Statistical relational learning for game theory

Marco Lippi

***Abstract*—In this paper we motivate the use of models and algorithms from the area of Statistical Relational Learning (SRL) as a framework for the description and the analysis of games. SRL combines the powerful formalism of first-order logic with the capability of probabilistic graphical models in handling uncertainty in data and representing dependencies between random variables: for this reason, SRL models can be effectively used to represent several categories of games, including games with partial information, graphical games and stochastic games. Inference algorithms can be used to approach the opponent modeling problem, as well as to find Nash equilibria or Pareto optimal solutions. Structure learning algorithms can be applied, in order to automatically extract probabilistic logic clauses describing the strategies of an opponent with a high-level, human-interpretable formalism. Experiments conducted using Markov logic networks, one of the most used SRL frameworks, show the potential of the approach.**

I. Introduction

The intersection between the research areas of artificial intelligence, machine learning and game theory has recently known a phase of development. Historically, artificial intelligence has always seen games as a source of inspiration and challenges for the design of intelligent systems capable of interacting with the environment and of taking rational autonomous decisions. Within this context, the application of algorithms and models from computer science, and artificial intelligence in particular, to game-theoretic problems, has produced the rapid growth of the research area called Algorithmic Game Theory [1], with specific interest in game playing and decision theory. Some works have instead pointed out that the link between game theory and machine learning could be stronger, and that a more prolific collaboration between the two research fields has only been established in the last decade [2].

Nevertheless, without doubt, there are several research directions where the contribution of machine learning to game theory problems has been much significant. For example, with the development of graphical games [3], where game models are derived from combinatorial graphs, machine learning methodologies have been successfully applied to build efficient algorithms for the computation of Nash equilibria in large multi-player games. In this direction, [4] introduces a mapping between graphical games and Markov random fields, describing how inference algorithms such as belief propagation or Markov Chain Monte Carlo can be applied to the decision problem of whether a graphical game has a pure Nash equilibrium. Another case in which machine learning has found application within game theory is given by opponent modeling, that is the problem of predicting the moves of the adversaries, given observations of their behavior in past situations. Among the many techniques which have been employed throughout the years to address this task we can cite reinforcement learning [5], [6], Q-learning [7], artificial neural networks [8], and many others. Iterated games, such as the Iterated Prisoner's Dilemma (IPD), represent the ideal scenario to test this kind of approaches (e.g., see [9] and references therein). In [2] a wider comparison between inference algorithms for game theory and machine learning is given, and a connection between fictitious play [10] and variational learning is presented, producing an interesting efficient solution for the opponent modeling problem. Several classical applications of learning and inference algorithms in game-theoretical frameworks can be found in [11]. Learning from game logs has become an extremely popular research field, especially in the context of perfect information games: deep convolutional neural networks have been applied to the game of Go [12], while for Othello approaches based on N-tuple systems and self-learning [13], [14] and evolution functions and preference learning [15] have been proposed. Deep learning techniques have also been applied to learn game strategies from visual input configurations: this is the case, for example, of the reinforcement learning approach for playing Atari games [16], while recently deep convolutional neural networks have also been applied within this context to learn from visual patterns in the game of Go [17]. Another recent, prominent line of research is given by the use of Monte-Carlo tree search to predict the evolution of game configurations (e.g., see [18], [19] and references therein).

It is clear from these works that there are several aspects where machine learning and, more in general, artificial intelligence, can give a significant contribution to game theory. First of all, the problem of knowledge representation is crucial in games, where it is very often necessary to describe the domain of interest in terms of strategies, alliances, rules, relationships and dependencies among players, information concerning the external environment: this is a context where logic formalisms can play a crucial role in representation issues, whereas techniques from inductive logic programming [20] can be applied in order to extract rules from a knowledge base represented by logic predicates. Moreover, probability also plays a very important role in game theory, since in many games players have to deal with missing or incomplete information, and probabilistic reasoning is a key ingredient for decision making.

For these reasons, a recently developed research area which perfectly fits such aspects is given by Statistical Relational Learning (SRL), sometimes also called Probablistic Inductive Logic Programming [21]. SRL methods combine logic representations with the formalism of probabilistic graphical models and with methodologies coming from statistical learning. This framework allows to represent background knowledge in a straightforward way, thanks to the expressive power of first-order logic, while maintaining the capability of managing uncertainty in data within graphical models such as Markov random fields or Bayesian networks. Another feature of SRL models which might have a great impact on game theory is that of structure learning, that is the possibility of directly learning model structures from data.

In [22], a first attempt to employ SRL methodologies for game theory is given, by using Causal Probabilistic Time Logic (CPT-L) for the modeling of sequential game structures with fully observed data. The proposed approach has been tested in several domains, including a multiplayer online strategy game, with the goal of predicting player actions or identifying groups of cooperating alliances. CPT-L is an extension of traditional CP-Logic which represents causality among probability distribution over sequences of relational state descriptions. The restriction to sequential models and the hypothesis of dealing with fully observable data limits the representation power of the framework, but on the other hand it allows to employ efficient learning and inference algorithms.

Other SRL models which have recently found applications within decision making scenarios, reinforcement learning and opponent mod-

Marco Lippi is with Dipartimento di Informatica – Scienza e Ingegneria, Università degli Studi di Bologna, Italy (email: {marco.lippi3@unibo.it}).

eling include: Logical Markov Decision Programs (LOMDPs) [23], which combine Markov decision processes with logic programs; DTProbLog [24] which extends the Probabilistic Prolog language towards decision theory; Infinite Hidden Relational Trust Model (IHRTM) [25] which extend Infinite Relational Models towards trust learning; Relational Reinforcement Learning [26],[27], which has been applied to planning and policy learning; Independent Choice Logic [28] (ICL) to model strategic situations. The contribution of this paper, inspired by these recent works, is to show the potential of the SRL framework in the context of game theory, by clarifying the connections between the two research fields, and by highlighting how SRL can provide an extremely powerful formalism for modeling several different game categories, and for addressing a number of challenging problems.

In the next section we briefly review some notation of game theory and we describe some of the most commonly used categories of games, with the aim of presenting some suitable scenarios for the application of SRL methods. Then, we introduce more formally Statistical Relational Learning, with a particular focus on Markov Logic Networks, which will be used in our experiments. The final section will present some experimental results, showing how SRL models can be applied to classical game theory problems.

II. Game theory

Game theory is a field of mathematics whose aim is to analyze *strategic situations*, in which individuals interact during decision-making processes. In strategic situations, the success of an individual does not depend only on his/her choices, but also on the choices of other individuals, with whom there can be direct or indirect interaction. Game theory applications range in a number of different areas, including economics, social sciences, evolutionary biology, politics, computer science and many others [29].

A classic strategic game is defined as a model in which all the decision-makers (*players*) plan their decisions simultaneously, with no possibility of a change. Formally, a strategic game can be defined, using the normal form, as a triple $G = \langle \mathcal{P}, \mathcal{S}, \mathcal{U} \rangle$ where:

- $\mathcal{P} = \{p_1, \dots, p_N\}$ is the set of players
- $\mathcal{S} = \{S_1, \dots, S_N\}$ is a set of tuples S_i describing the possible actions (*strategies*) of player p_i
- $\mathcal{U} = \{u_1, \dots, u_N\}$ is the set of utility functions associated to each player, $u_i : S_1 \times \dots \times S_N \to \Re$.

The aim of each player is to maximize his/her utility, or *payoff*. A typical assumption of game theory is that players play rationally, that is they would not choose a strategy which would give them a lower payoff than another, other things being equal. This is a natural assumption when speaking in terms of *mathematical game theory*, that is in cases where the utility function of each player is easily formalized, and it is clear which of two outcomes is to be preferable for a player.

In a game with perfect information, each player knows all the relevant information to make a decision (i.e., chess, backgammon, etc.), clearly including the utilities of all the other players. Normal-form games are often represented in a matrix-form: in the following example, the well known prisoner's dilemma game is described using the matrix formalism.

Example 1 *The Prisoner's Dilemma (PD)*
Two suspects are arrested by the police. Since there is not enough evidence for a conviction, the two prisoners are separated and they are offered the same deal: if one testifies against the other (defects) and the other remains silent (cooperates), the betrayer goes free, while the silent accomplice receives a 10-year sentence. If they both betray, they receive a 5-year sentence each. Finally, if they both stay silent, they can only be sentenced for one year, due to a minor charge. Each prisoner must choose either to betray or to stay silent, independently from the other, and they cannot change their position. Which strategy is the most convenient for them ?

	C	D
C	(-1,-1)	(-10,0)
D	(0,-10)	(-5,-5)

Table I: The matrix payoff of the prisoner's dilemma game. Greater numbers are preferred to smaller ones.

Table I shows the matrix payoff of the game: rows correspond to strategies chosen by player one, while columns correspond to strategies chosen by player two. The numbers in parentheses indicate the payoffs of the first and second player, respectively. Positive numbers are supposed to be preferred to negative ones.

In order to answer the question in Example 1, it is necessary to introduce the concepts of *strategy profile* and *best reply*. A strategy profile for player k is a probability distribution $p = \{p_1, \dots, p_{S_k}\}$ over the set of strategies which can be chosen by player k. If a profile has only one non-zero element (that is therefore equal to 1) then the strategy is called *pure*, otherwise it is called *mixed*, meaning that different actions might be chosen by that player in different moments. If we now consider a two-player game, being A and B the two payoff matrices associated to each player, a profile p for the first player is said to be a best reply for a profile q of the second player if the following condition holds:

$$pAq \geq p'Aq, \ \ \forall p' \neq p.$$

The best reply definition implies that a player cannot obtain a better outcome by changing his/her strategy, other things being equal. A set of profiles is said to be a *Nash equilibrium* [30] if, for each player, the chosen profile is a best response with respect to all the profiles chosen by the other players. By employing the aforementioned two-player formalism, a pair (p^*, q^*) is a Nash equilibrium if p^* is a best response with respect to q^* and q^* is a best response with respect to p^*. If the profiles are all pure, we speak of a *pure Nash equilibrium*, otherwise we speak of a *mixed Nash equilibrium*. All these concepts can be naturally extended to multi-player games in a straightfoward way. In case of pure strategies, in a Nash equilibrum a strategy s_i is said to be a *dominating* strategy, which means that it gives at least the same payoff of any other $s_j \neq s_i$, for any possible combination of strategies of the other players: no unilateral deviation in strategy by any single player is profitable for that player. For any game with a finite number of players and finite sets of strategies for all players, it can be proven that at least one (mixed) Nash equilibrium exists [30].

In the case of the Prisoner's Dilemma, it is easy to prove that the Nash equilibrium consists in the situation where the two players defect, and it is a pure equilibrium. Yet, the two players would obtain a higher payoff in the case of cooperation: for this reason, such an equilibrium is said to be not *Pareto-optimal*. A set of strategy profiles is said to be Pareto-optimal if no player can increase his/her payoff without decreasing that of another player. In a two-player game, a pair of profiles (p^*, q^*) is a Pareto optimum if there does not exist another profile pair (p, q) such that one of the two following conditions holds:

$$pAq > p^*Aq^*, \quad pBq \geq p^*Bq^*$$
$$pAq \geq p^*Aq, \quad pBq > p^*Bq^*.$$

In general, games can have one or more equilibria. The following example, describing the well known *Battle of the Sexes* game, will describe a scenario where multiple equilibria exist.

Example 2 *Battle of the Sexes*[1]
A married couple has to decide where to spend the evening. The husband would like to go to a Soccer game (in another version, to a Stravinsky concert). The wife, on the other hand, would prefer a Ballet (in another version, a Bach concert). Both prefer going to the same place rather than to different ones. Where should they go ? Table II shows the matrix payoff of the game.

	B	S
B	(1,2)	(0,0)
S	(0,0)	(2,1)

Table II: The matrix payoff of the Battle of the Sexes game. Greater numbers should be preferred to smaller ones.

In this case, two different pure Nash equilibria exist, corresponding to the two situations `(S,S)` and `(B,B)`: if the husband chooses `S`, then the best choice for his wife is `S` too. The other way round, if the wife chooses `B`, then her husband should choose `B` too. It can be proven that a mixed Nash equilibrium also exists, where each player plays the preferred strategy more frequently: given the matrix payoff in II, each player would play the preferred strategy with probability $2/3$ and the other with probability $1/3$ [29].

Throughout the years, a wide variety of game categories has been introduced, in order to model different scenarios and real-world situations in disparate fields, including biology, economics, political sciences, sociology, psychology and computer science.

A category of games which has recently gained an increasing attention is that of *graphical games*, or *graph games* [3]. In a graphical game, players are considered as nodes in a graph, where edges represent their interactions. In such a game, the payoff of a player does not depend on the strategies of all the other players, but only on the subset of his/her *neighbors*, according to the graphical structure describing the game. Hence, there are several *local* payoff matrices, one for each player, rather than a unique global payoff matrix. By exploiting the representation power of graphical models like Markov networks, SRL models are a suitable framework for this kind of scenario.

Stochastic games [31] are dynamic games which are played in stages by one ore more players: subsequent actions change the state of the game, as well as the outcome of the players, and are chosen according to transition probabilities. If there is a finite number of players and the action sets and the set of states are finite, then a stochastic game with a finite number of stages always has a Nash equilibrium. Stochastic games are sometimes called also Markov games, as they share analogies with Markov Decisions Processes (MDPs) [32], [33], where actions (strategies) induce transitions between states of the model and all players act simultaneously.

Uncertainty can be introduced in games in several ways: in games with *incomplete information*, players do not know the payoff matrices of the other players, but they know the actions each other player takes, while in games with *imperfect information* the opposite happens, the actions and strategies of the opposing players being unknown for each player. Actually, a game with incomplete information can always be transformed into a game with complete but imperfect information by applying the so-called Harsanyi transformation [34], which models Nature randomness as another player whose choices are not yet known (basically, Nature player acts as a random number generator).

Statistical relational learning can easily handle uncertainty in data and model scenarios with incomplete or unknown information. One of the classic problems in game theory, when dealing with incomplete or imperfect information, is to model opponents behavior, in order to predict their actions and consequently plan adapted strategies. This is a fundamental task also in repeated games and stochastic games, and it has particular significance in domains where the opponent plays *sub-optimally*, that is not following strategies that correspond to strategic equilibria. In that case, taking advantage of correctly guessing the behavior of the opponent, exploiting weaknesses in his/her strategy, and taking decisions according to such knowledge will most likely bring to larger rewards than those that would be obtained by simply playing according to an optimal equilibrium strategy. In complex games such as Poker or Risk, the behavior of players is clearly sub-optimal, and in those cases opponent modeling plays a crucial role in constructing successful game strategies. As it will be shown in the experimental section, SRL models can be used in order to describe probabilistic rules which can capture the strategy of one or more players, thus providing an extremely powerful instrument for opponent modeling.

[1]Also known as Ballet or Soccer, Bach or Stravinsky (BoS).

III. Statistical Relational Learning

Since the end of the nineties, machine learning has run across what has been called a *relational revolution*: one of the main assumptions of traditional machine learning algorithms, like neural networks, support vector machines or decision trees, was to consider examples as independent and identically distributed, while on the contrary in many real-world problems there exist several relations and inter-dependencies among the objects of the domain. Taking into account these inter-dependencies within the statistical learning framework and addressing problems in complex relational domains are the key ingredients of Statistical Relational Learning (SRL) [35]. The SRL community was born at the intersection of different areas of artificial intelligence, logic and statistics, with the aim of combining the descriptive formalism of first-order logic with the representative power of graphical models, like Bayesian networks or Markov networks.

Given a data set represented in the form of a complex structure (e.g., a graph), SRL allows to approach several tasks exploiting the relationships and dependencies in the data, and performing collective classification: examples of such tasks are link prediction (i.e., predicting whether two nodes in a graph are connected), object classification (i.e., predicting whether a node has some property) or group detection (i.e., identfying underlying substructures in a graph). Among the many domains where SRL has been succesfully applied it is worth citing social networks, bioinformatics, chemoinformatics, natural language processing, the semantic web.

While first-order logic offers an extremely powerful language to describe a certain domain in terms of quantified logic formulae [36], graphical models conveniently represent dependencies between random variables, and naturally handle uncertainty in data [37]. These two separate frameworks have been combined in a wide variety of different formalisms. Historically, the first formalism which has been introduced to extend inductive logic programming towards handling probability theory has been that of stochastic logic programs (SLPs). SLPs have been introduced in [38], and in first instance they can be seen as a generalization of Hidden Markov Models and probabilistic context-free grammars [39]. Approaches using the formalism of Bayesian networks include Relational Bayesian networks (RBNs) [40], Bayesian Logic Programs (BLPs) [41], PRISM [42], Independent Choice Logic (ICL) [28] and Probabilistic Relational Models (PRMs) [43]. RBNs have been developed to represent probabilistic relations on multiple random events, by modeling a probability distribution over such relations using a Bayesian network; BLPs are generalizations of Bayesian networks and logic programs, with an expressive power very similar to SLPs; PRMs extend Bayesian networks with the concept of objects, with properties and relations between them, by specifying a template for a probability distribution over a relational database, by means of a description of the relational schema of the domain, and the probabilistic dependencies which exist among its objects. Type extension trees [44], [45] (TETs) were introduced as a representation formalism for complex combinatorial features in relational domains. ProbLog [46] is an extension of Prolog programming language [47] to handle uncertainty in data, which was initially conceived to efficiently answer probabilistic queries. kLog [48] is a recently developed language, designed to perform kernel-based learning on expressive logical and relational representations. The framework of Learning from Constraints (LfC) [49], [50] extends the classical framework of kernel machines to incorporate knowledge in the form of logic rules, encoded in a sort of semantic regularization term.

This multitude of similar systems and models is sometimes referred to by the term *alphabet-soup* [51]. On the one hand it indicates the great interest which this field is gaining, but at the same time it represents also a weakness of the area, since analogies and differences between some of these formalisms are often unclear.

As stated in the introduction, a few of the existing SRL systems have also been used in the context of decision theory. For example, ProbLog has been extended in this direction, with the development of DTProbLog (Decision-Theoretic ProbLog) [24], that models strategic situations with weighted formulae and allows to infer the utility of a set of strategies in a probabilistic setting. A similar idea was sketched in [52] for SLPs, while an extension of MLNs towards decision theory was outlined in [53], but in those two cases the research line was not further developed. Independent Choice Logic (ICL) [28] has been proposed to combine logic programming and probability, by modeling a set of independent strategies, named *choices*, with a probability distribution over each choice, and a logic program specifying the consequences of such choices. The logic formalism of ICL is restricted to Horn clauses, whereas uncertainty is modeled with the formalism of Bayesian networks. A further restriction requires the theory to be *acyclic* (no recursive clauses). ICL provides a suitable framework for representing game-theoretic scenarios, and in fact it has been used to model strategic situations and problems, including Nash equilibria [28]. Yet, the applicability of this framework to real-world, large-scale domains still remains to be tested [54].

The aim of this paper is to highlight the opportunities of an SRL-based approach to game theory, and actually many of the existing SRL models proposed in recent years might be employed for this task. Yet, we believe that choosing one particular model for a more accurate description of the approach might greatly improve the clarity and the readability of the paper: we resulted in choosing the framework of Markov Logic Networks (MLNs) [55], which combine first-order logic with Markov networks, providing a formalism with which it is possible to describe a domain in terms of logic predicates and weighted formulae. In the next section an overview of this model will be given, as well as some considerations which will motivate this choice.

IV. Markov Logic Networks

Markov logic networks [55] (MLNs) combine first-order logic with Markov networks, providing a formalism with which it is possible to describe a domain in terms of logic predicates and weighted formulae.

In general, a first-order logic knowledge base can be seen as a set of *hard* constraints over possible worlds:[2] if a world violates even only one formula, then it has zero probability. On the other hand, in Markov logic violations of formulae are allowed: a world violating a formula will be *less probable*, but not impossible. A Markov logic network (MLN) consists in a set of first-order logic formulae $\mathcal{F} = \{F_1, \dots, F_n\}$, and a set of real-valued weights $w = \{w_1, \dots, w_n\}$, where weight w_j is associated to formula F_j. Together with a *finite* set of constants $C = \{c_1, \dots c_k\}$ (corresponding to the objects of the domain), an MLN defines a Markov network where the set of nodes corresponds to all possible ground atoms,[3] and there is an edge between two nodes if and only if the corresponding ground atoms appear together in at least one grounding of some formula F_j. An MLN defines a probability distribution over possible worlds:

$$P(X = x) = \frac{1}{Z} \exp\left(\sum_{j}^{|\mathcal{F}|} w_j n_j(x)\right) \tag{1}$$

where $n_j(x)$ is the number of true groundings of formula F_j in x. The magnitude of a weight indicates how "strong" the corresponding rule is: the higher the weight, the lower the probability of a world violating that formula. Hard constraints can be modeled with rules having an infinite weight.

An MLN can therefore be defined by i) a set of (possibly typed) constants, as the following one:

```
people = {Alice,Bob,Charles,Daniel}
  movies = {Movie1,Movie2,Movie3}
```

and ii) a set of weighted formulae, such as:

```
2.3 Friends(x,y) <=> Friends(y,x)
1.5 Likes(x,m) ∧ Friends(x,y) => Likes(y,m)
```

where we adopt the notation of using lowercase strings to indicate variables within forumlae (e.g., `x`, `m`), and strings starting with an uppercase letter for constants (e.g., `Alice`). The first formula models a sort of transitivity of the relation of friendship between three people x, y and z, while the second describes a sort of movie-suggestion rule: if one person x likes a certain movie m, and if x and y are friends, then probably also y will like movie m. In general, such rules are clearly not always true for all the possible values that each variable can take. A knowledge base within this context will also contain a collection of ground predicates (which are known to be true) like the following ones:

```
Friends(Alice,Charles)
Friends(Bob,Charles)
Friends(Charles,Daniel)
Likes(Alice,Movie2)
Likes(Bob,Movie1)
Likes(Charles,Movie1)
```

Although we want to stress the point that Markov Logic is certainly not the only SRL framework which can be applied to game theory modeling, we hereby motivate our choice of adopting it, based on the following considerations:

1) it represents an extremely general formalism, which allows to model many different game categories and scenarios;
2) it can be applied both to tasks which require to learn the weights of the probabilistic logic formulae (weight learning), and to problems where the formulae themselves have to be discovered directly from the data (structure learning);
3) it allows to employ a discriminative learning setting, where some facts are observed (evidence) while some others are to be predicted, which is a very common framework in game theory;
4) it has been succesfully applied to many different application domains, ranging from natural language parsing to bioinformatics, from computer vision to traffic forecasting [56], [57], [58], [59], [60];

[2]A world, or Herbrand interpretation, is a truth value assignment for all the predicates in our domain.

[3]A ground atom is an atom where all variables have been substituted by constants.

5) it offers a publicly available software, Alchemy,[4] which has become quite popular in recent years in the machine learning community.

Three main problem categories can be approached within the Markov Logic framework, which consist in inference, parameter learning and structure learning. The following subsections will give an overview of the existing algorithms for each of these problems.

A. Inference

When using a graphical model to represent a domain, some of the random variables are usually given as *evidence*, in the sense that their configuration is always observable, both during training and at prediction time, while the remaining are usually called *query* variables, being the ones upon which prediction must be performed. The problem of *inference* within such a setting consists in retrieving either the conditional probability of query variables, given the evidence, or directly the maximum a posteriori (MAP) configuration, that is the most probable state of the model.

Exact inference in general can be proven to be a $\#P$-complete problem [61], and it is such in the case of MLNs. Using approximate algorithms is therefore necessary to deal with such a complex task. Finding the most probable world corresponds, in Markov logic, to find the truth assignment maximizing the sum of weights of satisfied clauses. Any (weighted) satisfiability solver can be employed for this task: MaxWalkSAT [62], a weighted variant of WalkSAT local-search satisfiability solver, is one of the commonly used methods. MaxWalkSAT is a stochastic algorithm which, at each iteration, picks an unsatisfied clause at random and flips one of its atoms: with a certain probability p, the atom is chosen as the one maximizing the sum of satisfied clause weights when flipped; with probability $1-p$ it is chosen randomly. These stochastic moves avoid the algorithm to get stuck in local minima. Being a stochastic algorithm, it is not guaranteed to retrieve the same optimum in multiple restarts: different tries are therefore usually run, and the best solution is finally picked. On the other hand, if the goal is to retrieve the conditional probability of query variables, among the many methods developed to address approximate probabilistic inference, one of the most used is *Monte Carlo* (MC) algorithms, which compute approximate expectations of a probability distribution $p(x)$ using sampling techniques and applying marginalization. Markov chain Monte Carlo (MCMC) [63] algorithms obtain the samples employing a Markov chain whose stationary distribution is the desired $p(x)$. Gibbs sampling, Metropolis-Hastings algorithm and importance sampling are the most used techniques for MC methods applied to graphical models. Marginal and conditional probabilities can also be computed using MC-SAT algorithm [64], which applies a slice sampling technique in combination with SampleSAT routine, in order to perform sound inference also in the presence of deterministic dependencies.

As it happens with many other SRL frameworks, by employing the aforementioned algorithms, inference on a Markov Logic Network is performed over the *ground* graphical model generated by propositionalization over the first-order logic template, using the universe of constants which are available in the domain of interest. This kind of approaches can quickly become extremely costly and often completely unfeasible. In order to solve this propositionalization problem, lifted inference [65], [66], [67] algorithms have been recently introduced, in order to perform inference directly over the first-order logic model, therefore saving both memory occupation and computational time. The approaches by [65] and [66] are based on variable elimination [68], generalized to first-order logic, while Singla and Domingos [67] proposed to build a *lifted* network which aggregates ground atoms into super-nodes with super-features, therefore reducing considerably the size of the graph upon which inference is performed. Recently, also inference algorithms based on probabilistic theorem proving [69], which have shown to have a dramatic impact to the scalability of MLNs. Despite these recent improvements, exploiting efficient inference algorithms for large scale data sets is still one of the most important open problems for MLNs, and in general for the whole SRL community.

[4]The project webpage is http://alchemy.cs.washington.edu

B. Parameter learning

Both conditional and generative learning algorithms have been developed to learn the set of parameters w_j associated to the rules within an MLN. In a generative setting, MLN weights can be learned by maximizing the likelihood of a database of ground predicates, under the closed-world assumption that clauses not present in the database are false. These methods typically require the computation either of the likelihood function itself, or of its gradient, and are therefore extremely expensive. For this reason, the pseudo-likelihood is typically employed in place of the likelihood function, which is much faster to compute but indeed less accurate [67]. Yet, in many applications, it is a priori known which variables are always observed, and which will be queried at prediction time. This is the case of discriminative settings, where one wants to predict the truth value of a set of *query* variables Y, given the set of *evidence* ones X. For example, at prediction time an MLN could be used to infer the probability of some query atoms (e.g., is `Likes(Daniel,Movie2)` true ?), given the evidence of the knowledge base.

In the discriminative setting, the weights of an MLN can be learned by maximizing the *conditional log-likelihood* (CLL) of query atoms Y, which are the target of the learning, given the evidence X, which are always observed. The conditional probability $P(Y=y|X=x)$ is defined as follows:

$$P(Y=y|X=x) = \frac{1}{Z_x} \exp\left(\sum_{i \in F_Y} w_i n_i(x,y)\right) \tag{2}$$

where F_Y is the set of first-order formulae containing query predicates Y and $n_i(x,y)$ is the number of true groundings of clause i in world (x,y). The gradient of the CLL can be computed as follows:

$$\begin{aligned} \frac{\partial}{\partial w_i} \log P_w(y|x) &= n_i(x,y) - \textstyle\sum_{y'} P_w(y'|x) n_i(x,y') && (3) \\ &= n_i(x,y) - E_w[n_i(x,y)] && (4) \end{aligned}$$

Since the computation of the expected number of true groundings $E_w[n_i(x,y)]$ is intractable, an approximation is given by the counts in the Maximum A Posteriori (MAP) state y^*_w: MAP inference is therefore called as a subroutine at every gradient descent step. Several variants to the described method have been implemented to speed up the convergence of the learning procedure: the most efficient is a scaled conjugate gradient method (SCG) which uses the inverse diagonal Hessian as a preconditioner, and which is based on a trusted-region approach to select appropriate search directions.

C. Structure learning

Structure learning is one of the hardest tasks within SRL, and it is in fact an open problem for many existing models. This is particularly true for Markov networks in general, owing to the undecomposable form of the likelihood function, and therefore this also holds for MLNs. Among the most used algorithms for Markov networks structure learning, it is worth mentioning greedy local heuristic search [70], which iteratively adds and removes features from the network, if such moves contribute to improve the model likelihood. When using an SRL framework which employs a logical formalism in addition to a graphical model representation, the structure learning task also relies on inductively learn logic clauses from data description and background knowledge. Learning the graph structure and the logic representation of the model are therefore, in this case, two inter-related tasks which are typically jointly addressed. The main approaches specifically developed for MLNs structure learning are iterated local search [71], top-down [72] and bottom-up [73] clause learning, but in principle other algorithms from inductive logic programming could be employed. As a matter of fact, cascade methods which first learn the clauses following inductive logic programming procedures, and subsequently learn the weights of such clauses, have shown to be an appropriate trade-off between expressivity and efficiency [72].

V. Game modeling with Markov logic

Modeling games with a logic formalism is a well known task. For example, the game description language (GDL) [74] has been designed for this goal, although the rules that can be written with this language are thought more for computer players than for humans. The LUDOCORE game engine [75] has instead been developed to design game rules and specifications on a higher level, by linking game-level concepts developed by game designers to first-order logic rules that can be used by AI reasoning tools.

Being a very general framework, Markov logic allows to model knowledge in terms of logic predicates, which can be either *evidence* (known facts) or *query* (facts to be inferred). When modeling a strategic game, for example, evidence predicates can be used to describe the payoff matrix:[5]

```
Payoff(strategy1,strategy2,outcome1,outcome2)
```

where `strategy1` and `strategy2` are the types corresponding to the sets of possibile actions for players 1 and 2, respectively (and the same for outcomes). Query predicates can describe players' choices:

```
PlayerOneAction(strategy1)
PlayerTwoAction(strategy2).
```

By introducing a hard rule in the model which prevents multiple predicates modeling actions to be true at the same time, a Herbrand interpretation will correspond to a specific choice of strategies simultaneously made by all the players. Inference in the MLN can be used in order to retrieve the Herbrand interpretation with the highest probability, given the (weighted) rules specified in the model. Section VI will show that different problems can be approached by specifying different rules in the MLN.

Sequential or repeated games need a further element in the modeling, that is a sort of timing variable which allows to describe the temporal relationships between events. Consider for example a repeated Rock-Paper-Scissors (Roshambo) game, where each player tries to model the opponent's strategy; in this case, observed game sequences can be modeled with predicates like the following one:

```
PlayerOneObservedAction(object,game)
```

where the grounding

```
PlayerOneObservedAction(ROCK,G3)
```

means that Player 1 played rock during the third game. A predicate `Next(game,game)` can be used to model the information that two games were played consecutively. This kind of context may allow opponent modeling using Markov logic in repeated and stochastic games, by learning the weight of rules which model the dependencies between subsequent choices:

```
PlayerOneObservedAction(object1,game1) ∧
Next(game1,game2) =>
PlayerOneAction(object2,game2)
```

When the number of players in the game grows, more sophisticated rules can be included in the model, so as to incorporate background knowledge of the game (e.g., information about the map of the world where a strategic/military game takes place), or to introduce complex relationships and structures, such as teams, alliances, objectives.

A small set of elementary built-in functions over integer and string types is implemented in Alchemy, so that these can be naturally used without the need of defining extended predicates. For example, integer outcomes can be modeled with the `int` type, upon which operators such as > or <= can be applied (rather than introducing predicates such as `MoreThan` or `LessOrEqualTo`). Apart from such facilities, the pure logic representation of constants does not allow to directly embed numerical attributes in an MLN. Yet, there are some attempts in this direction, namely with grounding-specific weights (GS-MLNs) [59], that allow to have different weights for different groundings of the same first-order clause, and with HMLNs [76], that allow to embed continuous values, and functions over them, as MLN features. These could represent very interesting frameworks when dealing with game domains where a categorical description of the world is not sufficient, and the use of numerical features is required. In this paper, anyhow, the considered domains are described in terms of classic first-order logic, thus without exploiting hybrid features.

Finally, it is also worth mentioning that Markov logic has been extended towards a decision-theoretic framework [53] by attaching utility functions to first-order clauses, heading to the so-called Markov Logic Decision Networks (MLDNs). Such model can be used to represent several decision problems, such as Markov Decision Processes. Yet, for the purpose of this paper, MLDNs do not provide particular advantages with respect to classic MLNs. In particular, the inference algorithm of MLDNs maximizes the overall utility of the network, that is the sum of the utility functions for all players, but the model does not allow to write constraints on such utilities (for example, to encode rules for Nash equilibria). Thus, we preferred to encode utilities within the `Payoff` predicate, as detailed above, and to employ the standard MLN model.

VI. Experiments

We run experiments on MLNs, using the publicly available Alchemy software,[6] version 2.0, developed at the University of Washington. We first present three opponent modeling applications: (1) predicting the serving direction of tennis players; (2) learning the opponent's strategy in an iterated Roshambo game; (3) predicting an adversary's hand in Texas Holdem poker. We will present results on parameter learning of a handcrafted MLN in the first case, while structure learning will be used in the second case, and, finally, both the rules and their parameters will be learned in the third application. Then, we will present some worked examples showing how to model the problem of finding Nash equilibria and Pareto optima in strategic games with Markov logic, showing the advantages but also the main limitations of this approach.

A. Predicting serving direction in tennis matches

Predicting tennis serving direction is a crucial element in a tennis match, as correctly guessing the direction of the serve would give

[5]For the remaining of the paper, we adopt the syntax of the Alchemy system: predicates start with uppercase letters as well as constants, while both variables and type identifiers are lowercase. A tutorial on Alchemy can be found at http://alchemy.cs.washington.edu/tutorial.html.

[6]http://alchemy.cs.washington.edu

a player a great advantage in terms of effectiveness of his/her return. This task has been subject of several studies [77], and it is a common scenario in many sports, being similar to predicting the pitching sequence in baseball [78], or the direction of penalty kicks in soccer [79]. We use the data set presented in [77], which consists in a collection of annotated tennis matches from Grand Slam finals in the last 30 years. For each point played in the match, we extracted the following information in the form of logic predicates:

- the set and game in which the point was played
- which player served
- the serving court (Even/Odd)
- the serving direction (Left/Right)
- which player won the point

Using this information, the goal is to predict, at each point in the match, the serving direction of a player (query predicate), given evidence of all the facts describing the match until that point. The query predicate is therefore defined as:

```
Left(set,game,point)
```

which is true for a certain grounding `Left(S,G,P)` if in point `P` of game `G` of set `S` the serving direction was left; the truth value is false if the serving direction was right.

We model the situation with a MLN whose rules try to capture the strategy of the serving player, in relation to the current game state, and to the points played in the immediate past. We consider two tennis matches as our test-bed: the 1995 match between Agassi and Sampras (AS) at the US Open, and the 1987 match between Cash and Wilander (CW) at the Australian Open. For each match, the first three sets were considered as the training set (184 points for CW, 172 for AS), while the remaining sets (one for AS, two for CW) were the test set (132 points for CW, 64 for AS). We first consider a very simple MLN (MLN_1) modeling only the following rules:

```
Serve(set,game,PLR1) ∧ Court(set,game,point,EVEN)
            => Left(set,game,point)

Serve(set,game,PLR1) ∧ Court(set,game,point,ODD)
            => Left(set,game,point)

Serve(set,game,PLR2) ∧ Court(set,game,point,EVEN)
            => Left(set,game,point)

Serve(set,game,PLR2) ∧ Court(set,game,point,ODD)
            => Left(set,game,point)
```

Such rules model just the prior probability that each player (`PLR1` or `PLR2`) will serve left/right, according to the odd/even court from which they will serve. In Alchemy syntax, a shortcut can be used to quickly write different rules for each different grounding of a certain variable, using a $+$ sign: in this case the four rules above would be summarized by a single formula using such notation:

```
Serve(set,game,+player) ∧
Court(set,game,point,+court)
  => Left(set,game,point)
```

The second model (MLN_2) adds a more complex set of rules, considering also the relationships between either two consecutive points:

```
   Serve(set,game,+player1) ∧
 ServedLeft(set,game,point1) ∧
Won(set,game,point1,+player2) ∧
      Next(point1,point2)
    => Left(set,game,point2)
```

or two consecutive points with the same serving court:

Match	Predictor	Accuracy
AS	Baseline AL	45.3
AS	Baseline AR	54.7
AS	Baseline CRT_1	58.3
AS	Baseline CRT_2	41.7
AS	MLN_1	64.1
AS	MLN_2	68.8
CW	Baseline AL	47.0
CW	Baseline AR	53.0
CW	Baseline CRT_1	59.4
CW	Baseline CRT_2	40.6
CW	MLN_1	61.4
CW	MLN_2	63.6

Table III: Results obtained by MLN on the task of predicting the serving direction in a tennis match. The baseline predictors either predict the server to serve always left (AL), always right (AR), right from odd and left from even courts (CRT_1), right from even and left from odd courts (CRT_2).

```
   Serve(set,game,+player1) ∧
 ServedLeft(set,game,point1) ∧
Won(set,game,point1,+player2) ∧
      Next(point1,point2) ∧
      Next(point2,point3)
    => Left(set,game,point3)
```

The first rule models the probability that a player will serve left (or right), given the observed facts that at the previous point (i) he had served left (or right), (ii) he had won (or lost) the point. The second rule is almost identical to the previous, but considers as observed fact the previous point played in the same (odd or even) court. For each of the two matches, we considered the first three sets as the training set to learn rule weights, while the remaining sets were used as the test set, where inference algorithms were run, in order to predict the truth value of the query atoms (`Left` predicate), given the evidence ones (all the remaining predicates). The weights of the rules were therefore learned from the training set in a discriminative setting, by employing the rescaled conjugate gradient algorithm, running with MAP inference to estimate the expected counts. For all the other parameters of learning and inference algorithms, we relied on Alchemy default values.

Table III shows the results obtained by the two MLN models, as well as some baseline classifiers which trivially always predicts the serving direction as left/right, or depending on the serving court (odd/even). The reported accuracy is simply computed as the percentage of points for which the serving direction was correctly predicted. These results show that the proposed approach is indeed promising and might be applied to several different domains, even in multi-player situations.

Table III shows the results obtained by the two MLN models, as well as some baseline classifiers which trivially always predict the serving direction as left/right, or depending on the serving court (odd/even). The reported accuracy is simply computed as the percentage of points for which the serving direction was correctly predicted. These results show that the proposed approach is indeed promising and might be applied to several different domains, even in multi-player situations.

B. Learning strategies in Iterated Roshambo

As a first simple structure learning experiment, we considered an iterated Roshambo game. The interest in iterated games, even very simple ones, has largely spread after the pioneering work by Axelrod on the iterated prisoner's dilemma [80], where many different strategies have been studied, with the goal of understanding the dynamics of repeated game situations, where the history of past matches was known to each player. The scenario is totally different from a static situation, as players can naturally learn from their errors and from the actions observed in the behavior of their opponent, and thus change their strategies accordingly.

In order to test the ability of SRL models to capture strategies in iterated games, we generated a sequence of iterated Roshambo games between two players who follow these strategies:

- player one adopts a *random* strategy, so that, at each game, his/her action is randomly chosen between the three possible `R`/`P`/`S` strategies (therefore independently from the previous games);
- player two implements a *win-stay, lose-switch* strategy, which consists in keeping playing a winning strategy, and randomly changing it if the previous game ended up with a draw or a loss.

We generated a sequence of 1,000 games following the aforementioned criteria, and we used the built-in structure learning algorithm of Alchemy to learn logic clauses whose query predicate is the strategy adopted by the second player. The set of facts from which the clauses are learned consists in a set of ground predicates which describe the sequence of played games, like the following ones:

```
PlayerOneRock(G12)
PlayerTwoPaper(G12)
WonByPlayerTwo(G12)
Next(G12,G13)
```

which, for example, indicate that game `G12` was won by player two (paper vs. rock), and the predicate `Next` indicates the sequentiality between games `G12` and `G13`. By observing this set of ground facts, the system is capable of learning a set of clauses which explain the *win-stay, lose-switch* strategy for the second player. Such strategy is in fact represented by rules such as the following one, correctly learned by Alchemy[7]:

```
PlayerOneRock(game1) ∧ WonByPlayerTwo(game1) ∧
  Next(game1,game2) => PlayerTwoPaper(game2)
```

Such rule explains that, if player one played `Rock` in `game1` and lost, then player two keeps playing `Paper` (the winning strategy in `game1`) also in `game2`, which is the game following `game1` in the generated sequence (`Next` predicate). The set of rules learned by the structure learning algorithm could now be used to run probabilistic inference on new test cases, in order to understand the behavior of an adversary. This is what will be done in the next experiment.

C. Predicting opponent's hand in Texas Holdem poker

As a third and more complete case study, we considered Texas Holdem poxer, with the aim of predicting the hand of an adversary at the showdown. We collected data from the IRC Poker Logs data set[8] by considering the three players with the largest number of played hands: `kwikfish`, `tigger2` and `Poki_S2`. For each of these three players, we collected all the hands which arrived at the showdown, in order to have the ground truth of the real hand of the player. In this way, we collected a total of 1664 (`Poki_S2`), 1094 (`tigger2`) and 894 (`kwikfish`) hands, respectively.

We modeled the domain with a set of 156 evidence predicates, describing observable facts, and 9 query predicates describing the possible hands, to be predicted (`HighCard`, `Pair`, `TwoPair`, `ThreeOfAKind`, `Straight`, `Flush`, `FullHouse`, `FourOfAKind`, `StraightFlush`). The evidence predicates describe the development of the hand, by including information regarding the bets and the cards on board at each phase of the hand. For example, the following ground predicate:

```
PairOnBoardAtFlop(947366462)
```

indicates that a pair was on board after the flop during game having id 947366462 in the IRC Poker Logs database, while

```
RaiseAtPreflop(947366462)
```

indicates that in the same game the observed opponent performed a raise at preflop game stage. We also encoded the information regarding the a-priori probability of a hand, given the seven cards on board, with a set of predicates used to indicate the most probable point: for example, the predicate

```
MostProbableTwoPair(947366462)
```

means that, considering all possible preflop configurations, two pair is the most probable hand.

We prepared a set of 76 hard rules in order to model elementary information regarding poker rules, such as the fact that a player cannot have a FullHouse if there is neither a pair nor three cards of a kind on board:

```
!PairOnBoard(g) ∧ !ThreeOfAKindOnBoard(g)
        => !FullHouse(g).
```

or the fact that a player cannot have a Flush if there are not at least three cards of the same seed on board:

```
!AtLeastThreeCardsSameSeedOnBoard(g) => !Flush(g).
```

Within these hard rules, we also modeled the trivial information that two different points are mutually exclusive, which means, for example, that a player cannot have both a Flush and a FullHouse at the same time, since the actual hand of each player is given by the highest point that can be obtained with the given seven cards:

```
!(Flush(g) ∧ FullHouse(g)).
```

Then we independently run three different experiments, one for each opponent (`kwikfish`, `tigger2`, `Poki_S2`), following the same common procedure. For each player we built a distinct dataset, by splitting the available hands in a training set and a test set (the first 2/3 and the remaining 1/3 of the total amount of given hands, respectively). In order to learn the clauses from training data, we employed both the Alchemy's built-in algorithm named `learnstruct`, and the well-known FOIL algorithm [81]. While Alchemy is capable of jointly learning both the clauses and their weights, clearly FOIL can learn only the clauses, and thus weight learning was subsequently performed with Alchemy on the set of clauses returned by FOIL. In order to have a large set of clauses for FOIL, we run the algorithm with different values for its precision-level parameter a. FOIL, in fact, returns clauses only having a precision $> a$, and therefore by selecting different values for a we obtain different clause sets. Whereas using the built-in structure learning algorithm of Alchemy is certainly more appropriate for this problem, the use of FOIL in combination with Alchemy's parameter learning wants to highlight the fact that, in principle, any combination of clause and weight learning algorithms could be employed for this task. In particular, this includes also those SRL frameworks that do not allow yet to directly perform structure learning (such as, for example, ProbLog). Being FOIL a very simple, well-known and fast method for clause learning (it took just a few seconds to run on the Poker data set), we used it for this comparison in place of more sophisticated systems such as Aleph[9] or TILDE [82].[10]

[7]We used the following parameters: `-minWt 0.001 -penalty 0.001 -beamSize 10 -numClausesReEval 20 -tryAllFlips true`

[8]For our experiments we employed the 200001 sample data, available at http://poker.cs.ualberta.ca/IRCdata/.

[9]http://www.cs.ox.ac.uk/activities/machlearn/Aleph/aleph.html

[10]Following the framework described in [72], when learning the weights of the rules extracted by FOIL, we used the modified version of Alchemy that performs on-line max-margin weight, developed by Tuyen N. Huynh and available at http://www.ai.sri.com/~huynh/, acheiving slightly better performance than the original algorithm in Alchemy.

By following the aforementioned procedures, we obtained two distinct MLNs (one from Alchemy's structure learning, the other from FOIL plus Alchemy's weight learning) to be used for inference on the test set, where, given the evidence predicates for each test hand, the system inferred the most probable query atom, that corresponded to the most probable player hand.

In order to compare against other classical machine learning algorithms for opponent modeling, we considered Support Vector Machines (SVMs) with Gaussian kernel and Artificial Neural Networks (ANNs). We encoded the evidence predicates into a boolean feature vector $v = [v_1, \ldots, v_k]$, where $v_j = 1$ if the j-th predicate is true for the considered hand, and $v_j = 0$ otherwise. We employed a validation set to select the number of hidden neurons for ANNs and the kernel parameters (C and γ for the RBF kernel, C for the linear kernel) for SVMs. For ANNs, 10 restarts were used to perform training with early stopping on the validation set to select the best architecture. It is worth mentioning that, differently from the tennis serving direction problem, in this case the problem can be easily modeled as a multi-class classification task for SVMs and ANNs, since the sequentiality of data is not considered, while being a crucial ingredient in the tennis task. We also employed plain decision trees using the C4.5 algorithm [83] to compare against a pure rule-based, interpretable approach.

Table IV reports the accuracy achieved by our classifiers (MLN and FOIL+MLN) (simply computed as the percentage of correctly predicted test hands), compared against the C4.5, SVM and ANN classifiers, as well as a baseline predictor which simply predicts, as the opponent's hand, the most probable one given the five cards on board. We can notice how the MLN approach largely outperforms the baseline, and achieves a better accuracy than ANNs too, while the SVM model performs slightly better on two out of three opponents. Yet, the distinguishing characteristic of the proposed approach is in terms of *interpretability*. We can also observe that MLN performs much better than decision trees, and also slightly better than FOIL+MLN, which yet achieves good quality results. By observing the learned rules and their associated weights, we can discover some strategies driving each player's choices. For example, consider the following rule learned for the `kwikfish` player:

```
CheckRiver(g) ∧ MostProbableFlush(g)
        => Flush(g)
```

It suggests that this player has adopted the strategy of checking after the river when having a Flush in hand. In other cases, the learned rule suggests that the player does *not* have a certain hand. For example, the following clause for the `kwikfish` player:

```
RaiseAtRiver(g) => !Pair(g)
```

suggests that, in case the player raises after the river, he likely does not have a pair.

More complex rules can also be learned. For the `tigger2` player, for example, the following clause indicates that, when a `Pair` is the most probable hand for that player (as there is a pair on board), and the player has checked both at turn and at river, he is likely to really have a `Pair`:

```
CheckAtTurn(g) ∧ CheckAtRiver(g) ∧
MostProbablePair(g) ∧
PairOnBoard(g) ∧ OpponentBetAtFlop(g) =>
Pair(a1)
```

Another rule for the same player suggests instead a different kind of behavior. With a high pot (over 100 dollars), even if the most probable hand is a `Pair`, in case he has made a `Call` both at flop and at river, the player might have two pairs:

```
CallAtFlop(g) ∧ CallAtRiver(g) ∧
MoreThan100DollarsAtTurn(g) ∧
MostProbablePair(g) ∧ OpponentBetAtTurn(g) =>
TwoPair(g)
```

Clearly, in some cases, there are some evidence predicates which, if true, automatically exclude the possibility of some query predicates. For example, if the predicate `BestPossibleStraight(g)` is true, then the player cannot have a `Flush`, nor a `FullHouse`, which in turns rules out the possibility that, for example, there are three cards of the same seed, or a pair, on board. The following rule, learned for player `Poki_S2`, indicates that, if `Straight` is the best possible hand for that player, but he has always checked at flop, turn and river, he is likely to have just a `HighCard`.

```
BestPossibleStraight(g) ∧ CheckAtFlop(g) ∧
CheckAtTurn(g) ∧ CheckAtRiver(g) => HighCard(g)
```

Note that, in some cases, Alchemy learns rules that are similar to the hard constraints that we added to the model to represent basic Texas Hold'em rules and principles. For example, the following rule is learned with a very high score:

```
!MostProbableFlush(g) ∧
!ThreeCardsSameSeedOnBoard(g) => !Flush(g)
```

Such a rule is indeed very close to the constraint explaining that without at least three cards of the same seed on board, it is not possible to have a flush.

Moreover, it is worth saying that, in the considered setting, the MLN approach did not exploit sequentiality in the data (as it happened in the tennis serving direction problem), since, in our present formulation of the task, we only analyzed hands arrived at the showdown and not the sequence of all the played hands. Therefore, it was not possible to employ in the MLN predicates that model the consequentiality of two hands, nor collective classification algorithms that would also allow to jointly predict the strategies of multiple players. Despite not taking advantage of relational information, this experiment still shows the potential of the approach. In principle, Alchemy allows to perform inference also on partially observed databases, that is data collections containing predicates with unknown truth values. Such predicates would correspond to hidden variables in the underlying model, and the Expectation-Maximization (EM) algorithm could be employed to infer the value of such predicates [84] and thus to use, in principle, the whole sequence of played hands, without limiting to the ones arrived at showdown. Nevertheless, the computational cost to perform learning and inference in such partially-observed databases is clearly much greater [84].

It is also worth mentioning that different classifiers could obviously be developed for the different game stages, in order to support players throughout the match. Moreover, alternative models could be conceived, for example aiming to directly predict the cards in the hand of the opponent: yet, in that case a much larger training set would likely be needed, in order to help generalization and avoid overfitting.

The data set encoded in the Alchemy language, as well as the MLN models learned for each of the three considered players, are available for download at the following link:

http://lia.disi.unibo.it/~ml/TexasHoldemMLN.tgz

D. Finding Nash equilibria and Pareto optimal solutions

The experiments described so far have focused on the opponent modeling problem, where SRL-based approaches can give a great contribution in constructing interpretable models. Yet, in this section

	kwikfish	Poki_S2	tigger2
MostProbableHand	40.2	44.4	45.2
ANN	44.4	52.1	57.0
C4.5	45.9	54.6	59.2
SVM	53.1	60.5	59.8
FOIL+MLN	50.7	56.5	58.9
MLN	50.2	57.3	61.1

Table IV: Accuracy achieved on the problem of predicting an opponent's hand in Texas Holdem poker. MLN is compared against Artificial Neural Networks (ANNs), Support Vector Machines (SVMs), and a baseline which predicts the most probable hand.

we also want to point out that SRL frameworks are also suitable for finding equilibria in games, although some considerations have to be taken into account.

Suppose we want to model with Markov logic the Prisoner's Dilemma (Example 1). We employ three (typed) predicates:

```
Payoff(strategy,strategy,outcome,outcome)
        PlayerOneAction(strategy)
        PlayerTwoAction(strategy)
```

where the arguments of predicate `Payoff` represent, respectively, the strategies played by players `P1` and `P2`, followed by their outcomes. The set of possible strategies will be defined by the set of constants:

```
strategy={C,D}
```

with `C` for `Cooperate` and `D` for `Defeat`. For example, the ground predicate `Payoff(C,D,-10,0)` will represent the top-right element in the payoff matrix represented in Table I.

Looking for Nash equilibria corresponds to imposing a constraint in the form of first-order logic rule, as in the following formula:

```
PlayerOneAction(s1) ∧ PlayerTwoAction(s2) ∧
(Payoff(s1,s2,o1,o2) ∧ Payoff(s3,s2,o3,o4) ∧
              s1!=s3 ∧ o3>o1).
```

which means that it is impossible that `P1` prefers strategy `s1` to strategy `s3`, if the outcome `o3` is greater than `o1`, given that `P2` plays the same strategy `s2` in the two cases. Clearly, a symmetric version of this rule also has to be added for `P2`. Including these formulae in the MLN and running an inference algorithm, such as MaxWalkSAT, results in finding the correct Nash equilibrium (both players defeating), as shown in the following predicate list, where each ground predicate is associated to the truth value (0=false, 1=true) retrieved by the inference algorithm.

```
PlayerOneAction(C)   0
PlayerOneAction(D)   1
PlayerTwoAction(C)   0
PlayerTwoAction(D)   1
```

It is worth remarking that inference algorithms which compute the MAP state aim at finding the configuration of truth values of query variables minimizing the cost of unsatisfied clauses: in the case of Nash equilibria, this corresponds to find only *pure-strategies* equilibria, since the solution is searched through the search space of truth value logic assignments to query variables, with no possibility of exploiting mixed strategies.

In the case of games with multiple pure Nash equilibria, different runs of the MAP inference algorithm with different seed initialization would find different equilibria, obviously always obtaining a total sum of unsatisfied clauses equal to zero. Yet, to avoid performing multiple restarts to retrieve the multiple equilibria, it could be possible to simply apply the enumeration mode of a SAT solver within the inference engine, so that all the pure Nash equilibria of the game could be retrieved (this is currently not available in the Alchemy software). This MLN approach with MAP inference can be naturally applied to find Nash equilibria in several game categories. With the Gamut software[11] we generated a random graphical game with 5 players and 4 actions each. The MLN model was easily adapted to a multi-player configuration, and the inference engine was able to correctly retrieve the pure Nash equilibria detected with the `gambit-enumpure` software,[12] a suite of game theory software tools.

Things are instead different when dealing with mixed strategies. Alchemy in fact does not allow a proper and direct representation of mixed strategies, which means that the probabilistic clauses in the model can only be quantified over pure strategies. Inference algorithms which compute the marginal probabilities of query predicates given the evidence, such as MC-SAT or MCMC (see Section IV) also basically estimate the frequencies of the strategies at pure Nash equilibria, but they cannot directly model mixed strategies. An extension of the MLN framework would thus be necessary in order to handle also mixed strategies: within this context, GS-MLNs and HMLNs could represent two notable starting points, as they allow to introduce numerical features in the MLN model. Other formalisms, such as DTProblog and ICL, could be extended as well in this direction: as a matter of fact, in [28] a brief discussion on the representation of stochastic strategies in the ICL framework is given. Also Probabilistic Soft Logic (PSL) [85] could be employed within this context, as it allows to assign soft truth values to predicates (as it happens, for example, in fuzzy logic) and thus it could be used to model probablity distributions upon strategies.

We conclude with a final example on finding Pareto optima. As already stated, in the PD game the Nash equilibrium does not correspond to a Pareto optimal solution. In order to find the Pareto optima of the game, the rule used to find Nash equilibria must be substituted with the following one:

```
(Payoff(s1,s2,o1,o2) ∧ Payoff(s3,s4,o3,o4) ∧
PlayerOneAction(s1) ∧ PlayerTwoAction(s2) ∧
     (s1!=s3 ∨ s2!=s4) ∧ (o4>o2 ∨ o3>o1)).
```

In this way, the inference algorithm will look for a configuration of strategies such that there does not exist a different configuration where every player is at least as well off and at least one player strictly better off: in this case, the configuration with both players cooperating is found. Note that, since in the current implementation Alchemy needs to build the set of grounded clauses in order to perform MAP inference, this rule will scale up with the fourth power of the number of strategies, because four different strategies (`s1`, `s2`, `s3` and `s4`) need to be grounded, while they are three for Nash equilibria. Nevertheless, this is an additional example of exploiting the proposed framework to quickly model different strategic constraints and situations using a powerful interpretable language.

VII. Conclusions and future work

In this paper we presented a statistical relational learning (SRL) approach to game theory. This framework allows to use the formalism of first-order logic to describe several different categories of games, as well as to use graphical models in order to handle uncertainty in data. A number of classic game-theoretic problems can therefore be easily modeled and addressed by employing the formalism of SRL, such as predicting the strategies of an adversary, or finding Nash equilibria. Preliminary results obtained with the formalism of Markov logic networks show the potential of the approach, in

[11]http://gamut.stanford.edu/
[12]http://gambit.sourceforge.net/

particular by highlighting the great expressive power of the proposed framework. Clearly, there are some limitations within this approach: MLNs currently have limited support for the integration of numerical features within the model, and thus mixed strategies are difficult to handle; in addition, the implementation of the enumeration mode of a SAT solver within the framework would allow to directly retrieve multiple pure Nash equilibria, which currently can be obtained only by performing multiple inference runs.

Many future directions of research can be identified. From a machine learning point of view, one of the most important and challenging tasks would be that of developing fast ad-hoc inference algorithms for certain categories of games represented with an SRL language: within this context, applications of the recently introduced lifted inference paradigm and probabilistic model checking could be of great interest. Moreover, different solutions to the problems described in this paper could certainly be attained by employing different SRL frameworks, such as DTProbLog, Probabilistic Soft Logic or Independent Choice Logic, which have the potential to model several game-theoretic tasks, in particular regarding mixed strategies. To address the challenging task of handling mixed strategies, in fact, one possibility could be that of using a soft assignment of truth values to predicates (as it happens, for example, in PSL) or to exploit utility functions, such as the ones defined in DTProbLog or ICL. Applying SRL to a self-play learning setting would be another very interesting line of research: in this case, the SRL approach would need to be integrated within a reinforcement learning framework, thus novel algorithms for both structure and parameter learning within this context should be conceived.

On the game theory perspective, an extremely challenging opportunity would be that of applying the described approach to multi-player situations in real-world strategy games, such as Poker or Risk. This is a framework where collective classification algorithms could be very powerful, and the highly relational nature of data could be naturally exploited. The automatic discovery of interpretable strategies has also a wide variety of applications in many different domains, including social sciences, economics, and bioinformatics.

References

[1] N. Nisan, T. Roughgarden, E. Tardos, and V. Vazirani, *Algorithmic game theory*. Cambridge University Press, 2007.

[2] I. Rezek, D. S. Leslie, S. Reece, S. J. Roberts, A. Rogers, R. K. Dash, and N. R. Jennings, "On similarities between inference in game theory and machine learning," *J. Artif. Intell. Res.*, vol. 33, pp. 259–283, 2008.

[3] M. J. Kearns, M. L. Littman, and S. P. Singh, "Graphical models for game theory," in *UAI Proceedings, 2001*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2001, pp. 253–260.

[4] C. Daskalakis and C. H. Papadimitriou, "Computing pure nash equilibria in graphical games via markov random fields," in *ACM Conference on Electronic Commerce*, J. Feigenbaum, J. C.-I. Chuang, and D. M. Pennock, Eds. ACM, 2006, pp. 91–99.

[5] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *CoRR*, vol. cs.AI/9605103, 1996.

[6] M. L. Littman, "Value-function reinforcement learning in Markov games," *Cognitive Systems Research*, vol. 2, no. 1, pp. 55–66, 2001.

[7] C. J. C. H. Watkins and P. Dayan, "Technical note q-learning," *Machine Learning*, vol. 8, pp. 279–292, 1992.

[8] A. Davidson, D. Billings, J. Schaeffer, and D. Szafron, "Improved opponent modeling in poker," in *2000 International Conference on Artificial Intelligence*, 2000, pp. 1467–1473.

[9] D. Ashlock, E.-Y. Kim, and N. Leahy, "Understanding representational sensitivity in the iterated prisoner's dilemma with fingerprints," *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, vol. 36, no. 4, pp. 464–475, 2006.

[10] G. Brown, "Iterative solution of games by fictitious play," *Activity analysis of production and allocation*, vol. 13, no. 1, pp. 374–376, 1951.

[11] N. Cesa-Bianchi and G. Lugosi, *Prediction, learning, and games*. Cambridge University Press, 2006.

[12] C. J. Maddison, A. Huang, I. Sutskever, and D. Silver, "Move evaluation in go using deep convolutional neural networks," *CoRR*, vol. abs/1412.6564, 2014. [Online]. Available: http://arxiv.org/abs/1412.6564

[13] S. M. Lucas, "Learning to play othello with n-tuple systems," *Australian Journal of Intelligent Information Processing*, vol. 4, pp. 1–20, 2008.

[14] K. Krawiec and M. G. Szubert, "Learning n-tuple networks for othello by coevolutionary gradient search," in *Proc. of the 13th annual conference on Genetic and evolutionary computation*. ACM, 2011, pp. 355–362.

[15] T. Runarsson and S. M. Lucas, "Preference learning for move prediction and evaluation function approximation in othello," *Computational Intelligence and AI in Games, IEEE Transactions on*, vol. 6, no. 3, pp. 300–313, 2014.

[16] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[17] C. Clark and A. Storkey, "Training deep convolutional neural networks to play go," in *ICML-15 Proceedings, Lille, France*, 2015, pp. 1766–1774.

[18] S. Gelly and D. Silver, "Monte-carlo tree search and rapid action value estimation in computer go," *Artificial Intelligence*, vol. 175, no. 11, pp. 1856 – 1875, 2011.

[19] D. Robles, P. Rohlfshagen, and S. M. Lucas, "Learning non-random moves for playing othello: Improving monte carlo tree search," in *IEEE Conference on Computational Intelligence and Games (CIG)*. IEEE, 2011, pp. 305–312.

[20] S. Muggleton and L. de Raedt, "Inductive logic programming: Theory and methods," *The Journal of Logic Programming*, vol. 19–20, Supplement 1, no. 0, pp. 629 – 679, 1994, special Issue: Ten Years of Logic Programming.

[21] K. Kersting, "An inductive logic programming approach to statistical relational learning," *AI Commun.*, vol. 19, no. 4, pp. 389–390, 2006.

[22] I. Thon, N. Landwehr, and L. D. Raedt, "Stochastic relational processes: Efficient inference and applications," *Machine Learning*, vol. 82, no. 2, pp. 239–272, 2011.

[23] S. Joshi, K. Kersting, and R. Khardon, "Generalized first order decision diagrams for first order Markov decision processes," in *IJCAI-09 Proceedings, Pasadena, USA*, C. Boutilier, Ed., 2009, pp. 1916–1921.

[24] G. V. den Broeck, I. Thon, M. van Otterlo, and L. D. Raedt, "Dtproblog: A decision-theoretic probabilistic prolog," in *AAAI Proceedings, Atlanta, Georgia, USA*, M. Fox and D. Poole, Eds. AAAI Press, 2010.

[25] A. Rettinger, M. Nickles, and V. Tresp, "Statistical relational learning of trust," *Machine Learning*, vol. 82, no. 2, pp. 191–209, 2011.

[26] P. Tadepalli, R. Givan, and K. Driessens, "Relational reinforcement learning: An overview," in *ICML-04 Workshop on Relational Reinforcement Learning*, 2004.

[27] T. Croonenborghs, J. Ramon, H. Blockeel, and M. Bruynooghe, "Online learning and exploiting relational models in reinforcement learning," in *IJCAI-07 Proceedings, Hyderabad, India*, M. M. Veloso, Ed., 2007, pp. 726–731.

[28] D. Poole, "The independent choice logic for modelling multiple agents under uncertainty," *Artificial intelligence*, vol. 94, no. 1, pp. 7–56, 1997.

[29] M. J. Osborne, *An Introduction to Game Theory*. Oxford University Press, USA, August 2003.

[30] J. Nash, "Non-cooperative games," *The Annals of Mathematics*, vol. 54, no. 2, pp. 286–295, 1951.

[31] L. S. Shapley, "Stochastic Games," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 39, no. 10, pp. 1095–1100, 1953.

[32] R. A. Howard, *Dynamic Programming and Markov Processes*. The MIT Press, 1960.

[33] M. L. Littman, "Markov games as a framework for multi-agent reinforcement learning," in *ICML*, W. W. Cohen and H. Hirsh, Eds. Morgan Kaufmann, 1994, pp. 157–163.

[34] J. C. Harsanyi, "Games with incomplete information played by bayesian players, part iii. the basic probability distribution of the game," *Management Science*, vol. 14, no. 7, pp. 486–502, 1968.

[35] L. Getoor and B. Taskar, Eds., *Introduction to Statistical Relational Learning*. The MIT Press, 2007.

[36] R. Smullyan, *First-Order Logic*. Dover Publications, January 1995.

[37] C. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*, 1st ed. Springer, October 2007.

[38] S. Muggleton, "Stochastic logic programs," in *New Generation Computing*. Academic Press, 1996.

[39] ——, "Learning stochastic logic programs," in *Proceedings of the AAAI2000 Workshop on Learning Statistical Models from Relational Data*, vol. 5, 2000.

[40] M. Jaeger, "Relational bayesian networks," in *UAI-97 Proceedings, Providence, Rhode Island, USA*, D. Geiger and P. P. Shenoy, Eds. Morgan Kaufmann, 1997, pp. 266–273.

[41] K. Kersting and L. D. Raedt, "Basic principles of learning bayesian logic programs," in *Probabilistic Inductive Logic Programming - Theory and Applications*, ser. LNCS, L. D. Raedt, P. Frasconi, K. Kersting, and S. Muggleton, Eds., vol. 4911. Springer, 2008, pp. 189–221.

[42] T. Sato and Y. Kameya, "Prism: a language for symbolic-statistical modeling," in *IJCAI-97 Proceedings*, vol. 97, 1997, pp. 1330–1339.

[43] N. Friedman, L. Getoor, D. Koller, and A. Pfeffer, "Learning probabilistic relational models," in *IJCAI-99 Proceedings*, 1999, pp. 1300–1309.

[44] M. Jaeger, "Type extension trees: a unified framework for relational feature construction," in *Proc. MLG-06*, 2006.

[45] M. Jaeger, M. Lippi, A. Passerini, and P. Frasconi, "Type extension trees for feature construction and learning in relational domains," *Artif. Intell.*, vol. 204, pp. 30–55, 2013.

[46] L. De Raedt, A. Kimmig, and H. Toivonen, "Problog: A probabilistic prolog and its application in link discovery," in *IJCAI-07 Proceedings, Hyderabad, India*, M. M. Veloso, Ed., 2007, pp. 2462–2467.

[47] E. Shapiro and L. Sterling, *The Art of PROLOG: Advanced Programming Techniques*. The MIT Press, 1994.

[48] P. Frasconi, F. Costa, L. D. Raedt, and K. D. Grave, "klog: A language for logical and relational learning with kernels," *Artificial Intelligence*, vol. 217, no. 0, pp. 117 – 143, 2014.

[49] M. Diligenti, M. Gori, M. Maggini, and L. Rigutini, "Bridging logic and kernel machines," *Machine Learning*, vol. 86, no. 1, pp. 57–88, 2012.

[50] G. Gnecco, M. Gori, S. Melacci, and M. Sanguineti, "Foundations of support constraint machines," *Neural Computation*, vol. 27, no. 2, pp. 388–480, 2015.

[51] L. Getoor, "Tutorial on statistical relational learning," in *ILP 2005, Bonn, Germany, Proceedings*, ser. Lecture Notes in Computer Science, S. Kramer and B. Pfahringer, Eds., vol. 3625. Springer, 2005, p. 415.

[52] J. Chen and S. Muggleton, "Decision-theoretic logic programs," in *Proceedings of ILP*. Citeseer, 2009, p. 136.

[53] A. Nath and P. Domingos, "A language for relational decision theory," in *Proc. Int. Workshop on Statistical Relational Learning, Leuven*, 2009.

[54] D. Poole, "The independent choice logic and beyond," in *Probabilistic inductive logic programming*. Springer, 2008, pp. 222–243.

[55] M. Richardson and P. Domingos, "Markov logic networks," *Machine Learning*, vol. 62, no. 1-2, pp. 107–136, 2006.

[56] F. Wu and D. S. Weld, "Automatically refining the wikipedia infobox ontology," in *17th international conference on World Wide Web, Proceedings*, ser. WWW. New York, NY, USA: ACM, 2008, pp. 635–644.

[57] S. D. Tran and L. S. Davis, "Event modeling and recognition using Markov logic networks," in *ECCV 2008 Proceedings*. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 610–623.

[58] S. Kok and P. Domingos, "Extracting semantic networks from text via relational clustering," in *ECML/PKDD 2008 Proceedings, Antwerp, Belgium*. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 624–639.

[59] M. Lippi and P. Frasconi, "Prediction of protein beta-residue contacts by Markov logic networks with grounding-specific weights," *Bioinformatics*, vol. 25, no. 18, pp. 2326–2333, 2009.

[60] M. Lippi, M. Bertini, and P. Frasconi, "Collective traffic forecasting," in *ECML/PKDD 2010 Proceedings, Barcelona, Spain*, ser. Lecture Notes in Computer Science, J. L. Balcázar, F. Bonchi, A. Gionis, and M. Sebag, Eds., vol. 6322. Springer, 2010, pp. 259–273.

[61] D. Roth, "On the hardness of approximate reasoning," *Artificial Intelligence*, vol. 82, no. 1-2, pp. 273–302, 1996.

[62] H. Kautz, B. Selman, and Y. Jiang, "A general stochastic approach to solving problems with hard and soft constraints," 1996.

[63] W. R. Gilks, *Markov Chain Monte Carlo in Practice*. Chapman Hall/CRC, 1995.

[64] H. Poon and P. Domingos, "Sound and efficient inference with probabilistic and deterministic dependencies," in *AAAI Proceedings, Boston, USA*. AAAI Press, 2006, pp. 458–463.

[65] D. Poole, "First-order probabilistic inference," in *IJCAI-03 Proceedings, Acapulco, Mexico*, G. Gottlob and T. Walsh, Eds. Morgan Kaufmann, 2003, pp. 985–991.

[66] R. D. S. Braz, E. Amir, and D. Roth, "Lifted first-order probabilistic inference," in *IJCAI-05 Proceedings, Edinburgh*, L. P. Kaelbling and A. Saffiotti, Eds. Professional Book Center, 2005, pp. 1319–1325.

[67] P. Singla and P. Domingos, "Memory-efficient inference in relational domains," in *Proc. AAAI, Boston, USA*. AAAI Press, 2006.

[68] R. Dechter, "Bucket elimination: A unifying framework for probabilistic inference," pp. 211–219, 1996.

[69] V. Gogate and P. Domingos, "Probabilistic theorem proving," in *Proc. UAI 2011, Barcelona, Spain*. AUAI Press, 2011, pp. 256–265.

[70] A. McCallum, "Efficiently inducing features of conditional random fields," in *Proceedings of the Nineteenth Conference on Uncertainty in Artificial Intelligence*, ser. UAI-03. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2003, pp. 403–410.

[71] M. Biba, S. Ferilli, and F. Esposito, "Structure learning of Markov logic networks through iterated local search," in *Proc. ECAI-08*, ser. Frontiers in Artificial Intelligence and Applications, M. Ghallab, C. D. Spyropoulos, N. Fakotakis, and N. M. Avouris, Eds., vol. 178. IOS Press, 2008, pp. 361–365.

[72] T. N. Huynh and R. J. Mooney, "Discriminative structure and parameter learning for Markov logic networks," in *Proc. ICML-08, Helsinki, Finland*, ser. ACM Int. Conf. Proceeding Series, W. W. Cohen, A. McCallum, and S. T. Roweis, Eds., vol. 307. ACM, 2008, pp. 416–423.

[73] L. Mihalkova and R. J. Mooney, "Bottom-up learning of Markov logic network structure," in *Proc. ICML-07*, ser. ACM Int. Conf. Proceeding Series, Z. Ghahramani, Ed., vol. 227. ACM, 2007, pp. 625–632.

[74] M. Thielscher, "A general game description language for incomplete information games." in *AAAI*, vol. 10. Citeseer, 2010, pp. 994–999.

[75] A. M. Smith, M. J. Nelson, and M. Mateas, "Ludocore: A logical game engine for modeling videogames," in *Computational Intelligence and Games (CIG), 2010 IEEE Symposium on*. IEEE, 2010, pp. 91–98.

[76] J. Wang and P. Domingos, "Hybrid markov logic networks," in *Proc. AAAI-08 - Volume 2*. AAAI Press, 2008, pp. 1106–1111.

[77] M. Walker and J. Wooders, "Minimax play at Wimbledon," *American Economic Review*, vol. 91, no. 5, pp. 1521–1538, December 2001.

[78] T. Williams and J. Underwood, *The science of hitting*. Simon & Schuster, 1986.

[79] P.-A. Chiappori, S. Levitt, and T. Groseclose, "Testing mixed-strategy equilibria when players are heterogeneous: The case of penalty kicks in soccer," *The American Economic Review*, vol. 92, no. 4, pp. 1138–1151, 2002.

[80] R. Axelrod and W. D. Hamilton, "The evolution of cooperation," *Science*, vol. 211, no. 4489, pp. 1390–1396, 1981.

[81] J. R. Quinlan and R. M. Cameron-Jones, "FOIL: A midterm report," in *ECML-93 Proceedings, Vienna, Austria*, ser. Lecture Notes in Computer Science, P. Brazdil, Ed., vol. 667. Springer, 1993, pp. 3–20.

[82] H. Blockeel and L. De Raedt, "Top-down induction of first-order logical decision trees," *Artificial intelligence*, vol. 101, no. 1, pp. 285–297, 1998.

[83] J. R. Quinlan, *C4.5: Programs for Machine Learning*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1993.

[84] P. Domingos and D. Lowd, *Markov Logic: An Interface Layer for Artificial Intelligence*, 1st ed. Morgan and Claypool Publishers, 2009.

[85] A. Kimmig, S. Bach, M. Broecheler, B. Huang, and L. Getoor, "A short introduction to probabilistic soft logic," in *Proc. NIPS Workshop on Probabilistic Programming: Foundations and Applications*, 2012, pp. 1–4.

Marco Lippi received the M.S. Laurea degree (cum laude) in Information Engineering and the Ph.D. in Computer and Automation Engineering from the University of Florence in 2006 and 2010, respectively. Currently, he is a post-doc fellow at the Department of Informatics – Science and Engineering, at the University of Bologna. He previously was a post-doc fellow at the University of Siena, and a visiting scholar at the Laboratoire d'Informatique Paris 6, Université Pierre et Marie Curie, Paris. His work focuses on machine learning and artificial intelligence, with applications to the fields of bioinformatics, time-series forecasting, computer vision and argumentation mining. He has been and currently is a member of the program commitee in several international conferences and a reviewer for many international journals. In 2012 he was awarded the "E. Caianiello" prize for the best Italian Ph.D. thesis in the field of neural networks.