Integer Linear Programming strategies for first-order weighted MAX-SAT

James Cussens, University of York

York, 2015-10-21

A SAT instance

Can we satisfy these 4 clauses?

$$\neg a \vee b$$

$$\neg b \vee c \vee d$$

$$\neg c \vee \neg d$$

$$a$$

A SAT instance

Can we satisfy these 4 clauses?

$$\neg a \vee b$$
$$\neg b \vee c \vee d$$
$$\neg c \vee \neg d$$
$$a$$

Yes: $a = b = c = 1,\ d = 0$

A SAT instance

Can we satisfy these 4 clauses?

$$\neg a \vee b$$
$$\neg b \vee c \vee d$$
$$\neg c \vee \neg d$$
$$a$$

Yes: $a = b = c = 1$, $d = 0$
Adding $\neg a \vee d$ and $\neg c$ makes it unsatisfiable.

A weighted MAX-SAT instance

A cost (possibly infinite) for each broken clause

$$\begin{aligned}
\infty &: \neg a \vee b \\
1 &: \neg b \vee c \vee d \\
4 &: \neg c \vee \neg d \\
3 &: a \\
\infty &: \neg a \vee d \\
20 &: \neg c
\end{aligned}$$

Goal is to find an assignment with minimal cost

ILP encoding for SAT

$\neg a \vee b$	$(1 - a) + b \geq 1$
$\neg b \vee c \vee d$	$(1 - b) + c \geq 1$
$\neg c \vee \neg d$	$(1 - c) + d \geq 1$
a	$a \geq 1$

ILP encoding for weighted MAX-SAT

$\infty : \neg a \vee b$	$(1 - a) + b \geq 1$
$1 : \neg b \vee c \vee d$	$(1 - b) + c + x_1 \geq 1$
$4 : \neg c \vee \neg d$	$(1 - c) + d + x_2 \geq 1$
$3 : a$	$a + x_3 \geq 1$
$\infty : \neg a \vee d$	$(1 - a) + d \geq 1$
$20 : \neg c$	$(1 - c) + x_4 \geq 1$

Minimise $x_1 + 4x_2 + 3x_3 + 20x_4$ subject to the 6 (hard) constraints

First-order logic

- A first-order language has a finite number of predicate symbols and a finite number of function symbols.
- Suppose we have the following predicate symbols: $e/1, o/1, lt/2$,
- And these two function symbols 0 (a constant) and $s/1$.

Here are some formulae in that language:

$$e(0)$$
$$\forall X : e(X) \vee e(s(X))$$
$$\forall X : lt(X, s(X))$$
$$\forall X, Y, Z : \neg lt(X, Y) \vee \neg lt(Y, Z) \vee lt(X, Z)$$

First-order models

$$e(s(0))$$
$$\forall X : e(X) \vee e(s(X))$$
$$\forall X : lt(X, s(X))$$
$$\forall X, Y, Z : \neg lt(X, Y) \vee \neg lt(Y, Z) \vee lt(X, Z)$$

A first-order interpretation defines a truth-value for each ground atomic formulae in the language.

- M_1: All ground atoms are set to TRUE.
- M_2: True ground atoms are $\{e(0), e(s(s(0)), \ldots, lt(0, s(0)), lt(0, s(s(0))), \ldots\}$.

These two different interpretations are both *models* of the set of formulae.

First-order weighted MAX-SAT

Assuming we have defined some first-order language . . .

$$
\begin{aligned}
\infty &: \forall X, Y : \neg a(X, Y) \vee b(X, Y) \\
1 &: \forall X, Y, Z : \neg b(X, Y) \vee c(Y, Z) \vee d(Z) \\
4 &: \forall X, Y : \neg c(X, Y) \vee \neg d(X) \\
3 &: \forall X, Y : a(X, Y) \\
\infty &: \forall X, Y : \neg a(X, Y) \vee d(Y) \\
20 &: \forall X, Y : \neg c(X, Y)
\end{aligned}
$$

- For a given first-order interpretation, there is a cost for each ground instance of a clause that is broken in that interpretation.
- Total cost is the sum of these costs.

First-order weighted MAX-SAT without function symbols

- If a first-order language has no function symbols apart from constants, then it has only a finite number of ground atoms.
- So one has the option of:
  1. Treating each ground atomic formula as a propositional symbol
  2. Replacing each first-order clause by its set of ground instances
  3. Solving using a propositional weighted MAXSAT solver.

$$\infty : \forall X, Y : \neg a(X, Y) \vee b(X, Y)$$
$$1 : \forall X, Y, Z : \neg b(X, Y) \vee c(Y, Z) \vee d(Z)$$
$$4 : \forall X, Y : \neg c(X, Y) \vee \neg d(X)$$
$$3 : \forall X, Y : a(X, Y)$$
$$\infty : \forall X, Y : \neg a(X, Y) \vee d(Y)$$
$$20 : \forall X, Y : \neg c(X, Y)$$

ILP encoding for first-order weighted MAX-SAT

$$\begin{aligned}
\forall X, Y &: [1 - a(X, Y)] + b(X, Y) \geq 1 \\
\forall X, Y, Z &: [1 - b(X, Y)] + c(Y, Z) + d(Z) + x_1(X, Y, Z) \geq 1 \\
\forall X, Y &: [1 - c(X, Y)] + [1 - d(X)] + x_2(X, Y) \geq 1 \\
\forall X, Y &: a(X, Y) + x_3(X, Y) \geq 1 \\
\forall X, Y &: [1 - a(X, Y)] + d(Y) \geq 1 \\
\forall X, Y &: [1 - c(X, Y)] + x_4(X, Y) \geq 1
\end{aligned}$$

Minimise $\sum_{X,Y,Z} x_1(X, Y, Z) + 4 \sum_{X,Y} x_2(X, Y) + 3 \sum_{X,Y} x_3(X, Y) + 20 \sum_{X,Y} x_4(X, Y)$ subject to the 6 (hard) constraints

Cutting plane approach

1. Solve a relaxed problem with no constraints.
2. Then add ground clauses (cutting planes) which are violated by that solution
3. Repeat

- The hope is that only a small number of ground instances are 'necessary'.
- In mfoilp (my system) we search for cutting planes as soon as we have solved the *linear relaxation* of the current problem.
- In CPI [Rie08] and RockIT [NNS13] an integer solution (perhaps not an optimal one) must be found before cutting planes are sought.

Cutting plane algorithm

- mfoilp uses a depth-first search for a ground instance of a first-order clause that is violated by the current solution.
- This is implemented in Mercury, a logic programming language, so we get the depth-first search 'for free'.

Cutting plane algorithm implementation

```
clause("fo3") --> insol(smokes(X)), neglit(smokes(X)),
  insol(friends(X,Y)), neglit(friends(X,Y)), {X @<  Y},
  poslit(smokes(Y)), poslit(cb2(X,Y)).

% use this to generate atoms for negative literals
insol(Atom,In,In) :- In  = clause_cut(Sol,_,_,_),
                map.member(Sol,Atom,_).

poslit(Atom, clause_cut(Sol,ValIn,NegIn,PosIn),
     clause_cut(Sol,ValOut,NegIn,[Atom|PosIn])) :-
ValOut = ValIn+solval(Sol,Atom), ValOut < 1.0.
```

Column (variable) generation

- Each ground atom (including those representing that a ground clause has been 'broken') corresponds to an ILP binary variable.
- That's a lot of variables, possibly infinitely many so.
- Rather than create them all at the start they are created on demand.
- Old version of mfoilp: Separate processes for cutting plane generation and column generation
- Current version of mfoilp: If an ILP variable (= ground atomic formula) appears in a cutting plane (ground clause) then create it immediately.

Implementation

- mfoilp is implemented in C and Mercury and uses the SCIP library for ILP, and CPLEX (to solve the linear relaxations).
- A problem instance is defined as a (Mercury) logic program which is compiled (if not already) before solving begins.
- The relevant first-order language is defined by defining a Mercury type called `atom`. Function symbols are allowed in the language.
- SCIP ILP variables and constraints correspond to ground terms in the Mercury program.
- Time for some examples.

Acknowledgements

This work was supported by a Senior Postdoctoral Fellowship SF/14/008 from KU Leuven and by UK NC3RS grant NC/K001264/1.


Jan Noessner, Mathias Niepert, and Heiner Stuckenshmidt.
Rockit: Exploiting parallelism and symmetry for MAP inference in statistical relational models.
Arkiv 1304.4379, April 2013.


Sebastian Riedel.
Improving the accuracy and efficiency of MAP inference for Markov logic.
In *Proceedings of the Twenty-Fourth Conference Annual Conference on Uncertainty in Artificial Intelligence (UAI-08)*, pages 468–475, Corvallis, Oregon, 2008. AUAI Press.