There are no CNF problems

Peter J. Stuckey and countless others!
Conspirators


- All errors and outrageous lies are mine
Outline

• Modelling and solving
• Propagation based solving
• The advantages of keeping structure
  – Better (static) CNF encoding
  – Dynamic choice: propagation versus CNF encoding
  – Structure-based extended resolution
• MiniZinc
• Conclusion
A famous problem (in CNF)

c unknown problem
p cnf 6 9
1 2 0
3 4 0
5 6 0
-1 -3 0
-1 -5 0
-3 -5 0
-2 -4 0
-2 -6 0
-4 -6 0
### A famous problem (in CNF)

```
c unknown problem
p cnf 12 22
1 2 3 0 4 5 6 0 7 8 9 0 10 11 12 0
-1 -4 0 -1 -7 0 -1 -10 0
-4 -7 0 -4 -10 0 -7 -10 0
-2 -5 0 -2 -8 0 -2 -11 0
-5 -8 0 -5 -11 0 -8 -11 0
-3 -6 0 -3 -9 0 -3 -12 0
-6 -9 0 -6 -12 0 -9 -12 0
```
A famous problem (in MiniZinc)

```plaintext
int: n;
array[1..n] of var 1..n-1: x;
constraint alldifferent(x);
solve satisfy;

n = 4; % data could be
       % in different file
```
A famous problem (in MiniZinc)

```plaintext
int: n;
set of int: Pigeon = 1..n;
set of int: Hole = 1..n-1;
array[Pigeon] of var Hole: x;
constraint alldifferent(x);
solve satisfy;

n = 4;  % data could be
         % in different file
```
A famous problem (in SMT-LIB?)

(declare-fun x1 () Int)
(declare-fun x2 () Int)
(declare-fun x3 () Int)
(declare-fun x4 () Int)
(assert (and (< x1 4) (> x1 0)))
(assert (and (< x2 4) (> x2 0)))
(assert (and (< x3 4) (> x3 0)))
(assert (and (< x4 4) (> x4 0)))
(assert (and (distinct x1 x2) (distinct x1 x3) (distinct x1 x4) (distinct x2 x3) (distinct x2 x4) (distinct x3 x4))
A famous problem (in SMT-LIB?)

(declare-fun x1 () Int)
(declare-fun x2 () Int)
(declare-fun x3 () Int)
(declare-fun x4 () Int)
(assert (and (< x1 4) (> x1 0)))
(assert (and (< x2 4) (> x2 0)))
(assert (and (< x3 4) (> x3 0)))
(assert (and (< x4 4) (> x4 0)))
(assert (alldifferent x1 x2 x3 x4))
Modelling and Solving

- The conceptual model
  - A formal mathematical statement of the (simplified) problem
- The design model
  - In the form that can be handled by a solver
Modelling and Solving

Problem (hard)  \(\xrightarrow{\text{modeling}}\) Conceptual Model  \(\xrightarrow{\text{encoding}}\) Instance

Benefit  \(\xleftarrow{\text{use}}\) Answer  \(\xleftarrow{\text{decoding}}\) Solution

Problem Data  \(\xrightarrow{\text{solving}}\)
Modelling and Solving in SAT

- Problem (hard)
- Conceptual Model
- CNF
- CNF solving
- Solution
- Benefit
- Answer

Steps:
1. Problem
2. Conceptual Model
3. CNF
4. SAT solving
5. Solution
6. Benefit
7. Answer

Key Concepts:
- CNF encoding
- Problem Data

From imagination to impact
Modelling and Solving in MiniZinc

Problem (hard) modeling MiniZinc model translating FlatZinc instance solving

Benefit use Answer output Solution
Outline

• Modelling and solving
• Propagation based solving
• The advantages of keeping structure
  – Better (static) CNF encoding
  – Dynamic choice: propagation versus CNF encoding
  – Structure-based extended resolution
• MiniZinc
• Conclusion
Outline

• Modelling and solving
• Propagation based solving
• The advantages of keeping structure
  – Better (static) CNF encoding
  – Dynamic choice: propagation versus CNF encoding
  – Structure-based extended resolution
• MiniZinc
• Conclusion
Outline

• Modelling and solving
• Propagation based solving
• The advantages of keeping structure
  – Better (static) CNF encoding
  – Dynamic choice: propagation versus CNF encoding
  – Structure-based extended resolution
• MiniZinc
• Conclusion
Better encoding to SAT

Problem (hard) → Conceptual Model → SAT solving → Solution

Benefit → use → Answer → decoding
Better CNF encoding

• Not all SAT encodings are equal
• Significant research encoding constraints to SAT
  – Atmostone
  – Cardinality constraints
  – Psuedo-Boolean constraints
  – Integer variables

• Significant research on “improving” a CNF model after encoding: preprocessing/inprocessing.
Example: encoding Sudoku

\[ X_{ijk} = \text{cell } (i,j) \text{ contains value } k \]

\[ \bigwedge_{ij} \text{one}(X_{ij1}, \ldots, X_{ij9}) \land \bigwedge_{ik} \text{one}(X_{i1k}, \ldots, X_{i9k}) \land \bigwedge_{jk} \text{one}(X_{1jk}, \ldots, X_{9jk}) \land \bigwedge \text{inputs} \]

\[ \text{one}(b_1, \ldots, b_n) = \left( b_1 \lor \cdots \lor b_n \right) \land \]

\[ \bigwedge_{i<j} \left( \overline{b_i} \lor \overline{b_j} \right) \]

\[ \text{At least} \]

\[ \text{At most} \]
So? What’s the Problem?

Tedious task; often repetitive;

1,000,000’s of clauses;
100,000’s of variables;
Bugs are hard to track;
Optimizations are costly

Conceptual Model

encoding

CNF

sat solving

SAT ‘ing Assignm.

decoding

Answer

CNF preprocessors are many: eg, Satelite, Coprocessor

But, these tools apply weak forms of reasoning to cope with huge CNF sizes. (users sometimes prefer to turn them off)
Example: encoding Sudoku

Let the high level structured instance drive the CNF encoding.

Conceptual Model

Problem Data

encoding

High level Instance

encoding

CNF

var 1..9: x11;
var 1..9: x12;
...

alldifferent([x11, ..., x19]);
alldifferent([x21, ..., x29]);
...
x11 = 5;
x12 = 3;
...

Example: encoding Sudoku

5 3 7
6 1 9 5
9 8 6
8 6 3
4 8 3 1
7 2 6
6 2 8
4 1 9 5
8 7 9
The Usual Approach

The Usual Approach

C1

C2

C3

High level Instance

Cn

encode

encode

encode

encode

CNF

simplification
Our Approach

C1 encode CNF

C2 encode CNF

C3 encode CNF

Cn encode CNF

High level Instance

simplification

propagation
Our Approach

Equi-propagation is the process of inferring equations implied by a "single" constraint.

of the form $X = L$ where $L$ is a constant or a literal: $X = Y, X = -Y, X = 0, X = 1$

such $X$ can be removed from all constraints.

Core constraints: literal equations (complete solver is congruence closure)
Other constraints: infer new core constraints.

This is a propagation based solver!
Equi-Propagation

- Infer **equalities** between literals and constants
- Apply substitution to remove equated literals
- E.g. \( D(x) = [0..4], D(y) = [0..4] \)
  - Order encoding
  - \([x1,x2,x3,x4] \quad [y1,y2,y3,y4] \quad vi = (v \geq i)\)
- Constraint \( y \neq 2 \)
  - \( y2 = y3 \)
- Constraint \( x + y = 3 \)
  - \( x4 = 0, y4 = 0, y3 = !x1, y2 = !x2, y1 = !x3 \)
  - \([x1,x1,x3,0] \quad [-x3,-x1,-x1,0]\)
- Constraint \( 3x + 4z + 9t \geq 3 \)
Ben-Gurion Equi-Propagation Encoder

- BEE encoder
- Translates high level instance to CNF
- Integers represented by order/value/binary encoding
- Equi propagation by
  - Adhoc rules per constraint type
    - fast, precise in practice
  - Complete equi-propagation using SAT (?)
- And adhoc partial evaluation rules
BEE Comparisons

- Balanced Incomplete Block Design
- Compared with
  - Sugar (CSP encoder)
  - BEE minus equi-propagation + SatELite

Figure 20: BIBD symmetry breaking.

The naive model for a BIBD instance \([v, b, r, k, \lambda]\) introduces the following constraints on a Boolean incidence matrix: (1) exactly \(r\) ones in each row, (2) exactly \(k\) ones in each column, and (3) exactly \(r\) ones in each scalar product of two (different) rows.

This model does not contain a sufficient degree of information to trigger the equi-propagation process. In order to take advantage of the BEE simplifications we added symmetry breaking as described by Frisch, Je Be, and Miguel (2004) and illustrated in Figure 20: Each row is viewed as sequence of four parts \(A\ldots D\) with sizes \((r), (r), (r), (b_2r + r)\). The first row is fixed by assigning parts \(A\) and \(B\) with ones (marked in black) and parts \(C\) and \(D\) with zeros (marked in white). The second row is fixed by assign parts \(A\) and \(C\) with ones (marked in black) and parts \(B\) and \(D\) with zeros (marked in white). For the third and all subsequent rows (marked in gray), the sum constraints are decomposed into summing each part \((A\ldots D)\) and then summing the results as follows:

- \(A + B = r\)
- \(A + C = r\)
- \(C + D = \lambda\)
- \(B + D = \lambda\)

This ensures that the row contains exactly \(r\) ones and that the scalar product with the first (and second) row is \(r\). We denote this constraint model \(\text{SymB}\) (for symmetry breaking).

<table>
<thead>
<tr>
<th>instance</th>
<th>BEE (SymB)</th>
<th>Sugar (SymB)</th>
<th>SATELITE (SymB)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>comp (sec)</td>
<td>clauses (sec)</td>
<td>comp (sec)</td>
</tr>
<tr>
<td>([7, 420, 180, 3, 60])</td>
<td>1.65</td>
<td>698579</td>
<td>1.73</td>
</tr>
<tr>
<td>([7, 560, 240, 3, 80])</td>
<td>3.73</td>
<td>1211941</td>
<td>13.60</td>
</tr>
<tr>
<td>([12, 132, 33, 3, 6])</td>
<td>0.95</td>
<td>180238</td>
<td>0.73</td>
</tr>
<tr>
<td>([15, 45, 24, 8, 12])</td>
<td>0.51</td>
<td>116016</td>
<td>8.46</td>
</tr>
<tr>
<td>([15, 70, 14, 3, 2])</td>
<td>0.56</td>
<td>81563</td>
<td>0.39</td>
</tr>
<tr>
<td>([16, 80, 15, 3, 2])</td>
<td>0.81</td>
<td>109442</td>
<td>0.56</td>
</tr>
<tr>
<td>([19, 19, 9, 9, 4])</td>
<td>0.23</td>
<td>39931</td>
<td>0.09</td>
</tr>
<tr>
<td>([19, 57, 9, 3, 1])</td>
<td>0.34</td>
<td>113053</td>
<td>0.17</td>
</tr>
<tr>
<td>([21, 21, 5, 5, 1])</td>
<td>0.02</td>
<td>0</td>
<td>0.00</td>
</tr>
<tr>
<td>([25, 25, 9, 9, 3])</td>
<td>0.64</td>
<td>92059</td>
<td>1.33</td>
</tr>
<tr>
<td>([25, 30, 6, 5, 1])</td>
<td>0.10</td>
<td>24594</td>
<td>0.06</td>
</tr>
<tr>
<td>Total (sec)</td>
<td>36.66</td>
<td>&gt; 722.93</td>
<td>&gt; 219.14</td>
</tr>
</tbody>
</table>
BEE Comparison

- Applying SatELite on output of BEE
- YIKES!
  - Doesn’t shrink much, usually solves slower

<table>
<thead>
<tr>
<th>instance</th>
<th>BEE</th>
<th>Δ SatELite</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>comp</td>
<td>clauses</td>
</tr>
<tr>
<td>K8</td>
<td>143</td>
<td>0.51</td>
</tr>
<tr>
<td></td>
<td>142</td>
<td>0.27</td>
</tr>
<tr>
<td></td>
<td>141</td>
<td>0.20</td>
</tr>
<tr>
<td></td>
<td>140</td>
<td>0.19</td>
</tr>
<tr>
<td></td>
<td>139</td>
<td>0.18</td>
</tr>
<tr>
<td></td>
<td>138</td>
<td>0.18</td>
</tr>
<tr>
<td></td>
<td>137</td>
<td>0.18</td>
</tr>
<tr>
<td></td>
<td>136</td>
<td>0.18</td>
</tr>
<tr>
<td></td>
<td>135</td>
<td>0.18</td>
</tr>
<tr>
<td></td>
<td>134</td>
<td>0.18</td>
</tr>
<tr>
<td>K10</td>
<td>267</td>
<td>0.65</td>
</tr>
</tbody>
</table>
BEE Highlights

• Extremal Graph Theory
  – Extremely challenging combinatorics problems
  – Find the largest number of edges for a simple graph with $n$ nodes and no 3 or 4 cycles: $f_4(n)$
  – Huge amount of symmetry

• BEE solution
  – Encode advanced symmetry breaking constraints
  – Discovers two new values
    • $f_4(31) = 80, f_4(32) = 85$

• BEE is best where the initial problem and constraints fix/identify many variables
Outline

• Modelling and solving
• Propagation based solving

• The advantages of keeping structure
  – Better (static) CNF encoding
  – Dynamic choice: propagation versus CNF encoding
  – Structure-based extended resolution

• MiniZinc
• Conclusion
Propagation vs CNF Encoding

- Encode and SAT solve
- Propagate

Solution
Which is better?

- Experience with cardinality problems
- 501 instances of problems with a single cardinality constraint
  - unsat-based MAXSAT solving

<table>
<thead>
<tr>
<th>Suite</th>
<th>Speed up if encoding</th>
<th>Slow down if encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>TO 4 2</td>
<td>Win 1.5 2 4 TO Win</td>
</tr>
<tr>
<td>Card</td>
<td>168 54 14</td>
<td>7 243 215 12 258</td>
</tr>
</tbody>
</table>

- 50% of instances encoding is better, 50% worse
- Why can propagation be superior?
Example: Cardinality constraints

- \( x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 + x_8 \leq 3 \)

- Propagator
  - If 3 of \( \{x_1, \ldots, x_8\} \) are true, set the rest false.

- Encoding
  - Cardinality or sorting network:
    - \( z_{21} = z_{33} = z_{34} = z_{35} = z_{36} = 0 \)
Comparison: Encoding vs Propagation

• A (theory) propagator
  – Lazily generates an encoding
  – This encoding is partially stored in nogoods
  – The encoding uses no auxiliary Boolean variables
  – $\sum_{i=1..n} x_i \leq k$ generates $(n-k)^nC_k = O(n^k)$ explanations

• If the problem is UNSAT (or optimization)
  – CP solver runtime $\geq$ size of smallest resolution proof
  – Cannot decide on auxiliary variables
    • Exponentially larger proof
  – Compare $\sum_{i=1..n} x_i \leq k$ encoding is $O(n \log^2 k)$

• But propagation is faster than encoding
Lazy Encoding

• Choose at **runtime** between encoding and propagation
• All constraints are initially propagators
• If a constraint generates many explanations
  – Replace the propagator by an encoding
  – At restart (just to make it simple)
• **Policy**: encode if either
  – The number of different explanations is > 50% of the encoding size
  – More than 70% of explanations are new and > 5000
Lazy Encoding

- Propagate
- Replace with Encoding
Lazy Encoding results

- MSU4 results

<table>
<thead>
<tr>
<th></th>
<th>&lt;10s</th>
<th>&lt;30s</th>
<th>&lt;60s</th>
<th>&lt;120s</th>
<th>&lt;300s</th>
<th>&lt;600s</th>
</tr>
</thead>
<tbody>
<tr>
<td>Encoding</td>
<td>5374</td>
<td>5525</td>
<td>5578</td>
<td>5621</td>
<td>5659</td>
<td>5677</td>
</tr>
<tr>
<td>Propagation</td>
<td>4322</td>
<td>4530</td>
<td>4603</td>
<td>4667</td>
<td>4737</td>
<td>4767</td>
</tr>
<tr>
<td>Lazy Encoding</td>
<td>5222</td>
<td>5479</td>
<td>5585</td>
<td>5636</td>
<td>5666</td>
<td>5679</td>
</tr>
</tbody>
</table>

- Tomography

<table>
<thead>
<tr>
<th></th>
<th>&lt;10s</th>
<th>&lt;30s</th>
<th>&lt;60s</th>
<th>&lt;120s</th>
<th>&lt;300s</th>
<th>&lt;600s</th>
</tr>
</thead>
<tbody>
<tr>
<td>Encoding</td>
<td>773</td>
<td>1112</td>
<td>1314</td>
<td>1501</td>
<td>1759</td>
<td>1932</td>
</tr>
<tr>
<td>Propagation</td>
<td>1457</td>
<td>1748</td>
<td>1858</td>
<td>1962</td>
<td>2014</td>
<td>2021</td>
</tr>
<tr>
<td>Lazy Encoding</td>
<td>1556</td>
<td>1818</td>
<td>1935</td>
<td>1971</td>
<td>2012</td>
<td>2021</td>
</tr>
</tbody>
</table>
Lazy Encoding

• Keep the **structure** during solving
  – Use the **structure** to decide on solving method

• Almost always **equals or exceeds** the best of
  – Propagation
  – Encoding

• Obvious advantages when
  – Some constraints are **not/rarely** involved in failure
    • These are **never** encoded
Outline

• Modelling and solving
• Propagation based solving
• The advantages of keeping structure
  – Better (static) CNF encoding
  – Dynamic choice: propagation versus CNF encoding
  – Structure-based extended resolution
• MiniZinc
• Conclusion
What makes SAT encoding so good?

• Proof of optimality/unsatisfiability
  – is typically exponential in size

• Size can change exponentially
  – with a richer language of proof

• Encoding adds lots of literals !!!!!

• Extended Resolution
  – dynamically add new literals to the proof system
  – can be exponentially better

• Problem
  – which literals
The Language of Learning

• Is **critical**

• Consider the following MiniZinc model
  
  - `array[1..n] of var 1..n: x;`
  
  - `constraint alldifferent(x);`
  
  - `constraint sum(x) < n*(n+1) div 2;`

• Unsatisfiable
  
  - **No learning**
  
<table>
<thead>
<tr>
<th>n</th>
<th>Failures</th>
<th>Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>240</td>
<td>0.00</td>
</tr>
<tr>
<td>7</td>
<td>1680</td>
<td>0.01</td>
</tr>
<tr>
<td>8</td>
<td>13440</td>
<td>0.08</td>
</tr>
<tr>
<td>9</td>
<td>120960</td>
<td>0.42</td>
</tr>
<tr>
<td>10</td>
<td>1209600</td>
<td>4.47</td>
</tr>
</tbody>
</table>

  - **With learning**

<table>
<thead>
<tr>
<th>n</th>
<th>Failures</th>
<th>Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>270</td>
<td>0.00</td>
</tr>
<tr>
<td>7</td>
<td>1890</td>
<td>0.02</td>
</tr>
<tr>
<td>8</td>
<td>15120</td>
<td>0.20</td>
</tr>
<tr>
<td>9</td>
<td>136080</td>
<td>2.78</td>
</tr>
<tr>
<td>10</td>
<td>1360800</td>
<td>31.30</td>
</tr>
</tbody>
</table>
The Language of Learning

• Is critical

• Consider the following MiniZinc model

  - array[1..n] of var 1..n: x;
  - array[1..n] of var 0..n*(n+1) div 2: s;
  - constraint alldifferent(x);
  - constraint s[1] = x[1] \ /\ s[n] < n*(n+1) div 2;
  - constraint forall(i in 2..n) (s[i]=x[i]+s[i-1]);

• Unsatisfiable

  - No learning

<table>
<thead>
<tr>
<th>n</th>
<th>Failures</th>
<th>Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>240</td>
<td>0.00</td>
</tr>
<tr>
<td>7</td>
<td>1680</td>
<td>0.01</td>
</tr>
<tr>
<td>8</td>
<td>13440</td>
<td>0.08</td>
</tr>
<tr>
<td>9</td>
<td>120960</td>
<td>0.56</td>
</tr>
<tr>
<td>10</td>
<td>1209600</td>
<td>5.45</td>
</tr>
</tbody>
</table>

  - With learning

<table>
<thead>
<tr>
<th>n</th>
<th>Failures</th>
<th>Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>99</td>
<td>0.00</td>
</tr>
<tr>
<td>7</td>
<td>264</td>
<td>0.01</td>
</tr>
<tr>
<td>8</td>
<td>657</td>
<td>0.01</td>
</tr>
<tr>
<td>9</td>
<td>1567</td>
<td>0.04</td>
</tr>
<tr>
<td>10</td>
<td>3635</td>
<td>0.12</td>
</tr>
</tbody>
</table>
The Language of Lemmas

- **Critical** to improving proof size
- Choose the **right language** for expressing lemmas
- Constraint Programming has a **massive advantage** over other complete methods since we “know” the substructures of the problem

- **Structure based extended resolution**
Explanations

- Whenever a propagator makes a new inference
  - it must be explained
- Linear constraint \( x_1 + 2x_2 + 3x_3 + 4x_4 \leq 30 \)
  - \([x_1 = 1] \land [x_2 = 2] \land [x_3 = 3] \rightarrow [x_4 \leq 4]\)
- More general explanation
  - \( C \rightarrow r \) is more general than \( C' \rightarrow r \) if \( C \rightarrow C' \)
- E.g a more general explanation
  - \([x_1 \geq 1] \land [x_2 \geq 2] \land [x_3 \geq 3] \rightarrow [x_4 \leq 4]\)
Maximally General Explanations

- **Maximally general explanations**
  - An explanation which has no strictly more general explanation using the language of learning.
  - e.g. \([x_1 \geq 0] \land [x_2 \geq 1] \land [x_3 \geq 3] \rightarrow [x_4 \leq 4]\)

- **Universally maximal general**
  - An explanation which has no strictly more general explanation w.r.t. the universal language
  - e.g. \([x_1 + 2x_2 + 3x_3 \geq 11] \rightarrow [x_4 \leq 4]\)

- If universally maximally general explanations are not expressible in \(L\)
  - perhaps we should extend \(L\)
Structure based extended resolution

• Global constraints
  – understand the explanations they make
  – know what would literals them more general
  – and can reason about these new literals!

• e.g Linear constraints
  – propagate \([x_4 \leq 4]\) when \([x_1 = 1], [x_2 = 2], [x_3 = 3]\)
  – explanation \([x_1 + 2x_2 + 3x_3 \geq 11] \rightarrow [x_4 \leq 4]\)
  – later \([x_2 \geq 4], [x_3 = 1]\)
  – the linear propagates \([x_2 \geq 4] \land [x_3 \geq 1] \rightarrow [x_1 + 2x_2 + 3x_3 \geq 11]\) setting the new literal true!
  – better \([x_1 + 2x_2 \geq 8] \land [x_3 \geq 1] \rightarrow [x_1 + 2x_2 + 3x_3 \geq 11]\)
Structure based extended resolution

• Table constraints
  – e.g. table([x_1,x_2,x_3,x_4], [[1,2,3,4] [1,2,2,3] [4,3,2,1] [1,1,1,1]])
  – [x_1 = 1] \land [x_2 = 2] \rightarrow [x_4 \neq 1]
  – more general [x_1 \neq 4] \land [x_1 \neq 1] \rightarrow [x_4 \neq 1]
  – or [x_2 \neq 3] \land [x_2 \neq 1] \rightarrow [x_4 \neq 1]
  – max gen: \neg r_3 \land \neg r_4 \rightarrow [x_4 \neq 1]
    • where r_i is a literal meaning row i is the solution.

• Lex e.g. [x_1, \ldots, x_n] \leq [y_1, \ldots, y_n]
  – add literals [x_i > y_i] and [x_i \geq y_i]
  – maximally general explanations now possible
Extending globals with new literals

- Linear $\sum_{i=1..n} a_i x_i \leq a_0$
  - new literal $[\sum_{i=1..k} a_i x_i \geq d]$  
  - linear constraint computes $lb(\sum_{i=1..k} a_i x_i)$ during propagation

- Lex
  - new literal $[x_i > y_i]$
  - lex propagator computes $x_i = y_i, x_i > y_i, x_i < y_i$ during propagation, for i’s that matter!

- It is “a simple matter of programming” to extend a global to support new literals
## Results

<table>
<thead>
<tr>
<th>Problem</th>
<th>no-ngl</th>
<th>basic-ngl</th>
<th>er-ngl</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>fails</td>
<td>time</td>
<td>svd</td>
</tr>
<tr>
<td>Knapsack-30</td>
<td>24712</td>
<td>0.10 20</td>
<td></td>
</tr>
<tr>
<td>Knapsack-40</td>
<td>2549810</td>
<td>7.91 20</td>
<td></td>
</tr>
<tr>
<td>Knapsack-100</td>
<td>119925896</td>
<td>600 0</td>
<td></td>
</tr>
<tr>
<td>Concert-Hall-35</td>
<td>93231</td>
<td>2.95 20</td>
<td></td>
</tr>
<tr>
<td>Concert-Hall-40</td>
<td>1814248</td>
<td>63.12 18</td>
<td></td>
</tr>
<tr>
<td>Concert-Hall-45</td>
<td>10186173</td>
<td>375.1 7</td>
<td></td>
</tr>
<tr>
<td>Talent-14</td>
<td>81220</td>
<td>2.33 20</td>
<td></td>
</tr>
<tr>
<td>Talent-16</td>
<td>572341</td>
<td>17.67 20</td>
<td></td>
</tr>
<tr>
<td>Talent-18</td>
<td>8369293</td>
<td>256.3 16</td>
<td></td>
</tr>
<tr>
<td>Still-Life-9</td>
<td>726722</td>
<td>39.42 1</td>
<td></td>
</tr>
<tr>
<td>Still-Life-10</td>
<td>3390292</td>
<td>189.25 1</td>
<td></td>
</tr>
<tr>
<td>Still-Life-11</td>
<td>9727533</td>
<td>600 0</td>
<td></td>
</tr>
<tr>
<td>PC-Board</td>
<td>16944535</td>
<td>405.8 21</td>
<td></td>
</tr>
<tr>
<td>BIBD</td>
<td>4660894</td>
<td>126.3 7</td>
<td></td>
</tr>
<tr>
<td>Nonogram-small</td>
<td>13566</td>
<td>8.02 11</td>
<td></td>
</tr>
<tr>
<td>Nonogram-medium</td>
<td>299092</td>
<td>254.2 3</td>
<td></td>
</tr>
<tr>
<td>Nonogram-large</td>
<td>829090</td>
<td>600 0</td>
<td></td>
</tr>
<tr>
<td>Jobshop-8</td>
<td>16459</td>
<td>0.65 20</td>
<td></td>
</tr>
<tr>
<td>Jobshop-10</td>
<td>2260393</td>
<td>167.6 13</td>
<td></td>
</tr>
<tr>
<td>Jobshop-12</td>
<td>6848535</td>
<td>596.1 1</td>
<td></td>
</tr>
</tbody>
</table>

We use a fixed order search based on the structural ordering we described above. We will test the same 6 problems with a dynamic search in the fourth experiment.

For nonograms and jobshop, we use the weighted degree search heuristic \[7\], which works well for these two problems. The results are shown in Table 1. Clearly, we can get very significant reductions in node counts on a wide variety of problems.

However, depending on how large the node reduction is and the overhead of extending the language, we may not always get a speedup (e.g., BIBD). The node reduction tends to grow exponentially with problem size.

### 7.2 Frequency of partial sum literals

In the second set of experiments, we test what happens if we only introduce partial sum literals every 5, 10, 20 or 50 variables as described in Section 6. For ease of comparison, we also repeat the er-ngl column from above, where we introduced partial sum literals after every variable. The results are shown in Table 2. Problems not involving linear constraints are omitted here, and in later experiments on partial sums. The trend is very clear here. The fewer partial sum literals we add, the less reduction in node count we have. However, it also requires less overhead.

For many of the instances, introducing partial sum literals every 5 to 10 variables is optimal. Clearly it is also related to the length of the linear constraints, for example in Talent once we are every 10 or greater there is no benefit and the new literals are purely overhead.

### 7.3 Order for partial sum literals

We now compare different ways of picking the order for creating partial sum literals. We compare the structure based ordering (\[\text{struct}\]) which we used in the previous experiments, a random ordering (\[\text{random}\]), and an ordering based on sorting on the
Structure based extended resolution

• Other issues
  – linear constraints
  – exponential number of literals of interest
  – restrict to a single decomposition
    • but which decomposition
    • based on search
    • or, devised at runtime

• for most other globals
  – there are only a polynomial number of extra literals
Outline

• Modelling and solving
• Propagation based solving
• The advantages of keeping structure
  – Better (static) CNF encoding
  – Dynamic choice: propagation versus CNF encoding
  – Structure-based extended resolution
• MiniZinc
• Conclusion
MiniZinc

• A solver independent modelling language for combinatorial optimization problems
  – Open source, developed since 2007
  – Closest thing to a Constraint Programming standard
• **Domains**: Booleans, integers, floats, sets of integers
• **Globals**:  
  – User defined predicates + functions  
  – Reflection functions  
  – Customizable library of global constraint definitions
• **Features**  
  – Annotations for adding non-declarative information
libmzn

• A new open source framework: LLVM like
• Direct interface to solvers and C++ API
• Specialist transformations
  – Booleanization
  – Linearization
• A good modelling language for
  – SAT +
  – SMT solvers
• Released
  – December 2014
New features

- easy install (required for the MOOC)
- user-defined functions
  - functional view of globals (better CSE)
- option types
  - for modelling decisions that are only relevant if others are made (e.g. optional task scheduling)
- Coming soon
  - MiniSearch: meta-search in MiniZinc
  - half-reification: better complex constraint handling
  - pre-solving: reduction of model size
Conclusions

• Combinatorial problems often include
  – Substantial and well understood substructures

• Modelling should
  – allow these substructures to be expressed

• Solving should
  – allow these substructures to be taken advantage of

• Taking note of substructures can:
  – Improve design models (better translation)
  – Allow use to choose between encoding and propagation
  – Create powerful dynamic encodings
The Hard Word

• If you want to compete with all optimization technology
  – Competition is on a high level model, not CNF

• Then ignoring the structure
  – Will not compete!

• So remember

There are no CNF problems
The future directions

- Details of how modern LCG solvers work
- More about MiniZinc
  - [www.minizinc.org](http://www.minizinc.org)
- More about BEE
  - [http://amit.metodi.me/research/bee/](http://amit.metodi.me/research/bee/)
- Structure-based extended resolution
  - Advantages of encoding + propagation simultaneously
- Unsatisfiable cores for constraint programming
  - Easy to translate UNSAT core methods from SAT