Weighted Logic

Alan Fern

February 8, 2010

1 Problems with Inconsistency and Monotonicity

While logic provides an appealing formalism for knowledge representation and reasoning, it is not without drawbacks. There are many aspects of human reasoning that are not adequately handled by pure logic. (Of course there are also many aspects that we would rather avoid capturing.)

For example, suppose a human is told that "all aliens are blue". Later, however, the human also learns the fact that "aliens of type Klingon are not blue". While this new fact is contradictory to the original statement, a human would have no difficulty resolving the conflict. Most humans would treat aliens of type Klingon as a special case, and otherwise all other aliens will be believed to be blue.

Now consider how an automated reasoning system would handle this same situation. Suppose that the KB already has knowledge that no alien can be both blue and green, i.e.

$$\forall x \;\; alien(x) \Rightarrow (blue(x) \Leftrightarrow \neg green(x))$$

. We now give the system the formula,

$$\forall x \;\; alien(x) \Rightarrow blue(x)$$

and then give the system the formulas,

$$\begin{aligned} \forall x \;\; klingon(x) &\Rightarrow alien(x) \\ \forall x \;\; klingon(x) \wedge alien(x) &\Rightarrow green(x) \\ klingon(Orf) \end{aligned}$$

The result is a set of inconsistent formulas, which is very bad news for a logical reasoning systems as it will now all formulas. The reasoning system has become completely useless with the addition of a single inconsistent statement.

The above behavior is known as the monotonicity property of logical systems. In particular, we have that whenever we add formulas to a KB all previously entailed formulas will still be entailed. That is,

$$\forall KB, \phi, \alpha \;\; If \;\; KB \models \alpha \;\; then \;\; KB \wedge \phi \models \alpha$$

In other words we have no way of giving a logic-based reasoning system a statement that will make it stop believing in a fact that is currently entailed. The only way to correct the system is to actually dig into the knowledge base and perform "brain surgery". In the above example, we change the knowledge base by replacing the first rule with,

$$\forall x\, (alien(x) \wedge \neg klingon(x)) \Rightarrow blue(x)$$

but this is not a very elegant solution, as it could lead to very complicated knowledge bases, as we would need multiple versions of each formula, one for each possible exception. The fundamental problem above is that logical systems completely break in the face of inconsistency, yet in most real-world domains they will likely encounter contradictory bits of knowledge.

Intuitively the above problem could be avoided if the system could somehow treat the original statement about aliens as a rule-of-thumb or default rule. That is, by default we believe an alien is blue, unless some other bit of knowledge tells us otherwise. However, the semantics of pure logic do not explicitly allow us to incorporate such knowledge. AI researchers have studied such dilemmas for over 40 years now, offering many different approaches to dealing with inconsistency. One formalism is known as non-monotonic reasoning, where defaults are explicitly represented and reasoned about using non-monotonic rules of inference based on well-defined semantics.

Another approach to dealing with inconsistency is to use probabilistic knowledge representations. Here there is no notion of logical inconsistency, rather "contradictory" knowledge will interact to assign probabilities to potential queries. In your introductory AI course you learned about Bayesian networks, which are one such representation for propositional knowledge. These models are the analog of propositional logic and have some of the same limitations in expressive power with respect to representing objects and relations. These models have been extended in various ways to the first-order, or relational, case and this is a rapidly developing line of research. While we will not talk about these model in this course, I will likely be giving a lecture in this quarter's Bayes Net class on the topic. Further the weighted logic representation that we describe in this class for handling uncertainty, has immediate extensions to the probabilistic cast, which we will allude to at the appropriate time.

We now describe a simple approach to extending a logic to handle inconsistency and non-monotonic inference patterns. The idea is to attach a weight to each logical formulas, which is interpreted as the "cost" of violating the formula. Below we introduce this idea and discuss basic approaches to the reasoning and learning problems.

2 Weighted Propositional Logic

We first consider the propositional case. Later we will extend to the first-order case via propositionalization.

A weighted propositional knowledge base (WPKB) is a set of weighted formulas,

$$\{\langle \phi_1, w_1 \rangle, \langle \phi_2, w_2 \rangle, \ldots, \langle \phi_n, w_n \rangle\}$$

where each ϕ_j is a propositional formula and each weight w_j is an integer. We say that a WPKB is in clausal form if each formula is a single clause.

The weight associated with a formula is interpreted as the cost of violating the formula, or equivalently the degree of preference for satisfying the formula. In this sense, larger weights should be associated with formulas that we are more certain of.

Given a propositional model M (i.e. a truth assignment to the propositions) and a WPKB $KB = \{\langle \phi_i, w_i \rangle\}$, we define the "weight of a model wrt KB" as the sum of the weights associated with any formula that is satisfied in the model. More formally,

$$WEIGHT(M, KB) = \sum_i w_i \cdot I[\phi_i \text{ true in } M]$$

where I is the indicator function defined such that $I[true] = 1$ and $I[false] = 0$. Given a WPKB we say that a model is a "maximal model" iff there is no other model that achieves a larger weight. Note that a particular WPKB can have multiple maximal models. (Try to construct some examples on your own.)

A number of researchers have studied various ways of defining entailment for WPKBs along with corresponding inference procedures. For example, one approach is to say that $KB \models \alpha$ iff α is true in all maximal models of KB. In this course, we will not focus on a particular notion of entailment. Instead, below we will focus on the inference problem of finding a minimum-cost model, which is useful across a wide array of applications.

3 Weighted Maximum Satisfiability

Given a clausal WPKB the problem of finding a maximal model is often called the "weighted maximum satisfiability" problem, which we will abbreviate as MAX-SAT. This problem is a generalization of the satisfiability problem, or SAT, which asks for a model that satisfies all of the formulas. Rather MAX-SAT asks for a model that satisfies a set of formulas of largest possible weight. Thus, unlike SAT, MAX-SAT always returns a model, even when the clauses are inconsistent with one another. Consider the following example, where we are given the WPKB,

$$\begin{aligned}
\langle JonIsRepublican \Rightarrow \neg JonIsPacifist, 10\rangle \\
\langle JonIsQuaker \Rightarrow JonIsPacifist, 20\rangle \\
\langle JonIsQuaker, 1000\rangle \\
\langle JonIsRepublican, 1000\rangle
\end{aligned}$$

The first two weighted clauses state preferences for believing that Jon is a pacifist based on whether or not he is a republican or quaker. The third and fourth clauses assert facts about Jon. Since the weight on these clauses is so large compared to the first two, the clauses are forced to be true in any maximal model. Notice that since Jon cannot be both a pacifist and "not a pacifist", the clauses in the WPKB have no satisfying assignment. However, the MAX-SAT solution will find the model that maximizes the stated preferences,

$$\begin{aligned}
JonIsRepublican &= true \\
JonIsQuaker &= true \\
JonIsPacifist &= true
\end{aligned}$$

which has a weight of 2020. Here the WPKB has a stronger preference for believing that quakers are pacifists, compared to believing that republicans are not pacifists and hence the maximal model asserts that Jon is a pacifist.

This toy example shows the potential usefulness of being able to assert rules-of-thumb such as the first two clauses, even if they may not always be true, or consistent with one another. Humans are often able to write down many such rules for a particular domain, and weighted logic provides a simple formalism for encoding such knowledge. Humans, however, are not so good at assigning weights to the various formulas, in order to accurately reflect preferences. We will ignore this issue for now, and come back to it later when we discuss an approach to learning weights from training data.

4 Solving MAX-SAT

In general MAX-SAT is a hard problem. Notice that any instance of SAT can be encoded as a MAX-SAT problem (this will be a homework problem), so it is at least NP-Hard. Also MAX-SAT remains hard, even for special cases where SAT becomes efficiently solvable. For example, SAT can be solved in polynomial time when the input clauses are all Horn. However, MAX-SAT is still NP-Hard for Horn theories. This means that in the worst case it is likely impossible to do better than simply enumerate all possible models and returning the maximal one.

Nevertheless, many algorithms have been developed for exactly and approximately solving MAX-SAT. (A google search will attest to this.) Here we will review a simple approximate solution algorithm called MaxWalkSat that is based on the WalkSAT algorithm for satisfiability. This algorithm often works quite well and code is freely available on the web at,

http://www.cs.washington.edu/homes/kautz/walksat/Maxwalksat20.tgz

The algorithm is describe in detail in the following paper,

http://www.cs.washington.edu/homes/kautz/papers/maxsatDIMACSfinal.ps

Here I outline the main steps:

1) The algorithm initializes the search to a random truth assignment.

2) An unsatisfied clause c is selected at random. Note that since the clause is unsatisfied, we know that each of its literals is unsatisfied by the current model. Hence flipping the truth value of any proposition appearing in the clause will cause the clause to be satisfied. However, this flip may cause other clauses to become unsatisfied.

3) With probability p we select a random literal in c and flip the value of its proposition, else with probability (1-p) we flip the proposition in c that minimizes the weight of clauses that would become unsatisfied after the flip.

4) Loop back to 2 until either a maximum number of iterations is reached or a weight bound is exceeded.

In step three, the probability p is known as the "noise parameter" and is simply the probability that a random non-greedy step is selected on each search step. The idea of occasionally taking non-greedy steps is intended to help prevent the algorithm from staying in local maxima. In addition, the MaxWalkSat algorithm allows for a specified number of random restarts, which further helps to deal with local maxima.

5 Converting to Clausal Form

MAX-SAT is typically formulated for weighted clausal theories rather than for arbitrary WPKBs. In particular, most software and algorithms for MAX-SAT only accept clausal theories as input. Thus, it is important to know how to convert an arbitrary WPKB to clausal form.

Our goal here is to take as input a WPKB KB and return a WPKB CLAUSAL(KB) that is in clausal form and whose maximal models correspond to maximal models of KB. A naive approach to computing CLAUSAL(KB) is to convert each formula in KB to a logically equivalent set of clauses and then assign the weight of each clause to be the weight of the original formula. It turns out that this conversion can produce a very different WPKB with very different maximal models. Make sure that you understand why.

It turns out that there is a relatively simple procedure for computing CLAUSAL(KB) assuming that we are willing to introduce new dummy propositions. Given $KB = \{\langle\phi_1, w_1\rangle, \langle\phi_2, w_2\rangle, \ldots, \langle\phi_n, w_n\rangle\}$, we introduce a new propositional symbol P_i for each ϕ_i in KB. Also let $w+$ be 1 plus the sum of all positive weights in KB and $CNF(\phi)$ denote a CNF formula that is logically equivalent to ϕ. We construct CLAUSAL(KB) by adding for each ϕ_i the following weighted clauses,

$$\langle P_i, w_i\rangle, \langle c_1, w+\rangle, \langle c_2, w+\rangle, \ldots, \langle c_m, w+\rangle$$

where $\{c_1, ..., c_m\}$ is the set of clauses in $CNF(P_i \Leftrightarrow \phi_i)$. That is we add the proposition P_i with the weight of w_i, and then add all of the clauses in the CNF of the double implication $P_i \Leftrightarrow \phi_i$, with weight $w+$. This conversion provides a correspondence between maximal models of the resulting WPKB and the original. In particular, let M be any maximal model of CLAUSAL(KB), and let M' be identical to M only it does not include truth assignments for $P_1, \ldots, P_n$. We have then that M' is a maximal model of KB. Likewise for any maximal model M of KB, there is a corresponding maximal model of CLAUSAL(KB) that is identical to M ignoring $P_1, \ldots, P_n$.

You will be asked to give a proof of the above correspondence in your homework. A rough sketch is as follows, for which you will need to fill in the details. First, note that the $w+$ weights act as hard constraints, forcing the each formula $P_i \Leftrightarrow \phi_i$ to be satisfied in any maximal model of CLAUSAL(KB). This means that P_i will be true in a maximal model of CLAUSAL(KB) exactly when ϕ_i is also satisfied in the model. This can be used to yield the desired relationship between KB and CLAUSAL(KB).

One interesting aspect of this conversion is that it produces a WPKB such that the only non-hard constraints are those involving unit propositions. An unfortunate aspect of this conversion is that it can dramatically increase the number of propositional symbols, as it adds a new symbol for each formula in the original WPKB.

6 Weight Learning

As mentioned above, humans are often quite good at providing rules of thumb in the form of logical rules or formulas. But they are not so good at assigning appropriate weights to formulas, representing the relative preferences. Here we will describe an approach to learning appropriate weights given a set of formulas and training examples.

In order to define our learning task we will first introduce the concept of "maximal model completion". Let KB be a WPKB over the set of propositions $\{p_1, \ldots, p_n\}$, and P' be a partial truth assignment, which assigns a truth value to a subset of the propositions. A "maximal model completion" relative to KB and P' is a model that is consistent with P' and has maximum weight compared to all other such models. We will denote the maximal model completion as $MAX - SAT(KB|P')$. When P' is empty then the result is simply a MAX-SAT solution. If there are multiple maximal model completions, then we assume that $MAX - SAT(KB|P')$ returns one of them according to a lexicographical ordering. As an example, consider the KB

$$\langle p \Rightarrow q, 10\rangle$$
$$\langle u \Rightarrow \neg q, 5\rangle$$

and the partial truth assignment,

$$P' = \{p = true, u = true\}$$

In this case, we have

$$\text{MAX-SAT}(KB|P') = \{p = true, u = true, q = true\}$$

which has a weight of 10. Note, that for this example MAX-SAT solutions have a weight of 15, e.g. the model where all propositions are false. This shows that maximal model completions can have lower weight than a MAX-SAT solution, which makes sense since a maximal model completion must satisfy more constraints.

Note that it is straightforward to use a MAX-SAT solver to compute MAX-SAT($KB|P'$). One can simply augment KB with literals that have very large weights (essentially infinite) that force the truth values specified in P'. Using the above example, we could form a new WPKB KB',

$$\begin{aligned}\langle p \Rightarrow q, 10\rangle\\ \langle u \Rightarrow \neg q, 5\rangle\\ \langle p, 16\rangle\\ \langle u, 16\rangle\end{aligned}$$

for which all MAX-SAT solutions must set $p = true$ and $u = true$ and hence will correspond to possible solutions to MAX-SAT($KB|P'$). (Make sure you understand why this works.)

We now define our learning problem formulation. We will divide the set of propositions into an input set $X = \{x_1, \ldots, x_n\}$ and an output set $Y = \{y_1, \ldots, y_m\}$. Intuitively, we would like to learn a set of weighted formulas KB over X and Y such that given a truth assignment X' for the input propositions, MAX-SAT(KB, X') assigns the "correct" values to the output propositions.

As an example, consider the problem of selecting agent actions in a multi-agent real-time strategy (RTS) game. Here, X might correspond to a set of propositions that describe the current state of the game (giving e.g. the position of all agents, the health, etc.), and Y might correspond to a set of propositions that assign actions to each agent (e.g. "agent1 should attack enemy1"). Given an assignment X' to the state propositions, we would like a WPKB KB such MAX-SAT(KB, X') returns an assignment Y' to the set of action propositions that lead to good performance.

The input to our learning problem is a set of propositional formulas $\{phi_1, \ldots, phi_v\}$ over X and Y, and a set of training examples, $\{\langle X_1, Y_1\rangle, \ldots, \langle X_N, Y_N\rangle\}$ where each X_i is an input with desired output Y_i. Our goal is to learn a set of weights $\{w_1, ..., w_v\}$ giving us a WPKB $KB = \{\langle phi_1, w_1\rangle, \ldots, \langle phi_v, w_v\rangle\}$ such that, for each training example, MAX-SAT(KB, X_i) is consistent with Y_i. In our RTS example, we might obtain the $\langle X_i, Y_i\rangle$ pairs by observing a human expert play the game, or alternatively via "reinforcement learning" (reinforcement learning is an area of AI taught in CS533).

To learn weights we can use the generalized perceptron algorithm (Collins, 2002). Below we will introduce a feature function f_i for each formula ϕ_i. The value of f_i given a model (X, Y) is,

$$\begin{aligned} f_i(X, Y) &= 1, \quad if\ phi_i\ is\ true\ in\ (X, Y) \\ & \ 0, \quad otherwise \end{aligned}$$

You should verify that $WEIGHT((X, Y), KB) = \sum_i w_i * f_i(X, Y)$.

The generalized perceptron algorithm iterates through training examples $\langle X_i, Y_i\rangle$ and adjusts weights until either all training examples are correctly predicted, or a maximum number of iterations is met. For each example, the algorithm computes the

$$\langle X_i, Y'\rangle = \text{MAX-SAT}(KB, X_i)$$

using the current weights to define KB. Here Y' is the current prediction that the KB makes for the input X_i. If the prediction is correct, then we do not change the weights. Otherwise we change the weights so that the correct model $\langle X_i, Y_i\rangle$ gets more weight and the incorrect model $\langle X_i, Y'\rangle$ gets less weight. By repeating this process the hope is that the correct models will eventually have higher weight than all other alternatives, making $\langle X_i, Y_i\rangle = \text{MAX-SAT}(KB, X_i)$ as desired. The pseudo-code is given below:

```
for $i = 1$ to $v$ do
  $w_i = 0$
end for
repeat
  for $i = 1$ to $n$ do
    $KB = \{\langle\phi_1, w_1\rangle, \langle\phi_2, w_2\rangle, \ldots, \langle\phi_n, w_n\rangle\}$
    $\langle X_i, Y'\rangle = \text{MAX-SAT}(KB|X_i)$ {compute the best $Y^{'}$ according to the current weights}
    if $Y' \neq Y_i$ then
      for $j = 1$ to $v$ do
        $w_j = w_j + \alpha \cdot [f_j(X_i, Y_i) - f_j(X_i, Y')]$
      end for
    end if
  end for
until some number of iterations
```

The critical step is the weight update,

$$w_j = w_j + \alpha \cdot [f_j(X_i, Y_i) - f_j(X_i, Y')]$$

that occurs after an incorrect prediction. If the formula ϕ_i is true in the correct model (X_i, Y_i) but false in the incorrect model (X_i, Y') then the update will increase the weight for ϕ_i which will increase the weight on the correct model as desired. If ϕ_i is false in the correct model and true in the incorrect model then we decrease the weight for ϕ_i which will decrease the weight on the incorrect model as desired. Finally if ϕ_i is the same in both the incorrect and correct model we do not adjust the weight. The single parameter α is the "learning rate". Often α is set to 1.

While it is not obvious, it can be shown that if there exists a set of weights such that $\langle X_i, Y_i\rangle =$ MAX-SAT(KB, X_i) for all training examples, then the above algorithm with $\alpha = 1$ will eventually find such a set of weights.

The primary computational bottleneck of the above algorithm is the MAX-SAT calculation for each training example in each iteration. As we noted earlier, MAX-SAT is computationally hard in the worst case. One solution to dealing with this problem is to use fast approximate MAX-SAT solvers.

The above formulation assumed that we were given an initial set of formulas with no weights. Clearly it would be desirable to develop algorithms that can also induce new formulas. There are a number of heuristic approaches that could be tried for this purpose, though we will not discuss them in this course.

As an important side note, the above algorithm is straightforward to generalize to any type of weighted constraints. That is, instead of using propositional formulas as constraints on models, we can consider arbitrary constraint languages on arbitrary structures. For any such language we can attach weights to those constraints and provide an algorithm for the corresponding MAX-SAT problem—i.e. an algorithm that finds a structure that maximizes the weight of satisfied constraints. With the MAX-SAT solver in hand, the above algorithm can be applied directly.

7 Weighted First-Order Logic: A Template-based Approach

You have now seen approaches to inference and learning for propositional weighted logic. The key idea was to use WPKBs to assign a weights to models that represent a preference relationship over models. Here we would like to extend that idea to first-order models. This will provide us with the ability to compactly specify preference knowledge that generalizes over objects.

Let begin with an example of what we might like to represent, but can't easily represent with WPKBs. Recall our "pacifist example" from above. In that case we used the propositions JonIsPacifist, JonIsQuaker, and JonIsRepublican. If we wanted to have the same rules of thumb for another individual, e.g. Nixon, we would need to explicitly add three additional propositions, and add the corresponding propositional formulas. Rather, it would be more convenient if we could simply write down weighted formula template that could be instantiated for any individual. For example,

$$\begin{aligned} \langle Republican(x) \Rightarrow \neg Pacifist(x), 10\rangle \\ \langle Quaker(x) \Rightarrow Pacifist(x), 20\rangle \end{aligned}$$

are weighted formula templates where x is a free variable that serves as the template parameter. We can instantiate these templates for any number of individuals, yielding a set of ground or propositional weighted formulas. For example, if we are interested in Jon and Nixon, we would get the WPKB,

$$\begin{aligned}\langle Republican(x) \Rightarrow \neg Pacifist(x), 10\rangle \\ \langle Quaker(x) \Rightarrow Pacifist(x), 20\rangle \\ \langle Republican(Nixon) \Rightarrow \neg Pacifist(Nixon), 10\rangle \\ \langle Quaker(Nixon) \Rightarrow Pacifist(Nixon), 20\rangle\end{aligned}$$

which can be used to reason about Jon and Nixon, e.g. via a MAX-SAT solver. Intuitively the weight associated with each template can be viewed as a cost that must be paid for each instantiation of the template that is not satisfied. Thus, if a particular model violates both ground instances of the first template (one for Jon and one for Nixon) then the weight of that model will be $20 = 2 \times 10$ less than if the instances were satisfied.

7.1 Fixed-Domain Semantics

Before we define the syntax of weighted templates and how they are used to assign weights to first-order models, we first introduce the idea of "fixed-domain semantics", which will serve as our formalism for upgrading from propositional to first-order models.

Recall that for a given set of propositions there are a finite number of propositional models. This made it straightforward to define the notion of model weight and maximal model. The general first-order case is not so simple as there are an infinite number of first-order models, some of which have an infinite number of domain objects. This makes defining the weight of a model with respect to a set of weighted templates problematic. For example, we must worry about the possibility of models with infinite weight and the possibility of non-existence of maximal models.

We will avoid the above difficulties by stepping back from the full generality of first-order logic and restrict our attention to finite sets of objects. We will refer to this restriction as the "finite-domain semantics". The idea is that at any moment in time we will only be concerned with a finite set of "objects of interest" and will only consider models over those objects. Since there are only a finite number of such models (assuming a fixed set of predicates), we will see below that it is relatively straightforward to define the semantics.

Indeed, the finite-domain semantics typically correspond well with application goals. For example, in the RTS domain described above, at any point in the game there will only be a finite number of objects (e.g. friendly/enemy units, buildings, gold mines, etc). Our system need only be concerned with models involving those objects. e.g. to select the "MAX-SAT" actions for the friendly troops. However, over time the set of game objects will change and we would like our knowledge base to naturally generalize to these new situations. The ability to generalize in this way is the primary advantage of using a first-order rather than propositional formalism.

More formally, given a finite set of constants C (representing our objects) and a set of predicate symbols P, we define $MODELS(P, C)$ as the set of all first-order models with domain C (i.e. an object for every constant) involving only predicates in P. Note that we can view MODELS(P, C) as specifying a set of propositional models where the propositions correspond to all possible ground atoms constructed from P and C. For example, consider $C = \{Jon, Nixon\}$ and $P = \{Republican, Pacifist, Quaker\}$. Here each model in MODELS(P, C) will specify a truth assignment to the following propositions,

$$\{Republican(Jon), Republican(Nixon), Pacifist(Jon), Pacifist(Nixon), Quaker(Jon), Quaker(Nixon)\}$$

there are a total of 2^6 such models. In general, there will be on the order of $|C|^v$ propositions in MODELS(P, C) where v is the maximum arity of any predicate in P. This means that in general MODELS(P, C) will contain on the order of $2^{|C|^v}$ models.

The idea of defining semantics relative to a given finite domain of objects is quite common. For example, as we will see later in the course, this idea is the essence behind most work on probabilistic relational modeling.

7.2 Weighted Formula Templates

A "formula template" is simply a first-order logic formula with free variables. Free variables in a formula template should not be treated as universally quantified, as in first-order logic. Rather, we will think of the variables as template parameters. We will use these parameters in order to compile a template to a set of ground formulas relative to a set of constants. Given a set of constants C and a formula template T we define the "compilation of T relative to C", denoted by COMPILE$(T|C)$, as the set of ground formulas that results from the following steps:

Let $C = c_1, ..., c_n$

1) For each universally quantified subformula $(\forall x\ \phi(x))$ in T replace it by the conjunction $(\phi(c_1) \wedge \phi(c_2) \wedge \cdots \wedge \phi(c_n))$. This yields a new formula template T'.

2) Starting from T', replace each existentially quantified subformula $(\exists x\ \phi(x))$ by the disjunction $(\phi(c_1) \vee \phi(c_2) \vee \cdots \vee \phi(c_n))$. This yields a new formula template T''.

3) Return the set of all ground formulas that are a result of any way of substituting the free variables of T'' with constants from C. If there are m free variables in T'', then we will return $|C|^m$ formulas.

Essentially the above procedure assumes that the only objects in the world are those that will be denoted by the constants. The procedure then replaces universal quantification by explicit conjunctions, and existential quantification with explicit disjunctions. Consider the following example.

$$\begin{aligned} C &= \{A, B\} \\ T &= [\exists z \quad (R(x,z) \wedge R(z,y))] \Rightarrow (\forall w \quad Q(x,y,w)) \end{aligned}$$

where T is a formula template with free variables x and y (note that z is existentially quantified and hence is not free). The steps for computing COMPILE$(S|C)$ are as follows:

1) $T' = [\exists z\ (R(x,z) \wedge R(z,y))] \Rightarrow [Q(x,y,A) \wedge Q(x,y,B)]$

2) $T'' = [(R(x,A) \wedge R(A,y)) \vee (R(x,B) \wedge R(B,y)] \Rightarrow [Q(x,y,A) \wedge Q(x,y,B)]$

3) $\{\ SUBST(S'', x/A, y/A)$
$SUBST(S'', x/A, y/B)$
$SUBST(S'', x/B, y/A)$
$SUBST(S'', x/B, y/B)\ \}$

So we see that COMPILE($T|C$) results in a set of 4 ground formulas. Recall that we can think of first-order ground formulas as equivalent to propositional formulas where the propositions are the ones in MODELS(P, C) where P is the set of predicates in T. For example, the propositions in the above formulas include the following set of $2^2 + 3^2 = 13$ ground atoms.

$$\{R(A, A), R(A, B), R(B, A), R(B, B), Q(A, A, A), Q(A, A, B), Q(A, B, A), \ldots, Q(B, B, B)\}$$

Now that we have defined formula templates and compilation, it is straightforward to define weighted formula templates. A "weighted formula template" is simply a pair of a formula template and an integer weight. Given a weighted formula template $\langle T, w\rangle$ and a set of constants C, we define compilation as follows,

$$\text{COMPILE}(< T, w >, C) = \{\langle \phi, w\rangle | \phi \in \text{COMPILE}(T, C)\}$$

That is, COMPILE($\langle T, w\rangle, C$) is a WPKB whose formulas are those in COMPILE(S, C) and all have the same weight w.

A "weighted template knowledge base" (WTKB) KB is a set of weighted formula templates. The compilation of a WTKB KB relative to C is the union of weighted formulas that result from compiling any template in KB relative to C. For example, consider the following WTKB KB,

$$\begin{aligned}
\langle Republican(x) \Rightarrow \neg Pacifist(x), 10\rangle \\
\langle Quaker(x) \Rightarrow Pacifist(x), 20\rangle \\
\langle (Friend(x, y) \wedge Quaker(x)) \Rightarrow Quaker(y), 30\rangle \\
\\
\langle Quaker(Jon), 1000\rangle \\
\langle Republican(Jon), 1000\rangle \\
\langle Republican(Nixon), 1000\rangle \\
\langle Friend(Nixon, Jon), 1000\rangle \\
\langle Friend(Jon, Nixon), 1000\rangle
\end{aligned}$$

and constants $C = \{Jon, Nixon\}$. This is similar to our previous pacifist examples, only now we have a new preference that the friend of a quaker is a quaker. The compilation of KB denoted by

COMPILE(KB, C) yields the following WPKB.

$$\begin{aligned}
\langle Republican(Jon) \Rightarrow \neg Pacifist(Jon), 10\rangle \\
\langle Republican(Nixon) \Rightarrow \neg Pacifist(Nixon), 10\rangle \\
\\
\langle Quaker(Jon) \Rightarrow Pacifist(Jon), 20\rangle \\
\langle Quaker(Nixon) \Rightarrow Pacifist(Nixon), 20\rangle \\
\\
\langle (Friend(Jon, Nixon) \wedge Quaker(Jon)) \Rightarrow Quaker(Nixon), 30\rangle \\
\langle (Friend(Jon, Jon) \wedge Quaker(Jon)) \Rightarrow Quaker(Jon), 30\rangle \\
\langle (Friend(Nixon, Jon) \wedge Quaker(Nixon)) \Rightarrow Quaker(Jon), 30\rangle \\
\langle (Friend(Nixon, Nixon) \wedge Quaker(Nixon)) \Rightarrow Quaker(Nixon), 30\rangle \\
\\
\langle Quaker(Jon), 1000\rangle \\
\langle Republican(Jon), 1000\rangle \\
\langle Republican(Nixon), 1000\rangle \\
\langle Friend(Nixon, Jon), 1000\rangle \\
\langle Friend(Jon, Nixon), 1000\rangle
\end{aligned}$$

We can now use any propositional MAX-SAT solver to reason about the individuals in C. In general, given a WPKB KB and a set of constants C we define the "maximal models relative to C" to be the maximal models of COMPILE(KB, C). This model is guaranteed to be in MODELS(P, C) as intended under our finite-domain semantics. It is conceptually straightforward to compute such a maximal model by creating COMPILE(KB, C) and running a propositional MAX-SAT solver.

Continuing with the above example then, we see that a maximal model or MAX-SAT solution of KB relative to C is,

$$\begin{aligned}
Friend(Jon, Nixon) &= true \\
Friend(Nixon, Jon) &= false \\
Friend(Jon, Jon) &= false \\
Friend(Nixon, Nixon) &= false \\
Republican(Jon) &= true \\
Quaker(Jon) &= true \\
Pacifist(Jon) &= true \\
Republican(Nixon) &= true \\
Quaker(Nixon) &= true \\
Pacifist(Nixon) &= true
\end{aligned}$$

where in the above solution, $Friend(Jon, Jon)$ and $Friend(Nixon, Nixon)$ could both be true too.

This agrees with the preferences indicated in the above rules. If Nixon was not a friend of Jon then the MAX-SAT solution would conclude that Nixon was not a pacifist since he is a republican.

But the fact that Jon is a friend of Nixon causes the MAX-SAT solution to conclude that Nixon is a quaker and hence also a pacifist. Note that if the weight on rule 3 were less than that on rule 1, then the MAX-SAT would conclude that Nixon was not a quaker and also not a pacifist. (You should verify this for yourself.)

We see that the use of weighted templates can allow for knowledge to be expressed very compactly. However, one practical concern with the general compilation approach is that it can result in very large sets of weighted ground formulas. In general, the number of ground formulas will be on the order of $|C|^v$, where v is the maximum number of free variables in any template. In practice, if an application is time critical, we can limit the number of free variables to help keep the number of ground formulas under control. There are also various “partial compilation” tricks that can sometimes be used that avoid creating the entire WPKB for COMPILE($KB|C$) while guaranteeing correct inference. We will not discuss such approaches in this course.

8 Learning the Weights of First-Order Templates

Suppose now that we are given a set of first-order templates, along with training data. We now describe an approach to learning the weights. It turns out that we can use an algorithm that is almost identical to the propositional case.

We define the learning problem as follows. The input is:

1) A set of first-order formula templates $\{\phi_1, \ldots, \phi_v\}$ over a set of input predicates P_x and output predicates P_y.

2) A set of training examples $\{\langle C_1, X_1, Y_1\rangle, \ldots, \langle C_N, X_N, Y_N\rangle\}$ where each C_i is a set of constants, and $\langle X_i, Y_i\rangle$ is a model in MODELS($C_i, P_x \cup P_y$). Here we think of X_i as listing truth assignments to atoms involving input predicates, and Y_i listing truth assignments for atoms involving output predicates.

The output is a set of weights $\{w_1, \ldots, w_v\}$ giving a WTKB $KB = \{\langle\phi_1, w_1\rangle, \ldots, \langle\phi_v, w_v\rangle\}$ such that, for each training example, MAX-SAT($KB_i|X_i$) is consistent with Y_i, where KB_i is the WPKB COMPILE($KB|C_i$). That is, our goal is to find a set of weights such that when the WTKB is compiled relative to C_i for each example we are able to compute the target Y_i facts using MAX-SAT. Note that the set of constants C_i need not be the same across examples. In our RTS example, this means that the training data can come from different situations involving a different sets of game entities.

The main difference between the learning problem here versus the propositional case is that here the X_i and Y_i vary in size across the examples. Rather in the propositional setting the X_i and Y_i were truth assignments over a fixed set of propositions. Nevertheless we can adapt the generalized perceptron algorithm to our new setting in a straightforward way.

To describe the learning algorithm we will redefine our previous notion of feature function. The feature function f_i for formula template ϕ_i assigns a positive integer to any given example (C, X, Y) as follows:

$$f_i(C, X, Y) = |\{\phi' truein(X, Y) | \phi' \in COMPILE(\phi_i|C)\}|$$

That is, $f_i(C, X, Y)$ is the count of how many formulas in $COMPILE(\phi_i|C)$ are true in the model (X, Y). Thus f_i will have a high value if the template ϕ_i is typically true in the example, and will have a small value if the template is frequently violated in the model. This definition of feature function has the property that,

$$WEIGHT((X, Y), COMPILE(KB|C)) = \sum_i w_i * f_i(C, X, Y)$$

that is the weight of the model (X, Y) with respect to the compiled knowledge base is given by the weighted sum of feature functions. Just as in the propositional case, we can now use the perceptron algorithm to adjust the weights in a direction that increases the weight of the target models and decreases the weight of incorrect models. The pseudo-code is below:

```
for $i = 1$ to $v$ do
  $w_i = 0$
end for
repeat
  for $i = 1$ to $n$ do
    $KB = \{\langle\phi_1, w_1\rangle, \langle\phi_2, w_2\rangle, \ldots, \langle\phi_n, w_n\rangle\}$
    $KB' = COMPILE(KB|C_i)$ {WPKB relative to $C_i$}
    $\langle X_i, Y'\rangle$ = MAX-SAT($KB'|X_i$) {compute the best $Y^{'}$ according to the current weights}
    if $Y' \neq Y_i$ then
      for $j = 1$ to $v$ do
        $w_j = w_j + \alpha \cdot [f_j(C_i, X_i, Y_i) - f_j(C_i, X_i, Y')]$
      end for
    end if
  end for
until some number of iterations
```

Again the critical step here is the weight update,

$$w_j = w_j + \alpha \cdot [f_j(C_i, X_i, Y_i) - f_j(C_i, X_i, Y')]$$

that occurs after an incorrect prediction. This update makes intuitive sense. If ϕ_i is satisfied more frequently in the correct target Y_i than in the incorrect prediction Y', then the weights are increased. Otherwise if ϕ_i is violated more often in Y_i than in Y', then the weights are decreased.

Despite the simplicity of this update rule, we again have the theoretical guarantee that the algorithm will converge to weights that correctly classify all of the training data if such weights exists.

Again here we have assumed that we are given formula templates. One can also consider learning templates based on the training data. This is akin to the structure learning problem in graphical models, or feature discovery in more traditional machine learning. FOIL-like techniques have been used for this purpose with good results.