Lecture 8: Exact Optimization.
Maximum Satisfiability, part I
February 23, 2016
On This Lecture

- Exact optimization
- Maximum satisfiability, part I
Overview

Maximum Satisfiability—MaxSAT

Exact Boolean optimization paradigm

- Builds on the success story of Boolean satisfiability (SAT) solving
- Great recent improvements in practical solver technology
- Expanding range of real-world applications

Offers an alternative e.g. integer programming

- Solvers provide provably optimal solutions
- Propositional logic as the underlying declarative language: especially suited for inherently “very Boolean” optimization problems
Optimization

Most real-world problems involve an optimization component

Examples:

- Find a **shortest** path/plan/execution/... to a goal state
  - Planning, model checking, ...
- Find a **smallest** explanation
  - Debugging, configuration, ...
- Find a **least resource-consuming** schedule
  - Scheduling, logistics, ...
- Find a **most probable** explanation (MAP)
  - Probabilistic inference, ...

High demand for automated approaches to finding good solutions to computationally hard optimization problems
Optimization

Most real-world problems involve an optimization component

Examples:

- Find a **shortest** path/plan/execution/... to a goal state
  - Planning, model checking, ...

- Find a **smallest** explanation
  - Debugging, configuration, ...

- Find a **least resource-consuming** schedule
  - Scheduling, logistics, ...

- Find a **most probable** explanation (MAP)
  - Probabilistic inference, ...

High demand for automated approaches to finding good solutions to computationally hard optimization problems
Importance of Exact Optimization

Giving Up?
“The problem is NP-hard, so let’s develop heuristics / approximation algorithms.”

No!
Benefits of provably optimal solutions:
- Resource savings
  - Money, human resources, time
- Accuracy
- Better approximations
  - by optimally solving simplified problem representations

Key Challenge: Scalability
Exactly solving instances of *NP-hard* optimization problems
Importance of Exact Optimization

Giving Up?
“The problem is NP-hard, so let’s develop heuristics/approximation algorithms.”

No!
Benefits of provably optimal solutions:
- Resource savings
  - Money, human resources, time
- Accuracy
- Better approximations
  - by optimally solving simplified problem representations

Key Challenge: Scalability
Exactly solving instances of NP-hard optimization problems
Constrained Optimization
Declarative approaches to exact optimization

Model + Solve

1. **Modeling:**
   represent the problem declarative in a constraint language
   
   *so that optimal solutions to the constraint model corresponds to optimal solutions of your problem*

2. **Solving:**
   use an generic, exact solver for the constraint language
   
   *to obtain, for any instance of your problem, an optimal solution to the instance*

Important aspects
- Which constraint language to choose — applications-specific
- How to model the problem compactly & “well” (for the solver)
- Which constraint optimization solver to choose
Constrained Optimization
Declarative approaches to exact optimization

Model + Solve

1. **Modeling:**
   represent the problem declarative in a constraint language
   
   *so that optimal solutions to the constraint model corresponds to optimal solutions of your problem*

2. **Solving:**
   use a general, exact solver for the constraint language
   
   *to obtain, for any instance of your problem, an optimal solution to the instance*

**Important aspects**

- Which constraint language to choose — *applications-specific*
- How to model the problem compactly & “well” (for the solver)
- Which constraint optimization solver to choose
Constrained Optimization Paradigms

Mixed Integer-Linear Programming MIP, ILP

- Constraint language:
  Conjunctions of linear inequalities
  \[ \sum_{i=1}^{k} c_i x_i \leq b \]

- Algorithms: e.g. Branch-and-cut w/ Simplex

Normal form: integer domain variables \( x_i \), constants \( c_i, a^i_j, b_j \)

\[
\begin{align*}
\text{Minimize} & \quad \sum_{i=1}^{k} c_i x_i \\
\text{Subject to} & \quad \sum_{i=1}^{k} a^1_i x_i \leq b_1 \\
& \quad \ldots \\
& \quad \sum_{i=1}^{k} a^m_i x_i \leq b_m
\end{align*}
\]
Constrained Optimization Paradigms

Finite-domain Constraint Optimization (COP)
- Constraint language:
  Conjunctions of high-level (global) finite-domain constraints
- Algorithms:
  Depth-first backtracking search, specialized filtering algorithms

Maximum Satisfiability (MaxSAT)
- Constraint language:
  Weighted Boolean combinations of binary variables
- Algorithms:
  Building on state-of-the-art CDCL SAT solvers
  - Learning from conflicts, conflict-driven search
  - Incremental API, providing explanations for unsatisfiability
## Constrained Optimization Paradigms

### Finite-domain Constraint Optimization (COP)
- **Constraint language:** Conjunctions of high-level (global) finite-domain constraints
- **Algorithms:** Depth-first backtracking search, specialized filtering algorithms

### Maximum Satisfiability (MaxSAT)
- **Constraint language:** weighted Boolean combinations of binary variables
- **Algorithms:** building on state-of-the-art CDCL SAT solvers
  - Learning from conflicts, conflict-driven search
  - Incremental API, providing explanations for unsatisfiability
MaxSAT Applications

- probabilistic inference
- design debugging
- maximum quartet consistency
- software package management

Max-Clique
- fault localization
- restoring CSP consistency
- reasoning over bionetworks
- MCS enumeration
- heuristics for cost-optimal planning
- optimal covering arrays
- correlation clustering
- treewidth computation
- Bayesian network structure learning
- causal discovery
- visualization
- model-based diagnosis
- cutting planes for IPs
- argumentation dynamics

[Park, 2002]
[Chen, Safarpour, Veneris, and Marques-Silva, 2009]
[Chen, Safarpour, Marques-Silva, and Veneris, 2010]
[Morgado and Marques-Silva, 2010]
[Argelich, Berre, Lynce, Marques-Silva, and Rapicault, 2010]
[Ignatiev, Janota, and Marques-Silva, 2014]
[Li and Quan, 2010; Fang, Li, Qiao, Feng, and Xu, 2014; Li, Jiang, and Xu, 2015]
[Zhu, Weissenbacher, and Malik, 2011; Jose and Majumdar, 2011]
[Lynce and Marques-Silva, 2011]
[Guerra and Lynce, 2012]
[Morgado, Lifiton, and Marques-Silva, 2012]
[Zhang and Bacchus, 2012]
[Ansótegui, Izquierdo, Manyà, and Torres-Jiménez, 2013]
[Berg and Järvisalo, 2013; Berg and Järvisalo, 2016]
[Berg and Järvisalo, 2014]
[Berg, Järvisalo, and Malone, 2014]
[Hyttinen, Eberhardt, and Järvisalo, 2014]
[Bunte, Järvisalo, Berg, Myllymäki, Peltonen, and Kaski, 2014]
[Marques-Silva, Janota, Ignatiev, and Morgado, 2015]
[Saikko, Malone, and Järvisalo, 2015]
[Wallner, Niskanen, and Järvisalo, 2016]
MaxSAT Applications

Central to the increasing success:
Advances in MaxSAT solver technology

probabilistic inference
design debugging

maximum quartet consistency
software package management

Max-Clique
fault localization
restoring CSP consistency
reasoning over bionetworks
MCS enumeration
heuristics for cost-optimal planning
optimal covering arrays
correlation clustering
treewidth computation
Bayesian network structure learning
causal discovery
visualization
model-based diagnosis
cutting planes for IPs
argumentation dynamics

[Park, 2002]  
[Chen, Safarpour, Veneris, and Marques-Silva, 2009]
[Chen, Safarpour, Marques-Silva, and Veneris, 2010]
[Morgado and Marques-Silva, 2010]
[Argelich, Berre, Lynce, Marques-Silva, and Rapicault, 2010]
[Ignatiev, Janota, and Marques-Silva, 2014]

[Li and Quan, 2010; Fang, Li, Qiao, Feng, and Xu, 2014; Li, Jiang, and Xu, 2015]
[Zhu, Weissenbacher, and Malik, 2011; Jose and Majumdar, 2011]
[Lynce and Marques-Silva, 2011]
[Guerra and Lynce, 2012]
[Morgado, Liffiton, and Marques-Silva, 2012]
[Zhang and Bacchus, 2012]
[Ansótegui, Izquierdo, Manyà, and Torres-Jiménez, 2013]
[Berg and Järvisalo, 2013; Berg and Järvisalo, 2016]
[Berg and Järvisalo, 2014]
[Berg, Järvisalo, and Malone, 2014]
[Hyttinen, Eberhardt, and Järvisalo, 2014]
[Bunte, Järvisalo, Berg, Myllymäki, Peltonen, and Kaski, 2014]
[Marques-Silva, Janota, Ignatiev, and Morgado, 2015]
[Saikko, Malone, and Järvisalo, 2015]
[Wallner, Niskanen, and Järvisalo, 2016]
Basic Concepts
MaxSAT: Basic Definitions

**MaxSAT**

**INPUT:** a set of clauses $F$.  
**TASK:** find $\tau$ s.t. $\sum_{C \in F} \tau(C)$ is maximized.

Find truth assignment that satisfies a maximum number of clauses

This is the standard definition:

- Much studied in theoretical computer science
- Often inconvenient for modeling practical problems.
Central Generalizations of \textbf{MaxSAT}

\begin{itemize}
  \item \textbf{Weighted MaxSAT}
    \begin{itemize}
      \item Each clause $C$ has an associated weight $w_C$
      \item Optimal solutions maximize the sum of \textit{weights} of satisfied clauses
    \end{itemize}
  \item \textbf{Partial MaxSAT}
    \begin{itemize}
      \item Some clauses are deemed \textit{hard}—infinite weights
        \begin{itemize}
          \item Any solution has to satisfy the hard clauses
          \item \Rightarrow Existence of solutions not guaranteed
        \end{itemize}
      \item Clauses with finite weight are \textit{soft}
    \end{itemize}
  \item \textbf{Weighted Partial MaxSAT}
    \begin{itemize}
      \item Hard clauses (partial) + weights on soft clauses (weighted)
    \end{itemize}
\end{itemize}
Generalizations of MaxSAT

Weighted MaxSAT

“Satisfy as many $C_i$ as possible” $==$ “unit cost for not satisfying $C_i$”

More generally:

- Cost function $c : \{C_i\}_{i=1}^n \mapsto \mathbb{N}$
- Find an assignment $\tau$ that minimizes

$$\sum_{i=1}^{n} c(C_i) \cdot (1 - \tau(C_i))$$

Partial MaxSAT

Clause $C$ has weight $c(C) = \infty \Rightarrow$ hard clause
Clause $C$ has finite weight $c(C) < \infty \Rightarrow$ soft clause
**Generalizations of MaxSAT: Weighted Partial MaxSAT**

Weighted + partial

**Alternative, equivalent definition**

**Input:** $F_h, F_s, c$

- $F_h$: a set of *hard* clauses
- $F_s$: a set of *soft* clauses
- A function assigning a weight to each for clause: $c : F_s \rightarrow \mathbb{N}$

**Task:** Find an assignment $\tau$ that

- satisfies all hard clauses in $F_h$
- minimizes

$$\sum_{C \in F_s} c(C) \cdot (1 - \tau(C))$$
Terminology

- **Solution:** an assignment that satisfies all hard clauses
- **Cost of a solution:** the sum of weights of falsified soft clauses
- **Optimal solution:** minimizes cost over all solutions
Example: Encoding shortest paths

Shortest Path

Find shortest path in a grid with horizontal/vertical moves. Travel from S to G without entering blocked squares (black).

Note: Best solved with state-space search

Here: to illustrate MaxSAT encodings
Example: Encoding shortest paths

Shortest Path

Find shortest path in a grid with horizontal/vertical moves. Travel from S to G without entering blocked squares (black).

Note: Best solved with state-space search

Here: to illustrate MaxSAT encodings
MaxSAT: Example

- **Boolean variables**: one for each unblocked grid square \( \{S, G, a, b, \ldots, u\} \): true iff path visits this square.

- **Constraints**:
  - The \( S \) and \( G \) squares must be visited:
    In CNF: unit hard clauses \((S)\) and \((G)\).
  - A soft clause of weight 1 for all other squares:
    In CNF: \((\neg a), (\neg b), \ldots, (\neg u)\) "would prefer not to visit"
MaxSAT: Example

<table>
<thead>
<tr>
<th>n</th>
<th>o</th>
<th>p</th>
<th>q</th>
</tr>
</thead>
<tbody>
<tr>
<td>h</td>
<td>i</td>
<td>j</td>
<td>k</td>
</tr>
<tr>
<td>c</td>
<td>d</td>
<td>e</td>
<td>l</td>
</tr>
<tr>
<td>a</td>
<td>f</td>
<td>t</td>
<td></td>
</tr>
<tr>
<td>S</td>
<td>b</td>
<td>g</td>
<td>m</td>
</tr>
</tbody>
</table>

- **Boolean variables:** one for each unblocked grid square \{S, G, a, b, \ldots, u\}: true iff path visits this square.
- **Constraints:**
  - The S and G squares must be visited:
    - In CNF: unit hard clauses (S) and (G).
  - A soft clause of weight 1 for all other squares:
    - In CNF: \((\neg a), (\neg b), \ldots, (\neg u)\)  
      “would prefer not to visit”
MaxSAT: Example

- The previous clauses minimize the number of visited squares.
- ...however, their MaxSAT solution will only visit S and G!
- Need to force the existence of a path between S and G by additional hard clauses

A way to enforce a path between S and G:

- both S and G must have exactly one visited neighbour
  - Any path starts from S
  - Any path ends at G
- other visited squares must have exactly two visited neighbours
  - One predecessor and one successor on the path
MaxSAT: Example

- The previous clauses minimize the number of visited squares.
- ... however, their MaxSAT solution will only visit S and G!
- Need to force the existence of a path between S and G by additional hard clauses

A way to enforce a path between S and G:

- both S and G must have exactly one visited neighbour
  - Any path starts from S
  - Any path ends at G
- other visited squares must have exactly two visited neighbours
  - One predecessor and one successor on the path
MaxSAT: Example

Constraint 1:

*S and G must have exactly one visited neighbour.*

- For *S*: \( a + b = 1 \)
  - In CNF:
    \[(a \lor b), (\neg a \lor \neg b)\]
- For *G*: \( k + q + r = 1 \)
  - “At least one” in CNF:
    \[(k \lor q \lor r)\]
  - “At most one” in CNF:
    \[\neg k \lor \neg q, \neg k \lor \neg r, \neg q \lor \neg r\]

Disallow pairwise
MaxSAT: Example

Constraint 1:

*S and G must have exactly one visited neighbour.*

- For **S**: \( a + b = 1 \)
  - In CNF: \((a \lor b), (\neg a \lor \neg b)\)

- For **G**: \( k + q + r = 1 \)
  - “At least one” in CNF: \((k \lor q \lor r)\)
  - “At most one” in CNF:
    - \((\neg k \lor \neg q), (\neg k \lor \neg r), (\neg q \lor \neg r)\)
    - disallow pairwise
MaxSAT: Example

Constraint 1:
*S and G must have exactly one visited neighbour.*

- For **S**: \( a + b = 1 \)
  - In CNF: \((a \lor b), (\neg a \lor \neg b)\)
- For **G**: \( k + q + r = 1 \)
  - “At least one” in CNF: \((k \lor q \lor r)\)
  - “At most one” in CNF: \((\neg k \lor \neg q), (\neg k \lor \neg r), (\neg q \lor \neg r)\)
    *disallow pairwise*

![Diagram of grid with S and G marked]
MaxSAT: Example

Constraint 2:
Other visited squares must have exactly two visited neighbours.

- For example, for square e:
  \[ e \rightarrow (d + j + l + f = 2) \]
  Requires encoding the cardinality constraint \( d + j + l + f = 2 \) in CNF.

Encoding Cardinality Constraints in CNF

- An important class of constraints, occur frequently in real-world problems.
  A lot of work on CNF encodings of cardinality constraints recall lecture 5!
MaxSAT: Example

Constraint 2:
Other visited squares must have exactly two visited neighbours

- For example, for square e:
  - Requires encoding the cardinality constraint $d + j + l + f = 2$ in CNF

Encoding Cardinality Constraints in CNF

- An important class of constraints, occur frequently in real-world problems
  - A lot of work on CNF encodings of cardinality constraints
    recall lecture 5!
Properties of the encoding

- Every solution to the hard clauses is a path from S to G that does not pass a blocked square.
- Such a path will falsify one negative soft clause for every square it passes through.
  - **orange path**: assign 14 variables in \{S, a, c, h, \ldots, t, r, G\} to true
  - **MaxSAT solutions**: paths that pass through a minimum number of squares (i.e., is shortest).
    - **green path**: assign 8 variables in \{S, b, g, f, \ldots, k, G\} to true
Case Study: \textsc{MaxSAT}-based Correlation Clustering

[Berg and Järviselö, 2016]
Correlation Clustering

Partitioning data points into clusters based on pair-wise similarity information

- NP-hard optimization problem
- The number of clusters available not fixed
  - Intuitively: objective function under minimization aims at balancing precision and recall
- Several approximation algorithms proposed
  - Approximation guarantees under binary similarity information
  - Semi-definite relaxation, quadratic programming
- Applications in various settings
  - Biosciences, social network analysis, information retrieval, ...
Cost-Optimal Correlation Clustering

<table>
<thead>
<tr>
<th>V</th>
<th>f_1</th>
<th>f_2</th>
<th>f_3</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>v_1</td>
<td>0.5</td>
<td>1</td>
<td>3</td>
<td>...</td>
</tr>
<tr>
<td>v_2</td>
<td>-3</td>
<td>0</td>
<td>-2</td>
<td>...</td>
</tr>
<tr>
<td>v_3</td>
<td>0.7</td>
<td>1</td>
<td>5</td>
<td>...</td>
</tr>
<tr>
<td>v_4</td>
<td>4</td>
<td>1</td>
<td>7</td>
<td>...</td>
</tr>
<tr>
<td>v_5</td>
<td>6</td>
<td>0</td>
<td>10</td>
<td>...</td>
</tr>
</tbody>
</table>

⇒ \( W = \begin{bmatrix}
0 & 1 & 0.7 & 0 & 0.2 \\
1 & 0 & 4 & -7 & -5 \\
0.7 & 4 & 0 & \infty & 0 \\
0 & -7 & \infty & 0 & -3 \\
0.2 & -5 & 0 & -3 & 0
\end{bmatrix} \)

⇒ MAXSAT: encoding + solving

\( \text{SOLUTION CLUSTERING} \)

**INPUT:** a similarity matrix \( W \),

**TASK:** find a cost-optimal correlation clustering, i.e., a function \( cl^*: V \rightarrow \mathbb{N} \) minimizing

\[
\min_{cl: V \rightarrow \mathbb{N}} \sum_{cl(v_i) = cl(v_j), i < j} (\mathcal{I}[-\infty < W(i,j) < 0] \cdot |W(i,j)|) + \\
\sum_{cl(v_i) \neq cl(v_j), i < j} (\mathcal{I}[\infty > W(i,j) > 0] \cdot W(i,j))
\]

where the indicator function \( \mathcal{I}[b] = 1 \) iff the condition \( b \) is true.
Correlation Clustering as an Integer Program

[Ailon, Charikar, and Newman, 2008; Gael and Zhu, 2007]

- Use indicator variables $x_{ij} \in \{0, 1\}$.
- $x_{ij} = 1$ iff $cl(i) = cl(j)$, i.e., points $i$ and $j$ co-clustered

**IP formulation**

\[
\text{Minimize} \quad \sum_{i<j \quad -\infty < W(i,j) < 0} (x_{ij} \cdot |W(i,j)|) - \sum_{i<j \quad \infty > W(i,j) > 0} (x_{ij} \cdot W(i,j))
\]

where

\[x_{ij} + x_{jk} \leq 1 + x_{ik}\quad \text{for all distinct } i, j, k\]

\[x_{ij} = 1\quad \text{for all } W(i,j) = \infty\]

\[x_{ij} = 0\quad \text{for all } W(i,j) = -\infty\]

\[x_{ij} \in \{0, 1\}\quad \text{for all } i, j\]

**Transitivity-based encoding**

$O(n^2)$ variables and $O(n^3)$ constraints

very large
Reformulating the IP as \textbf{MaxSAT}

- Hard clauses encode well-defined clusterings
- Soft clauses encode the object function
- \(O(n^2)\) variables and \(O(n^3)\) clauses.
- Same indicator variables: \(x_{ij} = 1\) iff \(cl(v_i) = cl(v_j)\)

**Hard clauses**

Encoding the linear constraint \(x_{ij} + x_{jk} \leq 1 + x_{ik}\):

- \((x_{ij} \land x_{jk}) \rightarrow x_{ik}\)
  - as clause: \((\neg x_{ij} \lor \neg x_{jk} \lor x_{ik})\)

Encoding \(W(i,j) = \infty\): \((x_{ij})\)

Encoding \(W(i,j) = -\infty\): \((\neg x_{ij})\)

**Soft clauses**

Encode the cost function

- For \(W(i,j) \in (0, \infty)\): \((x_{ij})\) with weight \(W(i,j)\)
- For \(W(i,j) \in (-\infty, 0)\): \((\neg x_{ij})\) with weight \(|W(i,j)|\)
Reformulating the IP as **MaxSAT**

- Hard clauses encode well-defined clusterings
- Soft clauses encode the object function
- \( \mathcal{O}(n^2) \) variables and \( \mathcal{O}(n^3) \) clauses.
- Same indicator variables: \( x_{ij} = 1 \) iff \( cl(v_i) = cl(v_j) \)

### Hard clauses

Encoding the linear constraint \( x_{ij} + x_{jk} \leq 1 + x_{ik} \):

- \((x_{ij} \land x_{jk}) \rightarrow x_{ik}\)
  - as clause: \((\neg x_{ij} \lor \neg x_{jk} \lor x_{ik})\)

Encoding \( W(i, j) = \infty \): \((x_{ij})\)

Encoding \( W(i, j) = -\infty \): \((\neg x_{ij})\)

### Soft clauses

*encode the cost function*

- For \( W(i, j) \in (0, \infty) \): \((x_{ij})\) with weight \( W(i, j) \)
- For \( W(i, j) \in (-\infty, 0) \): \((\neg x_{ij})\) with weight \(|W(i, j)|\)
Reformulating the IP as **MaxSAT**

- Hard clauses encode well-defined clusterings
- Soft clauses encode the object function
- $O(n^2)$ variables and $O(n^3)$ clauses.
- Same indicator variables: $x_{ij} = 1$ iff $cl(v_i) = cl(v_j)$

### Hard clauses

Encoding the linear constraint $x_{ij} + x_{jk} \leq 1 + x_{ik}$:

- $(x_{ij} \land x_{jk}) \rightarrow x_{ik}$
  - as clause: $(\neg x_{ij} \lor \neg x_{jk} \lor x_{ik})$

Encoding $W(i, j) = \infty$: $(x_{ij})$

Encoding $W(i, j) = -\infty$: $(\neg x_{ij})$

### Soft clauses

*encode the cost function*

- For $W(i, j) \in (0, \infty)$: $(x_{ij})$ with weight $W(i, j)$
- For $W(i, j) \in (-\infty, 0)$: $(\neg x_{ij})$ with weight $|W(i, j)|$
Example

\[ W = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & -1 \\ 1 & -1 & 0 \end{bmatrix} \]

- **Hard clauses:**
  \[ \{(\neg x_{12} \lor \neg x_{23} \lor x_{13}), (\neg x_{12} \lor \neg x_{13} \lor x_{23}), (\neg x_{23} \lor \neg x_{13} \lor x_{12})\} \]

- **Soft clauses:**
  \[ \{(x_{12}; 1), (x_{13}; 1), (\neg x_{23}; 1)\} \]
Example

\[ W = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & -1 \\ 1 & -1 & 0 \end{bmatrix} \]

- Hard clauses:

\[ \{ (\neg x_{12} \lor \neg x_{23} \lor x_{13}), (\neg x_{12} \lor \neg x_{13} \lor x_{23}), (\neg x_{23} \lor \neg x_{13} \lor x_{12}) \} \]

- Soft clauses:

\[ \{ (x_{12}; 1), (x_{13}; 1), (\neg x_{23}; 1) \} \]
Example

\[ W = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & -1 \\ 1 & -1 & 0 \end{bmatrix} \]

- Hard clauses:
  \[ \{(\neg x_{12} \lor \neg x_{23} \lor x_{13}), (\neg x_{12} \lor \neg x_{13} \lor x_{23}), (\neg x_{23} \lor \neg x_{13} \lor x_{12})\} \]

- Soft clauses:
  \[ \{(x_{12}; 1), (x_{13}; 1), (\neg x_{23}; 1)\} \]
Example

\[ W = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & -1 \\ 1 & -1 & 0 \end{bmatrix} \]

- Hard clauses:
  \[ \{(\neg x_{12} \lor \neg x_{23} \lor x_{13}), (\neg x_{12} \lor \neg x_{13} \lor x_{23}), (\neg x_{23} \lor \neg x_{13} \lor x_{12})\} \]

- Soft clauses:
  \[ \{(x_{12}; 1), (x_{13}; 1), (\neg x_{23}; 1)\} \]

Example

\[ W = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & -1 \\ 1 & -1 & 0 \end{bmatrix} \]

- Hard clauses:
  \[ \{ (\neg x_{12} \lor \neg x_{23} \lor x_{13}), (\neg x_{12} \lor \neg x_{13} \lor x_{23}), (\neg x_{23} \lor \neg x_{13} \lor x_{12}) \} \]

- Soft clauses:
  \[ \{ (x_{12}; 1), (x_{13}; 1), (\neg x_{23}; 1) \} \]

A More Compact MaxSAT Encoding

Bit-level / log encodings

For representing non-binary variables with large domains

- To represent the value assignment of a variable with domain $D = \{0, \ldots, |D| - 1\}$:
  - use $\log |D|$ Boolean variables $b_1 \ldots b_{\log |D|}$
  - Interpret an assignment to $b_1 \ldots b_{\log D}$ as the bit-representation of a value in $D$.

Does not always pay off due to poor propagation properties!

However, in correlation clustering:

- Domain-size: number of clusters
- Can be up to number of points to be clustered
- For example: the cluster assignment of each of 512 points can be represented with $\log_2 512 = 9$ bits
Log Encoding of Correlation Clustering

Variables
- Cluster assignment of point $i$: variables $b_i^k$ for $k = 1..\log N$.
- $S_{ij} = 1$ iff points $i$ and $j$ are co-clustered
- Auxiliary: $EQ_{ij}^k = 1$ iff $b_i^k = b_j^k$

Hard clauses
- Semantics of $EQ_{ij}^k$: $EQ_{ij}^k \leftrightarrow (b_i^k \leftrightarrow b_j^k)$
- Semantics of $S_{ij}$: $S_{ij} \leftrightarrow (EQ_{ij}^1 \wedge \cdots \wedge EQ_{ij}^{\log N})$
- Encoding $W(i, j) = \infty$: $(S_{ij})$
- Encoding $W(i, j) = -\infty$: $(\neg S_{ij})$

Soft clauses
- For $W(i, j) \in (0, \infty)$: $(S_{ij})$ with weight $W(i, j)$
- For $W(i, j) \in (-\infty, 0)$: $(\neg S_{ij})$ with weight $|W(i, j)|$
Example

\[ W = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & -1 \\ 1 & -1 & 0 \end{bmatrix} \]

- **Hard clauses:**

\[ S_{12} \leftrightarrow (EQ_{12}^1 \land EQ_{12}^2) \]
\[ S_{13} \leftrightarrow (EQ_{13}^1 \land EQ_{13}^2) \]
\[ S_{23} \leftrightarrow (EQ_{23}^1 \land EQ_{23}^2) \]

\[ EQ_{12}^1 \leftrightarrow (b_1^1 \leftrightarrow b_1^2) \]
\[ EQ_{12}^2 \leftrightarrow (b_2^1 \leftrightarrow b_2^2) \]
\[ EQ_{13}^1 \leftrightarrow (b_1^1 \leftrightarrow b_3^1) \]
\[ EQ_{13}^2 \leftrightarrow (b_2^1 \leftrightarrow b_3^2) \]
\[ EQ_{23}^1 \leftrightarrow (b_1^2 \leftrightarrow b_1^3) \]
\[ EQ_{23}^2 \leftrightarrow (b_2^2 \leftrightarrow b_3^2) \]

- **Soft clauses:**

\[ \{(S_{12}; 1), (S_{13}; 1), (\neg S_{23}; 1)\} \]
Example

\[ W = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & -1 \\ 1 & -1 & 0 \end{bmatrix} \]

- **Hard clauses:**

  \[ S_{12} \leftrightarrow (EQ_{12}^1 \land EQ_{12}^2) \]

  \[ S_{13} \leftrightarrow (EQ_{13}^1 \land EQ_{13}^2) \]

  \[ S_{23} \leftrightarrow (EQ_{23}^1 \land EQ_{23}^2) \]

- **Soft clauses:**

  \{ (S_{12}; 1), (S_{13}; 1), (\neg S_{23}; 1) \}
Example

\[ W = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & -1 \\ 1 & -1 & 0 \end{bmatrix} \]

Clustering:
- Points 1, 2 in cluster 1
- Point 3 in cluster 2

- **Hard clauses:**
  - \( S_{12} \leftrightarrow (EQ_{12}^1 \land EQ_{12}^2) \)
  - \( S_{13} \leftrightarrow (EQ_{13}^1 \land EQ_{13}^2) \)
  - \( S_{23} \leftrightarrow (EQ_{23}^1 \land EQ_{23}^2) \)
  - \( EQ_{12}^1 \leftrightarrow (b_1^1 \leftrightarrow b_2^1) \)
  - \( EQ_{12}^2 \leftrightarrow (b_2^1 \leftrightarrow b_2^2) \)
  - \( EQ_{13}^1 \leftrightarrow (b_1^1 \leftrightarrow b_3^1) \)
  - \( EQ_{13}^2 \leftrightarrow (b_2^1 \leftrightarrow b_3^2) \)
  - \( EQ_{23}^1 \leftrightarrow (b_1^2 \leftrightarrow b_3^1) \)
  - \( EQ_{23}^2 \leftrightarrow (b_2^2 \leftrightarrow b_3^2) \)

- **Soft clauses:**
  - \( \{(S_{12}; 1), (S_{13}; 1), (\neg S_{23}; 1)\} \)
Experiments

MaxSAT solver: MaxHS (implicit hitting set approach)

Protein sequencing data: similarity information over amino-acid sequences

Compared with:
- Exact state-of-the-art IP solvers: CPLEX, Gurobi
- Approximation algorithms for correlation clustering: KwickCluster (KC), SDPC (semi-definite relaxation of the IP)
- SCPS: a dedicated spectral clustering algorithms for the specific type of data
Scalability of the Exact Approaches

- Log encoding scales further wrt number of datapoints considered

- Scalability under incomplete similarity information

- (IP does not scale up to the full set of points)
Scalability of the Exact Approaches

- Log encoding scales further wrt number of datapoints considered

- Scalability under incomplete similarity information

- (IP does not scale up to the full set of points)
Quality of Solutions

- Notably better solution costs, esp. on incomplete similarity info
- Consistently better than the data-specific SCPS

- Rand index: typical clustering quality measure
- User knowledge (UK) on a golden clustering: Rand index for MaxSAT goes quickly beyond the others
Quality of Solutions

- Notably better solution costs, esp. on incomplete similarity info
- Consistently better than the data-specific SCPS

Rand index: typical clustering quality measure

user knowledge (UK) on a golden clustering: Rand index for MaxSAT goes quickly beyond the others
MaxSAT allows for compactly encoding various types of high-level finite-domain soft constraints

- Due to Cook-Levin Theorem:
  Any NP constraint can be polynomially represented as clauses

**Basic Idea**

Finite-domain soft constraint $C$ with associated weight $W_C$.

Let $\text{CNF}(C) = \bigwedge_{i=1}^m C_i$ be a CNF encoding of $C$.

Softening $\text{CNF}(C)$ as Weighted Partial MaxSAT:

- Hard clauses: $\bigwedge_{i=1}^m (C_i \lor a)$, where $a$ is a fresh Boolean variable
- Soft clause: $(a)$ with weight $W_C$. 
MaxSAT allows for compactly encoding various types of high-level finite-domain soft constraints

- Due to Cook-Levin Theorem:
  Any NP constraint can be polynomially represented as clauses

**Basic Idea**

Finite-domain soft constraint \( C \) with associated weight \( W_C \).

Let \( \text{CNF}(C) = \bigwedge_{i=1}^{m} C_i \) be a CNF encoding of \( C \).

Softening \( \text{CNF}(C) \) as Weighted Partial MaxSAT:

- **Hard clauses**: \( \bigwedge_{i=1}^{m} (C_i \lor a) \),
  where \( a \) is a fresh Boolean variable

- **Soft clause**: \((a)\) with weight \( W_C \).
MaxSAT allows for compactly encoding various types of high-level finite-domain soft constraints

- Due to Cook-Levin Theorem:
  Any NP constraint can be polynomially represented as clauses

**Basic Idea**

Finite-domain soft constraint $C$ with associated weight $W_C$.

Let $\text{CNF}(C) = \bigwedge_{i=1}^{m} C_i$ be a CNF encoding of $C$.

Softening $\text{CNF}(C)$ as Weighted Partial MaxSAT:

- **Hard clauses**: $\bigwedge_{i=1}^{m} (C_i \lor a)$, where $a$ is a fresh Boolean variable
- **Soft clause**: $(a)$ with weight $W_C$.

Important for various applications of MaxSAT
MaxSAT: Complexity

Deciding whether $k$ clauses can be satisfied: NP-complete

**Input:** A CNF formula $F$, a positive integer $k$.

**Question:** Is there an assignment that satisfies at least $k$ clauses in $F$?

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
  - Polynomial number of oracle calls
  - Other $\text{FP}^{\text{NP}}$–complete problems include TSP
- A SAT solver acts as the NP oracle most often in practice

MaxSAT is hard to approximate

APX: class of NP optimization problems that
- admit a constant-factor approximation algorithm, but
- have no poly-time approximation scheme (unless NP=潘).
MaxSAT: Complexity

Deciding whether $k$ clauses can be satisfied: NP-complete

**Input:** A CNF formula $F$, a positive integer $k$.

**Question:** Is there an assignment that satisfies at least $k$ clauses in $F$?

MaxSAT is $FP^{NP}$–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
  - Polynomial number of oracle calls
  - Other $FP^{NP}$–complete problems include TSP
- A SAT solver acts as the NP oracle most often in practice

MaxSAT is hard to approximate

APX: class of NP optimization problems that
- admit a constant-factor approximation algorithm, *but*
- have no poly-time approximation scheme (unless NP=P).
MaxSAT: Complexity

Deciding whether $k$ clauses can be satisfied: NP-complete

**Input:** A CNF formula $F$, a positive integer $k$.

**Question:** Is there an assignment that satisfies at least $k$ clauses in $F$?

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
  - Polynomial number of oracle calls
  - Other $\text{FP}^{\text{NP}}$–complete problems include TSP
- 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, *but*
- have no poly-time approximation scheme (unless NP=P).
Unsatisfiable Cores in (Weighted Partial) MaxSAT

**UNSAT core in MaxSAT**

A subset $F_s' \subseteq F_s$ such that $F_h \land F_s'$ is unsatisfiable.

- The hard clauses act as background theory
- ...but are *not* part of an UNSAT core

**Fact**

For any UNSAT core $F_s'$, we know that *some clause* $C \in F_s'$ need to be removed to make $F_h \land F_s'$ satisfiable.

- This fact is exploited by core-guided MaxSAT algorithms

**MUSes and MCSes in MaxSAT**

Defined similarly over $F_s$ assuming $F_h$. 
MaxSAT and Hitting Set Duality

Hitting Set Duality

For any UNSAT CNF formula $F$:

- $M \in \text{MCS}(F)$ iff $M$ is an irreducible hitting set of $\text{MUS}(F)$.
- $M \in \text{MUS}(F)$ iff $M$ is an irreducible hitting set of $\text{MCS}(F)$.

MaxSAT is about . . .

. . . finding a *smallest* hitting set of the set of MUSes.

- Incorporating weights: *minimum-cost* hitting set
Algorithms for $\text{MAXSAT}$ Solving
Standard Solver Input Format: DIMACS WCNF

- Variables indexed from 1 to $n$
- Negation: $-$
  - $-3$ stand for $\neg x_3$
- 0: special end-of-line character
- One special header “p”-line:
  ```
p wcnf <#vars> <#clauses> <top>
  ```
  - #vars: number of variables $n$
  - #clauses: number of clauses
  - top: “weight” of hard clauses.
    - Any number larger than the sum of soft clause weights can be used.

- Clauses represented as lists of integers
  - Weight is the first number
  - $(\neg x_3 \lor x_1 \lor \neg x_{45})$, weight 2:
    ```
    2 -3 1 -45 0
    ```
- Clause is hard if weight == top

Example:
```wcnf
mancoosi-test-i2000d0u98-26.wcnf
p wcnf 18169 112632 31540812410
31540812410 -1 2 3 0
31540812410 -4 2 3 0
31540812410 -5 6 0
... truncated 2.4 MB
```
MaxSAT Evaluations

Objectives

- Assessing the state of the art in the field of Max-SAT solvers
- Creating a collection of publicly available Max-SAT benchmark instances
- Tens of solvers from various research groups internationally participate each year
- Standard input format

11th MaxSAT Evaluation
http://maxsat.ia.udl.cat

Affiliated with SAT 2016: 19th Int’l Conference on Theory and Applications of Satisfiability Testing
Push-Button Solvers

• Black-box, *no command line parameters necessary*

• Input: CNF formula, in the *standard* DIMACS WCNF file format

• Output: provably optimal solution, or UNSATISFIABLE
  ▶ Complete solvers

```plaintext
mancoosi-test-i2000d0u98-26.wcnf
p wcnf 18169 112632 31540812410
  31540812410 -1 2 3 0
  31540812410 -4 2 3 0
  31540812410 -5 6 0
  ...
  18170 1133 0
  18170 457 0
... truncated 2.4 MB
```

Internally rely especially on CDCL SAT solvers

*for proving unsatisfiability of subsets of clauses*
Example: $ openwbo mancoosi-test-i2000d0u98-26.wcnf

c Open-WBO: a Modular MaxSAT Solver
c Version: 1.3.1 – 18 February 2015
...
c — Problem Type: Weighted
c — Number of variables: 18169
c — Number of hard clauses: 94365
c — Number of soft clauses: 18267
c — Parse time: 0.02 s
...
o 10548793370
c LB : 15026590
c Relaxed soft clauses 2 / 18267
c LB : 30053180
c Relaxed soft clauses 3 / 18267
c LB : 45079770
c Relaxed soft clauses 5 / 18267
c LB : 60106360

... 
c Relaxed soft clauses 726 / 18267
c LB : 287486453
c Relaxed soft clauses 728 / 18267
o 287486453
c Total time: 1.30 s
c Nb SAT calls: 4
c Nb UNSAT calls: 841
s OPTIMUM FOUND
v 1 -2 3 4 5 6 7 8 -9 10 11 12 13 14 15 16 ...
... -18167 -18168 -18169 -18170
Progress in **MaxSAT** Solver Performance

Comparing some of the best solvers from 2010–2014:

In 2014: 50% more instances solved than in 2010!
Some Recent MaxSAT Solvers

Open-source:
- OpenWBO  
  http://sat.inesc-id.pt/open-wbo/
- MaxHS  
  http://maxhs.org
- LMHS  
  http://www.cs.helsinki.fi/group/coreo/lmhs/

Binaries available:
- Eva  
  http://www.maxsat.udl.cat/14/solvers/eva500a__
- MaxSatz  
- MSCG  
  http://sat.inesc-id.pt/~aign/soft/
- WPM3  
  http://web.udl.es/usuaris/q4374304/#software
- QMaxSAT  
  https://sites.google.com/site/qmaxsat/
A Variety of Approaches

- Branch-and-bound
- Integer Programming (IP)
- SAT-Based Algorithms
  - Iterative / “model-based”
  - Core-based
- Implicit hitting set algorithms (IP/SAT hybrid).
Summary

Take-home message
- Importance of exact optimization
- MaxSAT an increasingly attractive approach to Boolean optimization

Study goals
- Basic concepts on MaxSAT
- Modelling optimization problem via MaxSAT
- MaxSAT cores

Next time:
Algorithms for MaxSAT


Bibliography II


Bibliography III


