Inference with Relational Theories over Infinite Domains

Nicholas L. Cassimatis and **Arthi Murugesan** and **Perrin G. Bignoli**
Rensselaer Polytechnic Institute
Troy, NY 12180

Abstract

Many important tasks can be cast as weighted relational satisfiability problems. Propositionalizing relational theories and making inferences with them using SAT algorithms has proven effective in many cases. However, these approaches require that all objects in a domain be known in advance. Many domains, from language understanding to machine vision, involve reasoning about objects that are not known beforehand. Theories with unknown objects can require models with infinite objects in their domain and thus lead to propositionalized SAT theories that existing algorithms cannot deal with. To address these problems, we characterize a class of relational generative weighted satisfiability theories (GenSAT) over potentially infinite domains and propose an algorithm, GenDPLL, for finding models of these theories. We introduce the notion of a relevant model and an increasing cost theory to identify conditions under which GenDPLL is complete, even when a theory has infinite models.

Introduction

Propositionalizing first-order theories and making inferences using satisfiability algorithms has proven an effective means of solving problems in many domains (Domingos and Richardson, 2006; Jackson, 2000; Singla and Domingos, 2006). These methods generally assume that all the objects in a domain are known in advance and begin by translating first-order clauses into propositional clauses. However, many problems involve objects that are not known before inference begins. For example, visual inference must deal with objects that are initially unknown because they are occluded and parsing a sentence using context-free grammars involves reasoning about constraints on (potentially infinitely many) phrases that were not known in advance of parsing.

Unknown objects pose two difficulties for using satisfiability methods to reason over relational theories. First, theories of finite size that express relations over unknown objects often require infinite models. For example, the formula, $Mammal(a) \wedge \forall x(Mammal(x) \rightarrow Mammal(mother(x))$ (together with formulae stating that a mammal cannot be its own ancestor) require an infinite model because a's mother must also have a mother who must also have a mother, ad infinitum. Likewise, some context-free grammars with finite numbers of rules and terminals can generate an infinite number of sentences. Since an algorithm cannot enumerate an infinite model in finite time, we must find a way of finitely characterizing solutions to problems that have infinite models.

A second problem associated with unknown objects is that even if all models of a theory can be finitely characterized, there may nevertheless be infinitely many such models. Complete satisfiability algorithms (e.g., those based on DPLL (Davis et al., 1962)) over finite domains are guaranteed to halt because they perform exhaustive search through the space of possible models. Thus, developing model finding algorithms when there are infinitely many possible models poses additional difficulties over standard complete satisfiability algorithms.

Some approaches (Singla and Domingos, 2006) deal with the problem of large propositional translations by lazily grounding first-order clauses. However, their algorithm is not complete and their approach requires that all objects are known in advance. They thus do not deal with problems that have infinitely large and/or infinitely many models. Several approaches to probabilistic inference involve unknown objects and infinite domains. Infinite Markov Logic Networks (Domingos and Richardson, 2007) have been extended to infinite domains, but, to our knowledge, an algorithm has not been proposed for inference over these networks. Several other approaches (Kersting and De Raedt, 2001; Milch, Marthi, Sontag, Russell, and Ong, 2005; Muggleton, 1996) do not enable exact inference and/or involve graphical models that impose stronger restrictions on these networks (e.g. regarding the existence of directed cycles) than are imposed by satisfiability theories.

This paper describes an approach that addresses issues raised by unknown objects and enables in many cases inference over problems with infinite models. First, we propose the GenSAT language for expressing relational, weighted constraints over unknown objects. Even in many cases where GenSAT theories have infinite models, it is possible to identify finite partial models of interest to specific inference problems. We introduce the notion of a relevant model of a GenSAT theory to define such models. Next, we describe GenDPLL, an algorithm for finding models of GenSAT the-

ories. GenDPLL is a DPLL-like branch-and-bound algorithm that lazily posits new objects and instantiates clauses involving them. Finally, we prove that GenDPLL is guaranteed to find finite relevant models of certain classes of GenSAT theories with infinite models, which we call increasing cost models.

Generative Satisfiability Theories

We propose a language for generative satisfiability (GenSAT) theories that express constraints over relations among objects and licenses the "generation" of new objects during inference. The following simple example illustrates the main aspects of the language. It involves clauses that state that a telephone receiving a call and the striking of a bell both lead to a ringing sound and that a phone ringer is only on when the phone's battery is not empty.

$$BellStrike(?x, ?t) \rightarrow (10)Ring(?t), ?x$$
$$GetCall(?p, ?t) \wedge RingerOn(?p, ?t) \rightarrow (7)Ring(?t), ?p$$
$$RingerOn(?p, ?t) \rightarrow \neg EmptyBattery(?p, ?t)$$

The numbers "10" and "7" are the weights on the constraints represented by the clauses. Terms beginning with '?' are variables. Variables listed after the right-most literal are called posited variables and indicate the potential of an object to be generated. For example, ?x in the second clause indicates that if a ringing sound occurs, then a bell (unknown prior to inference) being stuck may have caused it. This will be made more precise below. Non-posited variables are implicitly quantified.

It is often the case that if an event occurs or a relation obtains, then it must have been caused. For example, in many planning problems, everything not true in the initial state must have been made true by the execution of an operator. This is often encoded with what we here call a mandatory support constraint stating that either one of the possible causes of an effect holds or the effect does not hold: $cause_1 \vee ... \vee cause_n \vee \neg effect$.

When unknown objects are potentially relevant, it is often not possible to create such a clause in advance because all possible causes are not known. For example, if the number of bells is not known in advance of reasoning, then the number of possible causes of a ring cannot be known.

It is thus impossible to explicitly state these mandatory support constraints in many GenSAT theories. We informally describe how a GenSAT theory deals with these constraints and in the next section provide more detail. In GenSAT, the possible supports of a literal are indicated by clauses with numbers after the arrow. These are called causal[1] clauses. The first two clauses above are causal. The weight associated with a causal clause, is the cost of a fully grounded instance of that constraint being violated. Thus, if two bells strike and do not cause ringing sounds, then a cost of 20 accrues. When a literal (e.g. $Ring(?t)$) occurs on the right hand side of a causal clause, it is called a *causal literal*. Causal literals, when true, must must be implied by at least one causal clause. Non-causal literals do not require support. Non-causal clauses, which we will call *logical clauses*, are always interpreted to be hard constraints with infinite cost. They are syntactically distinguished from causal clauses in not having an explicit weight.

If not all possible supports for a literal are encoded by a theory, then one can neutralize mandatory causation with a clause for an unknown support. For example, mandatory support for $Ring$ literals can be neutralized with the clause $UnknownCauseOfRinging(?t) \rightarrow (\infty)Ring(?t)$. This encodes the fact that ringing can have a support that is not listed among those enumerated in the theory.

More formally, a GenSAT theory is a set of causal and logical clauses. Causal clauses are of the form:

$$C_1 \wedge ... \wedge C_l \rightarrow (w), E_1, ..., E_m, ?p_1, ..., ?p_n$$

where w is a positive scalar or ∞ and the C's and E's are each either literals or negations of literals. Literals are of the form: $P(a_1, , a_n)$, where each argument is a term. Terms that are not variables are called objects. Logical clauses are of the form:

$$A_1 \wedge ... \wedge A_l \rightarrow B_1 \wedge ... \wedge B_m, ?p_1, ..., ?p_n$$

where the A's and B's are each either literals or negations of literals. Logical clauses can have zero literals in the antecedent and one in the consequent, in which case they can be called facts. The fact $\rightarrow P$ can be abbreviated as P. Clauses with no posited variables can be abbreviated by omitting the comma.

Translation to Propositional Form

In this section, we specify a translation from a GenSAT theory into a (potentially infinite) set of propositional clauses. The translation facilities defining models of GenSAT theories and is important in characterizing a procedure we propose for making inferences over GenSAT theories.

The translation of a GenSAT theory is the union of: 1. A set of propositional clauses formed by grounding clauses in the theory and 2. A set of clauses that capture mandatory support constraints implied by the other clauses in the theory.

We illustrate translation with the GenSAT theory with the following four clauses:

$$Hammer(?h) \wedge Bell(?b) \wedge Hit(?h, ?b) \rightarrow (56)Ring(?b), ?b,$$
$$Hammer(h), Bell(b), Hit(h, b)$$

The translation of this theory grounds variables with objects occurring as arguments of literals in the theory and splits causal clauses into two clauses, one logical and one causal. Reasons for this split will be discussed below.

$$Hammer(h) \wedge Bell(b) \wedge Hit(h, b) \rightarrow (56)CauseRing(h, b)$$
$$CauseRing(h, b) \rightarrow Ring(b)$$
$$Hammer(b) \wedge Bell(h) \wedge Hit(b, h) \rightarrow (56)CauseRing(b, h)$$
$$CauseRing(b, h) \rightarrow Ring(h)$$

[1]Note the term "causal" here is used merely to indicate that some literals imply other literals via causal clauses. It is not the aim of this paper to describe a framework for causal reasoning.

$$Hit(h,b), Hammer(h), Bell(b)$$

Note that the second pair of clauses above are in some sense spurious since $Hammer(b)$ should intuitively be false given that $Bell(b)$ and that bells are not hammers. A clause indicating that hammers are not bells could prevent models with $Hammer(b)$ from being true[2].

The translation also includes clauses representing the constraint that ringing events must be caused by a hitting event:

$$\neg CauseRing(h,b) \rightarrow \neg Ring(b)$$
$$\neg CauseRing(b,h) \rightarrow \neg Ring(h)$$

This relatively simple expression of mandatory support constraints is enabled by splitting or "bifurcating" ground GenSAT clauses with the $CauseRing$ literals. This is the purpose of splitting causal literals in the translation.

To illustrate a theory over unknown objects, consider adding a clause to the effect that ringing bells generates sounds:

$$Ring(?b) \rightarrow GenerateSound(?b, ?s), ?s$$

Adding this clause to the GenSAT theory above would add the following two clauses to its translation:

$$Ring(b) \rightarrow GenerateSound(b, bSound)$$
$$Ring(h) \rightarrow GenerateSound(h, hSound)$$

These clauses include terms $bSound$ and $hSound$ which, informally, are the sound generated by the ringing of b and of h.

We now describe the translation more precisely in terms of a function that maps a GenSAT theory onto a set of grounded propositional clauses.

The definition of the grounding of a GenSAT theory depends on a skolem function s that uniquely maps posited variables to objects not in P. The purpose of this function is to produce objects that are inferred during inference and that thus do not occur in P. For example, in the theory above, $s(b) = bSound$.

The translation of a GenSAT theory P is the union of groundings of clauses in P to propositional clauses and the set of clauses representing mandatory support constraints for the theory.

$Translation(P,s) = Grounding(P,s) \cup MCClauses(Grounding(P,s))$

A clause is in $Grounding(P,s)$ if it can be generated by successive steps of grounding.

$$Grounding(P,s) = OneStep(P, Objects(P), s) \cup OneStep(P, Objects(Grounding(P,s)), s)$$

A clause is produced in a step of grounding if it can be produced by assigning to variables objects that are either generated by a skolem function or are already in the translation.

[2]In the future, a sorted version of GenSAT could be developed to prevent some spurious matches

$$OneStep(P,O,s) = U_c ClauseGrounding(c,O,s)$$

The grounding of a clause with respect to a set of objects is the set of clauses that can be formed by all the possible assignments of those objects to variables in the clause. Where c_l is a logical clause, $l \rightarrow r, ...p_i...$, with variables $v_1...v_n$, first appearing in the clause in that order,

$$ClauseGrounding(c_l, objects, s) = \{l_{a,s} \rightarrow r_{a,s} | a \in A\}$$

where $l_{a,s}$ and $r_{a,s}$, are the antecedent and consequent of c_l formed by substituting non-posited variables in c_l according to the assignment a and skolum function s applied to posited variables.

For example, the grounding of $A(?x) \rightarrow B(?y), ?x$, with respect to s and objects $\{a,b\}$) includes (only) the following two clauses: $A(s(a)) \rightarrow B(a)$ and $A(s(b)) \rightarrow B(b)$.

The grounding of causal clause, $c_s = l_s \rightarrow (w)r_s, ...p_i...$ is similar except it is "bifurcated", as was illustrated previously and is described presently:

$$ClauseGrounding(c_s, objects, s) = l_{s,a} \rightarrow (w)m, m \rightarrow r_{s,a}$$

where m, called an intermediate literal, occurs nowhere else in the translation and is unique to c_s and a.

For example, the grounding of $P(?x) \rightarrow (15)Q(?y), ?y$ with objects a, b is the following set of four clauses

$$\{P(a) \rightarrow (15)m_a, m_a \rightarrow Q(s(a)),$$
$$P(b) \rightarrow (15)m_b, m_b \rightarrow Q(s(b))\}$$

As mentioned above, the purpose of bifurcating a causal clause using an intermediate literal, m is that it enables the creation of clauses that express mandatory support constraints. A mandatory support constraint for a literal states that if none of the possible "supports" of l is true, then l is false. Specifically:

$MCClause(l) = (...\wedge\neg m_i\wedge...) \rightarrow \neg l$, where the m_i are the intermediate literals of causal clauses in the translation that have l in the consequent.

$MCClauses(S) = \cup_l MCConstraint(l)$, where l ranges over all the literals in S that occur in clauses ground from causal literals.

For example, if $Rain() \rightarrow (\infty)Wet(g)$ and $Sprinkler(g) \rightarrow (\infty)Wet(g)$ are the only two causal clauses in the grounding of P, then the following mandatory support clause appears in the translation, $\neg m_{rain} \wedge \neg m_{sprinkler} \rightarrow \neg Wet(g)$, where m_{rain} and $m_{sprnkler}$ are the intermediate literals formed in the grounding of the clauses in P.

Note also that any MaxSAT theory can be expressed as a GenSAT theory. MaxSAT theories are a conjunction of disjunctions (often called clauses) of the form $(t_1 \vee ... \vee t_n, w)$, where each t_i is a possibly negated propositional literal and w is a weight. MaxSAT theories can be straightforwardly translated into GenSAT theories by translating clauses into the GenSAT clauses with equivalent truth conditions: $(\neg t_1... \vee ...\neg t_{n-1} \vee t_n, w)$. This equivalence between implication and disjunction can also be used to convert a translation of a GenSAT theory into a MaxSAT constraint

that has possibly infinitely many clauses, some of which have infinite many disjunctions.

Since a translation can have an infinite number of clauses with the same literal in the consequent, there can be mandatory causation constraints that have an infinite number of disjuncts. All other clauses have only finite numbers of disjuncts because they are translations of GenSAT clauses, all of which have finite relational disjuncts.

Models

Propositional translations of GenSAT theories motivate a straightforward approach to defining models of them. In many cases where the translation is infinite, it is possible to identify finite models of a theory that are suitable for the purposes of inference. We introduce the notion of relevance in order to characterize such models.

A model for a GenSAT theory is simply an assignment of truth values to the ground literals in its translation. The cost of a model is the sum of the weights on the clauses that are unsatisfied under this assignment. A clause, $...l_i... \rightarrow ...l_j...$, is unsatisfied if all the l_i are true and l_j is false or if one of the l_j are false and all of the l_i are true. A solution to a GenSAT theory is a model whose cost is finite and not exceeded by the cost of another model for that theory. We say that an object is an object of a model if it occurs in a literal assigned a truth value by the model. Except for the possibility of involving infinite numbers of literals, GenSAT models are similar to models of ordinary MaxSAT theories.

Satisfiability algorithms return models of SAT theories. This creates a problem for making inferences using GenSAT theories because they can have infinite models. In many cases, however, it is possible to identify "relevant" parts of models that suit many inferential purposes. In this section, we introduce and more precisely characterize the notion of a relevant model.

As a simple illustration, consider the GenSAT theory with two clauses, $Pet(?x) \rightarrow (10)Loved(?x)$ and $Dog(?x) \rightarrow Furry(?x)$, and one fact, $Dog(d)$. Intuitively, we can infer from one fact from this theory: that the dog, d, is furry. There are however four models for this theory. In each of these models, the d is furry, but the models vary according to whether d is a pet or loved.

Given what is known, however, whether d is a pet or loved cannot be inferred and can in no way affect whether d is furry. Ignoring the Pet and $Loved$ literals, we have one relevant model:

$Dog(d)$	$Furry(d)$	$Cost$
1	1	0

Definition 1 *A model of GenSAT theory P is **relevant** iff it can be derived by eliminating all irrelevant truth value assignments from a model for P. A literal l and a truth value assignment (l, tv) are **relevant in a model** if (i) l is a ground literal in a clause in P or if (ii) l is in a grounding of a clause, C, that has a relevant literal and l's truth value can affect whether C is satisfied.*

As an example of the second condition above, if $Furry(d)$ is false in a model, then $Dog(d)$ is relevant in that model because if $Dog(d)$ is true then the clause is unsatisfied. Since neither $Pet(d)$ nor $Loved(d)$ occur in the GenSAT theory and because they occur in no clauses in the translation that have a relevant literal, they are not assigned in any relevant models.

A relevant solution to a GenSAT theory is a relevant model whose cost is finite and not exceeded by another relevant model.

Finding models

An algorithm for finding models of GenSAT theories must address several challenges. First, in cases where a theory has both finite and infinite models, care must be taken to not endlessly explore models with infinite size. Second, even in cases where there are no infinite models, the size of the propositionalized translation can be very large. Finally, in cases where all objects are not known during inference, all possible supports for a literal cannot be known in advance either. Thus, an algorithm can only guess at the mandatory support constraint before exploring models, and it must potentially alter that guess as search proceeds.

The GenDPLL algorithm successfully addresses these issues under certain conditions described below. It is based on DPLL "branch and bound" (Borchers and Furman, 1999) algorithms. These algorithms perform a depth-first search of the space of possible models by branching on one literal at a time, first exploring one truth value for that literal and then the other. Each time DPLL assigns a truth value to a literal, it performs an elaboration step, which sets variables to values implied by current assignments. "Branch and bound" DPLL algorithms avoid needless search by ceasing to explore a partial assignment when its predicted cost exceeds the best model found so far (or some threshold set a priori).

Algorithm 1 $GenDPLL(P, t)$

Require: P is a GenSAT theory and $t \geq 0$

Choose an arbitrary skolem function s.
$initialClauses = OneStep(Objects(P), P, s)$
$q =$ the empty queue
$m = GenDPLLInner(P, t, q, \{\}, intialClauses, 0, nil)$
if $(m \neq fail)$ **then**
  return m
else
  return $GenDPLL(P, 2t)$
end if

In order to not endlessly explore infinite-cost models, GenDPLL begins searching (GenDPLLInner) with a cost threshold such that partial models are excluded if their cost exceeds it. If no such models are found, search begins again with the threshold doubled. This can lead to parts of the search space being explored more than once, but this can often be addressed in practice by estimating the cost of the best model and choosing a threshold significantly above that estimate.

The main procedural differences between GenDPLL and DPLL occur during the elaboration step. Elaboration involves three elements that address the three problems with

infinite models we have just described. First, in order to conserve space, GenDPLL lazily grounds GenSAT into weighted DPLL constraints. Second, the elaboration step uses a variable selection and unit propagation scheme that prevents infinite models from being explored unnecessarily. Finally, at any given time, it explicitly assumes that all the possible supports for a literal are known. When a new possible support for a literal is inferred, GenDPLL backtracks to the stage where it assumed that all supports for that literal were known. Thus, at any given time, the assumptions about all supports being known for a literal are encoded and can be retracted. These aspects of GenDPLL are explained in further detail presently.

Algorithm 2 *GenDpllInner(P, threshold, q, assignment, propClauses, depth, best)*

Require: P is a GenSAT theory, q is a queue of propositionalized literals, $t > 0$, $maxNewObjects \geq 1$, $assignment$ is a partial assignment from propositional literals, $propClauses$ is a set of propositionalized clauses, $d \geq 0$, $bestModel$ is a model of P or nil.

$elaborate(P, assignment, propClauses, depth)$
$weight \leftarrow$ total cost of unsatisfied clause in $propClauses$ under $assignment$.
if there is a hard contradiction or $weight > threshold$ **then**
  return $fail$
else if q is empty **then**
  return $assignment$
else
  $next \leftarrow q.next()$
  $newAssignment \leftarrow$ *GenDPLLInner(P, threshold, q, assignment[next/true], propClauses, depth, bestModel)*
  if $(newAssignment = fail)$ **then**
    return $GenDPLLInner(P, threshold, q, assignment[next/false], propClauses, depth, bestModel)$
  else
    return $GenDPLLInner(P, threshold, q, assignment[next/false], propClauses, depth, newAssignment)$
  end if
end if

GenDPLL takes as input a GenSAT theory and a maximum cost for models it considers. Its output is one of the models with the lowest cost. In describing GenDPLL we make use of the functions introduced in the section on propositional translation.

GenDPLL maintains several data structures. `groundConstraints` is the list of propositional constraints that have been translated from first-order GenSAT clauses. "The queue", or `q`, contains propositional literals to be branched on. If a literal is put on the queue in a partial model that has been backtracked from, it is removed from the queue during backtracking. Specifically, literals on the queue are associated with the depth of search (`d`) when they were put on the queue. When backtracking from a level, all literals from that level are dequeued.

Algorithm 3 $Elaborate(P, assignment, propClauses, depth)$

Require: P is a GenSAT theory, $assignment$ is a partial assignment of literals to truth values, $propClauses$ is a set of propositional clauses and $depth \geq 0$
$maxNewObjects \leftarrow$ an arbitrary nonzero
purge q of literals added at depth levels greater than $depth$
$n \leftarrow 0$
$oldMCCLauses = mcCLauses(D)$
repeat
  $propClauses \leftarrow propClauses \cup$ all relevant literals in $OneStep(P, objects(PropClauses))$
  $newLiterals \leftarrow$ literals in D not in $assignment$ or already in q
  add $newLiterals$ to q
  $n \leftarrow n+$ number of newly posited objects in $newLiterals$
until $n < maxNewObjects$
$newMCCLauses \leftarrow$
$mcClauses(D, P) - oldMCCLauses$
add to q all relevant literals in $newMCClauses$ not in $assignment$ or not already in q
$propClauses = propClauses \cup newMCCLauses$
move to front of q all known supports assumption literals in $newMCCLauses$

Lazy Instantiation. GenDPLL begins by finding all grounded GenSAT constraints and adds the literals in them to the queue. It then performs an elaboration step (described in more detail below) that grounds conditionals involving the current literal and adds new relevant (according to the the definition in section) literals to the queue. If the elaboration step finds that the current assignment's cost exceeds the threshold, failure is returned and literals put on the queue at this depth of search are removed. If there are no more literals on the queue, this assignment and weight are returned and the threshold is lowered to the new weight. If there are more literals to explore, GenDPLL "branches" on the next literal in the queue.

Finite elaborations. Since grounding can add new objects, which themselves can lead to further grounded clauses, it is a potentially infinite process. Consequently, GenDPLL only adds a finite number of objects (`maxNewObjects`) during each elaboration step. This prevents elaboration steps that do not terminate. Since literals that might lead to more objects being posited remain on the queue, they will be elaborated eventually, so long as this partial assignment is not backtracked from.

Mandatory causation. During the elaboration step, if a literal, l, is created for the first time and it appears on the right hand side of a ground causal rule, GenDPLL "assumes" all supports are known for it and creates a mandatory support constraint. Specifically, for a $P(a_1, ..., a_n)$ the literal $AllSupportsKnownPm(a_1, ..., a_n)$ (where m is the number of known-support literals that have been

instantiated for $P(a_1, ..., a_n)$) is added to the current assignment as true and the following constraint is added: $AllSupportsKnownPm(a_1, ...a_n) \wedge \neg C_1 \wedge \neg C_m \rightarrow \neg P(a_1, ..., a_n)$.

Literals whose predicates begin with $AllSupportsKnown$ [3] When another causal clause is added with $P(a_1, ..., a_n)$ on the right hand side, the point in search where its known-supports assumption was made is backtracked to and the assumption is constrained false.

Increasing Cost Theories

There are many cases where infinite numbers of models and/or models of infinite size are possible, but where a finite model is guaranteed to be the best solution. For example, some context-free grammars generate infinite numbers of trees. However, if the parses are weighted such that each phrase in the parse adds a "cost" to that parse tree, then if there are any finite parses at all, the parse with the best cost will be finite. As another example, consider path search problems with an unbounded numbers of moves being possible. If there is a path of finite length and there is a cost associated with each move, then the best (i.e., least costly) path will be finite, even if there are paths of infinite length.

In cases such as these, where the cost of a theory's model grows as the size of the model grows, GenDPLL can find models of GenSAT theories even when there are infinitely many models and/or some have infinite size. In this section we will precisely characterize such "increasing cost theories" and show that GenDPLL is guaranteed to find finite models of them, when they exist.

First, we show that GenDPLL is sound.

Proposition 1 *GenDPLL only returns models whose assigned literals are all relevant.*

Proof. When finding a model for a theory P, GenDPLL will only put the following two kinds of literals on the queue: ground literals in clauses in P and relevant literals in conditionals ground from other literals on the queue that have been assigned. These are precisely the conditions of literal relevance. Since GenDPLL will only make assignments for terms in the queue, all literals in the assignments it forms will be relevant. □

When models for a GenSAT theory increase in cost as objects are added, GenDPLL will halt. We can state this more precisely using the notion of an increasing cost theory.

Definition 2 *A GenSAT theory P is an **increasing cost theory** if for each object o that does not occur in P (i.e., for each object that will be posited during inference), o is involved in at least one satisfied clause (which by definition will have a positive nonzero cost).*

Proposition 2 *For increasing cost theories, GenDPLL terminates and when there is a relevent solution, GenDPLL returns one.*

[3] If predicates beginning with "AllSupportsKnown" exist in the GenSAT theory, GenDPLL will choose another name for this predicate.

Proof. First, we show that for one of the relevant solutions, s, for an increasing-cost theory, GenDPLL will find it and then we show that it will do so in finite time. If a literal, l, is relevant, it is, by definition, either because (1) it is a ground literal in P (in which case it is either put on the queue to be branched on or assigned at the beginning of GenDPLL) or (2) it is relevant under an assignment of other ground literals in P or in partial assignments whose literals are already relevant. In the second case, the elaboration step either branches on or unit propagates values for these literals. These two cases correspond exactly to the definition of a relevant literal. Thus, unless a relevant solution has already been found, s will be found.

This will occur in finite time because all models with higher cost will exceed threshold or the cost of the best model as objects and the cost of unsatisfied constraints they participate in are added. Such constraints are guaranteed to exist by definition of an increasing-cost model. □

Conclusions

Many problems can be formulated using relational theories over known and unknown objects. Such theories can have infinite numbers of models with infinite variables. On problems where each unknown object is associated with at least one broken constraint, DPLL-like search with lazy constraint instantiation and explicit assumptions about causes for objects being known can be used to find solutions.

References

Borchers, B., & Furman, J. (1999). A two-phase exact algorithm for MAX-SAT and weighted MAX-SAT problems. *Journal of Combinatorial Optimization*, 2, 299-306.

Domingos, P., & Richardson, M. (2006). Markov Logic Networks. *Machine Learning*, 62, 107-136.

Een, N., & Sorensson, N. (2005). MiniSat-A SAT solver with conflict-clause minimization. In *SAT 2005 Competition*.

Heras, F., Larrosa, J., & Oliveras, A. (2008). MiniMaxSAT: An Efficient Weighted Max-SAT Solve. *Journal of Artificial Intelligence Research* 31, 1-32.

Jackson, D. (2000). Automating first-order relational logic. Paper presented at the *ACM SIGSOFT Symposium on Foundations of Software Engineering*.

Kersting, K., & De Raedt, L. (2001). Towards combining inductive logic programming with Bayesian networks. Paper presented at the *ILP-01*.

Milch, B., Marthi, B., Sontag, D., Russell, S., & Ong, D. L. (2005). Approximate inference for infinite contingent Bayesian networks. Paper presented at the *AISTATS-05*.

Muggleton, S. (1996). Stochastic logic programs. In L. D. Raedt (Ed.), *Advances in inductive logic programming* (pp. 254-264): IOS Press.

Selman, B., Levesque, H., & Mitchell, D. (1992). A new method for solving hard satisfiability problems. Paper presented at the *Tenth National Conference on Artificial Intelligence*.

Singla, P., & Domingos, P. (2006). Memory-Efficient Inference in Relational Domains. Paper presented at *AAAI-06*.