Tutorial on MaxSAT and Weighted CSP

George Katsirelos
INRA

19/06/2018

Introduction

- A (somewhat opinionated) tutorial on MaxSAT
- A small application
- Two kinds of solvers, with different performance characteristics, using different techniques
- *Complete solvers*, because incomplete solvers can be spectacularly wrong
- Incomplete runs of complete solvers still provide information
  - Primal and dual bounds

MaxSAT

- $\mathbf{X}$ Boolean variables
- $S \cup \phi$ *soft* and *hard* clauses
- $w : S \rightarrow \mathbb{N}$
- objective: find assignment that
  - satisfies hard clauses
  - minimizes sum of weights of violated soft clauses
- Technically, Weighted Partial MaxSAT

Clauses vs monomials

A clause $x \vee y \vee \overline{z}$ is violated iff the monomial $\overline{xy}z$ evaluates to 1.
We write $\overline{c}$ for the monomial corresponding to clause c

MaxSAT

$$\min \sum_{c \in S} w(c)\overline{c}$$

such that

$$\phi$$

Example: Correlation Clustering

Problem

Given

$G = \langle V, E \rangle$ and $w : E \rightarrow R$, find

$cl : V \rightarrow N$ that minimizes

sum of $\begin{cases} w(uv) \text{ with } cl(u) \neq cl(v), w(uv) > 0 \\ |w(uv)| \text{ with } cl(u) = cl(v), w(uv) < 0 \end{cases}$

- Typically solved with approximations or heuristics
- Variant with side constraints: allow $w(uv) = \infty$ (must-link), $w(uv) = -\infty$ (cannot-link)

Example: Correlation Clustering

Variables x_{ij}: true iff i and j in same cluster
Hard clauses:

$$\begin{aligned} \overline{x}_{ij} \vee \overline{x}_{jk} \vee x_{ik} & \qquad \forall i,j,k \in V \\ x_{ij} & \qquad \forall \text{must-link constraints } ij \\ \overline{x}_{ij} & \qquad \forall \text{cannot-link constraints } ij \end{aligned}$$

Soft clauses:

$$\begin{aligned} ((x_{ij}), w(ij)) & \qquad \forall w(ij) > 0 \\ ((\overline{x}_{ij}), -w(ij)) & \qquad \forall w(ij) < 0 \end{aligned}$$

Weighted CSP

- A particular *dense* special case of MaxSAT
  - or: MaxSAT is a sparse special case of WCSP
- Given a hypergraph $G = \langle V, H \rangle$, a WCSP $\langle G, D, \mathbf{c}, k \rangle$ is the problem of finding a labeling $l : V \to D$

$$\min \sum_{h \in H} c_h(l(h))$$

$$\text{such that}$$

$$c_h(l(h)) < k \qquad \forall h \in H$$

* Includes self edges and empty edge
* $\mathbf{c}$ is a set of cost functions, hence Cost Function Network (CFN)

WCSP $\Leftrightarrow$ MaxSAT

- Variables $x_{ia} \iff l(i) = a$
-

Label each vertex

$$\phi = \bigwedge_{i \in V} \bigvee_{a \in D} x_{ia}$$

Forbid tuples with cost k

$$\wedge \bigwedge (\vee_{i \in h} \overline{x}_{il(i)}) \qquad \forall h \in H, c_h(l(h)) = k$$

Soft clauses for all other tuples

$$S = \{((\vee_{i \in h} \overline{x}_{il(i)}), c_h(l(h))) \quad \mid h \in H, 0 < c_h(l(h)) < k\}$$

- Denseness: each hyperedge generates many clauses

WCSP or MaxSAT?

Rules of thumb

- When the objective is sparse or satisfiability is hard, MaxSAT solvers should be better
- In certain problems with a dense objective, WCSP solvers are much better
- Exceptions abound
- Branch-and-bound MaxSAT solvers best in some kinds of problems (Max-Cut)

Solving WCSP

Solving WCSP

- Branch-and-bound
- Many preprocessing techniques, heuristics, etc
- Here we are interested in lower bounds

WCSP lower bound

$$\min_{l} \sum_{h \in H} c_h(l(h)) \geq \sum_{h \in H} \min_{l} c_h(l(h))$$

Reparameterization

Equivalence

$P \equiv P'$ if all assignments have the same cost

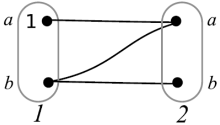
$\textsc{Move}(c_1, c_2, \mathbf{x}, \alpha)$

- Shifts α units of cost between c_1 and c_2 on the common assignment $\mathbf{x}$
- Shift direction: sign of α.
- α constrained: no negative costs!

$\Rightarrow$ $\textsc{Move}$ preserves equivalence

$\Rightarrow$ All equivalent subproblem with the same structure can be generated by a sequence of $\textsc{Move}$s

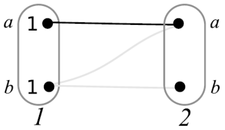
Reparameterization

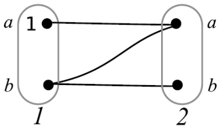


Reparameterization

$$\textsc{Move}\{1,2\},\{1\},b,1$$

$$\leftarrow$$

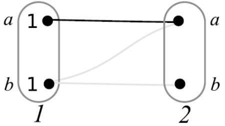


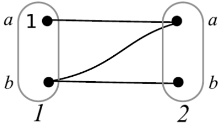


Reparameterization

$$\textsc{Move}\{1,2\},\{1\},b,1$$

$\leftarrow$



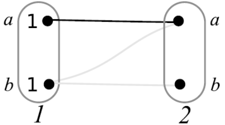


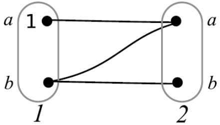
$\Downarrow$ $\textsc{Move}(\{1\},\varnothing,[],1)$

Reparameterization

$$\textsc{Move}\{1,2\},\{1\},b,1$$

$$\leftarrow$$





$\Downarrow$ $\textsc{Move}(\{1\},\varnothing,[],1)$

$c_\varnothing = 1$

Finding reparameterizations

- Each variable has at least one 0-cost value, supported by at least one 0-cost tuple in each constraint
- When does the current lower bound match the actual optimum?
  - $\Rightarrow$ When the 0-cost values can be used to construct a 0-cost solution
  - $\Leftarrow$ When they are inconsistent we can increase the lower bound
- $Bool(P)$: a (hard!) CSP that contains only the zero-cost subset of the WCSP P

VAC

- Iteratively construct $Bool(P)$
  - If arc inconsistent, increase lower bound by reparameterization
  - If arc consistent, finish
- $Bool(P)$ changes non-monotonically after each reparameterization
- Each inconsistent $Bool(P)$ corresponds to an inconsistent subset of the original WCSP P

From WCSP to MaxSAT

We generalize from arc inconsistent subsets to arrive at MaxSAT solving techniques

Solving MaxSAT

Minimal Correction Sets

- $F \setminus C$ is satisfiable, no larger subset of F is
- C: MCS
- $F \setminus C$: Maximal Satisfiable Subset (MSS)
- In the presence of hard clauses: $H \cup (S \setminus C)$ is satisfiable
- A *maximal* solution of a MaxSAT instance

Minimal Correction Sets

- $F \setminus C$ is satisfiable, no larger subset of F is
- C: MCS
- $F \setminus C$: Maximal Satisfiable Subset (MSS)
- In the presence of hard clauses: $H \cup (S \setminus C)$ is satisfiable
- A *maximal* solution of a MaxSAT instance

$$(x_1) \quad (x_2) \quad (x_3)$$
$$(\overline{x}_1 \vee \overline{x}_2) \quad (\overline{x}_1 \vee \overline{x}_3) \quad (\overline{x}_2 \vee \overline{x}_3)$$

Minimal Unsatisfiable Sets

- $U \subseteq F$ is unsatisfiable, no smaller subset of F is
- In the presence of hard clauses: $H \cup U$ is unsatisfiable
- *Also called minimal cores*

Minimal Unsatisfiable Sets

- $U \subseteq F$ is unsatisfiable, no smaller subset of F is
- In the presence of hard clauses: $H \cup U$ is unsatisfiable
- *Also called minimal cores*

$$
\begin{array}{ccc}
(x_1) & (x_2) & (x_3) \\
(\overline{x}_1 \vee \overline{x}_2) & (\overline{x}_1 \vee \overline{x}_3) & (\overline{x}_2 \vee \overline{x}_3)
\end{array}
$$

Hitting set duality

(x_1) (x_2) (x_3)
$(\overline{x}_1 \vee \overline{x}_2)$ $(\overline{x}_1 \vee \overline{x}_3)$ $(\overline{x}_2 \vee \overline{x}_3)$

(x_1) (x_2) (x_3)
$(\overline{x}_1 \vee \overline{x}_2)$ $(\overline{x}_1 \vee \overline{x}_3)$ $(\overline{x}_2 \vee \overline{x}_3)$

(x_1) (x_2) (x_3)
$(\overline{x}_1 \vee \overline{x}_2)$ $(\overline{x}_1 \vee \overline{x}_3)$ $(\overline{x}_2 \vee \overline{x}_3)$

Hitting set duality

(x_1)	(x_2)	(x_3)
$(\overline{x}_1 \vee \overline{x}_2)$	$(\overline{x}_1 \vee \overline{x}_3)$	$(\overline{x}_2 \vee \overline{x}_3)$

(x_1)	(x_2)	(x_3)
$(\overline{x}_1 \vee \overline{x}_2)$	$(\overline{x}_1 \vee \overline{x}_3)$	$(\overline{x}_2 \vee \overline{x}_3)$

(x_1)	(x_2)	(x_3)
$(\overline{x}_1 \vee \overline{x}_2)$	$(\overline{x}_1 \vee \overline{x}_3)$	$(\overline{x}_2 \vee \overline{x}_3)$

$(x_1), (\overline{x}_1, \overline{x}_2)$ not an MCS

Hitting set duality

$$(x_1) \quad (x_2) \quad (x_3)$$
$$(\overline{x}_1 \vee \overline{x}_2) \quad (\overline{x}_1 \vee \overline{x}_3) \quad (\overline{x}_2 \vee \overline{x}_3)$$

$$(x_1) \quad (x_2) \quad (x_3)$$
$$(\overline{x}_1 \vee \overline{x}_2) \quad (\overline{x}_1 \vee \overline{x}_3) \quad (\overline{x}_2 \vee \overline{x}_3)$$

$$(x_1) \quad (x_2) \quad (x_3)$$
$$(\overline{x}_1 \vee \overline{x}_2) \quad (\overline{x}_1 \vee \overline{x}_3) \quad (\overline{x}_2 \vee \overline{x}_3)$$

$(x_2), (\overline{x}_1, \overline{x}_3)$ an MCS

Hitting set duality

- Every (minimal) CS is a hitting set of all (minimal) USes
- Every (minimal) US is a hitting set of all (minimal) CSes

Algorithms

- Most algorithms exploit cores
  - sequence-of-SAT
  - branch and bound not competitive
- Most algorithms are dual: compute a lower bound and improve it until we reach SAT
  - But in fact they are anytime: core computation entails MCS computation, so they produce primal bounds as well
  - But not primal-dual

Hitting set based algorithms

- MCS $\equiv$ solution means minimum MCS $\equiv$ minimum solution
- MCS $\Leftrightarrow$ MUS duality means minimum MCS $\equiv$ minimum hitting set of all MUSes
- Minimum HS of *known* MUSes is a relaxation
- If minimum HS is a CS, relaxation is tight

$\Rightarrow$ Generate MUSes until minimum HS is a CS

Hitting set based algorithms: MaxHS

- A solver that solver minimum HS with ILP
- Optimizes communication between two sides
- One of the best in recent years

Core-guided algorithms

- Use core to transform the formula until it is satisfiable
- Each transformation increases the lower bound

Opinion

All maxsat algorithms are hitting-set based

Core-guided algorithms

Framework for presenting such algorithms

- Each core of the transformed formula corresponds to a set of cores of the original formula
- C_i: cores of the original formula accumulated after iteration i
- LB_i: bound computed by algorithm after iteration i
- HS_i: optimum of hitting set problem over $\cup_{k=1 \dots i} C_k$

Core-guided algorithms

- First algorithm: PM1 for unweighted MaxSAT only
- WPM1 generalized PM1 to weighted MaxSAT
- Many subsequent solvers improve on how WPM1 transforms the formula

PM1

1. Solve SAT formula $H \cup S$
2. If SAT, report solution
3. If UNSAT,
    1. extract core
    2. relax all clauses in core with extra var b_i
    3. add cardinality constraint $\sum b_i = 1$ to H

WPM1

Handles soft clauses with non-unit weight by *cloning*

$$\{(c, w_1 + w_2)\} \equiv \{(c, w_1), (c, w_2)\}$$

WPM1 example

Initial soft clauses $(c_1, 30), (c_2, 30), (c_3, 40), (c_4, 60)$

Core	Transformation
$\{(c_1, 30), (c_3, 40)\}$	$(c_1 \vee b_1^1, 30), (c_3 \vee b_3^1, 30), (c_3, 10)\}$ $b_1^1 + b_3^1 = 1$
$\{(c_2, 30), (c_4, 60)\}$	$(c_2 \vee b_2^2, 30), (c_4 \vee b_4^3, 30), (c_4, 30)\}$ $b_2^2 + b_4^2 = 1$
$\{(c_1 \vee b_1^1, 30), (c_2 \vee b_2^2, 30),$ $(c_3 \vee b_3^1, 30), (c_4 \vee b_4^2, 30)\}$	$(c_1 \vee b_1^1 \vee b_1^3, 30), (c_2 \vee b_2^2 \vee b_2^3, 30),$ $(c_3 \vee b_3^1 \vee b_3^3, 30), (c_4 \vee b_4^2 \vee b_4^3, 30)$ $b_1^3 + b_2^3 + b_3^3 + b_4^3 = 1$
$\{(c_3, 10), (c_4, 30)\}$	$(c_3 \vee b_3^4, 10), (c_4 \vee b_4^4, 10), (c_4, 20)\}$ $b_3^4 + b_4^2 = 1$

WPM1 cores

- Each core of PM1 is a compact representation of a set of cores of the original instance
- These cores can be generated as solutions of a linear system
- Exponentially many

WPM1 bounds

We have $WPM1_i < HS_i$

- Redundant discovery of cores
- Must iterate more after enough cores have been found to prove the optimum bound

PMRES

- A max resolution solver
- Among the state of the art

PMRES: Max-resolution

- A complete calculus for (weighted-, partial-) MaxSAT
- Here we use only a specific instantiation

$$\frac{\begin{array}{c} A \vee x, 1 \\ \overline{x}, 1 \end{array}}{\begin{array}{c} A, 1 \\ \overline{A} \vee \overline{x}, 1 \end{array}}$$

PMRES: Clause reification

- Given a soft clause (c, w), we can rewrite as

$$z \iff C$$
$$(z, w)$$

PMRES

1. Reify all soft clauses
2. Solve $H \cup S$
3. Extract core
4. Apply max-resolution with all unit soft clauses

Maintains invariant that all soft clauses are unit, hence max-resolution does not blow up

PMRES cores

- Each PMRES core is a compact representation of a set of cores of the original instance
- Generated by performing *variable elimination* of the auxiliary variables
- Exponentially many

PMRES bounds

$PMRES_i = HS_i$

- Perfectly exploits the cores it discovers
- Partially explains the advantage of PMRES over WPM1

Comparison

- Hitting set based solvers separate satisfiability concerns (SAT subsolver) from bounds reasoning (ILP subsolver).
- Core-guided solvers use SAT solvers for both satisfiability reasoning and bound reasoning
  - Should be worse intuitively
  - But often bound reasoning *combined* with SAT reasoning is more efficient

Conclusions

- Many more MaxSAT solvers
  - WPM2, WPM3, OLL, MSCG
  - Branch-and-Bound
- This viewpoint can explain (nearly?) all of them
- Research on maxsat centered on finding more efficient SAT encodings
- Can we exploit this viewpoint to identify better encodings? Build new hybrids?

Q?