Maximum Satisfiability

Ruben Martins
February 14, 2017

Automated Logical Reasoning, University of Texas at Austin
What is Boolean Satisfiability?

- **Fundamental problem** in Computer Science
  - The first problem to be proven NP-Complete
  - Has a wide range of applications

- Formula:
  - \( \varphi = (\neg x_2 \lor \neg x_1) \land (x_2 \lor \neg x_3) \land (x_1) \land (x_3) \)

- Boolean Satisfiability (SAT):
  - Is there an assignment of true or false values to variables such that \( \varphi \) evaluates to true?
Software Package Upgradeability Problem

```
sudo apt-get install bison++
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following extra packages will be installed:
  flex-old
The following packages will be REMOVED:
  bison flex libfl-dev
The following NEW packages will be installed:
  bison++ flex-old
0 upgraded, 2 newly installed, 3 to remove and 334 not upgraded.
Need to get 507 kB of archives.
After this operation, 995 kB disk space will be freed.
Do you want to continue? [Y/n] [Y]
```
Software Package Upgradeability Problem

<table>
<thead>
<tr>
<th>Package</th>
<th>Dependencies</th>
<th>Conflicts</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p_1$</td>
<td>${ p_2 \lor p_3 }$</td>
<td>${ p_4 }$</td>
</tr>
<tr>
<td>$p_2$</td>
<td>${ p_3 }$</td>
<td>{}</td>
</tr>
<tr>
<td>$p_3$</td>
<td>${ p_2 }$</td>
<td>${ p_4 }$</td>
</tr>
<tr>
<td>$p_4$</td>
<td>${ p_2 \land p_3 }$</td>
<td>{}</td>
</tr>
</tbody>
</table>

- Set of packages we want to install: $\{ p_1, p_2, p_3, p_4 \}$
- Each package $p_i$ has a set of **dependencies**:
  - Packages that must be installed for $p_i$ to be installed
- Each package $p_i$ has a set of **conflicts**:
  - Packages that cannot be installed for $p_i$ to be installed
NP Completeness

- Giving up?
  - The problem is NP-hard, so let’s develop heuristics or approximation algorithms.
- No! Current tools can find solutions for very large problems!
How can we encode this problem to Boolean Satisfiability?
Software Package Upgradeability Problem as SAT

Formula $\varphi$:

<table>
<thead>
<tr>
<th>Dependencies</th>
<th>$\neg p_1 \lor p_2 \lor p_3$</th>
<th>$\neg p_2 \lor p_3$</th>
<th>$\neg p_3 \lor p_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conflicts</td>
<td>$\neg p_4 \lor p_2$</td>
<td>$\neg p_4 \lor p_3$</td>
<td>$\neg p_1 \lor \neg p_4$</td>
</tr>
<tr>
<td>Packages</td>
<td>$p_1$</td>
<td>$p_2$</td>
<td>$p_3$</td>
</tr>
</tbody>
</table>

- Formula is unsatisfiable
- We cannot install all packages
- How many packages can we install?
What is Maximum Satisfiability?

- Maximum Satisfiability (MaxSAT):
  - Clauses in the formula are either **soft** or **hard**
  - Hard clauses: **must** be satisfied
    (e.g. conflicts, dependencies)
  - Soft clauses: **desirable** to be satisfied
    (e.g. package installation)

- **Goal**: Maximize number of satisfied soft clauses
How to encode Software Package Upgradeability?

Software Package Upgradeability problem as MaxSAT:

• What are the hard constraints?
  • (Hint) Dependencies, conflicts or installation packages?

• What are the soft constraints?
  • (Hint) Dependencies, conflicts or installation packages?
How to encode Software Package Upgradeability?

Software Package Upgradeability problem as MaxSAT:

- What are the hard constraints?
  - Dependencies and conflicts
- What are the soft constraints?
  - Installation of packages
Software Package Upgradeability Problem as MaxSAT

MaxSAT Formula:

$\varphi_h$ (Hard):

$\neg p_1 \lor p_2 \lor p_3$

$\neg p_2 \lor p_3$

$\neg p_3 \lor p_2$

$\neg p_4 \lor p_2$

$\neg p_4 \lor p_3$

$\neg p_1 \lor \neg p_4$

$\neg p_3 \lor \neg p_4$

$\varphi_s$ (Soft):

$p_1$

$p_2$

$p_3$

$p_4$

- Dependencies and conflicts are encoded as hard clauses
- Installation of packages are encoded as soft clauses
- **Optimal solution** (3 out 4 packages are installed)
What is MaxSAT Complexity?

- Deciding whether $k$ clauses can be satisfied: NP-Complete
  - Input: A CNF formula $\varphi$, a positive integer $k$
  - Question: Is there an assignment that satisfies at least $k$ clauses in $\varphi$?

- MaxSAT is $\text{FP}^{\text{NP}}$-Complete
  - The class of binary relations $f(x, y)$ where given $x$ we can compute $y$ in polynomial time with access to an NP oracle
  - A SAT solver acts as the NP oracle most often in practice

- MaxSAT is hard to approximate (APX-Complete)
  - APX: class of NP optimization problems that:
    - admit a constant-factor approximation algorithm, \textit{but}
    - have no poly-time approximation scheme (unless NP$=$P)
Why is MaxSAT Important?

• Many real-world applications can be encoded to MaxSAT:
  
  • Software package upgradeability

  ![Software package upgradeability icon]

  • Error localization in C code

  ![Error localization icon]

  • Haplotyping with pedigrees

  ![Haplotyping icon]

  • ...

• MaxSAT algorithms are very effective for solving real-word problems
The MaxSAT (r)evolution – Partial MaxSAT

- Best solver can solve 3× more benchmarks than in 2008!
- Better than tools like CPLEX (IBM) and Z3 (Microsoft)!
**The MaxSAT (r)evolution – Partial Weighted MaxSAT**

- Best solver can solve $2.5 \times$ more benchmarks than in 2008!
- Better than tools like CPLEX (IBM) and Z3 (Microsoft)!
• MaxSAT Algorithms:
  • **Upper bound search** on the number of unsatisfied soft clauses
  • **Lower bound search** on the number of unsatisfied soft clauses

• **Partitioning** in MaxSAT:
  • Use the structure of the problem to guide the search

• Using MaxSAT solvers
SAT Solvers

- Formula
- SAT Solver
  - SAT: Satisfying assignment
  - UNSAT: Unsatisfiable subformula
Satisfying assignment

Formula:

\[ x_1 \lor \neg x_1 \lor \neg x_3 \lor x_1 \lor \neg x_3 \lor \neg x_1 \lor x_2 \lor \neg x_3 \]

- Satisfying assignment:
  - Assignment to the variables that evaluates the formula to true
  - \( \mu = \{ x_1 = 1, x_2 = 1, x_3 = 0 \} \)
Unsatisfiable subformula

Formula:

$x_1 \land x_3 \land x_2 \lor \neg x_1 \land \neg x_3 \lor x_1 \land \neg x_2 \lor \neg x_1 \land x_2 \lor \neg x_3$

- Formula is unsatisfiable
- Unsatisfiable subformula (core):
  - $\varphi' \subseteq \varphi$, such that $\varphi'$ is unsatisfiable
MaxSAT Algorithms

• MaxSAT algorithms build on SAT solver technology

• MaxSAT algorithms use constraints not defined in causal form:
  • AtMost1 constraints, $\sum_{j=1}^{n} x_j \leq 1$
  • General cardinality constraints, $\sum_{j=1}^{n} x_j \leq k$
  • Pseudo-Boolean constraints, $\sum_{j=1}^{n} a_j x_j \leq k$

• Efficient encodings to CNF
Sequential counters

- AtMost1 constraints:
  - Clauses/Variables: $O(n)$

- General cardinality constraints:
  - Clauses/Variables: $O(n^k)$

Sequential weighted counters

- Pseudo-Boolean constraints:
  - Clauses/Variables: $O(n^k)$
Upper Bound Search for MaxSAT

Find upper bound $k$ for $\#$unsatisfied soft clauses

SAT Solver

Unsatisfiable subformula → Optimal Solution

Satisfying assignment → Refinement

ϕ → UnsAT

ϕ → ϕ' → UNSAT

ϕ' → ϕ'' → UNSAT

ϕ'' → ϕ → SAT
Can we unsatisfy less than 2 soft clauses? No!

Partial MaxSAT Formula:

\[ \varphi_h : \neg x_2 \lor \neg x_1 \lor x_2 \lor \neg x_3 \]

\[ \varphi_s : x_1 \lor x_3 \lor x_2 \lor \neg x_1 \lor \neg x_3 \lor x_1 \]

\[ \mu = 2 \quad V_R = \{ r_1, r_2, r_3, r_4 \} \]

- **Optimal solution**: given by the last model and corresponds to unsatisfying 2 soft clauses:
  - \[ \nu = \{ x_1 = 1, x_2 = 0, x_3 = 0 \} \]
MaxSAT algorithms

- We have just seen a search on the **upper bound**

- What other kind of search can we do to find an optimal solution?

- What if we start searching from the **lower bound**?
Lower Bound Search for MaxSAT

Can we satisfy all soft clauses?

SAT Solver

SAT

Satisfying assignment

Optimal Solution

Unsatisfiable subformula

Refinement

ϕ

ϕ' ⇝ ϕ''
Can we satisfy all soft clauses but 2? Yes!

Partial MaxSAT Formula:

\[
\varphi_h : \ \neg x_2 \lor \neg x_1 \quad x_2 \lor \neg x_3 \quad \text{CNF} \left( \sum_{r_i \in V_R} r_i \leq 2 \right)
\]

\[
\varphi_s : \ x_1 \lor r_1 \quad x_3 \lor r_2 \quad x_2 \lor \neg x_1 \lor r_3 \quad \neg x_3 \lor x_1 \lor r_4
\]

- Formula is satisfiable:
  - \( \mu = \{ x_1 = 1, x_2 = 0, x_3 = 0, r_1 = 0, r_2 = 1, r_3 = 1, r_4 = 0 \} \)

- Optimal solution unsatisfies 2 soft clauses
Unsatisfiability-based Algorithms

- What are the problems of this algorithm? *(Hint)* Number of relaxation variables? Size of the cardinality constraint? Other?

- We relax all soft clauses!
- The cardinality constraint contain as many literals as we have soft clauses!
- Can we do better?
Partial MaxSAT Formula:

$$\varphi_h : \neg x_2 \lor \neg x_1 \quad x_2 \lor \neg x_3 \quad \text{CNF}(r_1 + r_2 + r_3 + r_4 \leq 2)$$

$$\varphi_s : \quad x_1 \lor r_1 \quad x_3 \lor r_2 \quad x_2 \lor \neg x_1 \lor r_3 \quad \neg x_3 \lor x_2$$

- Formula is satisfiable:
  - $$\mu = \{x_1 = 1, x_2 = 0, x_3 = 0, r_1 = 0, r_2 = 1, r_3 = 1, r_4 = 0\}$$
- Optimal solution unsatisfies 2 soft clauses
Unsatisfiability-based Algorithms

- What are the problems of this algorithm? 
  (Hint) Number of relaxation variables? Size of the cardinality constraint? Other?

- We must translate cardinality constraints into CNF!

- If the number of literals is large than we may generate a very large formula!

- Can we do better?
Partial MaxSAT Formula:

- **Formula is satisfiable**
- **An optimal solution would be:**
  - $\nu = \{x_1 = 1, x_2 = 0, x_3 = 0\}$
- **This assignment unsatisfies 2 soft clauses**
Challenges for Unsatisfiability-based MaxSAT Algorithms

- Unsatisfiable cores found by the SAT solver are not minimal

- Minimizing unsatisfiable cores is computationally hard
Partitioning in MaxSAT

- Partitioning in MaxSAT:
  - Partition the soft clauses into disjoint sets
  - Iteratively increase the size of the MaxSAT formula

Advantages:
- **Easier formulas** for the SAT solver
- **Smaller unsatisfiable cores** at each iteration
Framework for Partitioning-based MaxSAT Algorithms

\[ \varphi_1 \rightarrow \varphi'_1 \cup \varphi_2 \rightarrow \varphi''_1 \cup \varphi'_2 \rightarrow \varphi'''_1 \cup \varphi'_3 \rightarrow \varphi''''_1 \cup \varphi''_2 \cup \varphi'_3 \rightarrow \text{SAT Solver} \rightarrow \text{Satisfying assignment} \rightarrow \text{Optimal Solution} \rightarrow \text{Yes} \]

\[ \varphi_1 \rightarrow \varphi'_1 \cup \varphi_2 \rightarrow \varphi''_1 \cup \varphi'_2 \rightarrow \varphi'''_1 \cup \varphi'_3 \rightarrow \varphi''''_1 \cup \varphi''_2 \cup \varphi'_3 \rightarrow \text{SAT Solver} \rightarrow \text{Satisfying assignment} \rightarrow \text{Optimal Solution} \rightarrow \text{No} \]

\[ \text{Solution may not be optimal!} \rightarrow \text{Refinement} \]

\[ \varphi_1 \rightarrow \varphi'_1 \cup \varphi_2 \rightarrow \varphi''_1 \cup \varphi'_2 \rightarrow \varphi'''_1 \cup \varphi'_3 \rightarrow \varphi''''_1 \cup \varphi''_2 \cup \varphi'_3 \rightarrow \text{SAT Solver} \rightarrow \text{Satisfying assignment} \rightarrow \text{Optimal Solution} \rightarrow \text{Yes} \]
How to Partition Soft Clauses?

- **Graph representation** of the MaxSAT formula:
  - Vertices: Variables
  - Edges: Between variables that appear in the same clause
Graph representations for MaxSAT

- There are many ways to represent MaxSAT as a graph:
  - Clause-Variable Incidence Graph (CVIG) [Martins et al. SAT 2013]
  - Variable Incidence Graph (VIG) [Martins et al. SAT 2013]
  - Hypergraph [Martins et al. SAT 2013]
  - **Resolution Graph** [Neves et al. SAT 2015]
  - ...
MaxSAT Formulas as Resolution-based Graphs

- MaxSAT solvers rely on the identification of **unsatisfiable cores**
- How can we capture sets of clauses that are closely related and are likely to result in unsatisfiable cores?
  - Represent MaxSAT formulas as **resolution graphs**!
  - Resolution graphs are based on the resolution rule
- Example of the resolution rule:

\[
(x_1 \lor x_2) (\neg x_2 \lor x_3) \quad \frac{(x_1 \lor x_3)}{(x_1 \lor x_3)}
\]
MaxSAT Formulas as Resolution-based Graphs

- Vertices: Represent each clause in the graph
- Edges: There is an edge between two vertices if you can apply the **resolution rule** between the corresponding clauses

Hard clauses:
\[ c_1 = x_1 \lor x_2 \]
\[ c_2 = \neg x_2 \lor x_3 \]
\[ c_3 = \neg x_1 \lor \neg x_3 \]

Soft clauses:
\[ c_4 = \neg x_1 \]
\[ c_5 = \neg x_3 \]
The techniques in Open-WBO have been adopted by other state-of-the-art MaxSAT solvers, e.g. MSCG, WPM.
Want to try MaxSAT solving?

- **Java:**
  - **SAT4J**
  - [http://www.sat4j.org/](http://www.sat4j.org/)

- **C++:**
  - **Open-WBO**
  - Winner of multiples tracks in the MaxSAT Competition 2014, 2015 and 2016!
  - [http://sat.inesc-id.pt/open-wbo/](http://sat.inesc-id.pt/open-wbo/)

- **Annual competition:**
  - [http://www.maxsat.udl.cat/](http://www.maxsat.udl.cat/)
  - Modify a solver today and enter this year competition!
Standard Solver Input Format: DIMACS WCNF

- Variables indexed from 1 to n
- Negation: -
  - -3 stands for ¬x₃
- 0: special end-of-line character
- One special header “p”-line:
  p wcnf #vars #clauses top
  - #vars: number of variables
  - #clauses: number of clauses
  - top: “weight” of hard clauses
- Clauses represented as lists of integers
  - Weight is the first number
  - \( (¬x₃ \lor x₁ \lor ¬x₄5) \), weight 2:
    2 -3 1 -45 0
- Clause is hard if weight is equal to top
Example: pointer analysis domain (pa-2.wcnf):

```
p wcnf 17997976 23364255 9223372036854775807
142   -11393180  12091478  0
200   -12496389  -1068725  13170751  0
209   -8854604   -8854942  -8854943  -8253894  9864153  0
174   -9406753   -8105076   11844088  0
200   -10403325  -8104972  12524177  0
142   -11987544  12096893  0
37    -10981341  -10980973  10838652  0
209   -9578314   -9579250  -9579251  -8254733  9578317  0
209   -8868994   -8870298  -8870299  -8254157  8868997  0
209   -9387012   -9387508  -9387509  -8253943  9387015  0
174   -9834074   -8106628  12074710  0
200   -10726788  -8105074  12909526  0
...
9223372036854775807  -13181184  0
9223372036854775807  -13181215  0
...
... truncated 763 MB
```
Example: $ open-wbo pa-2.wcnf

c Open-WBO: a Modular MaxSAT Solver
c Version: MaxSAT Evaluation 2016
c Authors: Ruben Martins, Vasco Manquinho, Ines Lynce
c Contributors: Miguel Neves, Saurabh Joshi, Mikolas Janota
...
c Problem Type: Weighted
c Number of variables: 17,997,976
c Number of hard clauses: 8,237,870
c Number of soft clauses: 15,126,385
c Parse time: 5.60 s
...
o 4699
o 4609
o 143
s OPTIMUM FOUND
c Total time: 361.26 s v 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15...
...17997976
References

Cardinality and Pseudo-Boolean Encodings:


Community Structure:

C. Ansótegui, J. Giráldez-Cru, Jordi Levy. The Community Structure of SAT Formulas. SAT 2012: 410-423

Web pages of interest:

MaxSAT Evaluation: http://www.maxsat.udl.cat/
Open-WBO: http://sat.inesc-id.pt/open-wbo/
SAT4J: http://www.sat4j.org/
SATGraf: https://bitbucket.org/znewsham/satgraf