Those who forget the past are doomed to repeat it

Peter J. Stuckey and countless others!
Those who forget the past are doomed to repeat it

Jorge Agustín Nicolás Ruiz de Santayana y Borrás, “Nunca me hablas de estas tema otra vez”
George Santayana, 1863-1952
Conspirators

• Ignasi Abio, Ralph Becket, Sebastian Brand, Geoffrey Chu, Michael Codish, Greg Duck, Nick Downing, Thibaut Feydy, Kathryn Francis, Graeme Gange, Vitaly Lagoon, Amit Metodi, Nick Nethercote, Roberto Nieuwenhuis, Olga Ohrimenko, Albert Oliveras, Enric Rodriguez Carbonell, Andreas Schutt, Guido Tack, Pascal Van Hentenryck, Mark Wallace

• All **errors and outrageous lies are mine**
How much of CP search is repeated?

• 4 colour the graph below

1. Inorder labelling: 462672 failures
   - With learning: 18 failures

2. Value symmetries removed: 19728 failures
   - With learning: 19 failures

3. Reverse labelling: 24 failures
   - With learning: 18 failures
How much of CP search is repeated?

- **Resource Constrained Project Scheduling**
  - BL instance (20 tasks)

  ![Diagram](image)

- **Input order**: 934,535 failures
  - With learning: 931 failures

- **Smallest start time order**: 296,567 failures
  - With learning: 551 failures

- **Activity-based search**: > 2,000,000 failures
  - With learning: 1144 failures
How much of CP search is repeated?

• Short answer: a lot

• Methods to alleviate the problem
  – Symmetry/dominance handling
  – Restarts + dynamic search strategies
  – Learning/Caching
Outline

• Propagation based solving
  – Atomic constraints

• Lazy clause generation basics
  – Explaining propagators
  – Conflict resolution

• LCG successes
  – Scheduling, Packing

• Improving LCG
  – How modern LCG solvers work

• Search is Dead

• Concluding remarks
Outline

• Propagation based solving
  – Atomic constraints

• Lazy clause generation basics
  – Explaining propagators
  – Conflict resolution

• LCG successes
  – Scheduling, Packing

• Improving LCG
  – How modern LCG solvers work

• Search is Dead

• Concluding remarks
Propagation Solving (CP)

• Complete solver for atomic constraints
  – $x = d$, $x \neq d$, $x \geq d$, $x \leq d$
  – Domain $D(x)$ records the result of solving (!)

• Propagators infer new atomic constraints from old ones
  – $x_2 \leq x_5$ infers from $x_2 \geq 2$ that $x_5 \geq 2$
  – $x_1 + x_2 + x_3 + x_4 \leq 9$ infers from $x_1 \geq 1 \land x_2 \geq 2 \land x_3 \geq 3$ that $x_4 \leq 3$

• Inference is interleaved with search
  – Try adding $c$ if that fails add not $c$

• Optimization is repeated solving
  – Find solution obj = $k$ resolve with obj < $k$
Finite Domain Propagation Ex.

```plaintext
array[1..5] of var 1..4: x;
constraint alldifferent([x[1], x[2], x[3], x[4]]);
constraint x[2] <= x[5];
```

<table>
<thead>
<tr>
<th></th>
<th>x₁=1</th>
<th>alldiff</th>
<th>x₂ ≤ x₅</th>
<th></th>
<th>x₅&gt;2</th>
<th>x₂ ≤ x₅</th>
<th>alldiff</th>
<th>sum≤9</th>
<th>alldiff</th>
</tr>
</thead>
<tbody>
<tr>
<td>x₁</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>x₂</td>
<td>1..4</td>
<td>2..4</td>
<td>2..4</td>
<td>2..4</td>
<td>2..4</td>
<td>2..4</td>
<td>2..4</td>
<td>2..4</td>
<td>2..4</td>
</tr>
<tr>
<td>x₃</td>
<td>1..4</td>
<td>2..4</td>
<td>2..4</td>
<td>2..4</td>
<td>2..4</td>
<td>2..4</td>
<td>2..4</td>
<td>2..4</td>
<td>2..4</td>
</tr>
<tr>
<td>x₄</td>
<td>1..4</td>
<td>2..4</td>
<td>2..4</td>
<td>3..4</td>
<td>3..4</td>
<td>3..4</td>
<td>3..4</td>
<td>3..4</td>
<td>3..4</td>
</tr>
<tr>
<td>x₅</td>
<td>1..4</td>
<td>1..4</td>
<td>2..4</td>
<td>3..4</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>
FD propagation

• **Strengths**
  – High level modelling
  – Specialized global propagators capture substructure
    • and all work together
  – Programmable search

• **Weaknesses**
  – Weak autonomous search (improved recently)
  – Optimization by repeated satisfaction
  – Small models can be intractable
Outline

• Propagation based solving
  – Atomic constraints
• Lazy clause generation basics
  – Explaining propagators
  – Conflict resolution
• LCG successes
  – Scheduling, Packing
• Improving LCG
  – How modern LCG solvers work
• Search is Dead
• Concluding remarks
Lazy Clause Generation (LCG)

• A hybrid SAT and CP solving approach
• Add explanation and nogood learning to a propagation based solver
• Key change
  – Modify propagators to explain their inferences as clauses
  – Propagate these clauses to build up an implication graph
  – Use SAT conflict resolution on the implication graph
LCG in a Nutshell

• Integer variable $x$ in $l..u$ encoded as Booleans
  – $[x \leq d]$, $d$ in $l..u-1$
  – $[x = d]$, $d$ in $l..u$

• Dual representation of domain $D(x)$

• Restrict to atomic changes in domain (literals)
  – $x \leq d$ (itself)
  – $x \geq d$ ! $[x \leq d-1]$ use $[x \geq d]$ as shorthand
  – $x = d$ (itself)
  – $x \neq d$ ! $[x = d]$ use $[x \neq d]$ as shorthand

• Clauses DOM to model relationship of Booleans
  – $[x \leq d] \rightarrow [x \leq d+1]$, $d$ in $l..u-2$
  – $[x = d] \iff [x \leq d] \land ! [x \leq d-1]$, $d$ in $l+1..u-1$
LCG in a Nutshell

• Propagation is clause generation
  – e.g. \([x \leq 2]\) and \(x \geq y\) means that \([y \leq 2]\)
  – clause \([x \leq 2] \Rightarrow [y \leq 2]\)

• Consider
  – \(\text{alldifferent}([x[1], x[2], x[3], x[4]])\);

• Setting \(x_1 = 1\) we generate new inferences
  – \(x_2 \neq 1, x_3 \neq 1, x_4 \neq 1\)

• Add clauses
  – \([x_1 = 1] \Rightarrow [x_2 \neq 1], [x_1 = 1] \Rightarrow [x_3 \neq 1], [x_1 = 1] \Rightarrow [x_4 \neq 1]\)
  – i.e. \(![x_1 = 1] \lor ![x_2 = 1], \ldots\)

• Propagate these new clauses
Lazy Clause Generation Ex.

\[ \text{alldiff} \quad x_2 \leq x_5 \quad x_2 \leq x_5 \quad \text{alldiff} \quad \text{sum} \leq 9 \quad \text{alldiff} \]

\[ x_1 = 1 \]

\[ x_2 \neq 1 \quad x_2 \geq 2 \]

\[ x_3 \neq 1 \quad x_3 \geq 2 \]

\[ x_4 \neq 1 \quad x_4 \geq 2 \]

\[ x_5 \geq 2 \]

\[ x_5 \leq 2 \quad x_5 = 2 \]

\[ x_3 \neq 2 \quad x_3 \geq 3 \]

\[ x_4 \neq 2 \quad x_4 \geq 3 \]

\[ x_3 \leq 3 \]

\[ x_4 \leq 3 \]

\[ x_3 = 3 \]

\[ x_4 = 3 \]

\[ \text{fail} \]
1 UIP Nogood Creation

\[ \text{alldiff} \quad x_2 \leq x_5 \quad x_2 \leq x_5 \quad \text{alldiff} \quad \text{sum} \leq 9 \quad \text{alldiff} \]

\[ \{x_2 \geq 2, x_3 \geq 2, x_4 \geq 2, x_2 = 2\} \rightarrow \text{false} \]
Backjumping

Backtrack to second last level in nogood

Nogood will propagate

Note stronger domain than usual backtracking

$\{x_2 \geq 2, x_3 \geq 2, x_4 \geq 2, x_2 = 2\} \Rightarrow false$
What’s Really Happening

• CP model = high level “Boolean” model
• Clausal representation of the Boolean model is generated “as we go”
• All generated clauses are redundant and can be removed at any time
• We can control the size of the active “Boolean” model
Comparing to SAT

• For some models we can generate all possible explanation clauses before commencement
  – usually this is too big

• Open Shop Scheduling (tai benchmark suite)
  – averages

<table>
<thead>
<tr>
<th></th>
<th>Time</th>
<th>Solve only</th>
<th>Fails</th>
<th>Max Clauses</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAT</td>
<td>318</td>
<td>89</td>
<td>3597</td>
<td>13.17</td>
</tr>
<tr>
<td>LCG</td>
<td>62</td>
<td>6651</td>
<td>1.0</td>
<td></td>
</tr>
</tbody>
</table>
Lazy Clause Generation

• **Strengths**
  – High level modelling
  – Learning avoids repeating the same subsearch
  – Strong autonomous search
  – Programmable search
  – Specialized global propagators (but requires work)

• **Weaknesses**
  – Optimization by repeated satisfaction search
  – Overhead compared to FD when nogoods are useless
LCG for CSPs

• If you are solving extensional CSPs
  – $LCG \equiv SAT$

• Hard to beat SAT on non-numeric CSPs

• Positive table of $n$ tuples of length $k$
  – $k \times n$ binary clauses
  – 1 $n$-ary clause
  – (for domain propagation) $k \times n$ literals in reverse clauses
  – Actually we can do better with MDDs

• Negative table of $n$ tuples of length $k$
  – $n k$-ary clauses
Outline

• Propagation based solving
  – Atomic constraints

• Lazy clause generation basics
  – Explaining propagators
  – Conflict resolution

• LCG successes
  – Scheduling, Packing

• Improving LCG
  – How modern LCG solvers work

• Search is Dead

• Concluding remarks
LCG Successes

• Scheduling
  – Resource Constrained Project Scheduling Problems (RCPSP)
    • (probably) the most studied scheduling problems
    • LCG closed 71 open problems
    • Solves more problems in 18s then previous SOTA in 1800s
  – RCPSP/Max (more complex precedence constraints)
    • LCG closed 578 open instances of 631
    • LCG recreates or betters all best known solutions by any method on 2340 instances except 3
  – RCPSP/DC (discounted cashflow)
    • Always finds solution on 19440 instances, optimal in all but 152 (versus 832 in previous SOTA)
    • LCG is the SOTA complete method for this problem
LCG Successes

• Real World Application
  – Carpet Cutting
    • Complex packing problem
    • Cut carpet pieces from a roll to minimize length
    • Data from deployed solution
  – Lazy Clause Generation Solution
    • First approach to find and prove optimal solutions
    • Faster than the current deployed solution
    • Reduces waste by 35%
LCG Successes

• MiniZinc Challenge
  – comparing CP solvers on a series of challenging problems
  – Competitors
    • CP solvers such as Gecode, Eclipse, SICstus Prolog
    • MIP solvers SCIP, CPLEX, Gurobi (encoding by us)
    • Decompositions to SMT and SAT solvers
  – LCG solvers (from our group) were
    • First (Chuffed) and Second (CPX) in all categories in 2011 and 2012
    • First (Chuffed) in all categories in 2010
  – Illustrates that the approach is strongly beneficial on a wide range of problems
Outline

- Propagation based solving
  - Atomic constraints
- Lazy clause generation basics
  - Explaining propagators
  - Conflict resolution
- LCG successes
  - Scheduling, Packing
- Improving LCG
  - How modern LCG solvers work
- Search is Dead
- Concluding remarks
Improving Lazy Clause Generation

• Don’t Save Explanations
• Lazy Literal Generation
• Lazy (Backwards) Explanation
• The Globality of Explanation
• Explaining Global Constraints
• Search for LCG
• Symmetries and LCG
Don’t Save Explanations

- Explanation clauses are only needed for conflict resolution
  - Don’t record them in the SAT solver
  - Just record them in the implication graph
  - Throw them away on backjumping
- **Advantages**
  - Less memory required
  - Faster
- **Disadvantages**
  - Memoizing complex explanations
  - Reprioritizing propagation to follow earlier paths
  - All our scheduling results save explanations
Lazy Literal Generation

• Generate Boolean literals representing integer variables on demand

• E.g.
  – decision $x_1 = 1$ generates literal $[x_1 = 1]$
  – alldiff generates $[x_2 \geq 2]$ (equivalently $![x_2 \neq 1]$)

• Integer domain maintains relationship of literals
  – DOM clauses disappear

• A bit tricky to implement efficiently
Lazy Literal Generation

- For constraint problems over large domains lazy literal generation is crucial (MiniZinc Chall. 2012)

<table>
<thead>
<tr>
<th></th>
<th>amaze</th>
<th>fastfood</th>
<th>filters</th>
<th>league</th>
<th>mspsp</th>
<th>nonogram</th>
<th>patt-set</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial</td>
<td>8690</td>
<td>1043k</td>
<td>8204</td>
<td>341k</td>
<td>13534</td>
<td>448k</td>
<td>19916</td>
</tr>
<tr>
<td>Root</td>
<td>6409</td>
<td>729k</td>
<td>6944</td>
<td>211k</td>
<td>9779</td>
<td>364k</td>
<td>19795</td>
</tr>
<tr>
<td>Created</td>
<td>2214</td>
<td>9831</td>
<td>1310</td>
<td>967</td>
<td>6832</td>
<td>262k</td>
<td>15490</td>
</tr>
<tr>
<td>Percent</td>
<td>34%</td>
<td>1.3%</td>
<td>19%</td>
<td>0.45%</td>
<td>70%</td>
<td>72%</td>
<td>78%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>proj-plan</th>
<th>radiation</th>
<th>shipshed</th>
<th>solbat</th>
<th>still-life</th>
<th>tpp</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial</td>
<td>18720</td>
<td>145k</td>
<td>2071k</td>
<td>12144</td>
<td>18947</td>
<td>19335</td>
</tr>
<tr>
<td>Root</td>
<td>18478</td>
<td>43144</td>
<td>2071k</td>
<td>9326</td>
<td>12737</td>
<td>18976</td>
</tr>
<tr>
<td>Created</td>
<td>5489</td>
<td>1993</td>
<td>12943</td>
<td>10398</td>
<td>3666</td>
<td>9232</td>
</tr>
<tr>
<td>Percent</td>
<td>30%</td>
<td>4.6%</td>
<td>0.62%</td>
<td>111%</td>
<td>29%</td>
<td>49%</td>
</tr>
</tbody>
</table>
Lazy Explanation

- Explanations only needed for nogood learning
  - Forward: record propagator causing atomic constraint
  - Backward: ask propagator to explain the constraint

- Standard for SMT and SAT extensions

- Only create \textit{needed explanations}

- Scope for:
  - Explaining a \textit{more general failure} than occurred
  - Making use of the \textit{current nogood} in choosing an explanation

- Interacts \textit{well} with lazy literal generation
(Original) LCG propagation example

- Variables: \( \{x,y,z\} \) \( D(v) = [0..6] \) Booleans \( b, c \)
- Constraints:
  - \( z \geq y, b \rightarrow y \neq 3, c \rightarrow y \geq 3, c \rightarrow x \geq 6, \)
  - \( 4x + 10y + 5z \leq 71 \) (lin)
- Execution

\[
\begin{align*}
\text{1UIP nogood: } & c \land [y \neq 3] \Rightarrow \text{false} \quad \text{or} \quad \text{[y \neq 3] } \Rightarrow \neg c \\
\end{align*}
\]
LCG propagation example

• Execution

<table>
<thead>
<tr>
<th>x ≥ 5</th>
<th>b → y ≠ 3</th>
<th>c → y ≥ 3</th>
<th>z ≥ y</th>
</tr>
</thead>
<tbody>
<tr>
<td>y ≤ 5</td>
<td>y ≠ 3</td>
<td>y ≥ 3</td>
<td>y ≥ 4</td>
</tr>
<tr>
<td></td>
<td>lin</td>
<td>lin</td>
<td></td>
</tr>
</tbody>
</table>

Explanation: \( x \geq 6 \land \neg \text{no good} z \land y \geq 4 \land 4x + y + z \leq 7 \) → false

Lifted Explanation: \( y \geq 4 \land \neg \text{no good} x \land z \geq 43 \land 4x + 10y + z \leq 7 \) → false

Lifted Explanation: \( y \geq 3 \land \neg \text{no good} x \land y \geq 4 \land z \geq 3 \) → false

Absorption
LCG propagation example

- Execution

\[
\begin{align*}
[x \geq 5] & \quad \text{lin} & \quad [x \geq 6] \\
[y \leq 5] & \quad b & \quad [y \neq 3] & \quad b \rightarrow y \neq 3 \\
[y \geq 3] & \quad c & \quad [y \geq 3] & \quad c \rightarrow y \geq 3 \\
[z \geq y] & \quad \text{lin} & \quad [z \geq 4] & \quad \text{false}
\end{align*}
\]

\[\text{Nogood: } [x \geq 5] \land [y \geq 4] \rightarrow \text{false}\]

\[\text{1UIP Nogood: } [x \geq 5] \land [y \geq 4] \rightarrow \text{false}\]

\[\text{1UIP Nogood: } [x \geq 5] \rightarrow [y \leq 3]\]
LCG propagation example

• Backjump

\[ x \geq 5 \]
\[ y \leq 5 \]

\[ x \geq 5 \implies y \leq 3 \]

\[ y \leq 3 \]

Nogood: \[ [x \geq 5] \land [y \geq 4] \implies \text{false} \]
# Backwards versus Forwards

<table>
<thead>
<tr>
<th>Class</th>
<th>$n$</th>
<th>forward</th>
<th>backward</th>
<th>clausal</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>time</td>
<td>fails</td>
<td>len</td>
</tr>
<tr>
<td>amaze</td>
<td>5</td>
<td>113</td>
<td>272546</td>
<td>16.8</td>
</tr>
<tr>
<td>fast-food</td>
<td>4</td>
<td>345</td>
<td>241839</td>
<td>44.3</td>
</tr>
<tr>
<td>filters</td>
<td>4</td>
<td>613</td>
<td>883948</td>
<td>11.2</td>
</tr>
<tr>
<td>league</td>
<td>2</td>
<td>11</td>
<td>74483</td>
<td>28.3</td>
</tr>
<tr>
<td>mspsp</td>
<td>6</td>
<td>23</td>
<td>55021</td>
<td>24.3</td>
</tr>
<tr>
<td>nonogram</td>
<td>4</td>
<td>1965</td>
<td>96461</td>
<td>141.5</td>
</tr>
<tr>
<td>pattern-set</td>
<td>2</td>
<td>451</td>
<td>81397</td>
<td>180.4</td>
</tr>
<tr>
<td>proj-plan</td>
<td>4</td>
<td>83</td>
<td>74531</td>
<td>42.1</td>
</tr>
<tr>
<td>radiation</td>
<td>2</td>
<td>1.5</td>
<td>7407</td>
<td>17.3</td>
</tr>
<tr>
<td>ship-sched</td>
<td>5</td>
<td>43</td>
<td>44897</td>
<td>16.0</td>
</tr>
<tr>
<td>solbat</td>
<td>5</td>
<td>696</td>
<td>337692</td>
<td>201.2</td>
</tr>
<tr>
<td>still-life</td>
<td>5</td>
<td>735</td>
<td>745949</td>
<td>21.9</td>
</tr>
<tr>
<td>tpp</td>
<td>4</td>
<td>613</td>
<td>8486</td>
<td>27.4</td>
</tr>
</tbody>
</table>

First we should note that there is no universal winning explanation strategy, while *backward* is generally the best there are a number of problems were *forward* performs better. Because it performs less explanation *backward* is faster per fail, and even though it performs more search is often faster than *forward*. *Clausal* is not really competitive. Because *backward* generates more varied atomic constraints its explanations clause lengths are longer, similarly *clausal* generates more literals for explanations. Interestingly, Gent et al [4] compare *forward* and *backward* explanation and find *backward* much better. This is possibly because their system does not include bounds atomic constraints, and hence a (single) bounds propagation creates many disequality propagations, which penalizes *forward* explanation more.

Table 3 illustrates the importance of lazily generating literals. It shows for each class the average number of Boolean variables that can be defined to represent all atomic constraints for all variables in the model both in the initial model, and at the root node after it reaches its first fixpoint. It then shows the average number of Boolean variables generated during the entire search when using *forward*, *backward* or *clausal* explanation. The results show that for problems with large domains (e.g. *fast-food* and *ship-sched*) only a tiny proportion of the possible literals are created. Very few problems (e.g. *mspsp* and *pattern-set*) generate more than half the possible literals. Comparing the explanation methods: *clausal* unsurprisingly generates more literals than the others, but still not very many on the large domain examples. *Backward* generates more literals than *forward* since its explanations are not so restricted.
The Globality of Explanation

• Nogoods extract **global** information from the problem
• Can overcome **weaknesses** of local propagators

**Example:**
- \( D(x_1) = D(x_2) = \{0..100000\}, \ x_2 \geq x_1 \land (b \iff x_1 > x_2) \)
- Set \( b = true \) and 200000 propagations later failure.

• A global difference logic propagator immediately sets \( b = false! \)
• Lazy clause generation learns \( b = false \) after 200000 propagations
  – But never tries it again!
Globals by Decomposition

• Globals defined by decomposition
  – Don’t require implementation
  – Automatically incremental
  – Allow partial state relationships to be “learned”
  – Much more attractive with lazy clause generation

• When propagation is not hampered, and size does not blowout:
  – can be good enough!
  – e.g. Resource constrained project scheduling!
Explaining Globals

• Globals are better than decompositions
  – More efficient
  – Stronger propagation

• Instrument global constraint to also explain its propagations
  – regular: each explanation as expensive as propagation
  – cumulative: choices in how to explain

• Implementation complexity
• Can’t learn partial state
• More efficient + stronger propagation + control of explanation
Explaining Globals Piecewise

• Splitting explanations makes them more reusable: e.g. cumulative constraint propagation

![Diagram showing splitting of explanations](image)
Explaining Globals Piecewise

• Break it into parts

\[
s \geq \text{est} \land \text{red1} \land \text{red2} \land \text{red3} \implies s \geq \text{new est}
\]

\[
s \geq \text{est} \land \text{red1} \implies s \geq \text{est1}
\]

\[
s \geq \text{est1} \land \text{red2} \implies s \geq \text{est2}
\]

\[
s \geq \text{est2} \land \text{red3} \implies s \geq \text{new est}
\]
Weak Propagation, Strong Explanation

- Explain a **weak** propagator **strongly**
- We get strong explanations, but later!

- TTEF propagation
- Energetic explanation
- Strong propagation algorithms **less important**
Weak Propagation, Strong Explanation

- **Late failure** discovery **doesn’t hurt** so much

- **Strong propagators** are not so important!
- **Strong explanations** are important
Search for LCG

- Strong Autonomous Search
- Activity based search
  - Michel and Van Hentenryck CPAIOR 2012
  - Chaff Moskewitz et al DAC 2001
    - Bump activity of all literals seen in conflict resolution
    - Decay activity of all literals periodically

- Concentrates search on literals causing local failure

- Highly local (1000 fails ago is irrelevant)
- The ONLY SEARCH used in SAT and SMT
Search for LCG

• Restarts are (almost) **FREE**
  – All failure detected in previous searches is recorded
  – Restarting never repeats work
    • Whether a fixed search
    • Or a dynamic search

• Aggressive Restarting

• Works well with activity based search
  – Concentrate on failure
Activity-based search can be BAD

- Car sequencing problem
  - production line scheduling
- Comparing different search strategies
  - Static: selecting in order
  - DomWDeg: weight variables appearing in constraints that fail
  - Impact: prioritising decisions that reduce domains
  - Activity based

<table>
<thead>
<tr>
<th></th>
<th>Static</th>
<th>DomWDeg</th>
<th>Impact</th>
<th>Activity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time (s)</td>
<td>206.3</td>
<td>0.8</td>
<td>951.3</td>
<td>1522.2</td>
</tr>
<tr>
<td>Solved (70)</td>
<td>66</td>
<td>70</td>
<td>55</td>
<td>47</td>
</tr>
</tbody>
</table>
Hybrid Searches

• Most of our state-of-the-art results use Hybrid searches
  – Problem specific objective based search
    • To find good solutions early
  – Switching to activity based search
    • To prove optimality

• Sometimes alternating the two!
• Or throwing a weighted coin to decide which
• More on why this works later
Symmetries and LCG

• LCG interacts well with symmetries
• Symmetry breaking constraints
  – Problem: search strategy disagrees with constraints
  – Solution: activity based search
    • Either the search agrees and constraints get no activity
    • Or the search disagrees and sym constraints get activity
• Dynamic symmetry breaking
  – SBDS is a nogood method
  – Adds symmetric versions of the decision nogood
  – LCG adds symmetric versions of the 1UIP nogood
    • Much stronger
  – No other symmetry breaking method can find these!
Symmetries and LCG

- 5-colour this graph (value symmetry)
- Already coloured \( x_1, x_2, x_3, x_4, x_5 \)
- Setting \( x_6 = 1, x_7 = 2 \), causes failure
- Dec. Nogood: \( x_1 = 1, x_2 = 2, x_3 = 3, x_4 = 4, x_5 = 5, x_6 = 1 \) \( \Rightarrow x_7 \neq 2 \)
- No value symmetric versions are applicable
Symmetries and LCG

5-colour this graph (value symmetry)

Already coloured $x_1, x_2, x_3, x_4, x_5$

Setting $x_6 = 1, x_7 = 2$, causes failure

1UIP Nogood: $x_4 = 4, x_5 = 5, x_6 = 1 \Rightarrow x_7 \neq 2$

Value Symmetric version is relevant

$- x_4 = 4, x_5 = 5, x_6 = 1 \Rightarrow x_7 \neq 3$
Symmetries and LCG

- 5-colour this graph (value symmetry)
- Already coloured $x_1, x_2, x_3, x_4, x_5$
- Setting $x_6 = 1, x_7 = 2$, causes failure
- Adding the two nogoods immediately fails with nogood
  - $x_4 = 4, x_5 = 5 \implies x_6 \neq 1$
  - Symmetry gives: $x_4 = 4, x_5 = 5 \implies x_6 \neq 2$ and $x_4 = 4, x_5 = 5 \implies x_6 \neq 3$
Outline

• Propagation based solving
  – Atomic constraints

• Lazy clause generation basics
  – Explaining propagators
  – Conflict resolution

• LCG successes
  – Scheduling, Packing

• Improving LCG
  – How modern LCG solvers work

• Search is Dead

• Concluding remarks
Search is Dead, Long Live Proof

• Search is simply a proof method
  – With learning its lemma generation

• Optimization problems
  – Require us to prove there is no better solution
  – As a side effect we find good solutions
  – Even if we cant prove optimality,
    • we should still aim to prove optimality

• Primal heuristics (good solutions fast)
  – Reduce the size of optimality proof

• Dual heuristics (good lower bounds fast)
  – Reduce the size of the optimality proof
Search is Dead, Long Live Proof

• The role of Search
  – Find good solutions
    • Only if this helps the proof size to be reduced
  – Find powerful nogoods (lemmas)
    • That are reusable and hence reduce proof size

• Other inferences can reduce proof size
  – Symmetries
  – Dominance
  – Stronger propagators (stronger base inference)

• And a critical factor for reducing proof size
  – Stronger languages of learning
Forget Consistency

• Domain consistency
  – Is irrelevant unless its cheap

• Bounds(R) or Bounds(Z) consistency
  – Also irrelevant

• A propagators value can be measured by
  – Strength of lemmas generated per unit time
  – This can clearly be problem dependent
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**

    | n  | Failures | Time (s) |
    |----|----------|----------|
    | 6  | 240      | 0.00     |
    | 7  | 1680     | 0.01     |
    | 8  | 13440    | 0.08     |
    | 9  | 120960   | 0.42     |
    | 10 | 1209600  | 4.47     |

  – **With learning**

    | n  | Failures | Time (s) |
    |----|----------|----------|
    | 6  | 270      | 0.00     |
    | 7  | 1890     | 0.02     |
    | 8  | 15120    | 0.20     |
    | 9  | 136080   | 2.78     |
    | 10 | 1360800  | 31.30    |
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*(n+1) div 2;`
  - `constraint forall(i in 2..n)(s[i]=x[i]+s[i-1]);`

- **Unsatisfiable**
  
  - **No learning**
    
    | n | Failures | Time (s) |
    |---|----------|----------|
    | 6 | 240      | 0.00     |
    | 7 | 1680     | 0.01     |
    | 8 | 13440    | 0.08     |
    | 9 | 120960   | 0.56     |
    | 10| 1209600  | 5.45     |
  
  - **With learning**
    
    | n | Failures | Time (s) |
    |---|----------|----------|
    | 6 | 99       | 0.00     |
    | 7 | 264      | 0.01     |
    | 8 | 657      | 0.01     |
    | 9 | 1567     | 0.04     |
    | 10| 3635     | 0.12     |
The Language of Lemmas

• **Critical** to improving proof size
• Choose the **right language** for expressing lemmas
• See
  – Lazy encoding. CP2013
  – Structure based extended resolution
• Constraint Programming has a **massive advantage** over other complete methods since we “know” the substructures of the problem
Outline

• Propagation based solving
  – Atomic constraints

• Lazy clause generation basics
  – Explaining propagators
  – Conflict resolution

• LCG successes
  – Scheduling, Packing

• Improving LCG
  – How modern LCG solvers work

• Search is Dead

• Concluding remarks
Conclusions

• Most of CP search is repeated
• Remember the past to avoid repeating it
• Search is only a mechanism for generating good lemmas
• Consider other mechanisms for proof size reduction
  – inference, language, dominance, relaxation, decomposition, primal heuristics, CEGAR
Whats left to be done?

- Language of Learning
- Explaining propagators
  - Sometime building strong explanation is hard
- Conflict directed explanation
  - We can take into account the current conflict while explaining
- Dominances and LCG
  - Dynamic dominance breaking search with learning
- Parallelizing LCG
  - Good luck! It seems proof is essentially sequential
Whats coming

• ObjectiveCP
  – CP based on a small micro kernel
  – See Pascals talk

• ObjectiveCPExplanation
  – An LCG solver in the ObjectiveCP framework

• ObjectiveCPSchedule
  – State of the art scheduling technology

• MiniZinc 2.0
MiniZinc 2.0

- Open LLVM architecture
- User-defined functions
  - Functional constraint modelling, functional globals
  - Better CSE
- Option types
  - Concise modelling of decisions that are only relevant dependent on other decisions
- Half reification
  - Better translation of complex logical constraints
  - Substantial efficiency improvements
  - More flexible use of globals
- Globalizer (powerful structural analysis)
Lightning Model and Solve Competition

• Be crowned the *worlds best optimizer* 😊
• Model and Solve Competition
  – Teams of 3
  – Solve 6 optimization problems with 5 instances each
    • Using any optimization technology you like
  – One laptop, 3 brains, 2 hours

• 16:30 Thursday 19th September
  – Lecture Hall X

• Signup now at the Registration Desk
  – Places are *limited*! First come first served.
Final Word

• NICTA optimization group is looking for a constraint programmer
  – Supply chains and logistics

• University of Melbourne should be advertising for a lecturer position soon in Optimization

• We are always keen to host interns in the “worlds most livable city”

• So come and join us!