Algorithms for Markov Logic Networks

José Francisco Pires Amaral
jfpa@sat.inesc-id.pt

INESC-ID / Instituto Superior Técnico, Lisboa, Portugal

November 2017

Abstract

Many real world applications have to deal with uncertainty. Consequently, they demand probabilistic methods to represent some (if not all) of their events to describe their likelihood. Moreover, these applications are usually large-scale and need a compact and plain representation: first-order logic. Markov Logic Network (MLN) combines both these aspects. MLN associates first-order logic formulas to a real value (weight) that represents their likelihood. This thesis explores two main hypothesis. First, we explore the use of incremental Maximum Satisfiability (MaxSAT) to solve MLN problems. Existing methods that solve MLN rely on various independent Weighted Partial Maximum Satisfiability (WPMS) solver calls. Our main goal is to benefit from recent improvements in WPMS solving techniques. In this particular case, we intend to make previous WPMS solver calls useful to the subsequent WPMS solver calls. State-of-the-art approaches make each WPMS call start the search from the beginning whereas our WPMS solver calls use previously learned information, thus converging faster to the solution. Secondly, we explore the hypothesis of using an Satisfiability Modulo Theories (SMT) solver to validate the constraints. In other words, during the algorithm we need to guarantee that the current solution does not violate any rules that are not yet instantiated. Some state-of-the-art approaches use a Datalog solver to validate each constraint. This method implies the use of queries to a database. Our main goal is to avoid having a database altogether. Finally, we evaluate Inference using PROpagation and Validation (ImPROV), i.e. our approach, with two state-of-the-art tools: Tuffy [29] and the Inference via Proof and Refutation (IPR) algorithm [19].
Keywords: Markov Logic Networks, Satisfiability Modulo Theories, Maximum Satisfiability

1. Introduction

Many real world applications or problems demand probabilistic methods to depict some (if not all) of their events to either mark their importance among the group or describe their likelihood. These applications are usually large-scale and need a compact and plain representation: first-order logic. Markov Logic Network (MLN) combines both these characteristics.

There are already practical examples of MLN. Advisor Recommendation (AR), also known as Link prediction, Entity Resolution (ER) [40], Information Extraction (IE) [31], Program Analysis (PA) [18] and Relational Classification (RC) [29] are some of the many possible applications of MLN.

MLN is of good use when there are certain objectives we want to optimize in addition to some rules we do not want to break.

We intend to improve the way some recently proposed algorithms pursue the solution to these problems. In order to solve MLN problems these algorithms mainly use independent Weighted Partial Maximum Satisfiability (WPMS) solver calls. These MaxSAT calls restart the search in each iteration. Since solving MLN problems usually involve many iterations, our main goal is to reuse computation throughout the algorithm. We want to minimize the amount of time spent on repeated computation, as well as make our algorithm use the information obtained in previous iterations. Furthermore, during the main loop of the algorithm, when looking for counterexamples, we use an Satisfiability Modulo Theories (SMT) solver instead of a Datalog solver. We aim to validate each constraint without using a database.

Important to note that we solve these problems using a Maximum A Posteriori (MAP) inference method. MAP inference can be used to find the most likely state of world given some evidence. Note that the same problem could be solved using a different inference and therefore have a distinct solution.

In the next section we introduce the most important concepts mentioned in the remainder of the document such as Propositional Satisfiability (SAT) problems, Maximum Satisfiability (MaxSAT) prob-

lems and Markov Logic Networks (MLN) in a more detailed way. A small and straightforward MLN example is also presented. Subsequently, we briefly overview the progression and evolution throughout the years on the different grounding techniques, on the distinct possible techniques used to solve MaxSAT problems as well as the most recent methods to solve MLN problems. Then, we describe each component of the developed tool, its architecture and the algorithms it uses during the search. Afterwards, we present the results obtained over the tested applications and a comparison between the described algorithm and two state-of-the-art tools. Lastly, we present a conclusion with a brief summary of the developed tool, describing its positives, negatives and the results obtained.

2. Preliminaries

In propositional logic, formulas are represented over a set of Boolean variables $X = \{x_1, x_2, ..., x_n\}$, where n is the total number of variables and each variable x_i must take one of two values: true or false.

These formulas are usually presented in Conjunctive Normal Form (CNF), that is a conjunction of clauses. Each clause is a disjunction of literals. A literal can be a variable or its negation (e.g. x_i or $\neg x_i$). At least one literal in each clause must be satisfied (have value true) so that the entire formula is satisfied. For instance, $(x_1 \vee x_2) \wedge (x_3 \vee \neg x_4)$ is a formula in CNF with two clauses.

2.1. Propositional Satisfiability

Given a CNF formula ϕ, the goal of Propositional Satisfiability (SAT) is to find an assignment to the variables in ϕ such that ϕ is satisfied or prove that such assignment does not exist.

For example, $(\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee \neg x_3)$ is a CNF formula describing a SAT problem. One of its solutions is $x_1 = false$, $x_2 = false$, $x_3 = false$.

On the other hand, there are formulas such that there is no satisfying assignment. For instance, $(x_1) \wedge (x_2) \wedge (\neg x_1 \vee \neg x_2)$ is an unsatisfiable CNF formula.

2.2. Maximum Satisfiability

MaxSAT is an optimization version of SAT. Given a CNF formula ϕ, the goal of MaxSAT is to find an assignment to the variables in ϕ such that it maximizes the number of satisfied clauses. This is equivalent to minimizing the number of falsified clauses. For the remainder of the document, every MaxSAT problem is considered as a minimization problem.

Consider the unsatisfiable CNF formula $(x_1) \wedge (x_2) \wedge (\neg x_1 \vee \neg x_2)$. A possible MaxSAT solution to this formula is $x_1 = true$, $x_2 = true$, since it falsifies only one clause (third) which is the minimum possible number of falsified clauses with any assignment.

There are some variations of MaxSAT [26]. Weighted MaxSAT occurs when some clauses have more relevance than others. This relevance is represented by its weight (a positive real number). Given a weighted CNF formula, the goal of weighted MaxSAT is to minimize the sum of the weights of the unsatisfied clauses.

The variations partial MaxSAT and weighted partial MaxSAT are extensions of unweighted MaxSAT and weighted MaxSAT respectively. These two variations distinguish hard clauses and soft clauses. Hard clauses must be satisfied while soft clauses do not need to be satisfied. The goal is to satisfy all hard clauses while minimizing the total weight of the unsatisfied soft clauses.

2.3. Markov Logic Networks

Markov Logic Network (MLN) is a simple way to merge both first-order logic and weighted atomic formulas, the latter being atoms with a real number attached to it. An MLN can be defined as a set of weighted first-order logic formulas. A weighted first-order logic formula is a pair (ϕ, w) where ϕ is a first-order logic formula and w is a real number (weight).

Since each first-order logic formula has an attached weight, the notion of hard and soft clauses (presented in the previous section) can be used in MLN. In other words, each first-order logic formula can be seen as hard or soft.

Figure 1 shows an abbreviated example of an MLN for scheduling classes [3] where section a) defines the set of relations used in the first-order logic formulas, section b) defines the set of formulas and section c) defines the evidence or facts.

Each weighted first-logic formula presented in figure 1 can be converted to clausal form. If we instantiate the facts over the relations using the converted formulas we can generate new clauses in conjunctive normal form (CNF). Note that CNF is used to represent clauses in both SAT and MaxSAT problems. If all the facts are instantiated over the relations, a full grounding of the problem in CNF is generated.

A fully grounded MLN can thus be considered as a weighted MaxSAT problem.

A fully grounded MLN represents a full instantiation of the facts over all constraints. An MLN can also be partially grounded if not all the facts are instantiated. This grounding topic is further explained in the next section.

Both the fully grounded MLN and the weighted MaxSAT problem have the same goals: first, they can not violate any hard constraints/clauses (soundness); secondly, both try to minimize the sum of the weights of the falsified soft con-

$Teaches(p, c)$: Professor p teaches course c.
$Friends(s1, s2)$: Student s1 is a friend of student s2.
$Likes(s, p)$: Student s likes professor p.
$Attends(s, c)$: Student s attends course c.
$Popular(p)$: Professor p is popular.

a) Relations:
$Teaches(Professor, Course)$, $Friends(Student, Student)$, $Likes(Student, Professor)$,
$Attends(Student, Course)$, $Popular(Professor)$

b) Weighted formulas:
(- Premise -)
(1) 1.0: $\forall s_1, s_2 \quad Friends(s_1, s_2) \implies Friends(s_2, s_1)$
(- Soft formulas -)
(2) 0.7: $\forall p, c, s \quad Teaches(p, c) \wedge Likes(s, p) \implies Attends(s, c)$
(3) 0.6: $\forall p, c, s \quad Teaches(p, c) \wedge Popular(p) \implies Attends(s, c)$
(4) 0.5: $\forall s_1, s_2, c \quad Friends(s_1, s_2) \wedge Attends(s_1, c) \implies Attends(s_2, c)$

c) Evidence:
$Teaches(D, Compilers)$
$Friends(J, P)$
$Popular(D)$
$\neg Attends(P, Compilers)$

Figure 1: An MLN for scheduling classes including evidence.

straints/clauses (selecting the optimal solution).

3. Algorithms for Markov Logic Networks

Due to artificial intelligence, combining logic and probability has been a studied research field for at least a few decades [33]. This section overviews the progress of the work related with solving both Maximum Satisfiability (MaxSAT) and Markov Logic Network (MLN) problems.

In MLN, one of the differences throughout the years has been the approach of each algorithm in the grounding phase. In this phase, constraints are grounded by instantiating the relations over all facts in their respective domain [19]. This phase can be accomplished by pursuing an eager approach, a lazy approach or even a combination of both.

An eager method grounds a set of constraints immediately (without knowing if they are going to be used) while a lazy method only grounds these constraints when they are needed.

Lazy approaches have been crafted to deal with the (large) size of the problem. Real world situations are usually very large. Therefore, most of the recently presented algorithms do not explore all the constraints extensively in the beginning of the method.

3.1. Grounding techniques

During the grounding phase, Tuffy [29] uses a bottom-up approach expressing grounding as a sequence of SQL queries. Using its own algorithm, a set of clauses is selected to be eagerly grounded. This can have scalability issues (if the set is excessively large) or downgrade the quality of the solution (if the set is too small).

On the other hand, lazy inference techniques [32] take advantage of the fact that most grounded facts have a specific value that is generally more common than others. Therefore, these techniques initiate all ground facts with a default value and only ground new clauses when these values can be changed for a better goal.

Soft-CEGAR [3] tries to combine both approaches: eagerly grounds the soft constraints and solves the hard constraints in a lazy manner.

Figure 2 shows the full grounding of the MLN formula of figure 1 in clausal form.

3.2. Algorithms for MaxSAT

Since a fully grounded MLN can be regarded as, or converted to a MaxSAT problem, some MLN algorithms use MaxSAT extensively. In light of this, we present an overview of those algorithms.

In the early MaxSAT solving days, the algorithms used were based on Stochastic Local Search (SLS) [12, 37, 36] and aimed to approximate the MaxSAT solution.

Another used technique consists of converting the MaxSAT problem into a known optimization problem such as an Integer Linear Program (ILP)

Evidence:
$Teaches(D, Compilers)$
$Friends(J, P)$
$Popular(D)$
$\neg Attends(P, Compilers)$

Grounding:
(1) 1.0: $\neg Friends(J, P) \vee Friends(P, J)$
(1) 1.0: $\neg Friends(P, J) \vee Friends(J, P)$
(1) 1.0: $\neg Friends(J, J) \vee Friends(J, J)$
(1) 1.0: $\neg Friends(P, P) \vee Friends(P, P)$
(2) 0.7: $\neg Teaches(D, Compilers) \vee \neg Likes(J, D) \vee Attends(J, Compilers)$
(2) 0.7: $\neg Teaches(D, Compilers) \vee \neg Likes(P, D) \vee Attends(P, Compilers)$
(3) 0.6: $\neg Teaches(D, Compilers) \vee \neg Popular(D) \vee Attends(J, Compilers)$
(3) 0.6: $\neg Teaches(D, Compilers) \vee \neg Popular(D) \vee Attends(P, Compilers)$
(4) 0.5: $\neg Friends(J, P) \vee \neg Attends(J, Compilers) \vee Attends(P, Compilers)$
(4) 0.5: $\neg Friends(P, J) \vee \neg Attends(P, Compilers) \vee Attends(J, Compilers)$
(4) 0.5: $\neg Friends(J, J) \vee \neg Attends(J, Compilers) \vee Attends(J, Compilers)$
(4) 0.5: $\neg Friends(P, P) \vee \neg Attends(P, Compilers) \vee Attends(P, Compilers)$

Figure 2: Full grounding of the MLN presented in figure 1 (clausal form).

problem [41], where several approaches take advantage of recent SAT-solvers with clause learning and backjumping techniques [8, 39, 27]. Another option is converting the MaxSAT problem to a Maximum Answer Set Programming (MaxASP) problem [10], where recent techniques take advantage of algorithms with a branch and bound approach [30].

A large group of optimal MaxSAT-solvers adopt a branch and bound algorithm [6, 11, 17]. On the other hand, in recent years, algorithms that iteratively call a SAT-solver have proved to be better than other algorithms, for industrial benchmarks[1].

This group of algorithms iteratively calls a SAT-solver until it finds an optimal solution. In order to find the optimal solution, these algorithms maintain a lower and an upper bound on the solution cost. These bounds can be calculated using relaxation variables. A relaxation variable is added to each soft clause such that whenever the soft clause is unsatisfied, the relaxation variable is assigned to true.

The iterative SAT-solver call that is used during these algorithms has as input a Weighted Conjunctive Normal Form (WCNF) formula. This SAT-solver call returns three fields. The first one has the information whether the formula given as input is SAT or Unsatisfiable (UNSAT). If the formula is SAT, a satisfying assignment (complete) is returned. If the formula is UNSAT, an unsatisfiable subformula (also known as an UNSAT core) is returned.

[1]http://www.maxsat.udl.cat/

Algorithms based on linear search only refine one (either lower or upper) bound during their search. These algorithms start by relaxing all soft clauses. Then, they try to refine their bound at each iteration.

Algorithms that iteratively improve their upper bound (called Linear Search Sat-Unsat (LSSU) [26]) present the formula as satisfiable for all the SAT-solver calls except the last one. SAT4J [2] and QMaxSAT [16] are two of various MaxSAT tools that follow a LSSU strategy.

On the other hand, algorithms that iteratively refine their lower bound (called Linear Search Unsat-Sat (LSUS) [26]) declare the formula as UNSAT for all the SAT-solver calls with the exception of the last call.

An UNSAT core is returned by the SAT-solver call whenever there is an UNSAT outcome. These cores can be used to guide the search. These algorithms are called core-guided MaxSAT algorithms [26]. The solving scheme of these algorithms is similar to the ones based on linear search.

Fu and Malik [9] introduced a significant and limited core-guided MaxSAT algorithm. It was limited because it was only capable of solving unweighted partial MaxSAT. At each iteration, a new UNSAT core is extracted. A relaxation variable is added to each soft clause that belongs to the UNSAT core and a new constraint limiting the total weight of relaxation variables that are added to the formula.

There were some improved versions of this algorithm [21, 22]. It was also extended in order to solve the weighted partial MaxSAT case [1, 20].

Lastly, we present the *Fu & Malik* algorithm that solves weighted partial MaxSAT (WPMS) using SAT incrementally [23]. This algorithm uses a set of assumptions that works as an enabler or disabler of soft clauses. The key difference between an assumption and an unit clause is that in order to change a unit clause, one needs to destroy the current SAT solver and recreate it from the ground up. The use of a set with assumptions permits the user to flip the value of a variable without needing to destroy the SAT solver. This distinction is most important since the goal of this algorithm is to be incremental and use the same SAT solver throughout the computation.

Important to note that since the working formula is always expanded, the SAT solver is never rebuilt and its internal state is kept (including the learnt clauses). *Fu & Malik* with incremental blocking significantly outperforms the non-incremental algorithm. Incremental blocking not only solves more instances but is also significantly faster than the non-incremental algorithm.

This section presents the evolution throughout the years of the techniques used to solve MaxSAT problems. The evolution of SAT solvers permits them to return UNSAT cores whenever the SAT solver call is unsatisfiable. These UNSAT cores are used to guide the search in the MaxSAT algorithms. An important point to consider is that any of these MaxSAT algorithms can be used as a tool when solving the MLN problems. Specifically in our approach, the MaxSAT call in the main loop can use any method from the ones presented in this section. In our case, the MaxSAT solver we use (Open-WBO [24]) uses the last algorithm from this section.

3.3. Algorithms for MLN

In this section, we present the evolution on how to solve Markov Logic Network (MLN) problems throughout the years. Important to note that we solve these problems using a Maximum A Posteriori (MAP) inference method. MAP inference can be used to find the most likely state of world given some evidence. Note that the same problem could be solved using a different inference and therefore have a distinct solution.

Riedel [34] introduced the Cutting Plane Inference (CPI) method. It offered a way to improve existing algorithms such as MaxWalkSAT (MWS) [14] and ILP [41]. It was inspired by the Cutting Plane Method (CPM) [5].

CPI tries to get around the problem of utilizing the full grounding of an MLN by incrementally using only part of the problem instead of the whole network. It optimizes the solution over time and solves these smaller problems using a Maximum A Posteriori (MAP) inference method. Most of the time, these smaller problems are easier to solve and less complex than the entire problem.

Riedel [34] shows that using CPI in conjunction with a base solver is better than using the solver alone. Later, Riedel [35] analyses in detail the usage of CPI with different base solvers, such as MWS. It is also formally demonstrated that CPI's accuracy depends on the base solver's accuracy.

More recently, the Soft-Cegar algorithm was introduced [3]. It computes a MAP solution for a given MLN. The Soft-Cegar algorithm was based on three ideas. Counterexample-guided abstraction refinement [4], theorem proving [7] and generalization techniques for accelerating convergence of the loop [25]. It is proved that the Soft-Cegar algorithm returns an exact MAP solution assuming that the MAP solver used during the algorithm always returns exact MAP solutions [3]. The authors compare Soft-Cegar with two state-of-the-art relational engines: Alchemy [15] and Tuffy [29] over four real world applications. This comparison consists of one instance per application. In a nutshell, Soft-Cegar is faster and gives better solutions (lower cost) than the other two engines over all four instances. Chaganty et al. [3] explain the results obtained in these experiments in more detail.

The Inference via Proof and Refutation (IPR) algorithm [19] is an iterative algorithm that follows three important pillars: eager proof exploitation, lazy counterexample refutation and termination with soundness and optimality. IPR eagerly analyses the relational constraints in order to produce an initial grounding. This step is used to speed up the convergence of the algorithm. At each iteration, after a solution is found IPR lazily grounds the constraints that are falsified by the current solution. If an exact WPMS solver is used, IPR's termination check assures the soundness and optimality of the solution [19].

The authors compared the IPR algorithm [19] with Tuffy [29] and CPI [34, 35] using three different benchmarks with three distinct inputs. In short, IPR outperforms the other two approaches in terms of runtime in these instances but has very similar solution costs when compared to CPI. Tuffy presented significantly higher solution costs than the other two.

All in all, Maximum Satisfiability (MaxSAT) solvers are the most researched and developed solvers regarding problems that have certain objectives we want to optimize in addition to some rules we do not want to break. Furthermore, all of the Markov Logic Network (MLN) algorithms presented in this section have identical algorithmic structure, that is, an initial phase that usually decreases the algorithm convergence time, followed by a main loop that calls a solver and then proceeds to

validate each constraint with the previously computed solution.

4. ImPROV (Inference using PROpagation and Validation)

In this section we describe our software step by step, explaining the progression of the work and the difficulties we had to face. We present the architecture of the tool, its pseudo-code as well as a comparison with previously presented algorithms.

4.1. Architecture

We assume that the Markov Logic Network (MLN) problem is represented in three files: the MLN file where both the predicates and the rules are described; the evidence file contains all the literals used as facts and the query file informs the algorithm of what predicates it should output in the final solution.

Each parsed rule is either soft or hard. If a weight is present at the beginning then it is a soft rule, otherwise the rule is considered hard. These rules are presented in first-order logic form.

An MLN input file contains a list of predicates followed by the list of rules of the problem. An evidence input file is a list of unary literals, containing a single fact in each line. These facts can be positive or negative. All of these facts from the evidence file need to be satisfied by the solution. A query input file is an enumeration of the predicates that are shown in the output.

4.2. Algorithms

This section describes all algorithms we use in our tool. The algorithms and approaches presented in this section correspond to the entire computation of ImPROV, from the parsing of the input files to the computation of a solution.

4.2.1 Algorithmic structure

Algorithm 1 contains the pseudo-code for the ImPROV algorithm. It receives an MLN M composed of two sets of relational constraints as input, one containing hard constraints (M_H) and another containing soft constraints (M_S). We assume that, throughout the algorithm, the set of grounded hard clauses is always satisfiable.

Lines 2 to 8 contain the preprocessing and initializations of the algorithm. The main loop of the algorithm is represented from line 11 to line 29. ImPROV has a similar algorithmic structure as the Inference via Proof and Refutation (IPR) algorithm, presented in section 3.3. ImPROV exploits the structure of Horn constraints and does the same initial propagation as IPR. Throughout the main loop, both algorithms call a Weighted Partial Maximum Satisfiability (WPMS) solver and validate each rule using the current MaxSAT solution. Both main loops are an adaptation of the Linear Search Unsat-Sat algorithm. However, there is a key difference between both algorithms: whenever a rule is not validated, the IPR algorithm finds all the counterexamples of the tested rule and grounds them while ImPROV only finds two counterexample per validation call. The termination check of ImPROV (line 28) verifies if all the rules that are not grounded are satisfied by the current MaxSAT solution. As soon as none of the rules (not yet grounded) are falsified by the current MaxSAT solution, the algorithm terminates (line 29). On the other hand, since IPR returns the solution found in the previous iteration, the algorithm needs to check if the current solution cost is equal to the solution cost calculated in the previous iteration. The solution found by both algorithms is optimal, meaning it has the lowest solution cost possible.

4.2.2 Preprocessing and Initializations

Since MaxSAT solvers can not deal with first-order logic formulas, in the beginning of the algorithm, these formulas have to be converted to Conjunctive Normal Form (CNF), so that the MaxSAT solver can solve the formulas presented in each iteration.

Before solving the MLN instance we need to construct our Satisfiability Modulo Theories (SMT) solver. The SMT solver can be used in default mode or it can be used with a tactic or even a combination of tactics. In our algorithm, we use the Z3 SMT solver [7]. We use a built-in tactic called macro-finder. In our case, the use of this tactic significantly improves the search and therefore saves some time per SMT solver call.

After these initializations are done, our algorithm can be divided into two distinct sections. Firstly, in order to speed up the search and diminish the number of iterations, our tool uses an initial eager propagation. Subsequently, it uses a lazy method in the main loop of the algorithm.

In order to do our initial eager propagation we check if there are any Horn constraints in the input. A Horn constraint [13] is a disjunction of literals with at most one positive literal. It is typically written in the form of an inference rule ($u_1 \wedge ... \wedge u_n \implies u$). Our eager exploration is done only with Horn constraints. If these Horn constraints contain only one single hidden predicate, our algorithm propagates the current values and new grounding instances are discovered. We have this eager approach because these instantiated constraints would be found by the lazy section anyhow. If we do them all at once, not only do we save time, but also diminish the number of iterations of the main loop. If there are no Horn constraints or none of the Horn constraints have a single hidden predicate this step is ignored and the algorithm pro-

Algorithm 1: ImPROV Algorithm

```
1  Function ImPROV(M = M_H ∪ M_S, evidence)
       Input: MLN M = M_H ∪ M_S, evidence
       Output: Assignment ν_sol
2      φ ← evidence                                              // evidence from input
3      (ψ, φ', ψ') ← (∅, ∅, ∅)
4      foreach r ∈ M do
5          convertRule(r)                                        // Convert from first-order logic to CNF
6          if isHorn(r) then
7              (φ', ψ') ← initialPropagation(r, evidence)        // eager initial propagation
8              (φ, ψ) ← (φ ∧ φ', ψ ∧ ψ')                         // update hard/soft clauses
9      foreach (s_i, w_i) ∈ M_S do
10         CE_i ← ∅                                              // initialization of counterexample set
11     while true do
12         φ' ← ∅
13         ψ' ← ∅
14         rulesValidated ← true
15         ν_sol ← WPMS(φ, ψ)                                    // call WPMS solver
16         foreach h ∈ M_H do
17             (st, ν_SMT) ← SMT(¬h ∧ ν_sol)
18             if st = SAT then
19                 φ' ← φ' ∪ ν_SMT                               // add SMT's counterexample to the current grounding
20                 rulesValidated ← false
21         foreach (s_i, w_i) ∈ M_S do
22             (st, ν_SMT) ← SMT(¬(s_i) ∧ ν_sol ∧ CE_i)
23             if st = SAT then
24                 ψ' ← ψ' ∪ ν_SMT                               // add SMT's counterexample to the current grounding
25                 CE_i = CE_i ∪ ¬ν_SMT                          // add negation of counterexample to set
26                 rulesValidated ← false
27         (φ, ψ) ← (φ ∧ φ', ψ ∧ ψ')                             // update hard/soft clauses
28         if rulesValidated = true then
29             return ν_sol
```

ceeds to the lazy section. Important to note that in practice, most constraints in many inference tasks are Horn [19], which makes this eager exploration useful in most cases. As a result, it tends to improve our algorithm.

4.2.3 Rule validation

In ImPROV's main loop, after the MaxSAT solver call, our algorithm checks, one by one, if there are any rules that are falsified by the current MaxSAT solution. To do that, we feed the SMT solver with the current MaxSAT solution in conjunction with the negation of the rule we want to validate.

If the SMT solver returns UNSAT then the rule is not falsified by the current MaxSAT solution. On the other hand, if the SMT solver returns SAT then there is at least one counterexample that falsifies the current MaxSAT solution. The counterexample given by the SMT solver is then grounded and added to the MaxSAT solver.

4.2.4 Incremental MaxSAT

This section describes ImPROV's MaxSAT solver call. This call is the first computation in ImPROV's main loop.

This section of the algorithm is mentioned as incremental since our method only adds variables and clauses to the MaxSAT solver. The algorithm never removes variables or clauses from the MaxSAT solver. Thus, the MaxSAT formula grows with each iteration. Our goal with this approach is to keep expanding the MaxSAT formula without drastically changing its structure. This concept is used to make sure we take full advantage of using an incremental MaxSAT solver. Using an incremental MaxSAT solver certifies that the computation of the solver is not restarted in each iteration. It continues directly from the computation from the last iteration.

Since our goal is to lazily ground the problem at hand, the use of a solver that allows the addition of new variables and clauses step by step is ideal in this method. This recently presented MaxSAT solver [38] uses UNSAT cores to guide the search and aims to improve the efficiency on solving a sequence of similar problem instances. It is known as an incremental core-guided MaxSAT algorithm. This algorithm uses the *Fu & Malik* algorithm with Incremental SAT [23] to solve its MaxSAT formulas.

Si et al. [38] consider the case when the sequence is constructed by adding new hard or soft clauses. Now this circumstance reflects our own problem perfectly. The solver's approach is similar to the idea of incremental SAT solving presented by Fu and Malik [9] however there are some significant differences.

Note that, since the algorithm only adds clauses, the search does not need to restart from the beginning for each formula in the sequence. The computation and the information learnt can be used for future iterations. Furthermore, the incremental MaxSAT solver does not redo the computation from previous iterations.

This approach is then, incremental at two levels: first, it uses incremental SAT for each MaxSAT formula; second, it uses incremental MaxSAT across the sequence that grows throughout the algorithm.

5. Results

In this section, we test, analyse and present the results of our tool: Inference using PROpagation and Validation (ImPROV). First, we describe the instances we use to evaluate ImPROV; secondly, we find ImPROV's best configuration; lastly, we run these instances with the state-of-the-art algorithms and analyse their results with the ones from ImPROV. All experiments were done on a Linux machine with a 1.4 GHz processor and 132GB of RAM. We set a time out of 30 minutes (1800 seconds) for all the experiments.

5.1. MLN Instances

We evaluate ImPROV with problems generated from three different applications: entity resolution (ER), relational classification (RC) and advisor recommendation (AR). Singla and Domingos [40] used Markov Logic Networks to solve instances of the entity resolution problem. The entity resolution problem was introduced by Newcombe et al. [28]. It is the problem of finding out which records in a database refer to the same object. Relational classification [29] consists of labelling papers using some information about them. In this case, we use the author and the references of each paper to give them a proper category. Finally, Advisor recommendation [3] tries to find the ideal PhD advisor for a starting graduate student. Throughout the evaluation, we use three different instances of each application, with a different number of facts.

5.2. ImPROV's configuration

Following the selection of the instances, we set the proper and best configuration for our tool. This configuration is key to the performance of the algorithm. Some different configurations can change the total time of some instances by several minutes.

5.2.1 Initial eager propagation

The first configuration we can set is the initial eager propagation that was thoroughly explained in section 4.2.2. Note that this step is optional and does not change the result. We use it since it helps the algorithm to converge faster. As mentioned in section 4.2.2, the initial eager propagation can be used when there are Horn constraints [13] and these Horn constraints only have one hidden predicate.

The instances from the RC application have different results when using initial eager propagation, while the remaining six (ER1, ER2, ER3, AR1, AR2 and AR3) do not have any changes (iterations or total time) when using initial eager propagation. ER and AR do not change since the initial eager propagation method can not be used.

In RC, when we use the initial eager propagation, the number of iterations of the main loop decreases, as well as the total time of the algorithm. This step helps the convergence of the algorithm since it discovers some of the literals upfront. If we use the eager propagation, the algorithm finds these new literals before the main loop (first part of the algorithm). Otherwise, the algorithm finds the literals during the main loop, using the counterexamples from the SMT solver.

We conclude that the use of this initial eager propagation can not be always used, but when it can, it significantly improves the time of the algorithm by diminishing the number of iterations needed.

5.2.2 Counterexample generation

Our goal with these following experiments is to verify what is the best possible amount of counterexamples that the algorithm should return per validation call. We test ImPROV with five different numbers of counterexamples returned per validation call: one, two, five, ten and all counterexamples.

In the entity resolution (ER) application, the results show that the best configuration is to demand all counterexamples per rule validation. When asking for all the counterexamples available, the algorithm finishes much faster than the other configurations.

In the relational classification (RC) application, the results show that the best number of counterexamples per rule validation is five. In this application, these results show us two different aspects: first, since the algorithm is much slower to finish when asking for all the counterexamples (RC3 even reaches the time out limit), there is an excessive amount of counterexamples for at least one rule; secondly, since the other tests (one, two, five and ten counterexamples) finish in a timely manner, we can deduce that there is at least one counterexample (probably more) that cover all the others, i.e.

by adding a few counterexamples, these few may cover the validation of many others.

In the advisor recommendation (AR) application, the results show that the best option is to ask for all the counterexamples upfront, each time there is a rule validation.

This configuration (amount of counterexamples) is the hardest. It can vary immensely with each instance or application. The aforementioned Inference via Proof and Refutation (IPR) algorithm presented in section 3.3 demands all the counterexamples per validation call. In our case, we consider the use of two returned counterexamples per validation call the one with the best results overall. Hence, we request two counterexamples per validation call throughout the rest of the evaluation section.

5.2.3 Incremental vs non-incremental MaxSAT

This section describes the possible configuration of line 15 of the ImPROV algorithm (algorithm 4.2.1). ImPROV uses an incremental core-guided algorithm [38] that was developed specifically to solve sequential MaxSAT problems. The overall performance is improved by reusing computation across similar MaxSAT instances in the same sequence.

5.3. Results with ImPROV's best configuration

There are two critical calls in our main loop: first, the MaxSAT solver call with all the current grounded clauses; secondly, the SMT solver call, used to validate the rules. For each instance, we analyse how much time do both these critical calls take.

The results from all the instances suffer from the same problem. On average, across all the instances, 96.39% of the computation is spent in the SMT solver calls. These calls are vital to the algorithm, considering that they guide the search. Considering these results, the clear bottleneck of the computation is the rule validation method.

5.4. Analysis of the ImPROV algorithm

This section presents the results from using Tuffy [29], the Inference via Proof and Refutation (IPR) algorithm [19] and Inference using PROpagation and Validation (ImPROV). The most important aspect of the results is the correctness of the solution, that is, if the solution is sound and optimal. Sound, meaning the solution does not violate any hard constraints and optimal, meaning the lowest possible solution cost. The second most important aspect is the total time of the computation.

5.4.1 Correctness of the algorithms

As expected, since both ImPROV and IPR are sound and optimal, they present the same solution for all nine instances. These matched solutions are expected because of the similar algorithmic structure and how both groundings evolve during the computation. On the other hand, the results show that Tuffy is not optimal.

5.4.2 Number of iterations

Tuffy does not have the same algorithmic structure as IPR or ImPROV. Simply put, Tuffy does not work with iterations. Thus, when evaluating the iterations of each algorithm, we just compare the results of IPR and ImPROV. Important to note that ImPROV uses two as the amount of counterexamples returned per validation, while IPR requests all existing counterexamples per validation. The ImPROV algorithm has a lot more iterations than the IPR algorithm, in every instance tested. This fact is not necessarily bad for any of the algorithms. A high number of iterations does not correlate directly to a good or a bad computation time, and neither does a low number of iterations.

5.4.3 Solution cost

In the instances from ER and RC, IPR and ImPROV have a solution cost of 0, meaning that the solution does not falsify any soft constraints. On the other hand, Tuffy presents solution costs higher than 0. These solution costs are expected since Tuffy is not optimal. In AR, ImPROV and IPR have the same solution cost across all three instances. These solution costs are higher than 0 since the optimal solution falsifies some soft constraints. Once again, Tuffy presents higher solution costs for all the instances from the AR application.

5.4.4 Performance of the algorithms

As explained in the beginning of this section, after the correctness of the solution is guaranteed, we evaluate the time each algorithm takes to finish the computation. Recall that unlike IPR and ImPROV, Tuffy is neither sound nor optimal. The results from the ER application show that Tuffy is the slowest method in all three instances. In these same instances, IPR is the fastest method. The results from both the other applications (RC and AR) are similar to one another. Tuffy and IPR terminate the overall computation in under one minute in all of these instances. They are both faster than ImPROV in these six instances.

Regarding the overall performance of ImPROV, there are two distinct reasons that slow down the computation. First, as explained in the previous section, the validation of the rules, that is, the SMT solver calls are the bottleneck of the algorithm. This technology is not very good and simply underdeveloped when trying to get numerous counterexamples. Secondly, the amount of counterexamples requested per validation is not an easy value to configure. It depends on the examples and on the technology used.

Table 1: Results of evaluating TUFFY, IPR and ImPROV on three benchmark applications. Experiments that timed out (denoted '-') exceeded the time limit.

	data bases	# iterations		# ground clauses			Solution cost			total time (m = min, s = sec)		
		IPR	ImPROV	TUFFY	IPR	ImPROV	TUFFY	IPR	ImPROV	TUFFY	IPR	ImPROV
Entity Resol.	ER1	3	240	27900	1119	1164	300	0	0	2m23s	3s	28s
	ER2	3	525	93150	2259	2320	809	0	0	13m05s	5s	3m02s
	ER3	3	926	55901	3849	3939	-	0	0	-	7s	13m17s
Relational Classif.	RC1	13	102	2161	1367	1653	18	0	0	5s	3s	42s
	RC2	13	116	3987	2488	3119	10	0	0	7s	4s	2m26s
	RC3	13	273	55901	6124	4758	633	0	0	55s	21s	8m37s
Advisor Recom.	AR1	3	728	1236	3240	9291	5314.8	3320	3320	5s	4s	1m26s
	AR2	3	1641	2956	7260	21216	12710.8	7260	7260	5s	6s	12m20s
	AR3	3	2138	3868	9316	27380	16632.4	9261.6	9261.6	6s	8s	24m44s

5.5. Summary

In conclusion, this section explains the instances used in all the experiments, the possible configurations of the Inference using PROpagation and Validation (ImPROV) algorithm and an analysis of its results.

We compare ImPROV with two state-of-the-art engines: Tuffy and IPR. Results show that IPR is the best engine. IPR returns correct solutions and it is the fastest to conclude the computation. With ImPROV, the use of an SMT solver to validate each rule is the main bottleneck of the algorithm. The validation calls slow the algorithm down considerably. Nevertheless, ImPROV is better than Tuffy. ImPROV always returns a sound and optimal solution, while Tuffy does not guarantee neither soundness nor optimality of the solution.

6. Conclusions

Markov Logic Network (MLN) combines two aspects that help depict logic problems: first-order logic and weights. Together with a set of facts that represent the domain of each relation, it can present a real world application or situation.

Our main goal is creating an algorithm that returns correct answers. First, the solution must be sound: it can not violate any hard constraints. Secondly, the solution must be optimal: the algorithm must return a solution with the lowest possible cost. The following goal is having a better performance than the other state-of-the-art engines.

ImPROV uses the structure of Horn constraints to perform an initial eager propagation. Secondly, we use an SMT solver (Z3 [7]) to validate the rules throughout the computation. The main difference between ImPROV and the other state-of-the-art algorithms is the use of an incremental core-guided MaxSAT algorithm that reuses information from the previous iterations.

ImPROV's algorithmic structure is split into two distinct sections: the initializations and the main loop. The initializations contain the processing of the input, the conversion of all the rules and the initial eager propagation. The main loop always starts by calling a MaxSAT solver. In our case, we use the *Fu & Malik* implementation in the Open-WBO [24] MaxSAT solver. Next, we validate each rule using Z3 and ground the counterexamples found. At the end of each iteration, if all the rules that are not grounded are satisfied by the current MaxSAT solution, the algorithm returns the current MaxSAT solution. This solution is sound and optimal.

Unlike Tuffy, ImPROV always returns sound and optimal solutions. Regarding ImPROV's performance, it is clear that the bottleneck of the algorithm is the use of an SMT solver to validate the rules. In comparison with the state-of-the-art engines, IPR has a better performance than ImPROV and Tuffy in every instance. ImPROV is better than Tuffy due to the quality of the solution.

As future work, we propose to integrate the use of existential quantifiers in the MLN problems. Regarding the solution, the future work is focused on the SMT solver calls. In order to improve our rule validation method, we have two distinct options: first, we can replace the SMT solver with a different technology altogether; secondly, we can improve how the SMT solver is used. In particular, improving the incrementality in the SMT solver can show great results since the SMT formulas are similar in structure from one iteration to the other. Finally, since the validation of each rule is independent, we can parallelize the computation, i.e. split the validation of the rules into different cores.

References

[1] C. Ansótegui, M. L. Bonet, and J. Levy. Solving (weighted) partial maxsat through satisfiability testing. In O. Kullmann, editor, *Theory and Applications of Satisfiability Testing - SAT 2009, 12th International Conference, SAT 2009, Swansea, UK, June 30 - July 3, 2009. Proceedings*, volume 5584 of *Lecture Notes in Computer Science*, pages 427–440. Springer, 2009. ISBN 978-3-642-02776-5. doi: 10.1007/978-3-642-02777-2_39. URL http://dx.doi.org/10.1007/978-3-642-02777-2_39.

[2] D. L. Berre and A. Parrain. The sat4j library, release 2.2. *JSAT*, 7(2-3):59–6, 2010. URL http://jsat.ewi.tudelft.nl/content/volume7/JSAT7_4_LeBerre.pdf.

[3] A. T. Chaganty, A. Lal, A. V. Nori, and S. K. Rajamani. Combining relational learning with SMT solvers using CEGAR. In N. Sharygina and H. Veith, editors, *Computer Aided Verification - 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings*, volume 8044 of *Lecture Notes in Computer Science*, pages 447–462. Springer, 2013. ISBN 978-3-642-39798-1. doi: 10.1007/978-3-642-39799-8_30. URL http://dx.doi.org/10.1007/978-3-642-39799-8_30.

[4] E. M. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-guided abstraction refinement. In E. A. Emerson and A. P. Sistla, editors, *Computer Aided Verification, 12th International Conference, CAV 2000, Chicago, IL, USA, July 15-19, 2000, Proceedings*, volume 1855 of *Lecture Notes in Computer Science*, pages 154–169. Springer, 2000. ISBN 3-540-67770-4. doi: 10.1007/10722167_15. URL http://dx.doi.org/10.1007/10722167_15.

[5] G. B. Dantzig, D. R. Fulkerson, and S. M. Johnson. Solution of a large-scale traveling-salesman problem. *Operations Research*, 2 (4):393–410, 1954. doi: 10.1287/opre.2.4.393. URL http://dx.doi.org/10.1287/opre.2.4.393.

[6] J. Davies, J. Cho, and F. Bacchus. Using learnt clauses in maxsat. In D. Cohen, editor, *Principles and Practice of Constraint Programming - CP 2010 - 16th International Conference, CP 2010, St. Andrews, Scotland, UK, September 6-10, 2010. Proceedings*, volume 6308 of *Lecture Notes in Computer Science*, pages 176–190. Springer, 2010. ISBN 978-3-642-15395-2. doi: 10.1007/978-3-642-15396-9_17. URL http://dx.doi.org/10.1007/978-3-642-15396-9_17.

[7] L. M. de Moura and N. Bjørner. Z3: an efficient SMT solver. In C. R. Ramakrishnan and J. Rehof, editors, *Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings*, volume 4963 of *Lecture Notes in Computer Science*, pages 337–340. Springer, 2008. ISBN 978-3-540-78799-0. doi: 10.1007/978-3-540-78800-3_24. URL http://dx.doi.org/10.1007/978-3-540-78800-3_24.

[8] N. Eén and N. Sörensson. An extensible sat-solver. In E. Giunchiglia and A. Tacchella, editors, *Theory and Applications of Satisfiability Testing, 6th International Conference, SAT 2003. Santa Margherita Ligure, Italy, May 5-8, 2003 Selected Revised Papers*, volume 2919 of *Lecture Notes in Computer Science*, pages 502–518. Springer, 2003. ISBN 3-540-20851-8. doi: 10.1007/978-3-540-24605-3_37. URL http://dx.doi.org/10.1007/978-3-540-24605-3_37.

[9] Z. Fu and S. Malik. On solving the partial MAX-SAT problem. In A. Biere and C. P. Gomes, editors, *Theory and Applications of Satisfiability Testing - SAT 2006, 9th International Conference, Seattle, WA, USA, August 12-15, 2006, Proceedings*, volume 4121 of *Lecture Notes in Computer Science*, pages 252–265. Springer, 2006. ISBN 3-540-37206-7. doi: 10.1007/11814948_25. URL http://dx.doi.org/10.1007/11814948_25.

[10] E. Giunchiglia, Y. Lierler, and M. Maratea. Answer set programming based on propositional satisfiability. *J. Autom. Reasoning*, 36(4):345–377, 2006. doi: 10.1007/s10817-006-9033-2. URL http://dx.doi.org/10.1007/s10817-006-9033-2.

[11] F. Heras, J. Larrosa, and A. Oliveras. Minimaxsat: An efficient weighted max-sat solver. *J. Artif. Intell. Res. (JAIR)*, 31:1–32, 2008. doi: 10.1613/jair.2347. URL http://dx.doi.org/10.1613/jair.2347.

[12] H. H. Hoos. An adaptive noise mechanism for walksat. In R. Dechter and R. S. Sutton, editors, *Proceedings of the Eighteenth National Conference on Artificial Intelligence and Fourteenth Conference on Innovative Applications of Artificial Intelligence, July 28 - August 1, 2002, Edmonton, Alberta, Canada.*, pages 655–660. AAAI Press / The MIT Press,

2002. URL http://www.aaai.org/Library/AAAI/2002/aaai02-098.php.

[13] A. Horn. On sentences which are true of direct unions of algebras. *J. Symb. Log.*, 16(1):14–21, 1951. doi: 10.2307/2268661. URL https://doi.org/10.2307/2268661.

[14] H. A. Kautz, B. Selman, and Y. Jiang. A general stochastic approach to solving problems with hard and soft constraints. In D. Du, J. Gu, and P. M. Pardalos, editors, *Satisfiability Problem: Theory and Applications, Proceedings of a DIMACS Workshop, Piscataway, New Jersey, USA, March 11-13, 1996*, volume 35 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 573–586. DIMACS/AMS, 1996. URL http://dimacs.rutgers.edu/Volumes/Vol35.html.

[15] S. Kok, M. Sumner, M. Richardson, P. Singla, H. Poon, D. Lowd, J. Wang, and P. Domingos. The alchemy system for statistical relational {AI}. 2009.

[16] M. Koshimura, T. Zhang, H. Fujita, and R. Hasegawa. Qmaxsat: A partial max-sat solver. *JSAT*, 8(1/2):95–100, 2012. URL http://jsat.ewi.tudelft.nl/content/volume8/JSAT8_6_Koshimura.pdf.

[17] J. Larrosa, F. Heras, and S. de Givry. A logical approach to efficient max-sat solving. *Artif. Intell.*, 172(2-3):204–233, 2008. doi: 10.1016/j.artint.2007.05.006. URL http://dx.doi.org/10.1016/j.artint.2007.05.006.

[18] R. Mangal, X. Zhang, A. V. Nori, and M. Naik. A user-guided approach to program analysis. In E. D. Nitto, M. Harman, and P. Heymans, editors, *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2015, Bergamo, Italy, August 30 - September 4, 2015*, pages 462–473. ACM, 2015. ISBN 978-1-4503-3675-8. doi: 10.1145/2786805.2786851. URL http://doi.acm.org/10.1145/2786805.2786851.

[19] R. Mangal, X. Zhang, A. Kamath, A. V. Nori, and M. Naik. Scaling relational inference using proofs and refutations. In D. Schuurmans and M. P. Wellman, editors, *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA.*, pages 3278–3286. AAAI Press, 2016. URL http://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/view/12466.

[20] V. M. Manquinho, J. P. M. Silva, and J. Planes. Algorithms for weighted boolean optimization. In O. Kullmann, editor, *Theory and Applications of Satisfiability Testing - SAT 2009, 12th International Conference, SAT 2009, Swansea, UK, June 30 - July 3, 2009. Proceedings*, volume 5584 of *Lecture Notes in Computer Science*, pages 495–508. Springer, 2009. ISBN 978-3-642-02776-5. doi: 10.1007/978-3-642-02777-2_45. URL http://dx.doi.org/10.1007/978-3-642-02777-2_45.

[21] J. Marques-Silva and V. M. Manquinho. Towards more effective unsatisfiability-based maximum satisfiability algorithms. In H. K. Büning and X. Zhao, editors, *Theory and Applications of Satisfiability Testing - SAT 2008, 11th International Conference, SAT 2008, Guangzhou, China, May 12-15, 2008. Proceedings*, volume 4996 of *Lecture Notes in Computer Science*, pages 225–230. Springer, 2008. ISBN 978-3-540-79718-0. doi: 10.1007/978-3-540-79719-7_21. URL http://dx.doi.org/10.1007/978-3-540-79719-7_21.

[22] J. Marques-Silva and J. Planes. On using unsatisfiability for solving maximum satisfiability. *CoRR*, abs/0712.1097, 2007. URL http://arxiv.org/abs/0712.1097.

[23] R. Martins, S. Joshi, V. M. Manquinho, and I. Lynce. Incremental cardinality constraints for maxsat. In B. O'Sullivan, editor, *Principles and Practice of Constraint Programming - 20th International Conference, CP 2014, Lyon, France, September 8-12, 2014. Proceedings*, volume 8656 of *Lecture Notes in Computer Science*, pages 531–548. Springer, 2014. ISBN 978-3-319-10427-0. doi: 10.1007/978-3-319-10428-7_39. URL https://doi.org/10.1007/978-3-319-10428-7_39.

[24] R. Martins, V. M. Manquinho, and I. Lynce. Open-wbo: A modular maxsat solver,. In C. Sinz and U. Egly, editors, *Theory and Applications of Satisfiability Testing - SAT 2014 - 17th International Conference, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 14-17, 2014. Proceedings*, volume 8561 of *Lecture Notes in Computer Science*, pages 438–445. Springer, 2014. ISBN 978-3-319-09283-6. doi: 10.1007/978-3-319-09284-3_33. URL https://doi.org/10.1007/978-3-319-09284-3_33.

[25] K. L. McMillan. Interpolation and sat-based model checking. In W. A. H. Jr. and F. Somenzi, editors, *Computer Aided Verification, 15th International Conference, CAV*

2003, Boulder, CO, USA, July 8-12, 2003, Proceedings, volume 2725 of *Lecture Notes in Computer Science*, pages 1–13. Springer, 2003. ISBN 3-540-40524-0. doi: 10.1007/978-3-540-45069-6_1. URL http://dx.doi.org/10.1007/978-3-540-45069-6_1.

[26] A. Morgado, F. Heras, M. H. Liffiton, J. Planes, and J. Marques-Silva. Iterative and core-guided maxsat solving: A survey and assessment. *Constraints*, 18(4):478–534, 2013. doi: 10.1007/s10601-013-9146-2. URL http://dx.doi.org/10.1007/s10601-013-9146-2.

[27] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient SAT solver. In *Proceedings of the 38th Design Automation Conference, DAC 2001, Las Vegas, NV, USA, June 18-22, 2001*, pages 530–535. ACM, 2001. ISBN 1-58113-297-2. doi: 10.1145/378239.379017. URL http://doi.acm.org/10.1145/378239.379017.

[28] H. B. Newcombe, J. M. Kennedy, S. Axford, and A. P. James. Automatic linkage of vital records. *Science*, 130(3381):954–959, 1959.

[29] F. Niu, C. Ré, A. Doan, and J. W. Shavlik. Tuffy: Scaling up statistical inference in markov logic networks using an RDBMS. *PVLDB*, 4(6):373–384, 2011. URL http://www.vldb.org/pvldb/vol4/p373-niu.pdf.

[30] E. Oikarinen and M. Järvisalo. Max-asp: Maximum satisfiability of answer set programs. In E. Erdem, F. Lin, and T. Schaub, editors, *Logic Programming and Nonmonotonic Reasoning, 10th International Conference, LPNMR 2009, Potsdam, Germany, September 14-18, 2009. Proceedings*, volume 5753 of *Lecture Notes in Computer Science*, pages 236–249. Springer, 2009. ISBN 978-3-642-04237-9. doi: 10.1007/978-3-642-04238-6_21. URL http://dx.doi.org/10.1007/978-3-642-04238-6_21.

[31] H. Poon and P. M. Domingos. Joint inference in information extraction. In *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence, July 22-26, 2007, Vancouver, British Columbia, Canada*, pages 913–918. AAAI Press, 2007. ISBN 978-1-57735-323-2. URL http://www.aaai.org/Library/AAAI/2007/aaai07-145.php.

[32] H. Poon, P. M. Domingos, and M. Sumner. A general method for reducing the complexity of relational inference and its application to MCMC. In D. Fox and C. P. Gomes, editors, *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, AAAI 2008, Chicago, Illinois, USA, July 13-17, 2008*, pages 1075–1080. AAAI Press, 2008. ISBN 978-1-57735-368-3. URL http://www.aaai.org/Library/AAAI/2008/aaai08-170.php.

[33] M. Richardson and P. M. Domingos. Markov logic networks. *Machine Learning*, 62(1-2):107–136, 2006. doi: 10.1007/s10994-006-5833-1. URL http://dx.doi.org/10.1007/s10994-006-5833-1.

[34] S. Riedel. Improving the accuracy and efficiency of MAP inference for markov logic. In D. A. McAllester and P. Myllymäki, editors, *UAI 2008, Proceedings of the 24th Conference in Uncertainty in Artificial Intelligence, Helsinki, Finland, July 9-12, 2008*, pages 468–475. AUAI Press, 2008. ISBN 0-9749039-4-9. URL https://dslpitt.org/uai/displayArticleDetails.jsp?mmnu=1&smnu=2&article_id=1966&proceeding_id=24.

[35] S. Riedel. Cutting plane map inference for markov logic. 2009.

[36] B. Selman, H. J. Levesque, and D. G. Mitchell. A new method for solving hard satisfiability problems. In W. R. Swartout, editor, *Proceedings of the 10th National Conference on Artificial Intelligence. San Jose, CA, July 12-16, 1992.*, pages 440–446. AAAI Press / The MIT Press, 1992. ISBN 0-262-51063-4. URL http://www.aaai.org/Library/AAAI/1992/aaai92-068.php.

[37] B. Selman, H. A. Kautz, and B. Cohen. Noise strategies for improving local search. In B. Hayes-Roth and R. E. Korf, editors, *Proceedings of the 12th National Conference on Artificial Intelligence, Seattle, WA, USA, July 31 - August 4, 1994, Volume 1.*, pages 337–343. AAAI Press / The MIT Press, 1994. ISBN 0-262-61102-3. URL http://www.aaai.org/Library/AAAI/1994/aaai94-051.php.

[38] X. Si, X. Zhang, V. M. Manquinho, M. Janota, A. Ignatiev, and M. Naik. On incremental core-guided maxsat solving. In M. Rueher, editor, *Principles and Practice of Constraint Programming - 22nd International Conference, CP 2016, Toulouse, France, September 5-9, 2016, Proceedings*, volume 9892 of *Lecture Notes in Computer Science*, pages 473–482. Springer, 2016. ISBN 978-3-319-44952-4. doi: 10.1007/978-3-319-44953-1_30. URL http://dx.doi.org/10.1007/978-3-319-44953-1_30.

[39] J. P. M. Silva and K. A. Sakallah. GRASP - a new search algorithm for satisfiability. In *ICCAD*, pages 220–227, 1996. doi: 10.1109/ICCAD.1996.569607. URL http://dx.doi.org/10.1109/ICCAD.1996.569607.

[40] P. Singla and P. M. Domingos. Entity resolution with markov logic. In *Proceedings of the 6th IEEE International Conference on Data Mining (ICDM 2006), 18-22 December 2006, Hong Kong, China*, pages 572–582. IEEE Computer Society, 2006. ISBN 0-7695-2701-9. doi: 10.1109/ICDM.2006.65. URL http://dx.doi.org/10.1109/ICDM.2006.65.

[41] W. L. Winston, M. Venkataramanan, and J. B. Goldberg. *Introduction to mathematical programming*, volume 1. Thomson/Brooks/Cole Duxbury; Pacific Grove, CA, 2003.