To Encode or to Propagate?
The Best Choice for Each Constraint in SAT

I. Abío, R. Nieuwenhuis, A. Oliveras, E. Rodríguez-Carbonell, P. Stuckey

Rich Model Toolkit COST Action Meeting
16 June 2013
Malta
Overview

- Motivation: solving constraints with SAT technology
  - Eager Approach: SAT encodings
  - Lazy Approach: SMT/propagators

- Choosing Right: Related Work and Contributions

- Experimental Results

- Conclusions and Future Work
Motivation

**Goal:** solving systems of constraints with SAT tools

**Applications:**
- Many in scheduling, timetabling, planning, etc.
- Also in *constraint-based* program analysis/synthesis
Motivation

Goal: solving systems of constraints with SAT tools

Applications:
- Many in scheduling, timetabling, planning, etc.
- Also in constraint-based program analysis/synthesis

Why using SAT? (cf. Linear/Constraint Programming)
- SAT tech outperforms other tools on real-world problems with a single, fully automatic variable selection strategy!
- Hence problem solving is essentially declarative
Motivation

- **Goal:** solving systems of constraints with SAT tools

- **Applications:**
  - Many in scheduling, timetabling, planning, etc.
  - Also in *constraint-based* program analysis/synthesis

- **Why using SAT?** (cf. Linear/Constraint Programming)
  - SAT tech *outperforms* other tools on *real-world problems* with a *single, fully automatic* variable selection strategy!
  - Hence problem solving is essentially *declarative*

- **However,** propositional logic is a very *low-level* language for complex constraints
Cardinality and PB Constraints

Example: limited-resource problems

- Some tasks \(\{1,2,\ldots,n\}\) must be carried out
- Tasks require some (limited) resources
- Variable \(a_{i,t}\) is true if task \(i\) is active at time \(t\)
Cardinality and PB Constraints

Example: limited-resource problems

- Some tasks \( \{1, 2, \ldots, n\} \) must be carried out
- Tasks require some (limited) resources
- Variable \( a_{i,t} \) is true if task \( i \) is active at time \( t \)

**Constraint:** There are no more active tasks than machines:

\[
a_{1,t} + a_{2,t} + \ldots + a_{n,t} \leq 20
\]

In general, cardinality cons. are of the form \( \sum_{i=1}^{n} x_i \leq k \)
Cardinality and PB Constraints

Example: limited-resource problems

- Some tasks \( \{1, 2, \ldots, n\} \) must be carried out
- Tasks require some (limited) resources
- Variable \( a_{i,t} \) is true if task \( i \) is active at time \( t \)

- **Constraint:** There are no more active tasks than machines:

\[
a_{1,t} + a_{2,t} + \ldots + a_{n,t} \leq 20
\]

In general, **cardinality cons.** are of the form \( \sum_{i=1}^{n} x_i \leq k \)

- **Constraint:** The max number of workers is not exceeded:

\[
3a_{1,t} + 4a_{2,t} + \ldots + 10a_{n,t} \leq 50
\]

In general, **pseudo-Boolean (PB) cons.** are of the form \( \sum_{i=1}^{n} a_i x_i \leq k \)
SAT Encodings

Express constraint $C$ with (CNF) formula $F$ (the *encoding*) s.t.
- For each solution to $C$ there is a model of $F$
- For each model of $F$ there is a solution to $C$
Example: for a cardinality constraint $\sum_{i=1}^{n} x_i < k$ we have:

- **Naive encoding.**
  - Variables: the same $x_1, \ldots, x_n$
  - Clauses: $\overline{x_{i_1}} \lor \ldots \lor \overline{x_{i_k}}$ for all $1 \leq i_1 < \ldots < i_k \leq n$
  - This is $\binom{n}{k}$ clauses!
Example: for a cardinality constraint $\sum_{i=1}^{n} x_i < k$ we have:

- **Naive encoding.**
  - Variables: the same $x_1, \ldots, x_n$
  - Clauses: $\overline{x_{i_1}} \lor \ldots \lor \overline{x_{i_k}}$ for all $1 \leq i_1 < \ldots < i_k \leq n$
  - This is $\binom{n}{k}$ clauses!

- **Sorting network encoding.**
  - Build a circuit that sorts (say, decreasingly) $n$ bits with inputs $x_1, \ldots, x_n$ and outputs new variables $y_1, \ldots, y_n$
  - Variables: $x_1, \ldots, x_n$ and gates of the circuit
  - Clauses: Tseitin encoding of the circuit + unit clause $\overline{y_k}$
  - Can be done with $O(n \log^2(n))$ clauses and new vars!
Only first $k$ outputs suffice:

cardinality networks just use $O(n \log^2(k))$ clauses, vars
Only first $k$ outputs suffice:

*cardinality networks* just use $O(n \log^2(k))$ clauses, vars

In the following:

*cardinality networks* used for encoding cardinality constraints (among most robust, efficient encodings for these constraints)
Several encodings exist
- Unary/binary adder circuits
- Sorting networks
- BDD’s
Several encodings exist
- Unary/binary adder circuits
- Sorting networks
- BDD’s

Example of encoding $2x_1 + 3x_2 + 5x_3 \leq 6$ with a BDD:

Construct the (RO)BDD wrt. ordering $x_1 \succ x_2 \succ x_3$...

... and relate truth values of parents and children according to selector variables
In the encoding of $\sum_{i=1}^{n} a_i x_i \leq k$ with BDD’s:

- **Variables**: $x_1, \ldots, x_n$ and one for each node of the BDD.
- **Clauses**: if $n$ is a node with selector variable $x$ and true and false children $t$ and $f$, express

  $$x \rightarrow (n \leftrightarrow t) \quad \overline{x} \rightarrow (n \leftrightarrow f)$$

- **Linear number of clauses/variables in the size of the BDD**
In the encoding of $\sum_{i=1}^{n} a_i x_i \leq k$ with BDD’s:

- Variables: $x_1, \ldots, x_n$ and one for each node of the BDD
- Clauses: if $n$ is a node with selector variable $x$ and true and false children $t$ and $f$, express

$$x \rightarrow (n \leftrightarrow t) \quad \bar{x} \rightarrow (n \leftrightarrow f)$$

- Linear number of clauses/variables in the size of the BDD

There are families of PB constraints for which no ordering of variables yields polynomial-size BDD-based encodings

... but this rarely occurs in practice
In the encoding of \( \sum_{i=1}^{n} a_i x_i \leq k \) with BDD’s:

- Variables: \( x_1, \ldots, x_n \) and one for each node of the BDD
- Clauses: if \( n \) is a node with selector variable \( x \) and true and false children \( t \) and \( f \), express

\[
x \rightarrow (n \leftrightarrow t) \quad \overline{x} \rightarrow (n \leftrightarrow f)
\]

- Linear number of clauses/variables in the size of the BDD

There are families of PB constraints for which no ordering of variables yields polynomial-size BDD-based encodings... but this rarely occurs in practice

In the following:
BDD’s used for encoding PB constraints
(among most efficient encodings in practice)
Pros and Cons of SAT Encodings

Encodings introduce auxiliary variables that:

- Yield smaller formulations,
- May produce more general/shorter lemmas,
- Can be used for case splitting,
- But make search space larger
Pros and Cons of SAT Encodings

- Encodings introduce auxiliary variables that:
  - ✓ yield smaller formulations,
  - ✓ may produce more general/shorter lemmas,
  - ✓ can be used for case splitting,
  - ✗ but make search space larger

- ✗ Encodings impractical if problem has many/large constraints
Instead of *eagerly* encoding the constraint, deal with it *lazily*
Instead of *eagerly* encoding the constraint, deal with it *lazily*

DPLL($T$) approach for solving $\text{CNF} \land \text{Constraint}$:

- Assignment compatible with CNF
- Literals implied by assignment and constraint

**CNF**

**SAT solver**

**Propagator**

($T$-solver)

**Constraint**
Example: \( \overline{x}_1 \lor x_2, \ x_3 \lor x_4, \ x_1 + x_2 + x_3 + x_4 \leq 2 \)
Example: \( \overline{x}_1 \lor x_2, \ x_3 \lor x_4, \ x_1 + x_2 + x_3 + x_4 \leq 2 \)

To Encode or to Propagate? The Best Choice for Each Constraint in SAT – p.12/23
Example: \( \overline{x}_1 \lor x_2, \quad x_3 \lor x_4, \quad x_1 + x_2 + x_3 + x_4 \leq 2 \)

\[ x_1 \lor x_2 \]

\[ x_3 \lor x_4 \]

\[ x_1^d x_2 \]

\[ \text{UnitPropagate} \]
Example: \( \bar{x}_1 \lor x_2, \ x_3 \lor x_4, \ x_1 + x_2 + x_3 + x_4 \leq 2 \)

\[ x_1 d x_2 \bar{x}_3 \bar{x}_4 \]

SAT solver

Propagator (T-solver) \( x_1 + x_2 + x_3 + x_4 \leq 2 \)

\( \bar{x}_3, \bar{x}_4 \)

T-Propagate
SMT/propagators (2)

Example: \( \overline{x}_1 \lor x_2, \ x_3 \lor x_4, \ x_1 + x_2 + x_3 + x_4 \leq 2 \)

\( x_1^d x_2 \overline{x}_3 \overline{x}_4 \)

Conflict!

To Encode or to Propagate? The Best Choice for Each Constraint in SAT – p.12/23
Example: \( \overline{x}_1 \lor x_2, \; x_3 \lor x_4, \; x_1 + x_2 + x_3 + x_4 \leq 2 \)

\[ x_1 d \times 2 \times 3 \overline{x}_4 \]

SAT solver

\( \overline{x}_1 \lor x_2 \)

\( x_3 \lor x_4 \)

\( \overline{x}_1 \)

Propagator

\( (T\text{-solver}) \)

\[ x_1 + x_2 + x_3 + x_4 \leq 2 \]

Learn

To Encode or to Propagate? The Best Choice for Each Constraint in SAT – p.12/23
Example: \( \overline{x}_1 \lor x_2, \quad x_3 \lor x_4, \quad x_1 + x_2 + x_3 + x_4 \leq 2 \)

SAT solver

\( \overline{x}_1 \)

Propagator

\( (T\text{-solver}) \)

Backjump

\( x_1 + x_2 + x_3 + x_4 \leq 2 \)
SMT/propagators (2)

Example: \(\bar{x}_1 \lor x_2, \ x_3 \lor x_4, \ x_1 + x_2 + x_3 + x_4 \leq 2\)

\[\begin{array}{c}
\text{SAT solver} \\
\bar{x}_1 \lor x_2 \\
x_3 \lor x_4 \\
\bar{x}_1 \\
\end{array}\]

\[\begin{array}{c}
\text{Propagator} \\
(T\text{-solver}) \\
x_1 + x_2 + x_3 + x_4 \leq 2 \\
\end{array}\]

SAT solver requires that the propagator:
- Detects lits implied by partial assignment and constraint
- Gives explanations of propagated lits for conflict analysis

To Encode or to Propagate? The Best Choice for Each Constraint in SAT – p.12/23
Consider the constraint $x_1 + \ldots + x_n \leq k$

Let us count no. of true literals, i.e., the size of $A_1 = \{i \mid x_i = 1\}$
Consider the constraint $x_1 + \ldots + x_n \leq k$

Let us count no. of true literals, i.e., the size of $A_1 = \{i \mid x_i = 1\}$

If $|A_1| \geq k$, let $E \subseteq A_1$ such that $|E| = k$

For any $j \not\in E$, literal $\overline{x}_j$ can be propagated

Explanation: clause

$$\bigvee_{i_s \in E} \overline{x}_{i_s} \lor \overline{x}_j$$
Propagator for Cardinality Constraints

Consider the constraint $x_1 + \ldots + x_n \leq k$

Let us count no. of true literals, i.e., the size of $A_1 = \{i \mid x_i = 1\}$

If $|A_1| \geq k$, let $E \subseteq A_1$ such that $|E| = k$

For any $j \not\in E$, literal $\overline{x}_j$ can be propagated

Explanation: clause

$$\bigvee_{i_s \in E} \overline{x}_{i_s} \lor \overline{x}_j$$

Note that explanations are the clauses of the naive encoding

In general, SMT can be seen as lazily producing an encoding (without auxiliary variables)
Consider the constraint \( a_1x_1 + \ldots + a_nx_n \leq k \) with \( a_i \geq 0 \)

Let us count the weighted sum \( a_1x_1 + \ldots + a_nx_n \) for true lits, i.e. in \( A_1 = \{ i \mid x_i = 1 \} \)
Consider the constraint $a_1x_1 + \ldots + a_nx_n \leq k$ with $a_i \geq 0$.

Let us count the weighted sum $a_1x_1 + \ldots + a_nx_n$ for true lits, i.e. in $A_1 = \{ i \mid x_i = 1 \}$.

Assume there are $E \subseteq A_1$ and $j \not\in E$ s.t. $a_j + \sum_{i \in E} a_i > k$.

Literal $\overline{x}_j$ can then be propagated.

Explanation: clause

$$\bigvee_{i_s \in E} \overline{x}_{i_s} \lor \overline{x}_j$$
Propagator for PB Constraints

- Consider the constraint $a_1x_1 + \ldots + a_nx_n \leq k$ with $a_i \geq 0$

- Let us count the weighted sum $a_1x_1 + \ldots + a_nx_n$ for true lits, i.e. in $A_1 = \{i \mid x_i = 1\}$

- Assume there are $E \subseteq A_1$ and $j \not\in E$ s.t. $a_j + \sum_{i \in E} a_i > k$

Literal $\overline{x}_j$ can then be propagated

- Explanation: clause

$$\bigvee_{i_s \in E} \overline{x}_i \lor \overline{x}_j$$

- Again, explanations correspond to clauses of a naive encoding (generalization of the case of cardinality constraints)
SMT and SAT Encodings Are Complementary

Comparison of SMT / SAT encoding
(using same underlying SAT solver Barcelogic)

<table>
<thead>
<tr>
<th>Benchmark suite</th>
<th>SMT at least 1.5x faster</th>
<th>SAT enc. at least 1.5x faster</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tomography</td>
<td>86.49%</td>
<td>5.93%</td>
</tr>
<tr>
<td>PB evaluation</td>
<td>43.49%</td>
<td>7.02%</td>
</tr>
<tr>
<td>RCPSP</td>
<td>46.62%</td>
<td>0.69%</td>
</tr>
<tr>
<td>MSU4</td>
<td>15.39%</td>
<td>39.37%</td>
</tr>
<tr>
<td>DES</td>
<td>0.28%</td>
<td>92.06%</td>
</tr>
</tbody>
</table>
SMT and SAT Encodings Are Complementary

Comparison of SMT / SAT encoding
(using same underlying SAT solver Barcelogic)

<table>
<thead>
<tr>
<th>Benchmark suite</th>
<th>SMT at least</th>
<th>SAT enc. at least</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1.5x faster</td>
<td>1.5x faster</td>
</tr>
<tr>
<td>Tomography</td>
<td>86.49%</td>
<td>5.93%</td>
</tr>
<tr>
<td>(many card. cons.)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PB evaluation</td>
<td>43.49%</td>
<td>7.02%</td>
</tr>
<tr>
<td>(many PB/card. cons.)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RCPSP</td>
<td>46.62%</td>
<td>0.69%</td>
</tr>
<tr>
<td>(many PB cons.)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MSU4</td>
<td>15.39%</td>
<td>39.37%</td>
</tr>
<tr>
<td>(few card. cons.)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DES</td>
<td>0.28%</td>
<td>92.06%</td>
</tr>
<tr>
<td>(1 large card. cons.)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Can we get the best of both worlds?
Related Work

Conflict-Directed Lazy Decomposition: [Abío & Stuckey, CP’12]

- **Goal**: to get the best of SAT encodings and SMT
- **Basic idea**:
  - **Start off** with a full SMT approach for each constraint
  - **On the fly**, partially encode only *active parts* of constraints
  - **Active** = would appear in explanations in conflict analysis
Related Work

Conflict-Directed Lazy Decomposition: [Abío & Stuckey, CP’12]

- **Goal:** to get the best of SAT encodings and SMT
- **Basic idea:**
  - **Start off** with a full SMT approach for each constraint
  - **On the fly,** partially encode **only active parts** of constraints
  - **Active** = would appear in explanations in conflict analysis
- **Thus:**
  - Very active constraints end up completely encoded
  - Little active constraints are handled with SMT
Related Work

Conflict-Directed Lazy Decomposition: [Abíó & Stuckey, CP’12]

Goal: to get the best of SAT encodings and SMT

Basic idea:

- Start off with a full SMT approach for each constraint
- On the fly, partially encode only *active parts* of constraints
- Active = would appear in explanations in conflict analysis

Thus:

- Very active constraints end up completely encoded
- Little active constraints are handled with SMT

So far only available for encodings allowing partial decomposition (non-trivial):

- cardinality network encoding for cardinality cons.
- BDD encoding for PB cons.
Our Contribution: Pros of SMT (1)

- When is SMT effective?
- Often, while searching for solutions, constraints only
  - block the current solution candidate very few times (generate very few explanations)
  - or
  - they do it almost always in the same way (generate few different explanations)
- Generating these explanations can be much more effective than encoding all constraints from the beginning
Our Contribution: Pros of SMT (2)

Table below shows % of benchmark instances where at least half the constraints have a given % of repeated explanations

Recall: in **Tomography, PB evaluation, RCPSP** better is SMT; in **MSU4, DES** better are SAT encodings

<table>
<thead>
<tr>
<th>Suite</th>
<th>0-5%</th>
<th>5-10%</th>
<th>10-20%</th>
<th>20-40%</th>
<th>40-60%</th>
<th>60-80%</th>
<th>80-95%</th>
<th>95-100%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tomography</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>100</td>
<td>0</td>
</tr>
<tr>
<td>PB evaluation</td>
<td>6.2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.6</td>
<td>14.2</td>
<td>51.7</td>
</tr>
<tr>
<td>RCPSP</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>5.5</td>
<td>54.4</td>
<td>1.6</td>
</tr>
<tr>
<td>MSU4</td>
<td>66.9</td>
<td>11.0</td>
<td>19.9</td>
<td>12.4</td>
<td>2.8</td>
<td>0.9</td>
<td>0.2</td>
<td>0</td>
</tr>
<tr>
<td>DES</td>
<td>21.4</td>
<td>29.8</td>
<td>35.2</td>
<td>13.6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Our Contribution: Cons of SMT

- When is SMT not so effective?
- Sometimes some bottleneck constraints end up generating an exponential number of explanations, equivalent to a naive SAT encoding with no auxiliary variables.
Our Contribution: Cons of SMT

- When is SMT not so effective?
- Sometimes some bottleneck constraints end up generating an exponential number of explanations, equivalent to a naive SAT encoding with no auxiliary variables.

Example: in

\[ \begin{align*}
  x_1 + \ldots + x_n &< n/2 \\
  x_1 + \ldots + x_n &\geq n/2
\end{align*} \]

SMT forced to produce all explanations of the form

\[ \overline{x_{i_1}} \lor \overline{x_{i_2}} \lor \ldots \lor \overline{x_{i_{n/2}}} \]

and

\[ x_{i_1} \lor x_{i_2} \lor \ldots \]
Our Contribution: Cons of SMT

When is SMT not so effective?

- Sometimes some bottleneck constraints end up generating an exponential number of explanations, equivalent to a naive SAT encoding with no auxiliary variables.

Example: in

\[
\begin{align*}
x_1 + \ldots + x_n &< n/2 \\
x_1 + \ldots + x_n &\geq n/2
\end{align*}
\]

SMT forced to produce all explanations of the form

\[
\overline{x_{i_1}} \lor \overline{x_{i_2}} \lor \ldots \lor \overline{x_{i_{n/2}}}
\]

and

\[
x_{i_1} \lor x_{i_2} \lor \ldots
\]

A polynomial-sized encoding for such a bottleneck constraint (possibly with auxiliary variables) may be better.

To Encode or to Propagate? The Best Choice for Each Constraint in SAT – p.19/23
Our Contribution: Getting the Best

- We implemented an SMT solver equipped with the ability of encoding on the fly:
  - cardinality constraints with cardinality networks
  - PB constraints with BDD’s

Encoding is irreversible (once a constraint is encoded, its propagator is off forever) and not partial (all or nothing)

- When to encode a constraint?
  When SMT is producing too many different explanations:
Our Contribution: Getting the Best

- We implemented an SMT solver equipped with the ability of encoding on the fly:
  - cardinality constraints with cardinality networks
  - PB constraints with BDD’s

- **Encoding is irreversible** (once a constraint is encoded, its propagator is off forever) and not partial (all or nothing)

- **When to encode a constraint?**
  - When SMT is producing too many different explanations:
    - If number of generated explanations gets close to (> 50 %)
      the number of clauses of the compact SAT encoding
Our Contribution: Getting the Best

- We implemented an SMT solver equipped with the ability of encoding on the fly:
  - cardinality constraints with cardinality networks
  - PB constraints with BDD’s

- Encoding is irreversible (once a constraint is encoded, its propagator is off forever) and not partial (all or nothing)

- When to encode a constraint?
  - When SMT is producing too many different explanations:
    - If number of generated explanations gets close to (> 50 %) the number of clauses of the compact SAT encoding
    - More than $X$ % of the explanations are new and more than $Y$ explanations have already been generated;
      for us, $X = 70$ and $Y = 5000$
### Experimental Results

<table>
<thead>
<tr>
<th>Suite</th>
<th>No. solved instances within &lt; 600 secs.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SMT</td>
</tr>
<tr>
<td>Tomography</td>
<td>2021</td>
</tr>
<tr>
<td>PB evaluation</td>
<td>414</td>
</tr>
<tr>
<td>RCPSP</td>
<td><strong>272</strong></td>
</tr>
<tr>
<td>MSU4</td>
<td>4767</td>
</tr>
<tr>
<td>DES</td>
<td>1452</td>
</tr>
</tbody>
</table>

- No. of problems New solves close to best option for each suite
- Comparable, often better, results than lazy decomposition (LD) but much simpler and more widely applicable!
Conclusions and Future Work


- It is unnecessary to consider partial encodings: just encode on the fly the few really active constraints entirely.

- The method is widely applicable: unlike lazy decomposition, not just for constraints for which partial encodings are known.

- Future work:
  - Consider other kinds of constraints (alldifferent, ...)
  - Explore other adaptive strategies
Thank you!