pred(exp/2).
body_pred(rec/2).
body_pred(quar/2).
body_pred(tquar/2).

% background knowledge
% experience 1: hasn't, 2: has
```

```
%% rec 1: hasn't, 2: has

exp("a",1).
exp("b",1).
exp("c",2).
exp("d",2).

rec("a",1).
rec("b",2).
rec("c",1).
rec("d",2).

quar(A,B):—B is A/4.
tquar(A,B):—B is (3*A)/4.
sum(A,B,C):—C is A+B.

%% metarules
metarule([P,Q,R,S,T,U], [P,A,F], [[Q,A,B],[R,A,C],[S,B,D],[T,C,E],[U,D,E,F]]).

:—
 Pos = [
    prio("a",1),
    prio("b",1.25),
    prio("c",1.75),
    prio("d",2)
  ],
  Neg = [
  ],
  learn(Pos,Neg).
```

With this attempt we only wanted to check if Metagol was able to find the correct solution, `prio(A,T):-exp(A,E),rec(A,R),tquar(E,EW),quar(R,RW),sum(EW,RW,T).`, which has the same structure as the metarule we provided. Metarules give the structure of possible solutions, narrowing the search space to clauses that have that shape.

Unfortunately, this did not work. Metagol got stucked more than three hours trying to find the correct hypothesis, which in a problem as simple as this should not happen under any circumstances. There also is no trace and no error message so we are left to speculate with the problem as debugging with swipl showed nothing abnormal. Taking inspiration from the ILASP implementation we opted to not normalize the values and just multiply them with 3 and 1, but got similar results. We believe that the problem is with how ProLog handles assignations and mathematical operations and how Metagol looks for logical functions.

The next implementation we tried was based around Metagol finding a list of weights. We defined a new function `average(A,W,R):- average_aux(A,W,0,R).` where `average_aux([A1|A2] ,[W1|W2], C, R):- D is C + A1*W1,average_aux(A2`

`,W2,D,R).` is an auxiliary function that helps with recursion. With this implementation, Metagol must find `W`, the correct weight list. Another change we made was turning the experience and recommendation values into values in a list, so for a person with experience valued at 1 and recommendation valued at 2, the data we gave to Metagol was of the form `charac("a", [1,2]).` The solution then would be `prio(A,T):-charac(A, C), average(C,[3,1],T)`, opting again into not using rational numbers. The resulting program was

```
:- use_module('../metagol').

% tell Metagol to use the BK
body_pred(average/3).
body_pred(charac/2).

% background knowledge
% charac(-key, -Values)
% Values: array of 2 integers
% Values[1]: exp, 2: has, 1:hasn't
% Values[2]: recommendation, 2: has, 1: hasn't
charac("d",[1,1]).
charac("b",[1,2]).
charac("c",[2,1]).
charac("a",[2,2]).

average([],[],0).
average(A,W,R):- average_aux(A,W,0,R).
average_aux([],[],R,R).
average_aux([A1|A2],[W1|W2],C,R):- D is C+A1*W1, average_aux(A2,W2,D,R).

% metarules
metarule([P,Q,R,W], [P,A,E], [[Q,A,B],[R,B,W,E]]).

:-
 Pos = [
    prio("a",8),
    prio("b",5),
    prio("c",7),
    prio("d",4)
  ],
  Neg = [
  ],
  learn(Pos,Neg).
```

Once again, this attempt was unsuccessful. The reason for it was that the `is\2` function is not invertible, so ProLog itself returns an error when `D is C + A1*W1` is reached and `D`, `C` and `A1` are all instantiated while `W1` is not.

We tried a few variations over those attempts but gave up shortly after. ILP has showed the ability to solve numerical problems through time[81], but the burden of the problem falls on the user, that has to come up with a correct logical implementation, or even make changes in the paradigm[85]. For that reason we think that, while it has the ability to do it, ILP in its current form is not really suited to solve numerical problems.

4.4.2. ILASP4

ILASP's preference learning seems tailor-made for this kind of problems so we used it on our implementation. We, again, chose the same two attributes, but this time we valued them between 1 and 5, thought we kept the same weighted relationship between the two. Incrementing the number of possible values gives us more possible examples and more possible comparisons between them.

As we are using preference learning, the solution ILASP gives us will take the shape of a weak constraint. Weak constraints order examples minimizing a weight associated with it and have the following structure: `:-a1,...,an[w@l,t1,...,tn]`, where `a1,...,an` are atoms either positive or negative, `t1,...,tn` are terms in the atoms, `w`is a term that corresponds with the weight and `l`is the level of the constraint. When an example contains the atoms `a1,...,an` a weight of level `l`is associated to it according to the term `w`. The examples will be ordered minimizing the weights priorizing the levels in a descending ordering, with each level having an infinite priority over the next.

Once this concept is defined, we can write the correct solution to our program. Given that the experience is worth 75% of the average and the title 25%, the correct constraint is `:- exp(A), rec(B).[-3*A-B@1, A, B]`. Since dividing by four all weights yields an equivalent ordering, and given that ILASP does not work that well with rational numbers we chose to ignore the normalization. There is also the need of turning the weight into a negative number, as minimizing it would be equivalent to maximizing its absolute value, which is what we want.

Another important issue is how do we define the hypothesis space. ILASP accepts mode biases (a description of which predicates can build a hypothesis and how to use them), metarules or an ASP that describes the hypothesis space as valid descriptions of the this space, but interestingly enough, it also accepts a literal definition of the hypothesis space. This latter definition is what we implemented in the end, as none of the other options admitted mathematical operations in the weight term of the weak constraint. This brute force implementation of the hypothesis space is used by the creators of the system in many of the examples, as it is a widely accepted way of defining the hypothesis space. For this first proof of concept we chose to use a search space containing 25 possible solutions, the combinations of multiplying each attribute with a number between 1 and 5. This yields a lot of equivalent candidate solutions, but we also wanted to check how ILASP dealt with that.

This resulted in a program with 3 clearly differentiated parts:

- **Examples:** ILASP examples are of the form `#type(tag,{P},{N},{C}).`, where the type can be either `#pos` or `#neg` when the example is a positive one or a negative one, respectively. `P` is a list with all the positive atoms in the clause while `N`is a list with all the negative ones. `C` is used in case the example is a context-dependent example, allowing it to come with an extra bit of background knowledge. In our case we only context-dependent examples, allowing us to bypass the background knowledge, but we could have done it giving the background knowledge that experience and title are can take values between 1 and 5. Our examples were of the shape `#pos(eg_1, {}, {}, {exp(5).rec(5).}).`

- **Ordering examples:** Ordering examples show which examples should be preferred over which others. There are two types of ordering examples, *brave orderings*, which express that at least one pair of interpretations extending the examples must respect it and *cautious orderings*, which express that every pair of interpretations extending the examples must respect it. As there are no interpretations that may extend our examples, it does not matter for us which one are we using, but we chose to use cautious interpretations as technically, if there were any interpretations that extend our examples, we want them to respect the ordering. This gives us the following structure for an ordering example: `#cautious_ordering(comptag, egtag1, egtag2, rel).`, where `comptag` is the ordering example's tag, `egtag1` and `egtag2` are the tags of the examples we are comparing and `rel` is the ordering relation between those examples. It is important to note that the relation `<` means that the first example has a higher priority than the second, contrary to common sense.

- **Definition of the hypothesis space:** As we already discussed, we implemented a literal definition of the hypothesis space. Each possible solution is written as `length ~ rule`, so our rules are of the form `2 ~ :- exp(A), rec(B). [-w1*A-w2*B@1, A, B]`, with `w1` and `w2` being numbers between 1 and 5.

We will not include the code itself here, as it is nearly 200 lines long, while being repetitive and already qualitatively described. Once we run this program the results were:

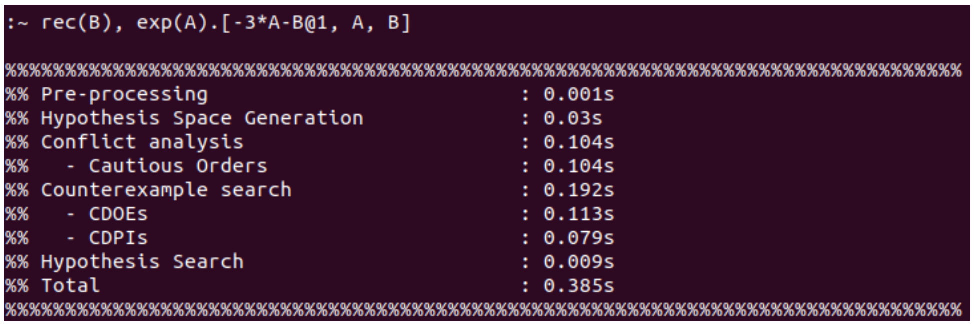
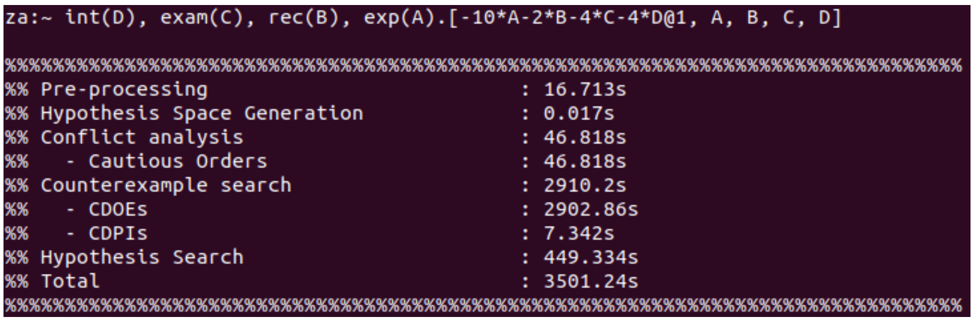

```
:~ rec(B), exp(A).[-3*A-B@1, A, B]

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Pre-processing                          : 0.001s
%% Hypothesis Space Generation             : 0.03s
%% Conflict analysis                       : 0.104s
%%   - Cautious Orders                     : 0.104s
%% Counterexample search                   : 0.192s
%%   - CDOEs                               : 0.113s
%%   - CDPIs                               : 0.079s
%% Hypothesis Search                       : 0.009s
%% Total                                   : 0.385s
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Figure 4.1: Results for the simple proof of concept

As we see, the system finds the correct solution pretty easily. It spent the majority of the time searching for counterexamples, with the most being spent looking for a

CDOE (context-dependant ordering example) counterexample and taking almost no time looking for a CDPI (context-dependant partial interpretation) counterexample. However, as this is a fairly small problem, it is very difficult to distinguish workload difference in the different parts of the process. That is why we decided to extend the size of the problem.

The new problem has 4 different attributes: experience(50%), recommendation(10%), interview(20%) and exam(20%). Each one is valued with an integer from 1 to 5, yielding 5^4 examples, while the amount of possible ordering examples increases to 5^8, all of them included in the program. We also increased the hypothesis space, including every possible combination of multiplying each of the values with a number from 1 to 10, which turns into 10^4 different hypothesis (and again, there are multiple equivalent candidate hypothesis, which will not matter as long as it finds a correct one). The program was generated using a script that wrote all the examples, then all the ordering examples and finally all the candidate hypothesis, as manually writing it would be impossible. After running it we got:

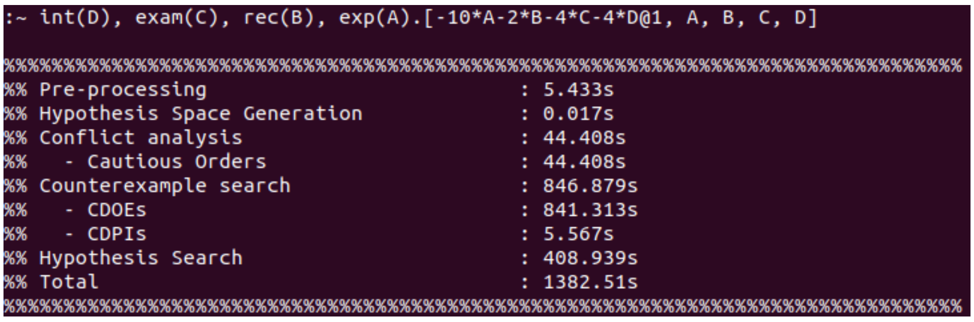


```
za:~ int(D), exam(C), rec(B), exp(A).[-10*A-2*B-4*C-4*D@1, A, B, C, D]

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Pre-processing                          : 16.713s
%% Hypothesis Space Generation             : 0.017s
%% Conflict analysis                       : 46.818s
%%   - Cautious Orders                     : 46.818s
%% Counterexample search                   : 2910.2s
%%   - CDOEs                               : 2902.86s
%%   - CDPIs                               : 7.342s
%% Hypothesis Search                       : 449.334s
%% Total                                   : 3501.24s
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Figure 4.2: Results for the large proof of concept, using all possible ordering examples.

As we see, it is even clearer now that the largest time sink is finding counterexamples among the ordering examples, but the hypothesis search also requires a large amount of time.

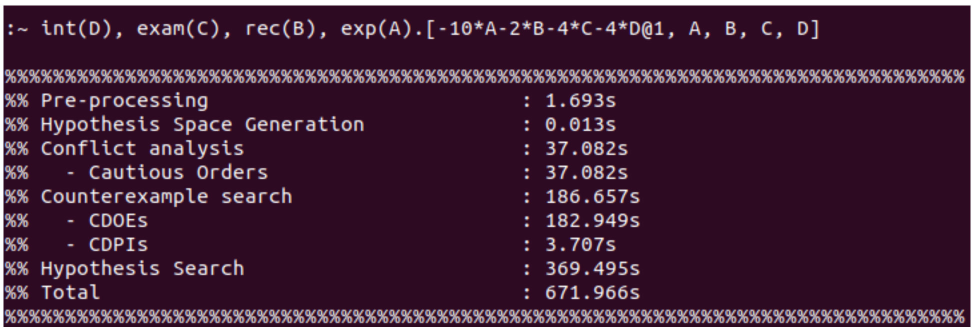
One of the key ideas behind symbolic systems is trying to learn from the fewest amount of examples possible. The amount of ordering examples in the previous test is huge, leading to a proof of concept with little practical utility. To solve this, we introduced a random variable in the script that would leave only one third of the examples, yielding the following result:



```
:~ int(D), exam(C), rec(B), exp(A).[-10*A-2*B-4*C-4*D@1, A, B, C, D]

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Pre-processing                          : 5.433s
%% Hypothesis Space Generation             : 0.017s
%% Conflict analysis                       : 44.408s
%%   - Cautious Orders                     : 44.408s
%% Counterexample search                   : 846.879s
%%   - CDOEs                               : 841.313s
%%   - CDPIs                               : 5.567s
%% Hypothesis Search                       : 408.939s
%% Total                                   : 1382.51s
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

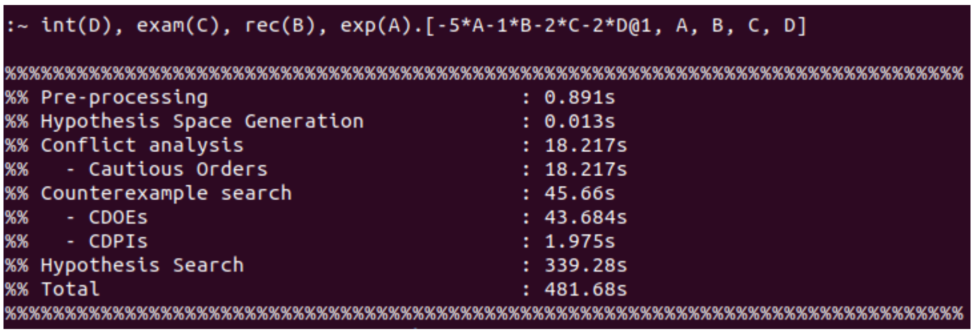
Figure 4.3: Results after using only a third of the ordering examples

The solution was still the correct one, so the system was robust enough to handle it. We then proceeded to keep only 10%, 5%, 2% and 1% of the examples.



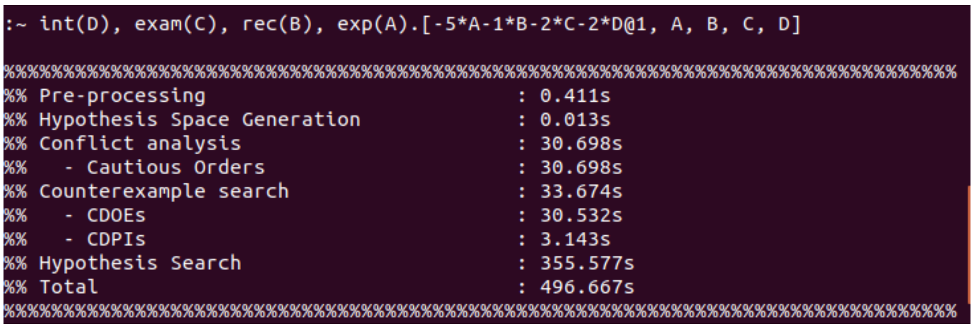
```
:~ int(D), exam(C), rec(B), exp(A).[-10*A-2*B-4*C-4*D@1, A, B, C, D]

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Pre-processing                          : 1.693s
%% Hypothesis Space Generation             : 0.013s
%% Conflict analysis                       : 37.082s
%%   - Cautious Orders                     : 37.082s
%% Counterexample search                   : 186.657s
%%   - CDOEs                               : 182.949s
%%   - CDPIs                               : 3.707s
%% Hypothesis Search                       : 369.495s
%% Total                                   : 671.966s
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Figure 4.4: Results after using only 10% of the ordering examples



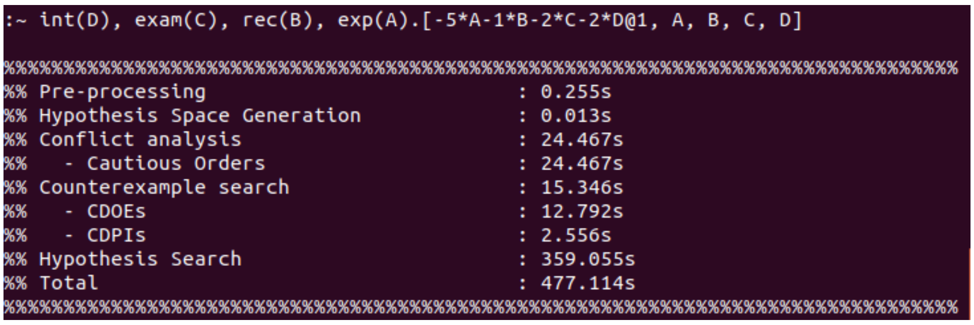
```
:~ int(D), exam(C), rec(B), exp(A).[-5*A-1*B-2*C-2*D@1, A, B, C, D]

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Pre-processing                          : 0.891s
%% Hypothesis Space Generation             : 0.013s
%% Conflict analysis                       : 18.217s
%%   - Cautious Orders                     : 18.217s
%% Counterexample search                   : 45.66s
%%   - CDOEs                               : 43.684s
%%   - CDPIs                               : 1.975s
%% Hypothesis Search                       : 339.28s
%% Total                                   : 481.68s
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Figure 4.5: Results after using only 5% of the ordering examples



```
:~ int(D), exam(C), rec(B), exp(A).[-5*A-1*B-2*C-2*D@1, A, B, C, D]

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Pre-processing                          : 0.411s
%% Hypothesis Space Generation             : 0.013s
%% Conflict analysis                       : 30.698s
%%    - Cautious Orders                    : 30.698s
%% Counterexample search                   : 33.674s
%%    - CDOEs                              : 30.532s
%%    - CDPIs                              : 3.143s
%% Hypothesis Search                       : 355.577s
%% Total                                   : 496.667s
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Figure 4.6: Results after using only 2% of the ordering examples



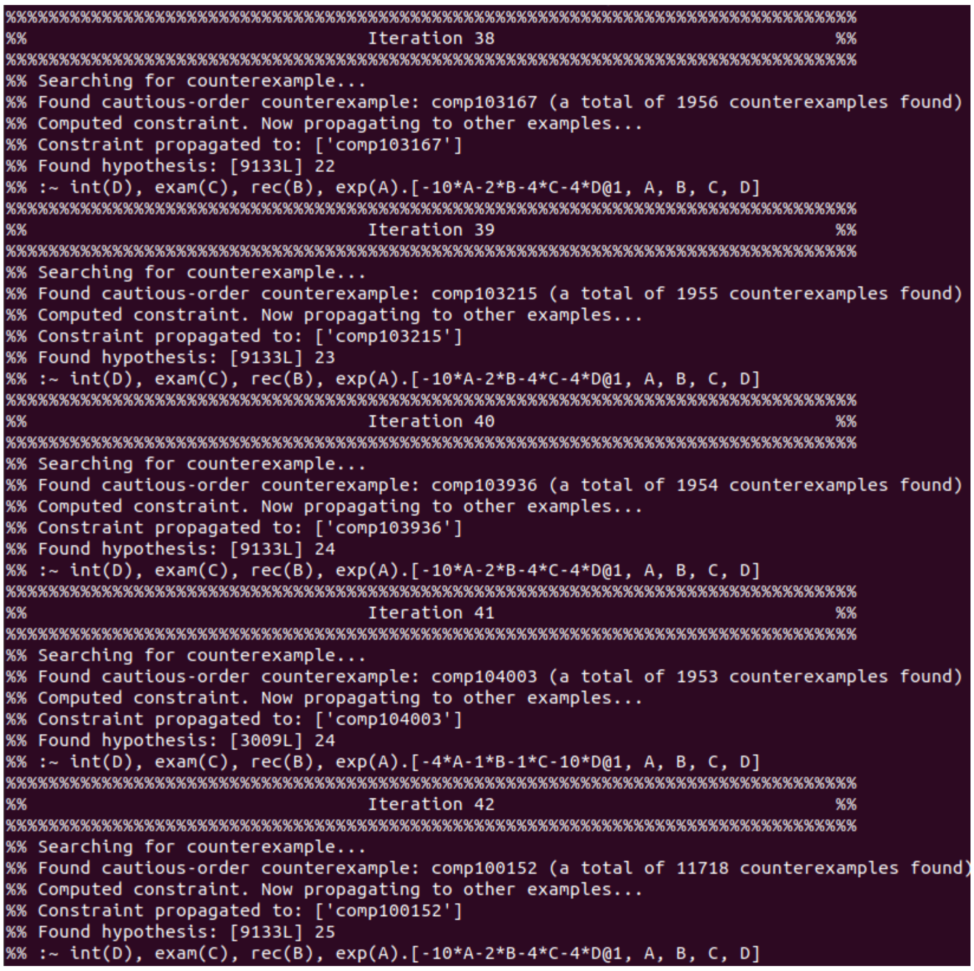
```
:~ int(D), exam(C), rec(B), exp(A).[-5*A-1*B-2*C-2*D@1, A, B, C, D]

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Pre-processing                          : 0.255s
%% Hypothesis Space Generation             : 0.013s
%% Conflict analysis                       : 24.467s
%%    - Cautious Orders                    : 24.467s
%% Counterexample search                   : 15.346s
%%    - CDOEs                              : 12.792s
%%    - CDPIs                              : 2.556s
%% Hypothesis Search                       : 359.055s
%% Total                                   : 477.114s
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Figure 4.7: Results after using only 1% of the ordering examples

As the results show, the system keeps finding the correct hypothesis (or one equivalent) even with only a hundredth of the ordering examples. We could not find an explanation of why the systems chose one hypothesis or the other, but whichever it chose in each variation, it was consistent. Printing the trace of the program did not show anything useful, as the first correct hypothesis the program found was the chosen one for the solution. Ultimately, it does not matter, as it found the correct solution.

As the number of ordering examples decreases we start to find a bottleneck in the system, the time spent search for hypothesis to test is now the largest time sink. This could be improved by removing all the equivalent hypothesis from the search space, but because we are testing the limits of ILASP and not trying to optimize this problem, we will leave that as future work.

ILASP is, in theory, capable of learning from noisy examples, so added it up to our problem in order to test this capabilities. We took the program with 10% of the ordering examples as our starting point. From there we introduced a random 10% chance that the ordering example was changed to a random order relation instead of the correct one. Unfortunately, this did not work as expected, as 8 hours into the experiment the system had only done 30 iterations, while probably needing around 2000. We will now show the printed trace of the last iterations before we ended ILASP and discuss it.



```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%                                Iteration 38                                %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Searching for counterexample...
%% Found cautious-order counterexample: comp103167 (a total of 1956 counterexamples found)
%% Computed constraint. Now propagating to other examples...
%% Constraint propagated to: ['comp103167']
%% Found hypothesis: [9133L] 22
%% :~ int(D), exam(C), rec(B), exp(A).[-10*A-2*B-4*C-4*D@1, A, B, C, D]
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%                                Iteration 39                                %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Searching for counterexample...
%% Found cautious-order counterexample: comp103215 (a total of 1955 counterexamples found)
%% Computed constraint. Now propagating to other examples...
%% Constraint propagated to: ['comp103215']
%% Found hypothesis: [9133L] 23
%% :~ int(D), exam(C), rec(B), exp(A).[-10*A-2*B-4*C-4*D@1, A, B, C, D]
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%                                Iteration 40                                %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Searching for counterexample...
%% Found cautious-order counterexample: comp103936 (a total of 1954 counterexamples found)
%% Computed constraint. Now propagating to other examples...
%% Constraint propagated to: ['comp103936']
%% Found hypothesis: [9133L] 24
%% :~ int(D), exam(C), rec(B), exp(A).[-10*A-2*B-4*C-4*D@1, A, B, C, D]
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%                                Iteration 41                                %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Searching for counterexample...
%% Found cautious-order counterexample: comp104003 (a total of 1953 counterexamples found)
%% Computed constraint. Now propagating to other examples...
%% Constraint propagated to: ['comp104003']
%% Found hypothesis: [3009L] 24
%% :~ int(D), exam(C), rec(B), exp(A).[-4*A-1*B-1*C-10*D@1, A, B, C, D]
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%                                Iteration 42                                %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Searching for counterexample...
%% Found cautious-order counterexample: comp100152 (a total of 11718 counterexamples found
%% Computed constraint. Now propagating to other examples...
%% Constraint propagated to: ['comp100152']
%% Found hypothesis: [9133L] 25
%% :~ int(D), exam(C), rec(B), exp(A).[-10*A-2*B-4*C-4*D@1, A, B, C, D]
```

Figure 4.8: Trace of the program with noise

Each iteration of the system took the hypothesis given from previous iteration, checks for counterexamples and creates a constraint for one of them. Finally it computes another hypothesis, optimal to the approximate coverage we talked about in chapter 2. As we see in the trace, ILASP was in the right track, computing the correct hypothesis, but cutting one counterexample at a time, it would take two thousand iterations for it to finish. We can also see in iteration 41 that ILASP gets sidetracked by an incorrect hypothesis, but as soon as it checks the number of counterexamples it goes back to the correct hypothesis. We thus can conclude that given enough time, ILASP would have returned the correct solution.

This tests showed ILASP's ability to deal with a numerical problem through preference learning. Even in a noisy setting, the system showed it could find the solution given enough time. As ILASP evolves with time, it will get more efficient, turning it into a practical option for this kind of problems.

CHAPTER 5

Conclusions & Future Work

The surveys of both symbolic systems able to generate explanations and XAI have been concluded successfully. We have shown the recent breakthroughs in those symbolic systems and studied their inner workings. We have also reviewed the state of the art of XAI, touching the hotspots and relevant points of the field. Once this was done, a classification of the symbolic systems among the existing surveys was done. We think that the taxonomy that suits this systems better is [25], where the symbolic systems fit perfectly in the category of **solutions to the transparent box problem**.

Regarding the testing of symbolic systems, two of them were tested, Metagol (an ILP implementation) and ILASP. Testing with Metagol was unsuccessful, as we were unable to find an implementation of our problem that worked. Recent research has shown, however, that ILP systems can handle numerical problems[85], but this are through modifications on the algorithm. On the other hand ILASP proved able to solve the problem, as preference learning was well suited to learn numerical scores by comparing the examples.

Future work has three possible branches, research in symbolic systems, research in XAI and research combining both. For symbolic systems, there is still room for newer and better algorithms that are more efficient and noise-tolerant. In XAI, metrics are still pretty much an open problem, while the need of better explaining methods have risen. Finally, in symbolic XAI, further tests are required with real world problems in order to check to which problems are this systems aplicable.

Bibliography

[1] Alejandro Barredo Arrieta, Natalia Diaz Rodriguez, Javier Del Ser, Adrien Bennetot, Siham Tabik, Alberto Barbado González, Salvador García, Sergio Gil-López, Daniel Molina, V. Richard Benjamins, Raja Chatila, and Francisco Herrera. Explainable artificial intelligence (xai): Concepts, taxonomies, opportunities and challenges toward responsible ai. *Information Fusion*, 58:82–115, 2020.

[2] Donald Michie. Machine learning in the next five years. 1988.

[3] Andrew Cropper, Rolf Morel, and Stephen Muggleton. Learning higher-order logic programs. *Machine Learning*, 109:1–34, 07 2020.

[4] Mark Law. *Inductive Learning of Answer Set Programs (thesis)*. PhD thesis, 09 2018.

[5] Richard Evans and Edward Grefenstette. Learning explanatory rules from noisy data. *Journal of Artificial Intelligence Research*, 61, 11 2017.

[6] Robin Manhaeve, Sebastijan Dumancic, Angelika Kimmig, Thomas Demeester, and Luc De Raedt. Deepproblog: Neural probabilistic logic programming, 07 2019.

[7] T. Ribeiro, M. Folschette, L. Trilling, N. Glade, K. Inoue, M. Magnin, and O. Roux. Les enjeux de l'inférence de modèles dynamiques des systèmes biologiques à partir de séries temporelles. In C. Lhoussaine and E. Remy, editors, *Approches symboliques de la modélisation et de l'analyse des systèmes biologiques*. ISTE Editions, 2020. In edition.

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

[9] Stephen Muggleton. Inverse entailment and progol, 1995.

[10] Gerhard Brewka, Thomas Eiter, and Mirosław Truszczyński. Answer set programming at a glance. *Commun. ACM*, 54(12):92–103, December 2011.

[11] Mark Law, Alessandra Russo, and K. Broda. The ilasp system for inductive learning of answer set programs. 2020.

[12] Martin Kato. Igor ii - an analytical inductive functional programming system. pages 29–32, 01 2010.

[13] Emanuel Kitzelmann. Data-driven induction of functional programs. pages 781–782, 01 2008.

[14] Susumu Katayama. An analytical inductive functional programming system that avoids unintended programs. pages 43–52, 01 2012.

[15] Susumu Katayama. Systematic search for lambda expressions. In *In Proceedings Sixth Symposium on Trends in Functional Programming (TFP2005)*, 2005.

[16] K. Inoue, T. Ribeiro, and C. Sakama. Learning from interpretation transition. *Machine Learning*, 94(1):51–79, 2014.

[17] Alfonso Ortega, Julian Fiérrez, Aythami Morales, Zilong Wang, and Tony Ribeiro. Symbolic AI for XAI: evaluating LFIT inductive programming for fair and explainable automatic recruitment. In *IEEE Winter Conference on Applications of Computer Vision Workshops, WACV Workshops 2021, Waikola, HI, USA, January 5-9, 2021*, pages 78–87. IEEE, 2021.

[18] A. Eiben and Jim Smith. *Introduction To Evolutionary Computing*, volume 45. 01 2003.

[19] Michael O'Neill and Conor Ryan. *Grammatical evolution - evolutionary automatic programming in an arbitrary language*, volume 4 of *Genetic programming*. Kluwer, 2003.

[20] Marina de la Cruz, Alfonso Ortega de la Puente, and Manuel Alfonseca. Attribute grammar evolution. In José Mira and José R. Álvarez, editors, *Artificial Intelligence and Knowledge Engineering Applications: A Bioinspired Approach: First International Work-Conference on the Interplay Between Natural and Artificial Computation, IWINAC 2005, Las Palmas, Canary Islands, Spain, June 15-18, 2005, Proceedings, Part II*, volume 3562 of *Lecture Notes in Computer Science*, pages 182–191. Springer, 2005.

[21] Alfonso Ortega, Marina de la Cruz, and Manuel Alfonseca. Christiansen grammar evolution: Grammatical evolution with semantics. *IEEE Trans. Evol. Comput.*, 11(1):77–90, 2007.

[22] César Luis Alonso, José Luis Montaña, Jorge Puente, and Cruz E. Borges. A new linear genetic programming approach based on straight line programs: Some theoretical and experimental aspects. *Int. J. Artif. Intell. Tools*, 18(5):757–781, 2009.

[23] James Woodward and Lauren Ross. Scientific Explanation. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, 2021.

[24] Leilani Gilpin, David Bau, Ben Yuan, Ayesha Bajwa, Michael Specter, and Lalana Kagal. Explaining explanations: An overview of interpretability of machine learning. pages 80–89, 2018.

[25] Riccardo Guidotti, A. Monreale, F. Turini, D. Pedreschi, and F. Giannotti. A survey of methods for explaining black box models. *ACM Computing Surveys (CSUR)*, 51:1 – 42, 2019.

[26] Derek Doran, Sarah Schulz, and Tarek Besold. What does explainable ai really mean? a new conceptualization of perspectives, 10 2017.

[27] Menaka Narayanan, Emily Chen, Jeffrey He, Been Kim, Sam Gershman, and Finale Doshi velez. How do humans understand explanations from machine learning systems? an evaluation of the human-interpretability of explanation, 02 2018.

[28] Hao Yuan, Haiyang Yu, Shurui Gui, and Shuiwang Ji. Explainability in graph neural networks: A taxonomic survey. 2020.

[29] Tameru Hailesilassie. Rule extraction algorithm for deep neural networks: A review. *ArXiv*, abs/1610.05267, 2016.

[30] Jan Ruben Zilke. Extracting rules from deep neural networks, 2016.

[31] Ivan Donadello and L. Serafini. Integration of numeric and symbolic information for semantic image interpretation. *Intelligenza Artificiale*, 10:33–47, 2016.

[32] Ivan Donadello and M. Dragoni. Sexai: Introducing concepts into black boxes for explainable artificial intelligence. In *XAI.it@AI*IA*, 2020.

[33] Leilani H. Gilpin, David Bau, Ben Z. Yuan, Ayesha Bajwa, Michael A. Specter, and Lalana Kagal. Explaining explanations: An overview of interpretability of machine learning. *2018 IEEE 5th International Conference on Data Science and Advanced Analytics (DSAA)*, pages 80–89, 2018.

[34] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "why should i trust you?": Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, page 1135–1144, New York, NY, USA, 2016. Association for Computing Machinery.

[35] Cristian Bucilua, Rich Caruana, and Alexandru Niculescu-Mizil. Model compression. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '06, page 535–541, New York, NY, USA, 2006. Association for Computing Machinery.

[36] Scott M. Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In *NIPS*, 2017.

[37] A. Henelius, K. Puolamäki, and Antti Ukkonen. Interpreting classifiers through attribute interactions in datasets. *ArXiv*, abs/1707.07576, 2017.

[38] Paulo Cortez and Mark J. Embrechts. Using sensitivity analysis and visualization techniques to open black box data mining models. *Information Sciences*, 225:1–17, 2013.

[39] Alex Goldstein, A. Kapelner, J. Bleich, and E. Pitkin. Peeking inside the black box: Visualizing statistical learning with plots of individual conditional expectation. *Journal of Computational and Graphical Statistics*, 24:44 – 65, 2013.

[40] Satoshi Hara and K. Hayashi. Making tree ensembles interpretable: A bayesian model selection approach. In *AISTATS*, 2018.

[41] L. Breiman, J. Friedman, R. Olshen, and C. J. Stone. Classification and regression trees. 1983.

[42] Xiuju Fu, ChongJin Ong, S. Keerthi, Gih Guang Hung, and Liping Goh. Extracting the knowledge embedded in support vector machines. In *2004 IEEE International Joint Conference on Neural Networks (IEEE Cat. No.04CH37541)*, volume 1, pages 291–296, 2004.

[43] Zhenyu Chen, Jianping Li, and Liwei Wei. A multiple kernel support vector machine scheme for feature selection and rule extraction from gene expression data of cancer tissue. *Artificial Intelligence in Medicine*, 41(2):161–175, 2007. Integrative data mining in systems biology.

[44] B. Üstün, W.J. Melssen, and L.M.C. Buydens. Visualisation and interpretation of support vector regression models. *Analytica Chimica Acta*, 595(1):299–309, 2007. Papers presented at the 10th International Conference on Chemometrics in Analytical Chemistry.

[45] Lars Rosenbaum, Georg Hinselmann, and Andreas Zell. Interpreting linear support vector machine models with heat map molecule coloring. *Journal of cheminformatics*, 3:11, 03 2011.

[46] Interpreting support vector machine models for multivariate group wise analysis in neuroimaging. *Medical Image Analysis*, 24(1):190–204, 2015.

[47] Avanti Shrikumar, P. Greenside, A. Shcherbina, and A. Kundaje. Not just a black box: Learning important features through propagating activation differences. *ArXiv*, abs/1605.01713, 2016.

[48] Matthew D. Zeiler, Graham W. Taylor, and R. Fergus. Adaptive deconvolutional networks for mid and high level feature learning. *2011 International Conference on Computer Vision*, pages 2018–2025, 2011.

[49] Anh M Nguyen, A. Dosovitskiy, J. Yosinski, T. Brox, and J. Clune. Synthesizing the preferred inputs for neurons in neural networks via deep generator networks. In *NIPS*, 2016.

[50] L. Arras, Grégoire Montavon, K. Müller, and W. Samek. Explaining recurrent neural network predictions in sentiment analysis. In *WASSA@EMNLP*, 2017.

[51] A. Karpathy, J. Johnson, and Li Fei-Fei. Visualizing and understanding recurrent networks. *ArXiv*, abs/1506.02078, 2015.

[52] Edward Choi, M. T. Bahadori, Jimeng Sun, Joshua A Kulas, A. Schuetz, and W. Stewart. Retain: An interpretable predictive model for healthcare using reverse time attention mechanism. In *NIPS*, 2016.

[53] F. Baldassarre and Hossein Azizpour. Explainability techniques for graph convolutional networks. *ArXiv*, abs/1905.13686, 2019.

[54] Phillip E. Pope, S. Kolouri, Mohammad Rostami, C. E. Martin, and Heiko Hoffmann. Explainability methods for graph convolutional neural networks. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10764–10773, 2019.

[55] Rex Ying, Dylan Bourgeois, Jiaxuan You, M. Zitnik, and J. Leskovec. Gnnexplainer: Generating explanations for graph neural networks. *Advances in neural information processing systems*, 32:9240–9251, 2019.

[56] Dongsheng Luo, W. Cheng, Dongkuan Xu, Wenchao Yu, Bo Zong, H. Chen, and Xiang Zhang. Parameterized explainer for graph neural network. *ArXiv*, abs/2011.04573, 2020.

[57] M. Schlichtkrull, Nicola De Cao, and Ivan Titov. Interpreting graph neural networks for nlp with differentiable edge masking. *ArXiv*, abs/2010.00577, 2020.

[58] Hao Yuan, H. Yu, Jie Wang, Kang Li, and Shuiwang Ji. On explainability of graph neural networks via subgraph explorations. *ArXiv*, abs/2102.05152, 2021.

[59] Q. Huang, M. Yamada, Yuan Tian, Dinesh Singh, Dawei Yin, and Yi Chang. Graphlime: Local interpretable model explanations for graph neural networks. *ArXiv*, abs/2001.06216, 2020.

[60] Yue Zhang, David DeFazio, and Arti Ramesh. Relex: A model-agnostic relational model explainer. *ArXiv*, abs/2006.00305, 2020.

[61] Minh N. Vu and M. Thai. Pgm-explainer: Probabilistic graphical model explanations for graph neural networks. *ArXiv*, abs/2010.05788, 2020.

[62] Thomas Schnake, Oliver Eberle, Jonas Lederer, Shinichi Nakajima, K. T. Schutt, K. Muller, and Grégoire Montavon. Higher-order explanations of graph neural networks via relevant walks. *arXiv: Learning*, 2020.

[63] Hao Yuan, Jiliang Tang, Xia Hu, and Shuiwang Ji. Xgnn: Towards model-level explanations of graph neural networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '20, page 430–438, New York, NY, USA, 2020. Association for Computing Machinery.

[64] B. Zhou, A. Khosla, À. Lapedriza, A. Oliva, and A. Torralba. Learning deep features for discriminative localization. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2921–2929, 2016.

[65] J. Yosinski, J. Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? *ArXiv*, abs/1411.1792, 2014.

[66] Q. Zhang and Song-Chun Zhu. Visual interpretability for deep learning: a survey. *Frontiers of Information Technology & Electronic Engineering*, 19:27–39, 2018.

[67] Been Kim, Justin Gilmer, Fernanda Viegas, Ulfar Erlingsson, and Martin Wattenberg. Tcav: Relative concept importance testing with linear concept activation vectors, 11 2017.

[68] Andrew Slavin Ross, Michael C. Hughes, and Finale Doshi-Velez. Right for the right reasons: Training differentiable models by constraining their explanations. In *IJCAI*, 2017.

[69] Diederik P. Kingma and M. Welling. Auto-encoding variational bayes. *CoRR*, abs/1312.6114, 2014.

[70] Aishwarya Agrawal, Jiasen Lu, Stanislaw Antol, Margaret Mitchell, C. L. Zitnick, Devi Parikh, and Dhruv Batra. Vqa: Visual question answering. *International Journal of Computer Vision*, 123:4–31, 2015.

[71] Yin Lou, R. Caruana, J. Gehrke, and G. Hooker. Accurate intelligible models with pairwise interactions. *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2013.

[72] S. Krishnan and Eugene Wu. Palm: Machine learning explanations for iterative debugging. *Proceedings of the 2nd Workshop on Human-In-the-Loop Data Analytics*, 2017.

[73] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron C. Courville, R. Salakhutdinov, R. Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *ICML*, 2015.

[74] A. Datta, Shayak Sen, and Yair Zick. Algorithmic transparency via quantitative input influence: Theory and experiments with learning systems. *2016 IEEE Symposium on Security and Privacy (SP)*, pages 598–617, 2016.

[75] M. Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic attribution for deep networks. *ArXiv*, abs/1703.01365, 2017.

[76] J. Krause, Adam Perer, and Kenney Ng. Interacting with predictions: Visual inspection of black-box machine learning models. *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, 2016.

[77] Xiaoxin Yin and Jiawei Han. Cpar: Classification based on predictive association rules. In *SDM*, 2003.

[78] J. R. Quinlan. Generating production rules from decision trees. In *IJCAI*, 1987.

[79] J. Bien and R. Tibshirani. Prototype selection for interpretable classification. *The Annals of Applied Statistics*, 5:2403–2424, 2011.

[80] Been Kim, C. Rudin, and J. Shah. The bayesian case model: A generative approach for case-based reasoning and prototype classification. In *NIPS*, 2014.

[81] Andrew Cropper, Sebastijan Dumancic, and Stephen Muggleton. Turning 30: New ideas in inductive logic programming. 04 2020.

[82] Céline Hocquette and Stephen Muggleton. Can meta-interpretive learning outperform deep reinforcement learning of evaluable game strategies?, 02 2019.

[83] Andrew Cropper and Stephen H. Muggleton. Metagol system. https://github.com/metagol/metagol, 2016.

[84] Andrew Cropper and Rolf Morel. Popper system. https://github.com/rolfmorel/popper, 2021.

[85] Stephen Muggleton, Wang-Zhou Dai, Claude Sammut, Alireza Tamaddoni-Nezhad, Jing Wen, and Zhi-Hua Zhou. Meta-interpretive learning from noisy images. *Machine Learning*, 107:1–22, 07 2018.