Practical SAT Solving

Lecture 11

Carsten Sinz, Tomas Balyo  |  July 17, 2018
Lecture Outline

- Derandomizing Local Search
- SAT Upper Bounds
- Covering Codes
Repetition: Local Search

```latex
Maybe[Assignment] GSAT(ClauseSet S)
{
    for \( i = 1 \) to MAX_TRIES do {
        \( \alpha = \) random-assignment to variables in S
        for \( j = 1 \) to MAX_FLIPS do {
            if ( \( \alpha \) satisfies all clauses in S ) return \( \alpha \)
            \( x = \) variable that produces least number of unsatisfied clauses when flipped
            flip \( x \)
        }
    }
    return Nothing // no solution found
}
```
Open questions

- How often should the inner/outer loop be repeated?
- How good are the chances to find a satisfying assignment?
- Can the algorithm be modified to prove unsatisfiability?
- Is there a general upper bound for deciding SAT that is better than $2^n$?

```c
Maybe[Assignment] GSAT(ClauseSet S)
{
    for i = 1 to MAX_TRIES do {
        $\alpha$ = random-assignment to variables in S
        for j = 1 to MAX_FLIPS do {
            if ($\alpha$ satisfies all clauses in S) return $\alpha$
            $x$ = variable that produces least number of unsatisfied clauses when flipped
            flip $x$
        }
    }
    return Nothing  // no solution found
}
```
## Complexity of k-SAT: Upper Bounds

<table>
<thead>
<tr>
<th>3-SAT</th>
<th>4-SAT</th>
<th>5-SAT</th>
<th>6-SAT</th>
<th>type</th>
<th>ref.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.782</td>
<td>1.835</td>
<td>1.867</td>
<td>1.888</td>
<td>det.</td>
<td>[PPZ97]</td>
</tr>
<tr>
<td>1.618</td>
<td>1.839</td>
<td>1.928</td>
<td>1.966</td>
<td>det.</td>
<td>[MS85]</td>
</tr>
<tr>
<td>1.588</td>
<td>1.682</td>
<td>1.742</td>
<td>1.782</td>
<td>prob.</td>
<td>[PPZ97]</td>
</tr>
<tr>
<td>1.579</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>det.</td>
<td>[Sch92]</td>
</tr>
<tr>
<td>1.505</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>det.</td>
<td>[Kul99]</td>
</tr>
<tr>
<td><strong>1.481</strong></td>
<td><strong>1.6</strong></td>
<td><strong>1.667</strong></td>
<td><strong>1.75</strong></td>
<td>det.</td>
<td>[DGH+02]</td>
</tr>
<tr>
<td>1.362</td>
<td>1.476</td>
<td>1.569</td>
<td>1.637</td>
<td>prob.</td>
<td>[PPSZ98]</td>
</tr>
<tr>
<td>1.334</td>
<td>1.5</td>
<td>1.6</td>
<td>1.667</td>
<td>prob.</td>
<td>[Sch99]</td>
</tr>
<tr>
<td>1.3302</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>prob.</td>
<td>[HSSW02]</td>
</tr>
<tr>
<td>1.3290</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>prob.</td>
<td>[BS03]</td>
</tr>
<tr>
<td>1.324</td>
<td>1.474</td>
<td>-</td>
<td>-</td>
<td>prob.</td>
<td>[*]</td>
</tr>
</tbody>
</table>

Algorithm of Dantsin et al.

```plaintext
Maybe[Assignment] Local-Search(ClauseSet S)
{
    for i = 1 to MAX_TRIES do
        α = random-assignment
        for j = 1 to MAX_FLIPS do
            if α satisfies all clauses in S return α
            choose variable x to flip
            flip x
        return Nothing
}
```

Variant of local search algorithmus (as above)

- **Input:** Formula in \( k\)-CNF with \( n \) variables.

**Idea:**

1. Make inner loop deterministic: systematically check all assignments in the „neighborhood“ of \( α \).
2. Choose starting assignment \( α \) in outer loop deterministically.
Idea 1: Hamming Ball Algorithm

Hamming distance between two assignments \( \alpha, \beta \):

**Def.:** \[ d(\alpha,\beta) := |\{ x \mid \alpha(x) \neq \beta(x) \}| \]

**Ex.:** \( \alpha=(0,1,1,0,0), \beta=(1,0,1,0,1), \ d(\alpha,\beta) = 3 \)

Hamming ball of radius \( r \) around \( \alpha \):

**Def.:** \[ H(\alpha, r) := \{ \beta \mid d(\alpha,\beta) \leq r \} \]

Volume \( V(r) \) of the Hamming ball: (given \( n \) variables, i.e. assignments are vectors of length \( n \)):

\[
V(r) := |H(\alpha, r)| = \sum_{i=0}^{r} \binom{n}{i} \quad \text{independent of } \alpha.
\]

Define normalized radius \( \rho := r/n \) (i.e. \( 0 \leq \rho \leq 1 \))
Illustration: Hamming Ball

Example: n=4

Ball around $\alpha=\langle 0001 \rangle$ with radius 1 and radius 2.
Hamming Ball Algorithm

- Known approximation of the Hamming ball volume (from information theory, Shannon) for $\rho \leq 1/2$:

$$\frac{1}{\sqrt{8np(1 - \rho)}} 2^{h(\rho)n} \leq V(r) \leq 2^{h(\rho)n}$$

where $h(\rho) := -\rho \log \rho - (1 - \rho) \log(1 - \rho)$ (Entropy function; log = logarithm for base 2).

- i.e.: $V(\rho n) \approx 2^{h(\rho)n}$ (within polynomial factor)
Search in Hamming Ball

```plaintext
Maybe[Assignment] HB-Search(\alpha, r)
{
    if(\alpha \models F) then return \alpha
    if (r = 0) return Nothing
    choose C=(L_1 \lor \ldots \lor L_k) with \alpha(C)=0
    for i = 1 to k do
        \alpha_i = \alpha with \alpha(L_i)=1
        \beta = HB-Search(\alpha_i, r-1)
        if(\beta \neq Nothing) return \beta
    return Nothing
}
```

- **Run time HB-Search:** $k^r$

- **Note:** $k^r$ can be considerably smaller than $V(r)$.
  - E.g.: $k = 3$, $r = n/2$:
    
    \[ V(r) = 2^{n-1}, \quad k^r = 3^{n/2} \approx 1.7321^n \]
Search in Hamming Ball

- HB-Search provides simple deterministic algorithm for 3-SAT:
  - Execute two HB searches around $\alpha_0=(0,...,0)$ and $\alpha_1=(1,...,1)$ with radius $n/2$.
  - Runtime (as seen): $1.7321^n$

- Can we improve that?
  - Idea: Choose radius $r = \rho n$ with $\rho<1/2$.
  - Of course, more than two starting points for $\alpha$ in the outer loop are needed then.
Probabilistic Variant of HB LS Algorithm

```haskell
Maybe[Assignment] Local-Search-HB-Prob
{
    for i = 1 to w do
        αᵢ = random-assignment
        β = HB-Search(αᵢ, ρ ⋅ n)
        if(β ≠ Nothing) return β
    return Nothing
}
```

- Choose αᵢ uniformly and independently at random from the $2^n$ possibilities, w iterations.
- Runtime: $w \cdot k^{ρn}$.
- Goal: Choose w and ρ such that (a) error probability is neglectably small and (b) runtime is optimized.
Probabilistic Algorithm: Error Probability (I)

- Only possible error: $F$ satisfiable, algorithm doesn’t find a satisfying assignment
- Assumption: $F$ satisfiable, $\alpha \models F$.
- HB-Search($\alpha_i, \rho n$) finds $\alpha$, if $\alpha \in H(\alpha_i, \rho n)$.
- Probability for that: $V(\rho n) \cdot 2^{-n}$
  - With Shannon approximation $V(\rho n) \approx 2^{h(\rho)n}$: probability approx.
    
    $2^{h(\rho)n} \cdot 2^{-n} = 2^{h(\rho)n-n} = 2^{-(1-h(\rho))n}$
  - Probability that $\alpha$ is not found in $w$ iterations:
    
    $$(1 - 2^{-(1-h(\rho))n})^w \leq e^{-w \cdot 2^{-(1-h(\rho))n}}$$

(with approximation $1+x \leq e^x$ for all $x$)
Error Probability and Runtime Optimization

- Probability that $\alpha$ is not found in $w$ iterations:

$$\left(1 - 2^{-(1-h(\rho))}\right)^w \leq e^{-w \cdot 2^{-(1-h(\rho))}}$$

(with approximation $1+x \leq e^x$ for all $x$)

- If error shall be below $e^{-c}$: $w = c \cdot 2^{(1-h(\rho))n}$ iterations needed.

- Runtime thus: (within polynomial factor):

$$w \cdot k^\rho n \approx \left(2^{(1-h(\rho))} \cdot k^\rho\right)^n = \left(2 \cdot \rho^\rho \cdot (1 - \rho)^{(1-\rho)} \cdot k^\rho\right)^n$$

(with def. entropy: $h(\rho) = - \rho \log \rho - (1 - \rho) \log(1 - \rho)$)

- Minimize runtime by differentiating (*) w.r.t. $\rho$ and solving for $\rho$.

Result: $\rho = 1/(k+1)$.

- Substituting that in (*) provides runtime:

$$\left(\frac{2k}{k+1}\right)^n$$
Result for Probabilistic Algorithm

- Runtime: $\left( \frac{2k}{k+1} \right)^n$

- Base for small values of $k$:

<table>
<thead>
<tr>
<th>$k$</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>$2k/(k+1)$</td>
<td>1.5</td>
<td>1.6</td>
<td>1.66667</td>
<td>1.71429</td>
<td>1.75</td>
</tr>
</tbody>
</table>

- Next step: Derandomize this algorithm (i.e. outer loop)
Covering Codes

- **Derandomization**: Deterministic algorithm for selecting starting assignments $\alpha_i$.

- **Definition**: A set $C$ of assignments is a covering code of radius $r$, if for each assignment $\alpha$: $\alpha \in H(\gamma, r)$ for at least one $\gamma \in C$.

- **Example**: Covering code of radius 1.
Construction of Covering Codes

- Problem is a special case of SET-COVER:
  - Given: Set $S$ and family $F \subseteq 2^S$ of subsets of $S$
  - Wanted: Subset $C \subseteq F$, which covers $S$, i.e., $S = UC$
  - Goal: Minimize $|C|$

- Greedy algorithm provides $\log|S|$-approximation for SET-COVER
Construction of Covering Codes

- Problem is a special case of SET-COVER.

Given: Set $S$ and family $F \subseteq 2^S$ of subsets of $S$

Wanted: Subset $C \subseteq F$, which covers $S$, i.e., $S = UC$

Goal: Minimize $|C|$

Greedy algorithm provides $\log|S|$-approximation for SET-COVER.
Greedy algorithm for SET-COVER

1. Set $C = \emptyset$.
2. As long as there are not-covered elements in $S$:
   a) Choose $c \in F$, covering largest number of yet not-covered elements from $S$.
   b) Add $c$ to $C$.
3. If $\cup C = S$, then $C$ is a solution; otherwise none exists.

Known:
Result of greedy alg. is $\log|S|$-approximation for SET-COVER, i.e.,
$|C| \leq \log|S| \cdot |C^*|$, where $C^*$ is optimal solution.
(in our case: $|S| = 2^n$, i.e. $\log|S| \in O(n)$)

Needed for approximation of $|C|$:
Upper bound for $C^*$
Upper Bound for $C^*$

- Obviously, for all covering codes $C$ (for a given radius $r$) the following holds: $|C| \geq \frac{2^n}{V(r)}$.
- Next lemma: optimal code is at most by a factor of $n$ larger.
Lemma: Size of Optimal Covering Code

For all $n$ and $r = \rho n$ with $0 \leq \rho \leq 1/2$, there is a covering code for radius $r$

of size $s := n \cdot \sqrt{8n\rho(1 - \rho)} \cdot 2^{(1-h(\rho))n}$.

Proof: probabilistic method

A randomly selected set of $s$ assignments is with high probability a
covering code. Thus, in particular, such a code must exist.
Proof of Lemma

- \( \beta \) fixed assignment, \( \alpha \) selected at random.
- Then: \( \beta \in H(\alpha, r) \) with probability \( V(r)/2^n \).
- \( C \): set of \( s \) independently selected random assignments.
- Then: Probability that \( \beta \) is not covered by \( C \):

\[
\left(1 - \frac{V(r)}{2^n}\right)^s \leq e^{-s \frac{V(r)}{2^n}} \leq e^{-s \frac{2^{h(\rho) - 1}n}{\sqrt{8n\rho(1 - \rho)}}} \leq e^{-n}
\]

(with approximation \( 1 + x \leq e^x \) for all \( x \), \( V(r) \geq \frac{2^{h(\rho)n}}{\sqrt{8n\rho(1 - \rho)}} \))

and \( s = n \cdot \sqrt{8n\rho(1 - \rho)} \cdot 2^{(1 - h(\rho))n} \)

- In total: probability that \( C \) is covering code:

\[
\geq 1 - 2^n \cdot e^{-n}
\]

(approaches 1 for \( n \to \infty \))
Analysis of the Greedy Algorithm

- Optimal code: $|C^*| \leq s$
- Greedy algorithm:
  - $|C| \leq O(n) \cdot s = O((5/2)^n) \cdot 2^{(1-h(\rho))n}$
  - Runtime of greedy algorithm: $2^{3n}$ (without proof)
- Result: $|C|$ ok, but runtime of SET-COVER algorithm too high!
- Idea:
  - Reduce runtime by allowing increased $|C|$. 

Carsten Sinz, Tomas Balyo – SAT Solving
Block Codes

Idea:
- Let $C$ be a covering code of radius $r$ for $n$ variables
- Obtain covering code $C^d$ of radius $d \cdot r$ for $d \cdot n$ variables by repeating covering code $C$ for $d$ times

Example:

- $d=4$, $|C|=3$: covering code for $4n$ variables with $|C|^4$ elements.

In general:
- Block code $C^d$ has size $|C|^d$ and radius $r \cdot d$. 
Deterministic Hamming-Ball Algorithm

- Choose \( d=6 \) and use block code \( C^6 \).
- Generate covering code \( C \) for \( n/6 \) variables with radius \( \rho n/6 \) using greedy SET-COVER algorithm
  - Runtime: \( 2^{3n/6} \leq 1.414^n \)
- Size of covering code \( C^6 \):
  \[
  O \left( \frac{n^{5/2}}{6} \cdot 2^{(1-h(\rho))n/6} \right)^6 = O(n^{15}) \cdot 2^{(1-h(\rho))n}
  \]
  - Same as non-blockcode, ignoring polynomial factor
- Complete run time of algorithm:
  1. Generating covering code: \( O(1.414^n) \) (once before start of Hamming-ball search)
  2. Hamming-ball search with radius \( r = n/(1+k) \): \( O(2k/(k+1))^n \)
     --> The latter dominates for \( k \geq 3 \), thus: \( O(2k/(k+1))^n \) (i.e. same run-time as probabilistic algorithm, e.g. \( 1.5^n \) for 3-SAT)
Further Improvements

- Hamming-ball search can also be done in time $2.848^r$ (instead of $3^r$).
  - Refined algorithm: Dantsin et al., 2002.

- Therefore:
  - Run-time of $1.481^n$ for deterministic 3-SAT algorithm (best upper bound for deterministic 3-SAT so far)