Solving Sparse Instances of Max SAT via Width Reduction and Greedy Restriction

Takayuki Sakai (Kyoto University)
Joint work with
Kazuhsa Seto (Seikei University)
Suguru Tamaki (Kyoto University)
This Talk

- Theoretical Aspects
  - Worst case upper bound
  - Moderately exponential time algorithm

- Practical Aspects (not our scope)
  - Some branching rules (*width reduction & greedy restriction*) might be worth implementing
  - Sparse instances might be easy
Outline

- Introduction
  - Previous and our results
  - Motivation for sparse instances
- Previous Exponential Time Algorithms
  - Memorization
  - Branch and Bound
- Our Algorithm and its Analysis
  - Width reduction
  - Greedy restriction
- Summary
Input: a set of clauses with weights
\[ \{ (C_1, w_1), (C_2, w_2), \ldots, (C_m, w_m) \} \]
- \( C_i \): conjunction of literals, \( w_i \): positive value

Output: Maximum total weight of clauses satisfied

Example
\[ \{ (x_1 \lor x_2, 1), (x_1 \lor \overline{x}_2, 4), (\overline{x}_1 \lor x_2, 5), (\overline{x}_1 \lor \overline{x}_2, 2), (x_1 \lor x_3, 3) \} \]
### Maximum Satisfiability (Max SAT)

\[\{(x_1 \lor x_2,1), (x_1 \lor \bar{x}_2,4), (\bar{x}_1 \lor x_2,5), (\bar{x}_1 \lor \bar{x}_2,2), (x_1 \lor x_3,3)\}\]

<table>
<thead>
<tr>
<th>(x_1)</th>
<th>(x_2)</th>
<th>(x_3)</th>
<th>(x_1 \lor x_2)</th>
<th>(x_1 \lor \bar{x}_2)</th>
<th>(\bar{x}_1 \lor x_2)</th>
<th>(\bar{x}_1 \lor \bar{x}_2)</th>
<th>(x_1 \lor x_3)</th>
<th>total weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>11</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>14 (max)</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>8</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>11</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>13</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>13</td>
</tr>
</tbody>
</table>
Complexity of Max SAT

Facts
- Max SAT is NP-hard
- Exhaustive Search: \( O(2^n) \)

Question
- Moderately exponential time algorithm?

\[ \exists \mu > 0 \text{ s.t. } \text{Max SAT} \in \text{DTIME} \left[ 2^{(1-\mu)n} \right] ? \]
## Previous Results

<table>
<thead>
<tr>
<th>Running Time</th>
<th>Problem</th>
<th>Space</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>(O(2^{0.4414k}))</td>
<td>Max SAT</td>
<td>Poly.</td>
<td>[BG12]</td>
</tr>
<tr>
<td>(O(2^{0.1000l}))</td>
<td>Max 2-SAT</td>
<td>Poly.</td>
<td>[GH03]</td>
</tr>
<tr>
<td>(O(2^{0.1450l}))</td>
<td>Max SAT</td>
<td>Poly.</td>
<td>[BR99]</td>
</tr>
<tr>
<td>(O(2^{0.1583m}))</td>
<td>Max 2-SAT</td>
<td>Poly.</td>
<td>[GS12]</td>
</tr>
<tr>
<td>(O(2^{0.1901m}))</td>
<td>Max 2-CSP</td>
<td>Poly.</td>
<td>[GS12]</td>
</tr>
<tr>
<td>(O(2^{0.4057m}))</td>
<td>Max SAT</td>
<td>Poly.</td>
<td>[CK04]</td>
</tr>
<tr>
<td>(O(2^{0.7909n}))</td>
<td>Max 2-SAT</td>
<td>Exp.</td>
<td>[Koi06] [Wil05]</td>
</tr>
</tbody>
</table>

- k: objective value
- l: length of instance
- m: # constraints
- n: # variables
## Previous and Our Results

Sparse instance: \#clauses = $cn$

<table>
<thead>
<tr>
<th></th>
<th>Weight of clause</th>
<th>Running Time</th>
<th>Space</th>
<th>det./rand. type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dantsin and Wolpert 2006</td>
<td>1</td>
<td>$O\left(2^{\left(1-\Omega\left(\frac{1}{c \log c}\right)\right)n}\right)$</td>
<td>Exp.</td>
<td>deterministic memorization</td>
</tr>
<tr>
<td>Kulikov and Kutzkov 2007</td>
<td>unbounded</td>
<td>$O\left(2^{\left(1-\Omega\left(\frac{1}{2^{O(c)}}\right)\right)n}\right)$</td>
<td>Poly.</td>
<td>deterministic branch &amp; bound</td>
</tr>
<tr>
<td><strong>Our Result 1</strong></td>
<td>unbounded</td>
<td>$O\left(2^{\left(1-\Omega\left(\frac{1}{c^2 \log^2 c}\right)\right)n}\right)$</td>
<td>Poly.</td>
<td>deterministic branch &amp; bound</td>
</tr>
<tr>
<td><strong>Our Result 2</strong></td>
<td>unbounded</td>
<td>$O\left(2^{\left(1-\Omega\left(\frac{1}{c \log^3 c}\right)\right)n}\right)$</td>
<td>Poly.</td>
<td>randomized branch &amp; bound</td>
</tr>
</tbody>
</table>
Best upper bounds for Max $k$-SAT (any instances)
- Max 2-SAT : $O\left(1.732^n\right)$ [Williams 2004]
- Max 3-SAT : $O\left(2^n\right)$
  (cf. $k$-SAT : $O\left(2^{\left(1-1/k\right)n}\right)$)

Strong Exponential Time Hypothesis [CIP09]
“SAT (without restriction on clause length) cannot be solved in time $2^{\left(1-\varepsilon\right)n}$ for constant $\varepsilon > 0$”

... and Max SAT is at least as difficult as SAT
If we consider linear size instances (\#caluses = \(cn\)), then

\[
\text{CNF-SAT}_{cn} \in \text{DTIME} \left[ 2^{(1-\mu_c)n} \right]
\]

\[
\text{Max SAT}_{cn} \in \text{DTIME} \left[ 2^{(1-\mu_c)n} \right]
\]

\(\mu_c > 0\) : constant

Our Goal
Design an exponentially faster algorithm for linear size instances of Max SAT
Outline

- Introduction
  - Previous and our results
  - Motivation for sparse instances
- Previous Exponential Time Algorithms
  - Memorization
  - Branch and Bound
- Our Algorithm and its Analysis
  - Width reduction
  - Greedy restriction
- Summary
Algorithms for Max SAT

- Complete Algorithm
  - Branch and bound
  - Memorization (often exponential space)
  - Split and list (often exponential space)
  - Reduction to SAT (#variables increases, e.g. $n \log n$)
  - ...

- Incomplete Algorithm
  - Branch and bound with heuristics
  - Local search
  - ...

## Previous and Our Results

Sparse instance: \#clauses = \( cn \)

<table>
<thead>
<tr>
<th></th>
<th>Weight of clause</th>
<th>Running Time</th>
<th>Space</th>
<th>det./rand. type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dantsin and Wolpert 2006</td>
<td>1</td>
<td>( \mathcal{O}\left(2^{\Omega(1/c \log c)}n\right) )</td>
<td>Exp.</td>
<td>deterministic memorization</td>
</tr>
<tr>
<td>Kulikov and Kutzkov 2007</td>
<td>unbounded</td>
<td>( \mathcal{O}\left(2^{\Omega(1/2^{O(c)})}n\right) )</td>
<td>Poly.</td>
<td>deterministic branch &amp; bound</td>
</tr>
<tr>
<td><strong>Our Result 1</strong></td>
<td>unbounded</td>
<td>( \mathcal{O}\left(2^{\Omega(1/c^2 \log^2 c)}n\right) )</td>
<td>Poly.</td>
<td>deterministic branch &amp; bound</td>
</tr>
<tr>
<td><strong>Our Result 2</strong></td>
<td>unbounded</td>
<td>( \mathcal{O}\left(2^{\Omega(1/c \log^3 c)}n\right) )</td>
<td>Poly.</td>
<td>randomized branch &amp; bound</td>
</tr>
</tbody>
</table>
Ideas of Algorithm by Dantsin and Wolpert

- Make many small instances by partial assignments
- Find each small instance and its optimal value from prepared database (memorization)
- Running Time = \#partial assignments + preparing database
### Previous and Our Results

**Sparse instance: \#clauses = cn**

<table>
<thead>
<tr>
<th></th>
<th>Weight of clause</th>
<th>Running Time</th>
<th>Space</th>
<th>det./rand. type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dantsin and Wolpert 2006</td>
<td>1</td>
<td>$O\left(2^{\left(1-\Omega\left(1/c \log c\right)\right)n}\right)$</td>
<td>Exp.</td>
<td>deterministic memorization</td>
</tr>
<tr>
<td>Kulikov and Kutzkov 2007</td>
<td>unbounded</td>
<td>$O\left(2^{\left(1-\Omega\left(1/2^{O(c)}\right)\right)n}\right)$</td>
<td>Poly.</td>
<td>deterministic branch &amp; bound</td>
</tr>
<tr>
<td><strong>Our Result 1</strong></td>
<td>unbounded</td>
<td>$O\left(2^{\left(1-\Omega\left(1/c^2 \log^2 c\right)\right)n}\right)$</td>
<td>Poly.</td>
<td>deterministic branch &amp; bound</td>
</tr>
<tr>
<td><strong>Our Result 2</strong></td>
<td>unbounded</td>
<td>$O\left(2^{\left(1-\Omega\left(1/c \log^3 c\right)\right)n}\right)$</td>
<td>Poly.</td>
<td>randomized branch &amp; bound</td>
</tr>
</tbody>
</table>
Idea #1 (high frequency)
If there is a literal $x$ occurring at least $d$ times in the instance
($d=d(c)$: some parameter)
⇒ Recursively solve two cases $x=0$ and $x=1$
(This case: we can reduce the size of instance efficiently)
Branch and Bound Algorithm based on two ideas:

**Idea #2 (pruning)**
Otherwise, consider Max of #satisfiable clauses including $x_1$

\[
(x_1 \lor x_2 \lor x_3), (x_1 \lor \overline{x_2} \lor \overline{x_3}), (x_1 \lor \overline{x_2} \lor x_3), (x_1 \lor \overline{x_2}), (x_1 \lor x_2 \lor \overline{x_3}), (\overline{x_1} \lor x_2)
\]

- $x_1 = 1 \implies$ OPT is 6 when $x_2 = 1$
- $x_1 = 0 \implies$ OPT is 5 when $x_2 = 0, x_3 = 0/1$

$(x_1, x_2) = (0, 1)$ is always worse than $(x_1, x_2) = (1, 1)$

We only have to examine $x_1 = 1$ and $(x_1, x_2) = (0, 0)$
(This case: we can reduce search space efficiently)
Outline

- Introduction
  - Previous and our results
  - Motivation for sparse instances
- Previous Exponential Time Algorithms
  - Memorization
  - Branch and Bound
- Our Algorithm and its Analysis
  - Width reduction
  - Greedy restriction
- Summary
Branch and Bound Algorithm based on two ideas:

**Idea #1 (width reduction)**
If there is a clause $C$ of width $> k$ in the instance 
($k = k(c):$ some parameter) 
⇒ Apply clause-shortening algorithm  
(producing two sub-problems)

**Idea #2 (greedy restriction)**
Otherwise, pick a variable $x$ that occurs most frequently  
⇒ Recursively solve two cases $x=0$ and $x=1$
Details of Our Deterministic Algorithm

- **Width reduction**
  - Max SAT instance $\Rightarrow$ a set of Max k-SAT instances

- **Greedy restriction**
  - Max k-SAT is a special case of Max Formula SAT
  - Greedy restriction algorithm for (Max) formula SAT
**Width Reduction**

- **Key Idea:** reduce a given Max SAT instance to the union of Max-k-SAT instances.
  - **Using clause shortening algorithm!**
    
    \[ C = (x_1 \lor x_2 \lor \cdots \lor x_k \lor x_{k+1} \lor \cdots) \]

  - **Branch #1:** Pick up the first k literals and remove other variables
    \[ C'' = (x_1 \lor x_2 \lor \cdots \lor x_k) \]

  - **Branch #2:** Set the first k literals to be false.
    \[ C''' = (x_{k+1} \lor \cdots) \text{ and } x_1 = 0, x_2 = 0, \ldots, x_k = 0 \]
Width Reduction

\[ C = (x_1 \lor x_2 \lor \cdots \lor x_k \lor x_{k+1} \lor \cdots) \]

Branch #1: Pick up the first k literals and remove other variables

\[ C'' = (x_1 \lor x_2 \lor \cdots \lor x_k) \]

Branch #2: Set the first k literals to be false.

\[ C''' = (x_{k+1} \lor \cdots) \text{ and } x_1 = 0, x_2 = 0, \ldots, x_k = 0 \]

• OPT of original instance = max of OPTs of two new instances
Width Reduction

\[ C = (x_1 \lor x_2 \lor \cdots \lor x_k \lor x_{k+1} \lor \cdots) \]

Branch #1: Pick up the first k literals and remove other variables

\[ C' = (x_1 \lor x_2 \lor \cdots \lor x_k) \]

Branch #2: Set the first k literals to be false.

\[ C'' = (x_{k+1} \lor \cdots) \text{ and } x_1 = 0, x_2 = 0, \ldots, x_k = 0 \]

- Each Branch #1 reduce one clause of width > k
  - After cn times, all clauses have length at most k
- Each Branch #2 reduce k variables
  - After n/2k times, the number of variables is at most n/2
Width Reduction

\( k=2 \) : when we reduce original instance to Max 2-SAT instances

**Input** \( \{ (x_1 \lor \overline{x}_2 \lor x_3 \lor x_4 \lor x_5), \ldots, (\overline{x}_2 \lor x_3), \ldots \} \)

\( C_1 = (x_1 \lor \overline{x}_2 \lor x_3 \lor x_4 \lor \overline{x}_5) \)

Branch 1 \( C_1' = (x_1 \lor \overline{x}_2) \)

Branch 2

\( x_1=0, x_2=1 \)

\( C_1'' = (x_3 \lor x_4 \lor \overline{x}_5) \)

\( C_5 = (x_3) \)
Details of Our Deterministic Algorithm

- **Width reduction**
  - Max SAT instance $\Rightarrow$ a set of Max $k$-SAT instances

- **Greedy restriction**
  - Max $k$-SAT is a special case of Max Formula SAT
  - Greedy restriction algorithm for (Max) formula SAT
Max formula SAT

- We need some definitions
  - De Morgan formula
  - formula SAT
  - Max formula SAT
- Binary Tree
  - Internal node: AND, OR
  - NOT appears in only leaves
  - Size = #leaves

$\#\text{variables} = 3$
Size = 5
De Morgan formula SAT

Input: De Morgan formula
Output: yes if formula is satisfiable
Max De Morgan formula SAT

- Input: a set of weighted De Morgan formula
  \[ \Phi = \{ (\phi_1, w_1), (\phi_2, w_2), \ldots, (\phi_m, w_m) \}, \text{ each } w_i > 0 \]
- Output: Maximum total weight of constraints satisfied
Max De Morgan formula SAT

- Max De Morgan formula SAT
  - Input: a set of weighted De Morgan formula
    \[ \Phi = \{ (\phi_1, w_1), (\phi_2, w_2), \ldots, (\phi_m, w_m) \}, \text{ each } w_i > 0 \]
  - Output: Maximum total weight of constraints satisfied

- Max k-SAT is a special case of Max formula SAT
  - Each De Morgan formula has size at most k
  - Each De Morgan formula has only OR label
Details of Our Algorithm

- **Width reduction**
  - Max SAT instance \(\Rightarrow\) a set of Max \(k\)-SAT instances

- **Greedy restriction**
  - Max \(k\)-SAT is a special case of Max Formula SAT
  - Greedy restriction algorithm for (Max) formula SAT
Greedy Restriction and Simplification

Greedy Restriction = Assign 0/1 to \( x \) which occurs most frequently

Simplification = Simplify the formula via the following rules

1. \( 1 \land \psi \to \psi \), \( 0 \lor \psi \to \psi \)
2. \( 0 \land \psi \to 0 \), \( 1 \lor \psi \to 1 \)
3. \( z \land \psi \to z \land \psi[z = 1] \)
4. \( z \lor \psi \to z \lor \psi[z = 0] \)

(\( \psi \): any subformula of \( \phi \))
Example of Simplification

(1) $1 \land \psi \rightarrow \psi, \ 0 \lor \psi \rightarrow \psi$

(2) $0 \land \psi \rightarrow 0, \ 1 \lor \psi \rightarrow 1$

(3) $z \land \psi \rightarrow z \land \psi[z = 1]$

(4) $z \lor \psi \rightarrow z \lor \psi[z = 0]$

($\psi$: any subformula of $\phi$)
Example of Greedy Restriction

(1) $1 \land \psi \rightarrow \psi$, $0 \lor \psi \rightarrow \psi$
(2) $0 \land \psi \rightarrow 0$, $1 \lor \psi \rightarrow 1$
(3) $z \land \psi \rightarrow z \land \psi[z = 1]$
(4) $z \lor \psi \rightarrow z \lor \psi[z = 0]$

($\psi$: any subformula of $\phi$)

For each formula, we use simplification rules.
The complexity of Max formula SAT

Lemma

Max formula SAT with $cn$ size can be solved by deterministic polynomial space algorithm in time

$$O\left(2^{\left(1-\Omega\left(\frac{1}{c^2}\right)\right)n}\right)$$

Corollary

Max $k$-SAT with $cn$ clauses ($kcn$ literals) can be solved by deterministic polynomial space algorithm in time

$$O\left(2^{\left(1-\Omega\left(\frac{1}{k^2c^2}\right)\right)n}\right)$$
Running time analysis of combining width reduction and k-SAT algorithm was given by [Sch05, CIP06]

- Basically we replace k-SAT algorithm by Max k-SAT algorithm

**Our Result 1**

Max SAT with $cn$ clauses can be solved by deterministic polynomial space algorithm in time

$$O\left(2^{\left(1-\Omega(1/c^2 \log^2 c)\right)n}\right)$$
Outline

- Introduction
  - Previous and our results
  - Motivation for sparse instances
- Previous Exponential Time Algorithms
  - Memorization
  - Branch and Bound
- Our Algorithm and its Analysis
  - Width reduction
  - Greedy restriction
- Summary
## Previous and Our Results

Sparse instance: \#clauses = $cn$

<table>
<thead>
<tr>
<th></th>
<th>Weight of clause</th>
<th>Running Time</th>
<th>Space</th>
<th>det./rand. type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dantsin and Wolpert 2006</td>
<td>1</td>
<td>$O\left(2^{\Omega\left(1/c\log c\right)}n\right)$</td>
<td>Exp.</td>
<td>deterministic memorization</td>
</tr>
<tr>
<td>Kulikov and Kutzkov 2007</td>
<td>unbounded</td>
<td>$O\left(2^{\Omega\left(1/2^{O(c)}\right)}n\right)$</td>
<td>Poly.</td>
<td>deterministic branch &amp; bound</td>
</tr>
<tr>
<td><strong>Our Result 1</strong></td>
<td>unbounded</td>
<td>$O\left(2^{\Omega\left(1/c^2\log^2 c\right)}n\right)$</td>
<td>Poly.</td>
<td>deterministic branch &amp; bound</td>
</tr>
<tr>
<td><strong>Our Result 2</strong></td>
<td>unbounded</td>
<td>$O\left(2^{\Omega\left(1/c\log^3 c\right)}n\right)$</td>
<td>Poly.</td>
<td>randomized branch &amp; bound</td>
</tr>
</tbody>
</table>
Some Remarks

- Our randomized algorithm uses random restriction instead of greedy restriction.

- Our algorithms can count the number of optimal assignments.

- Our algorithms can handle hard constraints (partial Max SAT).
Future Work

- Improve the applicable size?
  - Now, applicable for size $O(n^{1.5})$ in time $O(2^n - \omega(\log n))$
  - De Morgan Formula SAT Algorithm is applicable to
    - Deterministic: $O(n^{2.5-o(1)})$ size
      - Chen, Kabanets, Kolokolova, Shaltiel, Zuckerman (ECCC 2012)
    - Randomized: $O(n^{3-o(1)})$ size
      - Komargodski, Raz, Tal (FOCS 2013)

- Fast Max 2-SAT Algorithm with Polynomial Space
- Non-Trivial Algorithm for Max 3-SAT

Thank you for your attention!